Alma Mater Studiorum · Università di Bologna

DIPARTIMENTO DI INFORMATICA
Corso di Laurea in Ingegneria e Scienze Informatiche

Progetto e realizzazione di un dispositivo per la sostituzione sensoriale

Elaborato in Visione Artificiale

Relatore:
Chiar.mo Prof.
Raffaele Cappelli

Presentata da:
Luca Rossi

Seconda Sessione
2019/2020

A mio nonno, che quando con l'età non riusciva a ricordarsi di me mi chiedeva dove studiavo, e quando gli dicevo "Cesena", se ne usciva con "Anche Luca studia lì!".
Mi manchi tanto.

Instead of reality being passively recorded by the brain, it is actively constructed by it

David Eagleman

Indice

Elenco delle figure

Elenco dei listati

Capitolo 1

Introduzione

L'idea di questo progetto è nata dopo aver visto il TED talk del neuroscienziato David Eagleman riguardo alla sostituzione sensoriale[3]. Nel talk ha infatti spiegato come il cervello umano, come anche quello animale, sia molto bravo ad interpretare segnali provenienti dall'esterno; solitamente usiamo questa abilità per ricavare informazioni, per spostarci, orientarci, e sopravvivere, creando una rappresentazione astratta della realtà, diversa per ogni specie animale e in generale per ogni essere vivente. Ci si potrebbe spostare sul piano filosofico con domande come "Il verde che vedo io è lo stesso verde che vede un'altra persona?" ma non è questo che intendo per rappresentazione diversa, bensì la vera e propria sfera di percezione che ci creiamo. Noi uomini abbiamo una vista molto sviluppata, percepiamo tre colori primari e tendiamo ad usarla come metodo principale per estrapolare informazioni; in natura ad esempio questa abilità ci ha sicuramente aiutati nella ricerca di cibo e nel riconoscere e differenziare piante, erbe, fiori e frutti, e, ora più che mai, in una società organizzata e progredita come la nostra la vista è sicuramente molto importante, basti pensare che se si è affamati è sufficiente cercare la scritta "ristorante" per sfamarci o come prima di attraversare la strada si guardi sempre da entrambi i lati per evitare pericoli. La vista dei cani invece non è minimamente comparabile alla nostra, percepiscono solo due colori, tuttavia il loro olfatto è estremamente sviluppato. Le due sfere di rappresentazione che si creano sono estremamente diverse

anche se nessuna delle due è migliore dell'altra, un cane riuscirebbe a capire, annusando a terra, quanti altri cani ci sono nel territorio, quando sono passati l'ultima volta e se ci sono potenziali partner per l'accoppiamento, ma il rosso e il verde per loro sono lo stesso colore.

Sta di fatto che qualsiasi specie percepisce un minuscolo spicchio di quello che è effettivamente la realtà, e alcuni individui, che nascono con un qualche tipo di invalidità ne catturano una fetta ancora più piccola, talvolta rendendo difficile una vita indipendente.

Paul Bach-y-Rita[8] è stato un importante neuroscienziato americano che ha studiato molto il fenomeno della neuroplasticità, i suoi lavori più importanti riguardano infatti la cosiddetta "sostituzione sensoriale" come metodo per curare pazienti con problemi neurologici. È importante specificare che tramite questa tecnica si può anche ampliare la nostra bolla di percezione, rendendo visibile la luce ultravioletta o infrarossa ad esempio. Tratteremo in maniera più approfondita la sostituzione sensoriale nel capitolo 2, dato che è alla base del progetto.

La mia idea era quella di creare un dispositivo indossabile, leggero, economico, discreto e funzionante in ogni condizione ambientale (similmente all'occhio umano) che permettesse a non vedenti e ipovedenti di avere una rappresentazione della realtà mediante una sistema di motori di vibrazione o un display tattile. A questo proposito mi serviva un microcontrollore per computare le varie operazioni e un metodo per scansionare l'ambiente circostante, ricavando le distanze dagli oggetti, così da poterle trasmettere all'utilizzatore. Dopo aver soddisfatto la maggior parte degli obiettivi prefissati, si è riusciti anche ad aggiungere una funzione di descrizione audio dell'ambiente. L'analisi dell'immagine e l'estrapolazione delle distanze si è rivelata computazionalmente molto pesante, infatti, l'utilizzo del sistema come stand-alone non è reattivo come dovrebbe e per questo motivo si è spostata la parte adibita a computazione su un server, raggiungibile via internet dal microcontrollore. Ai fini di una migliore visualizzazione dei risultati e per motivi di costi si è deciso di usare una breadbord con dei led anziché un display tattile, questo cambiamento non dovrebbe essere rilevante, infatti si potrebbero molto facilmente sostituire

i led con dei motori di vibrazione mediante l'utilizzo di transistor collegati ad un'alimentazione appropriata, al microcontrollore e al motore stesso. In figura 1.1 propongo il risultato finale.

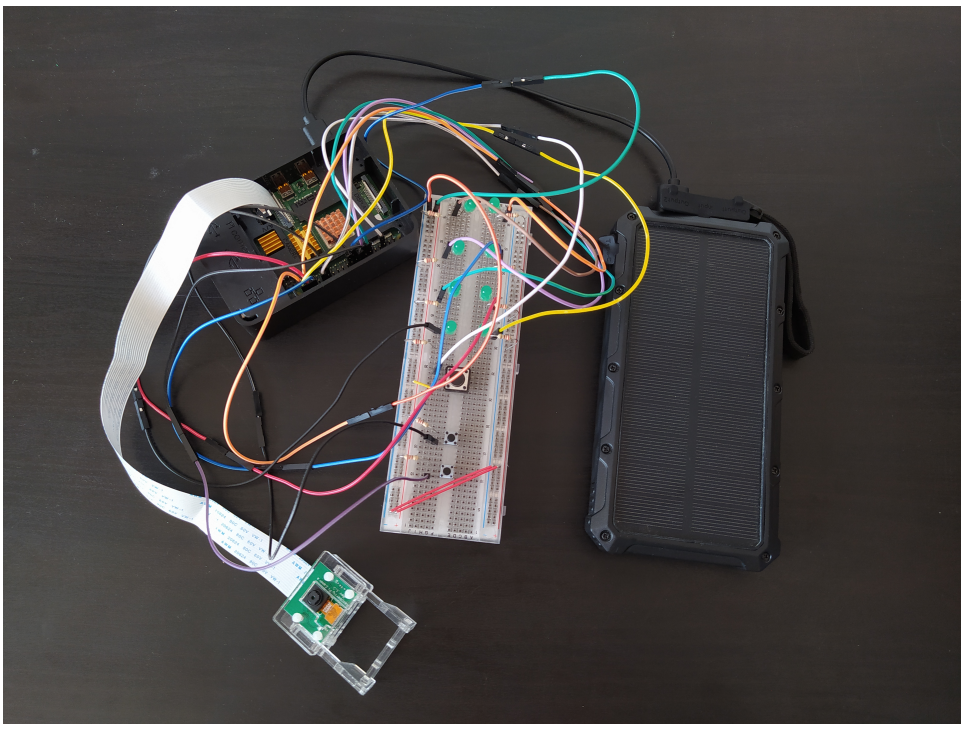

Figura 1.1: Risultato finale

Capitolo 2

Sostituzione sensoriale

2.1 Introduzione

In generale, la cosidetta "sostituzione sensoriale" trasforma stimoli caratteristici di una modalità sensoriale (ad esmpio la vista) in stimoli di un'altra modalità sensoriale (ad esempio il tatto).

Un sistema di sostutuzione sensoriale può essere decomposto in tre componenti distinte. Un *sensore* permette la conversione di un certo tipo di energia (luminosa, sonora, meccanica o di altro genere) in un segnale che può essere letto da un *sistema elettronico*, responsabile del funzionamento di uno *stimolatore*. La stimolazione viene generalmente indirizzata a recettori cellulari di un organo sensoriale, che convertono l'energia elettrica proveniente dal sistema elettronico in suono (nel caso di disositivi visivi-acustici), in energia meccanica (nel caso di dispositivi visivi-tattili) e così via.

Un sistema così organizzato permette agli utilizzatori di non dover subire alcun intervento chirurgico, e questo porta due grandi vantaggi. Il primo è che si eliminano alla base i rischi legati all'intervento e il tempo di recupero ad esso associato. Il secondo è il costo, non paragonabile a quello di un intervento fatto da un'equipe di medici, sommato al costo delle attrezzature e della sala operatoria. Questa caratteristica li rende ideali per paesi in via di sviluppo o per persone che non possono permettersi questi interventi

dove la sanità è privata.

Uno dei più grandi contributi in questo ambito è sicuramente quello del neuroscienziato Paul Bach-y-Rita, che ha iniziato gli studi di questa materia sviluppando principalmente sistemi per pazienti affetti da cecità congenita.

Il neuroscienziato David Eagleman spiega la sostituzione sensoriale in maniera molto affascinante nel TED Talk del 2015[3]. Lui sostiene che il nostro cervello sia un *general purpose computer*, ovvero, un computer adatto a molteplici utilizzi, bloccato nella scatola cranica, in cui non arrivano altro che segnali elettrochimici. Ogni sensazione che ci arriva e che crea quella che è la nostra personale rappresentazione della realtà è frutto di veri e propri sensori plug-n-play, che leggono valori dall'esterno e li inviano così come sono al cervello, che ha l'incarico di decifrarli e interpretarli. Finora per ristabilire la vista o l'udito le procedure erano sempre le stesse: si impiantava una camera o un microfono, che raccoglieva segnali e li trasmetteva direttamente ai nervi. Agli inizi non si sapeva quanto potesse funzionare un procedimento del genere, infatti questi erano dispositivi digitali, che inviavano stream di dati al cervello, abituato a riceverne un altro tipo ben preciso. Il risultato è stato però sorprendente, le persone reduci dall'intervento riuscivano tranquillamente ad interpretare questo nuovo "dialetto di Silicon Valley" senza particolari problemi[4].

L'idea dietro la sostituzione sensoriale era quella di addestrare parti del corpo inutilizzate ad interpretare i segnali inviati dallo stimolatore. Viene spesso usata la stimolazione tattile perché la pelle è la parte più estesa del nostro corpo e solitamente non viene utilizzata molto, per questo motivo i primi esperimenti del dottor Bach-y-Rita, pioniere in quest'ambito, sfruttavano una vecchia sedia di un dentista, il cui schienale era stato modificato con l'aggiunta di una rete di solenoidi che facevano muovere dei pistoncini, in modo da ricreare l'immagine presa da una fotocamera sulla schiena del paziente. Altri tipi di stimolazione fanno affidamento sull'udito, o usano matrici da applicare sulla lingua, molto sensibile e che si è rivelata adatta alla trasposizione di stimoli visivi in stimoli tattili.

Andremo ora ad analizzare qualche principio dietro al funzionamento di questo fenomeno.

