UNIVERSITÀ DEGLI STUDI DI UDINE

Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Corso di Laurea Triennale in Internet of Things, Big Data e Web

Tesi di Laurea

REALIZZAZIONE DI UN SOFTWARE PER UN ROBOT UMANOIDE AUTONOMO

Relatore:
Prof. IVAN SCAGNETTO

Laureando:
EMANUELE ALTAMURA

ANNO ACCADEMICO 2022-2023

Un po' a tutti,
un po' a nessuno.

Indice

0.1 Introduzione 1

1 Architettura del kernel **3**
1.0.1 Requisiti di sistema 3
1.0.2 Scelte implementative 3
1.1 Implementazione del kernel 5
1.1.1 Integrazione dello stream video 5
1.1.2 Controllo dei servo motori 5
1.1.3 Sintesi e riconoscimento del linguaggio naturale 7
1.2 Strato per l'implementazioni di applicazioni 9

2 Implementazione di applicazioni **11**
2.1 Wikipedia 11
2.2 Il gioco; sasso, carta, forbice 13
2.2.1 Nodo switch: struttura per la feature detection 15
2.3 Interazione Uomo-Robot 17
2.3.1 Architettura e implementazione 18

3 Conclusioni e sviluppi futuri **21**
3.1 Conclusioni 21
3.2 Sviluppi futuri e upgrade 22

Bibliografia **25**

0.1 Introduzione

Ambito Il mondo della robotica sta vivendo una fase di sviluppo e crescita notevole negli ultimi anni, con l'obiettivo di creare macchine sempre più avanzate e interattive con l'uomo. Lo sviluppo dell'IA,la diffusione della stampa 3D e la diffusione di progetti open source permettono la democratizzazione della cultura.

Stato dell'arte Negli ultimi anni sono stati realizzati vari progetti di robot umanoidi, da quelli più avanzati come Sofia[1] della Hanson Robotics, Spot e Atlas della Boston Dynamics[2], ad altri più democratici e accessibili come Poppy [3] o Inmoov[7].
Questi ultimi sono progetti open source stampabili in 3D.

Obiettivi Il lavoro svolto mira a creare un software Solid utilizzabile in questo caso sul robot InMoov, ma non si preclude l'utilizzo con qualche modifica all'utilizzo su altri robot come Poppy, che permetta l'implementazione di applicazioni sfruttando la moltitudine di librerie di Python. Questo linguaggio, che è uno dei linguaggi di programmazione più popolari degli ultimi anni, è stato scelto per la sua chiara sintassi e la leggibilità che lo rendono il linguaggio di codifica perfetto per i principianti. Un altro motivo della scelta di Python è la presenza di ROS, un framework fatto appositamente per lo sviluppo di Robot.

Risultati La tesi quindi descrive la realizzazione di un kernel per gestire le funzionalità a basso livello. Si permetterà di avere accesso allo stream video di una telecamera presente nel progetto Inmoov, di controllare i motori, di avere accesso allo speaker e ad un microfono.
Oltre al kernel verrà descritto come sono state implementate tre applicazioni:

1. Wikipedia.
2. Il gioco: sasso, carta, forbice.
3. Interazione uomo-robot.

Nel primo capitolo si discuterà dell'implementazione del kernel e dei protocolli per accedere alle sue funzioni. Il secondo capitolo invece descriverà le applicazioni realizzate, le loro strutture interne e come si interfacciano con il kernel.

Capitolo 1

Architettura del kernel

Per l'implementazione di un software aperto e modulare per il robot è necessaria la creazione di un kernel che permetta l'implementazione di applicazioni senza dover pensare al funzionamento dei motori, del Text-to-speech o dello Speech-to-text.

Nello specifico, sono stati ideati quattro nodi che implementano:

1. Gestione dei motori.
2. Text-to-Speech(TtS).
3. Speech-to-Text(StT).
4. Utilizzo della telecamera.

In questo capitolo vengono descritte le strutture di questi nodi e i modi in cui possono essere utilizzati.

1.0.1 Requisiti di sistema

Il software sviluppato ha rispettato i principi Solid per essere espandibile e decentralizzato. Il software viene eseguito su un LattePanda Delta 3 con Ubuntu 20.4 OS e ROS Noetic.

1.0.2 Scelte implementative

La scelta di utilizzare Robot Operating System, che permette la distribuzione dell'elaborazione su più macchine, è basata sul fatto che la capacità computazionale dei Single-Board Computer non è in grado di supportare analisi video importanti come quelle implementabili su un comune pc. Quindi

è stato possibile sfruttare le potenzialità del cloud computing. Il robot utilizzato, una versione di Inmoov, è stato stampato in 3D e alimentato con un alimentatore a 50W. Per controllare i servomotori vengono utilizzate delle board PCA9685 che possono controllare fino a 16 servomotori partendo solo da 2 jumper collegati al SBC. Dato che in complesso il robot ha 22 motori, si è scelto di utilizzare due schede PCA9685, anche perché il LattePanda utilizzato ha il pinout di un Arduino Leonardo che non permette di controllare direttamente questo numero di motori. Si è scelto di lasciare anche dei pin liberi per eventuali implementazioni di sensori futuri.
Componenti fondamentali e compresi nel progetto Inmoov sono una webcam, un microfono e un altoparlante.

Nel complesso è stata realizzata una struttura stratificata che permetterà di sviluppare applicazioni conoscendo solo i topic esterni al kernel.

