UNIVERSITÀ DEGLI STUDI DI PARMA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea in Ingegneria Informatica Elettronica e delle Telecomunicazioni

PROGETTAZIONE E SVILUPPO DI UNA SMART CAMERA BASATA SU TECNOLOGIE DI INTERNET OF THINGS PER SISTEMI DI SORVEGLIANZA EFFICIENTE

Relatore:
Chiar.mo Prof. Simone Cirani

Correlatore:
Dott. Ing. Marco Picone

Tesi di laurea di:
Lorenzo Bisi

Anno Accademico 2013-2014

Eppur si muove!

Galileo Galilei

Ringraziamenti

Il primo ringraziamento va alla mia famiglia che mi ha insegnato l'importanza della cultura e mi ha dato i mezzi per poter studiare, sostenendomi sempre nelle mie scelte.

Un sentito ringraziamento va anche al Relatore di questa tesi Prof. Simone Cirani e al Correlatore Prof. Marco Picone che mi hanno seguito costantemente e con il loro entusismo mi hanno spinto a dare il meglio durante il percorso di internato e di tesi, dal quale credo di aver imparato davvero tanto.

Ringrazio anche gli studenti e dottorandi che hanno condiviso in questo periodo il laboratorio con me, sempre disponibili a dare una mano quando serviva, soprattutto Luca!

Grazie anche ai compagni di studi di questi 3 anni, in particolare un grazie a Tomaso, compagno di mille progetti, con cui ho condiviso gran parte delle fatiche universitarie e senza il quale sarebbe stato tutto più difficile e, soprattutto, meno "giocoso".

Un grazie infine a tutti i miei amici per la pazienza sopportata in quest'ultimo periodo, ma soprattutto per esserci sempre stati quando davvero ne avevo bisogno.

Indice

1 Stato dell'arte **3**
- 1.1 Smart Camera . . . 3
  - 1.1.1 Esempi di Smart Camera . . . 3
- 1.2 The Internet of Things . . . 5
- 1.3 Architetture e protocolli per IoT . . . 5
  - 1.3.1 Constrained networks . . . 6
  - 1.3.2 Wireless Sensor Network . . . 6
  - 1.3.3 Machine to machine . . . 8

2 Rilevazione di oggetti in movimento **9**
- 2.1 Raspberry Pi . . . 9
- 2.2 OpenCV . . . 10
- 2.3 OpenCV: principi base . . . 11
  - 2.3.1 La classe Mat . . . 11
  - 2.3.2 Mask operations . . . 12
- 2.4 Image processing . . . 12
  - 2.4.1 Smoothing . . . 13
  - 2.4.2 Eroding and Dilating . . . 14
  - 2.4.3 Thresholding . . . 15
  - 2.4.4 Finding Contours . . . 15
  - 2.4.5 Floodfill . . . 16
- 2.5 L'algoritmo MotionBubble . . . 17
  - 2.5.1 Passo 1: Rilevare movimento: absdiff vs BackgroundSubtractor . . . 17
  - 2.5.2 Identificazione dei singoli oggetti . . . 18
  - 2.5.3 Dilation . . . 18
  - 2.5.4 Passo 2: Trovare i contorni . . . 19
  - 2.5.5 Passo 3: Ellissi . . . 19
  - 2.5.6 Passo 4: Colorare le ellissi . . . 20
  - 2.5.7 Passo 5: Bounding Rectangles . . . 21
  - 2.5.8 Ottimizzazioni e correzione errori . . . 22

3 IoT Smart Camera **23**
3.1 CoAP - Costrained Application Protocol 23
3.1.1 Messaging Model 25
3.1.2 Request/Response Message 26
3.2 MjCoAP 27
3.3 RpiMotionServer 28
3.3.1 Comunicazione MotionBubble-RpiMotionServer 29
3.3.2 Struttura interna RpiMotionServer 30

4 Valutazione e Sperimentazione **33**
4.1 Prestazioni camera 33
4.1.1 Scatto: OpenCV vs Raspistill 33
4.1.2 Tempi 34
4.2 Scenario Applicativo e Sperimentazione 35
4.2.1 PIR client 36
4.2.2 AlarmCheckerServer 36
4.2.3 SmartCamera 37
4.3 Valutazione Sperimentale 38
4.3.1 Rilevazione 38
4.3.2 Allarme e invio foto 39

Elenco delle figure

1.1 Da sinistra: Datalogic DataVS2,Teledyne Alsa BOA, Matrox Iris GT 5
1.2 Layers 8

2.1 Raspberry Pi + Camera Module 10
2.2 Esempio di mask operation 12
2.3 Smoothing Normalizzato 13
2.4 Smoothing Gaussiano 13
2.5 Smoothing Mediano 14
2.6 Esempio di Dilation 14
2.7 Esempio di Erosion 15
2.8 Bynary Thresholding invertito 15
2.9 findContours applicato su un disegno 16
2.10 floodFill 16
2.11 Passi dell'algoritmo MotionBubble 17
2.12 Algoritmo di rilevamento contorni applicato ad una immagine binarizzata 19
2.13 Le ellissi vengono disegnate 19
2.14 Le ellissi vengono colorate 20
2.15 Bounding Box 21
2.16 Catena di passaggi di MotionBubble 22

3.1 Layers 24
3.2 Message Format 25
3.3 Risposta piggy-baked 27
3.4 Il servizio fornito dalla Rpi è costituito da due parti cooperanti: la parte C++/OpenCV si occupa della visione, la parte Java/CoAP si occupa di gestire le risorse e di interagire con gli altri SmartObject 28
3.5 Aggiornamento risorsa motion 29
3.6 Class Diagram della struttura interna di RpiMotionServer 30
3.7 Esempio di richiesta GET 31
3.8 Esempio di Observer 32

4.1 Magazzino: alle pareti sono installati dei PIR, in alto la SmartCamera 36
4.2 Registrazione del PIR presso il server 38
4.3 Esempio di interazione tra un client PIR, AlarmCheckerServer e RpiMotionServer. Il client invia una PUT a AlarmCheckerServer. Il server valuta il numero di PIR on e decide se chiedere conferma a RpiMotionServer 39
4.4 Esempio di registrazione come Observer di MotionClient a RpiMotionServer e invio delle notifiche 40

Elenco delle tabelle

4.1 Tabella prestazioni di MotionBubble su Rpi: tempi per ogni fase . . 34
4.2 Tabella prestazioni di MotionBubble su PC: tempi per ogni fase . . 35

Introduzione

Le Smart Camera sono sistemi compatti che, oltre ad una telecamera, integrano funzioni di elaborazione dell'immagine. Per la loro efficienza e praticità si stanno diffondendo in una gamma di campi sempre maggiore. La quasi totalità di queste Smart Camera, tuttavia, è fortemente dipendente dai programmi applicativi o dalle interfacce proprietarie. Integrare un dispositivo di questo genere in una rete significa sviluppare un applicativo software dedicato, partendo dall'API fornite dalla casa. Uno sforzo del genere può essere evitato se si danno a questo oggetto i mezzi per poter esporre nella rete un servizio che sia di per sè riconoscibile dagli altri nodi. Un nodo che presenta tali caratteristiche è detto SmartObject e la realizzazione di una rete di oggetti di questo tipo, connessi ad Internet, in grado di interagire e comunicare tra loro e con l'ambiente, utilizzando protocolli standard, viene detta Internet of Things. Nella visione di Internet of Things, gli oggetti sono destinati a diventare partecipanti attive in processi di affari, informazione e sociali in cui sono in grado di interagire e comunicare tra loro e con l'ambiente attraverso lo scambio di dati e di informazioni percepite, reagendo autonomamente al mondo reale e fisico. Una Smart Camera IoT perciò potrebbe essere facilmente integrata o rimossa in una rete qualsiasi di SmartObject, senza che debba essere fatta nessuna modifica, hardware o software, dell'oggetto o della rete stessa. Affinchè la Smart Camera IoT possa mantenere le sue caratteristiche di flessibilità e integrabilità, nella sua realizzazione si dovranno utilizzare tecnologie open-source e protocolli standard. Si è scelto come supporto hardware Raspberry Pi: questo single-board computer infatti, nonostante le dimensioni ridotte, è capace di una gestione eccellente dei video, grazie ad una scheda grafica integrata ottimizzata per sistemi con risorse limitate, ed è fornito di divesi supporti per la connettività. Ultimamente inoltre è stato prodotto anche un Camera Module dedicato. Per quanto riguarda l'elaborazione delle immagini, la scelta è ricaduta su OpenCV, una delle librerie di Computer Vision open-source più ricche e diffuse. Per le tecnologie IoT-based attualmente esiste un protocollo a livello applicativo chiamato CoAP (Constrained Application Protocol) che recentemente è diventato standard. CoAP è un protocollo software destinato a essere utilizzati in dispositivi elettronici molto semplici, che permette loro di comunicare inter- attivamente su Internet. E'

particolarmente indirizzato a piccoli sensori a bassa potenza, interruttori, valvole e componenti simili che devono essere controllate o sorvegliate da remoto, attraverso le reti Internet standard. Il nodo che si vuole progettare sarà dunque una IoT Smart Camera, basata sul protocollo CoAP, installata su una Raspberry Pi che sfrutterà applicativi per la visione sviluppati con OpenCV. In particolare lo SmartObject così delineato offrirà un servizio di rilevamento di oggetti in movimento e di acquisizione di immagini. Successivamente si testerà la SmartCamera sviluppata in uno scenario sorveglianza, in cui lo SmartObject dovrà interagire con altri oggetti IoT.

Capitolo 1

Stato dell'arte

1.1 Smart Camera

Una Smart Camera è un sistema di visione compatto che integra all'interno di un unico componente una telecamera ed un sistema di digitalizzazione ed elaborazione delle immagini, oltre a dispositivi accessori per l'interfacciamento col mondo esterno (porte di comunicazione, ingressi/uscite digitali, uscita video...). Questa architettura si pone come alternativa ad altri sistemi in cui i componenti risiedono in diversi dispositivi come per esempio quelli basati su telecamere, frame grabber e PC. La differenza tra le Smart Camera ed i sistemi di visione tradizionali risiede essenzialmente nelle dimensioni ridotte e nelle limitate risorse hardware disponibili. Una Smart Camera, al pari degli altri sistemi di visione, esegue uno o più programmi applicativi in grado di elaborare le immagini e comunicare le informazioni estratte al mondo esterno attraverso le interfacce di rete (come per esempio WiFi ed Ethernet) di cui è dotata. L'affidabilità di un circuito semplice, di dimensioni ridotte, senza organi meccanici (come i dischi fissi) e dotato di sistemi operativi spesso molto snelli costituisce spesso un notevole fattore di vantaggio per le Smart Camera rispetto alle architetture tradizionali in termini di rapidità di avvio, ingombro e facilità di spostamento.

1.1.1 Esempi di Smart Camera

I campi tipici di impiego delle Smart Camera sono i seguenti:

- ispezione automatica per controllo qualità (rilevazione di difetti, mancanza di componenti...)
- misure senza contatto.
- selezione ed orientamento di pezzi.

- lettura e verifica di codici (Barcode, data Matrix, caratteri alfanumerici - OCR o OCV- etc.).

- Ispezione di materiali continui (bobine, fili, tubi, linee di estrusione) per la rilevazione di difetti o la verifica dimensionale.

- Individuazione di posizione e rotazione di pezzi per la guida robot ed il prelievo automatico.

- Sorveglianza (rilevazione di intrusioni o della presenza di fiamme o fumo, sorveglianza di siti od eventi geologici come frane, dighe).

- Lettura targhe veicoli (intercettazione veicoli rubati, pagamento automatico nei parcheggi, prevenzione frodi e furti nei parcheggi, controllo accessi, controllo tempi di percorrenza, etc.)

- Monitoraggio del traffico (rilevazione automatica infrazioni stradali: contromano, sorpasso, sosta, lancio di oggetti da cavalcavia e viadotti, etc.)

- Riconoscimento biometrico e controllo accessi (riconoscimento facciale, di impronte digitali, dell'iride etc.)