2.2 Fisiologia

Quando un individuo diventa cieco o sordo, generalmente non perde la capacità di vedere o udire, perde semplicemente l'accesso ad una certa periferica. Ad esempio normalmente nella retina viene raccolto il segnale luminoso che viene convertito in segnale elettrico nel nervo ottico e mandato al cervello: siccome è quest'ultimo a dover elaborare il segnale, la sostituzione sensoriale è possibile. In particolare, il punto cardine su cui si basa la sostituzione sensoriale è che le immagini vengono interpretate in una specifica area del cervello e quell'area continua a rimanere funzionante anche quando non arrivano più stimoli. Se andassino ad analizzare l'attività cerebrale, tramite risonanza magnetica funzionale, di un soggetto collegato ad un sistema di sostituzione sensoriale, vedremmo illuminarsi la parte del cervello adibita alla gestione della modalità sensoriale sostituita. Sono stati condotti altri esperimenti proprio per capire quanto fossero collegati gli stimoli del dispositivo di sostituzione con determinate parti del cervello.

È anche importante distinguere i concetti di percezione e sensazione. Esiste infatti una sottile linea che separa questi due processi molto collegati. Con sensazione si intende l'input dal mondo fisico ottenuto da un sensore di ricezione, con percezione invece si intende il processo con cui il cervello sceglie, organizza e interpreta queste sensazioni[2]. Con la sostituzione sensoriale si ha quindi la percezione di un senso tramite un altro, ad esempio si riuscirebbero a percepire immagini da suoni, o stimoli tattili.

2.2.1 Neuroplasticità

La caratteristica per cui il dottor. Eagleman definisce il cervello come un "general purpose computer" è quella che prende il nome di neuroplasticità, detta anche plasticità cerebrale, cioè la capacità dell'encefalo di modificare la propria struttura e le prorpie funzionalità

a seconda dell'attività dei propri neuroni, correlata ad esempio a stimoli ricevuti dall'ambiente esterno, in relazione a lesioni traumatiche o modificazioni patologiche e in relazione al processo di sviluppo dell'individuo[7].

È anche ragionevole pensare che sia questo meccanismo a consentire la riorganizzazione dei neuroni e della corteccia in risposta alla perdita di un senso, permettendo agli altri di funzionare meglio. L'immagine funzionale di pazienti affetti da cecità congenita ha mostrato il coinvolgimento di più modalità sensoriali nella corteccia occipitale, generalmente usata per la visione o durante prove quali lettura breille, percezione tattile, riconoscimento di oggetti tramite il tatto e localizzazione e riconoscimento di suoni. Questo potrebbe suggerire che le persone affette da cecità possano usare il lobo occipitale per riconoscere oggetti tramite altre modalità sensoriali. Questa plasticità modale incrociata potrebbe spiegare la tendenza comune dei ciechi a mostrare abilità incrementate negli altri sensi.

2.3 Applicazioni

Le applicazioni non sono ristrette a portatori di disabilità, ma includono anche presentazioni, giochi, realtà aumentata e, secondo il già citato Dr. Eagleman, si potrebbero anche creare nuovi sensi, ad esempio nel talk del 2015[3] la sua squadra stava sviluppando un dispositivo per gli investimenti in borsa, che permettesse di trasmettere in tempo reale tutte le fluttuazioni di mercato. Il Dr. Eagleman ha anche ipotizzato un sistema che aiuti i piloti a mantenere sotto controllo tutti gli apparecchi di cui può essere dotato un aereo; non essendo più vincolati a dover usare la vista per controllare tutti gli strumenti non solo il pilota riuscirebbe a mantenere più facilmente l'attenzione in volo, ma avrebbe anche la sensazione che qualcosa non va in caso di problemi.

Alcuni dei sistemi più popolari sono probabilmente quello sviluppato da Bach-y-Rita "Tactile Vision Sensory Substitution" (TVSS) che converte le immagini in stimolazioni tattili e il "vOICe", che converte stimoli visivi in suono. Per l'avanzamento in questo

campo sono molto importanti fattori come la miniaturizzazione dei componenti e una buona stimoalzione elettrica.

2.3.1 Sistemi Tattili

Per capire la sostituzione sensoriale tattile è essenziale avere delle basi di fisiologia dei recettori della pelle. Ci sono cinque tipi di recettori tattili:

- Corpuscolo di Pacini,
- Corpuscolo di Meissner,
- Corpuscolo di Ruffini,
- Corpuscolo di Merkel,
- Terminazioni nervose libere.

Questi recettori sono caratterizzati principalmente dal tipo di stimolo che li attiva e dalla velocità di adattamento a stimoli sostenuti. Per via del rapido adattamento di alcuni di questi recettori, questi richiedono stimolazioni che cambiano rapidamente nel tempo per far sì che vengano attivati in maniera ottima. Su tutti i meccanorecettori, i corpuscoli di Pacini offrono la maggior sensibilità alle alte frequenze di vibrazioni, partendo da poche decine di Hz a qualche kHz con l'aiuto di alcuni meccanismi di meccanotrasduzione[1].

Gli stimolatori principalmente usati in questo ambito sono due: elettrotattili o vibrotattili. Gli stimolatori elettrotattili usano la stimolazione elettrica direttamente sulle terminazioni nervose della pelle per iniziare il potenziale d'azione e le sensazioni provate sono: bruciore, prurito, dolore, pressione ecc. in base al voltaggio applicato. Gli stimolatori vibrotattili usano la pressione e le proprietà dei meccanorecettori della pelle per

[1]Nella meccanotrasduzione un segnale meccanico viene convertito in un segnale nervoso tramite i meccanorecettori. Alla base di questa forma di trasduzione ci sono i canali ionici a controllo meccanico. Nella recezione tattile l'infossamento della cute innervata dai recettori tattili, causato da uno stimolo, determina una deformazione della loro membrana con conseguente apertura dei canali ionici e depolarizzazione.

iniziare il potenziale d'azione. Ci sono vantaggi e svantaggi per entrambi i tipi di stimolazione. Con la stimolazione elettrotattile molti fattori possono influenzare la sensazione avvertita, come ad esempio: voltaggio, corrente, forma d'onda, taglia degli elettrodi, materiale, forza di contatto, posizione sulla pelle, spessore e idratazione della pelle. La stimolazione elettrotattile può coinvolgere la diretta stimolazione dei nervi (percutanea), oppure quella dalla pelle (transcutanea). Le applicazioni percutanee causano stress addizionale al paziente, svantaggio principale di questo approccio, inoltre la stimolazione della pelle senza inserzione necessita di un maggior voltaggio per via della secchezza cutanea, a meno che non venga usata la lingua come recettore, che richiede all'incirca il 3% del medesimo voltaggio. Alternativamente anche il palato è stato proposto il palato come zona in cui possono essere sentite correnti deboli.

I sistemi vibrotattili devono tenere in considerazione il rapido adattamento della pelle allo stimolo e le aree in cui i recettori sono più abbondanti, come: polpastrelli, faccia e lingua. Sono invece più sparsi su schiena, braccia e gambe. Questa densità di recettori è il principale aspetto da tenere in considerazione per la realizzazione di display vibrotattili, in quanto vanno a determinare la risoluzione della sostituzione sensoriale.

Sistemi tattili-visivi

Oltre ai sistemi datati come il TVSS, sono stati sviluppati altri tipi di dispositivi per interfacciare le immagini ai recettori tattili su diverse aree del corpo come petto, fronte, polpastrello, addome o lingua. L'immagine tattile viene riprodotta da centinaia di attivatori posizionati sulla persona. Gli attivatori sono solenoidi, arbitrariamente piccoli, con diametri di circa un millimetro. In alcuni esperimenti soggetti ciechi o bendati equipaggiati con il TVSS hanno imparato a riconoscere forme e orientarsi. Nel caso di semplici forme geometriche, sono servite solo circa 50 prove per raggiungere l'accuratezza del 100%. Per riconoscere oggetti orientati diversamente servono invece molte ore di pratica.

Un sistema basato sulla lingua risulta essere particolarmente pratico, in quanto esso

è contenuto in un ambiente protetto (la bocca chiusa) e la saliva fa da buon ambiente elettrolitico che consente un buon contatto per gli elettrodi. In uno studio condotto da Bach-y-Rita si nota che la stimolazione della lingua richiede il 3% del voltaggio richiesto per la stimolazione di un polpastrello. Risulta inoltre comodo poterlo agganciare ad un apparecchio odontoiatrico in modo da tenerlo fisso nella posizione desiderata. È possibile vederne un esempio in figura 2.1

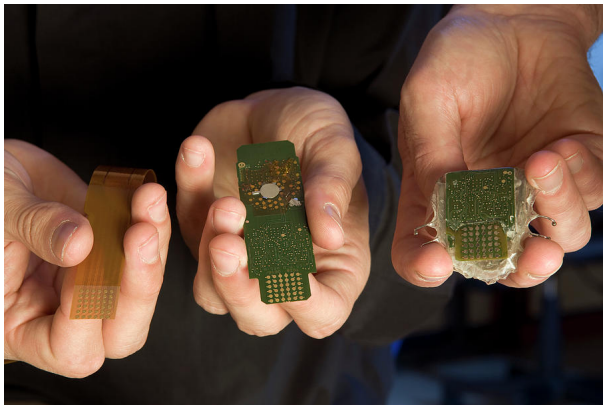

Figura 2.1: Tongue display

Questo tipo di display è quello più utilizzato con sistemi TVSS e per funzionare manda stimoli al dorso della lingua tramite una matrice flessibile di elettrodi posizionata nella bocca. Questa matrice di elettrodi è collegata ad una unità Tongue Display (detta TDU) da un cavo a nastro che passa fuori dalla bocca. Una videocamera registra le immagini, le trasmette al TDU per la conversione in immagini tattili. L'immagine tattile viene poi proiettata sulla lingua tramite il cavo a nastro dove i recettori della lingua raccolgono il segnale. Dopo l'allenamento i soggetti sono in grado di associare certi tipi distimoli ad immagini visive, consentendo l'uso del tatto per percepire la vista.

La sostituzione sensoriale ha avuto successo anche con l'insorgenza di attuatori indossabili come motori vibrotattili, solenoidi o diodi termici. Sono state sviluppate tecnologie che permettono a persone affette da cecità di percepire informazioni "sociali" sull'ambiente circostante usando una cintura vibrotattile (Haptic Belt) e dei guanti (VibroGlove) che

possiamo osservare in figura 2.2. Entrambe le tecnologie usano delle piccole fotocamere montate in un paio di occhiali indossati dall' utilizzatore. La cintura fornisce la distanza e direzione dalla persona di fronte all'utilizzatore mentre i VibroGloves usano una mappatura spaziotemporale di pattern di vibrazione per esprimere le espressioni facciali del partner con cui si interagisce. Alternativamente è stato mostrato come anche piccoli indizi che indichino la presenza o l'assenza di ostacoli possano essere utili per la navigazione, per l'andatura e riducano l'ansia di muoversi in ambienti sconosciuti. Questo approccio chiamato "Haptic Radar" è stato studiato dall'università di Tokyo in collaborazione con l'Università di Rio de Janeiro. Un prodotto simile ad esempio è lo "Eyeronman vest and belt".

