ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienze Informatiche

INSPECTOR-PI
Progettazione e Sviluppo di un Sistema Mobile per il Controllo e il Monitoraggio basato su Tecnologie Embedded e Internet of Things

Elaborata nel corso di: Programmazione di Sistemi Embedded

Tesi di Laurea di:
SIMONE GALASSI

Relatore:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2014/2015
Sessione III

Alla mia famiglia, che mi è sempre stata vicino,

ai miei amici, che mi fanno evadere dalla quotidianità,

e alla mia ragazza, che ogni giorno mi sprona a dare il meglio.

Parole chiave

InspectorPi
Ispezione e monitoraggio
Internet of Things
Sistemi Embedded
Raspberry Pi
Android
Cloud

Indice

Introduzione **6**

Capitolo 1 - Quadro tecnologico di riferimento **8**

1.1 Internet of Things 8

1.1.1 Gli oggetti IoT e le loro applicazioni 10

1.1.2 I protocolli di comunicazione IoT 14

1.1.3 Il Cloud e le piattaforme IoT 15

1.2 I dispositivi Embedded 16

1.2.1 Comunicazioni tra sistemi Embedded 18

1.2.2 Tecnologie wireless abilitanti 19

1.3 La robotica 21

Capitolo 2 - Il progetto InspectorPi **25**

2.1 L'idea del progetto 25

2.2 Ambiti di utilizzo di InspectorPi 26

2.2.1 Monitoraggio di fughe di gas pesanti 27

2.2.2 Ispezione dei condotti di areazione 28

2.2.3 Utilizzo ricreativo 29

2.3 Funzionalità generali del sistema 30

2.3.1 Principali sottosistemi 32

2.4 Sensori integrabili 33

2.4.1 DHT11 e DHT22: sensori di temperatura e umidità 33

2.4.2 MQ-2 e MQ-5: rilevatori di gas 34

2.4.3 HC-SR04: sensore di distanza 35

Capitolo 3 - Analisi dei requisiti **37**

3.1 Casi d'uso del sistema 37

3.2 Analisi delle interazioni 38

3.2.1 Invio del comando riguardante il movimento 39
3.2.2 Visualizzazione dello streaming video in real-time 42
3.2.3 Visualizzazione dei sensori tramite piattaforma Cloud....44
Capitolo 4 - Il prototipo InspectorPi....46
4.1 La portabilità del progetto e il prototipo InspectorPi....46
4.1.1 Raspberry Pi come cuore di InspectorPi 47
4.1.2 Pi2Go, la kit-board su cui montare Raspberry Pi....48
4.1.3 Lo streaming video real-time 54
4.1.4 I sensori associati e il pinout finale 55
Capitolo 5 - Progettazione e implementazione....59
5.1 L'architettura del sistema....59
5.2 Modellazione e progettazione 61
5.2.1 Progettazione delle classi per InspectorPi....61
5.2.2 Progettazione delle classi per dispositivo Android 64
5.2.3 Progettazione degli stati per InspectorPi....67
5.2.4 Progettazione degli stati per dispositivo Android 70
5.2.5 Progettazione delle interazioni 72
5.3 Implementazione....75
5.3.1 Implementazione di InspectorPi....75
5.3.2 Implementazione di InspectorPi Controller 82
Capitolo 6 - Test, collaudo e verifica....87
6.1 Test iniziali sul prototipo 87
6.2 Collaudo del sistema....88
6.2.1 Problematiche riscontrate....88
6.2.2 Esecuzione del sistema....89
6.2.3 Reattività del sistema....89
6.3 Evoluzioni future 90
Conclusioni....91
Bibliografia e sitografia....93

Introduzione

Negli ultimi decenni si sono compiuti grandi passi nello sviluppo di dispositivi in grado di comunicare tra loro, nonché di trasmettere dati e informazioni agli utenti.

Grazie alle tecnologie ideate, è stato infatti possibile connettere in rete non solo computer, smartphone e tablet, ma anche numerosi oggetti di uso quotidiano: oggi il nostro frigorifero, il nostro termostato e i nostri occhiali, così come tutti gli altri device contenenti parti elettroniche, possono potenzialmente connettersi al resto del mondo attraverso Internet.

Questo processo d'interconnessione ha aiutato l'uomo ad affrontare la quotidianità, portando innovazione in numerosi campi, dalla domotica all'ingegneria militare. I nuovi sistemi hanno inoltre permesso una maggiore efficienza degli impianti industriali, rendendo le macchine autonome rispetto all'intervento dell'uomo: oggi i macchinari sono in grado di gestire e ottimizzare i processi produttivi, minimizzare i costi e moltiplicare i profitti. Proprio per questa ragione si sente sempre più spesso parlare di sistemi Embedded e Internet of Things, due concetti chiave, al centro delle evoluzioni tecnologiche in atto.

Ciò che abbiamo voluto fare nel corso di questa tesi è stato studiare queste innovazioni e sfruttarne le potenzialità per creare un nuovo progetto: la nostra sfida è stata quella di sviluppare un dispositivo che potesse riunire differenti funzionalità, dalla telepresenza alla visione dei dati on demand, e fosse in grado di portare innovazione nel panorama attuale.

In particolare, abbiamo deciso di realizzare un device che potesse svolgere attività d'ispezione e monitoraggio, concentrandoci nel corso dell'implementazione su alcuni possibili campi di utilizzo. Abbiamo così realizzato un sistema open-source dinamico, in grado di rispondere a esigenze diverse e facilmente riadattabile, garantendo modularità e duttilità. Il prototipo progettato sarà in grado di comunicare con uno smartphone, grazie al quale verrà guidato dall'utente primario, e di trasmettere in rete i dati rilevati dai diversi sensori che saranno integrati. Le informazioni generate saranno in seguito gestibili attraverso una piattaforma online: il device utilizzerà il Cloud per storicizzare i dati, rendendoli potenzialmente accessibili a chiunque.

Per la configurazione hardware abbiamo stabilito di usare la kit-board Pi2Go e la piattaforma Raspberry Pi, alle quali abbiamo unito una videocamera e alcuni sensori di

prossimità, temperatura e umidità e distanza. È nato così il prototipo InspectorPi, un veicolo telecomandato tramite dispositivo mobile in grado di esplorare ambienti ostili in cui vi sono difficoltà fisiche o ambientali alle quali sovvenire.

Struttura della tesi

Al fine di giungere al compimento di questa tesi, affronteremo diverse fasi di ricerca: dallo studio del quadro di riferimento alla definizione del progetto, dall'analisi del software fino alla realizzazione e al test del prototipo.

Il primo capitolo, in particolare, sarà un'introduzione ai concetti base. Racconteremo in dettaglio le tecnologie che ci hanno permesso di realizzare il nostro device, presentando Internet of Things, i sistemi Embedded e la robotica, delineandone sviluppo, meccanismi ed elementi caratterizzanti.

All'interno del capitolo 2 descriveremo quindi i caratteri generali del progetto, trattando l'idea di base e chiarendo le scelte fatte. In questa sede affronteremo i possibili ambiti di utilizzo del dispositivo ed elencheremo le componenti e i sensori necessari al suo corretto funzionamento.

Il terzo capitolo sarà incentrato invece sull'analisi del sistema. Attraverso l'uso di diagrammi UML, verrà descritto come si vorrebbe implementare il software, individuando i casi d'uso del sistema e specificando come dovrebbero avvenire le interazioni tra le parti. Quest'analisi costituirà il punto di partenza per la modellazione e programmazione del software vero e proprio, che sarà introdotto nel capitolo 5.

Il capitolo successivo si concentrerà poi sul prototipo: parleremo della portabilità del progetto, esponendo e giustificando le decisioni prese in materia di hardware, esponendo infine un elenco completo dei componenti che costituiscono InspectorPi.

Nel quinto capitolo andremo dunque a presentare l'architettura del sistema e, a partire da quanto esposto nel capitolo 3, a modellare il software. Verranno quindi riportate alcune parti significative di codice, riguardanti l'implementazione del software.

L'ultimo capitolo, in conclusione, riguarderà il collaudo del prototipo. Testeremo il device, analizzandone reattività, efficienza e rispondenza ai comandi, in modo tale da comprendere quali aspetti del progetto funzionano al meglio e quali sono invece i punti deboli del sistema sviluppato. In questo frangente i risultati ottenuti si mostreranno in larga parte positivi, facendo presupporre che, in futuro, InspectorPi potrebbe essere impiegato in numerosi ambiti e contesti.

Capitolo 1 -
Quadro tecnologico di riferimento

Nel corso del nostro progetto di tesi, ci siamo dovuti confrontare con l'utilizzo di una serie di tecnologie di natura differente. I principali oggetti d'indagine, grazie all'approfondimento dei quali abbiamo potuto sviluppare il nostro device, sono stati Internet of Things, i sistemi Embedded e la robotica contemporanea. Abbiamo infatti attinto informazioni fondamentali da tutti e tre questi ambiti di ricerca: la conoscenza dei metodi di condivisione dell'informazione, delle basi utili alla progettazione dei sistemi integrati e dei meccanismi che permettono l'automazione dei veicoli robotizzati è stata fondamentale per portare avanti il nostro progetto di ricerca.

All'interno di questo capitolo abbiamo quindi deciso di presentare questi campi d'indagine, fornendo una panoramica generale della loro evoluzione e dei caratteri che li contraddistinguono.

1.1 Internet of Things

Nel corso degli ultimi decenni si sono sviluppate tecnologie che hanno connesso a Internet, oltre ai computer a cui eravamo abituati, molti dispositivi che utilizziamo quotidianamente. Siamo ormai abituati alla possibilità che ogni oggetto contenente parti elettroniche possa comunicare con tutto il mondo attraverso la rete: edifici, device, veicoli e altri prodotti, integrando software e componenti hardware, possono rilevare informazioni dall'ambiente e, di conseguenza, sfruttando la connessione con altri dispositivi collegati in rete, produrre azioni utili all'utente.

L'estensione della connessione Internet anche agli oggetti che vengono utilizzati quotidianamente è oggi comunemente definita Internet of Things (IoT). Il primo a utilizzare il termine IoT fu l'imprenditore Inglese Kevin Ashton nel 1999[1], mentre lavorava nei laboratori Auto-ID (chiamati così riferendosi allo standard globale RFID che verrà introdotto più avanti); usò il termine per descrivere come i device connessi a Internet avrebbero cambiato le nostre vite. Previde che nel mondo del futuro i

[1] http://www.theguardian.com/media-network/2015/mar/31/the-Internet-of-things-is-revolutionising-our-lives-but-standards-are-a-must

dispositivi connessi ininterrottamente a Internet avrebbero fatto risparmiare all'uomo tempo e denaro. Nello stesso anno, Neil Gershenfeld scrisse il libro *When Things Start to Think*, il quale recita:

> "Beyond seeking to make computers ubiquitous, we should try to make them unobtrusive. […] For all the coverage of the growth of the Internet and the World Wide Web, a far bigger change is coming as the number of things using the Net dwarf the number of people. The real promise of connecting computers is to free people, by embedding the means to solve problems in the things around us."[2]

Gershenfeld aveva intuito che Internet, grazie all'innumerevole quantità di dispositivi connessi, ci avrebbe aiutato a risolvere i piccoli problemi relativi alla nostra quotidianità. Negli anni a venire si susseguirono diversi articoli e libri, che prevedevano che gli oggetti attorno a noi, attraverso la rete, avrebbero potuto cambiare il nostro modo di vivere; nel frattempo Internet entrava anche nei nostri telefoni cellulari.

Oggi, quindici anni dopo, l'idea di Ashton è diventata mainstream, e abbiamo la possibilità di collegare alla rete numerosissimi oggetti di utilizzo quotidiano. Possono essere connessi anche animali e individui, stabilendo quindi, oltre alla connessione tra oggetti (Machine-to-Machine), anche quella con le persone (Machine-to-Human). Questo settore è quindi in grande crescita e gli investimenti sono sempre più ingenti. L'azienda Gartner ha stimato che dai 4,9 milioni di unità facenti parte della categoria IoT del 2015, si passerà a 6,3 nel 2016 (30% in più), fino ad arrivare a 20 milioni di unità nel 2020.

Categoria	2014	2015	2016	2020
Consumo	2,277	3,023	4,024	13,509
Commercio: Transettoriale	632	815	1,092	4,408
Commercio: Settore specifico	898	1,065	1,276	2,880
Totale	**3,807**	**4,902**	**6,392**	**20,797**

Tab.1 – Dispositivi Internet of Things installati per categoria (in Milioni di Unità)[3]

[2] Gershenfeld N., *When Things Start to Think*, Henry Holt & Co, 1999
[3] https://www.gartner.com/

Per quanto riguarda gli investimenti, vi sarà un incremento dagli 1,1 miliardi di dollari versati nel 2015 ai 3 miliardi che verranno spesi nel 2020. Questi dati fanno ben sperare per le innumerevoli aziende che stanno capitalizzando in questo campo; tra queste possiamo citare Cisco, uno dei maggiori leader del networking mondiale.

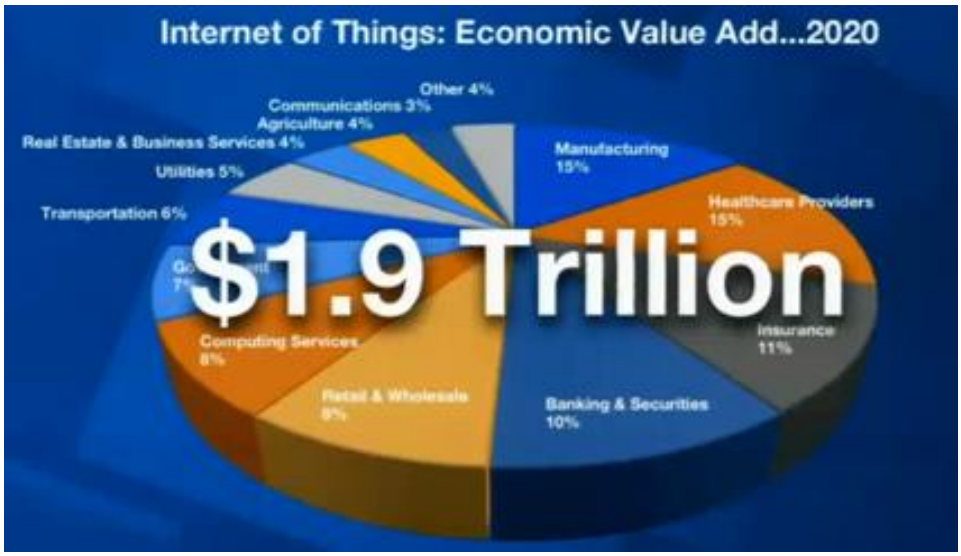

Fig. 1 – Aumento al 2020 sul mercato globale del fatturato associato a IoT[4]

1.1.1 Gli oggetti IoT e le loro applicazioni

Gli oggetti di uso comune possono entrare a far parte della categoria Internet of Things grazie all'integrazione di un'incalcolabile varietà di dispositivi in grado comunicare tra loro, impiegando tecnologie quali RFID, NFC, Bluetooth e Wi-Fi: sfruttando la sinergia di questi dispositivi è possibile interagire in modo maggiormente proficuo con gli oggetti che utilizziamo ogni giorno.

L'idea di Internet of Things è quindi quella di passare gradualmente da pochi dispositivi computerizzati molto potenti a molti dispositivi (un braccialetto, uno specchio, un frigo, un paio scarpe), apparentemente poco potenti, che collaborano insieme per rendere più semplici le azioni e le previsioni quotidiane. Un esempio significativo può essere quello dell'ombrello: se questo oggetto venisse reso "intelligente", potrebbe consultare le previsioni del tempo e, quando usciamo di casa, avvisarci della necessità o meno di portarlo con noi[5].

[4] https://www.gartner.com/
[5] www.materious.com/#/projects/forecast/

Per creare un dispositivo che possa collaborare con IoT, però, non basta dotarlo di un hardware capace di comunicare. Per entrare a far parte della categoria IoT, un oggetto necessita delle seguenti caratteristiche[6]:

- **Intelligenza**: un microprocessore o microcontrollore in cui viene eseguito un software crea quella "scintilla" che rende il nostro prodotto capace di elaborare le informazioni, rendendolo smart. Consideriamo per esempio il Misfit Shine, una smart-band per il fitness, confrontandola con il termostato intelligente della Nest. Mentre la smart-band ha bisogno di poca potenza di calcolo perché deve solo informare il telefono del lavoro svolto dalla persona che indossa il braccialetto, il termostato ha bisogno di più potenza di calcolo per amministrare l'AI che rende intelligente l'autogestione controllata dell'oggetto.

- **Connettività**: la connettività IoT, come abbiamo anticipato, è qualcosa di più che avere un modulo Wi-Fi e utilizzarlo una volta al giorno: è accessibilità alla rete e compatibilità. L'accessibilità ci inserisce nella rete, mentre la compatibilità ci dà la possibilità prelevare e mandare dati all'interno della rete, permettendo di trarre beneficio sia al dispositivo, sia a chi è connesso alla rete. Recita infatti così la legge di Robert Metcalfe, l'ideatore delle reti Ethernet: "L'utilità e il valore di una rete sono pari a «n^2 - n», dove 'n' è il numero degli utenti"[7]. Dalla compatibilità del dispositivo, dunque, trae vantaggio tutta la rete.

- **Rilevazione**: tendiamo a dare per scontati i sensi e le capacità che abbiamo di percepire il mondo intorno a noi. I sensori ci danno la possibilità di dare la coscienza del mondo fisico anche ai dispositivi, fornendoci ancora più informazioni su ciò che ci circonda.

- **Espressione**: gli attuatori permettono ai sistemi l'interattività con le persone e con il mondo fisico. Che si tratti di una casa intelligente o una fattoria con agricoltura intelligente, l'espressione attraverso schermi, bottoni, o più generalmente comandi, ci permette di creare un dispositivo che interagisce con noi in modo comprensibile. Questo non significa soltanto creare delle

[6] https://designmind.frogdesign.com/2014/08/Internet-things-six-key-characteristics/
[7] http://www.dse.uniba.it/Corsi/docenti/Coco/Lezione%205.pdf

interfacce utente in uno schermo, ma anche riuscire a interagire con l'ambiente circostante, che si tratti di oggetti, piante o animali.

- **Energia**: senza energia non è possibile far funzionare i nostri dispositivi. Il problema è che non possiamo creare miliardi di oggetti che funzionano a batterie. L'immagazzinamento di energia, l'efficienza e i metodi di ricarica a basso costo sono strumenti necessari, che in futuro dovranno essere ottimizzati per costituire ecosistemi intelligenti basati su questi dispositivi.

- **Sicurezza**: sia il dispositivo che crea le informazioni, sia il destinatario (persone, server, Cloud) devono essere progettati in modo da essere sicuri: è necessario garantire la sicurezza dei dati personali dell'utente e il suo benessere fisico. Proteggere gli apparati, la rete e le comunicazioni significa quindi creare un paradigma di sicurezza scalabile, dove ogni apparato aggiunto dovrà rispettare certi requisiti, così da non dover riprogettare la sicurezza degli altri elementi.

Gli oggetti conformi a questi parametri, a seconda delle loro caratteristiche intrinseche, possono essere impiegati in numerosi campi di utilizzo. Gli ambiti di applicabilità vanno infatti dalle applicazioni produttive alla logistica, dalla mobilità all'efficientamento energetico, dal supporto a distanza alla tutela degli ecosistemi. Di conseguenza, i domini applicativi e gli ambiti operativi che traggono vantaggio dalla comunicazione tra i dispositivi IoT sono vari e numerosi[8]:

- Domotica
- Robotica
- Avionica
- Industria automobilistica
- Ingegneria biomedica
- Monitoraggio in ambito industriale

[8] http://www.innovaway.it/iot-Internet-of-things-lInternet-delle-cose/

- Telemetria
- Reti wireless di sensori
- Sorveglianza
- Rilevazione di eventi avversi
- Smart City
- Sistemi Embedded
- Telematica

Tra questi, uno dei più significativi è il campo della produzione industriale, nel quale la diffusione di IoT ha apportato grandi innovazioni. Le strutture produttive collegate in rete, infatti, sono maggiormente efficienti, in quanto permettono una continua e immediata riparametrazione e ottimizzazione della fabbricazione, che, in base alle esigenze di mercato, si modifica in tempo reale. Grazie al Cloud, inoltre, è possibile tenere costantemente d'occhio lo stato delle risorse, il sistema di produzione e le informazioni relative alla sua funzionalità.

Accanto alla produzione aziendale, uno degli ambiti in cui si è riusciti a sfruttare al meglio queste nuove tecnologie è poi la domotica. Ogni oggetto domestico di utilizzo comune è infatti un potenziale candidato IoT: grazie alla connessione dei dispositivi, si possono gestire in modo controllato le risorse facenti parte dell'unità abitativa, come uno specchio, la lavastoviglie o l'impianto di riscaldamento. Agendo in quest'ottica, è per esempio possibile monitorare e amministrare lo stato della casa in montagna a distanza, riscaldarla in tempo prima dell'arrivo o eseguire le pulizie in remoto tramite l'uso di robot, mettendo quindi in atto un controllo molto più pervasivo di quanto non fosse possibile in precedenza.

Un altro settore rilevante tra quelli elencati è infine quello delle Smart City, ovvero le città intelligenti, che integrano le nuove tecnologie nella gestione del suolo pubblico (mobilità, ambiente, energia, economia). In quest'ambito, Internet of Things si configura come una risorsa per amministrare la città in modo più efficiente e migliorare la qualità della vita dei cittadini: grazie all'utilizzo di una rete di sensori capaci di misurare e controllare numerosi parametri, è per esempio possibile

controllare i livelli di inquinamento dell'aria, il dispendio di energia elettrica della rete d'illuminazione stradale e lo stato del traffico.

1.1.2 I protocolli di comunicazione IoT

Dato che Internet of Things è un concetto relativamente nuovo, anche i protocolli di comunicazione che si utilizzano in quest'ambito sono stati ideati in tempi recenti e, per questo, in larga parte sono ancora in fase di progettazione. Tra quelli attualmente esistenti, i più importanti sono[9]:

- **HTTP/REST**: questo protocollo ASCII è a oggi il più comune per la connessione dei device al Cloud; l'architettura prevede una comunicazione client-server, priva di stato. Può rivelarsi inefficiente per la comunicazione tra dispositivi, in quanto, per ogni comunicazione, invia grandi quantità di dati a margine, anche quando l'informazione da trasmettere è breve.

- **CoAP**: è un protocollo HTTP-like, ma, invece di utilizzare il formato ASCII, sfrutta un formato binario. In questo modo è in grado di evitare parte dell'overhead implicato nell'uso di HTTP.

- **UPnP**: il protocollo Universal Plug and Play è stato creato per semplificare l'uso dei device all'interno della rete. Questo standard, infatti, attraverso il meccanismo di configurazione dinamica detto zero-configuration, senza sfruttare il servizio fornito da un server DHCP, permette all'utente di impiegare un dispositivo non appena viene immesso in rete. Una volta ottenuto un indirizzo IP, infatti, ogni device è in grado di mostrare le proprie funzionalità e conoscere le caratteristiche degli altri dispositivi connessi alla rete.

- **MQTT**: il Message Queue Telemetry Transport è un protocollo nato per la telemetria, molto leggero e affidabile. Utilizza un intermediario chiamato "broker", che si occupa di smistare le informazioni tra i richiedenti.

- **XMPP**: l'Extensible Messaging and Presence Protocol è usato principalmente per la messaggistica istantanea e sfrutta il sistema XML. È un

[9] Waher P., *Learning Internet of Things*, Packt Publishing, 2015

sistema aperto che impiega un metodo di condivisione simile a quello delle e-mail e può essere personalizzato dagli sviluppatori.

Tra i protocolli sopracitati ve ne sono alcuni in grado di funzionare meglio rispetto agli altri: i più efficienti sono quelli publish/subscribe, come CoAP, MQTT e XMPP, che, sfruttando un intermediario, permettono di separare il sistema dall'utente che richiede le informazioni.

1.1.3 Il Cloud e le piattaforme IoT

Per far sì che i dati e le informazioni trasmesse da IoT siano sempre disponibili e reperibili dagli utenti, è importante che questi vengano immagazzinati in rete. Questo processo è possibile grazie al Cloud, un sistema grazie al quale gli utenti possono consultare, aggiornare e sincronizzare i propri contenuti e le proprie risorse all'interno di un server online. I caratteri che definiscono il Cloud sono quindi l'accessibilità immediata e globale da parte dell'utente, nonché la possibilità di condivisione delle informazioni e il loro monitoraggio.

I servizi che possono essere distribuiti attraverso una piattaforma Cloud, invece, sono di tre tipologie differenti:

- **IaaS** (Infrastructure as a Service): viene fornita una virtualizzazione delle risorse hardware (RAM, CPU, dischi e schede di rete), che premette di ottenere un ampio grado di flessibilità dell'infrastruttura fisica senza dover modificare le componenti hardware. Si può inoltre scegliere quale sistema operativo installare e quali applicazioni creare.

- **PaaS** (Platform as a Service): il servizio fornisce al cliente l'infrastruttura di base, sulla quale potrà sviluppare le proprie applicazioni e i propri programmi, utilizzando il database fornito e i linguaggi supportati dalla piattaforma.

- **SaaS** (Software as a Service): è il servizio che permette di lavorare al più alto livello: l'applicazione usata dall'utente finale è messa a disposizione direttamente dal fornitore.

L'utilizzatore sceglierà quale tipo di servizio impiegare in base alle proprie esigenze e ai protocolli scelti.

1.2 I dispositivi Embedded

Gli oggetti IoT, in particolare, sono sistemi Embedded in grado di comunicare all'interno di una rete condivisa e sfruttarne le potenzialità. Nello specifico, si definiscono dispositivi Embedded tutti quei sistemi elettronici di elaborazione che, tramite l'utilizzo di uno o più processori, una piattaforma hardware e un software dedicati, servono un'applicazione specifica; questi dispositivi, al contrario dei computer generici, non possono essere riprogrammati per altri scopi.

Sono utili in particolare perché, svolgendo in modo univoco il compito per il quale sono stati creati, riescono a garantire la massima efficienza. Le loro caratteristiche principali, che li distinguono rispetto a un sistema general-purpose, sono le seguenti:

- la loro funzione primaria è differente da quella di un computer generico;
- le funzioni amministrate dal sistema sono definite a priori;
- grazie all'ottimizzazione dell'hardware, evitano sprechi di risorse e riducono i costi di produzione;
- il software è più complesso tanto più lo sono le funzionalità che deve eseguire;
- la struttura del sistema può variare grandemente a seconda dello scopo dell'applicazione.