Le Smart Camera solitamente sono accompagnate da un pacchetto software: viene fornita un'API per programmi personalizzate ed anche una applicazione generica che permette già da sola di sfruttare le funzionalità integrate, tramite un'interfaccia. Esistono sul mercato anche sistemi di sorveglianza dotati di Smart Camera in cui interagiscono diversi sensori. Oltre ad una interfaccia, spesso web-based,h accessibile da PC, vengono solitamente fornite anche applicazioni per smartphone. Tuttavia le Smart Camera in commercio sono totalmente dipendenti dal software proprietario: si è quindi costretti ad utilizzare l'applicativo della casa produttrice, o in ogni caso la rigida interfaccia delle API. Ciò limita fortemente la flessibilità di questi dispositivi. Una Smart Camera in grado di rispondere secondo un protocollo standard avrebbe un potenziale ben maggiore: sarebbe infatti da subito integrabile facilmente in qualsiasi rete in modo dinamico.

Figura 1.1: Da sinistra: Datalogic DataVS2,Teledyne Alsa BOA, Matrox Iris GT

1.2 The Internet of Things

La definizione di Internet of Things fa riferimento alla visione e alla realizzazione di una rete di reti con miliardi di nodi di SmartObject organizzati in una struttura Internet-like. Con SmartObject si indicano oggetti come sensori o dispositivi consumatori che sono connessi sia ad Internet che tra di loro. Internet of Things indica anche un insieme di tecnologie che permettono di collegare oggetti come sensori e attuatori a Internet, permettendo di accedere al mondo fisico tramite software. Internet of Things è parte integrante dell' Internet Futuro e potrebbe essere definito come una infrastruttura di rete globale dinamica con funzionalità di self-configuration basata su protocolli di comunicazione standard e interoperabili in cui oggetti virtuali e fisiche hanno identità, attributi fisici, personalità virtuali, usano interfacce intelligenti e sono perfettamente integrati nella rete d'informazione. Nella visione di Internet of Things, gli oggetti sono destinati a diventare partecipanti attive in processi di affari, informazione e sociali in cui sono in grado di interagire e comunicare tra loro e con l'ambiente attraverso lo scambio di dati e di informazioni percepite sull'ambiente, reagendo autonomamente al mondo reale e fisico e influenzandolo facendo avviando processi che attivano azioni e creano servizi, con o senza il diretto intervento umano [1].

1.3 Architetture e protocolli per IoT

L'infrastruttura di Internet of Things permette combinazioni di oggetti intelligenti (cioè sensori wireless, robot mobili, ecc), tecnologie di rete di sensori, e di esseri umani, utilizzando protocolli standard di comunicazione diversi, ma interoperabili. Si realizza una rete dinamica, multimodale/eterogenea che può essere distribuita anche negli spazi più nascosti o remoti (piattaforme petrolifere, miniere, foreste, gallerie, tubi, ecc) o nei casi di emergenze o situazioni di pericolo (terremoti, incen-

di, inondazioni, aree radiazioni). In questa infrastruttura, queste diversi SmartObject si scoprono e si esplorano l'un l'altro e imparano a trarre vantaggio dai dati degli altri mettendo in comune le risorse e migliorando drasticamente la portata e l'affidabilità dei servizi risultanti.

1.3.1 Constrained networks

Per Constrained Network si intende una rete che ha dimensioni di pacchetto limitate,che può presentare un grado elevato di perdita di pacchetti, e può avere un numero consistente di oggetti, i quali possono essere spenti in ogni istante, ma periodicamente si svegliano per brevi periodi di tempo. Queste reti e i nodi all'interno di esse sono caratterizzati da forti limiti su capacità computazionali, batteria disponibile, e in particolare sulla complessità che può essere supportata con le dimensioni del codice limitate e RAM limitata per nodo. Più in generale, si parla di Constrained Network se almeno alcuni dei nodi e delle reti coinvolte mostrano queste caratteristiche.

L'architettura generale si compone di nodi della Constrained Network, chiamati SmartObject, che sono responsabili di una o più risorse che possono rappresentare sensori, attuatori, combinazioni di valori o altre informazioni.Gli SmartObject inviano messaggi per modificare e ottenere risorse altri SmartObject. Gli oggetti possono inviare notifiche di valori delle risorse modificati agli SmartObject che si sono registrati per ricevere la notifica delle modifiche. Uno SmartObject può anche pubblicare o essere interrogato in merito alle sue risorse.

1.3.2 Wireless Sensor Network

La confluenza di comunicazione senza fili, computazione e sensori poco costosi ha creato una nuova generazione di dispositivi intelligenti. L'utilizzo di decine di migliaia di questi dispositivi in reti di auto-organizzanti ha creato una nuova tecnologia denominata reti di sensori wireless (WSN) [2]. Una WSN è costituita dalla convergenza di sensori, sistemi con micro-elettro-meccanismi e tecnologie delle reti. Le WSN sono costituite da piccoli nodi con capacità di comunicazione wireless,rilevamento e computazione. Varie architetture sono state sviluppato per WSNs, a seconda della necessità di applicazione. Sono utilizzate in diverse applicazioni ad esempio monitoraggio ambientale, monitoraggio degli habitat, domotica e applicazioni militari. Un nodo sensore consiste tipicamente in cinque componenti: rilevamento, memoria, processore, transceiver (trasmettitore e ricevitore) e la batteria. Al giorno d'oggi, i nodi devono essere piccoli e poco costosi. Di conseguenza, le loro risorse sono limitate (tipicamente, batteria limitata, ridotta memoria e capacità di elaborazione). A causa della potenza di trasmissione contenuta, i nodi di sensori wireless sono in grado di comunicare solo localmente, con

un certo numero di vicini. Quindi, i nodi devono collaborare per realizzare i loro compiti: rilevamento, elaborazione del segnale, calcolo, routing, localizzazione, sicurezza, ecc. Quindi,le WSN sono, per loro natura, reti di collaborazione [3]. Al fine di sviluppare soluzioni per constrained-network, IETF (Internet Engineering Task Force) ha creato tre gruppi di lavoro.

- 6LoWPAN Working Group IPv6 sulle Low power Wireless Personal Area Networks: Una LoWPAN è una semplice rete di comunicazione low-cost che permette connettività wireless in applicazioni con bassa potenza e requisiti di throughput contenuti. Una LoWPAN tipicamente include oggetti che lavorano insieme per trasmettere informazioni sull'ambiente alle applicazioni del mondo reale. Le LoWPANs sono conformi allo standard IEEE 802.15.4-2003 [4] (Figura 3.1).

- ROLL Working Group: le Low power and Lossy networks (LLNs) sono costituite da molti dispositivi embedded con potenza,memoria e potenza di calcolo limitati. Sono interconnessi da una varietà di link come IEEE 802.15.4, Bluetooth, Low Power WiFi, cavi o altr low power PLC (Power- line Communication) links. LLNs stanno passando ad una soluzione end-to-end IP-based per evitare problemi di networks interoperabili interconnesse gateways con protocolli di traduzione and proxy. Il gruppo di lavoro si concentra solo su framework architetturale di routing IPv6 per questi scenari applicativi. Il quadro prenderà in considerazione vari aspetti, inclusa elevata affidabilità in presenza di caratteristiche di perdita variabile nel tempo e connettività mentre consente il funzionamento a bassa potenza con memoria molto modesta e la pressione sulla CPU nelle reti potenzialmente comprendenti un numero molto elevato (diverse migliaia) di nodi. Una volta che il gruppo di lavoro ha verificato che i già esistente protocolli di routing non erano adatte per questo tipo di rete, hanno creato RPL (Routing Protocol for Low power and lossy networks).

- CoRE Working Group: il Constrained RESTful Environments (CoRE) Group sta fornendo un quadro di riferimento per le applicazioni orientate alle risorse destinate a funzionare su constrained IP networks. Il gruppo di lavoro CoRE definirà un quadro per una cerchia ristretta di applicazioni: quelle che riguardano la manipolazione di semplici risorse su constrained networks. Questo include applicazioni per monitorare semplici sensori (ad esempio sensori di temperatura, interruttori della luce, e misuratori di potenza), per controllare attuaratori (per esempio gli interruttori della luce, controller di riscaldamento e serrature), e di gestire dispositivi. Nell'ambito del quadro per la costruzione di queste applicazioni, il WG ha definito un Constrained Application Protocol (CoAP) [5] per la manipolazione di risorse su un dispositivo.

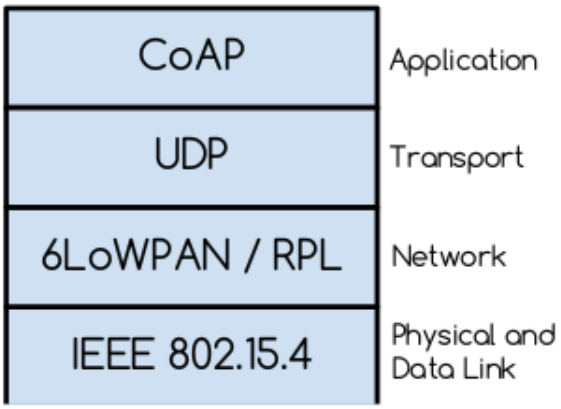

Figura 1.2: Layers

1.3.3 Machine to machine

Machine to Machine (M2M) si riferisce a tecnologie che permettono sia a sistemi wireless che cablati di comunicare con dispositivi con la stessa abilità. M2M usa un dispositivo come un sensore per catturare un evento(come la temperatura, inventory level, etc.), che viene trasmesso attraverso una rete wireless cablata o ibrida ad una applicazione(progrmma software), che traduce l'evento catturato in informazione significativa (per esempio oggetti da essere che devono essere immagazzinati). Una tale comunicazione era originariamente realizzata con una rete remota di macchine che trasmettevano l'informazione a un hub centrale per l'analisi che poi sarebbe stato reindirizzato indietro ad un sistema come un personal computer. In ogni caso la moderna M2M comunicazione è diventata una connessione uno-a-uno ed è si è trasformata in un sistema di reti che trasmettono dati ad applicazioni personali. L'espansione di reti IP nel ha reso molto più facile per la comunicazione M2M prendere piede ed ha diminuito l'ammontare di energia e tempo necessario all'informazione per essere comunicati tra macchine. Queste reti permettono anche una varietà di nuove opportunità di business e connessioni tra consumatori e produttori in termini di vendita di prodotti.

Capitolo 2

Rilevazione di oggetti in movimento

La rilevazione del movimento è un problema frequente nella Computer Vision poichè ha svariate applicazioni. Il problema è di semplice risoluzione per una camera , ma si complica notevolmente se viene richiesto anche di individuare gli oggetti mobili. Uno degli obiettivi di questo lavorod i tesi è valutare il costo e la complessità di algoritmi di visione artificiale per il riconoscimento del movimento che possano essere integrati su nodi a bassa complessità all'interno di scenari di Internet of Things. Gli oggetti che l'algoritmo deve individuare sono oggetti piccoli: la rilevazione di oggetti in movimento implica uno scenario dove ci si aspetta che possano essere presenti più oggetti alla volta, perciò è chiaro che un oggetto che ricopre da solo più della metà della scena non rientra nelle dimensioni della categoria scelta e può essere considerato un'anomalia. L'hardware utilizzato richiede, per avere una risposta real time, che l'algoritmo non sia eccessivamente costoso, computazionalmente parlando. Il tempo di esecuzione sarà perciò un vincolo fondamentale di progetto nella scelta dell'algoritmo e ne condizionerà fortemente le prestazioni in termini di qualità. In questo capitolo verranno descritti i principali strumenti e componenti software utilizzati per la realizzazione e valutazione di soluzioni di visione artificiale applicabili a scenari di Internet of Things.