Figura 2.2: VibroGlove

Sistemi tattili-acustici

Questo è il sistema di cui si parla anche nel capitolo 1. Presentato dal Dr. David Eagleman al TED nel 2015[3], questo prodotto cattura suoni e li trasforma in pattern tattili multidimensionali sulla pelle.

Esperimenti hanno confermato come stimolazioni tattili possano attivare la corteccia uditiva. Attualmente questi stimoli vengono usati per migliorare l'udito in persone affette da ipoacusia o non-udenti. Negli esperimenti si è osservato come la stimolazione delle dita portasse all'attivazione della cintura uditiva (un'area del cervello), suggerendo una relazione tra udito e tatto.

Il progetto del Dr. Eagleman si è rivelato molto efficace, permettendo a soggetti non udenti di capire parole dopo brevi periodi di utilizzo e allenamento con il dispositivo. Il progetto si rivela inoltre incredibilmente utile anche perché gli utilizzatori riescono ad imparare, correggersi ed allenarsi semplicemente con l'utilizzo della propria voce. Il progetto è stato sviluppato ulteriormente dopo il TED del 2015, a tal punto da essere diventato un wearable acquistabile online chiamato "neosensory buzz", di cui viene fornita un'immagine in 2.3.

Figura 2.3: Neosensory Buzz

Sistema tattile-vestibolare

Alcune persone con problemi di equilibrio o reazioni collaterali agli antibiotici possono avere danni al sistema vestibolare. Questi individui possono avere difficoltà a mantenere la postura, possono avere un'andatura instabile e oscillare. Per risolvere questo problema

è stato montato un accelerometro su un caschetto e un'interfaccia cervello-macchina che utilizza la stimolazione elettrotattile sulla lingua per fornire informazioni sull'orientazione della testa rispetto al corpo, in modo da facilitare l'andatura del paziente. Si è anche visto che in alcuni casi dopo l'utilizzo si avevano dei "residui" che permettevano al soggetto di mantenere l'equilibrio in maniera indipendente dall'apparecchio per una quantità di tempo che variava in funzione del tempo di utilizzo del sistema, portando alcuni pazienti a non doverlo più usare o usarlo molto raramente[6].

Sistema tattile-tattile

In questo sistema le informazioni tattili vengono trasmesse da un'area all'altra. In un esperimento con un paziente che aveva perso la sensibilità periferica a causa della lebbra, è stato messo un guanto con dei sensori che inviavano segnali ad una matrice tattile sulla fronte. Dopo un periodo di allenamento e dopo essersi abituato il paziente ha riportato di aver provato "la fantastica sensazione di toccare sua moglie", che non provava da 20 anni.

2.3.2 Sistemi Uditivi

I sistemi di sostituzione uditivi come quelli tattili mirano ad usare una modalità sensoriale per compensare la mancanza di un'altra in modo da ottenere la percezione di quella mancante. Con la sostituzione uditiva, sensori tattili o visivi registrano informazioni dall'esterno, queste informazioni vengono poi trasformate da delle interfacce in suono. La maggior parte dei sistemi di sostituzione uditivi-visivi mirano ad usare il senso dell'udito per trasmettere informazioni ai ciechi.

Il display uditivo vOICe

Il "vOICe" è il principale concorrente del TVSS e per funzionare trasforma le immagini prese da una fotocamera montata su un paio di occhiali in pattern uditivi, che associano

l'altezza alla tonalità e la luminosità al volume in una scannerizzazione che va da sinistra a destra per ogni frame del video. Si ha un esmpio del sistema vOICe in figura 2.4.

Nonostante sia datato (è stato sviluppato nel 1990), questo sistema è molto utilizzato e sono stati fatti anche esperimenti che dimostrano come con la sostituzione sensoriale si vadano effettivamente ad attivare con altre modalità sensoriali (in questo caso l'udito) aree del cervello adibite a precisi compiti (in questo caso abbiamo una reazione della corteccia visiva). In questi esperimenti si è provato a far riconoscere degli oggetti ai partecipanti. Dopo averli istruiti e addestrati, si andava a deattivare temporaneamente la corteccia visiva mediante tecniche non invasive come la stimolazione magnetica transcranica. Questo tipo di stimolazione consiste fondamentalmente nel tenere un potente magnete estremamente potente vicino all'area del cervello che si vuole influenzare, permettendo di inibirla per un breve lasso di tempo. Dopo il trattamento i soggetti affermavano di percepire tutto più offuscato e meno nitido. Nonostante arrivassero loro solamente stimoli uditivi il cervello andava ad utilizzare la parte adibita alla vista per analizzare gli input e trasformarli in vere e proprie immagini. Per questo motivo i soggetti "vedevano" meglio prima della deattivazione.

Figura 2.4: The vOICe

Capitolo 3

Il prototipo hardware

Come si può vedere in figura 1.1, il sistema comprende:

- un Raspberry Pi 4 model B;
- una modulo camera;
- una breadbord con led e tre pulsanti, dall'alto al basso rispettivamente;
    1. accensione/spegnimento,
    2. start/stop del programma,
    3. descrizione audio,
- una batteria da 20.000 mAh.

I led sono solo otto e rappresentano quattro pixel equidistanziati nella riga in alto (la fila verticale di destra) e altri quattro pixel equidistanziati nell'ultima riga dell'immagine.

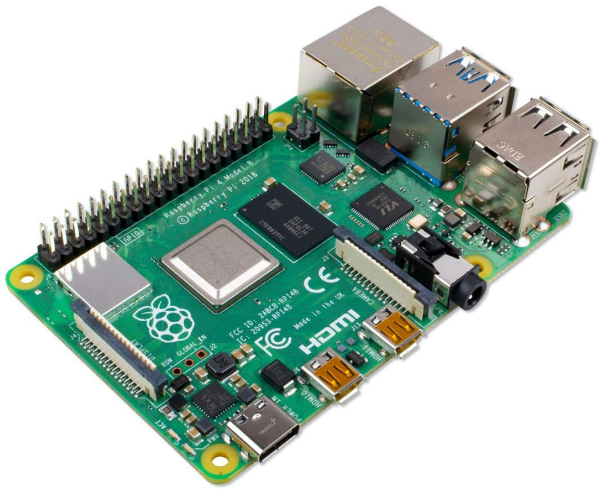

Figura 3.1: Raspberry Pi 4 model B

3.1 Componenti

3.1.1 Raspberry Pi 4 model B

Inizialmente si pensava di far girare l'intero programma sul dispositivo indossabile, che avrebbe dovuto quindi:

- avere un numero sufficientemente grande di pin per facilitare la gestione di input e output;
- essere abbastanza potente da permettere lo svolgimento dei calcoli e delle operazioni di output necessarie in tempi discretamente brevi;
- avere dimensioni contenute;
- avere un consumo ridotto;
- avere un input (che per questo progetto è una camera ma potrebbe essere anche sostituito con altro, come spiegheremo in seguito).

Per questi motivi si è scelto di utilizzare il Raspberry Pi model B, un single-board computer sviluppato nel Regno Unito dalla Raspberry Pi Foundation. La scheda è stata

progettata per ospitare sistemi operativi basati sul kernel Linux o RISC OS. Utilizzato con un sistema operativo appositamente dedicato, chiamato Raspbian.

Specifiche:

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz;
- 4GB LPDDR4-3200 SDRAM;
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE;
- Gigabit Ethernet;
- 2 USB 3.0 ports; 2 USB 2.0 ports;
- 40 pin GPIO Raspberry Pi standard;
- 2 porte micro-HDMI (che supportano fino a 4kp60);
- 2-lane MIPI DSI display port;
- 2-lane MIPI CSI camera port;
- 4-pole stereo audio and composite video port;
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode);
- OpenGL ES 3.0 graphics;
- Micro-SD card slot per caricare il sistema operativo e immagazzinare dati;
- 5V DC via USB-C connector (minimo 3A);
- 5V DC via GPIO header (minimum 3A);
- Power over Ethernet (PoE) abilitato (richiede PoE HAT separati).

Nonostante la potenza di questo computer, la computazione dell'immagine impiegava troppo tempo, si è quindi deciso di sviluppare anche una parte server che si occupasse delle parti di calcolo, sfruttando anche una GPU interna per velocizzare il tutto. A questo punto non è più necessario un computer così potente, e si consente lo sviluppo dello stesso programma anche su dispositivi più piccoli e leggeri, sempre a patto che abbiano accesso ad internet, un modo per controllare l'output e un modo per prendere input dall'esterno. L'aspetto del Raspberry Pi 4 model B è quello mostrato in figura 3.1

3.1.2 Camera B01

Come metodo di input avremmo potuto utilizzare molte tecnologie diverse, tra quelle prese in considerazione ci sono:

- Lidar;
- Kinect;
- Sensore a ultrasuoni;
- Stereocamera;
- Camera.

Bisogna ricordare che questo progetto non ha ricevuto fondi ed è stato svolto in un periodo limitato di tempo, il che ha influito sulla scelta delle tecnologie da utilizzare, sia per quanto riguarda il prezzo che per i tempi di consegna dei pezzi.

Lidar

LIDAR (acronimo dall'inglese Light Detection and Ranging o Laser Imaging Detection and Ranging) è una tecnica di telerilevamento che permette di determinare la distanza di un oggetto o di una superficie utilizzando un impulso laser, ma è anche in grado di determinare la concentrazione di specie chimiche nell'atmosfera e nelle distese d'acqua.

Figura 3.2: Scanner Lidar

Questa tecnica è già molto usata nella guida autonoma ed è molto performante e preciso. Non è stato usato per il progetto per motivi di costo anche se sarebbe stato la prima scelta. Tuttavia non è da escludere che in futuro non vengano fatte altre tipologie di dispositivi che sfruttano questa tecnica anziché la camera. La figura 3.2 mostra un esempio di lidar.

Kinect

Figura 3.3: Kinect

Un altro sensore che è stato preso in considerazione è il kinect, che tuttavia è stato

scartato per non essere molto discreto e per non essere molto performante in situazioni di luce intensa. In figura 3.3 ne abbiamo un esempio.

Sensore a ultrasuoni

Figura 3.4: Sonar

Il funzionamento alla base di questo dispositivo si è già dimostrato molto efficace in natura, essendo utilizzato ad esempio dai pipistrelli e il prezzo è veramente irrisorio. Tuttavia funzionando col suono ha delle limitazioni, non ha un raggio molto ampio, in caso di rumore forte può non funzionare bene, in caso ci siano materiali fonoassorbenti non si riuscirebbe ad avere una misurazione accurata (la neve è un ottimo assorbente naturale che avrebbe potuto comprometterne il funzionamento in inverno), inoltre (anche se è un problema minore) bisogna aspettare che le onde sonore tornino indietro per effettuare la misurazione ed il suono è molto più lento rispetto alla luce. La figura 3.4 mostra una delle tipologie più economiche in commercio, che può essere acquistato per pochi euro.

Stereocamera

Figura 3.5: Stereocamera

La seconda miglior scelta sarebbe stata una stereocamera, ovvero due camere che permettano la comunicazione sincrona con il dispositivo che le controlla.