Alcuni esempi di dispositivi Embedded sono i lettori mp3, gli elettrodomestici intelligenti, gli ATM e i sistemi ABS. Tutti questi dispositivi, infatti, sono sviluppati e programmati appositamente per svolgere determinate funzioni.

Nonostante la varietà dei sistemi esistenti, però, i dispositivi Embedded sono accomunati dalla loro architettura hardware: ogni elemento è composto da CPU, RAM, ROM, sensori e attuatori, interfacce di comunicazione e circuiti a uso specifico.

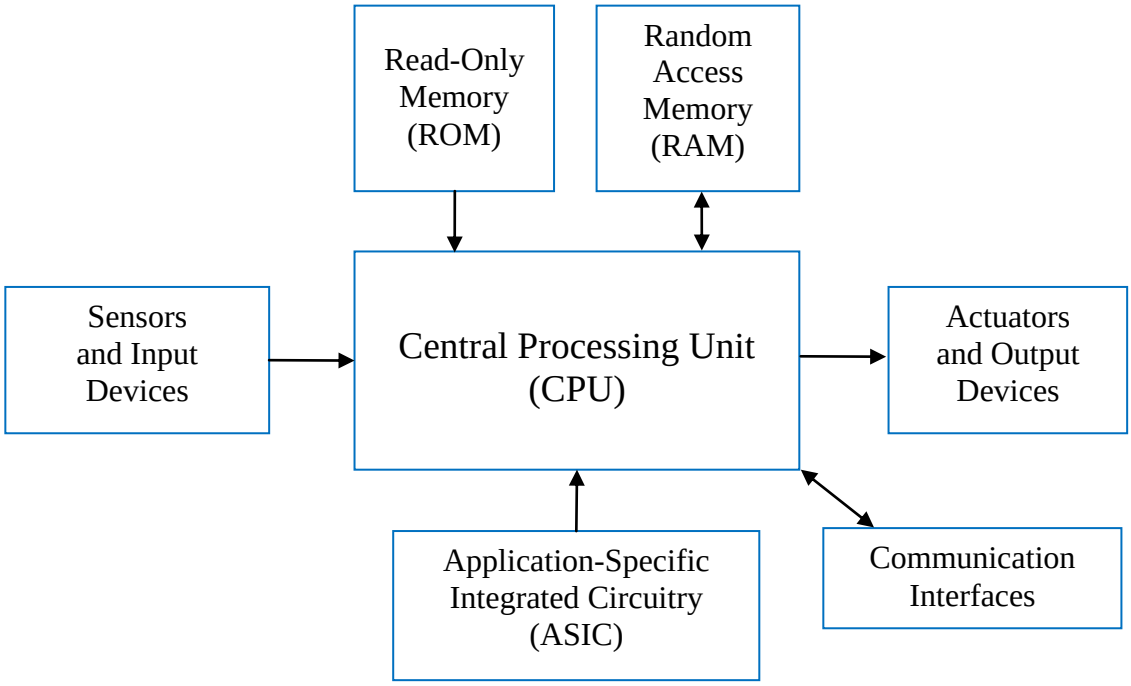

Fig. 2 – Schema a blocchi dell'architettura hardware di un sistema Embedded

Questi elementi, rappresentati nello schema a blocchi in figura 2, sono in genere di tipologie anche molto differenti tra loro. I sensori, per esempio, possono impiegare numerosi tipi di tecnologie: esistono sensori ottici e sensori di posizionamento, sensori di contatto e sensori di movimento.

L'unico elemento che può essere configurato solo in tre modi distinti è la Central Processing Unit (CPU). In particolare, i processori che si possono utilizzare per la sua configurazione sono i seguenti:

- **Single-purpose**: sono circuiti digitali progettati per adempiere a una specifica funzionalità o eseguire un particolare programma.

- **General-purpose**: sono i processori più complessi; possiedono un set di istruzioni predefinito e un'architettura di base articolata.

- **Application-specific**: a metà tra single-purpose e general-purpose, questi processori, progettati in base alle esigenze di calcolo delle applicazioni in cui saranno impiegati, sono programmabili per funzionalità specifiche.

Della categoria degli Application-Specific Processors (ASIP), in particolare, fanno parte i microcontrollori (MCU), come quello presente in Arduino, e i System-on-a-Chip (SoC), come quello di Raspberry Pi. Arduino e Raspberry Pi sono i dispostivi elettronici più diffusi per realizzare prototipi ed effettuare test; il loro hardware è di piccole dimensioni: grazie a esso, si riducono le tempistiche di elaborazione, i consumi energetici e i costi di produzione delle schede. Nello specifico, possiamo descrivere questi due esempi di tecnologie ASIP come segue:

- **Arduino**: è una scheda a microcontrollore progettata in Italia nel 2005, che, grazie all'IDE fornito dall'azienda, permette di sviluppare semplici progetti senza la necessità di possedere vaste competenze di programmazione informatica. Il suo hardware permette di gestire diversi I/O, sia analogici che digitali. Viene utilizzato nello specifico per applicazioni che non devono eseguire funzionalità eccessivamente complesse, in quanto il suo microcontrollore non è molto potente.

- **Raspberry Pi**: è una board con SoC sviluppata in Inghilterra e lanciata sul mercato nel 2012. Può montare diversi sistemi operativi, installabili separatamente, in quanto manca un'unità integrata allo stato solido. Il dispositivo, infatti, prevede l'utilizzo di una scheda SD per eseguire il sistema operativo scelto. Non dispone di molti pin I/O, ma prevede la possibilità di collegare dispositivi USB. Rispetto ad Arduino, questa scheda possiede una maggiore potenza di calcolo ed è progettata per applicazioni più complesse, con un esigente consumo di memoria.

1.2.1 Comunicazioni tra sistemi Embedded

La comunicazione tra sistemi Embedded è fondamentale per permettere ai dispositivi lo svolgimento dei compiti più complessi. Interagendo tra loro, infatti, i device sono in grado di trasmettersi l'un l'altro le informazioni necessarie a garantire una corretta esecuzione del sistema.

Dati i molteplici vantaggi forniti dall'efficienza di questi impianti, sono sempre più numerose le aziende che utilizzano dispositivi Embedded per ottimizzare i processi produttivi. Si pensi soltanto che la famosa società di commercio elettronico statunitense Amazon ha "assunto" 15.000 robot per spostare gli scaffali, che permettono al singolo dipendente di risparmiare 32 chilometri al giorno.

Oggi, infatti, all'interno di una fabbrica le macchine non si occupano più soltanto della realizzazione dei prodotti, ma sono anche in grado di controllare ed eventualmente correggere i processi produttivi. Prendiamo come esempio una fabbrica che produce tappi: se pensiamo al processo che avviene, possiamo immaginare in modo semplificato che la plastica venga scaldata e modellata da una macchina, definita macchina A, poi successivamente vengano controllati i tappi non conformi alle aspettative da un'altra macchina, definita macchina B, e vengano quindi scartati da quest'ultima quelli non idonei. Supponiamo ora che i tappi scartati diventino una percentuale significativa per l'azienda e quindi i parametri della macchina che li produce debbano essere reimpostati. Grazie alla comunicazione tra i dispositivi, se queste due macchine dispongono della tecnologia necessaria, non sarà necessario l'intervento di un tecnico: la macchina B segnalerà alla macchina A gli errori riscontrati e quest'ultima modificherà i propri parametri.

Questo tipo di comunicazione tra macchine, chiamata Machine-to-Machine, può avvenire non solo tramite connessioni via cavo, ma anche attraverso connessioni wireless, impiegate sempre più comunemente per via della loro comodità d'uso.

1.2.2 Tecnologie wireless abilitanti

Le tecnologie wireless sono oggi essenziali per permettere la comunicazione tra dispositivi senza dover ricorrere all'impiego di cavi. Sono usate sempre più spesso nei sistemi Embedded, ma non solo: molte delle connessioni wireless inizialmente create per i dispositivi Embedded, sono in seguito diventate essenziali anche per lo scambio di informazioni tra oggetti IoT.

Le comunicazioni wireless possono però avvenire in numerosi modi differenti, che si distinguono in base ai protocolli e alle tecnologie impiegate. Alcuni dei sistemi di comunicazione principalmente usati sono i seguenti:

- **Wi-Fi**: questo tipo di comunicazione sfrutta lo standard IEEE 802.11 e i protocolli TCP/IP. È la più comune in qualsiasi tipo di dispositivo: si può per esempio trovare all'interno di smartphone, smart tv e computer. La distanza che può coprire varia da qualche centimetro fino a poche centinaia di metri e solitamente viene adoperata per il collegamento a Internet. La connessione Wi-Fi può inoltre operare in due modalità, Ad-Hoc e infrastruttura: la prima permette la comunicazione diretta tra i device, senza prevedere l'utilizzo di un'infrastruttura già esistente, mentre la seconda sfrutta gli access point per

inviare le informazioni tra gli host. In quest'ultimo caso la larghezza di banda viene preservata e vi sono meno interferenze.

- **Bluetooth**: la sua rete di base è detta Piconet e sfrutta un'architettura master-slave, che permette la connessione anche tra più dispositivi: connettendo due Piconet, si possono amministrare più di sette device alla volta. Questo tipo di comunicazione è comunemente usata per trasmettere informazioni tra device non molto lontani l'uno dall'altro, distanti non più di dieci metri. Nonostante il suo raggio di azione sia piuttosto ristretto e la sua velocità sia moderata, grazie allo standard BLE (Bluetooth Low Energy) consuma meno del Wi-Fi, permettendo di comunicare anche a dispositivi che non possiedono grandi riserve di energia. Nata da uno standard industriale, è oggi impiegata quotidianamente da milioni di utenti in tutto il mondo, che la utilizzano per connettere tra loro i propri device, come per esempio auricolari e smartphone.

- **ZigBee**: sviluppata nel 2005, è più recente rispetto ai due tipi di comunicazione precedenti. Il suo raggio di azione è simile a quello del Bluetooth e utilizza lo standard IEEE 802.15.4, che permette di impiegare due tipi di nodi: FFD (Full-Function Device), che coordina i dispositivi, e RFD (Reduced-Function Device), meno potenti e in grado di interagire solo con nodi FFD. Grazie a questo standard, i device ZigBee riescono a collegarsi gli uni agli altri e ricoprire una vasta area, superando così i limiti di distanza della singola connessione.

- **RFID**: la tecnologia Radio Frequency Identification è stata ideata per il tracking degli oggetti a distanza. Utilizza dei dispositivi elettronici di dimensioni ridotte, chiamati tag, che possono essere identificati univocamente da specifici lettori RFID. Oggi sono impiegati sempre più spesso nell'ambito di IoT, perché, qualora ci sia prossimità, permettono di identificare un dispositivo all'interno di uno spazio condiviso in modo immediato. Si dividono in tre tipologie, distinguibili in base al tipo alimentazione: passiva, se alimentati dal RFID reader, attiva, se hanno batterie proprie e agiscono in broadcast, e ibrida se possiedono batterie proprie ma vengono attivati solo dal lettore RFID.

- **NFC**: la tecnologia Near Field Communication consente lo scambio di informazioni tra due dispositivi in contatto tra loro (alla distanza massima di dieci centimetri). La comunicazione può avvenire in modo bidirezionale e i dati possono essere trasmessi anche in modo passivo tramite l'uso di tag, come nel caso del RFID.

- **iBeacon**: si basa sulle tecnologie BLE e ha un trasmettitore a bassissimo consumo che permette ai device di essere identificati e localizzati all'interno di un'area limitata. Questo tipo di comunicazione è stata sviluppata dalla Apple.

È utile precisare che le comunicazioni wireless di tipo ZigBee, RFID, NFC e iBeacon, benché più efficienti rispetto ai sistemi Wi-Fi e Bluetooth, sono meno utilizzate in quanto meno diffuse. Per questa ragione, in genere, sono impiegate in ambito industriale e non dai singoli utenti.

1.3 La robotica

La robotica è una branca dell'ingegneria che può servirsi dei sistemi Embedded per creare automi in grado d'interagire con l'ambiente e muoversi al suo interno, come nel caso del prototipo sviluppato nell'ambito del nostro progetto di tesi.

Questa disciplina riunisce in sé elementi di meccanica, informatica ed elettronica; si occupa in particolare della progettazione di robot in grado di operare l'esecuzione fisica o decisionale di un compito anche a prescindere dal supporto umano. Secondo la definizione del Robot Institute of America, infatti, un robot è "un manipolatore funzionale riprogrammabile, progettato per muovere materiali, parti, attrezzi o dispositivi specialistici attraverso vari movimenti programmati, per l'esecuzione di diversi compiti".

I primi robot furono ideati a scopo ludico, ma in seguito il loro utilizzo si è ampiamente diffuso: vengono sfruttati in campo industriale per eseguire i compiti più ripetitivi e pesanti, per l'esplorazione spaziale e per rispondere a esigenze militari, ma anche per assistere i singoli utenti nello svolgimento dei lavori quotidiani. A seconda delle diverse applicazioni in cui vengono impiegati, possiamo quindi operare un'importante distinzione:

- La **robotica industriale**, nata per prima, studia lo sviluppo di macchinari in grado di sostituire gli operatori umani e rendere maggiormente veloci ed efficienti il processo produttivo e l'elaborazione delle informazioni. I reparti di saldatura e verniciatura dell'industria automobilistica sono stati tra i primi ad applicare la robotica su vasta scala.

- La **robotica avanzata** sviluppa macchine da utilizzare in ambienti ostili come lo spazio, gli oceani e le aree ad alto rischio per l'uomo, nonché robot utili a supportare l'utente nel corso delle proprie attività, per esempio in ambiente domestico, in campo medico o in campo agricolo. Appartengono dunque a questa categoria numerose sottodiscipline, tra cui la domotica, la microrobotica, la robotica biomedica, la robotica di intrattenimento, la robotica umanoide, la robotica marina, la robotica militare e la robotica spaziale.

Entrambi questi rami della robotica hanno subito negli anni grandi evoluzioni e oggi le tecniche di costruzione dei robot industriali e non sono sempre più sofisticate. In particolare, la ricerca tecnologica si è articolata in tre settori principali, che identificano le diverse funzionalità delle macchine: la tecnologia costruttiva, la capacità di ragionamento e l'integrazione con l'ambiente. La tecnologia costruttiva si occupa dei materiali e degli attuatori impiegati per la costruzione degli automi, mentre la capacità di ragionamento è dipendente dai tipi di elaboratori che costituiscono il "cervello" dei robot, grazie ai quali le macchine possono eseguire compiti anche molto complessi. L'integrazione con l'ambiente operativo, infine, dipende dalla capacità del sistema di studiare l'ambiente in cui si trova e dalla reattività del robot ai comandi a esso inviati, ormai quasi sempre real-time.

Possiamo quindi descrivere un robot come l'unione di quattro componenti distinte:

- **Sistema meccanico**: è l'insieme degli organi di locomozione e manipolazione che permettono al robot di muoversi e interagire con gli oggetti.

- **Sistema di attuazione**: è l'insieme dei componenti che si occupano di controllare il moto e animare le componenti meccaniche del robot.

- **Sistema sensoriale**: è l'insieme dei sensori propriocettivi ed esterocettivi che consentono rispettivamente al robot di monitorare il proprio stato interno e lo stato dell'ambiente esterno.

- **Sistema di governo**: è l'organo che si occupa di controllare il corretto svolgimento dei compiti assegnati, nel rispetto di quanto pianificato in precedenza e dei limiti derivanti dalla struttura della macchina e dall'ambiente in cui si trova. Affinché un robot sia capace di autocontrollarsi vengono impiegati meccanismi di retroazione, grazie ai quali la macchina è in grado di reagire agli input esterni.

Il sistema di governo, detto anche unità di controllo, per essere progettato correttamente e possedere un'organizzazione flessibile deve prevedere un'architettura ripartita su più livelli gerarchici, in modo tale da separare le attività elementari dalle attività di alto livello. Seguendo un approccio top-down, troviamo in primo luogo la scomposizione dell'azione da eseguire in compiti astratti, mentre in ultimo vi sono i driver dei sensori e degli attuatori che permettono d'interpretare i segnali e comandare i dispositivi. Per programmare un robot in modo ottimale, si possono quindi seguire tre strade differenti:

- **object-oriented**, attraverso un linguaggio orientato agli oggetti;

- **robot-oriented**, attraverso un linguaggio di programmazione di livello più alto rispetto al precedente, con routine e strutture dati complesse;

- **teaching-by-showing**, attraverso l'apprendimento della sequenza delle posizioni guadagnate tramite la guida dei sensori, che poi il robot sarà in grado di ripetere in autonomia.

Tutti e tre questi metodi di programmazione rappresentano valide alternative per realizzare un robot che impieghi le attuali tecnologie: lo sviluppatore utilizzerà uno o l'altro di questi sistemi in base allo specifico contesto di utilizzo del proprio robot.

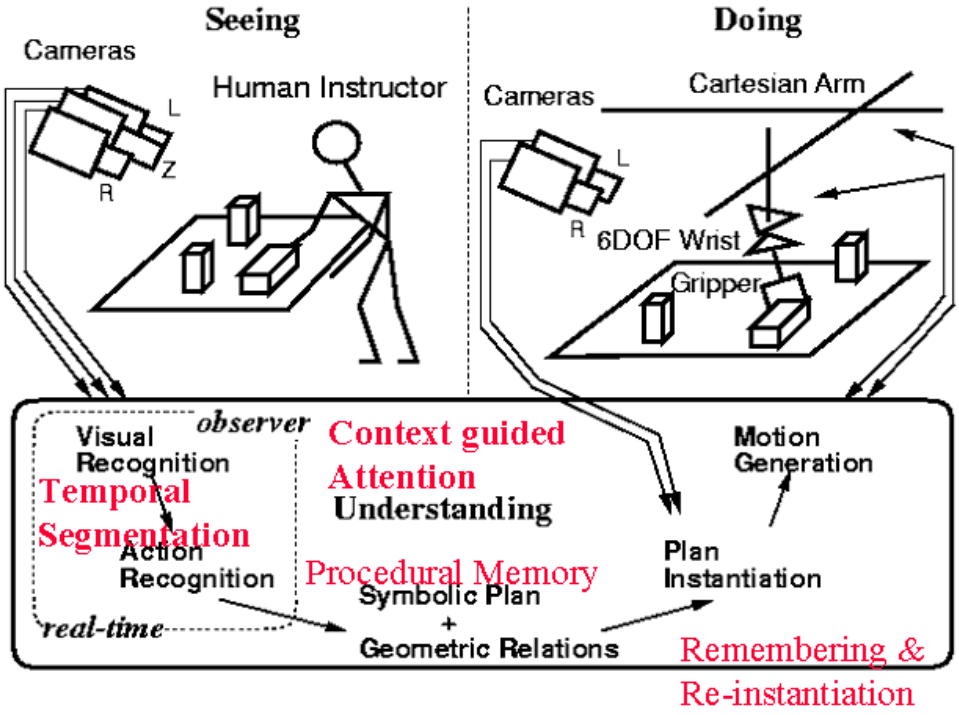

Fig. 3 – Immagine esplicativa del processo teaching-by-showing

Dopo aver approfondito la nostra conoscenza di IoT, dei sistemi Embedded e della robotica, ci siamo concentrati sull'elaborazione del nostro progetto. L'idea di base del nostro prototipo, gli obiettivi che ci siamo prefissi e le caratteristiche del sistema ideato saranno presentati in dettaglio nel prossimo capitolo.

Capitolo 2 -
Il progetto InspectorPi

2.1 L'idea del progetto

In fase di elaborazione di questa tesi abbiamo deciso di concentrarci sulla progettazione di un device che potesse avere un'utilità pratica. Abbiamo ritenuto infatti opportuno lavorare a un progetto che non avesse validità prettamente teorica, ma che fosse al contrario in grado di produrre un'innovazione tangibile. Per questa ragione abbiamo sviluppato un dispositivo impiegabile in diversi ambiti, capace di rispondere a esigenze di diversa natura.

In particolare, il nostro obiettivo è stato quello di realizzare un device che potesse svolgere funzioni di monitoraggio e controllo. Secondo la nostra idea, il dispositivo doveva essere in grado d'ispezionare ambienti difficilmente accessibili all'uomo, in cui vi sono difficoltà fisiche o ambientali cui sovvenire.

In fase di realizzazione progettuale, abbiamo quindi dovuto stabilire che tipo di device impiegare: una delle prime scelte che abbiamo compiuto è stata quella di utilizzare un veicolo di terra. Durante la fase di documentazione abbiamo infatti verificato che i progetti professionali riguardanti droni capaci di volare si stanno espandendo capillarmente, per cui, se avessimo deciso di sviluppare un progetto in ambito aero, avremmo incontrato maggiori difficoltà nel creare innovazione. Ci saremmo dovuti confrontare con i numerosi progetti esistenti, come per esempio il drone realizzato dalla startup milanese Project EMS, che ispeziona i tetti in volo per stabilire se nelle abitazioni sia presente l'amianto[10].

Escluso il cielo si è dunque pensato all'acqua, studiando le difficoltà che può incontrare l'uomo nel monitoraggio dell'ambiente marino. Anche in questo campo, però, a oggi si stanno sviluppando numerosi progetti, che si concentrano sia sull'osservazione dei fondali che sull'ispezione degli strati più superficiali degli specchi d'acqua. Uno dei progetti che più ambiziosi è PiShip, un'imbarcazione senza pilota progettata dalla Pitom (un'azienda di ex ricercatori dell'Università di Pisa) che

[10] http://www.projectems.it/ispezione-tetti/

permette di eseguire missioni di monitoraggio in laghi, fiumi e mari tramite l'utilizzo di diversi sensori[11].

Per queste ragioni abbiamo quindi stabilito di impiegare un veicolo a ruote, in grado di percorrere medie distanze via terra. Il veicolo sarà telecomandato tramite dispositivo mobile e, grazie alla presenza di telecamera e sensori, sarà in grado d'ispezionare ambienti ostili all'uomo.

2.2 Ambiti di utilizzo di InspectorPi

Nello sviluppo del nostro progetto abbiamo mirato sin dall'inizio alla realizzazione di un prodotto modulare, che, con modifiche minime, potesse essere utilizzato in diversi ambiti per il controllo e il monitoraggio.

Come già accennato in precedenza, nel corso della nostra ricerca abbiamo preso in considerazione alcuni possibili esempi d'impiego coerenti con le potenzialità del nostro device, InspectorPi. Le funzioni analizzate sono state in particolare le seguenti:

- monitoraggio di fughe di gas pesanti (come propano, butano e GPL);
- ispezione dei condotti di areazione;
- utilizzo ricreativo.

Per quanto riguarda i primi due casi, il veicolo viene sfruttato per adempiere a una necessità: a oggi esistono solo pochi semplici prototipi in grado di svolgere funzioni di ispezione e monitoraggio via terra e, in nessuno dei casi analizzati in fase di ricerca, si tratta di progetti che prevedono modularità d'uso. Nel terzo caso, invece, il device non viene adoperato per rispondere a una esigenza, ma a scopo ludico. Se inizialmente questo tipo d'impiego del veicolo non era stato considerato, in un secondo momento si è considerato che le caratteristiche del prodotto ne avrebbero in effetti consentito anche un utilizzo ricreativo da parte dell'utente.

[11] http://pitom.eu/index.php/it/prodotti/piship-asv

2.2.1 Monitoraggio di fughe di gas pesanti

Nel corso della nostra vita quotidiana siamo continuamente esposti a differenti tipi di gas che, se sono in bassa concentrazione o tenuti sotto controllo, non costituiscono un pericolo. Un'improvvisa concentrazione di gas, causata magari da una perdita, può però causare un'esplosione; altri gas, come il monossido di carbonio, possono invece essere letali anche senza esplosioni. Le perdite che possono crearsi a causa di usura o danneggiamenti generano quindi una condizione di pericolo, che va scongiurata verificando il punto di perdita. Questo procedimento, se la perdita è consistente, può essere rischioso e deve essere effettuato con le dovute precauzioni. Il caso più pericoloso è quello in cui la perdita avvenga in un locale ubicato nel sottosuolo e si tratti di un gas più pesante dell'aria, difficile da rimuovere tempestivamente. Per ovviare al problema, è possibile introdurre nell'area a rischio un dispositivo telecomandato, capace di localizzare la perdita in sicurezza, senza mettere a repentaglio vite umane: per questa ragione, creare un device che possa svolgere i controlli sul campo al posto di una persona ci è sembrata un'idea valida e spendibile.

Anche in questo caso sarà tuttavia necessario prendere le giuste precauzioni, poiché, introducendo un veicolo elettrico in una zona contenente alte percentuali di gas, si può andare incontro a problematiche di vario genere. Il veicolo dovrà quindi essere progettato in modo tale da minimizzare i rischi e i suoi componenti dovranno essere strutturati come segue:

- Qualsiasi motore elettrico non deve contenere lamelle o spazzole, ma essere puramente induttivo.

- Nel caso in cui vi siano relè, questi devono essere già schermati all'uscita dalla casa di produzione. Qualsiasi schermatura eseguita in un secondo momento su questo particolare componente, che può creare scintille al momento del contatto, potrebbe risultare inefficace.

- È consigliato creare un'armatura isolante nella parte elettrica ed elettronica del dispositivo per ottenere sicurezza aggiuntiva.

Un'ultima considerazione da fare riguarda la tipologia di gas. Ogni gas ha un proprio peso molecolare, in base al quale i gas si suddividono in tre categorie: quelli più leggeri dell'aria che tendono a salire, quelli più pesanti dell'aria che tendono a

scendere e quelli che hanno un peso simile all'aria che stazionano insieme a essa. Di seguito la tabella mostra alcuni esempi di gas con il loro peso molecolare.

TIPO GAS	*PESO MOLECOLARE* *(u.m.a.)*
Ammoniaca NH3	17
Monossido di Carbonio CO	28
Aria	29
Diossido di Azoto N02	30
Idrogeno Solforato H2S	34
Propano	44
Etanolo	46
N-Butano	58
Acetone	58
Alcool Isopropilco	60
N-Pentano	72
Benzene	78
N-Esano	86
Etil Acetato	88
Toluene	92
N-Ettano	100
N-Ottano	114
Benzina Verde	114

Tab. 2 – Peso molecolare di alcuni gas[12]

Considerando che il dispositivo che vogliamo costruire sarà dotato di ruote e si moverà via terra, il device, dotato degli opportuni sensori, potrà monitorare in modo ottimale i gas pesanti. Sarà inoltre in grado di rilevare i gas stazionari, benché in modo meno preciso.