2.1 Raspberry Pi

Raspberry Pi (Figura 2.1) è un single-board computer di dimensioni ridotte che può essere connesso a più periferiche o dispositivi esterni. E' un piccolo PC che può essere utilizzato per molte attività di un normale pc, come fogli di calcolo, elaborazione di testi e giochi. Visualizza anche video ad alta definizione. Raspberry Pi ha un sistema Broadcom BCM2835 su un chip (SoC), che include un processore

ARM1176JZF-S 700 MHz, GPU VideoCore IV, ed era originariamente venduto con256 megabytes di RAM, poi portata a 512MB. Non include un hard disk, ma utilizza una SD card sia per il S.O che per lo storage a lungo termine. Vi sono diversi sistemi operativi installabili su Raspberry Pi, quasi tutti su kernel Linux. Il S.O. utilizzato è stato Raspbian, una distribuzione basata su Debian e ottimizzata per Raspberry. Come camera è stato utilizzato il Raspberry Pi camera module, una camera da 5 megapixel con fixed-focus che supporta le modalità video 1080p30, 720p60 and VGA90, così come l'acquisizione di fotografie. Si connette via cavo alla porta CSI della Raspberry Pi. Vi si può accedere traminte le API MMAL and V4L APIs, e ci sono numerose librerie di terze parti realizzate per essa inclusa la libreria Python Picamera.

Figura 2.1: Raspberry Pi + Camera Module

2.2 OpenCV

OpenCV è una libreria di computer vision cui obiettivo principale è quello di elaborare e manipolare immagini. OpenCV è stata sviluppata originariamente da Intel, ma attualmente è sotto licenza open source BSD. E' una libreria multipiattaforma, è quindi compilabile sotto molti sistemi operativi (Windows, Mac OS X, Linux, iOS Android). Il linguaggio di programmazione usato è C++, ma possiede anche interfacce C,Python e Java. OpenCV è un framework molto ricco, ma di conseguenza necessita di un notevole spazio in memoria. L'installazione completa occupa di per se circa 1GB anche se in realtà uno spazio quasi altrettanto grande è richiesto per i file prodotti durante la compilazione. Il processo è inoltre molto

lungo, quasi una decina di ore su un dispositivo come la Raspberry Pi che non è particolarmente prestante. E' stata perciò utilizzata per far fronte alle necessità si spazio su hard-disk un dispositivo rimovibile nella fase di compilazione. In seguito per poter utilizzare la camera anche per le applicazioni di OpenCV è stato indispensabile installare anche i drivers UV4L.

2.3 OpenCV: principi base

Esistono diversi modi per acquisire immagini digitali dal mondo reale: fotocamere digitali, scanner, tomografia computerizzata e la risonanza magnetica per citarne alcuni. In ogni caso, ciò che un essere umano vede sono immagini. Tuttavia, sui dispositivi digitali, ciò che viene registrato sono valori numerici per ciascuno dei punti dell'immagine. Come vengano ottenuti e memorizzati i valori dei pixel può variare in base a particolari esigenze, ma alla fine tutte le immagini possono essere ridotte a matrici numeriche.

2.3.1 La classe Mat

Mat è fondamentalmente una classe con due parti di dati: l'header della matrice (contenenti informazioni quali la dimensione della matrice, il metodo utilizzato per la memorizzazione, in cui indirizzo è la matrice memorizzato, e così via) e un puntatore alla matrice contenente i valori dei pixel. La dimensione dell'intestazione matrice è costante, ma la dimensione della matrice stessa può variare in base all'immagine e solitamente è più grande di ordini di grandezza. OpenCV utilizza un sistema di refecerence counting. L'idea è che ogni oggetto Mat ha il suo header, però la matrice può essere condivisa tra due istanze differenti, ma con puntatori allo stesso indirizzo. Inoltre, gli operatori copia copieranno solo gli header e i puntatori a matrici di grandi dimensioni, non i dati stessi. Quando il contatore raggiunge lo zero anche la matrice viene rimossa dalla memoria. È possibile selezionare lo spazio dei colori e il tipo di dati utilizzato. Lo spazio dei colori si riferisce a come si combinano componenti di colore al fine di codificare un determinato colore. Il più semplice è la scala di grigi dove i colori a nostra disposizione sono bianco e nero. La combinazione di questi ci permette di creare molte sfumature di grigio. Per il colore abbiamo molti più metodi tra cui scegliere. In ognuno si hanno tre o quattro componenti di base e possiamo usare la combinazione di queste per creare gli altri colori. Il più popolare è RGB, soprattutto perché questo è anche il modo il nostro occhio percepisce colori. I suoi colori di base sono il rosso, il verde e blu. Per codificare la trasparenza di un colore a volte viene aggiunto un quarto elemento: alfa (A). Vi sono, tuttavia, molti altri sistemi di colore ciascuno con i propri vantaggi: RGB è il più comune siccome i nostri occhi usano qualcosa

di simile e quindi i nostri sistemi di visualizzazione compongono anch'essi i colori usando questi. L'HSV e HLS decompongono i colori nelle loro componenti di valore/luminosità, tonalità, saturazione, che è un modo più naturale per noi per descrivere i colori. Si potrebbe, per esempio, eliminare l'ultima componente, rendendo l'algoritmo meno sensibile alle condizioni di luce dell'immagine in ingresso. YCrCb è utilizzato da popolare formato immagine JPEG. CIE L * a * b * è uno spazio percettivamente uniforme di colore , che si rivela utile se si ha bisogno di misurare la distanza di un dato colore a un altro colore [6].

2.3.2 Mask operations

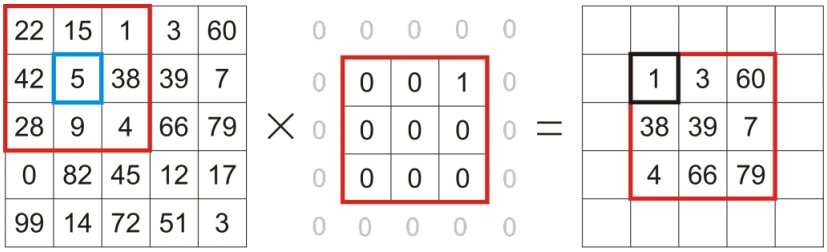

Figura 2.2: Esempio di mask operation

Le Mask operations sulle matrici sono abbastanza semplici. L'idea è quella di ricalcolare il valore di ogni pixel dell'immagine secondo una matrice di mask (anche conosciuta come kernel). Ciò avviene tramite l'applicazione della convoluzione (Figura 2.2) di due matrici bidimensionali di cui la prima rappresenta l'immagine originale e la seconda, detta kernel, rappresenta il filtro da applicare. Le matrici kernel possono essere di dimensione 3x3, 5x5, 7x7, e cosı' via. Consideriamo la matrice A, che rappresenta la matrice contenente i valori di grigio di tutti i pixel dell'immagine originale, e la matrice B che rappresenta la matrice kernel. Sovrapponiamo la matrice B alla matrice A in modo che il centro della matrice B sia in corrispondenza del pixel della matrice A da elaborare. Il valore di ciascun pixel della matrice A oggetto di elaborazione viene ricalcolato come la somma dei prodotti di ciascun elemento della matrice kernel con il corrispondente pixel della matrice A sottostante. Si itera così il procedimento per ogni pixel di A ottenendo una nuova immagine filtrata nel modo opportuno.

2.4 Image processing

Si analizzeranno ora alcune delle principali operazioni di image processing della libreria OpenCV [7], in particolari quelle effettivamente utilizzati in seguito per

l'esecuzione dell'algoritmo.

2.4.1 Smoothing

Una operazione di base è quella di Smoothing, che consiste nel ricalcolare ogni pixel interpolandolo (in qualche modo) con i pixel a fianco: questa operazione ha lo scopo di ridurre il rumore nell'immagine. Ci sono vari tipi di Smoothing a seconda di come viene calcolata questa media:

- Normalizzato: è il più semplice, il kernel consiste in una matrice di soli 1 moltiplicata per l'inverso del prodotto di righe per colonne (Figura 2.3). Ne risulta un calcolo equivalente alla media aritmetica.

Figura 2.3: Smoothing Normalizzato

- Gaussiano: probabilmente il più utile anche se non il più veloce computazionalmente. Il filtro gausssiano è ottunuto facendo la convuluzione di ogni punto con un kernel Gaussiano e poi sommmandoli tutto per produrre l'output. Il peso assegnato ai pixel vicini decresce allontanandosi dal centro del kernel (Figura 2.4).

Figura 2.4: Smoothing Gaussiano

- Mediano: ad ogni pixel si sostituisce il valore alla posizione intermedia tra i pixel vicini (Figura 2.5).

Figura 2.5: Smoothing Mediano

2.4.2 Eroding and Dilating

Nell'operazione di dilation, il kernel B ha un anchor point, di solito al centro del kernel. Facendo scorrere il kernel su tutta la matrice, calcoliamo il valore massimo tra i pixel ricoperti da B e rimpiazziamo con questo il valore del pixel nella posizione di anchor point. Come si deduce, questa massimizzazione fa sì che le regioni chiare crescano (da cui il nome dilation). Di seguito viene dato un esempio. Nell'immagine (Figura 2.6) lo sfondo (chiaro) si dilata sulle regioni nere della lettera.

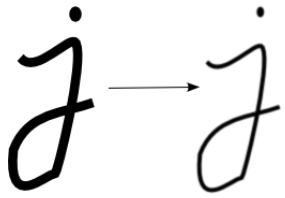

Figura 2.6: Esempio di Dilation

L'erosion è la complementare della dilation. Ciò che fa è calcolare un minimo lacare sull'area del kernel. Applicandola alla medesima immagine si nota che le aree chiare dell'immagine (Figura 2.7) diventano,apparentemente più sottili mentre le aree scure si allargano.

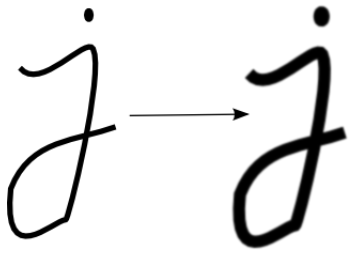

Figura 2.7: Esempio di Erosion

2.4.3 Thresholding

Il Thresholding è il metodo più semplice di segmentazione di un'immagine, ovvero di dividere i pixel in gruppi diversi. Questa separazione è basata sulla differenza di intensità tra i pixel. Per differenziare i pixel a cui siamo interessati dal resto, eseguiamo un confronto dell'intensità di ogni pixel rispetto ad una soglia. Una volta separati in modo appropriato i pixel possiamo settarli con un determinato valore per idenficarli. Nel Binary Thresholding (Figura 2.8) ogni pixel, se inferiore alla soglia, viene sostituito da 0, altrimenti dal massimo valore della matrice.

Figura 2.8: Bynary Thresholding invertito

2.4.4 Finding Contours

Questa funzione restituisce un array di contorni da un'immagine binarizzata presa in ingresso (Figura 2.9). L'algorito utilizzato è quello descritto in questo testo [8].

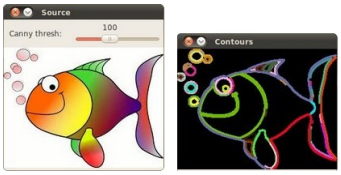

Figura 2.9: findContours applicato su un disegno

2.4.5 Floodfill

La funzione floodFill colora una componente connessa di un particolare colore. La connessione è determinata dalla vicinanza di colore/luminosità rispetto ai pixel vicini. La funzione prende in ingresso un seed point (da cui partire), una matrice e un colore. Per essere aggiunto alla componente connessa un pixel deve avere un colore vicino abbastanza a:

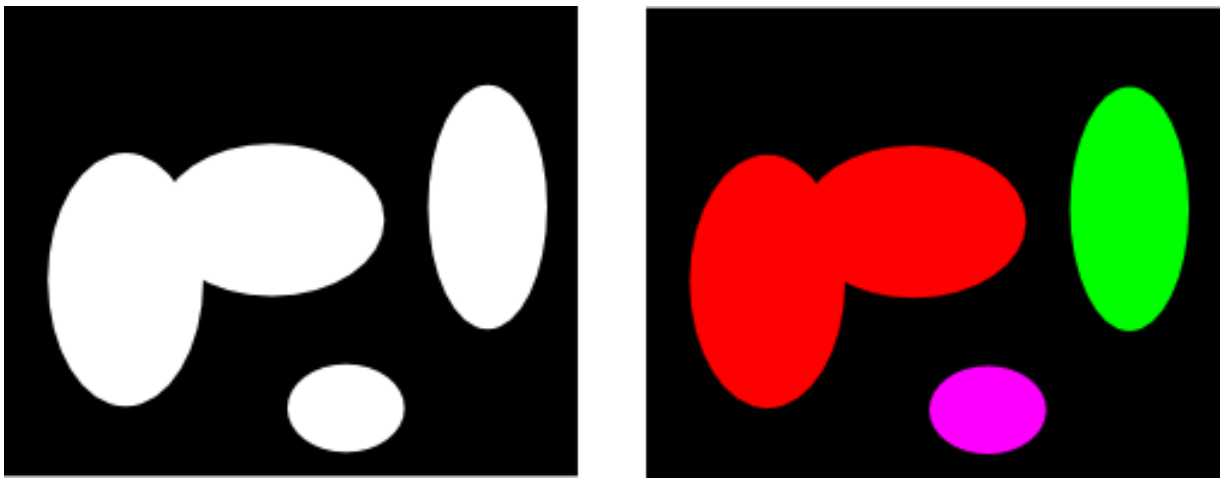

Figura 2.10: floodFill

- il colore di uno dei suoi vicini che appartiene alla componente connessa.

