

true

Contents

Progetto Ultrasonic Vision	3
Il sistema Ultrasonic Vision	4
Architettura	4
Funzionamento del sistema	4
Propagazione degli ultrasuoni	5
Risoluzione spaziale del sistema di misura	6
Limitazioni note	6
Il prototipo Ultrasonic Vision 1.0	7
Requisiti	7
Hardware	8
Misuratori di distanza ad ultrasuoni HC-SR04	8
Massimo numero di sensori gestibili	8
Numero di sensori utilizzati	9
Software	9
ultrasonic-vision.py	9
Modalità riconoscimento	10
Modalità Addestramento	11
Gestione avvio della misurazione	11
Dati Prodotti da sistema di misura	11
Configurazione a sette sensori	11
Assemblaggio e cablatura	12
Riconoscimento degli oggetti	12
Classificatore Locale (Edge Computing)	12
Classificatore Remoto (Cloud Computing)	12
Posizionamento degli oggetti e dataset di training	13
Dati training 3D	13
Posizionamento oggetti	13
Interfacciamento moduli HC-SR-04 / HC-SR-04+	14
Collegamento alla morsettiera	14
Cablaggio pannello sensori superiore	14

Test del prototipo nella configurazione a sette sensori	15
Analisi risultati del test	15
Analisi dati acquisiti senza oggetto presente	16
Test modello AWGN (Additive White Gaussian Noise)	16
Analisi dati esperimenti con oggetti (Caso di studio esperimento “BEAN_CAN”)	17
Stime delle distanze orizzontali	17
Stime delle distanze verticali	18
Conclusioni	18
Misuratori di distanza ad ultrasuoni HC-SR04	19
Pinout HC-RS04	19
Altro materiale sui moduli HC-SR04 e sensori ad ultrasuoni in generale	20
Basic Raspberry setup	20
Hardware per configurazione iniziale	20
Raspian Install/Ugrade Rasperry OS (Raspian)	21
Upgrade Raspberry OS	21
Pulsante di avvio della misurazione	21
Pull Up /Pull down su input digitali	21
gestione software I/O digitale	22
Sessione remota su Raspberry da PC di sviluppo	23
Sessione remota SSH verso Raspberry	23
Installazione Python3.8 su Raspian Buster (dicembre 2020)	24
Installazione di Python 3.8	24
1) scaricare i sorgenti e compilare	24
2) Install	24
3) Remove the files	25
4) Aprire il file .bashrc con Nano	25
5) rileggere il file .bashrc	25
6) Verificare gli alias	25
Installare pip, wheel	25
Ultrasonic sensors test	26
Sessione Desktop Remoto da Windows a Raspian (Raspberry OS)	26
Installazione server RDP su RaspberryOS	26
Modificare i permessi del file di log	27
Inferenza con modelli ML su Raspberry con ONNX Runtime per applicazioni di Edge AI	27
Installazione ONNX Runtime	28
Prerequisites	28
Installazione	29

Sviluppo di un modello di classificazione degli oggetti	30
Analsi delle singole features	30
Rimozione outliers	31
Conclusioni e futuri sviluppi	31
Addestramento manuale dei modelli di classificazione con libreria SciKit Learn	31
Azure AutoML	32
Dataset	33
Servizio Automated Machine Learning	33
Risultati ottenuti	33
Metriche del modello	33
Deploy come web service	36
Azure ML SDK Installation	37
Risultati ottenuti	37
Futuri Sviluppi Ultrasonic - Vision	38
Connettività ed invio dati verso storage esterno	38
Trigger automatico	38
Caratterizzazione dei sensori ad ultrasuoni utilizzati	38
Rappresentazione grafica 3D dell'oggetto	39
Ridurre il tempo necessario ad eseguire una misura	39
Modello avanzato per riconoscimento oggetti	39
Pipeline di miglioramento continuo del classificatore	39
Calibrazione del sistema	39
Versione "Industrial grade" del sistema	39
Versione 2.0: rete di sensori distribuiti realizzata con moduli WiFi	
ESP82266 che scambiano dati tramite MQTT	40
Strumenti e tecnologie utilizzate	40

Progetto Ultrasonic Vision

Ultrasonic Vision è un sistema di acquisizione dati in grado di rilevare la presenza di un oggetto, determinarne la posizione e riconoscere il tipo oggetto. Il sistema utilizza dei misuratori di distanza ad ultrasuoni e dei modelli di classificazione addestrati con tecniche di machine learning.

L'obiettivo del progetto è quello di realizzare un primo prototipo funzionante al fine di individuare le criticità che dovranno essere affrontate nella progettazione del prodotto finale.

Il sistema Ultrasonic Vision

Il sistema adotta una architettura di tipo *Edge computing*: in cui l'elaborazione è eseguita quanto più possibile vicino alla sorgente dei dati. Anche la componente di intelligenza è eseguita sul campo per la parte di inferenza (*Edge AI*), mentre l'addestramento dei modelli è eseguito su cloud (*cloud computing*) oppure su server raggiungibili tramite rete LAN/WAN (*fog computing*) per sfruttare le maggiori risorse computazionali disponibili.

Architettura

L'architettura di riferimento è a quattro livelli

image-20201229054638716

1. sensori ed attuatori che interagiscono direttamente con il mondo fisico sul campo
2. gateway di campo (*edge*) che raccoglie i dati di tutti i sensori, esegue le elaborazioni a livello di edge computing e gestisce le comunicazioni con l'esterno
3. sistema informativo dell'impianto produttivo o dell'organizzazione (MES, ERP, SCADA, database, servizi enterprise,)
4. applicazione e servizi su cloud (*cloud computing*) usate principalmente per l'addestramento dei modelli di machine learning e lo stoccaggio dei dati a lungo termine

Funzionamento del sistema

La componente di campo utilizza dei misuratori di distanza ad ultrasuoni. Ogni misuratore è indipendente ed è composto da un emettitore di impulsi ad ultrasuoni e da un ricevitore. La distanza del bersaglio viene stimata misurando il tempo (*Time Of Flight - TOF*) trascorso tra l'emissione dell'impulso e la ricezione del primo eco riflesso dal bersaglio.

I misuratori sono fissi ed in posizioni note. L'insieme delle distanze stimate rispetto ai diversi misuratori, combinato con le informazioni sulla configurazione geometrica del sistema, viene utilizzato per stimare la posizione del bersaglio ed riconoscere il tipo di oggetto. L'aggregazione e la prima elaborazione dei dati provenienti dai sensori viene eseguita localmente a livello di *edge* (*edge computing*). A questo livello viene elaborata la stima della posizione e può essere eseguita anche la classificazione dell'oggetto. I dati acquisiti vengono poi inviati alla componente *cloud* per elaborazioni di secondo livello, miglioramento del modello di classificazione e memorizzazione a lungo termine (*cloud computing*).

Il riconoscimento dell'oggetto viene eseguito utilizzando un classificatore multi-classe addestrato con dati ottenuti da misurazioni precedentemente eseguite con

il sistema nella stessa configurazione (numero e posizione dei sensori) su oggetti noti a priori. Il classificatore può essere eseguito localmente sul dispositivo di campo che gestisce i sensori, come web server su rete locale oppure come web server remoto.

In generale la configurazione geometrica del sistema ed il numero di sensori devono essere scelta in base ai requisiti di simmetria e risoluzione che si vogliono soddisfare. Nel caso del prototipo realizzato per questo progetto non erano stati posti particolari vincoli e quindi sono stati utilizzati i sensori che erano disponibili in laboratorio, sperimentando diverse configurazioni geometriche.

Propagazione degli ultrasuoni

Nel caso di oggetti con dimensioni molto maggiori della lunghezza d'onda all'interfaccia ARIA-OGGETTO le onde sonore incidenti generano un'onda riflessa. La grande differenza di impedenza acustica all'interfaccia "ARIA - ACQUA" provoca una riflessione quasi totale, però solo l'onda riflessa che raggiunge il sensore è utile per stimare la distanza. La direzione dell'onda riflessa dipende dall'angoli di incidenza. (vale la legge di Snell).