2.2.2 Ispezione dei condotti di areazione

Numerose ricerche hanno confermato l'esistenza di una relazione tra la qualità dell'aria negli edifici e la salute delle persone che li occupano. La "Sick Building Syndrome", o "Sindrome da Edificio Malato", si manifesta all'interno degli edifici nei quali una percentuale considerevole di occupanti (oltre il 20%) soffre di una serie di sintomi come irritazione agli occhi, al naso, alla gola, alla pelle o d'ipersensività non

[12] http://www.ntaimpianti.it/impianti-rilevazione-gas-bologna.html

specificata, con simultaneità di malattie infettive. Nella maggior parte dei casi i soggetti colpiti riportano una remissione dei sintomi quando si allontano dall'edificio[13].

Per scongiurare una scarsa efficienza dell'impianto di areazione degli edifici ed evitare quindi rischi per la salute, i circuiti aeraulici devono essere puliti e bonificati in ogni parte integrata. Affinché ciò avvenga, una soluzione possibile potrebbe essere quella di eseguire un'ispezione periodica, al fine di verificare lo stato di pulizia dell'impianto, facilitandone così la manutenzione. Il monitoraggio dei condotti di areazione, inoltre, può prevedere il controllo di temperatura e umidità: grazie a esso si può stabilire dove le percentuali di umidità sono incoerenti con lo standard ambientale e, se le temperature sono molto diverse da quelle impostate, fornire un ulteriore supporto alla prevenzione della "Sindrome da Edificio Malato".

L'ispezione dei canali di ventilazione è un altro dei compiti che possono essere eseguiti dal device che abbiamo progettato, InspectorPi. In caso di necessità, infatti, il dispositivo potrà essere introdotto all'interno del canale di ventilazione: percorrerà il condotto di aerazione in tutta la sua lunghezza, creando un feedback istantaneo sullo stato dell'impianto; a seconda dei sensori utilizzati, potrà fornire informazioni su pulizia, temperatura e umidità dell'aria presente nei canali.

Affinché l'utente possa introdurre il dispositivo d'ispezione all'interno dei condotti, tipicamente costituiti da leghe metalliche, il veicolo dovrà però possedere una serie di caratteristiche specifiche, come connessione via cavo e dimensioni contenute, che saranno elencate in dettaglio nel paragrafo 2.3.

2.2.3 Utilizzo ricreativo

La società francese Parrot, che dal 1994 sviluppa apparecchi Bluetooth per la casa e l'automobile, negli ultimi anni si è dedicata alla produzione di droni. Il successo della compagnia in questo campo è stato fulmineo: se nel 2014 la vendita di droni capaci di volare costituiva il 16% del fatturato dell'azienda, nel primo trimestre del 2015 rappresentava ben il 49% del totale. La crescita maggiore è avvenuta nella vendita di droni a scopo ludico: la società ha quindi diretto tutti gli sforzi in questo settore, introducendo nel mercato anche i propri mini-droni su acqua e terra[14].

Dato l'esempio appena citato e la costante espansione del mercato dei droni, riteniamo che anche il veicolo realizzato attraverso il nostro progetto di tesi potrebbe essere impiegato dagli utenti con fini ricreativi. Rispetto ad altri device attualmente in

[13] www.nhs.uk/conditions/sick-building-syndrome/Pages/Introduction.asxp
[14] http://www.parrot.com/it/prodotti/minidrones/

vendita, InspectorPi permette di utilizzare diversi sensori e, con piccole modifiche, può svolgere funzioni più complesse rispetto al semplice spostamento o alla perlustrazione di un'area attraverso la telecamera del dispositivo. Al device, per esempio, possono essere aggiunti un microfono e un altoparlante per poter comunicare a distanza con altre persone. Modulato in modo opportuno, il dispositivo potrebbe quindi fornire all'utilizzatore un intrattenimento multiforme.

2.3 Funzionalità generali del sistema

Quali sono le funzionalità che il sistema dovrebbe integrare? Per rispondere a questa domanda, abbiamo individuato i caratteri che il nostro prototipo avrebbe dovuto possedere per essere utilizzato negli ambienti designati e il tipo di informazioni utili da conservare anche dopo la verifica in real-time. Siamo così giunti a definire i seguenti punti:

- **Capacità di muoversi nell'ambiente**: per poter monitorare un'area e individuare eventuali problematiche è necessario che il device abbia la capacità di muoversi nell'ambiente. Questa capacità rende il dispositivo adatto anche a un utilizzo ricreativo.

- **Totale controllo da parte dell'utente tramite dispositivo mobile**: dato che i problemi possono essere molteplici, all'utente verrà data la possibilità di comandare il device in prima persona e di avere il totale controllo su di esso; il veicolo non sarà quindi in grado di verificare autonomamente le condizioni dei luoghi nei quali viene introdotto. Questo sarà possibile grazie all'ausilio di un dispositivo, come un telecomando, che permetterà di comandare a distanza il veicolo.

- **Trasmissione attraverso una telecamera**: date le due caratteristiche precedenti, il dispositivo necessita di una telecamera per poter essere teleguidato. Un rapporto visivo dell'ambiente fornisce inoltre all'utente un quadro dettagliato sia del luogo che sta ispezionando che dell'entità dei problemi eventualmente riscontrati. Nel caso in cui il dispositivo venga impiegato a scopo ludico, poi, "vedere con gli occhi del veicolo" è sicuramente importante per l'utilizzatore. Negli ultimi anni gli utenti sono infatti sempre più

esigenti nei confronti dei device acquistati e l'integrazione di una videocamera è diventata un requisito quasi essenziale nei dispositivi elettronici; l'espansione del mercato dei droni per uso personale, in molti casi muniti di supporto video, ha dato un'ulteriore spinta in questo senso.

- **Dimensioni ridotte**: affinché il veicolo possa muoversi in spazi ristretti, come i tubi di ventilazione, dovrà essere di piccole dimensioni. Dato che si tratta di un prototipo, il veicolo che realizzeremo nel corso del progetto di tesi avrà dimensioni maggiori rispetto a quelle che potrebbe avere il dispositivo qualora venisse commercializzato in futuro.

- **Sensori ambientali intercambiabili**: i sensori sono essenziali per l'ispezione e monitoraggio, in quanto permettono ai device di percepire l'ambiente: non si possono monitorare temperatura, umidità o presenza di gas senza sensori adeguati; alcuni esempi di sensori integrabili saranno riportati nel paragrafo 2.4. Dato che andremo a creare un dispositivo che potrà essere utilizzato per sopperire a esigenze diverse, è necessario che il device preveda modularità, in modo tale che l'utente possa di volta in volta inserire i sensori più adatti ai propri scopi.

- **Possibilità della trasmissione dei sensori in Cloud**: dato che le informazioni trasmesse dai sensori possono esserci utili, oltre che real-time, anche in un secondo momento, può essere vantaggioso immagazzinarle all'interno di una piattaforma Cloud. In questo modo l'utente avrà a disposizione in ogni momento tutti i dati di rilevamento e potrà verificare a posteriori i risultati ottenuti durante l'ispezione. Si può inoltre pensare che, grazie alle piattaforme Cloud disponibili online, i dati trasmessi dai sensori durante l'attività di monitoraggio possano essere visibili anche a un utente esterno.

- **Connessione wireless e connessione via cavo**: qualora il dispositivo venga introdotto all'interno di spazi chiusi, sarà circondato da diversi tipi di materiali e, probabilmente, non avrà la possibilità di ricevere segnali wireless: in queste circostanze sarà quindi opportuno che venga guidato via cavo. Quando invece verrà utilizzato in spazi più ampi, sarà conveniente comandare il veicolo tramite una connessione senza fili, poiché la connessione via cavo potrebbe limitarne movimenti e mobilità. Per queste ragioni abbiamo deciso di fornire il veicolo di entrambe le dotazioni, connessione via cavo e connessione wireless**.**

Tutte e sette le caratteristiche individuate dovranno essere integrate nel dispositivo, così che il veicolo possa essere impiegato in modo ottimale negli ambiti di utilizzo designati.

2.3.1 Principali sottosistemi

Il sistema che abbiamo appena introdotto è costituito da un veicolo dotato di sensori capace di muoversi nell'ambiente, comandato da un dispositivo mobile. I valori acquisiti dai sensori sono inoltre trasmessi in Cloud dal device, in modo tale da poter essere analizzati dall'utente in un secondo momento.

Il sistema contiene quindi tre componenti:

- Il **device InspectorPi**, comandato dal dispositivo mobile, che si muove nell'ambiente e trasmette i rilevamenti dei sensori al servizio Cloud.

- Il **dispositivo mobile**, che comanda il device e preleva i dati dei sensori dal servizio Cloud.

- La **piattaforma Cloud**, che riceve i valori dei sensori dal device e li trasmette al dispositivo mobile.

Date le informazioni base in merito alla struttura del progetto, possiamo presentare un'immagine rappresentativa della struttura del sistema.

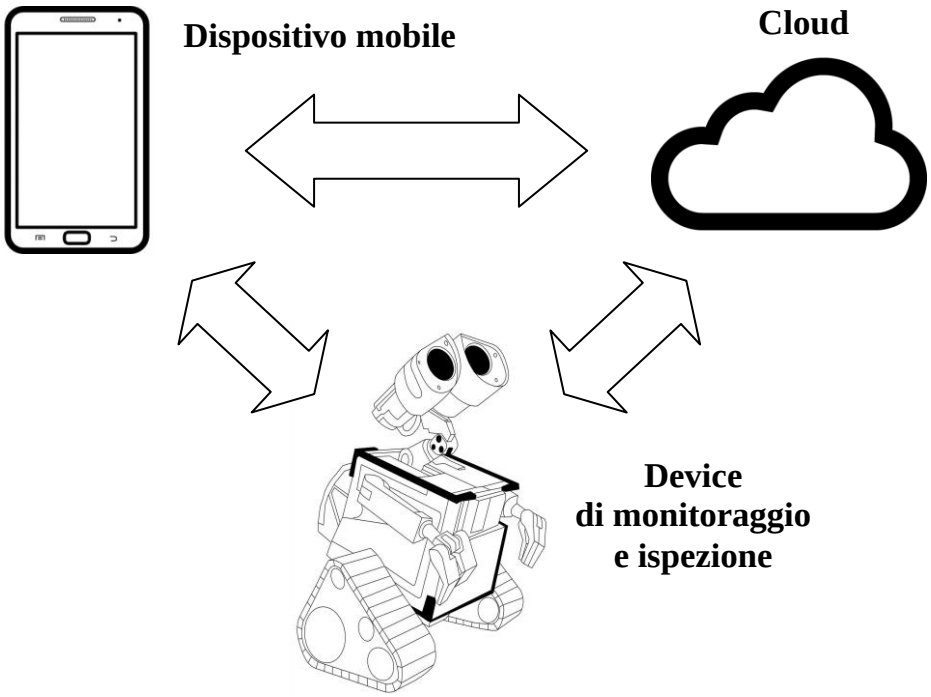

Fig. 4 – Principali sottosistemi

2.4 Sensori integrabili

Come già accennato in precedenza, per permettere un uso mirato del dispositivo possiamo affiancare al veicolo diversi dispositivi di controllo, differenti in base al campo di utilizzo prescelto. Di seguito riportiamo dunque alcuni esempi di sensori che potrebbero essere ingrati in InspectorPi per implementarne le funzionalità.

2.4.1 DHT11 e DHT22: sensori di temperatura e umidità

DHT11 è un sensore di base dell'Adafruit. Utilizza un sensore capacitivo di umidità e un termistore per misurare l'aria circostante, restituendo i valori sul pin dati (non ha bisogno di segnali analogici in ingresso). È abbastanza semplice da utilizzare, l'unica accortezza deve essere adottata nei tempi di campionamento. Questo sensore digitale, infatti, ha bisogno di due secondi per eseguire una nuova misura, quindi la misura effettuata sarà "vecchia" di due secondi.

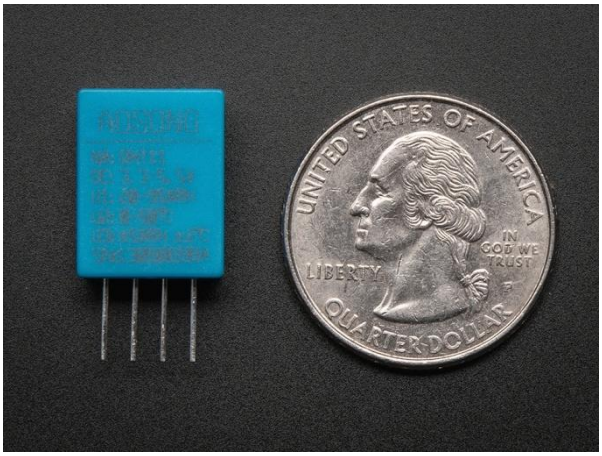

Fig. 5 – Sensore di temperatura e umidità DHT11

Il sensore necessita di una tensione dai 3v ai 5v e una corrente massima di 2.5 mA; essendo economico non è molto preciso, ma può servire per effettuare delle prove. Lavora bene dal 20 all'80% di umidità con una sensibilità del 5%, mentre, per quanto riguarda la temperatura, il funzionamento va dagli 0 ai 50°C, con una sensibilità di 2°C.

DHT11 potrà quindi essere utile per effettuare test sul nostro device, che però, per effettuare misure professionali, avrà bisogno di un sensore più preciso. Un esempio potrebbe essere il sensore digitale DHT22, anch'esso prodotto dall'Adafruit, che misura temperatura e umidità in modo più accurato del precedente. Anche in questo caso si tratta di un sensore digitale che ha la limitazione di utilizzare valori campionati nei 2 secondi precedenti. La differenza sta nel range di lavoro e sensibilità: per quanto riguarda l'umidità è utilizzabile dallo 0 al 100% e ha una sensibilità del 2-5%; inoltre lavora a un range di temperatura che va dai -40 agli 80°C, con un errore di 0,5°C. La tensione di utilizzo e la corrente massima sono identiche a quelle di DH11.

2.4.2 MQ-2 e MQ-5: rilevatori di gas

Il sensori di gas analogici della famiglia MQ utilizzano un piccolo riscaldatore interno e un sensore elettro-chimico. Il sensore MQ-2 può rilevare metano, butano, GPL e fumo, mentre MQ-5 può rilevare GPL, metano, idrogeno, alcool e monossido di carbonio. Entrambi hanno un range che varia a seconda del tipo di gas, ma sempre in ppm (point per million); l'uscita che viene fornita dai sensori è resistiva, quindi per poterli utilizzare è necessario un circuito regolatore. Le opzioni per la regolazione possono essere duplici in base alla piattaforma in cui andremo a inserire il sensore: nel caso di MCU come Arduino è possibile creare un range di tensione di lavoro e inserire

il circuito creato in un'entrata analogica. Nel caso di una piattaforma come Raspberry Pi, invece, si possono valutare due possibilità per rendere l'uscita digitale: o si integra nel circuito regolatore del sensore un ADC (convertitore analogico-digitale) in modo da acquisire digitalmente il valore specifico, oppure si può decidere di ottenere soltanto un'informazione di positività o meno che il sensore abbia superato una certa soglia di ppm, inserendo un trimmer e regolando la soglia desiderata. In quest'ultimo caso esistono dei moduli (FC-22) che eseguono questa procedura con un trimmer regolabile.

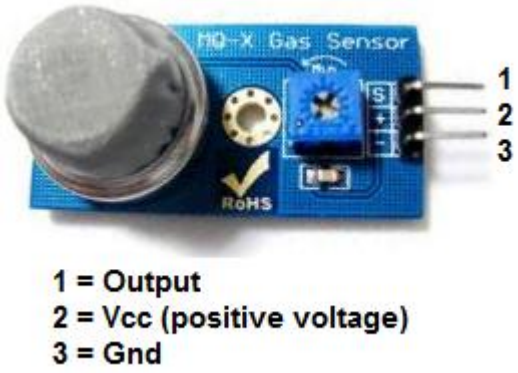

Fig. 6 – Modulo con sensore di gas

Purtroppo questo sensore per essere preciso ha bisogno di un pre-riscaldamento di 24 ore, quindi, per un corretto utilizzo all'interno del nostro device, bisognerebbe creare un circuito separato, così da poter riscaldare il sensore in anticipo senza dover necessariamente alimentare tutto il dispositivo.

2.4.3 HC-SR04: sensore di distanza

Il sensore HC-SR04 è un sensore a ultrasuoni, che misura il tempo impiegato dalle onde sonore emesse da una sorgente a tornare indietro dopo aver incontrato un ostacolo: la durata dell'impulso di ritorno determinerà la distanza dall'oggetto. Ha una tensione di utilizzo di 5v e una corrente di 15mA. Lavora a 40KHz e effettua misure alla distanza massima di 4 metri.

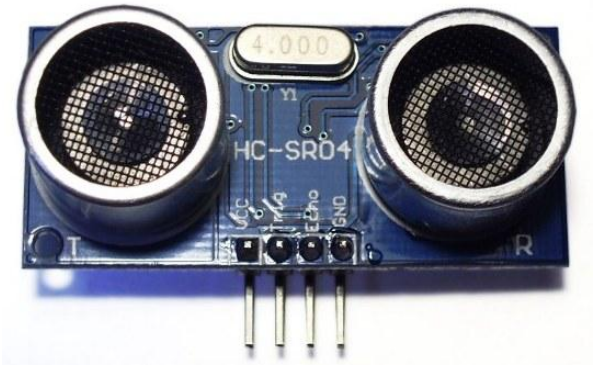

Fig. 7 – Sensore di distanza HC-SR04

L’impiego di questo sensore, anche all’interno del nostro progetto, può dunque essere utile per stabilire la profondità di un’area osservata, in alcuni ambienti difficilmente distinguibile con il solo ausilio di una telecamera.

Dopo aver delineato le caratteristiche generali del sistema, nel prossimo capitolo presenteremo l’analisi dei requisiti necessari per la corretta progettazione del software che consentirà il funzionamento del nostro device, proponendone la visualizzazione attraverso alcuni schemi UML (Unified Modeling Language).

Capitolo 3 -
Analisi dei requisiti

Dopo aver presentato il progetto di tesi, definendone i campi di utilizzo, le caratteristiche principali e i sensori associabili, in questo capitolo effettueremo un'analisi approfondita del sistema, mostrando attraverso alcuni schemi il modo in cui dovrà avvenire l'interazione tra le diverse parti.

3.1 Casi d'uso del sistema

Per poter effettuare una corretta analisi e successivamente modellare e progettare il software, è innanzitutto molto importante individuare ciò che può fare chi interagisce con il sistema. Basandoci sulle caratteristiche principali del prototipo, presentate all'interno del capitolo 2, in questo paragrafo cercheremo quindi di riassumere in un unico, sintetico schema le possibili interazioni tra il sistema e gli attori. A partire da questo modello, analizzeremo poi in dettaglio il comportamento del sistema.

Lo schema che riportiamo in seguito rappresenta i possibili casi d'uso eseguibili dai due tipi di attore che abbiamo identificato: il pilota e il possibile visualizzatore esterno. In particolare, mentre il pilota può guidare il sistema attraverso diversi comandi, il visualizzatore esterno si limita a osservare i valori attuali dei sensori.

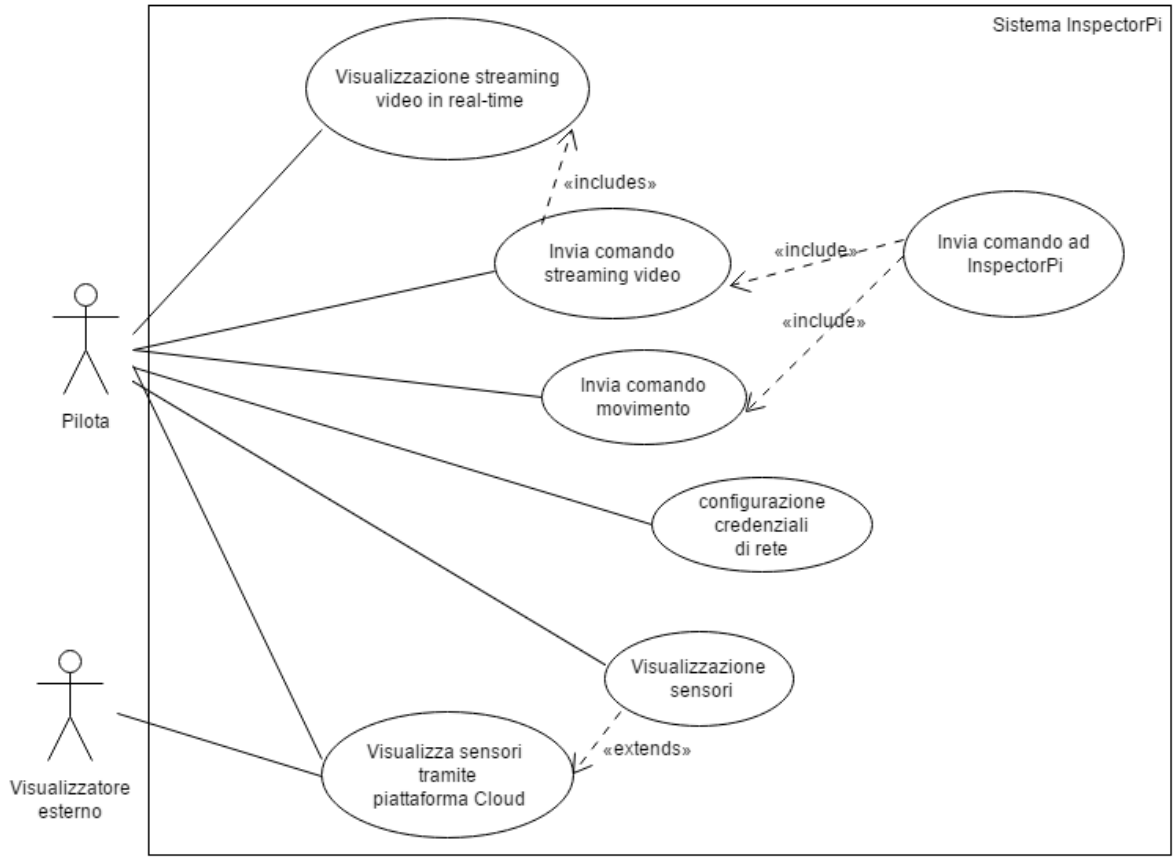

Fig. 8 – Casi d'uso del sistema

Come si evince in figura, saranno in larga parte i comandi dati dal pilota a definire la complessità del sistema. Il sistema dovrà quindi riuscire a rispondere nel modo più reattivo possibile ai comandi che gli saranno inviati.

3.2 Analisi delle interazioni

Mediante i diagrammi delle sequenze possiamo ora analizzare le interazioni nel tempo dei casi d'uso specifici, in modo tale da precisare ulteriormente quali sono i requisiti che poniamo al sistema. In primo luogo andremo a esprimere l'interazione tra gli oggetti in diversi casi d'uso, mostrando alcuni possibili scenari di funzionamento del sistema, per poi prelevare alcune porzioni dello schema rappresentato nel paragrafo 4.1 e focalizzare l'attenzione sui singoli casi d'uso, specificandone la sequenza di azione.

3.2.1 Invio del comando riguardante il movimento

Quando l'utente riesce a connettersi a InspectorPi tramite dispositivo mobile, può guidarlo attraverso l'invio di un messaggio contenente il comando da eseguire. Questo vincolo dinamico può essere espresso attraverso il caso d'uso inserito di seguito in figura.

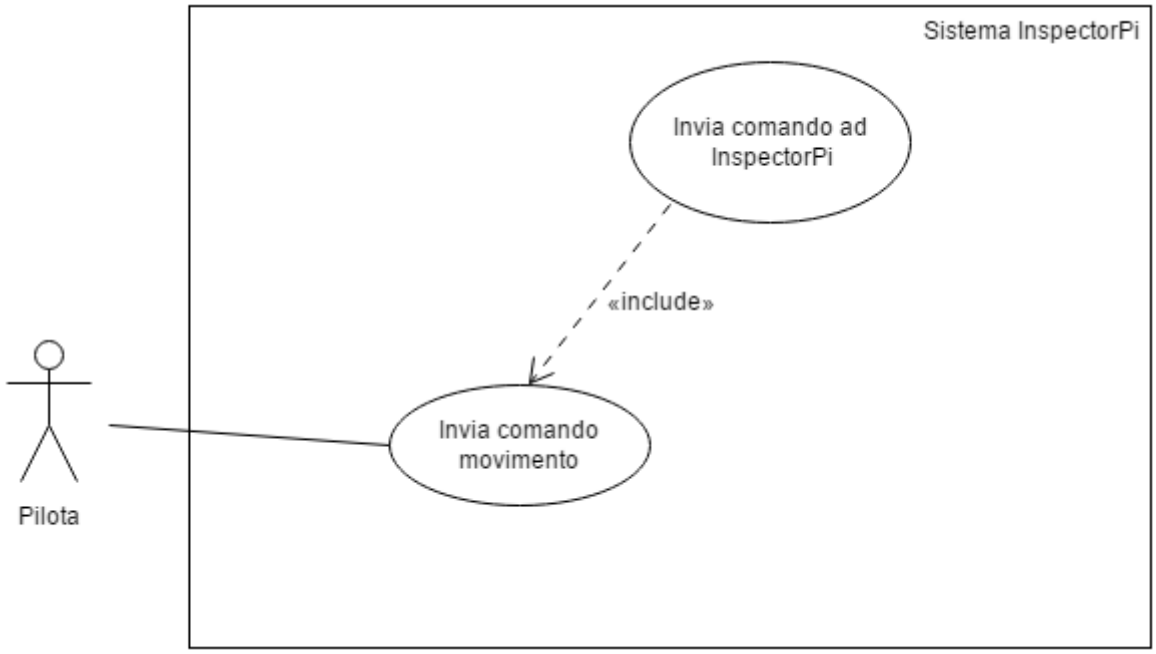

Fig. 9 – Caso d'uso riguardante il movimento di InspectorPi

Ciò che bisogna riuscire a esprimere è la sequenza delle interazioni tra gli elementi che avviene nel sistema per l'invio di uno specifico comando. Cerchiamo dunque di suddividere l'azione "Invia comando movimento" in una serie di azioni in sequenza: il dispositivo mobile, dopo la pressione del comando da parte dell'utente, invia il comando al device InspectorPi. Il dispositivo può quindi rispondere in due modi differenti:

- Nel caso il comando venga eseguito correttamente, con risposta positiva.