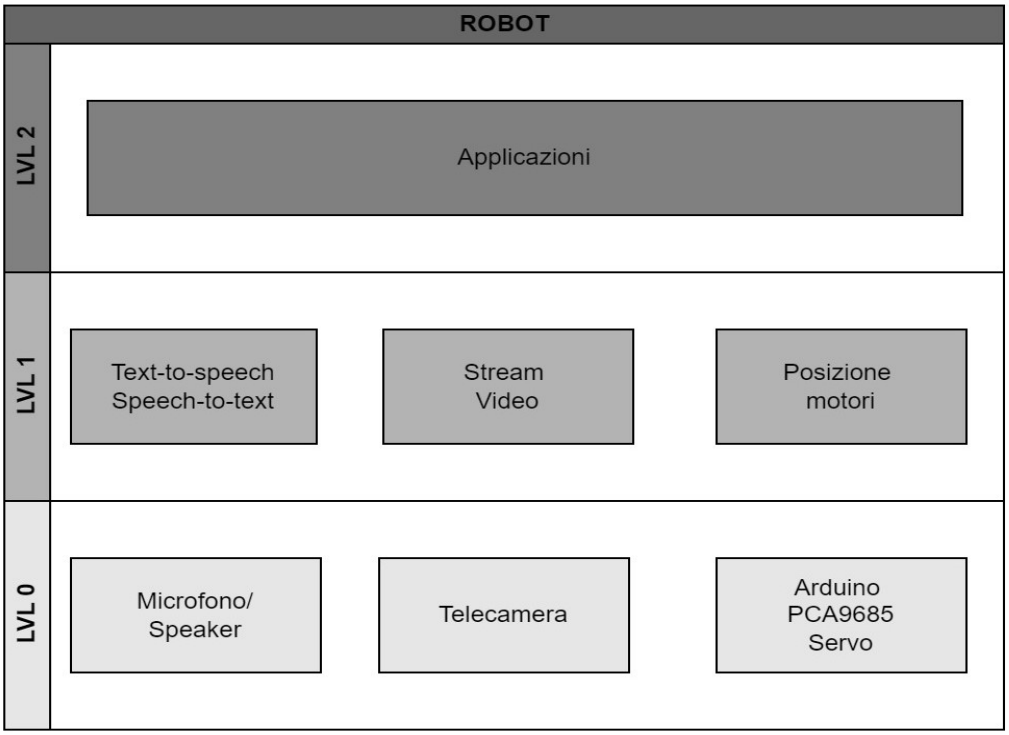

Figura 1.1: Divisione stratificata

1.1 Implementazione del kernel

Di seguito sono presentate le descrizioni dei singoli nodi che costituiscono il kernel.

1.1.1 Integrazione dello stream video

Lo scopo del nodo chiamato webcam stream è pubblicare sul topic $video\_frames$ lo stream della telecamera per permetterne l'utilizzo agli altri nodi, per esempio per estrarne delle features.

Le librerie che vengono utilizzate per questo sono cv_bridge[14] e opencv[5]. Dalla webcam ogni secondo vengono catturati 10 frames che prima di essere pubblicati vengono convertiti in $sensor\_msgs/Image$ che è un tipo di dato di ROS, ottimizzato per il sistema. Cvbridge permette infatti di codificare le immagini estratte dalla webcam con opencv[5] in Image e viceversa.

1.1.2 Controllo dei servo motori

I servo motori sono comandati dalle schede PCA9685 collegate all'Arduino Leonardo integrato nella scheda madre. In questa versione è necessario muovere solo i motori nella mano quindi si è creato il topic $hand\_pos$. Nel topic vengono pubblicati $UInt16MultiArray$ che sono vettori di interi ordinati contenenti l'angolo per ogni servo.

Arduino Come schematizzato in figura 1.2 le schede PCA9685 controllano i servo e sono collegate in serie e comunicano con Arduino tramite il protocollo I2C.

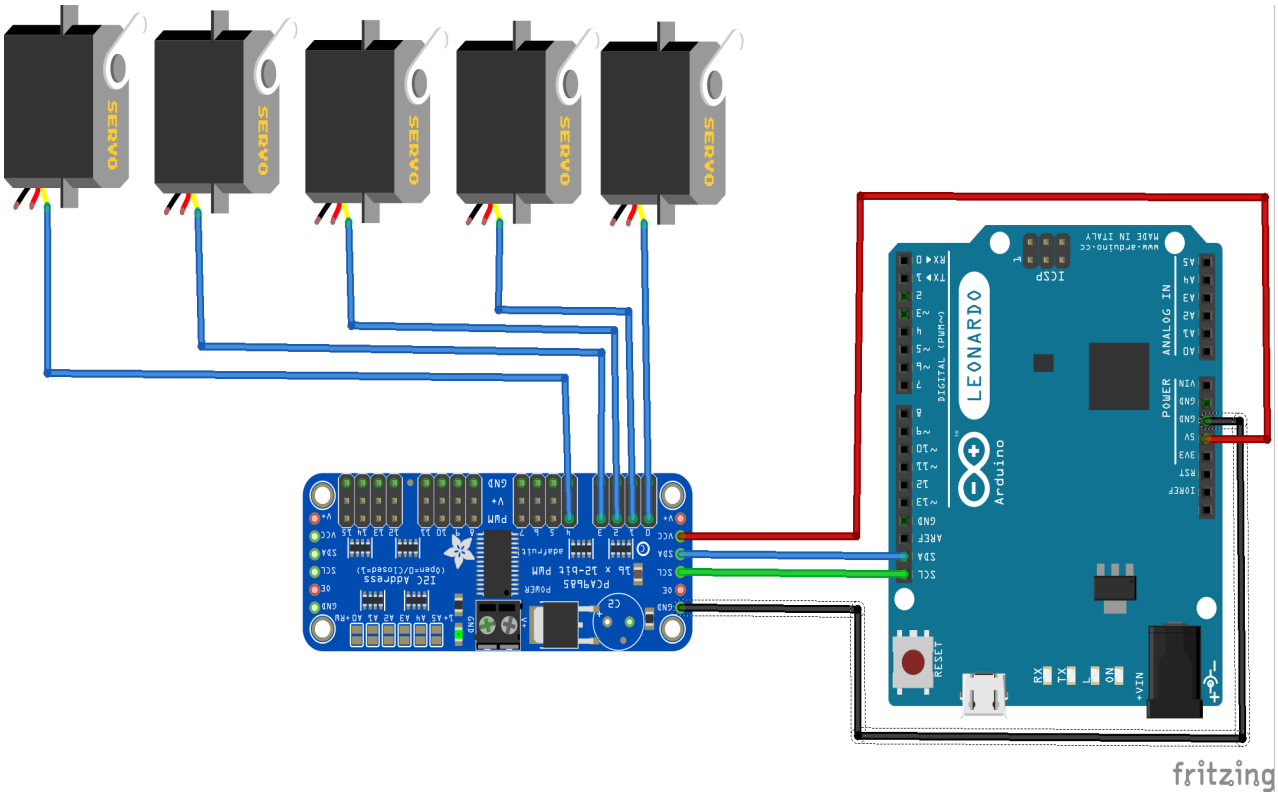

Figura 1.2: Schema dei collegamenti

1.1.3 Sintesi e riconoscimento del linguaggio naturale

Per l'interazione audio è stato scelto di unire in un nodo le funzionalità di text to speech e speech to text per questioni di sincronizzazione. Essendo necessario l'ascolto solo in momenti precisi sono state implementate due funzioni principali:

1. listen()
2. say()
3. ask()
4. ask_and_confirm()

Speech-to-Text Per la funzionalità è stata utilizzata la libreria speech_recognition. La funzione listen restituisce, quando chiamata, il testo di ciò che ha ascoltato in Italiano. [15]

Text-to-Speech Le librerie utilizzate sono gTTS [10] e playsound [11]. La traduzione da testo ad audio è affidata a gTTS mentre la riproduzione dell'audio viene delegata a playsound. Dato che la riproduzione continua e di audio lunghi non avveniva sempre correttamente è stata implementata una funzione analoga utilizzando la libreria pyttsx3 [12] che però ha una voce più metallica. Quando sul topic *text_to_speech* viene pubblicata una stringa, questa viene passata alla funzione say.