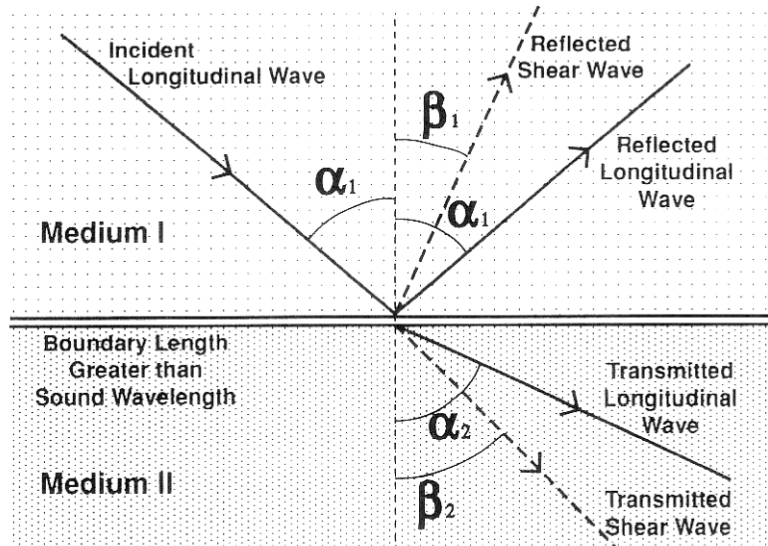


Figure 1: Riflessione Ultrasuoni

Il suono in aria si propaga a circa 330m/s. Nel caso dei sensori HC-SR.04 gli impulsi hanno una frequenza di 40Kz e quindi una lunghezza d'onda teorica pari a $330/40000 = 0.00825$ m (8mm).

- Oggetti con dimensioni dell'ordine di grandezza di alcuni millimetri o di qualche centimetro possono generare fenomeni di diffrazione che possono interferire con la stima della distanza.
- oggetti con superfici “complesse” o in generale non approssimabili come superfici piane di dimensioni molto maggiori di 8mm (ad esempio con parti in rilievo, bordi, superfici con raggio di curvatura dell'ordine dei 10mm, ...) potrebbero non essere compatibili con il tipo di sensore utilizzato
- oggetti con superfici curve o planari, ma non parallele al piano frontale del sensore possono riflettere l'impulso sonoro in una direzione diversa da quelle dove si trova il sensore e generare tre tipi di anomalie:
- eco non rilevato
- eco rilevato dopo riflessioni multiple e distanza molto maggiore di quella reale, ma nel range standard del sensore
- eco rilevato dopo riflessioni multiple e distanza molto maggiore di quella reale e fuori dal range standard del sensore

Risoluzione spaziale del sistema di misura

In linea di principio l'onda ultrasonora generata dal trasduttore si sposta in linea retta fino a quando incontra un limite materiale. Al di fuori della zona di “campo vicino” il fronte d'onda aumenta di diametro e diverge come il fascio di un proiettore. L'angolo di diffusione del fascio di una sonda può essere calcolato nel seguente modo 1:

BeamSpread_static BeamSpread_static NearFieldCalculation

Nel caso dei sensori HC-SR-004 il campo acustico lontano è quindi approssimabile con un cono con apertura di circa 15°. L'area di rilevamento del sensore ad una distanza d è quindi pari a $d \cdot \sin(15)$:

- un cerchio di raggio 5 cm a 20 cm di distanza
- un cerchio di raggio 13 cm a 50 cm di distanza
- un cerchio di raggio 26 cm a 100 cm di distanza

Limitazioni note

Il numero dei sensori utilizzabili (e quindi la risoluzione spaziale del sistema) è limitato dal numero di linee di input/output digitale (*GPIO*) del microcontrollore utilizzato e dal tempo disponibile per effettuare la misura (i misuratori vengono accesi singolarmente in sequenza). Solitamente i sensori ad ultrasuoni richiedono 2 linee (anche se alcuni modelli ne richiedono solo una). Nel caso del prototipo sono stati utilizzati dei sensori HC-SR04 che richiedono 2 GPIO

e un Raspberry PI 3 mette a disposizione 24 GPIO quindi il prototipo, senza circuiteria aggiuntiva per multiplexing, può gestire fino a 12 sensori.

La fisica degli ultrasuoni impone dei limiti alla massima risoluzione spaziale che è possibile ottenere e sul tipo di oggetti e materiali compatibili con il sistema di misura:

- il bersaglio deve avere un'area equivalente parallela al piano frontale di almeno uno dei sensori dell'ordine di grandezza dei 10 cm
- superfici complesse (bordi, elementi in rilievo, ...) con dimensioni dell'ordine di grandezza di alcuni millimetri potrebbero interferire con la misura

Il raggio d'azione del sistema dipende dalla configurazione geometrica del sistema e dalle caratteristiche dei misuratori utilizzati. Molti misuratori commerciali a basso costo hanno una portata operativa di 2-3 metri anche se ne esistono alcuni con portata fino a 7 metri. Ad esempio disponendo i misuratori ai vertici e lungo i lati di un quadrato si potrebbe agevolmente coprire un'area di almeno 2 mq anche con misuratori di fascia economica. Un'altra possibilità è quella di disporre i misuratori lungo una circonferenza di diametro minore o uguale alla portata massima dei dispositivi.

La classificazione degli oggetti dipende sia dalla geometria dell'oggetto che dalla posizione dell'oggetto all'interno dell'area di rilevamento. I classificatori addestrati fino ad ora hanno performance molto elevate se gli oggetti vengono posizionati esattamente come durante la fase di training, ma non riescono a riconoscere l'oggetto in maniera indipendente dalla posizione. Questo problema è affrontato nel progetto Ultrasonic Object Recognition che ha come obiettivo la realizzazione di un sistema di riconoscimento degli oggetti basato sui misuratori ad ultrasuoni utilizzando tecniche di machine learning e deep learning avanzate.

Il prototipo Ultrasonic Vision 1.0

Lo scopo di questo prototipo è quello di verificare la fattibilità ed evidenziare le criticità del sistema

Requisiti

1. il sistema deve riuscire a rilevare la presenza di un oggetto all'interno di un'area delimitata e stimarne la posizione rispetto ai misuratori di distanza
2. il sistema deve riconoscere il tipo di oggetto presente (classificazione multi-classe)

3. il prototipo deve essere realizzato con HW standard, a basso costo e facilmente reperibile. Possibilmente solo moduli già pronti, senza schede custom.
4. il sistema deve essere ben documentato e utilizzabile ad esempio per esercitazioni di laboratorio.

Hardware

Per l'assemblaggio del prototipo sono stati utilizzati:

- un Raspberry PI 3 completo di alimentatore
- misuratori di distanza ad ultrasuoni HC-SR04/HC-SR04+ (numero variabile in base alla configurazione geometrica desiderata. Il prototipo è stato testato con configurazioni a 4 e 7 sensori)
- kit di cavi Dupont per breadboard
- almeno 3 metri di cavo per sistemi di allarme a 4 fili
- tre breadboard piccole (half size)
- un pulsante
- sette resistenze 18Kohm
- sette resistenze 10Kohm

Tutti i componenti facilmente reperibili dai rivenditori di elettronica oppure su Amazon eBay e simili. Per un elenco più esteso di possibili rivenditori vedi Guida all'acquisto di componenti e strumenti.

Costo complessivo stimato per i componenti elencati inferiore a 50€ compresi breadboard e cavi.

Misuratori di distanza ad ultrasuoni HC-SR04

Sul mercato esistono diversi misuratori di distanza ad ultrasuoni, destinati al mercato dei maker, con funzionalità e prestazioni sostanzialmente equivalenti. Per il prototipo sono stati utilizzati misuratori di distanza ad ultrasuoni tipo HC-SR04/SR04+ che erano disponibili in laboratorio.

Le prestazioni ed il funzionamento sono descritti nel capitolo dedicato ai moduli HC-SR04

Massimo numero di sensori gestibili

Ogni sensore richiede GND, VCC + 2 GPIO (trigger + echo). Su Raspberry 2 ci sono 24 GPIO pin disponibili. Senza adottare particolari accortezze questo sistema può supportare fino a 12 sensori. Il timing dei segnali sui pin “echo” e “trigger” del sensore ne permette volendo anche la gestione con un solo GPIO. In

questo modo il numero massimo di sensori che un singolo raspberry può gestire sale a 24.

Numero di sensori utilizzati

L'hardware utilizzato consente di utilizzare fino a 12 sensori. Al momento in laboratorio sono disponibili 5 sensori HC-SR04+ e 5 sensori HC-SR04. Per l'addestramento del classificatore è stata utilizzata una configurazione con sette sensori in modo da lasciare alcuni sensori disponibili per testare l'utilizzo di moduli remoti con microcontrollore ESP8266, mantenendo invariata la configurazione del sistema principale.

Software

Software sviluppato in Python: facile da scrivere, mantenere e debuggare. Attività di sviluppo e debug possono avvenire direttamente sul Raspberry (in sessione locale oppure in sessione remota) senza bisogno di altri strumenti o ambienti di sviluppo.