- Nel caso in cui il comando non possa essere eseguito, con risposta negativa, fornendo all'utente una tra le seguenti motivazioni:

- Al dispositivo è stato inviato il comando di dirigersi verso sinistra, ma identifica un ostacolo tramite sensore di prossimità: in questo caso informerà l'utente che il sensore IR sinistro ha avuto un riscontro.

- Al dispositivo è stato inviato il comando di dirigersi verso destra, ma identifica un ostacolo tramite sensore di prossimità: in questo caso informerà l'utente che il sensore IR destro ha avuto un riscontro.

- Al dispositivo è stato inviato il comando di dirigersi in avanti, ma identifica un ostacolo tramite sensore di prossimità: in questo caso informerà l'utente che il sensore IR centrale ha avuto un riscontro.

- Il dispositivo, probabilmente a causa di un errore di comunicazione, ha ricevuto un comando che non è in grado di eseguire.

Cerchiamo di esprimere queste eventualità tramite un diagramma di interazione specifico per le sequenze. Si è scelto di escludere il pilota, perché l'unico modo che ha di agire nel sistema è tramite il dispositivo mobile.

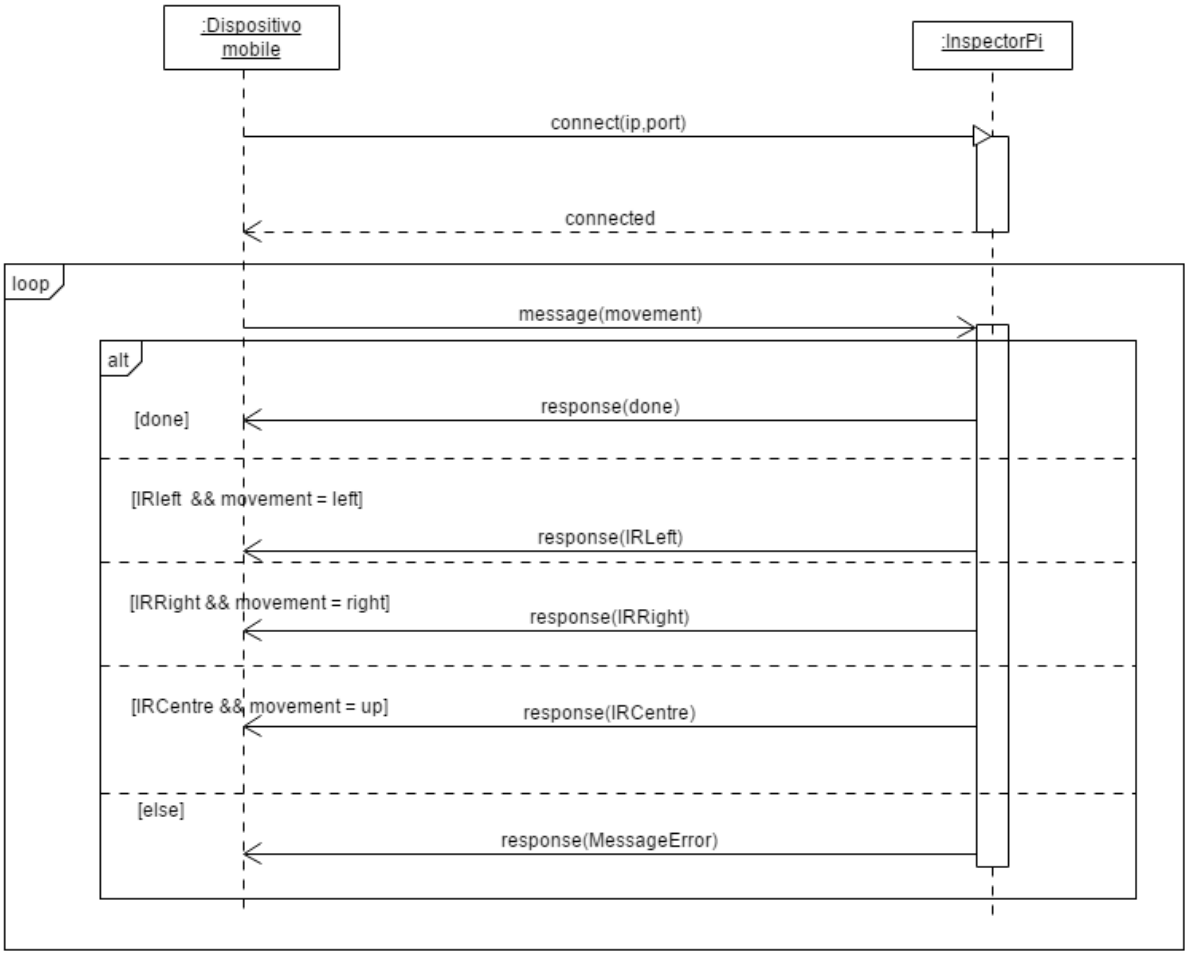

Fig. 10 – Diagramma delle sequenze riguardante il movimento di InspectorPi

Come si può constatare osservando lo schema, è stato scelto di creare una connessione persistente tra il dispositivo mobile e InspectorPi, poiché è stata data particolare importanza alla reattività della comunicazione del sistema. Per semplicità non sono stati espressi vincoli riguardanti la mancanza di connessione o il termine della stessa, così da focalizzare l'attenzione sui messaggi e le risposte tra i dispositivi.

Si noti che lo scambio dei messaggi tra il client (il dispositivo mobile) e il server (InspectorPi) avviene in modo asincrono, per permettere a entrambi i dispositivi l'invio di più messaggi prima di ottenere una risposta. Come già sottolineato, infatti, la capacità del sistema di rispondere in modo rapido è fondamentale per la nostra comunicazione. Non sarebbe infatti possibile comandare un dispositivo che reagisse ai comandi più lentamente rispetto all'utente, poiché la percezione alterata dell'utilizzatore pregiudicherebbe l'esecuzione stessa del sistema: un'automobile che rispondesse con un'eccessiva latenza ai nostri comandi, per esempio, sarebbe molto più complicata da guidare e non sarebbe reattiva in caso di imprevisti.

3.2.2 Visualizzazione dello streaming video in real-time

Dopo aver studiato le interazioni nel caso della richiesta di un comando riguardante il movimento, passiamo ad analizzare la visualizzazione in real-time dello streaming video. L'utente, tramite dispositivo mobile, riesce a visualizzare la telecamera posizionata sul device in real-time: in questo modo può monitorare l'ambiente e avere indicazioni sui movimenti del dispositivo. Per semplicità di comprensione, rappresentiamo quest'azione con la seguente figura:

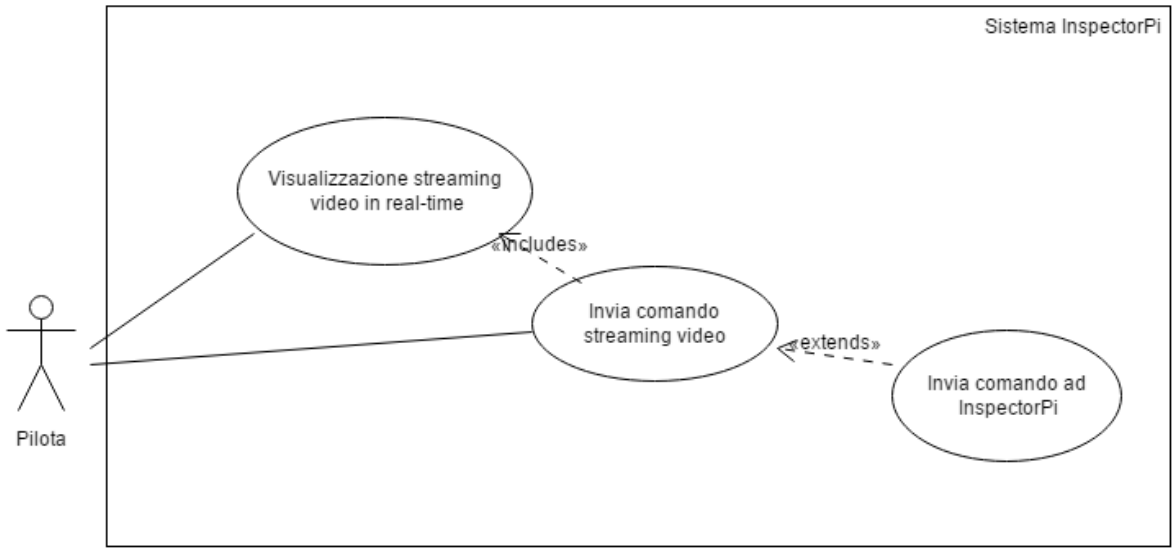

Fig. 11 – Casi d'uso riguardanti la visualizzazione dello streaming video

Come si può notare, abbiamo riportato sia il caso d'uso riguardante la visualizzazione dello streaming video in real-time che l'invio dei comandi riguardanti lo streaming. Questo perché è stato stabilito che la visualizzazione dello streaming video da parte dell'utente non sia possibile senza l'invio di un comando che ne stabilisca l'inizio. In altre parole, per quanto riguarda le interazioni all'interno del sistema, affinché il dispositivo mobile riceva lo streaming video dal device, dovrà prima far partire la comunicazione dando un comando al dispositivo. Ipotizzando che la gestione dell'invio dello streaming video sia attribuita a un server, quest'ultimo dovrà quindi essere inizializzato alla richiesta di comunicazione da parte del dispositivo. Riassumendo, i passi da eseguire sono i seguenti:

1. Invio del messaggio da parte del dispositivo mobile contenente la richiesta di inizializzare lo streaming video.

2. Creazione del canale di streaming video da parte di InspectorPi.
3. Risposta dell'avvenuta apertura del tunnel video.
4. Creazione dei client video per l'accesso allo streaming.
5. Invio della richiesta di visualizzazione dei contenuti.
6. Invio dello streaming al client del dispositivo mobile.

Possiamo esprimere questi passaggi tramite un diagramma delle sequenze.

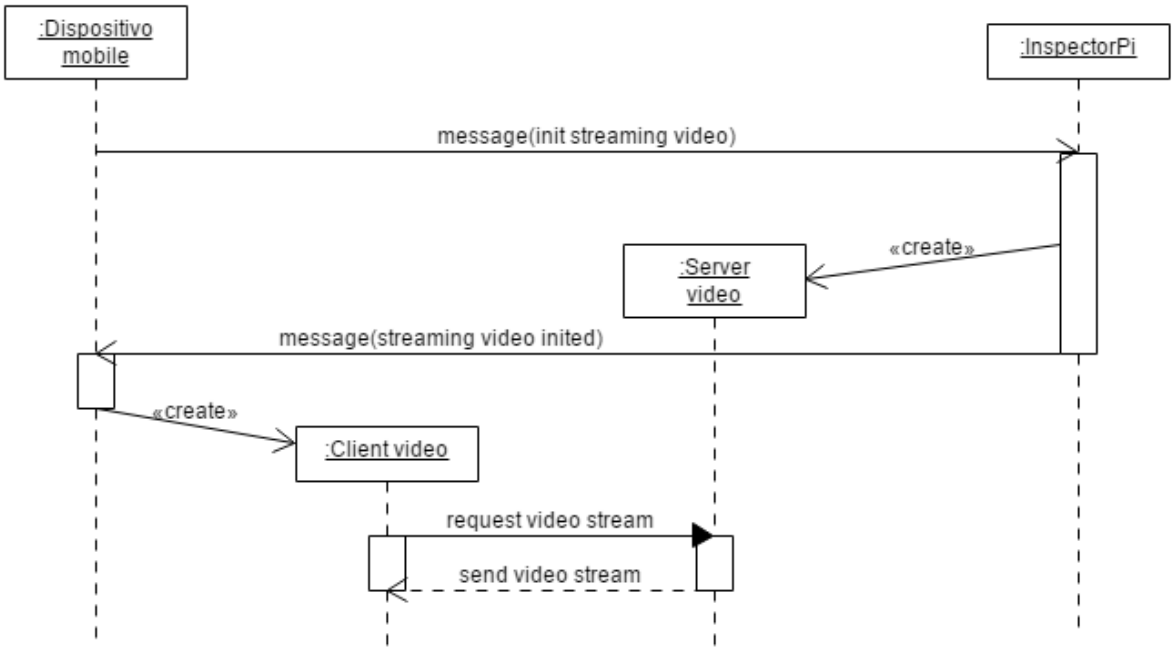

Fig. 12 – Diagramma delle sequenze riguardante la visualizzazione dello streaming video

Nel diagramma non è stato riportato il termine dello streaming real-time per concentrare l'attenzione sulla creazione e la continuità dello streaming video.

Anche se non è stata ancora definita la modalità di trasmissione dati dello streaming video, il processo che se ne occupa è stato chiamato "server", in modo tale da spiegare che la comunicazione tra i due dispositivi avviene similmente a una comunicazione Client-Server.

In questo schema, inoltre, non abbiamo inserito un ciclo per la condivisione video, perché i contenuti inviati sono sempre differenti. Nonostante questo, la comunicazione è persistente, poiché la cattura del video nel dispositivo non termina: grazie al mancato

termine di acquisizione, l'utente ha la possibilità di ricevere nuove informazioni in modo continuo. La realizzazione di questa comunicazione verrà spiegata in dettaglio attraverso la modellazione reale del comportamento specifico del prototipo.

3.2.3 Visualizzazione dei sensori tramite piattaforma Cloud

Come ultimo caso d'uso andiamo ad analizzare la visualizzazione dei sensori su piattaforma Cloud; questo è l'unico caso che, oltre all'utente, coinvolge un secondo attore. Utilizzare il Cloud come canale di trasmissione e visualizzazione dei valori dei sensori ha infatti una duplice utilità: grazie alla storicizzazione fornisce all'utente la possibilità di visualizzare i valori ottenuti durante l'ispezione in un secondo momento, dando inoltre la possibilità a un secondo utente esterno di visualizzare i valori in real-time. Grazie a questa prerogativa, si potrebbe prevedere la partecipazione all'ispezione di una seconda persona che analizzi da remoto i dati prelevati.

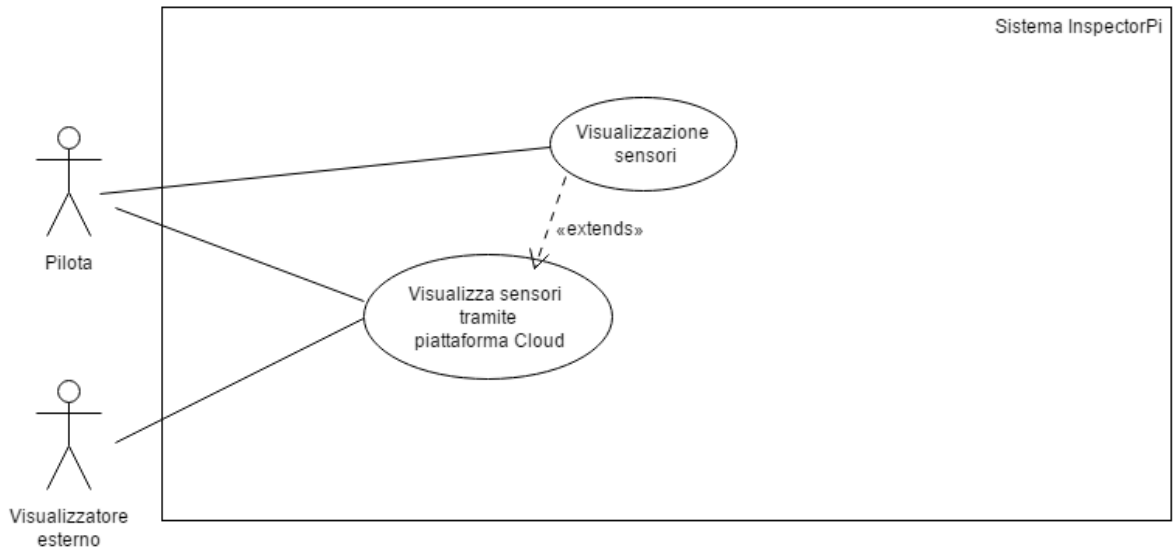

Fig. 13 – Caso d'uso della visualizzazione dei sensori tramite piattaforma Cloud.

Nel diagramma delle sequenze si è quindi scelto di inserire anche il secondo attore, che, pur essendo un partecipante opzionale, dall'istante in cui accede alla piattaforma ha la possibilità di visualizzare i sensori.

Il diagramma rappresenta dunque InspectorPi, che pubblica i sensori sulla piattaforma Cloud, il dispositivo mobile, che preleva i sensori dalla piattaforma, e l'utente esterno, che esegue lo stesso procedimento del dispositivo mobile. Come per i diagrammi precedenti, non ci siamo focalizzati sulle difficoltà della comunicazione.

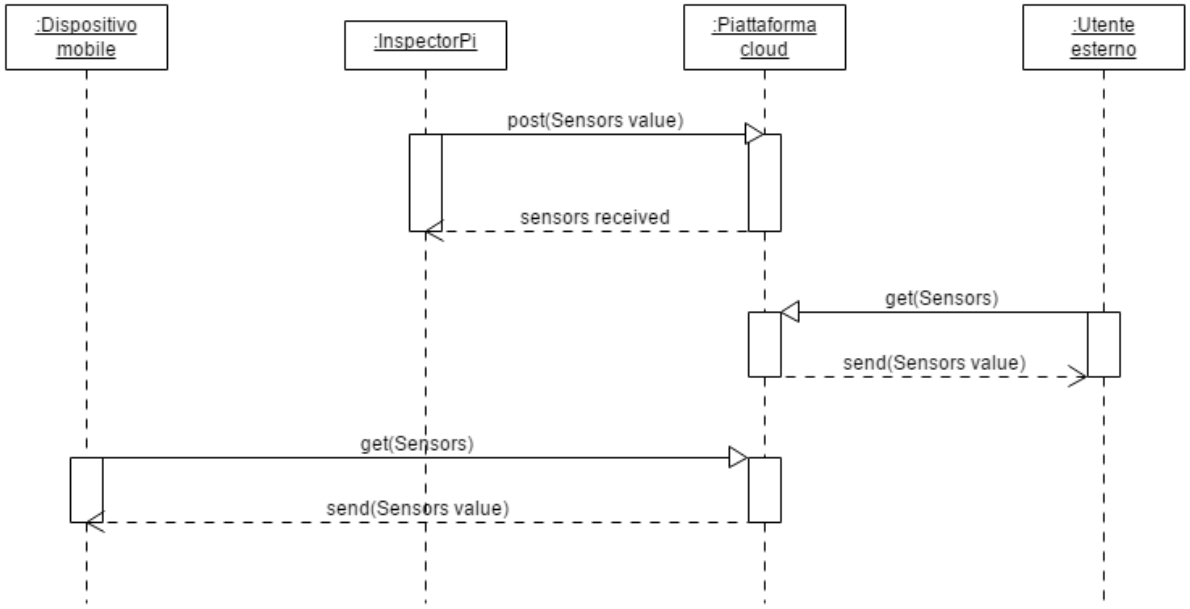

Fig. 14 – Diagramma delle sequenze riguardante la visualizzazione dei sensori

Dopo aver espresso le interazioni che avvengono all'interno del sistema sviluppando i principali casi d'uso, nel prossimo capitolo andremo a descrivere le scelte fatte in materia di hardware per la realizzazione del prototipo, presentando in modo dettagliato tutte le sue componenti.

Capitolo 4 -
Il prototipo InspectorPi

Dopo aver presentato il progetto di tesi, definendone i campi di utilizzo, le caratteristiche principali e un'analisi approfondita, in questo capitolo andremo ad analizzare la struttura del sistema, approfondendo le caratteristiche dell'hardware del prototipo InspectorPi, per poi passare all'analisi e alla progettazione del software associati al progetto di tesi.

4.1 La portabilità del progetto e il prototipo InspectorPi

In seguito alla scelta di fornire vari campi di utilizzo al dispositivo, si è deciso di rendere il progetto di tesi più portabile possibile e open-source, così che in futuro possa essere ampliato e sviluppato utilizzando più tecnologie e piattaforme. Nella prima parte di analisi si è quindi deciso di non focalizzarsi su una specifica tecnologia, per venire incontro alla possibilità di ulteriori sviluppi.

Grazie alla larga diffusione di smartphone e alle numerose piattaforme Cloud fornite in rete, anche free, si riesce a generalizzare con facilità i due componenti secondari, ovvero il dispositivo mobile e il servizio Cloud (del quale verrà fornito un esempio specifico in fase di prototipale) in modo da agevolare la portabilità.

Invece troviamo maggiore difficoltà a generalizzare il componente principale, ovvero il device di monitoraggio e ispezione, soprattutto in relazione all'hardware, che deve essere dotato delle caratteristiche specifiche elencate nel paragrafo 2.3.

A causa di questa difficoltà si è deciso di identificare la piattaforma e i componenti hardware del device del prototipo ancor prima di effettuare la progettazione del software che andremo a integrare nel prototipo. Partiremo quindi scegliendo l'hardware del prototipo, poi la telecamera e il protocollo per lo streaming real-time video e infine la sensoristica di bordo.

4.1.1 Raspberry Pi come cuore di InspectorPi

Le scelte in merito alla piattaforma e ai componenti hardware del device sono state compiute tenendo conto dell'ampia diffusione delle tecnologie da impiegare e del loro possibile utilizzo.

In primo luogo si è pensato a quale tipo di elaboratore impiegare per la realizzazione del nostro progetto. Abbiamo valutato che, per riuscire a trasmettere dati su Internet, il device, comandato da un dispositivo mobile al quale trasmette un video in real-time, ha bisogno di un elaboratore che riesca a gestire un flusso video, una connessione Internet e una comunicazione wireless con il dispositivo mobile. I dispositivi Embedded a largo consumo, analizzati nel paragrafo 1.2, sono quelli più adatti a svolgere questi compiti, perché hanno un consumo ridotto di energia pur avendo discrete capacità di elaborazione e diverse periferiche on-board.

La scelta dell'elaboratore doveva quindi necessariamente cadere su un dispositivo che, oltre ad avere larga diffusione, potesse processare un flusso video senza mettere a repentaglio la reattività del controllo da parte dell'utente: per questa motivazione tra le schede di ampia diffusione è stata esclusa Arduino, mentre sono state prese in considerazione BeagleBone (BeagleBone Black) e Raspberry Pi (Raspberry Pi 2 model B).

Le due schede, pur avendo caratteristiche simili, presentano alcune differenze, che ci hanno permesso di compiere una scelta tra le due.

- Prezzo: sensibilmente più basso quello di Raspberry Pi
- I/O: BeagleBone ha 128 GPIO pin, contro 40 GPIO di Raspberry Pi
- Community: ampiamente più supportato Raspberry Pi; esiste infatti una vasta community che offre diverse guide e spiegazioni a chi approccia per la prima volta questo single-board computer[15].
- Periferiche: Raspberry Pi ha due porte USB in più rispetto a BeagleBone e possiede un connettore per la telecamera dedicato.

Dati questi elementi, si è pensato di impiegare la scheda Raspberry Pi. Benché questa scelta vada a discapito dei pin general-purpose, grazie alla larga diffusione di

[15] https://www.raspberrypi.org/community/

Raspberry Pi e della possibilità di un'architettura multi-threading, questa scheda può essere impiegata nel nostro sistema a un basso costo, senza per questo inficiare il risultato finale. Dal nome della board è quindi derivato il nome del nostro device, InspectorPi: abbiamo unito il termine inglese che definisce la funzione del veicolo di ispezione progettato, "Inspector", ovvero "ispettore" in italiano, e la desinenza del nome dell'elaboratore utilizzato per il progetto, "Pi".

È però importante sottolineare che, essendo la BeagleBone Black una valida alternativa in futuro, quando si andranno a eseguire scelte progettuali riguardo l'architettura del software utilizzato, il sistema del device potrebbe anche essere implementato attraverso questa scheda.

Data la necessità di connettere il dispositivo alla rete e al dispositivo mobile è stato scelto di utilizzare due dongle da inserire nelle porte USB del prototipo, una per il Wi-Fi e l'altro per la connessione Bluetooth. La prima servirà per la connessione al dispositivo mentre la seconda per permetterci di dare le indicazione al dispositivo necessarie a connetterlo al Wi-Fi.

4.1.2 Pi2Go, la kit-board su cui montare Raspberry Pi

Dopo aver deciso di utilizzare Raspberry Pi come cuore del device, sono state compiute alcune considerazioni riguardo al tipo di hardware che il dispositivo avrebbe potuto comandare.

I motori elettrici che devono permettere al nostro dispositivo di muoversi nello spazio, dovranno essere comandati da Raspberry Pi per mezzo di un driver creato appositamente per motori, perché avranno bisogno di una determinata potenza per poter funzionare in modo corretto. Infatti i motori elettrici dovranno riuscire a spostare la propria struttura su un piano che, anche se molto leggero, avrà un peso. In questo senso, un'altra necessità è quella di un'alimentazione superiore.

Se Raspberry Pi consuma solo 4 Watt (dichiarati dalla casa di produzione) grazie a un'efficiente progettazione, in base al carico i motori elettrici possono arrivare a consumare più della scheda. Per questo avremo bisogno di un'alimentazione superiore rispetto a quella normalmente fornita alla scheda.

Dato che la progettazione dell'hardware esula dagli scopi di questa tesi, si è pensato di effettuare una ricerca delle board da inserire a Raspberry Pi, allo scopo di individuare quelle che avrebbero potuto venire in contro alle esigenze dei motori e dell'alimentazione.

Dopo un'approfondita ricerca, tenendo conto sia del prezzo che delle nostre esigenze, abbiamo ottenuto una breve lista di hardware facilmente integrabili con Raspberry Pi, tra i quali scegliere. Di seguito sono presentati in ordine di prezzo:

- Pi2Go: board che sfrutta due motori e un ruotino di supporto; possiede diversi sensori ed è alimentata da 6 batterie modello AA[16].

- GoPiGO: board che utilizza due motori e un ruotino di supporto per permettere il movimento; l'alimentazione viene fornita da 8 batterie modello AA[17].

- Initio: prodotto modulare che può essere utilizzato sia con Raspberry Pi che con Arduino; è dotato di quattro motori e alcuni sensori, utilizza uno sheild per Raspberry Pi per comandare i motori[18].

- DiddyBorg: piccola scheda che comanda sei motori, dotata di una corazza protettiva. Sopporta molto carico ed è alimentata da dieci batterie modello AA. Si tratta sicuramente di un'ottima scelta per il trasporto di piccoli carichi[19].

La nostra scelta è quindi ricaduta sulla prima opzione, Pi2Go, poiché questa board aveva più elementi utili delle altre e un costo minore.