Sincronizzazione audio

Entrambe le funzioni say e listen sono state utilizzate nella funzione ask. Questa funzione, callback del topic *question*, fa dire al robot il testo della domanda e poi, quando presente, pubblica sul topic *answer* il testo della risposta sotto forma di $std\_msgsString$.

Nel caso in cui servisse chiedere conferma della risposta si utilizza la funzione ask_and_confirm che, una volta ricevuta la risposta, la ripete chiedendo se è stata compresa corretamente la risposta. La risposta viene pubblicata sempre sul topic *answer*. Per chiamare questa funzione bisogna pubblicare sul topic $question\_confirm$.

Attivazione applicazioni e indicizzazione risposte audio La gestione delle risposte audio viene affidata al nodo command_dispatcher. Questo nodo, in base a ciò che viene pubblicato sul topic *answer*, si occuperà di indirizzare le risposte ai nodi appropriati.

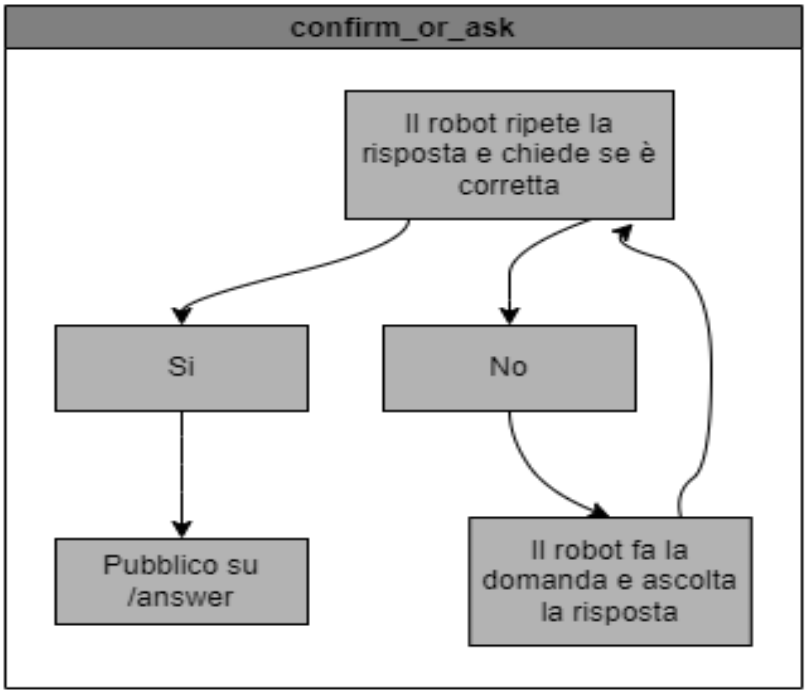

Figura 1.3: Schema di ask_and_confirm

1.2 Strato per l'implementazioni di applicazioni

Con questi nodi si possono implementare molte altre funzionalità. Le altre applicazioni dovranno quindi comunicare con i quattro topic:

- video_stream
- hand_pos
- question
- question_confirm

Ciò rende possibile realizzare applicazioni indipendenti dal tipo di robot utilizzato. Ad esempio, se si cambia il tipo di motori, bastera modificare il nodo del microcontrollore.