Con un dispositivo del genere si sarebbero potute calcolare le distanze esattamente come fa il cervello coi nostri occhi, permettendo una rappresentazione del tutto simile a quella naturale.

Tuttavia anche questo tipo di sensore è stato scartato per il prezzo troppo elevato. In figura 3.5 se ne può osservare un esempio.

Camera B01

Una camera singola si trova su internet a un costo di circa dieci euro, il che la rende molto economica. Anche se non è possibile misurare in modo preciso la distanza da un oggetto utilizzando una sola camera esistono progetti per ottenere una stima; in particolare possiamo citare **monodepth2**, pilastro fondamentale del presente progetto di tesi.

Avendo dimensioni molto contenute la camera è discreta, il funzionamento di base è equiparabile a quello dell'occhio umano, che fa passare la luce riflessa dagli oggetti e ne permette l'immagazzinamento in una memoria, inoltre in linea di principio funziona negli stessi ambienti in cui funziona l'occhio umano.

Di seguito vengono presentate le specifiche della camera:

Figura 3.6: Camera

- 5MP Omnivision 5647 Camera Module;
- immagine fissa: 2592 x 1944;
- video: supporta registrazioni a 1080p @ 30fps, 720p @ 60fps e 640x480p 60/90;
- dimensioni: 20 x 25 x 9mm;
- peso: 3g.

La camera utilizzata ha l'aspetto mostrato in figura 3.6

3.1.3 LED

Come output sono stati utilizzati dei led, il cui aspetto tipico è quello mostrato in figura 3.7. Questa scelta è stata dettata da due motivi principali:

- i led erano reperibili fin da subito e a costo nullo, grazie alla disponibilità del prof. Alessando Ricci che mi ha concesso di utilizzarli per la durata del progetto (insieme anche a pulsanti, breadbord, resistenze e jumpers);

Figura 3.7: Led

- al contrario di motori di vibrazione o display tattili, è possibile osservare il cambiamento di ciascun led a distanza, senza un contatto diretto.

In particolare il led funziona con una corrente di 20mA e una tensione diretta di 2.2V. Ciò ha reso necessario l'utlizzo di resistenze da 220Ohm, che si possono notare nella figura 1.1.

In una versione utilizzabile del progetto i led andrebbero sostituiti con dispositivi che permettano di trasmettere un feedback attraverso la pelle, come ad esempio una matrice di motori di vibrazione o un display tattile (trattati nel capitolo 2).

3.1.4 Batteria

In questo caso è stata utilizzata una batteria da 20.000 mAh, scelta unicamente per disponibilità, che garantisce una durata teorica di circa 6,4 ore. Questo modello in particolare ha anche un pannello solare che potrebbe dare qualche minuto di vita in più al sistema.

Nel sistema finito, la scelta della batteria potrebbe essere più oculata, andando a considerare anche peso, ingombro e periodo di attività da essa garantito.

Figura 3.8: Server

3.1.5 Server

Come già anticipato, per far sì che il dispositivo fosse realmente utilizzabile, si è reso necessario l'utilizzo dei un server. In questo caso l'unico a disposizione è stato uno Xiaomi mi notebook pro con le seguenti caratteristiche:

- CPU: Intel Core i5-8250U;
- GPU: Nvidia MX150 con 2 GB GDDR5;
- Memoria: SSD.

La parte cruciale riguarda la scheda video, che permette di eseguire calcoli parallelamente, velocizzando di molto l'esecuzione dell'intero programma.

Sono stati forniti questi dati unicamente a scopo illustrativo e per dare un minimo di contesto alle misurazioni sui tempi che verranno mostrate più avanti. Il server utilizzato è quello in figura 3.8.

Capitolo 4

Il progetto software

Prima di iniziare a programmare sono state scelte le librerie da utilizzare ed erano tutte dell'ecosistema Python, linguaggio interpretato multi-paradigma che fa della leggibilità il proprio punto di forza. Il Python permette di scrivere codice molto velocemente grazie anche al supporto di una comunità molto attiva che continua a sviluppare librerie che facilitano ampiamente il lavoro agli utilizzatori, spesso anche ottimizzate e che permettono di rendere il codice molto performante. Due esempi eclatanti sono NumPy e OpenCV, due librerie fondamentali per la visione artificiale, che snelliscono e velocizzano il codice permettendo di ottenere ottimi risultati in tempi brevi e in poche righe di codice.

È stato quindi scelto il Python come linguaggio di sviluppo per i seguenti motivi:

- il progetto è basato sulla visione artificiale, ambito in cui il Python si è già affermato come prima scelta, grazie anche alle librerie sopra citate;
- sono state trovate solo librerie utilizzabili da questo linguaggio;
- il raspberry lo supporta pienamente, consentendone l'utilizzo anche per la gestione di periferiche aggiuntive, come la camera e i GPIO[1].

[1]General Purpose Input/Output, interfacce programmabili di connessine con periferiche esterne che possono essere usate come input o come output.

A proposito dell'ultimo punto, di solito per la programmazione di dispositivi embedded e per l'IOT si usano linguaggi molto performanti, come ad esempio il C/C++, per limitare l'utilizzo di risorse e i consumi. Questo progetto però non deve svolgere dei task entro un certo tempo, il suo unico fine è quello di analizzare l'ambiente circostante con la frequenza più alta possibile e generare un conseguente output, inoltre come già accennato nel capitolo 1 nel caso di una connessione è possibile separare le due parti di computazione e gestione input output, il che permetterebbe di scrivere la parte embedded in un altro linguaggio a scelta.

4.1 Ambienti virtuali

È buona pratica, quando si programma in Python, creare ed utilizzare degli ambienti virtuali; in modo da non ritrovarsi con dei conflitti dovuti a versioni più o meno recenti di librerie usate, in particolare per il progetto ne sono stati usati tre, uno per il server, gestito con anaconda[2], e due su raspberry, costruiti grazie a venv[3]. Il raspberry purtroppo non supporta anaconda, ma una sua versione limitata, miniconda, che è stata provata ma che si è rivelata non adatta allo scopo per via delle librerie poco aggiornate che spesso andavano in conflitto le une con le altre. Gli ambienti virtuali permettono di scegliere con quale interprete Python far girare il codice e con il comando

```
pip freeze > requirements_file.txt
```

oppure con anaconda

```
conda list -e > requirements.txt
```

producono un file con i requisiti necessari per generare lo stesso ambiente virtuale. In questo modo sono stati creati i file

[2]Distribuzione open-source dei linguaggi Python e R per scientific computing, che mira a semplificare la gestione e il deployment di pacchetti

[3]Modulo per la creazione di ambienti virtuali leggeri, dove ogni ambiente ha il proprio binario Python e può avere il proprio set indipendente di pacchetti installati nelle proprie cartelle

- pi_requirements.tx;
- server_requirements.txt;
- sad_requirements.txt.

4.1.1 Requisiti ambiente generale

Questo è l'ambiente virtuale principale, localizzato nella cartella eye, viene gestito con venv e contiene il binario Python più utilizzato nel progetto.

certifi==2020.6.20
chardet==3.0.4
idna==2.10
picamera==1.13
pkg-resources==0.0.0
requests==2.24.0
RPi.GPIO==0.7.0
urllib3==1.25.9

Questo ambiente viene utilizzato per tutti gli script che devono connettersi ad internet, include poche librerie Python ed è abbastanza leggero. I pacchetti più importanti sono:

- picamera, per la gestione della fotocamera.
- requests, per la gestione delle richieste http.
- RPi.GPIO, per la gestione dei GPIO.

4.1.2 Requisiti server

Questo è l'ambiente creato grazie ad anaconda ed è l'interprete dello script che gira su server linux x64.

This file may be used to create an environment using:

```
# $ conda create –name <env> –file <this file>
# platform: linux-64
_libgcc_mutex=0.1=conda_forge
_openmp_mutex=4.5=1_llvm
aniso8601=8.0.0=pypi_0
ca-certificates=2020.6.20=hecda079_0
certifi=2020.6.20=py36h9f0ad1d_0
cffi=1.14.0=py36hd463f26_0
click=7.1.2=pypi_0
cycler=0.10.0=pypi_0
flask=1.1.2=pypi_0
flask-restful=0.3.8=pypi_0
freetype=2.10.2=he06d7ca_0
itsdangerous=1.1.0=pypi_0
jpeg=9d=h516909a_0
kiwisolver=1.2.0=pypi_0
libblas=3.8.0=16_openblas
libcblas=3.8.0=16_openblas
libffi=3.2.1=he1b5a44_1007
libgcc-ng=9.2.0=h24d8f2e_2
libgfortran-ng=7.5.0=hdf63c60_6
libgomp=9.2.0=h24d8f2e_2
liblapack=3.8.0=16_openblas
libopenblas=0.3.9=h5ec1e0e_0
libpng=1.6.37=hed695b0_1
libstdcxx-ng=9.2.0=hdf63c60_2
libtiff=4.1.0=hc7e4089_6
libwebp-base=1.1.0=h516909a_3
```

llvm-openmp=10.0.0=hc9558a2_0

lz4-c=1.9.2=he1b5a44_1

matplotlib=3.2.2=pypi_0

mkl=2020.1=219

ncurses=6.1=hf484d3e_1002

ninja=1.10.0=hc9558a2_0

numpy=1.18.5=py36h7314795_0

olefile=0.46=py_0

openssl=1.0.2u=h516909a_0

pillow=7.1.2=py36h8328e55_0

pip=20.1.1=py_1

pycparser=2.20=py_0

pyparsing=2.4.7=pypi_0

python=3.6.6=hd21baee_1003

python_abi=3.6=1_cp36m

pytorch=0.4.1=py36_py35_py27___9.0.176_7.1.2_2

pytz=2020.1=pypi_0

readline=7.0=hf8c457e_1001

setuptools=47.3.1=py36h9f0ad1d_0

six=1.15.0=pyh9f0ad1d_0

sqlite=3.28.0=h8b20d00_0

tensorboardx=1.4=pypi_0

tk=8.6.10=hed695b0_0

torchvision=0.2.1=py_2

werkzeug=1.0.1=pypi_0

wheel=0.34.2=py_1

xz=5.2.5=h516909a_0

zlib=1.2.11=h516909a_1006

zstd=1.4.4=h6597ccf_3

I pacchetti più importanti sono:

- pytorch, framework usato da monodepth2.
- torchvision, pacchetto comprendente datasets popolari, architetture di modelli e trasformazioni comuni di immagini per la computer vision, sempre usato da monodepth2.
- OpenCV, usato da monodepth2 e YOLOv3 per operazioni sulle immagini.
- flask, framework per gestione di richieste restful.