Pi2Go è una kit-board distribuita da 4tronix e sviluppata in Gran Bretagna. Viene presentata come un robot completamente integrato con Raspberry Pi: grazie ai numerosi led o sensori, viene incontro all'esigenza di realizzare esperimenti su Raspberry Pi. Questa board ha riscosso abbastanza successo da permettere a "Pi2Go" di diventare un marchio registrato nel maggio del 2014. Dopo diverse revisioni, oggi la board completamente assemblata e con a bordo un Raspberry Pi è disponibile in due versioni, che presentano poche differenze, mostrate di seguito in foto.

[16] http://www.pi2go.co.uk/
[17] http://www.dexterindustries.com/shop/gopigo-kit/
[18] http://4tronix.co.uk/blog/?p=169
[19] https://www.piborg.org/diddyborg

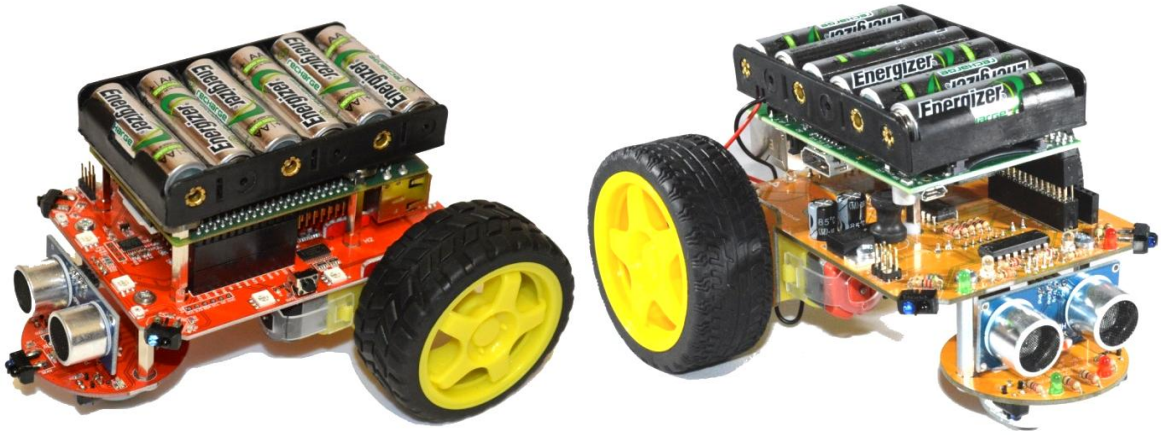

Fig. 15 – Pi2Go (a sinistra) e Pi2Go Lite (a destra)

Come si può notare dall'immagine, i due device hanno entrambi numerosi sensori e sono stati progettati appositamente per essere utilizzati unitamente alla piattaforma che abbiamo scelto in fase progettuale. Grazie al vano batterie riescono a essere autonomi, senza bisogno di cavi o fili pendenti. L'elenco delle specifiche, presentato in tabella, spiega al meglio le differenze tra le due versioni:

	Pi2Go	**Pi2Go-Lite**
Dimensione della PCB principale (mm)	110 x 80	100 x 80
Colore della PCB	Rosso	Giallo
Tecnologia dei componenti	Montaggio in superficie	Passante
Metodo di assemblaggio	Completamente saldato	Richiede saldature
Modelli supportati A, B e B+	Si	Si
Alimentazione	6 x AA	6 x AA
Regolatore a 5V	Tipologia switching a 3A	Tipologia switching a 3A
Monitoraggio della batteria	Si *	-
Sensori di luce	4 x analogici	-
LED RGB	4 accoppiati, hardware PWM	-

LED Bianchi	-	2 davanti, 2 dietro (accoppiati, s/w PWM)
Sensore a ultrasuoni	Si	Si
2 sensori di luce	Si	Si
Sensori IR di prossimità	3 con tecnologia anti-riflesso **	2
Ruote	65mm Nere	65mm Gialle
Connessione ai Servomotori	4 (Hardware PWM)	2 (Software PWM)
Connettore I2C per uscite aggiuntive	Si	Si
Indicatore LED per sensore IR	Si	Si
Rapporto di trasmissione	120:1	120:1
Connettore dei motori	Terminale a connettore	Terminale a morsetto
Connettore di alimentazione	Terminale a connettore	Terminale a morsetto
Tempo di montaggio (approssimativo/variabile)	20 minuti	90 minuti ***
Prezzo suggerito (IVA esclusa)	£50	£35

Tab. 3 – Specifiche tecniche di Pi2Go e Pi2Go Lite

Note:

** Il circuito di monitoraggio della batteria controlla continuamente che il livello carica sia sufficiente per alimentare sia il motore che il Raspberry Pi. Appena non è sufficiente per entrambi, disabilita i motori. Se la carica diminuisce ancora, manda un segnale a Raspberry Pi, in modo che possa essere spento in sicurezza. Questa funzione può essere sovrascritta cambiando il jumper della scheda.*

*** La "tecnologia anti-riflesso" previene i falsi segnali dovuti alla luce del sole. In ogni caso il sole può allarmare il sistema IR che previene i segnali ricevuti.*

**** 90 minuti sono calcolati considerando 1'ora per saldare la PCB e 30 minuti per l'assemblaggio. Il tempo delle saldature dipende dalla loro velocità e abilità, che può variare sensibilmente.*

Analizzando queste specifiche si nota una discreta differenza tra i due dispositivi per quanto riguarda la cura dei dettagli. La versione completa, infatti, si distingue da quella Lite per l'utilizzo di componenti SMD e per il monitoraggio della batteria. Per il prototipo che andremo a creare si è dunque scelto di utilizzare la versione completa, in modo da poter sfruttare queste caratteristiche del device.

Per facilitare la comprensione della PCB, sul sito del dispositivo viene fornito uno schema del circuito stampato, presentato di seguito.

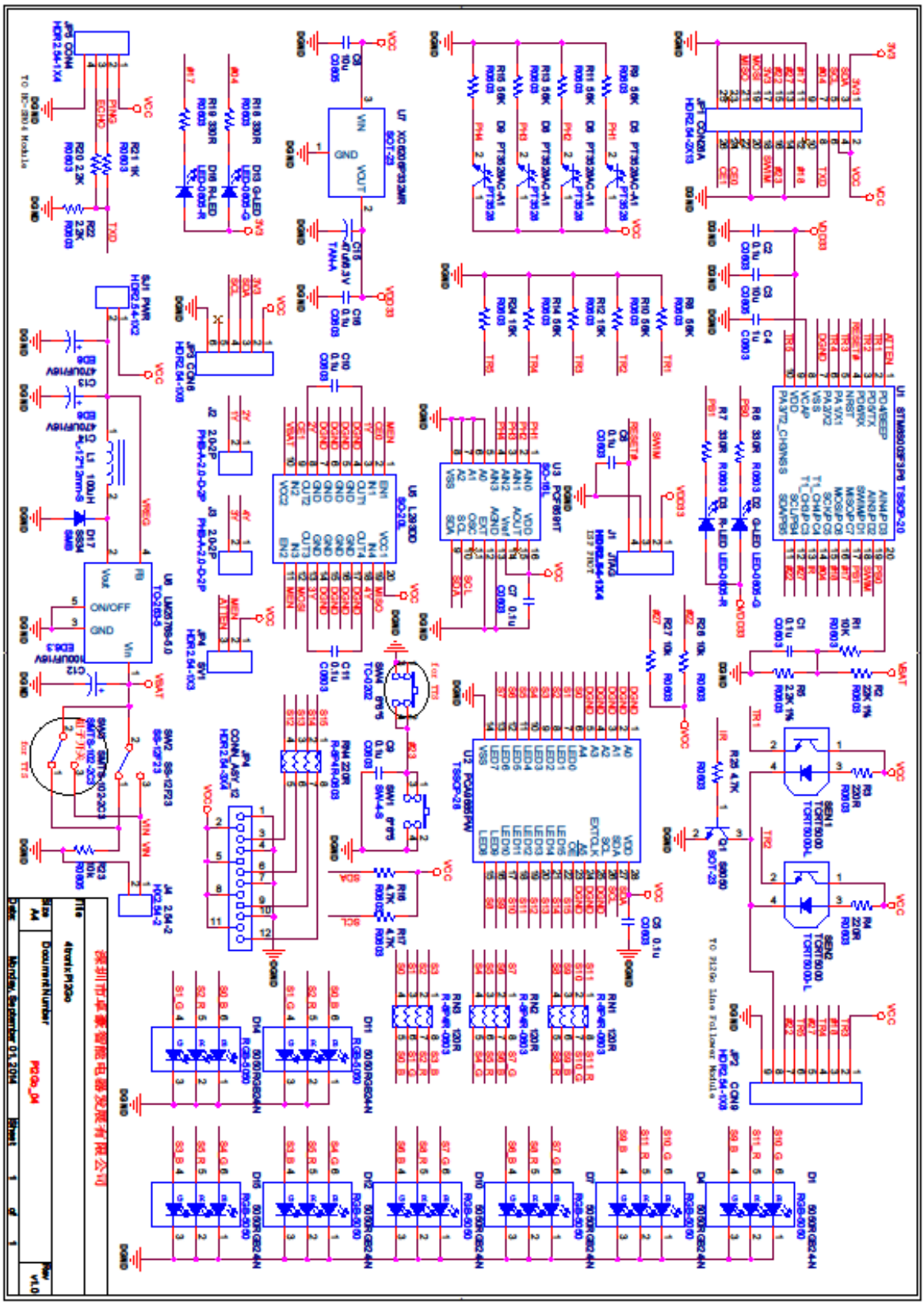

Fig. 16 – Schema del circuito stampato di Pi2Go

Come si evince dallo schema, la PCB, oltre ai sensori, monta un microcontrollore (nello schema chiamato U1), in modo da non dover totalmente dipendere da Raspberry Pi. Come vedremo anche nei prossimi capitoli, infatti, il dispositivo gestisce i sensori di prossimità e il monitoraggio dell'alimentazione senza bisogno che Raspberry Pi interagisca con esso, inviando in caso positivo un segnale per comunicare l'evento.

Dopo aver scelto la board e la piattaforma, passiamo ora a stabilire la telecamera utilizzata e il tipo di connessione dello streaming real-time video.

4.1.3 Lo streaming video real-time

Il single-board computer che abbiamo deciso di utilizzare ha un connettore dedicato per la telecamera che utilizza un'interfaccia MIPI CSI-2; quest'interfaccia seriale cerca di venire incontro all'esigenza di uno standard robusto, che può anche evolversi per rispondere alla richiesta di telecamere con sempre più definizione e potenza e una banda passante in costante aumento[20].

Sfruttando quest'interfaccia, la Raspberry Pi Foundation ha sviluppato un modulo telecamera da 5 Mpx, che ha la possibilità di scattare foto con una risoluzione di 2592x1944 pixel e video fino a 1080p a 30fps, mantenendo il costo del prodotto alla portata di tutti.

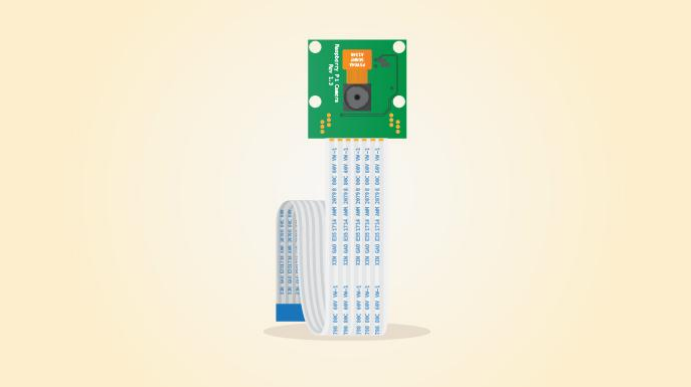

Fig. 17 – Modulo telecamera Raspberry Pi

Date queste prerogative, si è scelto di utilizzare il modulo telecamera NoIR del Raspberry Pi, che non contiene il filtro a raggi infrarossi, in modo che, nel caso in cui venga utilizzata una luce a infrarossi, si riesca a vedere l'immagine trasmessa dalla telecamera anche al buio.

[20] http://mipi.org/specifications/camera-interface

A questo punto è stata eseguita una ricerca e sono stati effettuati dei test, il cui andamento sarà riportato nel capitolo 5, in modo da stabilire il metodo migliore per trasferire in real-time il flusso della telecamera al dispositivo mobile. Le opzioni possibili erano tre:

- Flusso RTSP tramite VLC
- GStreamer
- Flusso raw h.264 tramite NetCat

Grazie ai test eseguiti in fase di collaudo è stato possibile stabilire che, nel nostro caso, il metodo migliore era tramite un flusso NetCat con una latenza media di 0.4 secondi.

4.1.4 I sensori associati e il pinout finale

Dopo aver realizzato il nostro prototipo con la scheda Raspberry Pi, il kit-board Pi2Go e il modulo telecamera NoIR, ci siamo occupati dei sensori da inserire. Per poter scegliere quali sensori da associare al nostro progetto, è necessario identificare uno dei possibili campi di utilizzo: come abbiamo visto nel paragrafo 2.4, si dovrà adottare una sensoristica differente per ogni ambito di utilizzo designato.

Dato l'interesse di testare il dispositivo sia con connessione via cavo che con connessione wireless, si è scelto di implementare il campo riguardante i condotti di areazione, dove avvieremo una connessione via Ethernet, e di testare il device a scopo ludico, guidando il veicolo per mezzo della connessione Wi-Fi 802.11.
Mentre nel secondo caso i sensori aggiuntivi sono opzionali, nel primo abbiamo scelto di integrare nel device un sensore di temperatura e umidità. Dato che il nostro device è solo un prototipo, abbiamo scelto di utilizzare il sensore economico introdotto nel paragrafo 2.4, il DHT11. Dal datasheet del sensore DHT11 siamo riusciti a estrapolare le specifiche riguardanti il componente e a capire come devono essere connessi i pin per riuscire a prelevare i dati. Nell'immagine seguente è rappresentato il pinout del dispositivo.

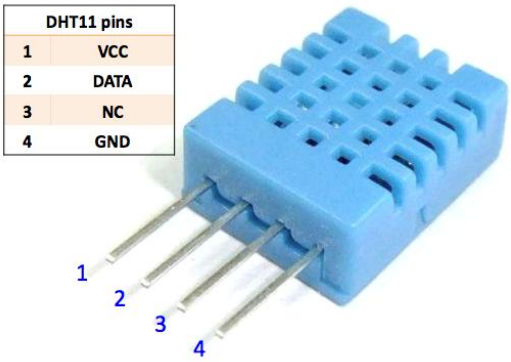

Fig. 18 – Pinout del componente DHT11

Grazie all'uscita digitale del dispositivo, il circuito che costruiremo per connettere il componente al Raspberry Pi sarà molto semplice: come suggerito nel datasheet, andremo a inserire una resistenza di pull-up da 4.7 KOhm per evitare guasti e effettueremo i restanti collegamenti con Raspberry Pi. Di seguito riportiamo un'immagine rappresentativa del circuito necessario per connettere il DHT11 al Raspberry Pi.

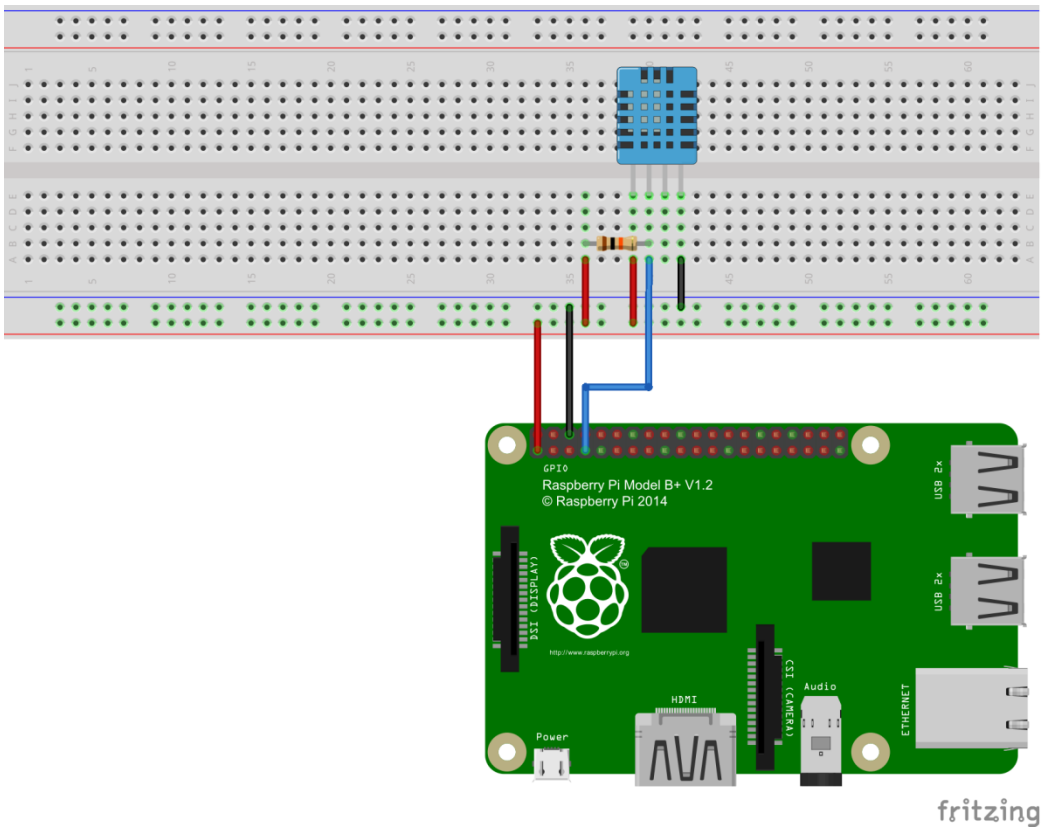

Fig. 19 – Circuito connessione DHT11 a Raspberry Pi

Il prototipo sarà composto quindi da un single-board computer Raspberry Pi cui saranno connessi diversi componenti. In tabella è rappresentato il Pinout finale di Raspberry Pi connesso al prototipo.

	InspectorPi
Function	**Pin**
Left Line Sensor	12
Right Line Sensor	15
Left IR Obstacle Sensor	11
Right IR Obstacle Sensor	7
Centre IR Obstacle Sensor	13
Front White LEDs	N/A
Rear White LEDs	N/A
Tact Switch	16
Battery Monitor	18
Ultrasonic (Ping & Echo)	8
DHT11 (Ping & Echo)	32
Left Motor	24, 26
Right Motor	19, 21

Tab. 4 – Pinout finale del prototipo

A questo punto possiamo presentare un indice completo dei componenti utilizzati nel prototipo, in modo tale da effettuare un riassunto finale:

- Raspberry Pi 2 model B
- Pi2Go
- Modulo telecamera NoIR per Raspberry Pi
- 6 batterie AA
- Dongle Wi-Fi
- Dongle Bluetooth
- DHT11

- Resistenza da 4.7 KOhm

Dato l’elenco degli elementi del prototipo, l’immagine dell’InspectorPi completo in ogni sua parte è la seguente:

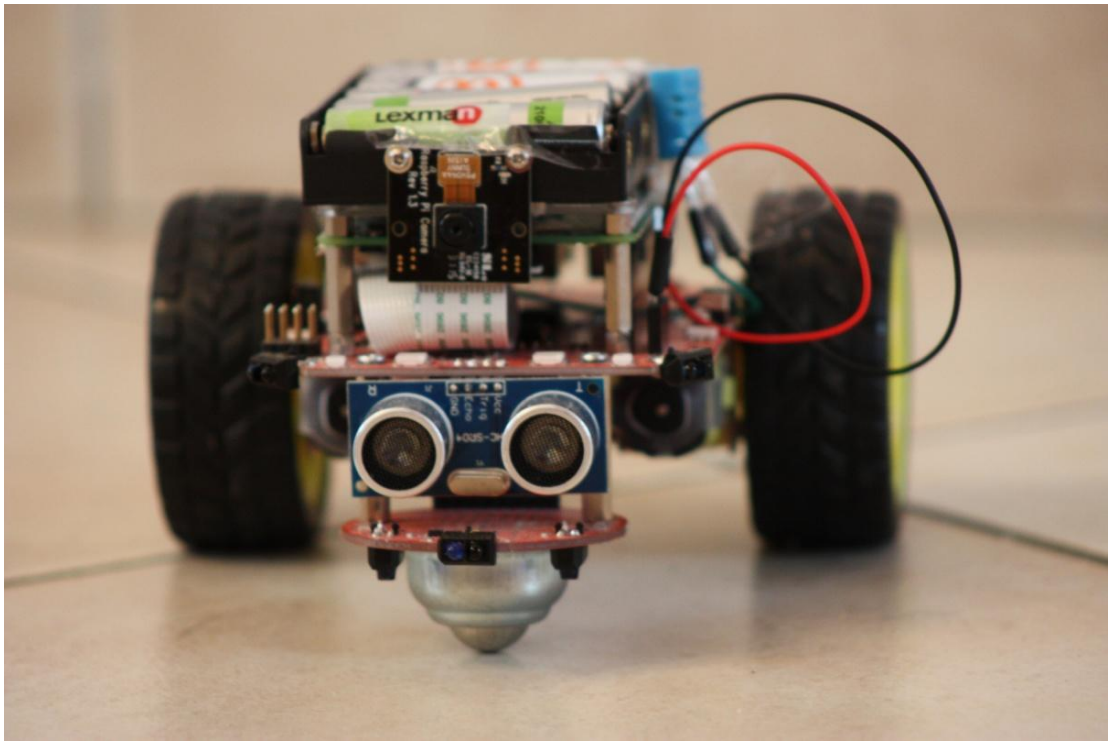

Fig. 20 – Prototipo InspectorPi completo

A partire dalla presentazione del device e dalle analisi effettuate, nel prossimo capitolo andremo quindi a modellare il software, mostrando sia gli schemi di progettazione che alcune parti essenziali della sua implementazione.

Capitolo 5 -
Progettazione e implementazione

Dopo aver presentato il prototipo realizzato, identificandone i singoli componenti, andremo ora a rappresentare l'architettura del sistema complessivo, individuando le tecnologie utilizzate, per poi progettare il software del sistema sulla base dell'architettura scelta e presentare la sua implementazione.

5.1 L'architettura del sistema

In fase di progettazione abbiamo preso importanti decisioni riguardanti l'architettura del sistema; in particolare abbiamo individuato le piattaforme che utilizzeremo per eseguire i programmi che ci permetteranno di comandare InspectorPi. Abbiamo quindi effettuato alcune scelte riguardanti i tre componenti del sistema, riportate in seguito:

- **Architettura di InspectorPi**: come già specificato nel capitolo 4, il device impiegherà come piattaforma Raspberry Pi. Per quanto riguarda il sistema operativo del dispositivo, si è scelto di utilizzare la Java Virtual Machine (JVM), per poter creare un programma il più possibile indipendente dalla piattaforma in esecuzione: nel caso in cui in futuro venga utilizzata una piattaforma diversa, come per esempio la BeagleBone, basterà modificare solo piccole parti del software per renderlo agibile dalla nuova scheda. Sopra la JVM verrà eseguito il programma da noi creato: InspectorPi.

- **Architettura del dispositivo mobile**: come dispositivo mobile di utilizzo si è scelto di impiegare uno smartphone o tablet in cui fosse presente il sistema operativo Android con architettura ARM. Su Android verrà quindi installata l'applicazione che comanderà il dispositivo, chiamata "InspectorPi Controller".

- **Architettura del Cloud**: date le tipologie di servizi Cloud disponibili in rete, si è scelto di utilizzare un servizio PaaS, che ci permetterà di fruire dei servizi

della piattaforma senza implementare un programma apposito. Tra le varie PaaS si è scelto di utilizzare Xively[21], gratuita per uso personale.

- **Architettura di rete**: per far comunicare tra loro i componenti del sistema, si è scelto di utilizzare un router cui è possibile accedere sia via Wi-Fi che via Ethernet, connesso alla rete Internet per permettere ai dispositivi di inviare informazioni alla piattaforma Cloud. Per rendere il progetto più autonomo possibile, si è scelto di implementare la possibilità di una connessione Bluetooth per fornire a InspectorPi le credenziali di rete.

Definite queste specifiche, possiamo esprimere l'architettura dell'intero sistema attraverso lo schema in figura.

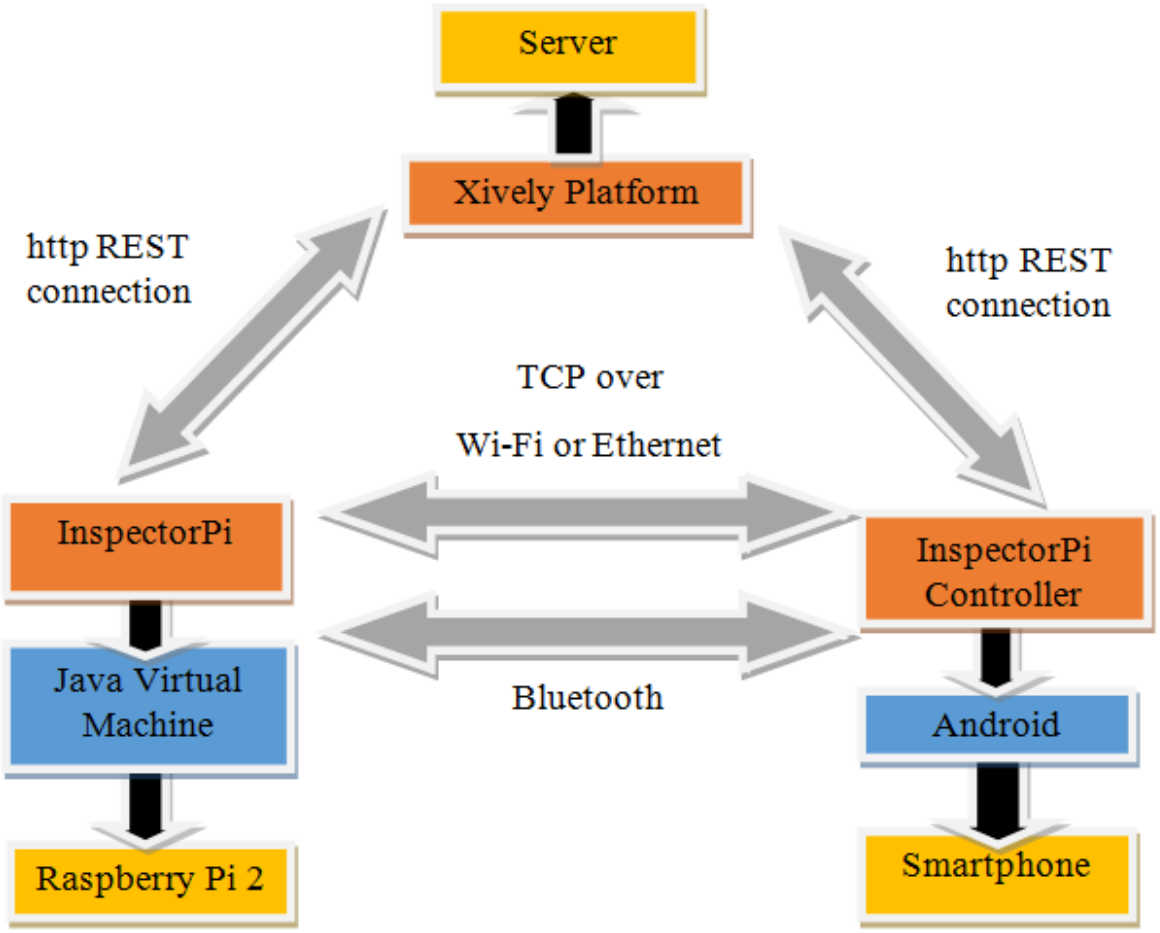

Fig. 21 – Architettura globale del sistema

[21] https://xively.com/

5.2 Modellazione e progettazione

Partendo dall'analisi effettuata nel capitolo 3, possiamo ora modellare il sistema progettandone le singole parti, ridefinendo gli schemi presentati in base all'architettura scelta. In particolare, cercheremo di dividere in due parti distinte ogni parte della progettazione: la progettazione sul prototipo InspectorPi e quella sul dispositivo mobile Android.