- il colore del seed point.

2.5 L'algoritmo MotionBubble

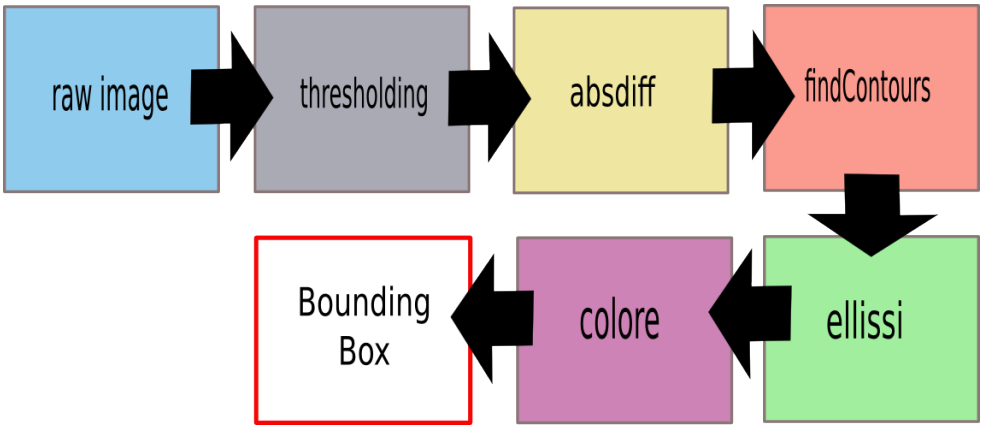

Figura 2.11: Passi dell'algoritmo MotionBubble

All'inizio della sua elaborazione, l'applicativo apre la camera e inizia un ciclo che ad ogni iterazione cattura una nuova immagine e la analizza(Figura 2.11).

2.5.1 Passo 1: Rilevare movimento: absdiff vs BackgroundSubtractor

Un movimento implica una variazione della posizione di uno o più oggetti. A livello di immagine questo si traduce in un cambiamento nei valori delle aree delle matrice che sono state perturbate. Se si va a fare una sottrazione non si può che ottenere una matrice con i seguenti valori: 0 nelle aree rimaste tali e quali e un valore diverso da 0 in quelle in cui qualcosa è variato. OpenCV fornisce la funzione absdiff per sottrarre due matrici di uguale dimensione e farne il valore assoluto (Listing 2.1). Applicando in successione un opportuno filtro di thresholding si ottiene infine una immagine binarizzata in cui le aree bianche rappresentano l'area in cui c'è stato movimento e quelle nere le altre. L'approssimazione della sagoma dell'oggetto che si è spostato dipende da diversi fattori, come il colore dell'oggetto e la velocità con cui l'oggetto si muove, deve perciò essere ulteriormente elaborata. Le prestazioni migliorano utilizzando algoritmi più complessi della semplice sottrazione i quali, per esempio, compensano il rumore di fondo e, configurando opportunamente alcuni parametri, tengono conto della storia dei confronti precedenti. Una classe OpenCV che effettua queste operazioni è BackgroundSubtractorMOG2: sfortunatamente, siccome prestazioni migliori hanno un costo computazionale maggiore, l'utilizzo di questa soluzione non è compatibile con i requisiti di rapidità di esecuzione, non potrà perciò essere preso in considerazione.

Listing 2.1: Differenza tra due immagini successive

```
iteration++;
if(iteration==1){
     oldCount=0;
     (*cap) >> oldFrame;//acquisisco 2 frame
     cvtColor(oldFrame,oldFrameGray,COLOR_BGR2GRAY);
     }
(*cap) >> frame;
cvtColor(frame,frame_gray,COLOR_BGR2GRAY);//conversione
absdiff(oldFrameGray,frame_gray,difference);
```

2.5.2 Identificazione dei singoli oggetti

Se ci si dovesse limitare alla rilevazione di movimento, l'algoritmo potrebbe terminare qui. Tuttavia funzionalità così semplici sono già disponibili in altri software molto più semplici che non necessitano nemmeno di OpenCV, ad esempio motion un programma installabile su Raspberry. Inoltre per quanto riguarda esigenza di sicurezza è importante avere un maggiore controllo sui dati rilevati. E' importante anche determinare il numero di oggetti in movimento e l'entità delle loro dimensioni per minimizzare i falsi allarmi. Fondamentale diventa a questo punto identificare oggetti diversi.

2.5.3 Dilation

Il primo filtro che si vorrebbe applicare all'immagine è una Dilation: con questa operazione si cerca, come già spiegato, di allargare le aree bianche. Lo scopo è cercare di unire i vari frammenti di oggetto che per varie ragioni possono essere rilevati separati gli uni dagli altri. Purtroppo anche questa operazione si rivela costosa computazionalmente con l'aumentare della dimensione del kernel. Se in una prima fase si era utilizzata, successive verifiche hanno dimostrato che non è compatibile con un utilizzo efficace su Rpi.

2.5.4 Passo 2: Trovare i contorni

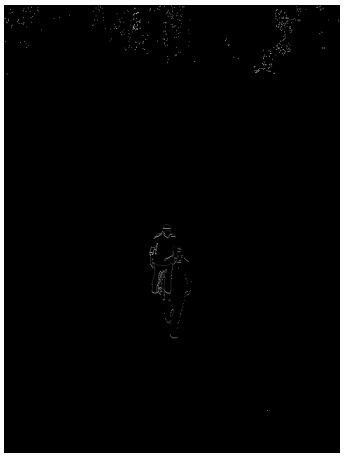

Figura 2.12: Algoritmo di rilevamento contorni applicato ad una immagine binarizzata

Il primo elenco di raggruppamenti di punti viene fornito dalla funzione findcontours (Figura 2.12) di OpenCV. I contorni purtoppo non sono precisi, ma individuano almeno una base per operazioni successive.

2.5.5 Passo 3: Ellissi

Figura 2.13: Le ellissi vengono disegnate

Siccome l'operazione di Dilation è risultata troppo costosa, per cercare di accorpare raggruppamenti appartenenti ad uno stesso oggetto si è scelta una strada differente. Viene disegnata una Bounding Ellipse colorata di bianco (Figura 2.13), ovvero un'ellisse contenente tutti i punti del raggruppamento in questione: tale ellisse vien poi dilatata di un fattore (configurabile). Si ottiene quindi un effetto di allargamento dell'area rilevata, simile a quello ottenuto con la Dilation, ma con un costo di esecuzione nettamente inferiore e soprattutto scalabile (Listing 2.2).

Listing 2.2: disegno le ellissi

```
for( int i = 0; i < contours.size(); i++ )
{
  if(contours[i].size() > 5){
    RotatedRect r=fitEllipse( Mat(contours[i]));
    enlarge(&r,1+(float(factor)/10));
    if(r.size.height>10 && r.size.width>10){
      ellipses.push_back(r);
    }
    ellipse( thresholdImage, r, color,-1, 8 );
  }
}
```

2.5.6 Passo 4: Colorare le ellissi

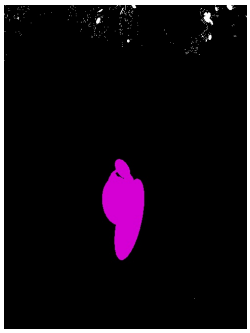

Figura 2.14: Le ellissi vengono colorate

A questo punto accade che diverse ellissi si intersechino: occorre riformare i raggruppamenti di punti in funzione di tali intersezioni. Si sfrutta perciò la funzione

floodFill in un ciclo che scorre la lista di ellissi e si prepara una nuova lista di raggruppamenti a cui saranno assegnati colori caratteristici. Si valuta il colore del punto centrale dell'ellisse: se è bianco allora si instanzia un nuovo raggruppamento, si sceglie un nuovo colore e viene assegnato a questo gruppo. Poi viene applicata la funzione floodFill con il colore selezionato al centro dell'ellisse (Figura 2.14). Come già detto verranno colorate del colore selezionato le zone che hanno già un colore simile a quello del seed point, ovvero nel caso attuale le zone bianche, cioè altri raggruppamenti. Nel caso che il colore sia non bianco si scorre la nuova lista per trovare il raggruppamento responsabile della colorazione e si assegnano i punti del gruppo attuale a quest'altro. Al termine del procedimento si sono formati gruppi di punti più grandi di quelli originali, individuanti gli oggetti mobili della scena.

2.5.7 Passo 5: Bounding Rectangles

Figura 2.15: Bounding Box

A questo punto si può calcolare il Bounding Box di questi raggruppamenti, ciò viene fatto inserendo, per ogni ellisse i quattro punti che la caratterizzano in un vettore del gruppo: di questo vettore poi si calcola il rettangolo contenente con la funzione boundingRect di OpenCV. I rettangoli individuati vengono poi disegnati sull'immagine originale che sarà poi quella che verrà salvata e resa disponibile agli utenti (Figura 2.15).

2.5.8 Ottimizzazioni e correzione errori

Per fare fronte alle situazioni in cui gli oggetti individuati derivano da errori o possono essere ignorati sono stati inseriti diversi controlli. Un primo controllo viene effettuato dopo il passo 2: non tutti i contorni sono giudicati validi, infatti servono almeno 5 punti per poter definire un'ellisse. D'altronde un contorno di meno di 5 punti non appartiene certamente ad un oggetto valido e più probabilmente si tratta di un disturbo da eliminare. Un secondo controllo viene fatto nella fase 3: prima di inserire un'ellisse nel vettore corrispondente si valutano le dimensioni della stessa. In caso siano troppo ridotte, l'ellisse è etichettata come disturbo e non viene inserita. Terzo e ultimo controllo viene fatto quando si tratta di determinare i Bounding Box da disegnare. Il vettore dei potenziali rettangoli viene prima di tutto ordinato in base all'area, si procede poi a disegnare i rettangoli in ordine. A partire dal secondo rettangolo viene prima valutato se è intersecato nei rettangoli precedenti: in caso positivo non viene disegnato. La comunicazione della rilevazione di movimento avviene perciò solo se si sono superati tutti i controlli.