Per il prototipo della versione 01 sono stati sviluppati:

- Modulo FakeRPi per eseguire il software senza l'hardware di acquisizione dati
- Applicazione console ultrasonic-vision (che include la modalità di simulazione per esecuzione senza hardware)

ultrasonic-vision.py

Il software di controllo del sistema di misura è stato sviluppato in Python 3. Lo script principale da eseguire sul sistema in produzione è l'applicazione console ultrasonic-vision.py

```
ultrasonic-vision.py <key> <scoring-uri> [<training-label>]
```

Parametro	Descrizione
key	API key per l'utilizzo del webservice REST che esegue il modello di classificazione. Obbligatorio.
scoring-uri	URI dell'endpoint pubblico del webservice REST del classificatore. Obbligatorio.
training-label	Label associata alle distanze stimate dai sensori. Se questo parametro è presente l'applicazione funziona in modalità <i>addestramento</i> . Opzionale.

L'applicazione può essere usata in tre modalità:

1. **modalità addestramento** in cui viene passato come parametro la "label" associata all'oggetto presente all'interno dell'area di rilevamento

```
bash    ultrasonic-vision.py asc12345567789 http://webseviceurl.azure.com/123-456-5567/score
BALL
```

2. **modalità riconoscimento** (produzione) in cui è il modello di classificazione ad assegnare la label in base alle distanze stimate dai sensori

```
bash    ultrasonic-vision.py asc12345567789 http://webseviceurl.azure.com/123-456-5567/score
```

3. **modalità riconoscimento con hardware simulato** in cui è il modello di classificazione ad assegnare la label in base, ma le stime delle distanze sono generate in maniera randomica.

```
ultrasonic-vision.py asc12345567789 http://webseviceurl.azure.com/123-456-5567/score
```

Questa modalità consente di eseguire l'applicazione su PC che non dispongono di periferiche di I/O digitale compatibili con la libreria GPIO presente nell'ambiente di runtime Python su Raspberry. Gli errori a runtime legati al modulo RPi.GPIO vengono evitati sostituendo il modulo originale con un semplice mockup creato per questo scopo denominato FakeRPi.GPIO

Per attivare la modalità simulazione decommentare l'import del modulo FakeRPi.GPIO

```
# import FakeRPi.GPIO as GPIO # real hardware sensors
```

e commentare quello della libreria reale

```
import RPi.GPIO as GPIO #emulated sensors
```

Modalità riconoscimento

Nella modalità riconoscimento l'applicazione svolge i seguenti compiti:

1. crea un file su disco locale per il salvataggio dei dati
2. entra in un loop infinito in cui se l'interruttore di avvio è chiuso esegue una misura
3. inizializza i sensori
4. acquisisce le stime della distanza del bersaglio restituita dai diversi sensori
5. Stampa sulla console le distanze stimate dai sensori
6. esegue la chiamata al web service del servizio di classificazione degli oggetti in cloud e stampa sulla console il risultato
7. esegue localmente il classificatore utilizzando le API Python del Runtime ONNX e stampa sulla console il risultato
8. scrive le distanze e il risultato della classificazione sul file della sessione di acquisizione dati

Modalità Addestramento

Nella modalità addestramento l'applicazione svolge i seguenti compiti:

1. crea un file su disco locale per il salvataggio dei dati
2. entra in un loop infinito in cui se l'interruttore di avvio è chiuso esegue una misura
3. inizializza i sensori
4. acquisisce le stime della distanza del bersaglio restituita dai diversi sensori
5. Stampa sulla console le distanze stimate dai sensori
6. scrive le distanze e la label passata come parametro sul file della sessione di acquisizione dati

Gestione avvio della misurazione

```
MAIN_TRIGGER_GPIO = 26
```

```
while True:
    if(FAKE_HW):
        mainTriggerState= True
    else:
        mainTriggerState= GPIO.input(MAIN_TRIGGER_GPIO)
```

Gestione evento “pulsante premuto” su Raspberry

Dati Prodotti da sistema di misura

L'applicazione ultrasonic-vision.py produce file in formato CSV. Ogni riga corrisponde ad una misura e contiene le seguenti informazioni:

- timestamp (in formato unix epoch) del inizio delle misura (tipo numerico intero)
- una colonna per ogni sensore presente contenente la distanza stimata in cm arrotondata alla seconda cifra decimale
- la label di classificazione associata alla misura

image-20210111130801736

Configurazione a sette sensori

Durante la realizzazione del prototipo sono state ipotizzate e testate diverse configurazioni geometriche. L'acquisizione dei dati utilizzati per l'addestramento del classificatore è stata eseguita nella configurazione a sette sensori con quattro sensori montati su due pannelli verticali orientati verso il centro dell'area di

rilevamento e tre sensori montati sul pannello orizzontale posizionato al di sopra l'area di rilevamento.

La portata utile dei sensori utilizzati è di circa 3 metri. Per ragioni logistiche la configurazione usata per l'addestramento ed il test è stata assemblata su una base di 40 x 70 cm in modo da rendere il sistema facilmente trasportabile.

configurazione_7_sensori

BALL_CENTER

Test della configurazione a sette sensori

Assemblaggio e cablatura

Il prototipo oltre al Raspberry utilizza solo un pulsante e sette moduli HC-SR04. Non sono necessari altri componenti esterni.

Per i dettagli sul collegamento dei moduli HC-SR04 al Raspberry vedi cablatura SR-HC04

Riconoscimento degli oggetti

Nella versione 1.0 del sistema sono stati sviluppati due diversi sistemi di classificazione, utilizzando due diversi approcci.

Classificatore Locale (Edge Computing)

In questo caso Raspberry Pi utilizzato per il prototipo dispone di un processore con architettura ARM32v7 per il quale sono disponibili i runtime delle principali librerie di machine learning e deep learning. Per il prototipo è stato selezionato ONNX che garantisce elevate prestazioni, interoperabilità con i principali framework di sviluppo e portabilità verso architetture diverse.

Classificatore Remoto (Cloud Computing)

Un secondo sistema di classificazione degli oggetti è stato pubblicato utilizzando il servizio Azure Machine Learning su un *container Docker* e reso accessibile tramite un web service REST con endpoint protetto da token di autenticazione.

Posizionamento degli oggetti e dataset di training

Dati training 3D

Ogni oggetto è stato posto approssimativamente al centro dell'altra di acquisizione dati, senza utilizzare riferimenti precisi per la posizione con lo scopo di rendere più robusto il riconoscimento da parte del classificatore. Per ogni oggetto acquisizione dati è stata ripetuta più volte dopo aver tolto e posizionato nuovamente l'oggetto con variazioni casuali di posizionamento.

Posizionamento oggetti

Sono stati sperimentati diversi posizionamenti degli oggetti all'interno del range dei sensori. I dati di training del classificatore sono stati acquisiti posizionando gli oggetti nella zona centrale in modo da avere potenzialmente letture significative da tutti i sensori presenti (compresi quelli ora non presenti nella configurazione a sette sensori)

posizionamento_oggetto

Acquisizione second dataset di training con configurazione a sette sensori e barriere parallele ai piani dei sensori

- SQUARE_MILK_90 *SQUARE_MILK_90*
- SQUARE_MILK_45 *SQUARE_MILK_45*
- BEAN_CAN *BEAN_CAN*
- SOAP_BOTTLE_FRONT *SOAP_BOTTLE_FRONT*
- SOAP_BOTTLE_SIDE *SOAP_BOTTLE_SIDE*
- GLASS *GLASS*
- RECTANGULAR_BOX *RECTANGULAR_BOX*
- RECTANGULAR_BOX_SIDE *RECTANGULAR_BOX_SIDE*
- WALL_BALL *BALL_WALL*
- BALL_CENTER *BALL_CENTER*
- BEER_BOTTLE *BEER_BOTTLE*

Ogni oggetto è stato posto approssimativamente al centro dell'altra di acquisizione dati, senza utilizzare riferimenti precisi per la posizione con lo scopo di rendere più robusto il riconoscimento da parte del classificatore. Per ogni oggetto l'acquisizione dati è stata ripetuta più volte dopo aver tolto e posizionato nuovamente l'oggetto con variazioni casuali di posizionamento.

Interfacciamento moduli HC-SR-04 / HC-SR-04+

I moduli HC-SR-04 in commercio sono più o meno tutti uguali e derivano da uno stesso progetto di base. La versione “+” è stata modificata per avere tensioni di ingresso/uscita a 3.3V al posto dei 5V dell’originale.

I moduli HC-SR-04 che funzionano a 5V non possono essere collegati direttamente ai GPIO di Raspberry, ma serve un adattatore di livello da 5V a 3.3V. Nel caso specifico è sufficiente un partitore di tensione.