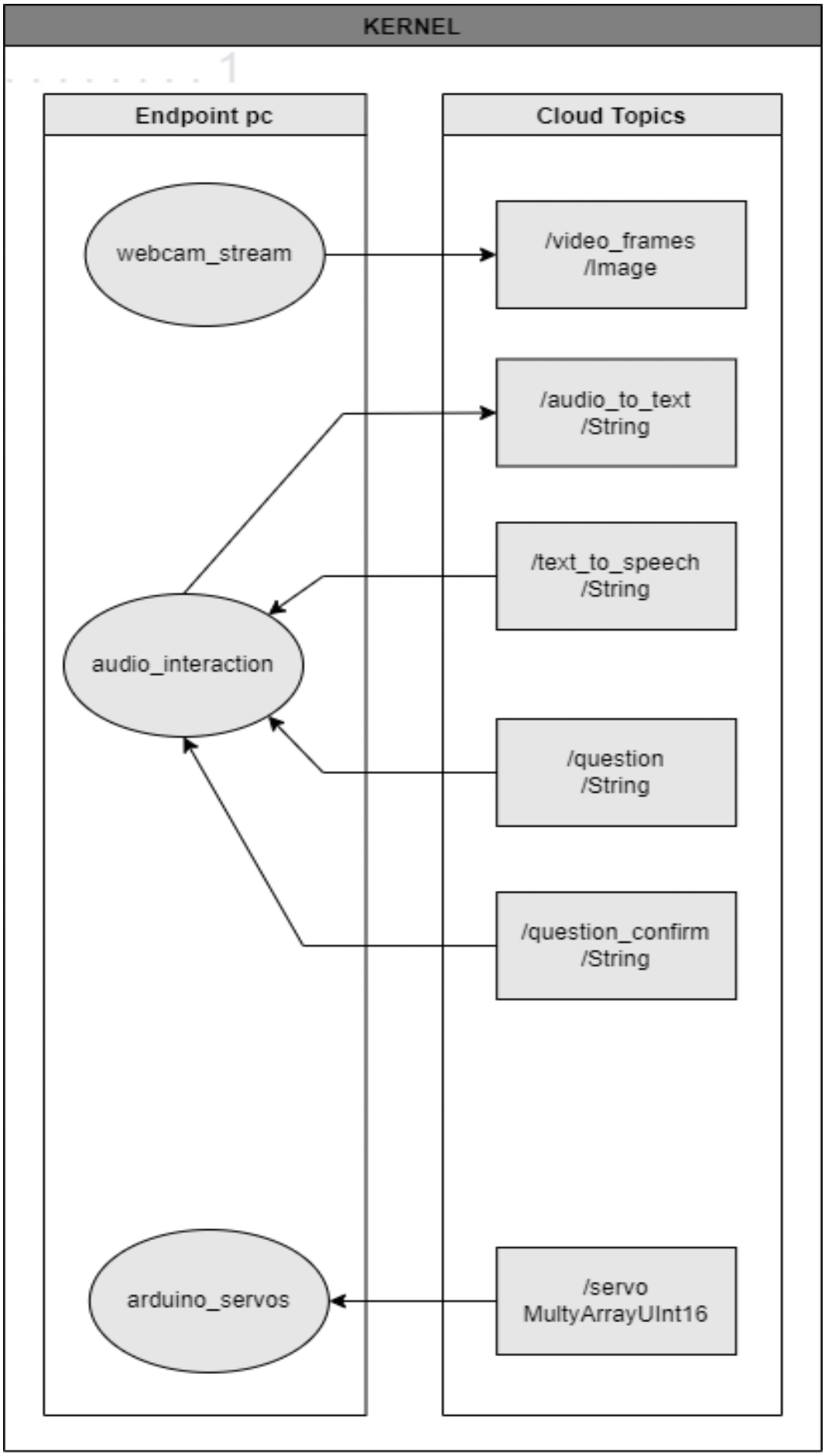

Figura 1.4: Schema kernel

Capitolo 2

Implementazione di applicazioni

Date queste premesse verrà mostrato ora come sono state implementate applicazioni basate sui nodi precedenti. La prima applicazione sviluppata permette di ricevere informazioni da Wikipedia e comunicarle all'utente. Attraverso la seconda applicazione si può giocare con il robot a sasso, carta, forbice, facendogli riconoscere la posizione della mano del giocatore e valutare il risultato del gioco.

L'ultima applicazione implementata permette al robot di imparare il nome di una persona e di ricordarsene.

2.1 Wikipedia

La prima app permette all'utente di chiedere vocalmente una ricerca su Wikipedia ed il robot risponderà a voce con il sommario della pagina relativa alla ricerca richiesta.

Ciò è permesso grazie alla libreria Wikipedia[9] di Python. Per questioni di facilità di programmazione al robot bisognerà chiedere "ricerca su wikipedia <richiesta>". Questa frase verrà elaborata dal nodo command_dispatcher che invierà al topic *wikipedia* solo l'oggetto della richiesta. Nel nodo wikipedia le informazioni sono recuperate tramite la funzione:

```
1 res = wikipedia.summary(<richiesta>, sentences = 1)
```

Metodo summary della libreria Wikipedia

Il risultato verrà poi inviato sul topic *text_to_speech*.

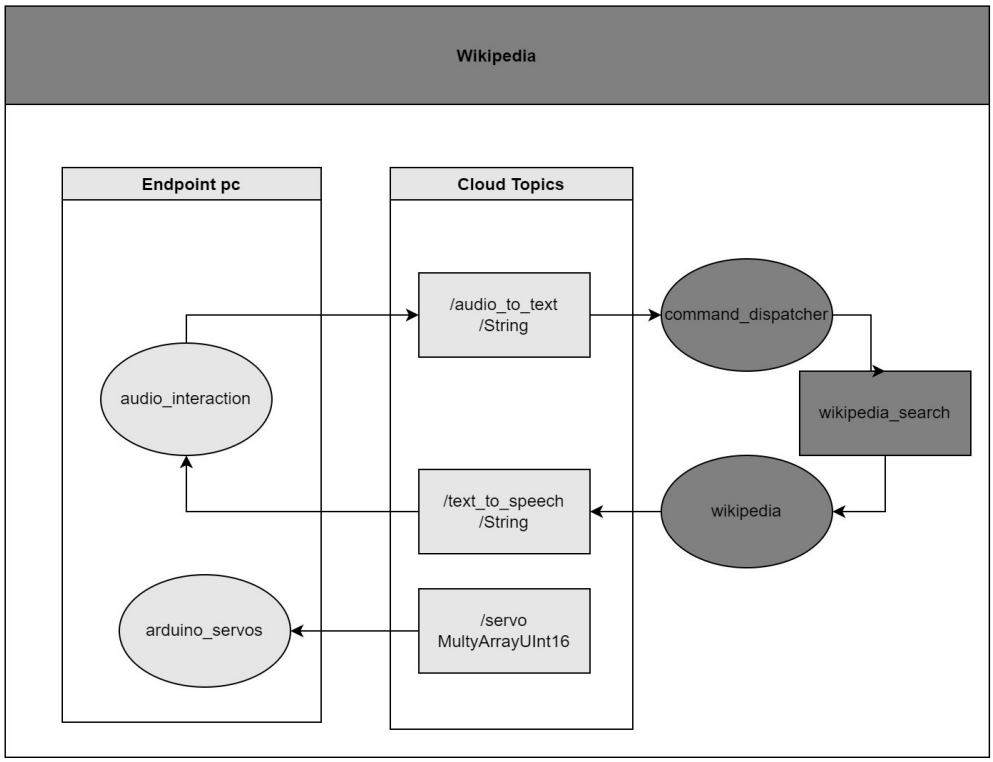

Figura 2.1: Architettura dell'app wikipedia

2.2 Il gioco; sasso, carta, forbice

Questo sistema, più sofisticato rispetto al precedente, è composto da due nodi:

- il primo per la gestione del gioco(rsp_game),
- il secondo per il riconoscimento della mano($hand\_recognition$).

Algoritmo di funzionamento La sequenza di gioco si attiva inviando una stringa qualsiasi sul topic $rsp\_activator$. Ciò verrà fatto dal nodo *command_dispatcher* ascoltando le parole dell'utente.

Il nodo r*sp_game* inizia quindi la sequenza di gioco e attiva il riconoscimento della mano nel nodo *hand_recognition* inviando sul topic *hand_manager* un valore booleano $True$.

A questo punto il nodo *hand_recognition* inizia a riconoscere la posizione della mano nei frame presenti sul topic $video\_frames$ e pubblica sul topic $fingers\_info$ le informazioni su quali dita siano alzate e quali no.

Il nodo rsp_game avvia il countdown e alla fine di esso prende l'ultimo valore ricevuto sul topic $fingers\_info$ e lo traduce in stringa sasso, carta o forbice. Allo stesso tempo invia $False$ su *hand_manager* così che il nodo *hand_recognition* non analizzi più lo stream video.

Una volta fatto ciò invia sul topic *hand_pos* un array di UInt16 che sono una posizione casuale tra sasso, carta, forbice, che dovrà prendere la mano del robot .

Dopo il confronto tra la scelta casuale del robot e la scelta rilevata dell'utente il robot annuncia il vincitore e chiede se si vuole giocare di nuovo.
Es. " Io ho scelto carta, tu forbici . Hai vinto! Vuoi giocare ancora?"