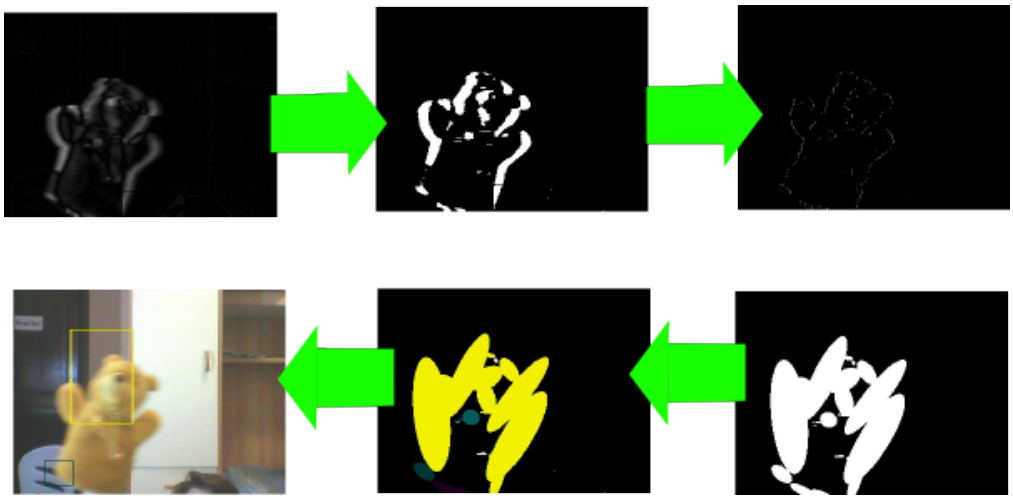

Figura 2.16: Catena di passaggi di MotionBubble

Capitolo 3

IoT Smart Camera

In questo capitolo verrà descritto il protocollo standard CoAP ed il suo funzionamento. Si parlerà poi dell'implementazione MjCoAP ed in particolare del suo utilizzo per lo sviluppo dei componenti software della SmartCamera.

3.1 CoAP - Costrained Application Protocol

Il Constrained Application Protocol (CoAP) è un protocollo software destinato a essere utilizzato in dispositivi elettronici molto semplici, che permette loro di comunicare interattivamente su Internet. E' particolarmente indirizzato a piccoli sensori a bassa potenza, interruttori, valvole e componenti simili che devono essere controllate o sorvegliate da remoto, attraverso le reti Internet standard. CoAP è un protocollo di livello applicativo (Figura 3.1) che è destinato per l'uso su dispositivi Internet con risorse limitate, come nodi WSN. CoAP è progettato per convertire facilmente HTTP [9] per l'integrazione semplificata con il web, ma anche per soddisfare requisiti specifici come supporto multicast, overhead molto basso e semplicità. Il Multicast,il basso overhead, e la semplicità sono estremamente importanti per dispositivi Internet of Things e Machine-to-machine, che tendono ad essere profondamente incorporati e hanno molta meno memoria e batteria di dispositivi Internet tradizionali. Pertanto, l'efficienza è molto importante. CoAP può funzionare sulla maggior parte dei dispositivi che supportano UDP o un protocollo analogo ad UDP. L'Internet Engineering Task Force Constrained RESTful environments (CoRE) Working Group ha fatto il maggior lavoro di standardizzazione per questo protocollo. Per rendere il protocollo adatto per applicazioni IoT e M2M, sono state aggiunte diverse nuove funzionalità. Il modello di interazione di CoAP è simile al modello client/server HTTP. Tuttavia l'implementazione CoAP del machine-to-machine vede spesso entrambi nei ruoli sia di client che di server. Una richiesta CoAP è equivalente ad una richiesta HTTP, e viene inviato da un

client per richiedere un'azione (con un codice di metodo) su una risorsa (identificata da un URI) su un server. Il server invia una risposta con un codice di risposta. Questa risposta può includere una rappresentazione della risorsa. A differenza di HTTP, CoAP si occupa di questi interscambi in modo asincrono su un trasporto orientato ai datagram come UDP. Questo viene fatto logicamente utilizzando uno strato di messaggi che supporta affidabilità opzionale (con esponenziale back-off).

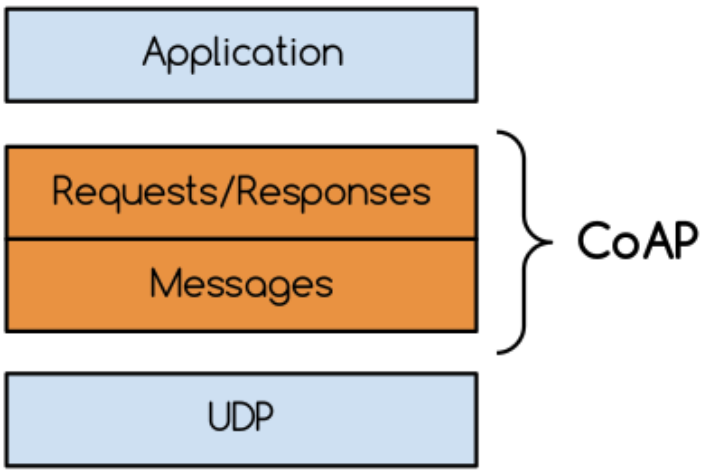

Figura 3.1: Layers

CoAP definisce quattro tipi di messaggi:

- confirmable
- non confirmable
- acknowledge
- reset

I codici di metodo e codici di risposta compresi in questi messaggi fanno in modo che si possano trasportare richieste o risposte. Gli scambi di base dei quattro tipi di messaggi sono in qualche modo ttrasversali alle interazioni di richiesta/risposta; le richieste possono essere effettuate in messaggi confirmable e non confirmable, e le risposte possono essere effettuate in questi così come piggy-backed nei messaggi di conferma. Si potrebbe pensare a CoAP utilizzando un approccio a due livelli:

un livello CoAP messaging utilizzato per trattare con UDP e la natura asincrona delle interazioni, e un altro livello di interazioni richiesta/risposta utilizzando i codici di metodo e di risposta. CoAP è comunque un unico protocollo, in cui messaging e richiesta/risposta sono solo caratteristiche dell'header (Figura 3.2).

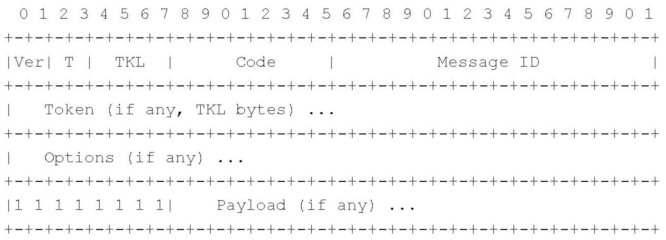

Figura 3.2: Message Format

3.1.1 Messaging Model

Il modello di messaggistica CoAP si basa sullo scambio di messaggi UDP tra endpoint. CoAP utilizza un header binario breve (4 byte) che può essere seguito da opzioni binarie compatte e un payload. Questo formato di messaggio è condiviso da richieste e risposte. Ogni messaggio contiene un ID di messaggio utilizzato per rilevare i duplicati e per affidabilità opzionale. Ecco i tipi di messaggio

- Confirmable (CON) Alcuni messaggi richiedono un riconoscimento. queste messaggi sono chiamati Confirmable. Quando non vengono persi pacchetti, ogni messaggio confirmable genera esattamente un messaggio di ritorno di tipo Acknowledgement

- Non confirmable (NON) Altri messaggi non richiedono un acknowledgement. Ciò è vero in particolare per i messaggi che si ripetono regolarmente, quali ripetute letture da un sensore.

- Acknowledge (ACK) Un messaggio di conferma riconosce che un specifico messaggio Confirmable arrivato. Non indica il successo o il fallimento di qualsiasi richiesta incapsulata

- Reset (RST) Un messaggio Reset indica che un messaggio specifico (Confirmable o non Confirmable) è stato ricevuto, ma mancano delle informazioni per poterlo elaborare correttamente. Questa condizione di solito è causata quando il nodo ricevente ha riavviato e ha dimenticato qualche stato che sarebbe stato necessari per inter- pretare il messaggio. Provocare un messaggio Reset (ad esempio, inviando un messaggio Confermabile vuoto) è anche utile come un controllo economico del liveness di un endpoint (CoAP ping).

3.1.2 Request/Response Message

Richieste CoAP e semantica della risposta sono trasportate nei messaggi CoAP, che comprendono o un codice di metodo o un codice di risposta, rispettivamente. Informazioni di richiesta o risposta, come l' URI o il tipo di payload vengono inviate come opzioni CoAP. Un token viene utilizzato per abbinare le risposte alle richieste in maniera indipendente dai messaggi sottostanti. Una richiesta viene effettuata in un Confirmable (CON) o in un messaggio non Confirmable (NON), e se immediatamente disponibile, la risposta ad una richiesta effettuata in un messaggio Confirmable viene trasportata nel risultante messaggio di Acknowledgement (ACK).Questa è chiamata una risposta piggy-backed. In CoAP, una richiesta ha il suo metodo indicato nella sezione Codice del suo header. In realtà ci sono quattro metodi diverso, ereditato da HTTP, che possono essere utilizzati:

- GET: il metodo GET recupera una rappresentazione per l'informazione che attualmente corrisponde alla risorsa identificata dalla richiesta URI. Se la richiesta include una Accet Option, questa indica il content-format preferito per una risposta. Il metodo GET è sicuro e idempotente.
- POST: Il metodo POST richiede che la rappresentazione racchiusa nella richiesta sia processata. La funzione effettiva eseguita dal metodo POST è determinata dal server di origine e dipendente dalla risorsa di destinazione. Esso di solito si traduce nella creazione di una nuova risorsa o nell'aggiornamento della risorsa bersaglio. POST non è né sicuro né idempotente.
- PUT: IL metodo PUT richiede che la risorsa identificata dall' URI della richiesta sia aggiornata o creata con la rappresentazione allegata. Il formato della rappresentazione è specificato dal tipo di supporto e la codifica dei contenuti riportato nella Content-Format Option, èfornitodai codici di media type e content dati nella Content-Format Option, se fornita. PUT non è sicuro, ma è idempotente.
- DELETE: IL metodo DELETE richiede che la risorsa identificata da la richiesta URI sia eliminata. DELETE non è sicuro, ma è idempotente.

Le risposte in CoAP sono identificate dal campo codice (che indica il codice di risposta) nell'intestazione. I codici di risposta sono usati per indicare al client il risultato del tentativo di soddisfare la richiesta. Il campo codice è lungo 8-bit . I 3 bit superiori indicano la classe del codice di risposta. Le classi di risposta CoAP sono:

- 2 - Successo, indica che la richiesta è stata ricevuta correttamente, intesa e accettata.

- 4 - Errore client, indica che la richiesta contiene cattiva sintassi o non può essere soddisfatta.
- 5 - Errore del server, indica che il server non è riuscito a soddisfare una richiesta apparentemente valida.

  I 5 byte inferiori forniscono ulteriori dettagli circa la risposta.

Nel caso più semplice, la risposta viene inviata direttamente nel messaggio di Acknowledgement che conferma la richiesta (ciò implica che la richiesta sia stata effettuata in un messaggio Confirmable). Questa si chiama una risposta Piggy-backed (Figura 3.3). La risposta viene restituito nel messaggio Acknowledgement indipendentemente dal fatto che la risposta indichi il successo o il fallimento. In effetti, la risposta è piggy-backed sul messaggio di conferma, e non è richiesto che sia inviato un messaggio separato per restituire la risposta.

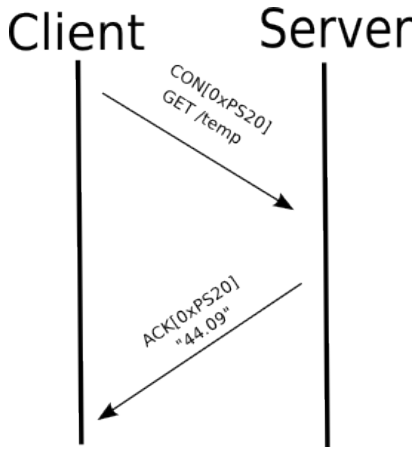

Figura 3.3: Risposta piggy-baked

3.2 MjCoAP

MjCoAP è una implementazione completa Java-based del protocollo CoAP sviluppata all'interno del Dipartimento di Ingegneria dell'Informazione dell'Università di Parma (http:
mjcoap.org). Include tutte le classi e i metodi per creare una applicazione CoAP-based, implementa la completa architettura ed è completamente conforme con il draft 18. MjCoAP è formato da divesi packages che includono:

- oggetti standard come CoAP Messages, Transactions, Providers, etc.
- varie estensioni come le CoAP Options

- oggetti di networking come IpAddress, SocketAddress e oggetti per gestire pacchetti UDP

Le classi principali di CoAP sono:

- CoapMessage: codifica e decodifica i CoAP Message in e da buffer binari.
- CoapProvider: CoAP servizio di message communication per inviare e ricevere CoAP messages a livello UDP.
- CoapReliableTransmission: processa il pacchetto ricevuto e fornisce ritrasmissione in caso di timeout della richiesta.
- CoapTransactionClient:gestisce la comunicazion lato client.
- CoapTransactionServer: gestisce la comunicazion lato server.