- *voltage divider formula*
- *voltage divider*

image source: Voltage divider - Partitore di tensione - Wikipedia

Tra resistenze disponibili in laboratorio sono stati scelti valori 18K e 10K per realizzare il partitore in modo da ottenere una tensione inferiore a 3.3V sul pin.

$$\Rightarrow V_{out} = 5 * 18K / (18K+10K)=3.2V$$

Nota: Nel caso di configurazioni misti di moduli 3.3V a 5v attenzione alla cablaggio delle linee Vcc!

Collegamento alla morsettiera

Nota: spegnere sempre Raspberry prima di modificare i cablaggi Raspberry 3 è dotato di una morsettiera a 40 pin che include massa, alimentazione, vari tipi di bus di comunicazione e GPIO Per maggiori informazioni sul pinout vedi documentazione specifica

GPIO pins

Cablaggio pannello sensori superiore

I moduli HCSC04 sono predisposti per collegamento con cavi Dupont a 4 poli. In laboratorio ho disponibili solo cavi corti (10-20cm), mentre per collegare tutti sensori sul tetto sono necessari cavi lunghi 1-2 metri.

Cavi dupont lunghi sono disponibili sul mercato, ma sono costosi Amazon.it : cavi dupont e non sempre facili da trovare. Per il prototipo ho individuato due possibili soluzioni economicamente sostenibili:

- cavo ethernet (8 fili)
- cavo per sensori impianto di allarme a 4 fili (Vcc, GND, signal01, signal02)
+ schermatura + anima in nylon per resistenza meccanica

Avevo disponibili in laboratorio degli spezzoni da circa due metri di cavo per sistemi di allarme ed ho utilizzato quelli per realizzare i cavi dupont.

cavo dupont per i sensori

Dopo aver realizzato i tre cavi di lunghezza sufficiente e stato possibile montare il pannello superiore con i sensori ed il relativo telaio di sostegno.

roof sensor panel

Test del prototipo nella configurazione a sette sensori

Per testare il sistema sono stati eseguiti una serie di esperimenti. I cui dati sono stati etichettati in base al tipo di oggetto presente nell'area di misura. I dati prodotti sono poi stati utilizzati per l'addestramento del modello di classificazione. Ogni esperimento è stato ripetuto più volte dopo aver rimosso e riposizionato l'oggetto all'interno dell'area di rilevamento.

Nel caso di funzionamento a vuoto, senza oggetti presenti si sono state osservati diversi episodi di funzionamento instabile dei moduli HC-SR04 riconducibili alla ricezione di echi da parte di superfici riflettenti dell'ambiente circostante.

Per evitare questo problema l'area di rilevamento è stata delimitata montando una barriera sul lato opposto a quello dei sensori.

In un primo momento la barriera di delimitazione risultava essere inclinata di circa 45° rispetto al piano frontale dei sensori. In queste condizioni la distanza stimata dai sensori non era corretta. (vedi acquisizione dati "WALL")

Il pannello originale è stato poi sostituito da due pannelli separati, paralleli ai sensori (vedi acquisizione dati "WALL_45_DEGREE"). In questa configurazione tutti i sensori riescono a stimare la distanza con un errore massimo di qualche centimetro (accuratezza 5-10%) che è ragionevole ai fini dell'esperimento corrente.

Analisi risultati del test

vedi anche Notebook Analisi Dati Sensori

Una prima analisi dei dati raccolti durante i test del sistema nella configurazione a sette sensori mostra la presenza di molti outliers

box plot seven sensors

Analisi dati acquisiti senza oggetto presente

Per tentare di individuare le cause dei numerosi dati anomali sono stati esaminati i dati provenienti dall'esperimento "EMPTY_SEVEN".

In questo esperimento nell'area di misura non era presente nessun oggetto, ma solo i pannelli di delimitazione paralleli al piano frontale dei sensori 001, 002, 003, 004 ad una distanza di circa 55 cm. Il pannello superiore che ospita i sensori 005, 006 e 007 è montato ad una altezza di circa 50 cm rispetto al piano al pavimento dell'area di misurazione.

Gli errori di stima presenti in questo esperimento sono dovuti esclusivamente al sistema di misura.

Empty seven dist plot

Il sensore HC-SR-04_007 è affetto da un errore sistematico che sposta il picco della distribuzione da 50 a 100 cm circa. La causa del problema non è stata individuata, ma l'analisi dei dati degli altri esperimenti dimostra che la misura del tempo di volo fornita dal sensore è comunque fortemente correlata alla distanza del bersaglio dal sensore. La distanza stimata da HC-SR-04_007 contiene quindi informazione utile e i dati prodotti dal sensore non sono stati scartati.

Tutti gli altri sensori forniscono una stima della distanza compatibile con la precisione dichiarate nel datasheet dell'ordine del centimetro.

Gli outliers presenti negli altri esperimenti sono quindi da attribuire ad interferenze ambientali, alla particolare geometria dell'oggetto bersaglio, al materiale e al tipo di superficie riflettente.

Test modello AWGN (Additive White Gaussian Noise)

L'ipotesi che la stima del tempo di volo fornita dai sensori sia modellabile come una variabile aleatoria di tipo AWGN intorno al valore della distanza del bersaglio non risulta supportata dai dati sperimentali. Il test di normalità viene superato da alcuni sensori in alcuni esperimenti, ma l'ipotesi non è verificata in generale per tutte le popolazioni di campioni.

Una possibile spiegazione potrebbe essere cercata nella presenza dei numerosi outlier che in alcuni casi danno origine a delle distribuzioni multivariate. Questo risultato merita un approfondimento ed una verifica sperimentale più accurata.

Il test di normalità applicato è quello implementato nella funzione `normaltest` della libreria SciPy

This function tests the null hypothesis that a sample comes from a normal distribution. It

References

- 1 D’Agostino, R. B. (1971), "An omnibus test of normality for moderate and large sample sizes", *Biometrika*, 58, 315-326.
- 2 D’Agostino, R. and Pearson, E. S. (1973), "Tests for departure from normality", *Biometrika*, 60, 1-19.

Analisi dati esperimenti con oggetti (Caso di studio esperimento “BEAN_CAN”)

Il dataset include i dati raccolti posizionando diversi tipi oggetti all’interno dell’area di misura approssimativamente nella stessa posizione. Gli esperimenti sono stati ripetuti più volte riposizionando ogni volta l’oggetto. Ad ogni ripetizione l’oggetto è stato posizionato in una posizione leggermente diversa da quella precedente, in modo da produrre un dataset di addestramento più simile alle condizioni operative del sistema in produzione.

A titolo di esempio viene riportata l’analisi dell’esperimento “BEAN_CAN” eseguito su un barattolo di fagioli in scatola con dimensioni vicine al limite inferiore ammissibile e una geometria idonea ad evidenziare eventuali problemi nella stima delle distanze (superficie cilindrica, presenza di superfici complesse (elementi tridimensionali e bordi))

esperimento bean_can

L’esperimento è stato ripetuto tre volte rimuovendo e riposizionando l’oggetto. Per ogni posizionamento sono state ottenute rispettivamente 10, 11 e 15 misure valide contenenti la stima della distanza da tutti i sensori. (in totale 36 misure) I risultati delle tre ripetizioni sono stati riportati nei file:

- 20210102_114551_TRAIN_BEAN_CAN.csv
- 20210102_153734_TRAIN_BEAN_CAN.csv
- 20210102_165012_TRAIN_BEAN_CAN.csv

I grafici delle distribuzioni mostrano due picchi ben distinti, mentre il terzo gruppo di dati non è evidente dai grafici perché le stime delle distanze ottenute nella seconda e nella terza ripetizione sono piuttosto simili.

bean_can dist plot

bean_can box plot

Stime delle distanze orizzontali

Nella prima ripetizione i sensori HCSR04_001 e HCSR04_003 hanno fornito delle stime molto vicine al caso “oggetto non presente” (EMPTY_SEVEN). Nelle altre ripetizioni ha dato delle stime vicine alla distanza reale di circa 20 cm

Nella prima ripetizione i sensori HCSR04_002 e HCSR04_004 hanno prodotto delle stime sostanzialmente corrette in tutte le ripetizioni

Stime delle distanze verticali

Il sensore 005 non “ha visto” l’oggetto in nessuna delle ripetizioni il sensore 006 ha fornito una stima completamente errata (2000 cm) nel primo esperimento e approssimativamente corretta (circa 40 cm) nelle altre due ripetizioni

I dati del sensore 007 presentano una distribuzione diversa dal caso “empty seven”, ma non è chiaro come questa differenza sia correlata alla presenza dell’oggetto nell’area di misura.