5.2.1 Progettazione delle classi per InspectorPi

Con il seguente diagramma delle classi cercheremo di rappresentare il modo statico ogni classe che compone il programma InspectorPi, escludendo la parte relativa alla comunicazione Bluetooth: come verrà spiegato nel capitolo 6, ci sono stati dei problemi iniziali, in conseguenza dei quali non abbiamo modellato e implementato questo tipo di comunicazione.

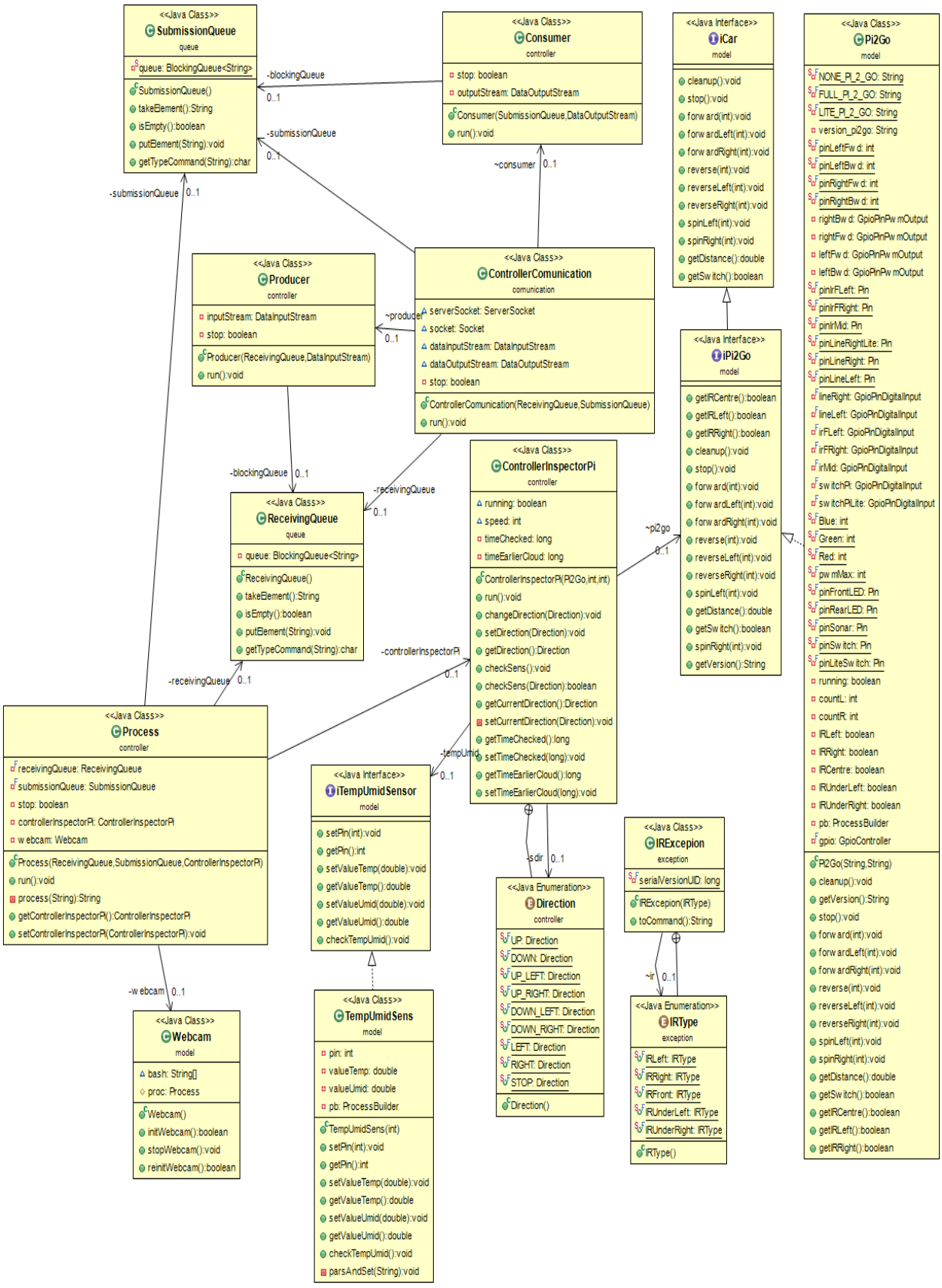

Fig. 22 – Diagramma delle classi di InspectorPi

Come si evince dallo schema, le tre classi in cui vengono gestite tutte le funzionalità del dispositivo sono *ControllerComunication, Process* e *ControllerInspectorPi*.

La prima si occupa di creare una socket server TCP/IP per la comunicazione con il dispositivo mobile e gestisce, tramite le classi che istanzia, tutto ciò che concerne lo scambio di messaggi con il dispositivo mobile.

La seconda processa i messaggi ricevuti inserendo le risposte nei messaggi inviati e avvia e arresta il processo che di acquisizione e invio dello streaming video.

La terza, invece, si occupa di gestire l'hardware associato al dispositivo e quindi comanda tutti i sensori e attuatori che vengono utilizzati.

Possiamo quindi descrivere le classi associate a *ControllerComunication* esponendo brevemente le loro funzionalità:

- **Producer**: si occupa di prelevare tutti i messaggi in entrata alla socket di comunicazione e inserirli nella coda di ricezione. Viene gestita da un *Thread* che la esegue fino al comando di termine comunicazione.

- **Consumer**: si occupa di prelevare dalla coda di trasmissione i messaggi e inserirli nei messaggi di uscita della socket di comunicazione; è sempre gestito da un *Thread* che ne esegue la funzionalità fino al termine della comunicazione.

- **SubmissionQueue**: istanzia una coda bloccante utilizzata in trasmissione e ne limita l'utilizzo.

- **ReceivingQueue**: istanzia una coda bloccante utilizzata per la ricezione e ne limita l'utilizzo.

Le classi utilizzate da *Process* sono invece le due seguenti:

- **Webcam**: si occupa della gestione del processo associato al Module camera NoIR, che tramite il metodo *initWebcam()* avvia un processo bash che utilizza *Raspivid e NetCat* per catturare e trasmettere il flusso video, arrestabile con il metodo *stopWebcam()*.

- **ControllerInspectorPi**: serve per eseguire i comandi processati. Questa classe possiede a sua volta questa ulteriori funzionalità.

Le classi che istanzia *ControllerInspectorPi* sono infine le seguenti:

- **iTempUmidSensor**: un'interfaccia implementata da *TempUmidSens*; ha la possibilità di catturare i valori del sensore DHT11 e restituirli singolarmente su richiesta, sia numerici che stringhe, con associata l'unità di misura.

- **iPi2Go**: un'interfaccia che estende *iCar*, implementata da *Pi2Go*; accede a tutte le funzionalità della kit-board Pi2Go e ne sfrutta quasi la totalità delle potenzialità, quali comandare i motoriduttori, catturare gli eventi sui sensori di prossimità e ottenere il valore del sensore di distanza.

Quelle appena elencate sono le classi principali associate al programma InspectorPi. Solo nel caso si utilizzi il Cloud per l'invio dei sensori, il programma utilizza un'ulteriore classe *XivelyTask*, lanciata da un *Thread* ogni due secondi per l'invio dei sensori di temperatura, distanza e umidità tramite un HttpClient e le api di Xively nella piattaforma Cloud.

5.2.2 Progettazione delle classi per dispositivo Android

Dato che si è scelto di realizzare l'applicazione che comanda il prototipo su piattaforma Android, in ogni parte della progettazione dell'applicazione sarà ricorrente il concetto di Activity. Le Activity sono uno degli elementi base di Android, con cui si definisce cosa può fare l'utente.

Con il seguente diagramma delle classi cercheremo di rappresentare le classi che compongono l'applicazione InspectorPi Controller, escludendo la parte relativa alla comunicazione Bluetooth per le ragioni spiegate in precedenza.

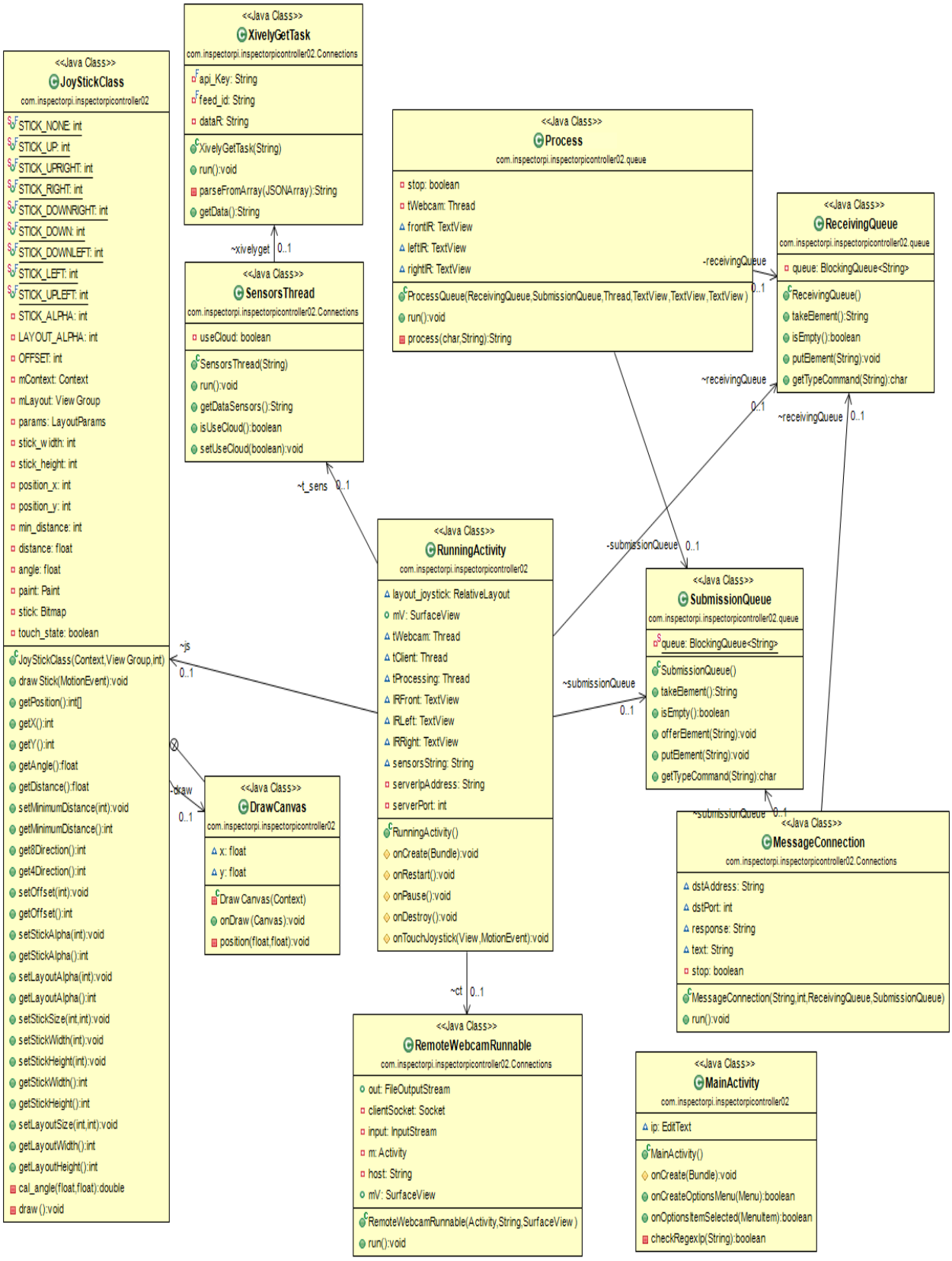

Fig. 23 – Diagramma delle classi di InspectorPi Controller

Le Activity che compongono il programma sono due: *MainActivity* e *RunningActivity*. In fase di modellazione era stata individuata una terza Activity riguardante lo scambio delle credenziali via Bluetooth, di cui sarà poi modellato il comportamento tramite il diagramma degli stati, che non è stata qui rappresentata perché nello schema si è cercato di rispecchiare ciò che è stato poi implementato.

MainActivity è l'Activity che viene eseguita all'avvio dell'applicazione, con la quale l'utente può inserire diversi parametri, quali l'IP dell'InspectorPi (se questo è già commesso alla rete Wi-Fi in comune) e come trasmettere e ricevere i sensori dal dispositivo. Al momento del tentativo di connessione, tramite Intent esplicito, esegue la seconda Activity, *RunningActivity*.

Su questa seconda Activity, che istanzia le classi che permettono di ottenere una telepresenza del dispositivo e ottenere i sensori, o dal Cloud o direttamente dal dispositivo, si basa tutto il funzionamento del programma. Le classi sono:

- **MessageConnection**: si occupa di aprire e gestire un socket client TCP/IP che si connette all'InspectorPi. Questa trasmetterà i comandi presenti nella coda di trasmissione e recupererà i comandi della coda di ricezione fino all'uscita dall'Activity.

- **Process**: processa i messaggi inseriti nella coda di ricezione tramite il metodo *Process(),* restituendo il messaggio da inviare tramite la coda di trasmissione.

- **ReceivingQueue**: utilizza e limita la coda bloccante di ricezione usata sia da *MessageConnection* che da *Process*.

- **SubmissionQueue**: utilizza e limita la coda bloccante di trasmissione usata sia da *MessageConnection* che da *Process*.

- **JoystickClass**: gestisce gli eventi *OnTouchListener()* che avvengono nell'Activity per determinare il punto preciso nel quale viene premuto il joystick. Questo viene poi sfruttato dall'Activity per determinare l'orientamento del comando da inserire nella coda di trasmissione.

- **SensorsThread**: questa classe, che estende *Thread*, istanzia la classe *XivelyGetTask*, la quale tramite metodo *run()* eseguito dalla prima classe, preleva i dati dal Cloud; questo soltanto se si è scelta l'opzione di ottenere i dati

dal Cloud, in caso contrario inserisce nella coda di trasmissione la richiesta dei sensori.

Dopo aver eseguito una breve descrizione delle classi principali implementate nel progetto, nel prossimo paragrafo andremo a studiarne il comportamento.

5.2.3 Progettazione degli stati per InspectorPi

Eseguiamo ora una modellazione del comportamento di alcune classi cercando di individuare il compito che verrà dato a ognuna di esse; seguendo il mapping dei task in thread secondo la tipologia one-to-one daremo poi in fase di implementazione un thread a ogni task sviluppato in questa fase. In particolare, non effettueremo una rappresentazione speculare di quella che sarà poi l'implementazione del comportamento, ma focalizzeremo l'attenzione sul comportamento delle singole parti.

Come primo diagramma presentiamo lo schema che modella il comportamento della classe *Producer*.

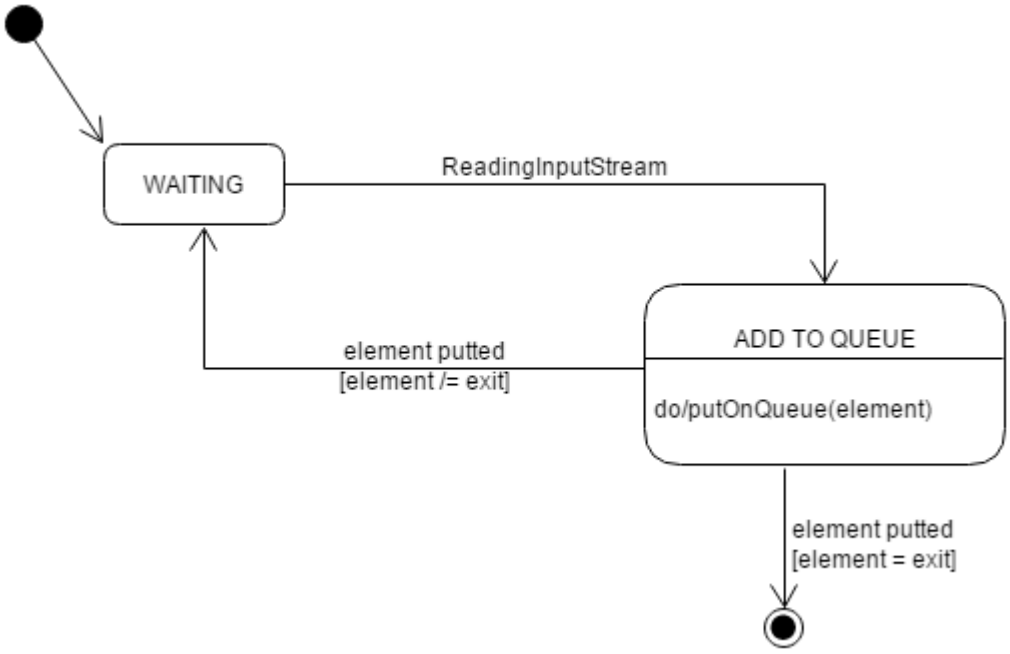

Fig. 24 – Diagramma degli stati della classe Producer

Come si può notare, all'avvio si entra in uno stato di attesa, fino a che non avviene la lettura di un elemento nell'*inputStream* della socket: questo fa scattare una transizione che porta a uno stato in cui viene inserito l'elemento nella coda di

ricezione. Alla fine dell'inserimento si va ad analizzare il messaggio: se era uguale a *exit*, avviene l'uscita dall'esecuzione, altrimenti si rientra nello stato di attesa, aspettando ulteriori messaggi. Questo comportamento serve per poter catturare tutti i messaggi che, dopo essere stati inseriti nella coda, possono essere processati da *Process*.

Contrapposta a *Producer* c'è *Consumer*, modellata con il diagramma seguente.

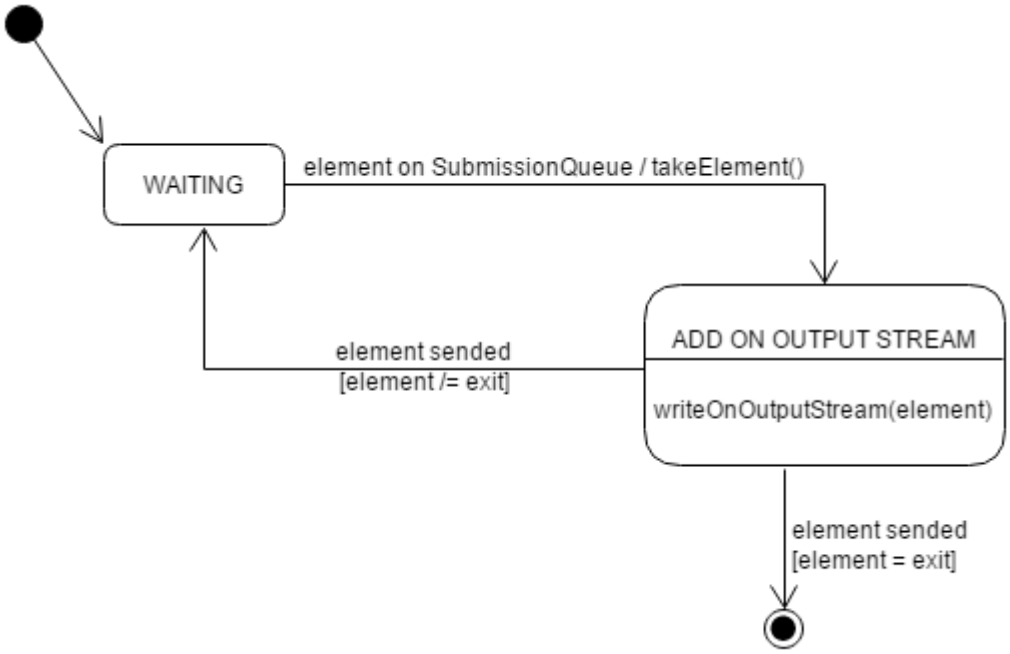

Fig. 25 – Diagramma degli stati della classe Consumer

Come si riesce facilmente a intuire, *Consumer* fa esattamente l'opposto di della classe precedente: entra inizialmente in uno stato di attesa e, quando è presente un elemento nella coda di trasmissione, lo preleva durante la transizione. Appena è possibile l'elemento viene poi inviato nello stato successivo tramite l'*OutputStream* della socket. Fatto ciò, *Consumer* esegue la stessa distinzione del messaggio di *Producer*, terminando il processo se necessario.

Accanto a questi due comportamenti ve ne è un terzo che chiude il ciclo eseguito dal messaggio. *Process*, infatti, come spiegato nel diagramma seguente, attende la presenza di un elemento nella coda di ricezione nello stato di *waiting*, poi preleva l'elemento durante transizione a un secondo stato, quindi lo processa ed estrapola la risposta ricordandosi, se corrispondente all'uscita, di terminare l'esecuzione una volta inserita la risposta nella coda di trasmissione.

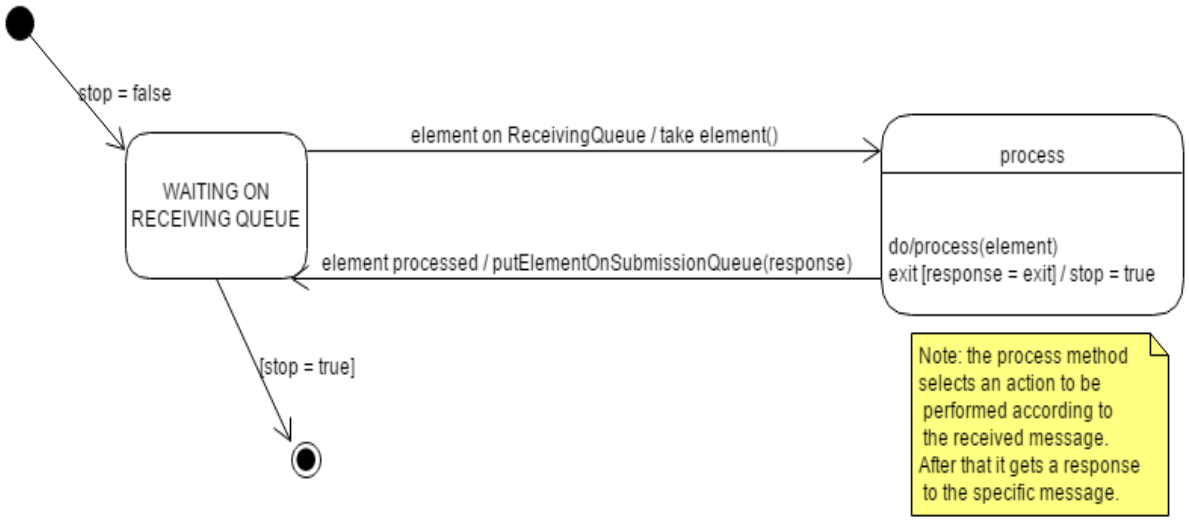

Fig. 26 – Diagramma degli stati della classe Process

Come ultimo schema riguardante il comportamento del device, abbiamo modellato *InspectorPiController*, che si occupa di verificare che la connessione al dispositivo non termini involontariamente quando il prototipo è in movimento e della gestione dei sensori. Se il dispositivo non viene contattato con una direzione per 500ms, infatti, *InspectorPiController* ferma il device. L'invio dei sensori, invece, nel caso in cui si sia scelto di utilizzare il Cloud come metodo di condivisione, avviene con una cadenza di due secondi. Tutto ciò è modellato nello schema seguente.

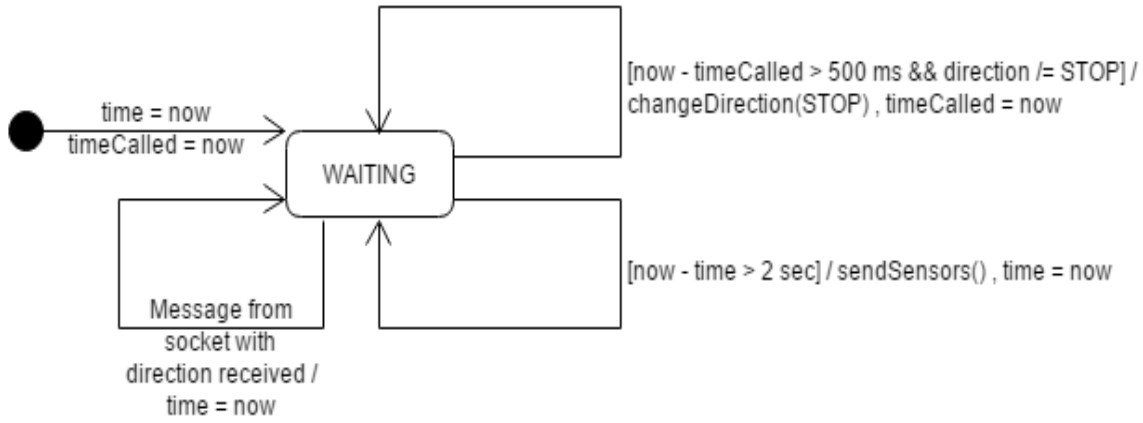

Fig. 27 – Diagramma degli stati della classe ControllerInspectorPi

Passiamo ora a modellare il comportamento in quella che sarà l'applicazione del dispositivo mobile.