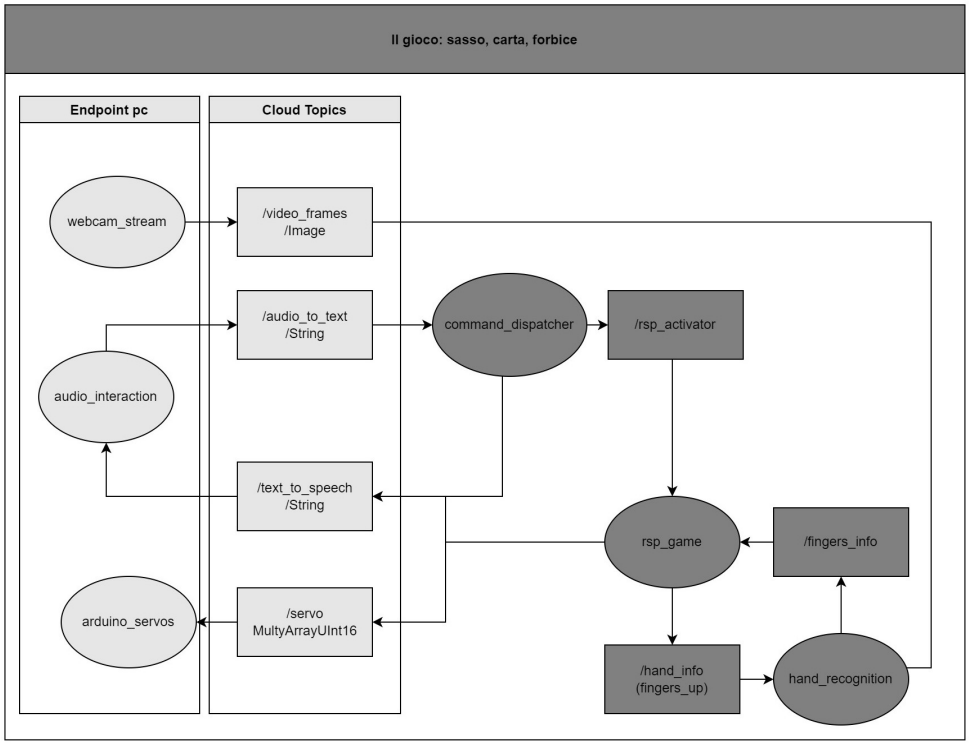

Figura 2.2: Architettura del gioco

Riconoscimento della mano Per l'estrazione delle infromazioni viene utilizzato il modulo HandTrackingModule di cvzone[6].

```
1 from cvzone.HandTrackingModule import HandDetector
```

Importazione modulo HandDetector

Per capire quale scelta è stata fatta dal utente si controlla quali dita siano aperte e quali chiuse. Le dita alzate vengono codificatecon un 1 e quelle chiuse con uno 0 su un array ordinato. Questo array viene confrontato con array predefiniti ([0,0,0,0,0] per sasso ad esempio) per poi trasformare quest'informazione in stringa che verrà confrontata con la scelta del robot.

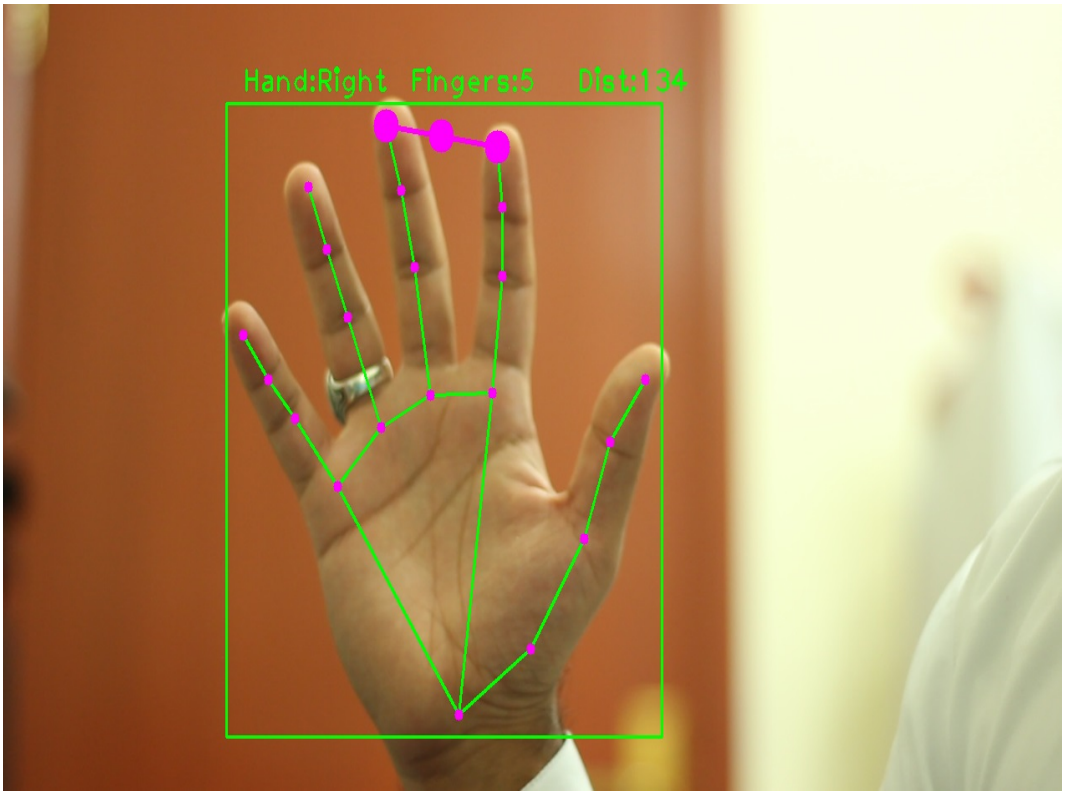

Figura 2.3: Riconoscimento della mano e della posizione

2.2.1 Nodo switch: struttura per la feature detection

Il riconoscimento di oggetti, in questo caso di una mano e di quante dita sono alzate, che richiede un utilizzo importante di risorse per il calcolatore, viene attivato solo quando necessario. Per ciò il nodo è stato strutturato in maniera da poter attivare o disattivare il riconoscimento della feature nell'immagine.

Questa funzione è stata implementata grazie all'utilizzo di due subscriber nello stesso nodo che gestisce la feature detection:

1. un subscriber al topic $feature\_manager$;
2. un subscriber al topic $video\_frames$ che attua effettivamente la feature detection.

Il primo subscriber, sempre attivo, chiama una callback alla quale viene passato un valore booleano letto dal topic $feature\_manager$: in base a questo valore verrà inizializzato, o disattivato quando presente, il subscriber al topic $video\_frames$.
Questo è necessario in quanto su $video\_frames$ vengono pubblicati ogni 10 Hz i frame, se non fosse possibile attivare o disattivare la feature detection, il nodo occuperebbe molte risorse della macchina. Questa struttura è utile in quanto il sistema finale dovrà gestire molte funzioni di feature detection ed altre che richiedono capacità computazionali.