3.3 RpiMotionServer

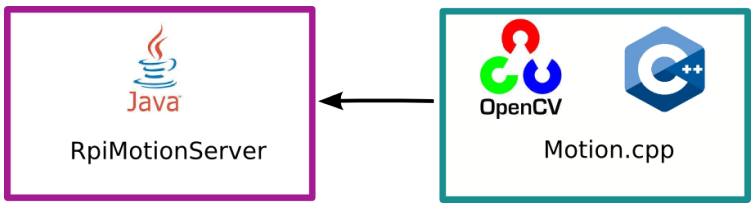

Figura 3.4: Il servizio fornito dalla Rpi è costituito da due parti cooperanti: la parte C++/OpenCV si occupa della visione, la parte Java/CoAP si occupa di gestire le risorse e di interagire con gli altri SmartObject

La peculiarità dell'IoT SmartCamera è quella di essere uno SmartObject: come tale interagisce con altri SmartObject utilizzando la rete e il protocollo standard CoAP. A differenza di altre SmartCamera non IoT infatti, l'accesso alle risorse e ai servizi non è mediato da una interfaccia rigida, ma avviene semplicemente attraverso le richieste/risposte previste da CoAP. In maniera complementare, la parte di visione si occupa di rilevare le informazioni di movimento dall'ambiente e di tenerne traccia acquisendo immagini. Il funzionamento della IoT SmartCamera è basato dunque su queste due componenti principali che lavorano in sinergia sulla Raspberry Pi e che sono rappresentate dai seguenti applicativi:

- MotionBubble: software in C++ che si occupa della rilevazione degli oggetti in movimento e dell'acquisizione di immagini

- RpiMotionServer: software in Java che gestisce la comunicazione con il protocollo CoAP

In particolare, l'IoT SmartCamera gestisce due tipi di risorse:

- la presenza/assenza di movimento
- l'URL dell'ultima foto scattata in condizione di presenza di movimento

Queste vengono continuamente aggiornate da MotionBubble e possono essere ottenute semplicemente tramite una richiesta GET di CoAP da parte di un qualsiasi SmartObject che effettui la corretta richiesta CoAP.

3.3.1 Comunicazione MotionBubble-RpiMotionServer

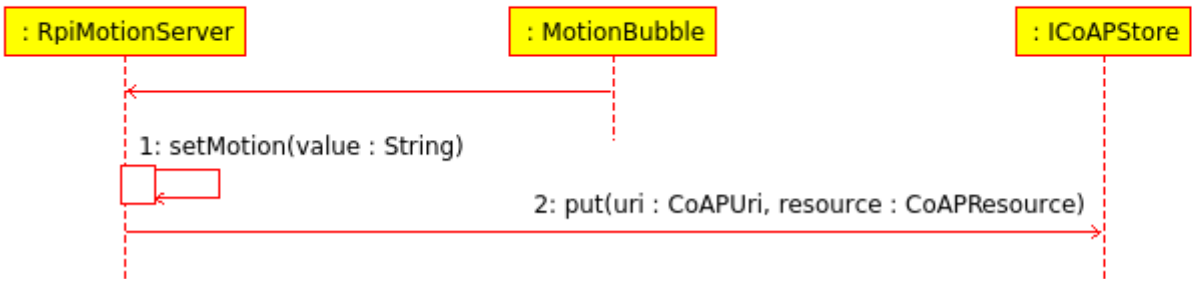

Figura 3.5: Aggiornamento risorsa motion

I due applicativi comunicano tramite una Socket locale, con RpiMotionServer come server e MotionBubble come client. RpiMotionServer, quando viene avviato, lancia su processo figlio una istanza di MotionBubble e si mette in ascolto finchè non riceve la richiesta di connessione da quest'ultimo. Una volta che la connessione è avvenuta, RpiMotionServer resta in ascolto su un thread dedicato attendendo aggiornamenti sui cambiamenti delle risorse interessate. In particolare da MotionBubble arrivano continui aggiornamenti sull'attuale condizione di movimento. In corrispondenza di una rilevazione di movimento positiva viene in oltre inviato a RpiMotionServer anche il nuovo url della foto: ciò invece non avviene nel caso opposto (rilevazione ad esito negativo) perchè non sarebbe utile, in quanto, posto che la rilevazione sia corretta, la scena dovrebbe essere priva di movimento e perciò non interessante. In realtà ad ogni aggiornamento ricevuto da MotionBubble

non corrisponde un effettivo aggiornamento della risorsa nello Store. Se c'è stata una rilevazione di movimento allora la risorsa verrà aggiornata istantaneamente e si terrà traccia dell'istante di aggiornamento (Figura 3.5). Ma un aggiornamento di stato negativo non potrà essere trasmesso allo Store a meno di n secondi dall' ultimo aggiornamento positivo: questo ritardo è stato inserito per permettere una consultazione più flessibile della risorsa. In uno scenario, come in quello che si vedrà nel capitolo successivo, di rilevazioni incrociate può essere utile settare il parametro n ad un valore diverso da zero perchè la rilevazione della camera potrebbe essere consultata dopo una rilevazione di altro tipo per avere una conferma della correttezza del segnale.

3.3.2 Struttura interna RpiMotionServer

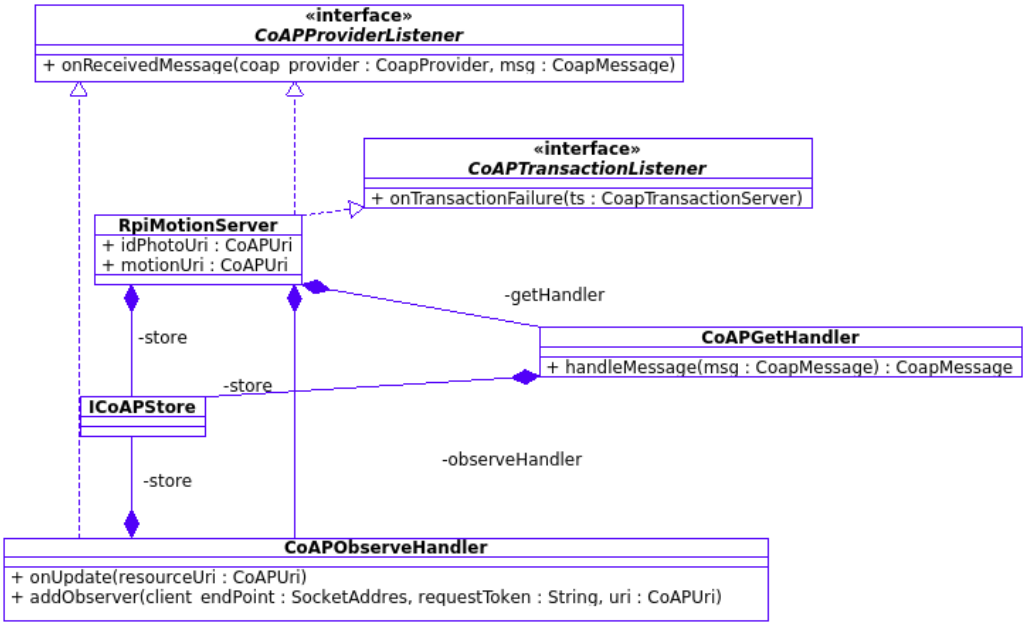

Figura 3.6: Class Diagram della struttura interna di RpiMotionServer

RpiMotionServer come detto prima implementa i protocolli CoapProviderListener, CoapTransactionServerListener e si configura quindi come un server CoAP (Figura 3.6). Le richieste in arrivo dai client nella rete sono gestite dal metodo

OnReceivedMessage(). A sua volta questo metodo delega la GET ad un oggetto già predisposto al suo soddisfacimento, un CoAPGetHandler(Figura 3.7).

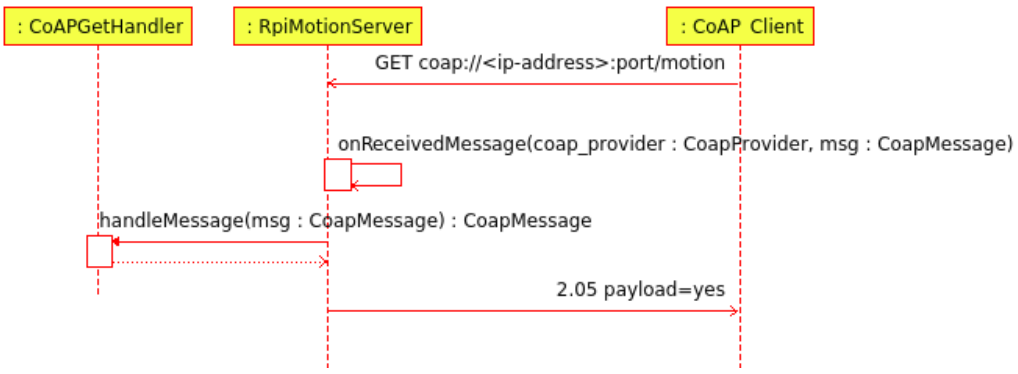

Figura 3.7: Esempio di richiesta GET

Il CoAPGetHandler consulta lo Store,poi costruisce e invia il messaggio di risposta. Inoltre OnReceivedMessage() inoltre controlla la presenza all'interno della richiesta della ObserveOption: nel caso sia presente con OBSERVE REGISTER VALUE con valore true, il client vuole essere registrato come Observer per la risorsa, viceversa vuole essere deregistrato. Tutto questo viene gestito da un altro oggetto l'ObserveHandler, che tiene al suo interno una lista di client registrati alle risorse. Se ci sono client registrati per la risorsa motion, quando avviene l'aggiornamento da MotionBubble viene chiamato il metodo OnUpdate() dell' ObserveHandler per inviare agli observer la notifica che conterrà nel payload la risorsa aggiornata (Figura 3.8). Non è previsto invece per quanto riguarda la risorsa idPhoto (URL della foto scattata) la registrazione come observer.

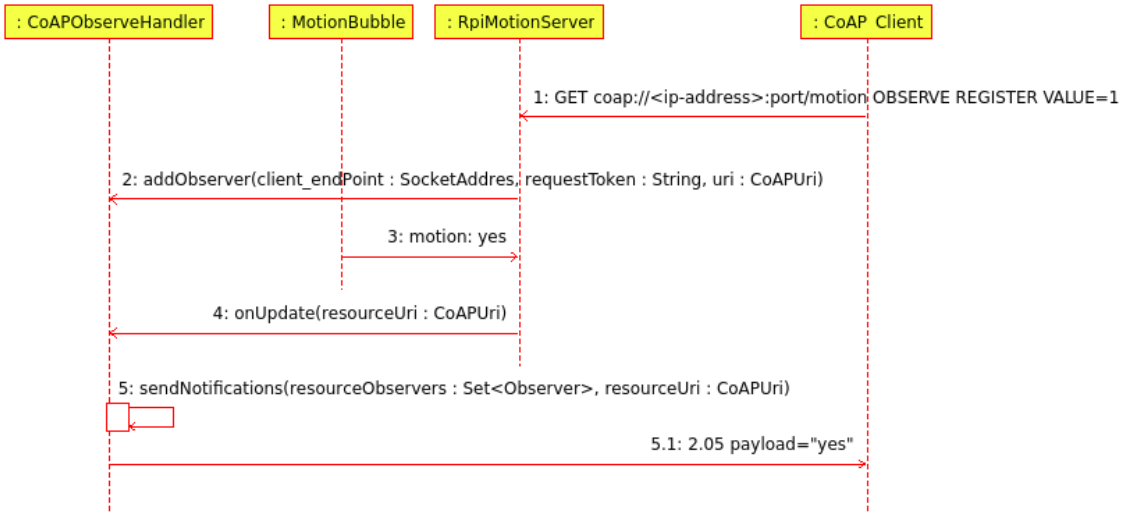

Figura 3.8: Esempio di Observer

Capitolo 4

Valutazione e Sperimentazione

In questo capitolo verranno valutate le prestazioni dell'algoritmo nel caso reale. Si farà un confronto dei tempi di esecuzione su Rasperry Pi e PC illustrando le motivazioni delle scelte progettuali. In seguito verrà presentato un possibile scenario in cui l'IoT SmartCamera, interagendo con altri SmartObject, realizza un sistema di sorveglianza efficiente.