Conclusioni

L’analisi dei dati dimostra che il tempo di volo (e quindi la distanza) restituito dai singoli sensori dipende fortemente dalla posizione dell’oggetto all’interno dell’area di rilevamento. Il sistema è quindi potenzialmente in grado di stimare la posizione del bersaglio all’interno dell’area di rilevamento.

L’analisi ha però evidenziato anche alcune importanti criticità:

- La stima della distanza restituita dai moduli HC-SR04 è influenzata da vari fattori ambientali (variazioni di temperatura, correnti d’aria, sorgenti di rumore o vibrazione significative ...)
- La relazione tra la distanza stimata e quella reale non è sempre lineare. Nel caso di posizionamento dell’oggetto non ideale la misura è affetta da errori anche molto ampi e difficilmente compensabili perché dipendenti dalle caratteristiche fisiche e geometriche del bersaglio.
- La forma ed il materiale degli oggetti ha un effetto importante nell’accuratezza della stima della distanza.
- Per la riflessione delle onde sonore vale la legge di Snell quindi se l’onda sonora colpisce una superficie non parallela al piano frontale del sensore è possibile che l’onda riflessa non raggiunga direttamente il ricevitore (distanza stimata superiore a quella reale, o non lo raggiunga affatto)
- L’impedenza acustica dell’interfaccia tra l’aria (gas) ed un corpo solido bersaglio è solitamente molto elevata e questo in generale genera una buona riflessione. In presenza di superfici fonoassorbenti e l’energia riflessa può essere molto minore e a volte l’eco potrebbe non essere rilevato dal ricevitore
- Nel caso di superfici irregolari l’onda riflessa può essere molto attenuata a causa di fenomeni di diffrazione)

Misuratori di distanza ad ultrasuoni HC-SR04

Sul mercato esistono diversi misuratori di distanza ad ultrasuoni, destinati al mercato dei maker, con funzionalità e prestazioni sostanzialmente equivalenti. Per il prototipo sono stati utilizzati misuratori di distanza ad ultrasuoni tipo HC-SR04/HC-SR04+ che erano disponibili in laboratorio.

Si tratta di moduli standard, realizzati e distribuiti da diversi fornitori, facilmente reperibili sul mercato al costo indicativo di 2-3€ l'uno.

SR-HC04

Nota: Dei moduli HC-SR04 esistono diverse versioni tra cui anche alcune funzionano a 3.3-3.5 Volt, mentre l'originale funziona a 5V.

Caratteristiche HC-SR04

- Operating voltage: +5V
- Theoretical Measuring Distance: 2cm to 450cm
- Practical Measuring Distance: 2cm to 80cm
- Accuracy: 3mm
- Measuring angle covered: <15°
- Operating Current: <15mA
- Operating Frequency: 40Hz

Per maggiori informazioni vedi sul funzionamento di questi moduli vedi i link in [Risorse.md]

Pinout HC-SR04

I moduli sono dotati di interfaccia standard a 4 pin utilizzata da molti sensori

Pin Number	Pin Name	Description
1	Vcc	The Vcc pin powers the sensor, typically with +5V
2	Trigger	Trigger pin is an Input pin. This pin has to be kept high for 10us to initialize measurement by sending US wave.
3	Echo	Echo pin is an Output pin. This pin goes high for a period of time which will be equal to the time taken for the US wave to return back to the sensor.
4	Ground	This pin is connected to the Ground of the system.

Altro materiale sui moduli HC-SR04 e sensori ad ultrasuoni in generale

<http://www.pcserviceselectronics.co.uk/arduino/Ultrasonic/index.php>

<https://www.sciencedirect.com/topics/engineering/ultrasonic-sensor>

<https://gndtovcc.home.blog/2020/04/18/complete-guide-for-ultrasonic-sensor-hc-sr04-with-arduino/>

<https://www.electronicwings.com/sensors-modules/ultrasonic-module-hc-sr04>

<https://www.seeedstudio.com/blog/2019/11/04/hc-sr04-features-arduino-raspberrypi-guide/>

<https://www.instructables.com/HC-SR04-Ultrasonic-Sensor-With-Raspberry-Pi-2/>

<https://thepihut.com/blogs/raspberry-pi-tutorials/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi>

<http://codeelectron.com/measure-distance-ultrasonic-sensor-pi-hc-sr04/>

<https://www.bluetin.io/sensors/python-library-ultrasonic-hc-sr04/>

<https://www.albertobarbisan.it/didattica/ultrasuoni.pdf>

<https://emanuelebuchicchio.wordpress.com/2016/07/18/misurare-la-distanza-di-un-oggetto/>

<https://emanuelebuchicchio.wordpress.com/2016/08/04/sensore-ad-ultrasuoni-hc-sr04-un-sonar-integrato-compatibile-con-esp8266-arduino-e-raspberry-per-3e/>

Basic Raspberry setup

Per la configurazione iniziale, fino a quando non viene abilitato l'accesso remoto, è necessario utilizzare tastiera, mouse e schermo collegati al raspberry. Successivamente sarà possibile eseguire una sessione remota.

Hardware per configurazione iniziale

- Raspberri Pi 3 Model B
- tastiera USB
- alimentatore
- accesso ad internet tramite rete locale ethernet (cablata o WiFi)

Raspian Install/Upgrade Rasperry OS (Raspian)

Per il prototipo è necessario installare una versione recente di Raspberry OS (Raspian). Si tratta di una distribuzione Linux derivata dalla Debian, ottimizzata per l'utilizzo su Raspberry.

Per l'installazione di Rasperry OS fare riferimento alla guida nella documentazione ufficiale.

Se si dispone di un Raspberry con Raspina già installata è invece consigliato eseguire un upgrade del software alla versione corrente.

Upgrade Raspberry OS

Per prima cosa verificare la versione installata del sistema operativo. Da terminale:

```
cat /etc/os-release
```

La versione corrente (dicembre 2020 è "Buster". Un elenco delle versioni è mantentuo sulla pagina Wikipedia dedicata al prodotto.

Per la procedura di aggiornamento seguire la guida nella documentazione ufficiale.
Dopo l'aggiornamento eseguire un riavvio del sistema.

Pulsante di avvio della misurazione

Il progetto prevede un pulsante/interuttore per abilitare l'acquisizione collegato ad uno dei pin GPIO.

La gestione di un pulsante è un tipico esempio di gestione di un input digitale: dal punto di vista elettrico lo schema deve includere, oltre all'interruttore, le resistenze di pull up/down. Mentre dal punto di vista software viene come un evento di cambio di stato dell'GPIO collegato al pulsante.

Pull Up /Pull down su input digitali

Raspberry PI dispone di resistenze Pull-Up / Pull-Down onboard configurabili via software. Si possono usare le resistenze interne e semplificare il circuito eliminando resistenze esterne.

```
#pull down for trigger button
```

```
GPIO.setup(MAIN_TRIGGER_GPIO, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

Alcuni esempi disponibili in rete sulla gestione di input digitali

- <https://kalitut.com/raspberrypi-gpio-pull-up-pull-down-resistor/>

- <https://raspi.tv/2013/rpi-gpio-basics-6-using-inputs-and-outputs-together-with-rpi-gpio-pull-ups-and-pull-downs>
- https://www.programcreek.com/python/example/98874/RPi.GPIO.add_event_detect

gestione software I/O digitale

Dal punto di vista software la gestione dell'evento "button press" in un sistema embedded tipicamente può essere fatta in tre modi:

1. associando una *callback function* all'evento "Input Up/Down" esposto nell'ambiente di programmazione ad alto livello
2. definendo una *Interrupt Service Routine (ISR)* associata al fronte di salita/discesa del segnale sul pin
3. andando a leggere lo stato dell'input all'interno di un pooling loop ed eseguire una azione in base allo stato rilevato.

Nel caso di Raspberry l'SDK in python non espone gli interrupt HW del micro, quindi il controllo dello stato del pin connesso all'interruttore è stato fatto con la tecnica del *pooling* all'interno del *main loop*.

Main loop:

“Python while True:

```
    mainTriggerState= GPIO.input(MAIN_TRIGGER_GPIO)

    if(mainTriggerState):
        distances,sampleTimestamp = doMeasure()

    else:
        time.sleep(0.1)
        print(".")
```

““

La gestione degli interrupt in Python

L'ambiente di programmazione Python ha una implementazione "software" simile a quella degli interrupt hardware attraverso la funzione `GPIO.wait_for_edge()`, ma si tratta di una sovrastruttura che non sfrutta realmente le interrupt service routine hardware del micro.

Alcuni esempi disponibili in rete:

- <https://roboticsbackend.com/raspberry-pi-gpio-interrupts-tutorial/>
- <https://raspberrypi.hq.com/use-a-push-button-with-raspberry-pi-gpio/>

Sessione remota su Raspberry da PC di sviluppo

Connettendo alla rete locale il raspberry è possibile aprire una sessione remota sul raspberry montato all'interno del sistema di misura.

Raspian OS supporta sessioni remote di terminale SSH oppure sessioni desktop verso il raspberry.

Da Windows è consigliabile usare *Putty* come client per le sessioni SSH e *Connessione a Desktop Remoto* (RDP) per le sessioni grafiche.

Nota: Per l'utilizzo del raspberry montato all'interno del sistema di misura conviene configurare la rete WiFi.

Sessione remota SSH verso Raspberry

Per aprire una sessione remota è necessario conoscere hostname oppure indirizzo IP del raspberry

Per impostazione predefinita il raspberry è configurato per connettersi alle reti wireless o cablate ed ottenere un indirizzo IP dinamico tramite DHCP quindi è meglio utilizzare l'hostname che invece rimane costante.

hostname predfinito configurato durante l'installazione di Raspberry OS è "raspberrypi", ma può essere cambiato.

Da terminale Raspian OS:

`hostname` per avere il nome host

`hostname -I` per avere l'indirizzo IP

Da terminale su PC windows: `ping -a raspberrypi` per ottenere indirizzo ip

Nota: l'utente root di default di Raspian è username: pi password: raspberry

Nota2: nelle versioni recenti il server SSH su RaspberryOS è disabilitato per default e deve essere abilitato dal pannello **Preferenze-> configurazioni**

Vedi SSH (Secure Shell) - Raspberry Pi Documentation nella documentazione ufficiale.

Dopo aver avviato il server SSH su Raspberry è possibile avviare la sessione remota da PC Linux/Windows.

Installazione Python3.8 su Raspian Buster (dicembre 2020)

Ambiente di sviluppo su MacOS (Anaconda) e su Windows utilizza Python 3.8. Per evitare problemi di portabilità dei modelli è preferibile utilizzare la stessa configurazione tra ambiente di sviluppo e produzione.

Inoltre in rete sono disponibili esempi e librerie richiedono Python 3.8 e non funzionano su Python3.7. Il pacchetto python3 per Raspian è aggiornato solo fino alla versione 3.7. Per installare la versione 3.8 è quindi necessario compilarla dai sorgenti.

I sorgenti si possono scaricare da Python Source Releases | Python.org - ultima versione stabile pubblicata ad oggi (dicembre 2020) è la 3.8.7

`wget https://www.python.org/ftp/python/3.8.7/Python-3.8.7.tgz`

Per la compilazione è necessario installare alcune librerie che potrebbero non essere presenti sul sistema. Meglio installare i prerequisiti prima di avviare il processo. Da terminale:

```
sudo apt-get update
```

```
sudo apt-get install -y build-essential tk-dev libncurses5-dev libncursesw5-dev libreadline
```

Installazione di Python 3.8

Sul web si trovano diverse guide per l'installazione. Ad esempio: How to install Python 3.8 on Raspberry Pi. La procedura che ho utilizzato io è questa:

1) scaricare i sorgenti e compilare

```
wget https://www.python.org/ftp/python/3.8.7/Python-3.8.7.tar.xz
```

```
tar xf Python-3.8.7.tar.xz
```

```
cd Python-3.8.7
```

```
./configure --prefix=/usr/local/opt/python-3.8.7
```

```
make -j 4
```

2) Install

```
sudo make altinstall
```


3) Remove the files

```
cd ..  
sudo rm -r Python-3.8.7  
rm Python-3.8.7.tar.xz
```

4) Aprire il file .bashrc con Nano

```
nano ~/.bashrc
```

In fondo al file aggiungere il path e gli alias per l'installazione di Python3.8 In questo modo l'OS continua a gestire la distribuzione di Python3.7 nella /usr/bin, ma per l'utente corrente gli alias *pip3* e *python3* punteranno all'installazione della versione 3.8 in /usr/local/opt

```
export PATH=/usr/local/opt/python-3.8.7/bin:$PATH  
alias pip3=/usr/local/opt/python-3.8.7/bin/pip3.8  
alias python3=/usr/local/opt/python-3.8.7/bin/python3.8
```

5) rileggere il file .bashrc

```
source ~/.bashrc
```

6) Verificare gli alias

```
python3 -V  
pip3 -V  
image-20210108140837948
```

Installare pip, wheel

Nel nuovo ambiente Python3.8 è ora necessario aggiornare pip ed installare wheel per poter poi scaricare tutti i pacchetti che saranno utilizzati.

Attenzione con il comando “sudo” gli alias non funzionano (la procedura del paragrafo precedente li ha creati solo per l'utente “pi”), quindi è necessario specificare il path completo per installare i pacchetti.

```
sudo /usr/local/opt/python-3.8.7/bin/pip3.8 install --upgrade pip wheel
```

Per installare gli altri pacchetti come ad esempio numpy:

```
sudo /usr/local/opt/python-3.8.7/bin/pip3.8 install numpy
```

Attenzione: i pacchetti binari precompilati PyWheel sono disponibili solo per alcune architetture e versioni di python. Spesso per Python 3.8 è necessario compilare dai sorgenti a questa operazione su Raspberry può richiedere diverse ore.

Ultrasonic sensors test

Per verificare il corretto funzionamento del sistema di acquisizione dati:

1. clonare il repository del progetto sul raspberry
2. eseguire lo script aprire la cartella `src` ed eseguire lo script `ultrasonic-sensor-test.py`
3. eseguire ora il test di misura della distanza e salvataggio dei dati su file system `save-sensor-data-to-file.py`

Sessione Desktop Remoto da Windows a Raspian (Raspberry OS)

Prerequisiti:

- Raspberry collegato a rete locale (WiFi o cablata)
- Raspberry configurato con Host name “raspberrypi”

Per aprire una sessione remota è necessario conoscere hostname oppure indirizzo IP del raspberry

Per impostazione predefinita il raspberry è configurato per connettersi alle reti wireless o cablate ed ottenere un indirizzo IP dinamico tramite DHCP quindi è meglio utilizzare l’hostname che invece rimane costante.

hostname predfinito configurato durante l’installazione di Raspberry OS è “raspberrypi”, ma può essere cambiato.

Da terminale Raspian OS:

`hostname` per avere il nome host

`hostname -I` per avere l’indirizzo IP

Da terminale su PC windows: `ping -a raspberrypi` per ottenere indirizzo ip

Installazione server RDP su RaspberryOS

Su Raspian `sudo apt-get install xrdp`

Xrdp is an open-source implementation of Microsoft's proprietary RDP Server, the same protocol that most installations of Windows can connect to and be connected from.

The xrdp software replicates Microsoft's RDP protocol so that other remote desktop clients can connect to your device. The software package even works with Microsoft's own remote desktop client built into Windows.

In Raspian Buster (ad oggi 28-dic-2020) ci sono due bug nella configurazione di XRDP che ne impedisce la corretta esecuzione dopo il rebbot.

In pratica ci sono due problemi con i diritti lettura e scrittura su alcuni file utilizzati da XRDP.

Modificare i permessi del file di log

Disable automatic start at boot time Da terminale: `(systemctl disable xrdp)`

Modificare il percorso dle file di log nel file di configurazione `/etc/xrdp/xrdp.ini`

```
Ini    LogFile=/tmp/xrdp.log
```

C'è anche un altro problema di configurazione di xRDP:

vedi RDP on Raspberry Pi | I.T. Plays Well With Flavors

By default Xrdp uses the `/etc/ssl/private/ssl-cert-snakeoil.key` file which is readable only by users that are members of the "ssl-cert" group. You'll need to add the user that runs the Xrdp server to the `ssl-cert` group.