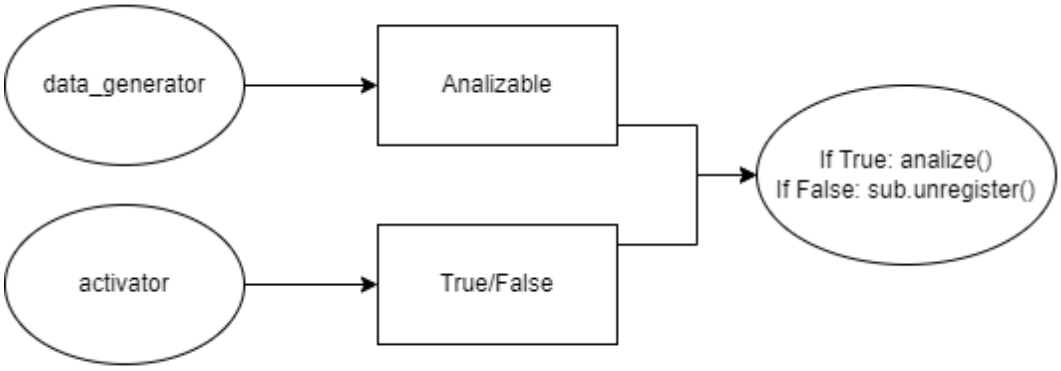

Figura 2.4: Schema della struttura switch

2.3 Interazione Uomo-Robot

Il robot, quando un umano entra nel suo campo visivo, se il volto non è salvato nel database, chiede il nome dell'utente e, dopo aver chiesto conferma, salva una codifica del volto e il nome. Quando invece la persona è stata conosciuta viene salutata con il proprio nome e viene chiesto quale app avviare.

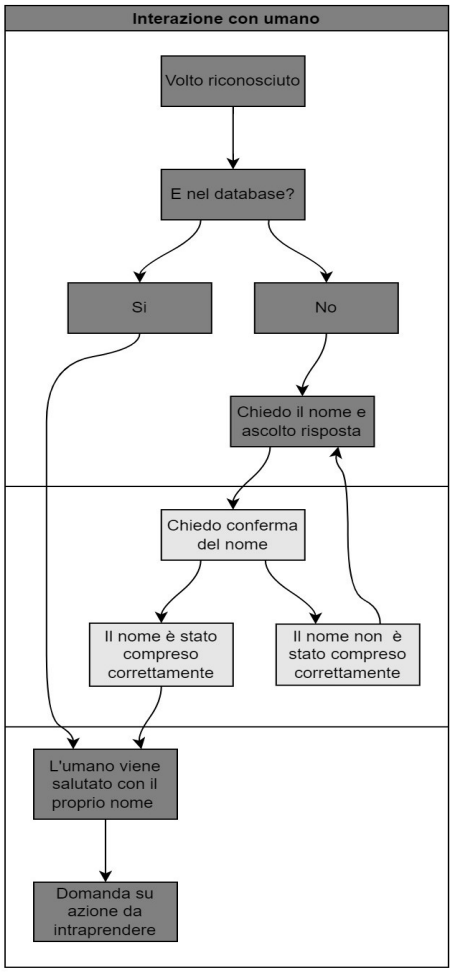

Figura 2.5: Algoritmo di interazione

2.3.1 Architettura e implementazione

Di seguito verranno giustificate le scelte implementative al fine di mantenere il sistema efficiente e semplice da gestire.

Nodo switch Il riconoscimento dei volti umani richiede risorse e, come nel caso dell'app gioco, si è scelto di utilizzare una struttura nodo switch come in Figura 2.4 per mantenere il carico della cpu basso. In questo caso il topic per attivare la feature-detection è *human_manager* che inizializzerà l'iscrizione del nodo *human_recognition* al topic *video_frames*.

Avvio app Si è scelto di attivare l'app con la pubblicazione del valore *True* sul topic *PIR* in quanto in futuro verrà aggiunto un sensore di presenza nell'Arduino che invierà questo valore quando rilevata una persona. Per il momento l'attivazione dell'app avviene inviando il valore manualmente.

Codifica volti e database Il database è costituito da un file contenente una lista di nomi degli utenti conosciuti chiamato *known_human_names.list* e la codifica dei volti invece viene salvata in un file per ogni utente chiamato $< nome\_utente > .json$. Questo è stato fatto per semplificare, in quanto il robot leggerà la lista dei nomi conosciuti e per ogni nome caricherà la codifica del volto, così da ricreare le strutture dati utilizzati nella documentazione della libreria face_recognition[13], utilizzata per riconoscere e confrontare i volti. Il salvataggio e la lettura di questi file è permessa dalla libreria pickle[16].

Architettura

La gestione dell'interazione è stata delegata a due nodi:

- il primo per la gestione del'interazione(*human_interaction*),
- il secondo per il riconoscimento dei volti(*human_recognition*).

Prima dell'avvio dell'app, il nodo *human_recognition* dal database carica i nomi e le relative codifiche dei volti delle persone conosciute attraverso la funzione *get_known_human*(). Una volta avviata l'app il nodo *human_interaction* attiva la feature detection. Quando una persona entra nel campo visivo, il suo volto viene confrontato con i volti caricati in memoria attraverso la funzione:

```
# known_face_encoding : lista di volti conosciuti
# face_encoding : volto rilevato dalla telecamera
face_recognition.compare_faces(known_face_encodings,
    face_encoding)
```

Confronto codifiche volti

Se c'è un riscontro viene inviata una stringa di saluto con il nome della persona sul topic *text_to_speech*. Quando invece non c'è riscontro viene chiamata la funzione $save\_face\_and\_get\_name()$ che invia sul topic $question\_confirm$ una domanda richiedendo il nome. Una volta ricevuta la risposta sul topic *answer_name* la fuzione aggiunge alla lista dei nomi conosciuti il nuovo nome e sovrascrive il file $known\_human\_names.list$ e salverà il file $< nome > .json$. Si è scelto di utilizzare la domanda con conferma in quanto i suoni brevi, come alcuni nomi, sono difficili da tradurre in testo.

Modifiche necessarie Per avere la risposta del nome sul topic *answer_name*, e non sul topic *answer* che sarebbe stato letto solo dal *command_dispatcher*, è stata modificata la funzione $ask\_and\_confirm()$ in modo da poter specificare il topic dove dovrà essere pubblicata la risposta. Ciò avviene anteponendo alla domanda il nome del topic diviso dal carattere ':'. Il topic verrà anteposto anche alla risposta pubblicata su *answer*. Sarà poi compito del nodo *command_dispatcher*, che deve essere modificato per ogni applicazione che si aggiunge, re-indirizzare la risposta al topic *answer_name*.