4.1 Prestazioni camera

4.1.1 Scatto: OpenCV vs Raspistill

Raspistill è la funzione nativa della Rasperry Pi Camera per scattare foto [10]. Questa funzione nativa è molto veloce e molto flessibile: accetta infatti diversi parametri per modificare le impostazioni dello scatto (per esempio risoluzione, luminosità, esposizione, compressione, rotazione). Dai test non emerge nessuna differenza significativa per le foto in bianco e nero mentre raspistill risulta essere molto più veloce (1.2s) rispetto all'applicativo che utilizza OpenCV (2.5s) che sale a 5-6s se si cambiano i settaggi (w, h, luminosità ecc..). Per questi motivi raspistill verrà preferito nel caso di acquisizione di immagini singole ma verrà utilizzato comunque OpenCV per il processing di immagini in tempo reale: infatti il ritardo scompare dopo l'apertura del video, probabilmente poichè è causato dai driver UV4L. Nel caso della SmartCamera il file video è sempre aperto per consentire la rilevazione di movimento quindi il problema non si pone. E' presente anche una funzione nativa per l'acquisizione di video: raspivid. Tuttavia non permette di fare alcuna rilevazione di movimento quindi non è stata presa in considerazione.

4.1.2 Tempi

Sono stati misurari i tempi di esecuzione medi delle varie fasi dell'algoritmo MotionBubble lanciato su Rasperry Pi e vengono riportati di seguito:

Tabella 4.1: Tabella prestazioni di MotionBubble su Rpi: tempi per ogni fase

Operazione	1	2	3	4	5	6	7	8	9	10	Media
Acquisizione	10	10	15	15	10	14	11	11	11	11	11,8
Conversione	18	18	14	14	19	13	19	18	18	18	16,9
Absdiff	8	7	10	13	8	1	8	7	8	7	7,7
Copia	1	1	5	2	1	1	1	1	1	1	1,5
Thresh	6	9	5	6	5	5	7	6	5	6	6
Findcontours	13	13	12	12	14	11	13	17	13	13	13,1
Ellissi	4	13	2	4	1	1	2	7	3	3	4
Ordinamento	0	8	1	6	0	0	0	1	0	0	1,6
Colore	2	0	0	0	0	0	0	0	2	2	0,6
BoundingBox	0	0	0	1	0	0	0	0	0	1	0,2
Salvataggio	131	145	123	131	165	146	122	137	130	154	138,4
TOTALE	193	224	187	204	223	192	183	205	191	216	201,8

Si può notare che per quanto riguarda la Rpi(Figura 4.1) il collo di bottiglia è dato dalle operazioni di conversione, acquisizione e ricerca contorni: queste non sono ulteriormente ottimizzabili e quindi rappresentano una limitazione inferiore. Come si può notare l'operazione di generazione di ellissi non è particolarmente costosa computazionalmente. Si è verificato sperimentalmente che l'utilizzo dell'operatore dilation in alternativa, pur essendo più preciso è più costoso: i tempi di esecuzione infatti aumentavano velocemente con la dimensione del kernel arrivando a varie decine di ms. L'operazione quindi, accettabile computazionalmente su un computer con prestazioni standard, è stata scartata per l'implementazione su Rasperry Pi. Complessivamente un'iterazione dell' algoritmo dura circa 60-70ms. A questo tempo bisogna aggiungere quello necessario a salvare l'immagine, circa 130ms. Molto diversi sono invece i valori che compaiono nella tabella relativa allo stesso algoritmo eseguito su un PC (Figura 4.2). Mentre gli altri valori sono praticamente trascurabili infatti, il tempo di acquisizione è notevolmente più alto. In realtà il dato è ingannevole, infatti, nel tempo durante il quale il programma della Rpi esegue operazioni lente, viene preparato il frame successivo: ciò non può

Tabella 4.2: Tabella prestazioni di MotionBubble su PC: tempi per ogni fase

	1	2	3	4	5	6	7	8	9	10	Media
acquisizione	91	100	103	103	101	101	94	111	102	109	101,5
conv	0	1	0	0	0	0	0	1	0	2	0,4
absdif	0	0	0	0	0	0	0	0	0	0	0
copia	0	0	0	0	0	0	0	0	0	0	0
thresh	0	0	0	0	0	0	0	0	0	0	0
cont	0	0	0	1	1	1	1	0	1	1	0,6
ell	1	0	0	1	0	0	0	0	0	0	0,2
ord	0	1	0	0	0	0	0	0	0	0	0,1
col	0	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0	0
imwrite	0	4	5	3	4	5	4	11	3	3	4,2
TOTALE	92	106	108	108	106	107	99	123	106	115	107

accadere sul PC che è molto più veloce. Il tempo è stato dunque giudicato accettabile su Rpi: le prestazioni infatti sono equivalenti, se non migliori, rispetto ad un normale PC fintanto che non diventa necessario salvare l'immagine.

4.2 Scenario Applicativo e Sperimentazione

Lo scenario all'interno del quale è stata valutata la SmartCamera IoT progettata prevede l'interazione di sensori diversi per la rilevazione di movimento con un server centrale che raccoglie i dati e decide se lanciare o meno un allarme. L'ambiente in cui si immagina di installare questo sistema è un ambiente domestico, un magazzino o un negozio da sorvegliare nei momenti di chiusura. In ogni caso un area estesa e illuminata, dove possono essere sfruttate al meglio le caratteristiche della SmartCamera. Sono posizionate due coppie di PIR sulle pareti vicino alle porte, in modo da proteggersi a vicenda da manomissioni e quindi sorvegliare i punti di ingresso. La SmartCamera è invece posizionata in alto in un angolo dell'edificio in modo da poter tener sotto controllo la situazione. I sensori comunicano tramite protocollo CoAP i dati rilevati al server centrale rappresentato da uno SmartPhone. Quest'ultimo decide, in base ad una politica prestabilita, quando è il momento di lanciare l'allarme. Tramite SmartPhone l'utente riceve dunque il segnale di allarme e può visualizzare una foto in tempo reale della situazione.

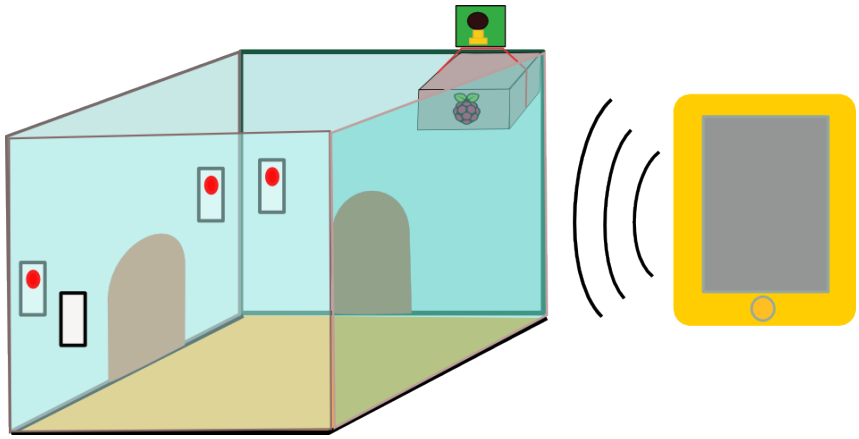

Figura 4.1: Magazzino: alle pareti sono installati dei PIR, in alto la SmartCamera

Si è quindi realizzato un sistema di sorverglianza che può essere schematizzato come di seguito:

- Un AlarmCheckerServer residente su smartphone.
- PIR client che ad intervalli di lunghezza casuale cambiano il loro stato e inviano l'aggiornamento.
- Una SmartCamera residente su Rpi che viene consultata per informazioni sul movimento e alla quale possono essere richieste foto.

4.2.1 PIR client

Nello scenario (Figura 4.1) sono presenti SmartObject con sensori PIR, connessi su rete IP, in grado di comunicare tramite protocollo CoAP, che agiscono da client. Come tali, non appena vengono avviati, lanciano un messaggio di POST con -pir come uriPath. Ogni volta che la risorsa del PIR subisce una variazione, il client invia una PUT al AlarmCheckerServer per aggiornarlo. I due stati che può assumere la risorsa pir sono detected e undetected.

4.2.2 AlarmCheckerServer

AlarmCheckerServer rappresenta il punto dal quale l'utente può controllare lo stato dei sensori. Ogni volta che un sensore PIR viene acceso e lancia una PUT il AlarmCheckerServer risponde assegnando al nuovo PIR un LocationPath in base al suo ordine di arrivo (Listing 4.1). In questo modo il server può identificare in modo univoco ogni sensore. Dal momento dell'assegnamento infatti, il PIR

client memorizza il LocationPath ed invia ogni messaggio di PUT con il proprio LocationPath.

Listing 4.1: AlarmCheckerServer assegna il LocationPath al PIR

```
if(msg.getCode() == CoapMethodCode.POST){
  if(new String(uriPath).equals("pir")){
    map.put(uri+"/"+countPir,
        new String(msg.getPayload()));
    int code = CoapResponseCode.responseCode(2, 1);
    CoapMessage resp =
     CoapMessageFactory.createResponse(msg, code, null);
    resp.addOption(new CoapOption
      (CoapOptionNumber.LocationPath,
       intToByteArray(countPir)));
    ts.respond(resp);
    countPir++;
  }
}
```

4.2.3 SmartCamera

L'altro sensore è rappresentato dalla SmartCamera. Come già detto nel capitolo precedente, per accedere alla risorse motion e photo basta effettuare una richiesta GET. Le modalità con cui AlarmCheckerServer mantiene aggiornate le risorse sono perciò diverse: nel caso dei PIR riceve in maniera passiva aggiornamenti PUT, mentre per quanto riguarda la camera deve inviare esplicitamente una richiesta GET. In realtà il AlarmCheckerServer può anche registrarsi come Observer: in questo modo riceve in automatico gli aggiornamenti della camera.

4.3 Valutazione Sperimentale

4.3.1 Rilevazione

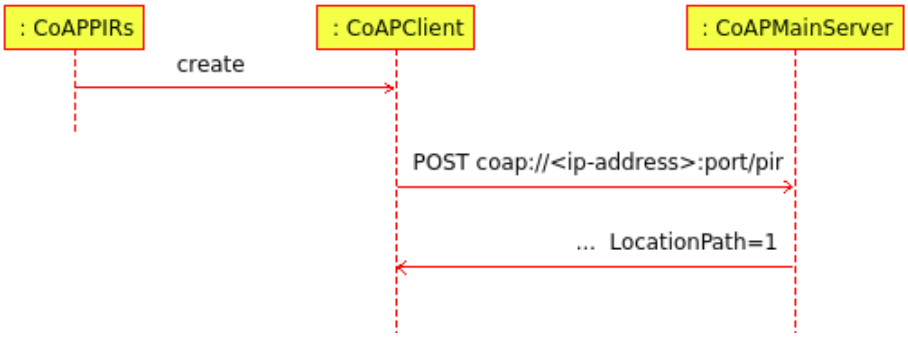

Figura 4.2: Registrazione del PIR presso il server

Per simulare l'attività dei client si è realizzato un applicativo Java che genera un determinato numero di thread (CoAPPirs) con l'obiettivo di emulare un insieme di SmartObject con sensore PIR che comunicano le loro rilevazioni. Ogni thread instanzia un CoAPClient e setta il suo stato ad undetected. Viene poi generato un numero casuale che corrisponde al numero di secondi che il thread deve attendere prima di invertire il suo stato e conseguentemente inviare la corrispondente PUT all'AlarmCheckerServer (Figura 4.2). L'AlarmCheckerServer memorizza in una HashMap al suo interno la coppia uri-valore. L'AlarmCheckerServer provvede inoltre a controllare il numero di PIR accesi per determinare o meno la necessità di un controllo sulla SmartCamera. Il criterio utilizzato è stato il seguente: se, dopo una PUT, il numero di PIR è in aumento ed è maggiore di 2 allora si passa al controllo della risorsa motion.