5.2.4 Progettazione degli stati per dispositivo Android

Quello che cercheremo di fare ora è individuare i comportamenti che l'applicazione su dispositivo mobile deve avere per essere eseguita correttamente. In particolare, modelleremo il comportamento di quattro classi distinte: *RunningActivity*, *SensorsThread*, *Process* e *MessageConnection*.

La prima è l'Activity principale dell'applicazione che deve eseguire i comandi principali, amministrabili runtime dall'utente che esegue l'applicazione. Come presentato di seguito nel diagramma degli stati, infatti, vi sono due transizioni in cui l'Activity reagisce: *onJoystickPressed* e *onSensorsButtonPressed*. Entrambe le transizioni sono determinate dalla pressione sull'Activity, che determinerà due azioni conseguenti: nel primo caso l'aggiunta nella coda di trasmissione della direzione premuta nel joystick, mentre nel secondo caso la visualizzazione dei sensori tramite Snackbar.

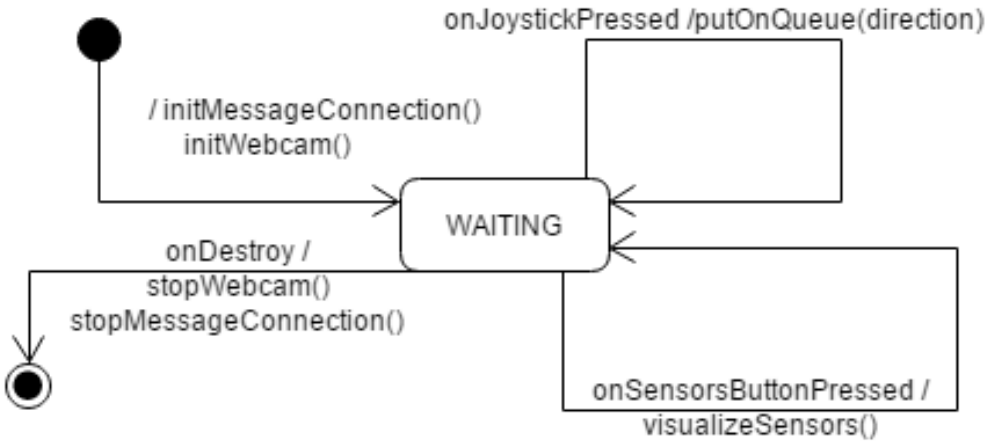

Fig. 28 – Diagramma degli stati della classe RunningActivity

La richiesta dei sensori, in Cloud o direttamente al dispositivo, viene effettuata ogni due secondi dal thread *SensorsThread*, tramite l'ausilio della classe che istanzia contenente il task. Si presuppone che in una futura versione del sistema, sia impostabile dall'utente nella *MainActivity* in base alle necessità.

Il comportamento del *SensorsTherad* parte quindi in uno stato di *waiting* e, dopo due secondi, entra in un secondo stato in base al tipo di trasmissione: se è stato scelta la comunicazione via Cloud, entrerà nello stato "*Get sensors from Cloud*", nel quale effettuerà la comunicazione http REST con le api della piattaforma Cloud prelevando i

valori dei sensori; se invece si è scelta la comunicazione diretta entrerà nello stato "*Get sensors from device*", dove inserirà il messaggio di richiesta dei sensori nella coda di trasmissione.

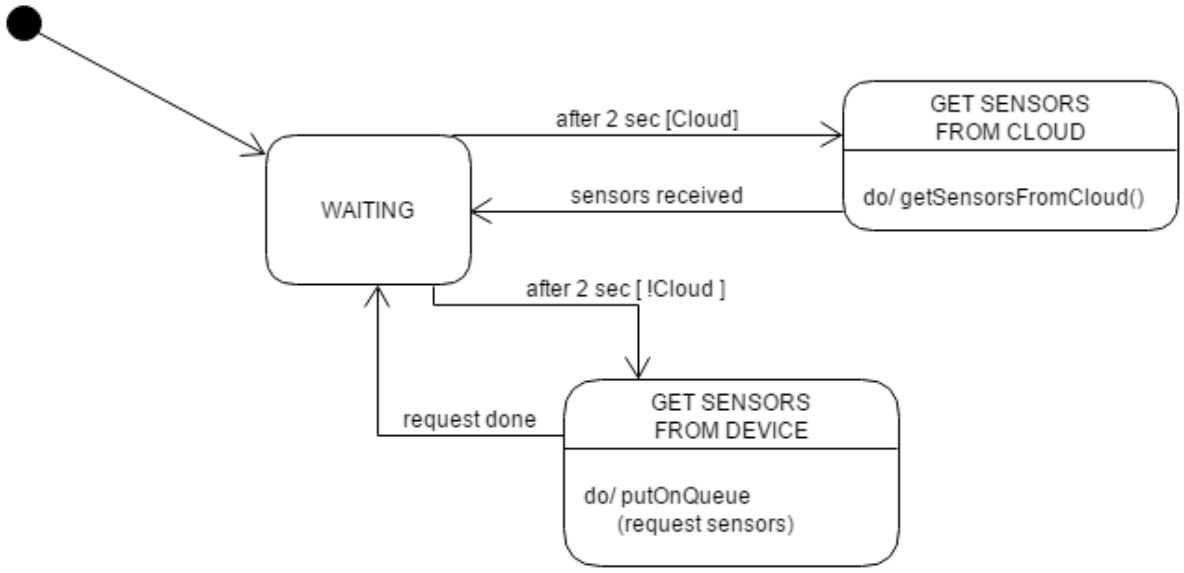

Fig. 29 – Diagramma degli stati della classe SensorsThread

Gli ultimi due comportamenti sono quello riguardante la processione dei comandi all'interno delle code di trasmissione e ricezione e quello riguardante la connessione con il device. Dato che il primo, eseguito da *Process*, è identico a quello di *Process* modellato all'interno del programma del prototipo, per la sua analisi si rimanda alla figura 26.

Il diagramma degli stati riguardante il comportamento di *MessageConnection* è invece rappresentato di seguito.

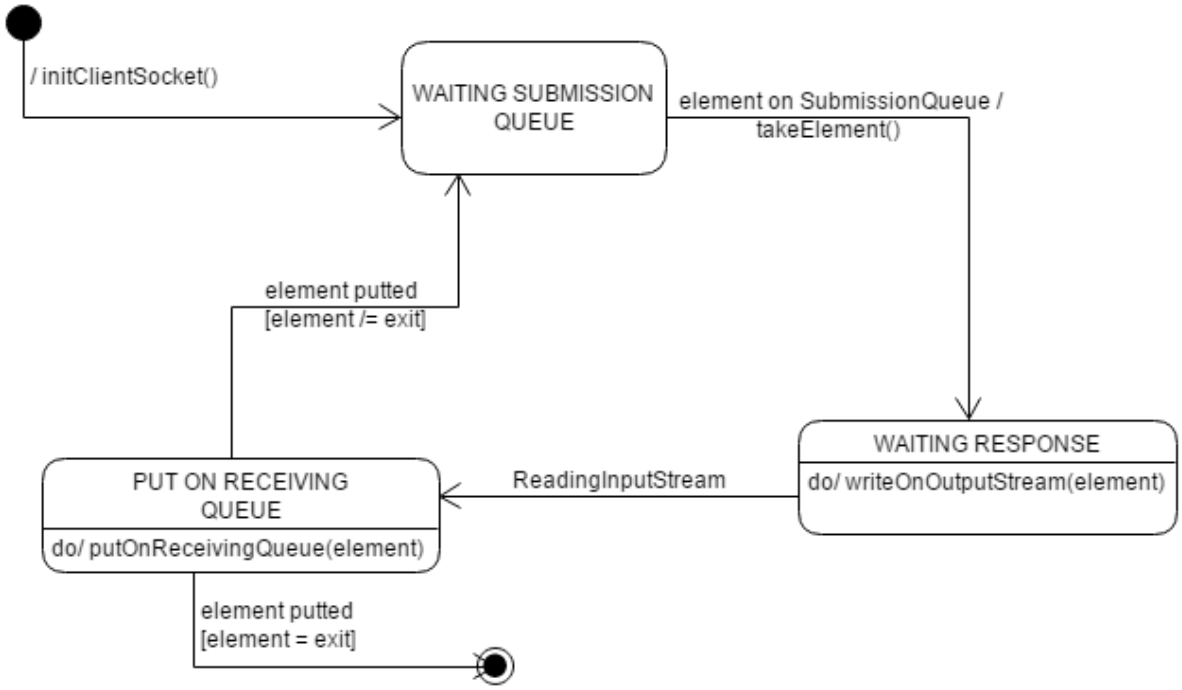

Fig. 30 – Diagramma degli stati della classe MessageConnection

Come possiamo vedere, vengono eseguiti quattro passi modellati da tre stati. Inizialmente, all'esecuzione del sottosistema viene inizializzata la socket e si entra in uno stato di attesa. Alla presenza di un elemento sulla cosa di trasmissione, poi, dopo che questo è stato prelevato, si entra in uno stato in cui si cerca di inviare l'elemento nell'*OutputStream* della socket. All'invio di quest'ultimo, avviene una transizione e quando si riceve un messaggio nell'*InputStream* della socket si cerca di inserirlo nella coda di ricezione durante lo stato successivo. A questo punto avviene una transizione in base al tipo di messaggio inserito: se riguarda l'uscita viene terminata l'esecuzione, in caso contrario si ritorna in attesa di un elemento nella coda di trasmissione, ricominciando il ciclo.

5.2.5 Progettazione delle interazioni

Dopo aver modellato i comportamenti riguardanti sia il programma sul device di monitoraggio e ispezione che sul dispositivo mobile, andremo a specificarne alcune interazioni tramite i diagrammi delle sequenze, cercando di sviluppare le interazioni analizzate nel paragrafo 3.2. Grazie alla modellazione effettuata precedentemente è stato infatti possibile riunire l'invio di tutti i messaggi di comando tra il dispositivo mobile e il prototipo in un due diagrammi delle sequenze, riportati in seguito.

Il primo riguarda la trasmissione di un messaggio e la possibile ricezione risposta da parte del dispositivo mobile. In figura è stato preso come esempio l'invio di una direzione dovuta alla pressione del joystick.

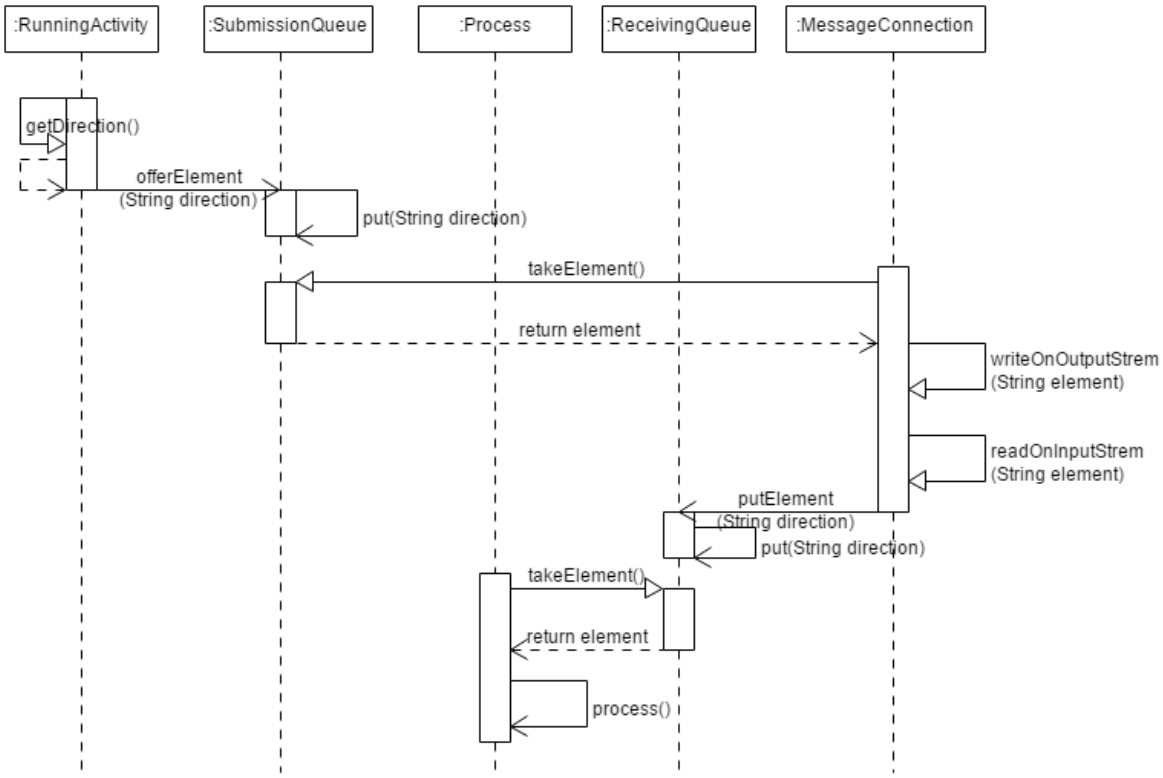

Fig. 31 – Diagramma delle sequenze relativo alla gestione dei messaggi da parte dell'applicazione Android

Il secondo riguarda invece la ricezione del messaggio dal prototipo, l'elaborazione dello stesso e l'invio della risposta. Così facendo, per qualsiasi messaggio ricevuto sarà *Process* a effettuare il comando e restituire una risposta.

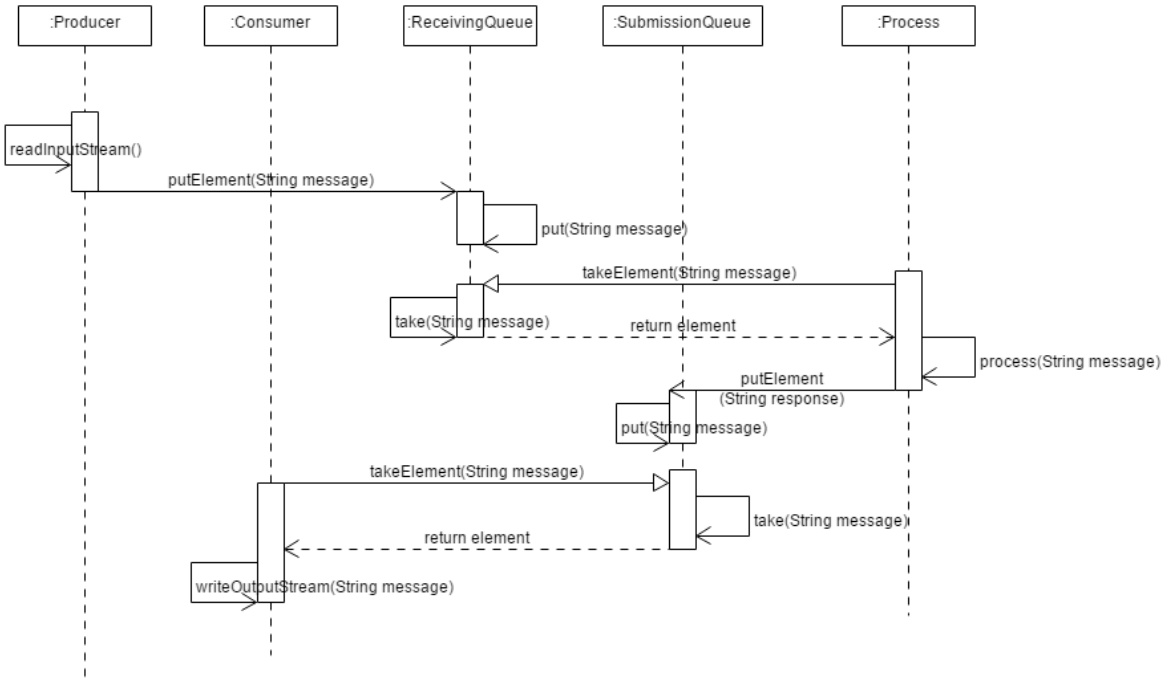

Fig. 32 – Diagramma delle sequenze relativo alla gestione dei messaggi da parte del prototipo

L'altra interazione importante da presentare è quella riguardante l'invio e la ricezione dei sensori. Questa interazione è possibile sia tramite Cloud (con la possibilità di storicizzazione e di visione simultanea dei valori), sia direttamente attraverso il dispositivo, sfruttando il canale di comunicazione mostrato nei diagrammi precedenti. Data la scelta, nel secondo caso, di utilizzare una comunicazione già presente, andiamo ora a presentare il diagramma delle interazioni nel caso in cui si sia scelto di utilizzare il Cloud service.

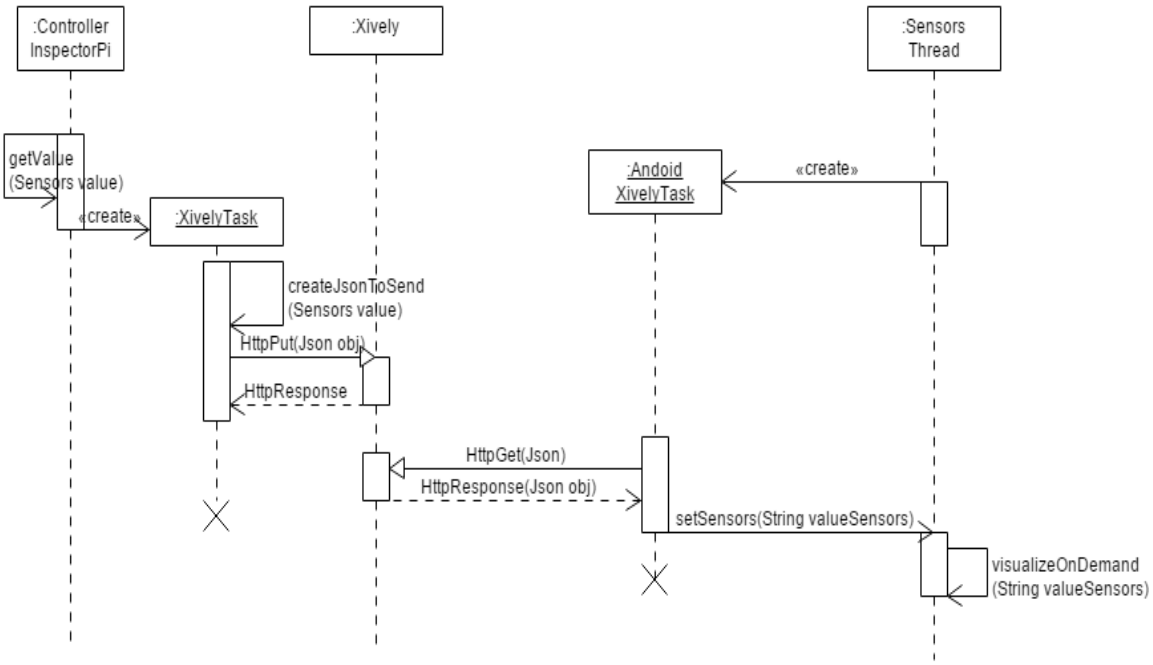

Fig. 33 – Diagramma delle sequenze relativo alla trasmissione e ricezione in Cloud

A sinistra della figura abbiamo il prototipo che invia un messaggio Json tramite comando *put* al server dove vi è la piattaforma. All'interno saranno presenti i dati che identificano il dispositivo per permettere a Xively di inserirli nell'account specifico. A destra troviamo invece il thread del dispositivo mobile, che tramite un task specifico richiede l'oggetto Json relativo al prototipo. Al momento della richiesta visualizza infine i sensori sullo schermo.

5.3 Implementazione

In questo paragrafo presenteremo alcune piccole parti del codice implementato, focalizzandoci su ciò che è stato trattato in modo meno approfondito nel resto del capitolo. Partiremo quindi dal programma sul prototipo per poi passare all'applicazione su dispositivo mobile.

5.3.1 Implementazione di InspectorPi

Per quanto riguarda il programma InspectorPi, mostreremo soltanto una minima parte del codice, che, per mancanza di spazio, non verrà presentato nella sua versione integrale.

Come descritto nel paragrafo 5.1 riguardante l'architettura del sistema, per la realizzazione del programma eseguito dal prototipo abbiamo utilizzato la piattaforma Java. In particolare, abbiamo usato l'IDE open-source Eclipse[22] per sviluppare l'applicazione e l'abbiamo poi testata direttamente sul prototipo. Come librerie esterne, oltre a quelle fornite dalla JDK, abbiamo utilizzato Pi4J, Json e le librerie http di Apache. Per l'acquisizione dei sensori, invece, abbiamo utilizzato dei programmi specifici in Python.

Per quanto l'esecuzione dei comandi, è dispensata alla classe *Process*. Il metodo *run()* di questa classe, che esegue il comportamento spiegato in figura 26, tramite il metodo *takeElement()* entra in attesa della disponibilità di un elemento sulla coda bloccante contenente stringhe. Quando questo è disponibile, lo preleva e lo processa tramite il metodo *process()*; quest'ultimo esegue una selezione dell'elemento in base al tipo e al comando specifico ed esegue il codice identificato.

[22] https://www.eclipse.org/

```
@Override
public void run() {
    while(!stop){
        try {
                String element = this.receivingQueue.takeElement();
            String response = process(element);
            if(response != null)
                this.submissionQueue.putElement(response);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    }
}

private String process(String element) {
    String response=null;
    char type = this.receivingQueue.getTypeCommand(element);
    switch(type){
    case MessageReceive.TYPE_GEN:
        switch(element){
            case MessageReceive.GEN_EXIT:
                […]
                response = MessageResponse.GEN_SHUTTINGDOWN;
            break;
            […]
        }
    […]
    }
    return response;
}
```

Listing 1 – Estratto di codice della classe Process per l'esecuzione del comandi

Come possiamo vedere, il metodo contiene un primo switch-case che identifica il tipo di comando e, per ogni tipologia, un secondo switch-case per identificare lo specifico comando ed eseguirne le operazioni. Per esempio, nel caso in cui ci si trovi in un comando riguardante il tipo "M", che corrisponde a "motion", il dispositivo modifica la direzione delle ruote o le ferma.

Tramite questo comando possiamo quindi accedere all'istanza della classe *ControllerInspectorPi*, che effettuerà il cambio di direzione come riportato in figura.

```
public void changeDirection(Direction d) throws IRExcepion{
    switch(d){
    case UP:
           if(!checkSens(Direction.UP)){
                  pi2go.forward(speed);
                  this.setCurrentDirection(Direction.UP);
           }
           else{
                  pi2go.stop();
                  this.setCurrentDirection(Direction.STOP);
           }
           break;
    case DOWN:
           if(!checkSens(Direction.DOWN)){
                  pi2go.reverse(speed);
                  this.setCurrentDirection(Direction.DOWN);
           }
           else{
                  pi2go.stop();
                  this.setCurrentDirection(Direction.STOP);
           }
           break;
           […]
    }
    if(d != Direction.STOP){
           this.running = true;
    }
    else{
           this.running = false;
    }
    this.timeChecked = System.currentTimeMillis();

}
```

Listing 2 – Estratto di codice della classe ControllerInspectorPi per l'esecuzione del comandi

Tramite il metodo *changeDirection(Direction d)* si accede infatti alla classe della kit-board e si cambia il movimento dei motori. Questo avviene soltanto se non è stato riscontrato un evento sui sensori. In questo modo, grazie ai sensori di prossimità, si protegge il dispositivo da eventuali scontri con ostacoli.

Come già anticipato, la cattura dei sensori, invece, avviene tramite l'esecuzione di appositi programmi per l'acquisizione dei sensori del prototipo. In particolare, come riportato nella figura seguente, al momento della richiesta da parte del programma di eseguire una cattura dei valori, la classe del sensore crea un processo tramite *ProcessBuilder* ed esegue il programma associato al dispositivo.

```
public class TempUmidSens implements iTempUmidSensor {

    private int pin;
    private double valueTemp;
    private double valueUmid;

    // Native call required for the DHT11 sensor. Java is not
    // fast enough for receiving.
    private ProcessBuilder pb = new ProcessBuilder("Python",
                "/home/pi/InspectorPi/DHT11.py");
    […]
    @Override
    public void checkTempUmid() {
          try {
                Process p = pb.start();

                      BufferedReader reader = new BufferedReader(
                            new InputStreamReader(
                                  p.getInputStream()));
                String temp_umid_values = reader.readLine();
                p.destroy();
                this.parsAndSet(temp_umid_values);
          } catch (IOException e) {
                e.printStackTrace();
          }
    }
    private void parsAndSet(String stringhe){
          int vir = stringhe.indexOf(',');
          this.setValueTemp(Double.parseDouble(
                            stringhe.substring(0, vir)));
          this.setValueUmid(
                      Double.parseDouble(stringhe.substring(vir+1)));
    }
```

Listing 3 – Estratto di codice della classe TempUmidSens per la cattura dei valori del sensore DHT11

Ciò che viene prelevato al termine del processo è soltanto la stringa contenente i valori dei sensori, che possono così essere distinti e settati nei campi della classe.

Presentiamo di seguito anche uno dei due programmi scritti in Python, nello specifico quello riguardante il sensore ultrasonico di distanza.

```
# UltraSonic Functions
#
# getDistance(). Returns the distance in cm to the nearest reflecting
# object. 0 == no object
# (Both versions)
#
def getDistance():
  GPIO.setup(sonar, GPIO.OUT)
  # Send 10us pulse to trigger
  GPIO.output(sonar, True)
  time.sleep(0.00001)
  GPIO.output(sonar, False)
  start = time.time()
  count=time.time()
  GPIO.setup(sonar,GPIO.IN)
  while GPIO.input(sonar)==0 and time.time()-count<0.1:
  start = time.time()
  count=time.time()
  stop=count
  while GPIO.input(sonar)==1 and time.time()-count<0.1:
  stop = time.time()
  # Calculate pulse length
  elapsed = stop-start
  # Distance pulse travelled in that time is time
  # multiplied by the speed of sound 34000(cm/s) divided by 2
  distance = elapsed * 17000
  return distance

try:
    dist = getDistance()
    print (int(dist * 10)) / 10.0

except:
  print
```

Listing 4 –Codice del programma in Python del sensore DHT11

Come si può notare, il programma manda un impulso brevissimo al sensore, aspettando subito dopo una risposta. L'onda in risposta determina poi la distanza campionata in base al tempo di pulsazione.