```
sudo adduser xrdp ssl-cert
```

Ora è possibile utilizzare l'ambiente desktop per sviluppo e debug remoto anche dopo il riavvio del Raspberry

image-20201228150623568

image-20201228150825761

Nota: dalla sessione remota desktop sembra non funzionare l'arresto ed il riavvio del sistema. è necessario eseguire da terminale (o da sessione ssh) i comandi `sudo halt` e `sudo reboot`

Inferenza con modelli ML su Raspberry con ONNX Runtime per applicazioni di Edge AI

Microsoft e una community di partner hanno creato ONNX come standard aperto per la rappresentazione di modelli di machine learning. I modelli di molti Framework , tra cui TensorFlow, PyTorch, SciKit-Learn, keras, Chainer,

MXNET, MATLAB e SparkML possono essere esportati o convertiti nel formato ONNX standard. Quando i modelli sono nel formato ONNX, possono essere eseguiti in un'ampia gamma di piattaforme e dispositivi.

L'ottimizzazione dei modelli di machine learning per l'inferenza o il punteggio del modello è difficile poiché è necessario ottimizzare il modello e la libreria di inferenza per sfruttare al meglio le funzionalità hardware. L'ottimizzazione di tutte le diverse combinazioni di Framework e hardware è molto dispendiosa in termini di tempo.

ONNX Runtime è un motore di inferenza a prestazioni elevate per la distribuzione di modelli ONNX in produzione. È ottimizzato per cloud e Edge e funziona in Linux, Windows e Mac. Scritto in C++, include anche API C, Python, C#, Java e JavaScript (Node.js) per l'utilizzo in diversi ambienti.

ONNX Runtime is backward compatible with all the operators in the ONNX specification. Newer versions of ONNX Runtime support all models that worked with the prior version.

Utilizzando ONNX è possibile risolvere contemporaneamente due importanti problemi

- la portabilità del modello su piattaforme hardware diverse
- interoperabilità tra i diversi framework di ML esistenti

Nell'ambito di questo progetto problemi di portabilità si sono verificati anche tra gli ambienti di sviluppo locali (Anaconda di Windows10 e MacOS) e ambienti di sviluppo cloud (Azure Machine Learning, Google Colaboratory) e successivamente anche tra gli ambienti di sviluppo locali e gli ambienti di produzione (raspberry e container docker su Azure Machine Learning)

Installazione ONNX Runtime

Guida Install - onnxruntime

Prerequisites

Raspian Update

```
sudo apt update
sudo apt upgrade
sudo reboot
```

English language package with the en_US.UTF-8 locale

- Install language-pack-en package
- Run `sudo locale-gen en_US.UTF-8`
- Run `sudo update-locale LANG=en_US.UTF-8`

OpenMP

- `sudo apt-get install libgomp1`, which installs **libgomp.so.1**

Python3.5 - 3.8

Le versioni recenti del runtime ONNX per python richiedono Python3.8 che però non è presente su Raspian. Anche la versione più recente della distribuzione (ad oggi dicembre 2020) purtroppo installa Python3.7

Installazione

Per l'architettura ARM32v7 del Raspberry non sono disponibili release binarie ufficiali né sul sito ONNX né su PyWheel · PyPI. Per utilizzare il runtime è quindi necessario eseguire un build ad hoc sul Raspberry stesso oppure come cross compilazione specificando il target ARM32v7.

Il processo di build è documentato su `onnxruntime/BUILD.md` at master · microsoft/onnxruntime (github.com). La build è comunque un processo lungo e richiede un ambiente di sviluppo non banale da configurare. Fortunatamente è possibile riparare dei pacchetti PyWheel già compilati per Raspberry. Ad esempio su questi repository:

- NagarajSMurthy/RaspberryPi-ONNX-Runtime: Install ONNX Runtime on Raspberry Pi 3B+ (github.com)
- nknytk/built-onnxruntime-for-raspberrypi-linux: Built python wheel files of `https://github.com/microsoft/onnxruntime` for raspberry pi linux.

A questo punto si può installare il pacchetto wheel precompilato del runtime di ONNX.(seguendo la procedura di build oppure scaricando il pacchetto binario già compilato per ARM32v7 e python3.7/3.8 da `nknytk/built-onnxruntime-for-raspberrypi-linux: Built python wheel files of https://github.com/microsoft/onnxruntime` for raspberry pi linux.)

Python 3.7

`https://github.com/nknytk/built-onnxruntime-for-raspberrypi-linux/raw/master/wheels/buster/`

`sudo pip3 install onnxruntime-1.6.0-cp37-cp37m-linux_armv7l.whl`

Python 3.8

`https://github.com/nknytk/built-onnxruntime-for-raspberrypi-linux/raw/master/wheels/buster/`

`sudo /usr/local/opt/python-3.8.7/bin/pip3.8 install onnxruntime-1.6.0-cp38-cp38-linux_armv7l.whl`

Ora è finalmente possibile testare l'esecuzione locale del modello eseguendo lo script di test dalla cartella `/src`

```
cd src
python3 test-onnx-runtime.py
image-20210108142125659
```

Sviluppo di un modello di classificazione degli oggetti

Vedi anche notebook analis dati sensori

L'analisi della correlazione tra le distanze stimate dai sensori ed il tipo di oggetto presente (ObjectClass) non evidenzia l'esistenza di correlazioni lineari rilevanti tra i dati del singol sensore e la classe di appartenza dell'oggetto bersaglio.

Matrice di correlazione sette sensori

scatter plot sette sensori

Analsi delle singole features

I range delle features sono molto diversi a causa degli errori di stima. Per una analisi visiva della separabilità delle classi in base alle features disponibili è necessario prima rimuovere gli outliers. Le distanze dipendono sia dalla posizione dell'oggetto che dal tipo di oggetto

Si riporta qui a titolo di esempio il boxplot dei dati di due sensori disegnato per i diversi tipi di oggetto.

analisi features analisi features

Anche dopo la rimozione degli outliers il pattern delle distanze prodotto dai diversi tipi di bersaglio dipende, oltre che dalla distanza del bersaglio, anche dalla posizione del bersaglio all'interno dell'area di misura.

La dipendenza dalla posizione rende complessa la relazione tra la distanza stimata e la classe di appartenenza del bersaglio.

Per lo sviluppo del modello di classificaione automatica degli oggetti sono stati proati due metodi:

1. Addestramento manuale dei modelli di classificazione con libreria SciKit Learn
2. Addestramento automatico mediante servizio AutoML di Azure Machine Learning

I modelli ottenuti dopo essere stati validati e testati nell'ambiente di sviluppo ed in seguito pubblicati si come web service in cloud (cloud AI) che su runtime locale (EdgeAI).

Rimozione outliers

Per la rimozione degli outliers è stato utilizzato il metodo Inter Quartile Range (IQR) con soglie $Q1-1.5$ e $Q3+1.5$

```
def RemoveOutlierIQR(rawData, columnNames, groupLabel):
    df_all_clean = pandas.DataFrame(columns = columnNames)
    for name, group in rawData.groupby(['ObjectClass']):
        Q1 = group.quantile(0.25)
        Q3 = group.quantile(0.75)
        IQR = Q3 - Q1

        data_clean = group[~((group < (Q1 - 1.5 * IQR)) |(group > (Q3 + 1.5 * IQR))).any(axis=0)]
        df_all_clean = pandas.concat([df_all_clean, data_clean])
    return df_all_clean
```

Conclusioni e futuri sviluppi

Il pattern delle distanze stimate dipende in maniera non facilmente separabile sia dal tipo di oggetto presente nell'area di rilevamento che dalla posizione dell'oggetto rispetto ai sensori. Il problema può essere mitigato (e forse risolto) scegliendo una opportuna configurazione geometrica (numero e posizione dei sensori).

L'impossibilità di separare gli effetti della posizione dell'bersaglio da quelli della geometria e della caratteristiche fisiche dell'bersaglio creano rendono particolarmente complesso il compito di sviluppare di classificazione degli oggetti indipendente dalla posizione utilizzando la configurazione "seven sensors".

I modelli di classificazione addestrati e testati si sono invece rivelati molto precisi se gli oggetti da identificare vengono posizionati in maniera precisa, ricreando una condizione molto vicina a quella in cui sono stati prodotti i dati di addestramento.

Si tratta di una forma di overfitting in cui il modello ha "memorizzato" i dati di training invece di generalizzare ed imparare a distinguere i tipi di oggetto.

Addestramento manuale dei modelli di classificazione con libreria SciKit Learn

L'addestramento è stato eseguito dal notebook model training

Sono stati testati diversi algoritmi comunemente usati per problemi di classificazione:

- Support Vector Machine (SVM)

- Linear SVC
- KNN

Una gamma molto più ampia di algoritmi è stata testata in maniera efficiente usando il servizio Azure AutoML.

Per i dettagli sulle implementazioni dei modelli si rimanda alla documentazione della libreria

Per il training sono stati utilizzati dati ripuliti dagli outliers e normalizzati.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.preprocessing import StandardScaler
import numpy as np
clf = make_pipeline(StandardScaler(), RandomForestClassifier())
clf.fit(x_train_all, np.ravel(y_train_all))
```