4.1.3 Requisiti ambiente stand-alone

Questo ambiente è stato il più complicato da creare in quanto gestisce il programma funzionante esclusivamente su raspberry. Molti dei pacchetti installati non sono le versioni consigliate dagli sviluppatori di monodepth2 (il modulo utilizzato per la stima delle distanze). A causa della scarsa disponibilità di librerie su questo tipo di piattaforma, sono state infatti installate versioni più o meno aggiornate dei pacchetti suggeriti, inoltre OpenCV è stato compilato direttamente dalla repo del progetto e non installato tramite pip o anaconda. Questo ambiente risiede nella cartella sad, qui sotto andremo ad elencarne i requisiti.

cycler==0.10.0

kiwisolver==1.2.0

matplotlib==3.3.0rc1+147.gdbea6f8a3

numpy==1.19.0

picamera==1.13

Pillow==7.2.0

pkg-resources==0.0.0

protobuf==3.12.2

pyparsing==3.0.0a2

python-dateutil==2.8.1

RPi.GPIO==0.7.0

six==1.15.0

tensorboardX==1.4

torch==1.1.0

torchvision==0.3.0

I pacchetti più importanti sono i seguenti:

- picamera, per la gestione della fotocamera.
- RPi.GPIO, per la gestione dei GPIO.
- pytorch, framework usato da monodepth2.
- torchvision, pacchetto comprendente datasets popolari, architetture di modelli e trasformazioni comuni di immagini per la computer vision, sempre usato da monodepth2.
- OpenCV, usato da monodepth2 per operazioni sulle immagini.

4.2 Monodepth2

Monodepth2 è la libreria usata per stimare le profondità in immagini acquisite con una singola camera, senza alcuna informazione aggiuntiva. In questo capitolo andremo a spiegare brevemente i concetti alla base del suo funzionamento.

I metodi basati su machine learning hanno mostrato risultati molto promettenti per la stima della profondità in immagini singole[4], tuttavia, la maggior parte degli approcci

[4]Con immagini singole o singole immagini si intende immagini non stereo

usati tratta la predizione delle distanze come un problema di regressione supervisionata che quinidi necessitano di grandi quantità di corrispondenti dati definiti "ground truth", ovvero le effettive distanze dal misuratore all'oggetto misurato. In questo ambito è però molto complicato ottenere dati di buona qualità in una grande quantità di ambienti diversi. La parte innovativa di questo progetto sta nella sostituzione di dati espliciti durante la fase di training con delle immagini stereo, che sono più facili da ottenere.

Usando i principi della geometria epipolare[5], vengono generate delle immagini di disparità[6] addestrando la rete a stimare le distanze dalla camera partendo direttamente da tali dati di training. Si mostra che risolvendo solamente per ricostruzione di immagine si ottengono delle predizioni della profondità di scarsa qualità. Per superare questo problema, viene proposta una nuova tecnica di training che rafforza la consistenza tra le disparità prodotte relativamente alle immagini sia di destra che di sinistra, portando ad un miglioramento delle perfermormance e ad una miglior robustezza rispetto agli approcci esistenti. Questo metodo produce risultati allo stato dell'arte per la stima delle profondità sul dataset di riferiremo KITTI, superando anche metodi supervisionati che erano stati addestrati con valori di ground truth delle distanze.

4.2.1 Metodo

Gli uomini riescono a dare una buona stima delle distanze con un solo occhio analizzando indizi come la prospettiva, scale rispetto a oggetti conosciuti o familiari, apparenze riguardo luci, ombre ed occlusioni. Questa combinazione di indizi top-down e bottom-up apparentemente collega la conoscenza di tutta la scena, nella sua completezza, alla nostra abilità di fare stime accurate sulla profondità. Il modello completamente convoluzionale che viene usato non richiede nessun dato e viene addestrato a partire da immagini stereo.

[5]La geometria epipolare è la geometria della visione stereoscopica. Essa descrive le relazioni e i vincoli geometrici che legano due immagini 2d alla stessa scena 3d catturata da due fotocamere con posizione e orientamento distinto.

[6]Anche detta disparità binoculare, ovvero la differenza fra le immagini dei due occhi della stessa immagine visiva. [1]

Esso impara a predire la corrispondenza a livello di pixel tra coppie di immagini stereo rettificate da una stereocamera di cui si conoscono le specifiche.

Questo metodo si concentra su lavori legati alla stima delle distanze su immagini singole, senza fare assunzioni riguardo la geometria della scena o i tipi di oggetti presenti. In particolare, data la difficoltà di acquisire buoni valori di ground truth della profondità, anche utilizzando costosi scanner laser che possono essere imprecisi (specialmente in situazioni con superfici riflettenti o in movimento), si è pensato di trattare il problema come una ricostruzione di immagine. L'intuizione è che avendo a disposizione una coppia di fotocamere appositamente calibrate, se si riuscisse a trovare una funzione che ricavi un'immagine dall'altra, allora si potrebbe apprendere qualcosa della struttura 3d della scena.

In particolare durante l'allenamento abbiamo a disposizione due immagini I^l e I^r, corrispondenti alle immagini a colori sinistra e destra di una coppia stereo calibrata e catturata nello stesso istante. Anziché cercare di trovare direttamente la profondità proviamo a trovare il campo di corrispondeza di densità d^r tale che, quando applicato all'immagine sinistra, ci permetterebbe di ricostruire l'immagine di destra. Ci riferiremo all'immagine $\mathrm{I}^l(\mathrm{d}^r)$ come $\mathrm{I'}^r$, allo stesso modo $\mathrm{I}^r(\mathrm{d}^l)$ sarebbe $\mathrm{I'}^l$.

Supponendo che le immagini siano rettificate[7], d corrisponde alla disparità dell'immagine (un valore scalare per ogni pixel che il modello imparerà a predire.). Data la distanza b fra le due fotocamere e la lunghezza focale f può essere quindi ricavata la profondità d' dalla disparità predetta come $d' = bf/d$

Come già accennato, in fase di allenamento come input nella rete vengono usate, sia le immagini di destra che di sinistra, in modo da avere una maggior consistenza sui risultati.

La figura 4.1 è un possibile input da fornire a monodepth2, che dopo la stima delle profondità restituisce un'immagine come quella in figura 4.2. Per semplicità nel progetto

[7]La rettificazione delle immagini è un processo usato per proiettare immagini su uno stesso piano comune. viene usato nella visione stereo nei computer per semplificare il problema di trovare punti in comune tra le immagini

si è deciso di usare un'immagine con scala di grigi, dove più il pixel è bianco più quella parte di foto è vicina al soggetto. Il risultato è sempre lo stesso e può essere osservato in figura 4.3.

Il progetto Monodepth2 è open-source e disponibile online. Per utilizzarlo ho scaricato il progetto, preso il file adibito al testing del progetto e l'ho modificato opportunamente, aggiungendo tutta la parte di loop, gestione di GPIO e camera da parte del raspberry, o, nel caso del server la parte di codice che permette l'analisi dell'immagine anche a YOLOv3.

Figura 4.1: Immagine di test

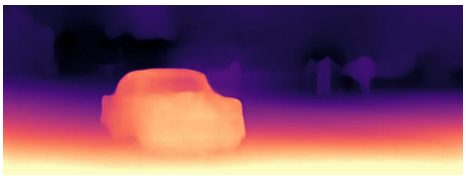

Figura 4.2: Immagine di test dopo la stima delle profondità con monodepth2

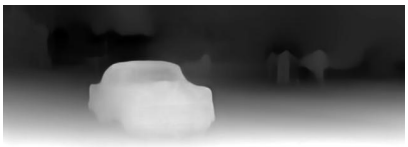

Figura 4.3: Immagine di test dopo la stima delle profondità con monodepth2 in bianco e nero

4.3 Funzionamento ad alto livello del programma

Prima di procedere nella descrizione ricordiamo che il raspberry è connesso a tre pulsanti:

1. accensione/spegnimento,

2. start/stop del programma,

3. descrizione audio.

In tutto vengono usati sei file:

1. starter.py,
2. restart_program.py ,
3. vision.py,
4. eye.py,
5. reader.py,
6. vision_webapi.py.

Per chiarezza daremo una spiegazione approssimativa del funzionamento di questi files e di come sono legati l'uno all'altro.

Per il controllo e la reattività dei pulsanti è necessario che ci sia sempre almeno uno script attivo (oltre quello della gestione del pulsante di accensione/spegnimento, che non è stato implementato da me, ma è stato installato tramite un progetto open-source).

starter.py Viene eseguito all'accensione del dispositivo e tenta di collegarsi ad un sito per due secondi (tempo scelto in maniera empirica). Se entro quel tempo non trova una connessione ad internet fa partire il programma che lavora solo sul raspberry, ovvero vision.py. In caso di connessione avvenuta con successo parte invece lo script che comunica col server, eye.py, mentre sul server gira il file vision_webapi.py.

restart_program.py È lo script addetto a far ripartire il programma. L'unica cosa che fa è far partire nuovamente starter.py alla pressione del pulsante di start.

vision.py Viene eseguito in assenza di connessione. Dopo aver istanziato le variabili per il controllo di camera e led inizia un loop infinito, che può essere interrotto solo dal pulsante di stop. Nel loop viene scattata una foto, ne viene fatta la stima delle distanze

e questa distanza viene utilizzata per far illuminare i led in funzione della distanza dall'oggetto, in particolare più si è vicini, maggiore sarà la luminosità.

Alla pressione del pulsante di stop l'analisi non viene interrotta, al termine del ciclo vengono liberate le variabili di controllo di led e camera, viene istanziato un thread con lo script restart_program.py e lo script termina.

In questo script la funzione di descrizione dell'ambiente è stata disabilitata per via dei tempi già proibitivi. Ogni ciclo di stima delle distanze, comprendente anche scatto della foto e output sui led, impiega infatti 4 secondi circa, che con la funzione aggiuntiva sarebbero potuti anche raddoppiare.

eye.py Questo script è molto più leggero, della controparte vision.py. Dopo aver istanziato le variabili per controllare camera e led inizia un loop interrompibile solo dal pulsante di stop. Nel ciclo viene scattata una foto, inviata al server, e viene attesa la risposta con i valori da assegnare a ciascun led e una lista contenente gli oggetti che sono stati riconosciuti da YOLO v3. Alla pressione del pulsante di descrizione dell'ambiente viene trascritta in un file (JSON) la lista di oggetti trovati e istanziato un thread che riproduce i corrispondenti file audio. Alla pressione del tasto stop, a ciclo terminato vengono liberate le variabili relative alla camera e ai led, viene istanziato un thread con lo script restart_program.py e lo script termina.

reader.py È lo script incaricato alla descrizione ambientale. Apre il file con la lista di oggetti, ne riproduce i rispettivi file audio e termina.

vision_webapi.py Questo script viene eseguito lato server. È diviso in due parti, una attende una richiesta POST con cui viene caricata l'immagine e la salva in locale, dopodiché alla ricezione di una richiesta di GET analizza l'immagine facendone la stima delle distanze (in circa 0.6-0.8 secondi) e le riporta in una lista. Identifica gli oggetti tramite l'utilizzo di YOLOv3 (in media 0.6 secondi), anch'essi riportati in una lista e ritorna un JSON con le due liste di valori.