Come ultima parte significativa dell'implementazione andiamo a visionare l'invio dei sensori eseguito da *XivelyTask*.

```
public class XivelyTask extends Thread{

    private final String api_Key = "0FEY1ODJ***DuwlgV1EouM1fZ";
    private final String feed_id = "1144609154";
    private String temp;
    private String umid;
    private String dist;

    public XivelyTask(String temperatura, String umidita, String distanza) {
            this.temp = temperatura;
            this.umid = umidita;
            this.dist = distanza;
    }

    @Override
    public void run() {
            String responseString;
            try {
                    JSONObject dataJSON = new JSONObject();
                    dataJSON.put("version", "1.0.0");
                    JSONObject tempValue = new JSONObject();
                    tempValue.put("id","Temperatura");
                    tempValue.put("current_value", this.temp);
                    JSONObject umidValue = new JSONObject();
                    umidValue.put("id","Umidity");
                    umidValue.put("current_value", this.umid);
                    JSONObject distValue = new JSONObject();
                    distValue.put("id","Distance");
                    distValue.put("current_value", this.dist);
                    JSONArray datastreamsJSON = new JSONArray();
                    datastreamsJSON.put(tempValue);
                    datastreamsJSON.put(umidValue);
                    datastreamsJSON.put(distValue);
                    dataJSON.put("datastreams", datastreamsJSON);
                    CloseableHttpClient httpclient =
                                            HttpClients.createDefault();
                    HttpPut httpPut =
                                    New HttpPut(
                                    "https://api.xively.com/v2/feeds/" +
                            this.feed_id);
                    httpPut.setHeader("X-ApiKey", this.api_Key);
                    httpPut.setHeader("content-type", "application/json");
                    StringEntity entity =
                                    new StringEntity(dataJSON.toString());
                    httpPut.setEntity(entity);
                    HttpResponse httpResponse = httpclient.execute(httpPut);
                    responseString = (httpResponse.getEntity()).toString();

            }
            […]
```

Listing 5 – Estratto di codice della classe XivelyTask per l'invio dei sensori in Cloud

Per inviare i sensori alla piattaforma Cloud, il programma crea un oggetto Json contenente tutte le informazioni da caricare dei sensori, poi li invia tramite il comando put all'indirizzo *api.xively.com.*

5.3.2 Implementazione di InspectorPi Controller

Passiamo ora a visionare alcune parti dell'implementazione dell'applicazione per Android "InspectorPi Controller". Per creare quest'applicazione ci siamo serviti del programma IDE Android Studio[23], che abbiamo testato su AVD (Android Virtual Device) e Samsung Galaxy S4. Per rappresentare l'applicazione, abbiamo inoltre elaborato un simbolo esplicativo:

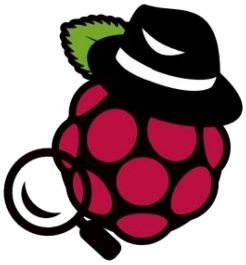

Fig. 34 – Simbolo dell'applicazione InspectorPi Controller

L'invio dei comandi tramite joystick virtuale è una delle parti che caratterizzano il codice di quest'applicazione. Grazie alla classe *Joystick*, infatti, possiamo determinare il punto del joystick sul quale l'utente ha premuto il dito. La pressione del dito determina un evento chiamato *onTouchListener()* gestito da Android, nel quale si inserisce il layout del joystick, che esegue il comando specifico come riportato nella figura seguente.

[23] https://developer.android.com/sdk/index.html

```
   @Override
protected void onCreate(Bundle savedInstanceState) {
       […]
         js = new JoyStickClass(getApplicationContext()
                  , layout_joystick, R.drawable.image_button);
       […]
         layout_joystick.setOnTouchListener(new View.OnTouchListener() {
                  public boolean onTouch(View arg0, MotionEvent arg1) {
                           try {
                           onTouchJoystick(arg0, arg1);
                           } catch (InterruptedException e) {
                           return false;
                           }
                                    return true;
                           }

       });
       […]
   }

protected void onTouchJoystick(View arg0, MotionEvent arg1) throws
InterruptedException {
  js.drawStick(arg1);
  if (arg1.getAction() == MotionEvent.ACTION_DOWN
    || arg1.getAction() == MotionEvent.ACTION_MOVE) {

  int direction = js.get8Direction();
  if (direction == JoyStickClass.STICK_UP) {
    this.submissionQueue.offerElement(MessageResponse.MOT_UP);
  } else if (direction == JoyStickClass.STICK_UPRIGHT) {
    this.submissionQueue.offerElement(MessageResponse.MOT_UP_RIGHT);
  } else if (direction == JoyStickClass.STICK_RIGHT) {
    this.submissionQueue.offerElement(MessageResponse.MOT_RIGHT);
  } else if (direction == JoyStickClass.STICK_LEFT) {
    this.submissionQueue.offerElement(MessageResponse.MOT_LEFT);
  } else
                  […]
                  if (direction == JoyStickClass.STICK_UPLEFT) {
    this.submissionQueue.offerElement(MessageResponse.MOT_UP_LEFT);
  } else if (direction == JoyStickClass.STICK_NONE){
  }
  } else if (arg1.getAction() == MotionEvent.ACTION_UP) {
  this.submissionQueue.offerElement(MessageResponse.MOT_STOP);
  }
  }
}
```

Listing 6 – Estratto di codice della classe RunningActivity per l'invio del comando associato al joypad

Tramite l'evento riusciamo così a inserire nella coda bloccante di trasmissione il comando che deve essere inviato al dispositivo. Così facendo riusciamo a muovere il dispositivo e a fermarlo nell'istante in cui rilasciamo la pressione del joystick.

Un’altra parte essenziale del programma riguarda la ricezione e la visualizzazione della telecamera in real-time. Per questa funzione è stato utilizzato un algoritmo sviluppato da Alexander K. Seewald, che ci permette di effettuare una decodifica real-time del flusso video grezzo h.264 ricevuto. Grazie a esso siamo infatti riusciti ad avere una latenza molto inferiore a un secondo.

Per quanto riguarda l’implementazione dei sensori, infine, il canale di ricezione può essere deciso dall’utente come presentato in figura.

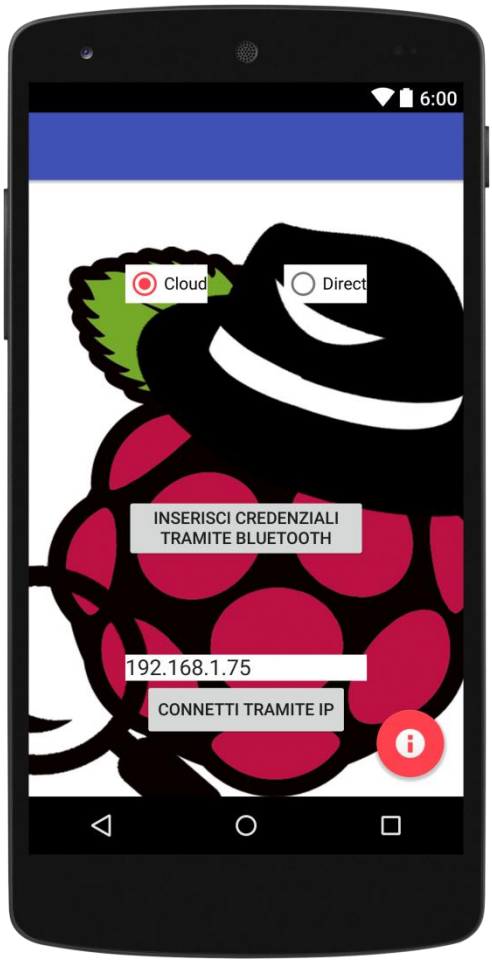

Fig. 35 – Schermata iniziale dell’applicazione InspectorPi Controller

Grazie alla possibilità di scegliere se sfruttare le potenzialità del Cloud o la semplicità di una connessione diretta per l’invio dei valori del sensore, l’utente riesce a gestire il dispositivo anche in assenza di connessione Internet.

Come ultimo estratto, andiamo a presentare quindi la ricezione dei sensori tramite l’ausilio della piattaforma Cloud fornita da Xively.

```
@Override
public void run(){

  Log.d("TEST", "Eseguendo Task per Xively");

  JSONObject dataJSON = null;
  try {

  // Contattare Xively per riceve il feed
  HttpClient httpClient = new DefaultHttpClient();
  HttpGet httpget = new HttpGet("https://api.xively.com/v2/feeds/" +
feed_id +".json");
  httpget.setHeader("X-ApiKey", api_Key);
  httpget.setHeader("content-type", "application/json");

  HttpResponse httpResponse = httpClient.execute(httpget);
  String responseString = EntityUtils.toString(httpResponse.getEntity());
  Log.d("TEST", "" + httpResponse.getStatusLine().getStatusCode());

  // Oggetto JSON ricevuto da far tornare.
  dataJSON = new JSONObject(responseString);

  } catch (JSONException e) {
  e.printStackTrace();
  Log.d("ERROR", e.getMessage());
  } catch (ClientProtocolException e) {
  e.printStackTrace();
  Log.d("ERROR", e.getMessage());
  } catch (IOException e) {
  e.printStackTrace();
  Log.d("ERROR", e.getMessage());
  }

  if(dataJSON == null){
  Log.d("TEST", "No data");
  return;
  }

  try {
  JSONArray datastreamsJSON = dataJSON.getJSONArray("datastreams");
  this.dataR = parseFromArray(datastreamsJSON);
  } catch (JSONException e) {
  e.printStackTrace();
  Log.d("ERROR", e.getMessage());
  }
    […]
```

Listing 7 – Estratto di codice della classe XivelyGetTask per la ricezione dei sensori dal Cloud

Come possiamo notare, l'applicazione utilizza il client http fornito di default per richiedere un oggetto Json ad *api.xively.com*. L'oggetto, dopo essere stato ricevuto, viene modificato in una stringa per poter essere presentato all'utente come nella figura seguente.

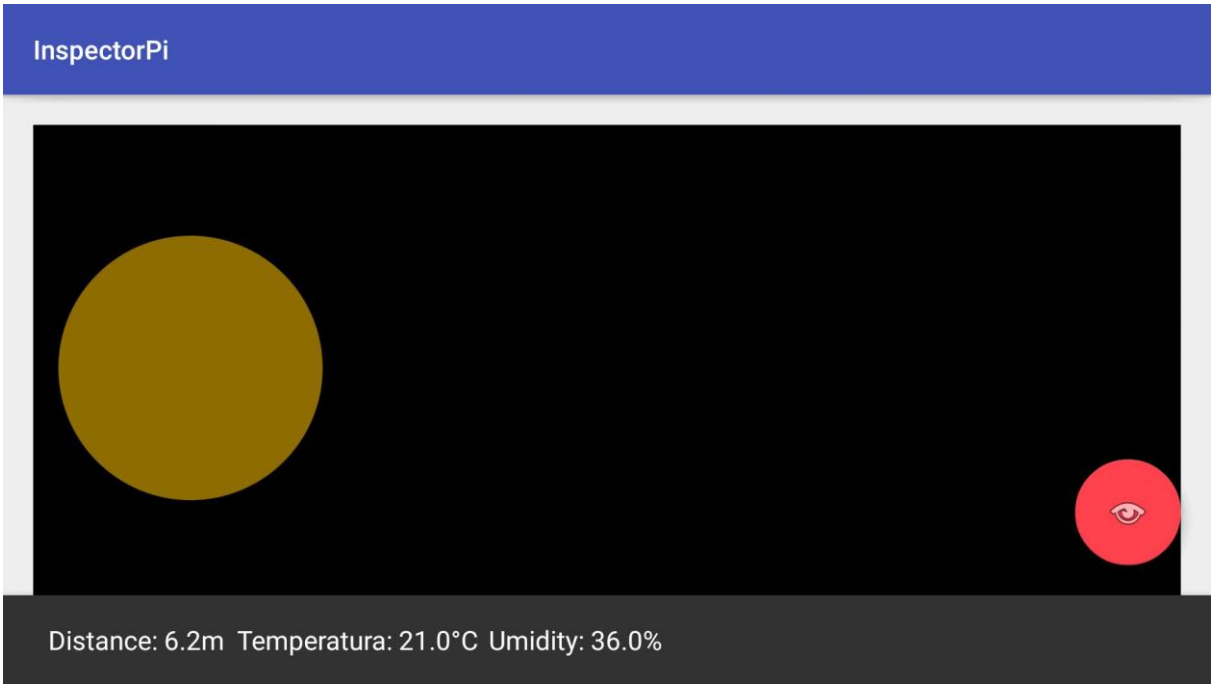

Fig. 36 – Visualizzazione dei valori dei sensori dall'applicazione InspectorPi Controller

Dopo aver progettato e implementato il software di InspectorPi e InspectorPi Controller all'interno di questo capitolo, nel prossimo andremo a testarne il funzionamento, collaudando così il nostro prototipo.

Capitolo 6 -
Test, collaudo e verifica

Nel corso di questo capitolo si testerà il corretto funzionamento di ogni parte del prototipo nei vari momenti di sviluppo. In particolare verranno riportati i test iniziali, le problematiche riscontrate durante lo sviluppo del sistema e il collaudo a sistema ultimato. Cercheremo infine di focalizzarci sui possibili sviluppi futuri del progetto e sulle eventuali migliorie da apportare al dispositivo.

6.1 Test iniziali sul prototipo

Dopo aver ultimato la realizzazione del prototipo, prima di utilizzare il programma da noi implementato, abbiamo testato la corretta esecuzione del dispositivo per assicurarci che eventuali problemi in fase di collaudo non fossero dovuti a limitazioni legate al prototipo stesso.

Come prima cosa abbiamo testato il corretto funzionamento della scheda Raspberry Pi 2. In seguito, dopo aver aggiornato i pacchetti di Raspbian tramite i comandi generalmente usati "apt-get update" e "apt-get upgrade", abbiamo installato i pacchetti utili per il collaudo della kit-borad Pi2Go. Per verificare il corretto funzionamento del dispositivo formato da Raspberry Pi e Pi2Go, sfruttando WiringPi e Python abbiamo potuto utilizzare i semplici programmi di test del sistema forniti dal produttore. In particolare, abbiamo esaminato le seguenti parti: motori, sensori e alimentazione.

Come previsto, i due motoriduttori riescono a far raggiungere una velocità elevata al device (il costruttore garantisce infatti 240rpm). Non siamo però riusciti a constatare se i motoriduttori riescono effettivamente a fare quattro giri al secondo, perché il progetto si basa sulla reattività, non sulla velocità.

Come già specificato nel capitolo 4, i sensori della kit-board presenti sono cinque sensori di prossimità anti-riflesso (i due nella parte inferiore vengono utilizzati per seguire un percorso predeterminato nel caso in cui ci si serva di una guida colorata) e un sensore di distanza. I primi, grazie alla presenza di led associati ai singoli sensori e a un microcontrollore, sono stati testati sia attraverso un programma di test che senza Raspberry Pi, monitorando la corretta accensione dei led alla vicinanza di un ostacolo

al sensore. Il sensore di distanza, invece, è stato provato tramite un programma di test, grazie al quale abbiamo verificato che misurasse correttamente la distanza dall'ostacolo presente di fronte a sé.

Per quanto riguarda l'alimentazione, invece, il consumo è dovuto a diversi fattori: la velocità dei motori, i dongle, i dispositivi associati a Raspberry Pi e gli eventuali sensori aggiuntivi. In fase di testing iniziale, quando ancora nel sistema non erano presenti i due USB dongle e i sensori, il dispositivo, con un'alimentazione data da sei batterie AA da 2100mAh ricaricabili, ha raggiunto un'autonomia di circa trenta minuti in continuo movimento.

Siamo passati poi al test della module camera NoIR: abbiamo verificato che, con l'ausilio dell'IR, riuscisse a catturare correttamente il video sia in condizioni di luce che di ombra, ottenendo esito positivo.

Infine, come ultimo test, abbiamo collegato il sensore DHT11 a Raspberry Pi e ne abbiamo monitorato i valori tramite un programma apposito, sviluppato in Python sfruttando le librerie fornite dal produttore del sensore, che verrà poi lanciato anche come processo nel programma InspectorPi del nostro sistema.

Durante questi test iniziali non sono stati riscontrati problemi di hardware o di misurazione riguardanti il prototipo.

6.2 Collaudo del sistema

6.2.1 Problematiche riscontrate

In fase d'implementazione del programma, sono stati effettuati dei test intermedi per stabilire il corretto funzionamento dei dispositivi con le piattaforme utilizzate. Come già accennato nel capitolo 5, abbiamo riscontrato degli errori nelle librerie BlueCove che forniscono le librerie Bluetooth per l'utilizzo su piattaforma Java, in particolare nell'identificazione del dongle Bluetooth. Pur provando a ricompilare quest'ultime, non sono stati ottenuti i risultati desiderati. Per ottenere la completa esecuzione del sistema, quindi, in futuro bisognerà trovare il modo di ovviare al problema. Si potrebbe per esempio a utilizzare le librerie Java RxTx per la comunicazione seriale, lasciando al sistema operativo l'inizializzazione e la gestione del Bluetooth.

Durante l'implementazione del software, poi, si è deciso di eseguire dei singoli processi per campionare correttamente i valori dei sensori, poiché questi, utilizzando lo stesso pin sia per ricevere che per trasmettere, non rilevavano correttamente le

misure tramite JVM. Invece di sfruttare le potenzialità di JNI (Java Native Interface), si è scelto di utilizzare dei processi per eseguire piccoli programmi in Python associati ai sensori, sia per semplicità, sia per garantire l'intercambiabilità dei sensori. Impiegando un sensore differente, infatti, se si modificasse il solo programma in Python non si rischierebbe di compromettere l'intera esecuzione del sistema.

6.2.2 Esecuzione del sistema

Durante l'esecuzione del sistema abbiamo notato che l'applicazione Android presenta un flickering (o sfarfallio) derivante dalla codifica che esegue l'applicazione in real-time. Questo effetto è dovuto alla scelta di eseguire la decodifica del flusso h.264 grezzo, che non dà al programma la possibilità di accedere alle informazioni riguardanti il video per comprendere quali parametri ne permetterebbero una fluida esecuzione. Prima di scegliere di utilizzare un flusso raw con l'ausilio di NetCat per la trasmissione, abbiamo testato altre due configurazioni differenti. In particolare, abbiamo provato a utilizzare il protocollo RTSP tramite VLC, il più comune per l'invio dello streaming video in real-time. Questa configurazione non è però stata usata perché comportava una latenza media di 1,5 secondi, ovvero di circa tre volte maggiore rispetto ai 0,4 secondi della configurazione scelta. L'altra opzione provata è stata GStreamer, che, pur avendo latenze simili a quelle di NetCat, era maggiormente complessa da utilizzare.

Per quanto riguarda la correttezza dei messaggi scambiati tra il dispositivo mobile e il prototipo, non abbiamo riscontrato errori: i messaggi vengono ricevuti, eseguiti e inviati correttamente, così come ci eravamo preposti in fase di analisi.

6.2.3 Reattività del sistema

La reattività di un sistema sta nella latenza di risposta da parte del dispositivo a uno specifico comando, nonché nella latenza dello streaming video in real-time. Bisogna quindi analizzare due casi distinti: il caso in cui si utilizza un cavo Ethernet e quello in cui si utilizza il dongle Wi-Fi.

Nel caso sul prototipo si impieghi la comunicazione tramite cavo Ethernet, la latenza riguardante lo scambio dei messaggi ci è parsa trascurabile. Il video in streaming real-time, inoltre, è sempre stato molto reattivo, con una latenza al di sotto degli 0,4 secondi.

Se invece si utilizza il dongle Wi-Fi la latenza è influenzata dalla distanza del Wi-Fi, secondo i limiti della tecnologia. Abbiamo infatti rilevato che, mentre l'attesa dei

comandi è pressoché trascurabile, quella del video è mediamente di 0,5 secondi in prossimità del router Wi-Fi. Se il segnale Wi-Fi è debole, però, il ritardo aumenta esponenzialmente: anche se la latenza dei comandi non pregiudica la corretta esecuzione, quella dello streaming video interferisce con la telepresenza, poiché il feedback medio supera gli 0,7 secondi.

6.3 Evoluzioni future

Benché il sistema creato nel corso di questa tesi funzioni secondo i criteri di analisi prefissati, in seguito per InspectorPi si potrebbero configurare ulteriori sviluppi.

La prima e più importante evoluzione del sistema sarebbe sicuramente l'implementazione della comunicazione Bluetooth: in questo modo si potrebbe ovviare alle problematiche dovute all'utilizzo del sistema su reti Wi-Fi differenti, che richiederebbero un inserimento manuale delle credenziali.

In relazione gli scenari utilizzo, poi, l'ispezione dei condotti di aerazione, il monitoraggio di fughe di gas o l'uso ricreativo del device sono solo alcuni tra i possibili modi in cui il prototipo potrebbe essere impiegato in futuro. Effettuando ricerche e analisi aggiuntive, infatti, il prototipo potrebbe essere utile per campionare dati in numerosi contesti. Potrebbe per esempio servire da rilevatore termografico, oppure potrebbe essere sfruttato in ambienti ad alto rischio, in cui l'uomo non ha la possibilità di agire in prima persona; un caso potrebbe essere quello dell'ispezione di zone con alte concentrazioni di radioattività.

Per quanto riguarda la guida del sistema, infine, grazie alle evoluzioni tecnologiche in atto, un domani il dispositivo potrebbe essere comandato da un'intelligenza artificiale (AI) in rete, che eseguirebbe il monitoraggio degli ambienti al posto della persona. Sfruttando i sensori IR rivolti a terra del dispositivo sarebbe infatti possibile analizzare il terreno su cui si muove il veicolo, determinando così automaticamente le linee guida da seguire.

Conclusioni

Nel corso di questo progetto di tesi sono state affrontare numerose fasi di ricerca, che ci hanno portato in primo luogo ad approfondire la nostra conoscenza delle tecnologie correnti. Ci siamo immersi nel mondo di Internet of Things, dei sistemi Embedded e della robotica, campi dai quali abbiamo tratto informazioni e ispirazione per poter realizzare un nuovo, innovativo prototipo.

Abbiamo quindi delineato le caratteristiche generali che avrebbe dovuto avere il nostro device, pensato sin dall'inizio per svolgere funzioni di monitoraggio e ispezione. Ciò che ci eravamo prefissi di fare, infatti, era sviluppare un prototipo la cui strutturazione e il cui utilizzo avessero valenza non solo teorica, ma che potesse anche avere un'utilità pratica.

A questo punto abbiamo deciso come realizzare il dispositivo, scegliendo le componenti hardware più adeguate agli scopi prefissi. Dopo aver stabilito come sarebbero dovute avvenire le interazioni tra la parti, ci siamo concentrati sulla modellazione e implementazione del software. In questa fase, le difficoltà maggiori che abbiamo riscontrato sono state dovute ad alcune lacune di informazioni in materia di streaming dei contenuti e sviluppo di applicazioni per dispositivi mobili. Per realizzare il progetto, invero, abbiamo dovuto studiare in modo approfondito le modalità di streaming video e gli elementi base delle applicazioni Android: questo ci ha permesso di aumentare la nostra consapevolezza e di ampliare il nostro bagaglio di competenze in materia.

Un secondo ostacolo che abbiamo affrontato ha riguardato poi la comunicazione tramite tecnologia Bluetooth: non è stato possibile implementarla all'interno del sistema a causa di alcuni errori delle librerie essenziali per questo processo; probabilmente i problemi sono stati dovuti a un'incompatibilità tra il sistema operativo presente nella scheda Raspberry Pi e le librerie scelte per la comunicazione. Questo impedimento non ha però pregiudicato il corretto funzionamento del prototipo.

Nonostante i problemi riscontrati, infatti, grazie alle nostre conoscenze pregresse e alle nozioni apprese, siamo riusciti a sviluppare implementare nel device le principali funzionalità pensate inizialmente: abbiamo creato per la prima volta un dispositivo di monitoraggio efficiente, con costi di realizzazione contenuti e adatto a un target universale. Il veicolo che abbiamo progettato è munito di videocamera, comandato da smartphone attraverso l'uso di un'apposita applicazione Android (InspectorPi Controller), integrato con diversi tipi di sensori e capace di comunicare in rete i dati

rilevati in fase di attività. Grazie a una piattaforma Cloud, le informazioni trasmesse online dal device sono quindi potenzialmente consultabili da chiunque.

Il progetto è inoltre open-source e quindi estendibile da parte di altri utenti. Come delineato nel paragrafo 6.4, infatti, le possibili evoluzioni della tecnologia sviluppata sono numerose e si dipanano in diverse direzioni. InspectorPi potrebbe essere impiegato non solo per svolgere le funzioni descritte nell'ambito di questa tesi, come l'ispezione dei condotti di aerazione, ma potrebbe essere d'aiuto all'uomo per rilevare dati di diversa natura o affrontare situazioni ad alto rischio: la compattezza del sistema, la possibilità della telepresenza e la guida da dispositivo mobile, rendono il veicolo adatto a molteplici utilizzi. Come accennato, InspectorPi potrebbe per esempio servire come rilevatore termografico o di radioattività, fornendo così all'utilizzatore un supporto indispensabile nei casi di emergenza. Per rendere il prototipo adeguato ad affrontare questi compiti, basterebbe in effetti integrare la struttura hardware con appositi sensori ed effettuare alcune piccole modifiche al software. Questo perché, nella realizzazione del device, è stata prevista modularità: la struttura del dispositivo è duttile e facilmente riadattabile alle esigenze specifiche dei singoli utilizzatori.

In caso di necessità o qualora lo si ritenesse opportuno, in futuro il sistema potrebbe inoltre essere ulteriormente ottimizzato, utilizzando per esempio componenti hardware più complesse, in grado di implementare un numero maggiore di funzionalità.

Bibliografia e sitografia

Gershenfeld N., *When Things Start to Think*, Henry Holt & Co, 1999

Waher P., *Learning Internet of Things*, Packt Publishing, 2015

---

https://designmind.frogdesign.com/2014/08/Internet-things-six-key-characteristics/

https://developer.android.com/sdk/index.html

http://www.dse.uniba.it/Corsi/docenti/Coco/Lezione%205.pdf

https://www.eclipse.org/

https://www.gartner.com/

http://www.innovaway.it/iot-Internet-of-things-lInternet-delle-cose/

http://www.materious.com/#/projects/forecast/

http://www.nhs.uk/conditions/sick-building-syndrome/Pages/Introduction.asxp

http://mipi.org/specifications/camera-interface

http://www.ntaimpianti.it/impianti-rilevazione-gas-bologna.html

http://www.parrot.com/it/prodotti/minidrones/

http://pitom.eu/index.php/it/prodotti/piship-asv

http://www.projectems.it/ispezione-tetti/

http://www.theguardian.com/media-network/2015/mar/31/the-Internet-of-things-is-revolutionising-our-lives-but-standards-are-a-must

https://xively.com/