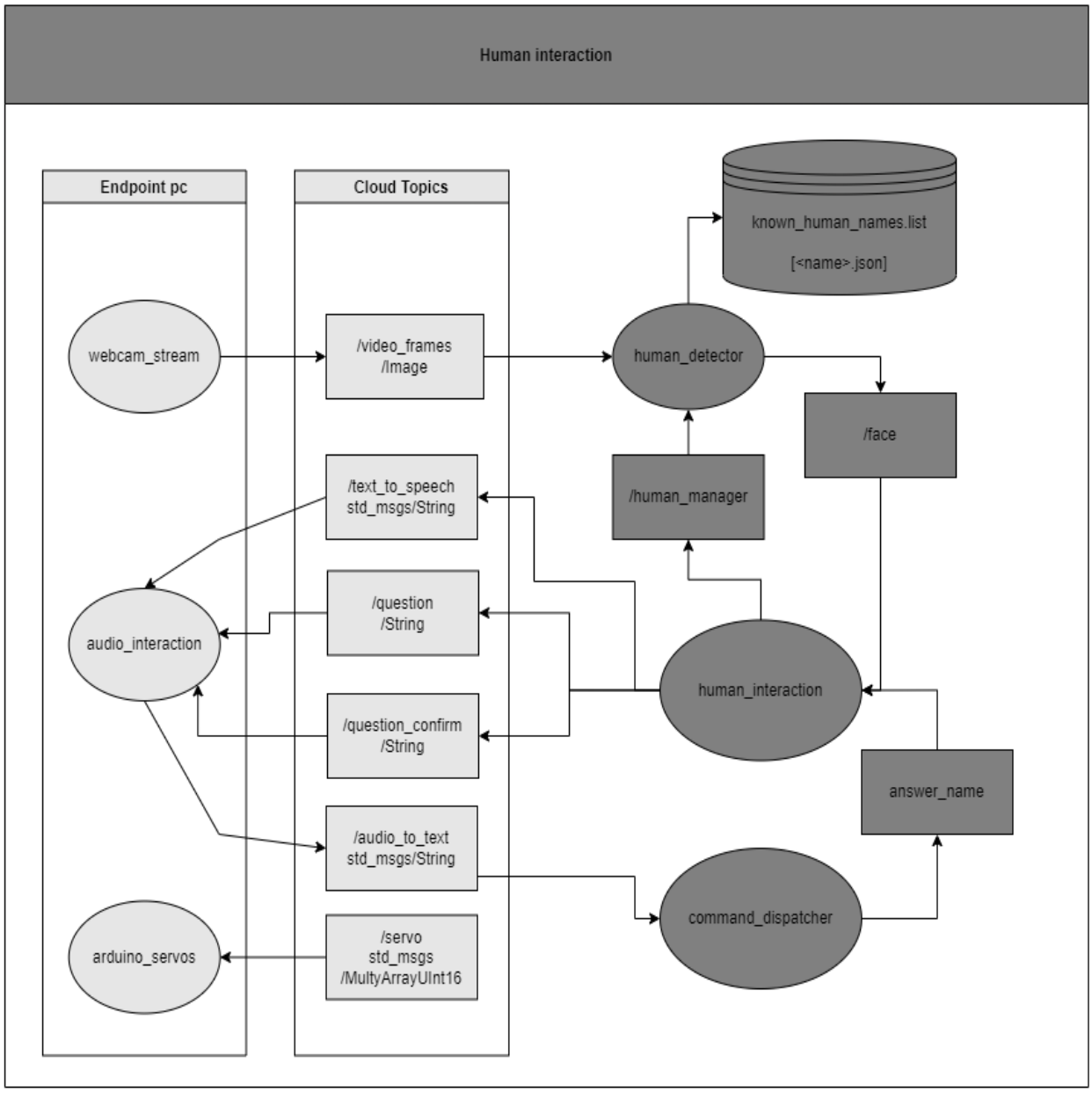

Figura 2.6: Architettura interazione Uomo-Robot

Capitolo 3

Conclusioni e sviluppi futuri

3.1 Conclusioni

Si è mostrato come è stato realizzato il kernel e alcune applicazioni che poggiano su di esso come in Figura 3.1.
L'implementazione garantisce un buon funzionamento anche se sono state riscontrate alcune criticità:

Audio brevi e Speech-to-Text Una delle prime criticità riscontrate è stata l'impossibilità del robot di riconoscere correttamente la risposta. Questo perché la maggior parte delle librerie ha difficoltà a riconoscere parole brevi e distinguere dai semplici rumori. Quindi per risolvere questo problema, presente ad esempio quando bisogna confermare una risposta, si devono utilizzare frasi lunghe contenenti la parola "sì", ma è stato inserito come parola di conferma anche "corretto" e "giusto".

Sincronizzazione audio e feedback uditivi Data la difficoltà nel lasciare il robot in ascolto continuo, a causa di limiti delle funzioni e dell'architettura, si è scelto di far ascoltare il robot solo in determinati momenti dell'interazione. Per comunicare i momenti in cui si può parlare con il robot sono stati introdotti dei feedback uditivi.

Feature-detection: HandTrackingModule Il modulo utilizzato riesce a riconoscere correttamente la posizione delle dita di una mano solo quando la mano è alzata. In caso contrario non viene garantito il corretto riconoscimento della posizione scelta.

Arduino La scelta di non implementare subito il sensore di presenza (PIR) è dettata dai limiti fisici dell'Arduino Leonardo integrato nella scheda. Questo è causato dal peso di gestione di un nodo ros presente direttamente nella scheda. Una delle soluzioni pensate sarebbe di implementare un nodo esterno all'Arduino che comunichi in seriale con il microprocessore così da avere più memoria libera per gestire più sensori.

Numero di persone con cui può interagire Durante la realizzazione si è pensato a far interagire il robot solo con una persona alla volta. Ciò ha portato il robot ad iniziare più sequenze di interazione contemporanee se sono presenti più persone. Questo però non permette una corretta interazione in quanto lo porta a pronunciare diverse frasi contemporaneamente, ascoltare e parlare allo stesso tempo e si perde coerenza del discorso.

3.2 Sviluppi futuri e upgrade

Si è pensato ad alcune modifiche oltre a quelle già citate nel corso della descrizione del lavoro fatto.