4.4 Analisi degli script

4.4.1 Scelta del file da eseguire all'accensione

```
1  import requests
2  import subprocess
3  from time import sleep
4
5  connecting = True
6  cicles = 0
7
8  def startProgram(command, filepath):
9      global connecting
10     connecting = False
11     subprocess.call([command, filepath, "&"])
12
13 while connecting:
14     try:
15         r = requests.get('http://google.com', timeout=0.2)
16         if r.status_code == 200:
17             startProgram("/home/pi/tirocinio/eye/bin/python",\
18                 "/home/pi/tirocinio/eye.py")
19     except:
20         print("echo" ,"waiting␣for␣connection")
21         sleep(1)
22         cicles += 1
23         if cicles > 1 and connecting is True:
24             subprocess.call(["sudo","./vision.sh"])
```

Listato 4.1: starter.py

Nelle prime tre linee di starter.py andiamo semplicemente ad importare i pacchetti per la gestione delle richieste http, per il controllo dei processi e per la funzione di sleep. A linea 13 entriamo nel loop, facciamo una richiesta all'indirizzo ip del server al quale vogliamo connetterci (in questo caso non avendo un server online si è deciso di usare google) e aspettiamo un tempo pari a quello di timeout (che è stato impostato a 0.2 secondi dopo qualche prova empirica). Se il codice di risposta è 200, siamo connessi a internet e possiamo utilizzare lo script eye.py (a linea 17-18 dichiariamo prima la posizione del binario Python da utilizzare, poi la posizione dello script), altrimenti si aspetta per un secondo (anche qui il tempo è stato scelto empiricamente), si aumenta il numero di cicli compiuti e se si è raggiunto il limite si fa partire il codice funzonante esclusivamente su raspberry. Si noti che a linea 24 non vengono chiamati interprete e file di esecuzione, bensì uno script bash, questo escamotage ha mi ha permesso di assegnare una priorità più alta al file che di default impiega circa 4.3 secondi, mentre con priorità massima ne impiega circa 4.

Lo script vision.sh è il seguente:

```
sudo nice -n -20 /home/pi/sad/bin/python3.7 /home/pi/tirocinio/vision.py
```

4.4.2 Gestione dello start & stop dei processi

```
1  import subprocess
2  from time import sleep
3  import RPi.GPIO as GPIO
4  import threading
5
6  GPIO.setmode(GPIO.BOARD)
7
8  wait_mode = True
9
10 #directly calls the starter.py
```

```
11 def restart_program():
12     subprocess.call(['/home/pi/tirocinio/eye/bin/python',\
13          '/home/pi/tirocinio/starter.py', '&'])
14
15 def set_wait_mode_OFF(channel):
16     if GPIO.input(channel) == GPIO.HIGH:
17         global wait_mode
18         wait_mode = False
19
20
21 #button to start program
22 # Set pin 29 to be an input pin and set initial value to be
      pulled low (off)
23 GPIO.setup(29, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
24 GPIO.add_event_detect(29, GPIO.BOTH, callback=set_wait_mode_OFF)
25 try:
26     while wait_mode:
27         sleep(0.5)
28     t = threading.Thread(target=restart_program)
29     t.start()
30 except:
31     print("program␣stopped␣from␣keyboard")
```

Listato 4.2: restart_progream.py

Nella prima parte importo i pacchetti per la gestione dei processi, per la funzione di sleep, per il controllo dei GPIO e per il multithreading. A linea 6 indico in che modo vengono numerati i pin, la modalità può essere infatti BOARD o BCM, con BOARD i numeri crescono progressivamente con 1, che è il primo pin in alto a sinistra, 2, il primo in alto a destra, 3, il secondo in alto a sinistra e così via. In tutti gli script che necessitano l'utilizzo dei GPIO si farà uso di questa modalità, non verrà quindi ripresa negli utilizzi

successivi. La funzione restart_program è praticamente autoesplicativa e fa partire in background lo script starter.py. Questa è la funzione che viene eseguita alla pressione del pulsante. Il suo unico scopo è quello di impostare la variabile wait_mode a False. In questo modo al ciclo successivo verrà eseguita in un thread a parte la chiamata che farà eseguire starter.py, consentendo il terminamento dello script corrente.

4.4.3 Programma stand-alone su raspberry

Dopo aver importato tutti i pacchetti necessari al funzionamento di monodepth2, vengono anche importati i pacchetti di gestione dei GPIO, della camera, dei processi, utility per il tempo e per la sleep, threading e OpenCV per lavorare con le immagini.

Viene dichiarata la camera, e ruotata di 180 gradi perché il supporto visibile in figura 1.1 della fotocamera risultava più stabile con la camera capovolta, impostata la risoluzione a quella richiesta da monodepth2 (640x192). Avendo a disposizione anche altri tipi di risoluzione con cui è stato addestrato il modello, avremmo potuto scegliere delle risoluzioni più elevate, ma ciò avrebbe implicato più tempo di analisi e un maggior utilizzo di dati per far viaggiare l'immagine in rete. Viene anche impostata la trasparenza della preview della camera, che nel sistema finale non risulta particolarmente rilevante. La camera ha anche un sensore di luminosità, che impiega circa 2 secondi per autocalibrarsi, per questo motivo inizio a misurare il tempo subito dopo l'inizializzazione di tutti i parametri della camera. Sfrutto le linee successive, mentre si aspetta che il sensore venga calibrato, per impostare i pin destinati ai led come output, con valore LOW e frequenza PWM di 50 Hz. Questa è la parte che in un progetto utilizzabile andrebbe sostituita se si usasse un qualche tipo di display tattile. Se invece si usasse una matrice di motori di vibrazione, questa parte resterebbe identica, ma andrebbero aggiunti più motori e andrebbero usati dei componenti hardware appositi chiamati I/O expanders come ad esempio il PCA9698BS, che aggiunge 40 slot.

La funzione exit_cicle, associata al bottone di start/stop, ha come fine ultimo quello di impostare a False il workingFlag, facendci uscire dal ciclo.

La funzione restart_program manda in esecuzione lo script restart_program.py.

Per il bottone di start/stop setto il pin 29 come input, ne attivo la restienza di pull-down[8] e vi aggancio un listener che esegue la funzione exit_cicle.

Distribuisco poi in una lista le posizioni dei led che si vogliono analizzare, in particolare in questo progetto vengono analizzati i pixel della prima e della terzultima riga.

Monodepth2 controlla poi la disponibilità di CUDA[9], che riesce a velocizzare l'esecuzione del programma in funzione alla potenza della scheda video. Purtroppo però, non avendo il raspberry una scheda video questa impostazione resterà disabilitata.

A questo punto se non esiste viene scaricato il modello, che nel nostro caso è mono_640x192 (mono perché l'immagine viene scattata da una sola camera e 640x192 è la risoluzione) e monodepth estrapola le informazioni necessarie alla stima delle distanze.

Terminata la parte di codice di "inizializzazione", prima di iniziare il ciclo che di scatto, analisi e output, viene controllato il tempo passato da quando è stata istanziata la camera, e se quest'ultimo è minore di due secondi si aspetta di arrivarci. Spesso quasi non c'è bisogno di aspettare ulteriormente, infatti si impiegano in media 1.8 secondi prima di arrivare a questo punto.

Inizia ora il ciclo, dove viene scattata una foto, vengono stimate le distanze con monodepth e viene salvato il risultato. L'immagine viene poi riaperta con OpenCV e viene trasformata in Grayscale per semplicità.

Per ogni pixel della lista precedentemente istanziata viene rimappato il valore della luminosità, che in grayscale va da 0 a 255, in un valore da 0 a 100, dove 0 simboleggia la distanza massima e 100 la vicinanza massima e questo valore viene posizionato in una lista apposita.

---

[8]La resistenza di pull-down ha il compito di annullare le fluttuazioni elettromagnetiche che il pin potrebbe percepire. Con questa resistenza attiva, il pin non rileverà alcun valore significativo se non quello della pressione del pulsante, in questo modo evitiamo che il rumore ambientale causi malfunzionamenti.

[9]Acronimo di Compute Unified Device Architecture, è un'architettura hardware per l'elaborazione parallela creata da NVIDIA. Tramite questo ambiente di sviluppo si possono scrivere applicazioni capaci di eseguire calcolo parallelo su GPU delle schede video NVIDIA.

La lista appena creata viene passata ad una funzione che farà illuminare i rispettivi led.

Il tempo impiegato per l'esecuzione dell'intero ciclo sul raspberry di cui sono state elencate le caratteristiche nella sezione 3.1.1 con priorità massima assegnata al processo si aggira sui 4 secondi, che non lo rende utilizzabile come dispositivo di orientamento e sostituzione sensoriale.

Infine, quando viene premuto il pulsante di stop, viene generato un interrupt che cambia il valore del flag del loop rendendolo false, in questo modo al controllo successivo si uscirà dal ciclo. Appena usciti viene disattivata la preview della fotocamera, stoppati i pin di uscita PWM, puliti i GPIO e si fa partire un thread apposito con lo script restart_program.py.

4.4.4 Programma funzionante tramite server

Questa è la controparte di vision.py funzionante tramite server. La prima parte di inizializzazione è analoga a quella di vision.py ma molto più leggera e veloce, non dovendo inizializzare tutto il materiale necessario a monodepth2, inoltre viene gestito anche il pulsante di descrizione ambientale con la funzione set_describeFlag_ON(), che cambia semplicemete il valore del flag di lettura. Viene inoltre definita la funzione play_voices_in_background() che fa una chiamata reader.py.

Per dare un'idea di quanto questa versione sia più leggera rispetto a vision.py, il tempo impiegato per arrivare all'inizio del ciclo è circa 0.2 secondi (contro gli 1.7-1.8 di vision.py) rendendo necessaria l'attesa di circa 1.8 secondi.

Nel ciclo viene quindi scattata una foto, viene inviata una richiesta POST al server con il file dell'immagine e se si riceve un response code 200 si và avanti con il programma mandando una richiesta GET e si attende la risposta di un json contente sia i valori da mandare in output sui led, sia gli oggetti riconosciuti da YOLOv3 (se presenti, altrimenti viene inviata una lista vuota).

La durata del ciclo comprendente la stima delle distanze e l'analisi da parte di Yolo, più l'invio del file di immagine[10] è diminuita notevolmente rispetto a quella di vision.py, passando da 4 secondi a 1.6 (ulteriormente migliorabili con la compilazione di OpenCV con CUDA). Se si decidesse di non utilizzare mai l'analisi di YOLO il ciclo impiegherebbe circa 0.8 secondi sul server presente in 3.1.5.

Se viene premuto il pulsante di lettura, il flag describeFlag diventa True, viene scritta su file la lista degli oggetti trovati con YOLO e viene istanziato un thread che esegue il programma reader.py insieme a un deamon che ne controlla lo stato.

L'utilizzo dei thread in questo progetto è nato proprio in questo punto, dove era necessario poter riprodurre dei file audio contemporaneamente all'invio e alla ricezione dati verso il server e all'output sui led.

Nel caso della pressione del pulsante di stop il comportamento è completamente analogo a quello in vision.py (terminazione preview, stop dell'output sui pin, pulizia GPIO e chiamata a restart_program.py).

4.4.5 Lettore della descrizione ambientale

Questo script aveva come scopo principale quello di leggere file mp3 senza fermare la rilevazione dell'ambiente né interrompere la fase di analisi.