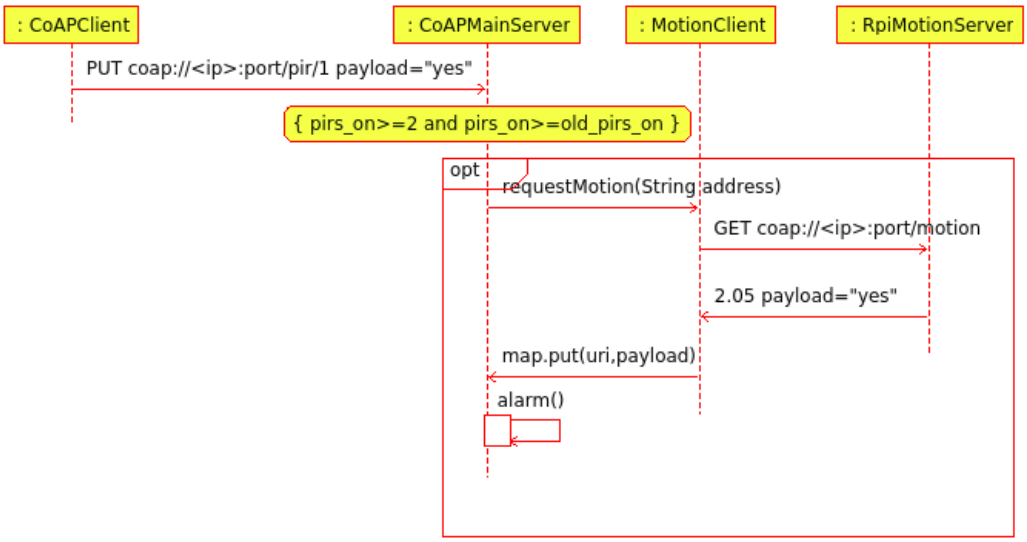

Figura 4.3: Esempio di interazione tra un client PIR, AlarmCheckerServer e RpiMotionServer. Il client invia una PUT a AlarmCheckerServer. Il server valuta il numero di PIR on e decide se chiedere conferma a RpiMotionServer

4.3.2 Allarme e invio foto

AlarmCheckerServer non genera immediatamente il segnale d'allarme ma provvede prima a consultare la SmartCamera. Se non ha mai interrogato prima la SmartCamera, il server utilizza il CoAPClient, che ha instanziato al momento della sua creazione, per inviare una richiesta GET a RpiMotionServer(Figura 4.3). Tuttavia la richiesta viene fatta con la Observe Option attiva, in modo da essere registrati come Observer. Perciò dopo la prima interrogazione in realtà non c'è più bisogno di utilizzare la GET: infatti ogniqualvolta la risorsa viene modificata, RpiMotionServer provvede ad inviare una notifica al server (Figura 4.4). Il AlarmCheckerServer riceve questa notifica all'interno dei suo MotionClient e provvede ad aggiornare lo stato della sua risorsa. Basterà perciò confrontare i risultati dei PIR con la propria risorsa motion per poter determinare la necessità di inviare il segnale di allarme.

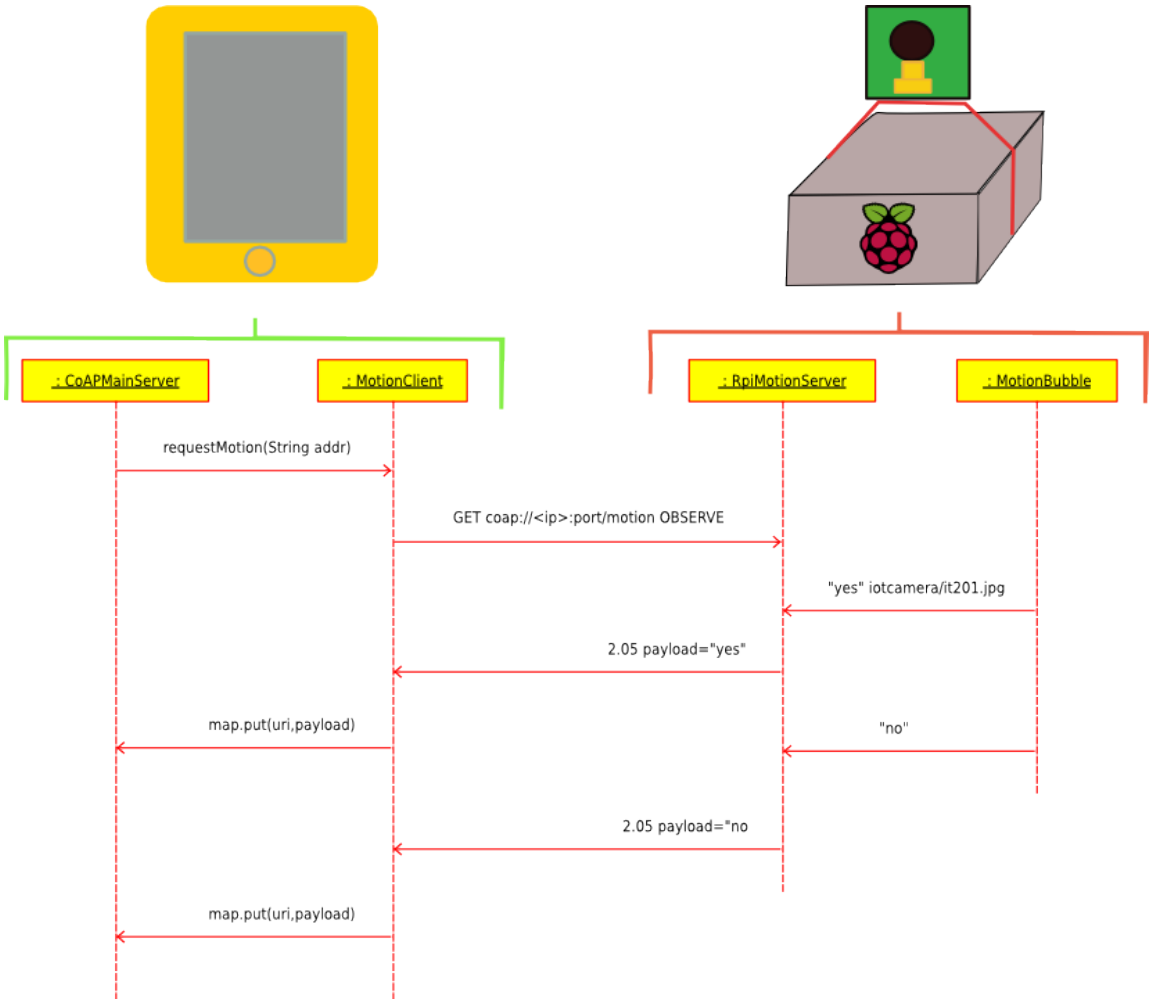

Figura 4.4: Esempio di registrazione come Observer di MotionClient a RpiMotionServer e invio delle notifiche

Conclusione

L'obiettivo di questa tesi è stata la progettazione di un IoT SmartObject che basandosi su algoritmi di visione artificiale, hardware a basso costo e protocolli di comunicazione standard consenta il monitoraggio in tempo reale dell'ambiente in cui si trova e l'interazione con altri nodi e servizi. Per fare questo è stato sviluppato un algoritmo, chiamato MotionBubble, che consente di individuare oggetti distinti in movimento all'interno di una scena, sfruttando in maniera opportuna le funzioni offerte dalla libreria OpenCV. L'algoritmo è inoltre in grado di conservare in memoria le immagini della scena in movimento. L'algoritmo implementato è stato integrato con un applicativo Java (RpiMotionServer) per aggiornare lo stato delle proprie risorse. RpiMotionServer gestisce inoltre la comunicazione e implementa lo stack protocollare che caratterizza i nodi di una rete IoT e si occupa dunque di soddisfare le richieste CoAP in arrivo da altri SmartObject. La Smart Camera IoT è in grado di rispondere a richieste GET per le risorse *motion* e *idPhoto*: ovvero condizione di movimento della scena e URL dell'ultima foto scattata. E' stato inoltre implementata la gestione di Observer esterni che si registrino per la risorsa *motion*: in questo modo la Smart Camera ha la possibilità di agire sia in un ruolo passivo che attivo all'interno della rete di SmartObject. Le scelte progettuali effettuate all'inizio del lavoro si sono dunque rivelate proficue. Raspberry Pi si è confermato un supporto ideale per nodi IoT e OpenCV ha dato la possibilità di valutare diverse soluzioni per i problemi affrontati, grazie alla presenza di molto materiale online, fornito da una nutrita community di sviluppatori. L'adozione del protocollo CoAP infine, che già dal alcuni anni è sperimentato all'Università di Parma, si è rilevata ancora una volta vincente: il protocollo è stato infatti dichiarato Standard da poco più di una settimana.

In realtà le potenzialità dell'algoritmo e del protocollo utilizzato lasciano ancora spazio per diversi possibili sviluppi futuri. In particolare attualmente le immagini possono essere ottenute soltanto tramite richiesta HTTP. Sarebbe perciò importante rendere disponibile la risorsa anche tramite richiesta CoAP: siccome si tratta di file di dimensioni elevate per il protocollo, è necessario l'utilizzo della CoAP Block Option che consente di inviare l'informazione richiesta in blocchi separati, i quali devono poi essere gestiti correttamente dal Client per una corretta rico-

struzione della risorsa originale. Un'altra importante integrazione potrebbe essere costituita dal Service Discovery. La Smart Camera IoT potrebbe in questo modo comunicare alla rete la sua esistenza, dando la possibilità di sviluppare comportamenti dinamici e intelligenti alla rete IoT in base agli SmartObject presenti. Per quanto riguarda la visione, attualmente La Smart Camera mette a disposizione come risorsa solo l'immagine della scena osservata: in seguito si potrà fare in modo di selezionare dall'immagine gli oggetti circondati dal Bounding Box e rendere disponibili le singole immagini come risorse.

Bibliografia

[1] *Internet of Things - Strategic Research Roadmap.* URL http://sintef.biz/upload/IKT/9022/CERP-IoTSRA_IoT_v11_pdf. pdf.

[2] J.A. Stankovic. *Wireless Sensor Networks*, volume 41. 2008. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4640674&isnumber=4640644.

[3] Nian Xiao-Hong Liu Yong-Min, Wu Shu-Ci. *The Architecture and Characteristics of Wireless Sensor Network*, volume 41. November 2009. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber= 5360030&isnumber=5359673.

[4] C. Schumacher N. Kushalnagar, G. Montenegro. *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs).* 2007. URL http://tools.ietf.org/html/rfc4919.

[5] Hartke K. Bormann C. Shelby, Z. *Constrained Application Protocol (CoAP).* 2013. URL http://tools.ietf.org/id/draft-ietf-core-coap.

[6] Gary Bradski Adrian Kaehler. *Learning OpenCV - Computer Vision in C++ with the OpenCV Library.* O'Reilly Media, 2013.

[7] *OpenCV API Reference.* URL http://docs.opencv.org/modules/refman.html.

[8] S. Suzuki and K. Abe. *Topological Structural Analysis of Digitized Binary Images by Border Following.*

[9] Gettys J. Mogul J. Frystyk H.-Masinter L. Leach P. Berners-Lee T Fielding, R. *Hypertext Transfer Protocol – HTTP/1.1.* 1999. URL http://tools.ietf.org/html/rfc2616.

[10] *Raspberry Pi Camera Module.* URL http://www.raspberrypi.org/documentation/raspbian/applications/camera.md.