Cloud computing Nel caso in cui si utilizzasse un single-board computer meno performante o se si volessero utilizzare applicazioni che richiedono più risorse sarebbe utile sfruttare la possibilità offerta da ROS di distribuire i nodi su più macchine. Un tipo di applicazione che potrebbe sfruttare questa possibilità è il gioco sasso carta forbice ad esempio, se si volesse aggiungere il riconoscimento delle emozioni nell'utente a fine gioco.

Database Si potrebbe utilizzare un database esterno per salvare i volti degli utenti o per recuperare degli audio pronti per sostituire alcune frasi per alleggerire l'utilizzo di CPU per il Text-to-Speech.

AI e capacità di linguaggio Una delle più importanti migliorie da implementare è l'utilizzo di un AI conversazionale per poter avere discrosi completi e naturali con il robot. Ciò andrebbe sicuramente distribuito su più macchine attraverso cloud computing.

Il progetto comunque verrà reso open source e quindi disponibile su git-hub. L'open source rappresenta un'eccezionale opportunità per contribuire al progresso della tecnologia e della società, creando soluzioni innovative e accessibili a tutti.

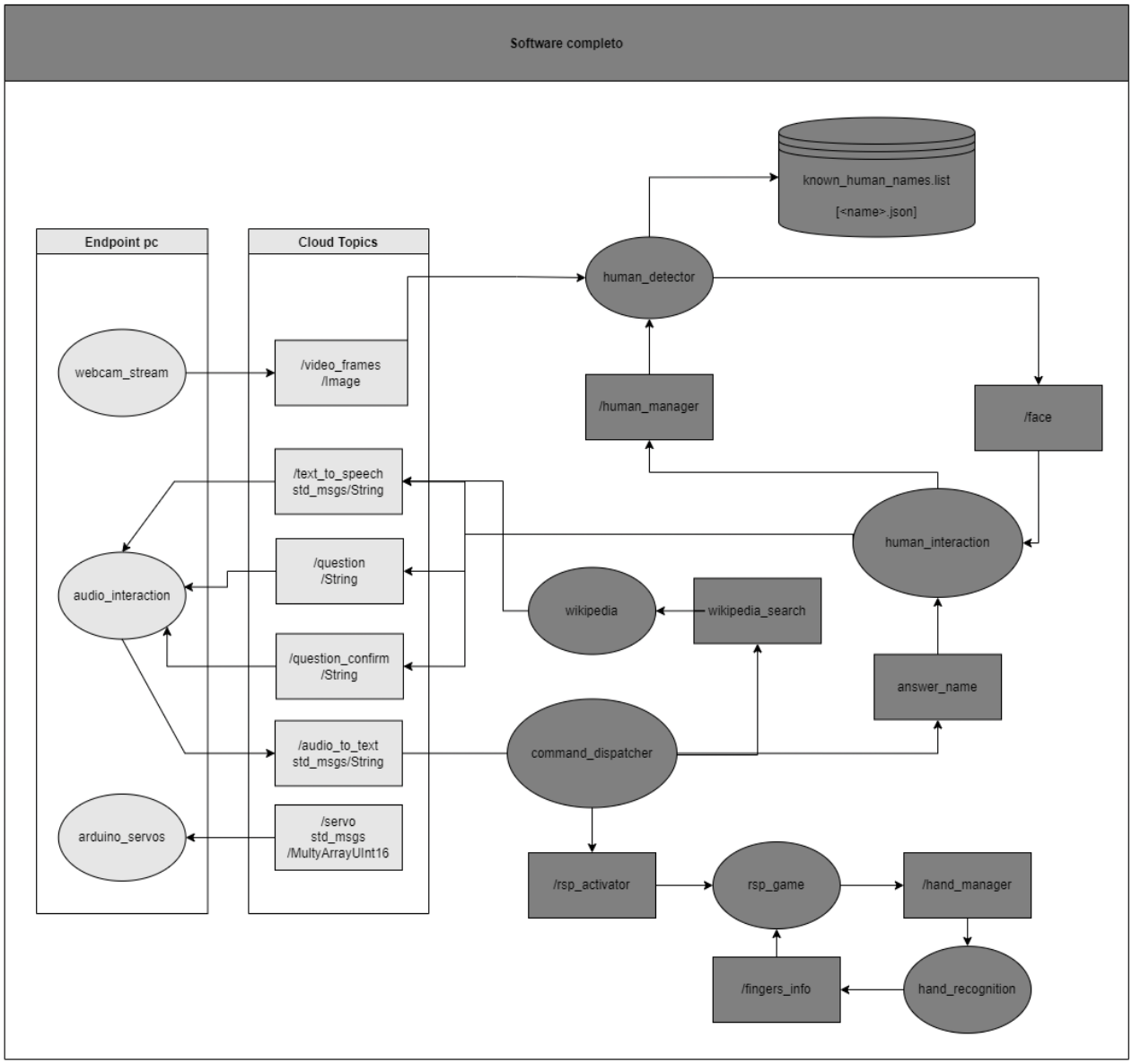

Figura 3.1: Architettura finale del software

Bibliografia

[1] Sofia -
https://www.hansonrobotics.com/sophia/

[2] Boston Dynamics -
https://www.bostondynamics.com/

[3] Poppy -
https://www.poppy-project.org/en/

[4] ROS - *The Robot Operating System (ROS)*,
www.ros.org.

[5] OpenCV - *Open computer vision library*,
www.opencv.org.

[6] Computer vision package - *Open computer vision library*,
github.com/cvzone/cvzone.

[7] InMoov - *the 3D printed open-source robot* ,
inmoov.fr.

[8] Unsubscribing from ROS Topic - *ROS Answer*,
answers.ros.org/question/239944/unsubscribing-from-ros-topic-python/.

[9] The Wikipedia library - *Wikipedia API for Python*,
pypi.org/project/wikipedia/.

[10] Google Text-to-Speech - *Python library and CLI tool to interface with Google Translate's text-to-speech API*,
gtts.readthedocs.io/en/latest/index.html.

[11] Playsound library -
https://pypi.org/project/playsound/

[12] Python Text-to-Speech library - pyttsx3.readthedocs.io/en/latest/.

[13] Python face recognition library - *Recognize faces from Python or from the command line*, pypi.org/project/face-recognition/, www.mygreatlearning.com/blog/face-recognition/.

[14] Working With ROS and OpenCV - automaticaddison, 2020, automaticaddison.com/working-with-ros-and-opencv-in-ros-noetic/.

[15] The Ultimate Guide To Speech Recognition With Python by David Amos, realpython.com/python-speech-recognition/ , pypi.org/project/SpeechRecognition/.

[16] The Python pickle Module: How to Persist Objects in Python by Davide Mastromatteo , realpython.com/python-pickle-module/.