Per questi motivi si è scelto di utilizzare uno thread apposito per la lettura. Viene utilizzato il lettore da linea di comando mpg321, leggero e semplice. Viene poi aperto il file JSON con le classi trovate e per ognuna di esse si legge il file al relativo path.

Una volta lette tutte le classi nel file il processo termina.

[10]L'immagine è stata inviata attraverso la stessa rete del raspberry. Non avendo un server esterno a disposizione non è stato possibile fare delle prove in condizioni più "reali"

4.4.6 Analisi profondità e riconoscimento tramite YOLO su server

Anche qui, come in vision.py, nella prima parte di codice si importano tutti i pacchetti necessari al funzionamento di monodepth2, in più vengono anche prese quelle per il funzionamento di flask, che ci permette di creare un server in grado di gestire richieste di tipo restful. A differenza di vision.py, questo programma analizza anche l'ambiente circostante in cerca di oggetti, animali o altre entità tra le più comuni con YOLOv3.

Per il suo funzionamento non serve importare altri pacchetti, esso si basa infatti su OpenCV che veniva già usato da monodepth2.

Si procede quindi al controllo della disponibilità di CUDA per monodepth2, si inizializza tutto il necessario al suo funzionamento e si procede a fare lo stesso con Yolo, il quale deve però avere già i pesi del modello. Viene impostata una confidenza minima affinché gli oggetti trovati vengano poi effettivamente ritornati da Yolo, in questo caso la confidenza minima è 0.5. YOLOv3 attribuisce ad un iniseme di pixel un valore da 0 a 100% che simboleggia con quanta sicurezza quei pixel sono un certo oggetto. In figura 4.4 possiamo notare come con yolo si possano disegnare dei bordi intorno agli oggetti e la loro classe di appartenenza, insieme alla confidenza.

Questa parte può essere assimilabile a quella di inizializzazione, dato che viene eseguita una sola volta.

Una volta terminata vengono creati due percorsi all'interno del server, il primo dei quali per la raccolta dell'immagine, che prende semplicemente un file in POST e lo deposita nel path che poi andremo ad usare per l'analisi.

Il secondo è quello che viene contattato per l'analisi.

Quando arriva la richiesta GET il server inizia a processare l'immagine con monodepth2. Il funzionamento è esattamente lo stesso che si ha in vision.py, a cambiare è solo la velocità di esecuzione, incrementata sia dal processore più potente del server che dalla presenza di una scheda grafica (con server si intende il computer in 3.1.5). I risultati vengono quindi scritti su una lista detta "values". Prima di inviare la lista l'immagine viene

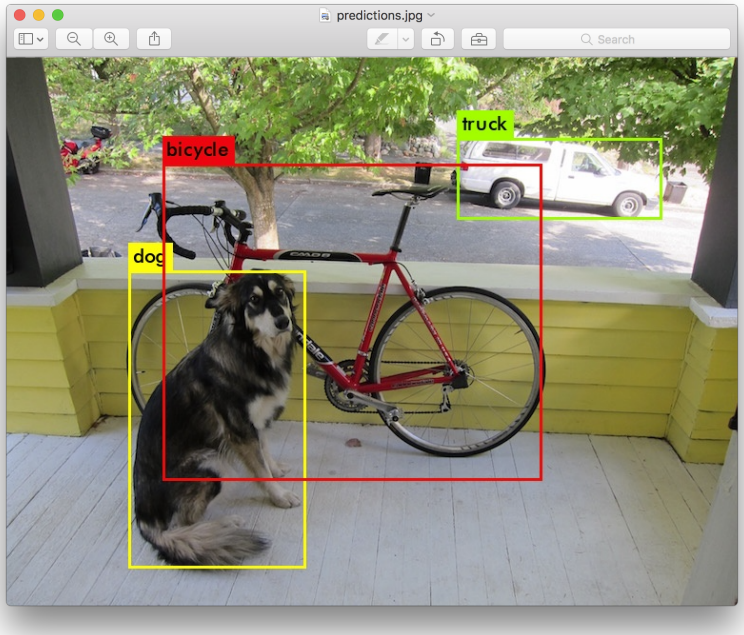

Figura 4.4: Esempio di analisi da parte di YOLOv3

analizzata anche tramite yolo che disegna i bounding boxes intorno agli oggetti trovati, e salva in una lista il nome delle classi trovate. Yolo necessita inoltre di un'immagine quadrata per funzionare in maniera ottimale. Essendo le dimensioni di quella che scatto di 640x192, provvedo a disegnare dei bordi in modo da far diventare l'immagine 640x640 px.

La parte in cui si disegnano i bounding boxes non è necessaria al funzionamento in un modello da utilizzare per la sostituzione sensoriale, ma è stata utile per controllare il corretto funzionamento del programma, si è quindi deciso di mantenerla visto che il tempo impiegato era trascurabile. Inoltre, se si riuscisse a compilare OpenCV su scheda grafica si avrebbero significativi miglioramenti, che permetterebbero di far andare yolo più veloce, fino al 1549% in più[5]. Una volta analizzata l'immagine, in un'altra lista, detta lables, vengono aggiunti i nomi di tutte le classi degli oggetti trovati.

Le due liste vengono quindi convertite in JSON e inviate al raspberry.

Capitolo 5

Prove sperimentali

In questo capitolo andremo ad analizzare le prove fatte con il dispositivo e le varie modifiche che sono state apportate per rendere più semplice il controllo dello stato del sistema e la raccolta dati.

È inoltre importante ricordare che per i test fatti all'esterno è stato impossibile testare il funzionamento del programma comunicante col server, per questo motivo è stato utilizzato il sistema stand-alone su raspberry.

5.1 Modifiche

Per rendere più utilizzabile il sistema e rendersi conto dello stato in cui ci si trova, è stato aggiunto un led, che si accende per una frazione di secondo, segnalando l'imminente acquisizione di un immagine da parte della fotocamera.

Una seconda modifica che è stata necessaria consiste nel mantenimento di uno storico delle immagini acquisite. In particolare il sistema originariamente conservava in memoria solamente l'ultima immagine acquisita; è stata quindi apportata una modifica che va a salvare le immagini rendendole tutte dispobili per un futuro recupero e utilizzo.

5.2 Prove sul campo

Prima di andare a testare il dispositivo sul campo ho organizzato un itinerario, e scelto alcuni punti in cui fare delle acquisizioni di prova. In quei punti ho misurato la distanza dagli oggetti che venivano ripresi con un metro al laser (Stanley TLM100i) per controllare la consistenza delle misure. Il sistema si è rivelato abbastanza preciso per strada, mostrandosi affidabile soprattutto quando posizionato a livello del petto con la camera perpendicolare al terreno. Ipoteticamente si potrebbe quindi scegliere di costruire una versione più utilizzabile del progetto da far indossare a livello del petto, in modo da offrire una superficie stabile per la camera (che auspicabilmente non farebbe foto eccessivamente mosse) e abbastanza sensibile da poter permettere una buona risoluzione per effettuare la sostituzione sensoriale.

Andremo ora a mostrare i vari scatti fatti nei punti in cui si conosceva la distanza. In ogni immagine sono presenti dei riquadri o delle strisce di scotch di carta che simboleggiano il punto in cui è stata presa la misura e oguna di esse è affiancata dalla rispettiva stima delle distanze. Per chiarezza andiamo a ripetere nuovamente che l'immagine delle predizioni è un'immagine in bianco e nero dove più i pixel sono considerati vicini,e più sono chiari.

In figura 5.1 è presente un piccolo quadrato al centro dell'immagine, in basso: ci troviamo a circa 7,67 metri da esso. L'immagine di predizione è abbastanza precisa ma la parte a sinistra, con il palo, risulta più vicina rispetto a quella che si ha di fronte. Questo errore potrebbe essere dovuto al fatto che il dataset su cui è stata addestrata la rete usata da monodepth è stato acquisito da macchine in movimento, che come ambiente principale hanno delle strade, spesso ben delimitate e quindi ambientazioni molto diverse da quella in figura 5.1.

In figura 5.2 siamo di fronte ad un cabinato a circa 6.8m di distanza. L'immagine di predizine delle distanze è buona e vengono anche visualizzati i paletti che sostengono la rete. purtroppo però si è troppo distanti per vedere quest'ultima. Si riescono anche a vedere gli alberi e più lontani dei paletti.

Figura 5.1: Immagine di test a 7.67m di distanza.

Figura 5.2: Immagine di test a 6.8m di distanza.

Nella figura 5.3 siamo di fronte alla stessa cabina ma a 4.43m di distanza e si può notare come tutti i colori siano più accesi. Anche qui alcuni dei problemi di visualizzazione potrebbero essere dovuti alla conformazione ambientale.

Figura 5.3: Immagine di test a 4.43m di distanza.

In figura 5.4 sono stati presi due punti, uno sull'angolo a destra del cabinato in figura, a circa 4.8m e l'altro a in basso a sinistra della stessa struttura a 6.2m. Si noti che in questo caso la struttura è la stessa ma si è in prospettiva. La stima delle distanze è tuttavia molto buona e si nota come i colori sfumino verso tonalità scure man mano che ci si allontana dall'osservatore.

Andremo ora a mostrare tre immagini che hanno riportato buone stime delle distanze; in quella in figura 5.5 si riescono a vedere i contorni di un'auto, nella seconda il paesaggio viene trasformato bene ma una nuvola viene probabilmente interpretata come la chioma di un albero (visti i colori dell'immagine con le stime molto simili a quelli del pino lì

Figura 5.4: Immagine di test con punti in prospettiva a 4.8 e 6.2 metri di distanza.

vicino), nella figura 5.7 si vede bene la strada, delimitata dalle recinzioni, anch'esse ben visibili.

Figura 5.5: Immagine di test con buone predizioni.

Figura 5.6: Immagine di test con buone predizioni ma con la nuvola mal interpretata.

Figura 5.7: Immagine di test con buone predizioni.

5.2.1 Le analisi fatte da YOLOv3

YOLOv3 solitamente funziona molto bene, tuttavia non è perfetto e talvolta confonde o non riconosce bene oggetti. Di seguito riporto qualche acquisizione di cui è stata fatta

l'analisi. Ricordando che yolo ha bisogno di immagini quadrate per la massima accuratezza, è stato perciò necessario aggiungere dei bordi neri che fanno diventare l'immagine quadrata.

La figura 5.8 è l'analisi fatta con Yolo della figura 5.6, tuttavia, yolo non riesce a riconoscere il prefabbricato, confondendolo per una serie di frigoriferi. Probabilmente con un buon display tattile, con l'allenamento e con l'abitudine una persona riuscirebbe da sola a capire cos'ha davanti, soprattutto con oggetti di dimensioni notevoli come appunto dei prefabbricati.

Figura 5.8: Analisi con yolo dello scatto in figura 5.6.

La figura 5.9 è l'analisi da parte di yolo della figura 5.5 e qui yolo riesce a riconoscere

la macchina, mentre in figura 5.10 viene evidenziata correttamente una persona.

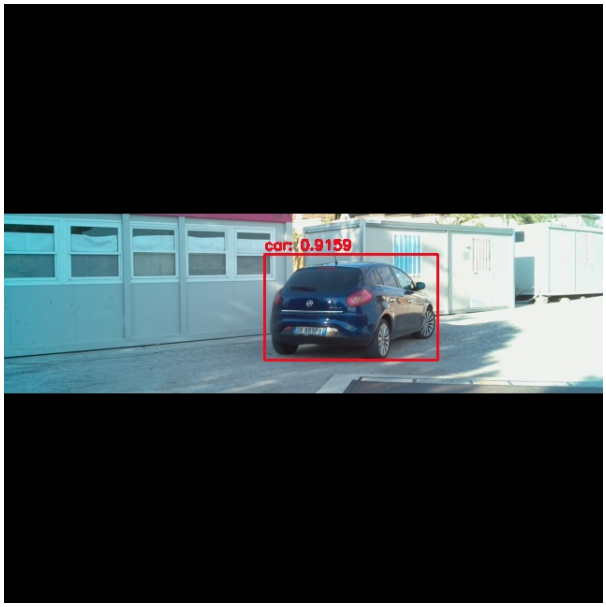

Figura 5.9: Analisi con yolo dello scatto in figura 5.5.

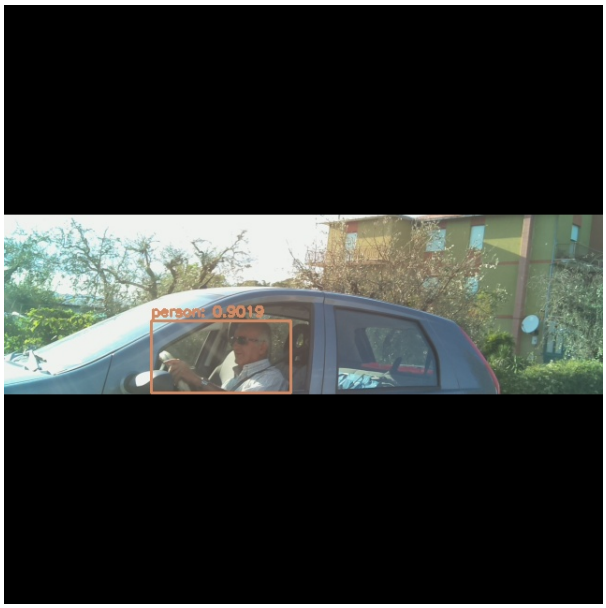

Figura 5.10: Yolo riesce a riconoscere una persona.

Capitolo 6

Conclusioni

L'obiettivo di questo lavoro era progettare e costruire un dispositivo di sostituzione sensoriale low-cost, innovativo, discreto ed affidabile, che permettesse a non vedenti e ipovedenti di spostarsi negli ambienti esterni, senza il bisogno di bastoni o altri tipi di ausili di orientamento e di percepire il senso della vista mediante altre modalità, come ad esempio il tatto.

Non avendo né grandi quantità di fondi né di tempo, è stato necessario ridimensionare il progetto, ad esempio al posto di un display tattile o di motori di vibrazione si è dovuto scegliere un insieme di led per la visualizzazione dei risultati, inoltre, la maggior parte delle scelte a livello hardware sono state vincolate dal prezzo e dalla disponibilità dell'università, che ha fornito pezzi come: breadbord, jumper, led, resistenze e bottoni.

L'idea del progetto era quella di dare un senso di spazio tridimensionale agli utilizzatori, grazie al quale sarebbero stati capaci di orientarsi in autonomia e di capire la tipologia di oggetti che avevano davanti.

La fase di progettazione è iniziata con la ricerca di librerie per la stima delle distanze tramite immagini singole, ovvero non stereo.

Una volta trovate le librerie giuste è iniziata la fase di progettazione software, che può essere scomposta in tre macro-compiti: acquisizione dell'immagine, analisi, output dei risultati. Per l'acquisizione è stata scelta una camera, per l'analisi e stima delle distanze

è stata scelto di usare monodepth2, e per l'output di esempio i led. L'idea era di eseguire l'intero programma sul dispositivo indossabile. Come dispositivo si è seclto il Raspberry Pi 4 model B, ma la creazione di ambienti virtuali con le giuste caratteristiche e con i pacchetti aggiornati alle versioni raccomandate per il funzionamento di monodepth2 si è rivelata molto più complicato del previsto.

L'utilizzo di gestori di ambienti, come "miniconda", non ha portato benefici, creando spesso conflitti tra i pacchetti usati. Si è quindi deciso di usare l'ambiente virtuale consigliato da Python, ovvero venv, e installare manualmente tutti i pacchetti, cercando di scaricare le versioni disponibili e aggiornate per Raspberry.

OpenCV, un'altra libreria usata per lavorare sulle immagini, è stata invece compilata perché non disponibile tramite pip su Raspberry.

Una volta preparato l'ambiente, scritto il programma e costruito il circuito sono state fatte delle prove per testarne il corretto funzionamento e si è cercato di velocizzare la fase di analisi assegnando la massima priorità al processo che, su Raspberry, impiegava troppo tempo. Nonostante i miglioramenti, il tempo impiegato era comunque eccessivo e si è deciso di spostare la fase di analisi su un server che, all'interno della stessa rete, impiegava meno di un quarto del tempo usato dal Raspberry per il completamento delle analisi.

Essendo i tempi impiegati col server abbastanza buoni si è scelto di implementare un'ulteriore funzionalità: la descrizione ambientale tramite YOLOv3. Dopo la fase di analisi, il Raspberry avrebbe dovuto aprire dei file audio coi nomi delle classi di oggetti riconosciuti.

Anche qui il Raspberry ha creato problemi, in particolare alcune librerie audio non funzionavano correttamente, spesso l'audio non veniva riprodotto e i file audio venivano tagliati nella parte iniziale.

Una volta risolti i problemi discussi sopra, sono stati aggiunti dei pulsanti, che consentissero un'utilizzo più comodo e intuitivo del dispositivo.

È stato molto interessante portare avanti un progetto nella quasi completa autonomia

in un ambito molto di nicchia e poco conosciuto, vedendo il progetto crescere e migliorare di giorno in giorno.

Questo è stato anche il primo progetto che ho sviluppato in Python, mi ha permesso di capire realmente come linguaggi meno performanti ma più di alto livello rispetto a linguaggi compilati come il C, possano velocizzare la fase di sviluppo, consentendo la creazione di programmi funzionanti in poco tempo e poche linee di codice.

Al fine di rendere realmente utilizzabile il dispositivo, andrebbero sostituiti i led con delle matrici tattili o con dei motori di vibrazione.

Si potrebbe inoltre creare una versione funzionante solo tramite server, che permetterebbe di utilizzare un microcontrollore con dimensioni e consumi ridotti, che sia più utilizzabile e discreto.

Come ultimo miglioramento si potrebbe utilizzare un server più potente, con una scheda grafica Nvidia più performante e si potrebbe compilare OpenCV su scheda video, in modo da velocizzare l'analisi sia da parte di monodepth2 che di YOLOv3.

Bibliografia

[1] Prof. Giuseppe Boccignone. *La percezione dello spazio: stereo e movimento*. URL: https://docplayer.it/58279175-La-percezione-dello-spazio-stereo-e-movimento.html.

[2] Kathryn Dumper. *Sensation versus Perception*. URL: https://bit.ly/2H0hv2B.

[3] David Eagleman. *Can we create new senses for humans?* URL: https://www.ted.com/talks/david_eagleman_can_we_create_new_senses_for_humans.

[4] The Guardian. *Brain implant restores partial vision to blind people*. URL: https://bit.ly/3iF962u.

[5] Adrian Rosebrock. *OpenCV 'dnn' with NVIDIA GPUs: 1549% faster YOLO, SSD, and Mask R-CNN*. URL: https://www.pyimagesearch.com/2020/02/10/opencv-dnn-with-nvidia-gpus-1549-faster-yolo-ssd-and-mask-r-cnn/.

[6] Wired Science. *Paul Bach-y-Rita and Neuroplasticity*. URL: https://www.youtube.com/watch?v=7s1VAVcM8s8&ab_channel=ChristianG.

[7] Wikipedia. *Neuroplasticity*. URL: https://en.wikipedia.org/wiki/Neuroplasticity.

[8] Wikipedia. *Paul Bach-y-Rita*. URL: https://en.wikipedia.org/wiki/Paul_Bach-y-Rita.

[9] Wikipedia. *Sensory substitution*. URL: https://bit.ly/3iqFJRn.

Ringraziamenti

Vorrei ringraziare prima di tutto la mia famiglia, che mi ha aiutato e sostenuto, tramite sforzi e sacrifici, non solo in questi tre anni ma durante tutta la vita. Grazie per essermi stati vicini quando non lo meritavo e per avermi permesso di intraprendere questo percorso.

Ringrazio gli amici con cui ormai ho passato una vita insieme, qualcuno mi accompagna fin dall'asilo, alcuni dalle medie o dai primi anni di superiori ma tutti mi conoscete benissimo. Non sono bravo con le smancerie, né coi contatti fisici, ma sapete che vi voglio un sacco di bene anche se non sono sempre lì a dimostrarvelo. Grazie per le seratine tranquille, per i sushi insieme, per le vacanze, per le migliaia di uscite e per essermi stati accanto quando ne avevo più bisogno, non riesco ad immaginare una vita senza di voi.

Ringrazio i coinquilini, sono stato fortunato a poter vivere insieme a voi, più o meno a lungo. Conoscervi, parlare di culture diverse, cene, carbonare, intossicazioni, film insieme, le lunghe chiacchierate, i basket, le lamentele e per aver condiviso lo stesso tetto, è stato davvero fantastico. Senza di voi la vita a Cesena non sarebbe stata la stessa.

Ringrazio tutte le amicizie fatte all'università, ormai non conto neanche più tutte le persone fantastiche che ho conosciuto. Tutti gli amici del corso, con cui ho condiviso gioie e dolori, esami e giornate di studio intenso. Tutti gli studenti erasmus, con cui non avrei mai pensato di legare così tanto. Grazie per tutti quanti i momenti belli e strani

che abbiamo passato insieme.

Ringrazio i professori, in particolar modo il Professor Raffaele Cappelli, che mi ha aiutato durante il periodo di tirocinio e tesi. Spero di aver suscitato interesse per il progetto e di non essere stato troppo noioso durante i lavori.

Ringrazio una persona speciale, che mi ha tenuto compagnia durante tutta la quarantena e che mi ha motivato a studiare di più durante quel periodo di confinamento forzato. Grazie a te stare in camera davanti al pc era diventata la cosa più bella che potessi fare.

Ringrazio tutte quelle persone che mi sono state accanto nei momenti di sconforto e depressione, ai quali non sono minimamente abituato. Grazie perché sò di poter contare su di voi.

Infine ringrazio tutte le persone che ho incontrato, belle o brutte che siano, e che non rientrano nei ringraziamenti precedenti. Grazie per avermi permesso di diventare la persona che sono oggi.