Il modello addestrato è stato poi serializzato salvato in un file nel formato nativo SciKit Learn e nel formato ONNX per garantire una migliore portabilità nei diversi ambienti di runtime.

```
import pickle
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

filename='../models/rf_classifier_model_pickle_outline_removed.pkl'
filenameONNX='../models/rf_classifier_model_pickle_outline_removed.onnx'

pickle.dump(clf, open(filename, 'wb'))

initial_type = [('distances', FloatTensorType([None, 7]))]
onx = convert_sklearn(clf, initial_types=initial_type)
with open(filenameONNX, "wb") as f:
    f.write(onx.SerializeToString())
```

Azure AutoML

Utilizzando l'account @studenti.unipg.it è stato possibile attivare un subscription “Azure for Students” con 100€ di credito e vari servizi gratuiti inclusi che ha permesso di istanziare un workspace utilizzare sulla piattaforma cloud *Azure Machine Learning*

image-20210102225641266

image-20210102225745809

Dataset

I file CSV prodotti dai vari esperimenti e dal notebook “create training dataset” sono stati importati nel data storage in cloud e registrati come *Tabular Dataset* *image-20210102224058440*

Servizio Automated Machine Learning

Per sviluppare e pubblicare rapidamente un classificatore da utilizzare insieme al prototipo è stato utilizzato il servizio *Automated Machine Learning* (AutoML). AutoML crea automaticamente delle *pipeline* in cui venono applicate varie tecniche standard di feature engineering e vengono addestrati modelli basati su un ampia gamma di algoritmi. Alla fine dell’esperimento il servizio seleziona il modello migliore rispetto alla metrica selezionata.

Tutti i risultati ottenuti nei diversi “run” generati dall’esperimento possono poi essere analizzati.

Per maggiori dettagli vedi notebook ultrasonic-vision-train-automl

Risultati ottenuti

Nel caso specifico come metrica primaria è stata scelta *weighted AUC* dato che il numero di campioni non era omogeneo per tutte le classi. Tra i vari modelli addestrati da *AutoML* ho selezionato quello prodotto dalla pipeline “RobustScaler + LightGBM”.

Il modello non era stato selezionato come *best run* da AutoML, Il modello presenta delle metriche di performance tutte molto alte *image-20210102234946336* e rispetto a quelli proposti da AutoML è caratterizzato da un *calibration curve* più “equilibrata”.

LightGBM è un algoritmo di classificazione ad albero ad elevate prestazioni sviluppato da Microsoft e rilasciato come open source.

Il modello addestrato è disponibile nella cartella dei modelli AutoML71880948429.zip

Metriche del modello

Di seguito un’analisi dettagliata i parametri suggeriti nella documentazione di AutoML con i punteggi ottenuti dal modello e i paragrafi applicabili del metodo di valutazione.

Confusion Matrix

Confusion matrices provide a visual for how a machine learning model is making systematic errors in its predictions for classification models. The word “confusion” in the name comes from a model “confusing” or mislabeling samples. A cell at row i and column j in a confusion matrix contains the number of samples in the evaluation dataset that belong to class C_i and were classified by the model as class C_j .

In the studio, a darker cell indicates a higher number of samples. Selecting **Normalized** view in the dropdown will normalize over each matrix row to show the percent of class C_i predicted to be class C_j . The benefit of the default **Raw** view is that you can see whether imbalance in the distribution of actual classes caused the model to misclassify samples from the minority class, a common issue in imbalanced datasets.

The confusion matrix of a good model will have most samples along the diagonal

image-20210102234838298

ROC curve

The receiver operating characteristic (ROC) curve plots the relationship between true positive rate (TPR) and false positive rate (FPR) as the decision threshold changes. The ROC curve can be less informative when training models on datasets with high class imbalance, as the majority class can drown out contributions from minority classes.

The area under the curve (AUC) can be interpreted as the proportion of correctly classified samples. More precisely, the AUC is the probability that the classifier ranks a randomly chosen positive sample higher than a randomly chosen negative sample. The shape of the curve gives an intuition for relationship between TPR and FPR as a function of the classification threshold or decision boundary.

A curve that approaches the top-left corner of the chart is approaching a 100% TPR and 0% FPR, the best possible model. A random model would produce an ROC curve along the $y = x$ line from the bottom-left corner to the top-right. A worse than random model would have an ROC curve that dips below the $y = x$ line.

image-20210103004240899

The precision-recall curve

The precision-recall curve plots the relationship between precision and recall as the decision threshold changes. Recall is the ability of a model to detect all positive samples and precision is the ability of a model to avoid labeling negative samples as positive. Some business problems might require higher recall and

some higher precision depending on the relative importance of avoiding false negatives vs false positives.

image-20210103004658289

Cumulative gains curve

The cumulative gains curve plots the percent of positive samples correctly classified as a function of the percent of samples considered where we consider samples in the order of predicted probability.

To calculate gain, first sort all samples from highest to lowest probability predicted by the model. Then take $x\%$ of the highest confidence predictions. Divide the number of positive samples detected in that $x\%$ by the total number of positive samples to get the gain. Cumulative gain is the percent of positive samples we detect when considering some percent of the data that is most likely to belong to the positive class.

A perfect model will rank all positive samples above all negative samples giving a cumulative gains curve made up of two straight segments. The first is a line with slope $1 / x$ from $(0, 0)$ to $(x, 1)$ where x is the fraction of samples that belong to the positive class ($1 / \text{num_classes}$ if classes are balanced). The second is a horizontal line from $(x, 1)$ to $(1, 1)$. In the first segment, all positive samples are classified correctly and cumulative gain goes to 100% within the first $x\%$ of samples considered.

The baseline random model will have a cumulative gains curve following $y = x$ where for $x\%$ of samples considered only about $x\%$ of the total positive samples were detected. A perfect model will have a micro average curve that touches the top-left corner and a macro average line that has slope $1 / \text{num_classes}$ until cumulative gain is 100% and then horizontal until the data percent is 100.

image-20210103005108612

Lift curve

The lift curve shows how many times better a model performs compared to a random model. Lift is defined as the ratio of cumulative gain to the cumulative gain of a random model.

This relative performance takes into account the fact that classification gets harder as you increase the number of classes. (A random model incorrectly predicts a higher fraction of samples from a dataset with 10 classes compared to a dataset with two classes)

The baseline lift curve is the $y = 1$ line where the model performance is consistent with that of a random model. In general, the lift curve for a good model will be higher on that chart and farther from the x-axis, showing that when the model

is most confident in its predictions it performs many times better than random guessing.

image-20210103005738688

Calibration curve

Nota: questa è l'unica metrica non soddisfacente dl modello analizzato. Un grafico molto lontano da quello del modello ideale che fa sospettare un problema di “overfitting”

The calibration curve plots a model's confidence in its predictions against the proportion of positive samples at each confidence level. A well-calibrated model will correctly classify 100% of the predictions to which it assigns 100% confidence, 50% of the predictions it assigns 50% confidence, 20% of the predictions it assigns a 20% confidence, and so on. A perfectly calibrated model will have a calibration curve following the $y = x$ line where the model perfectly predicts the probability that samples belong to each class.

An over-confident model will over-predict probabilities close to zero and one, rarely being uncertain about the class of each sample and the calibration curve will look similar to backward “S”. An under-confident model will assign a lower probability on average to the class it predicts and the associated calibration curve will look similar to an “S”. The calibration curve does not depict a model's ability to classify correctly, but instead its ability to correctly assign confidence to its predictions. A bad model can still have a good calibration curve if the model correctly assigns low confidence and high uncertainty.

Note

The calibration curve is sensitive to the number of samples, so a small validation set can produce noisy results that can be hard to interpret. This does not necessarily mean that the model is not well-calibrated.

image-20210103010400452

Deploy come web service

Il modello ottenuto tramite AutoML è stato poi pubblicato come webservice consumabile da parte del software presente a bordo del Raspberry. Il modello sarà eseguito all'interno di un container docker ed sarà accessibile tramite chiamata POST ad un endpoint REST.

image-20210103001251119

Endpoint: `http://64b32d7c-d926-4197-b807-1350e63adf7c.westeurope.azurecontainer.io/score`

Nota: Per l'utilizzo dell'endpoint è necessario includere nella chiamata le chiavi di autenticazione.

Azure ML SDK Installation

Azure Machine Learning può essere utilizzato anche da ambiente di sviluppo locale. Per questo utilizzo è necessario installare l'SDK

```
python -m pip install -U pip
pip install cryptography
pip install azureml-sdk
pip show azureml-core
```

Nota: la versione deve essere **la stessa** di quella usata nel workspace su AzureML per evitare problemi di compatibilità

Per maggiori informazioni fare riferimento alla documentazione online [Install the Azure Machine Learning SDK for Python - Azure Machine Learning Python | Microsoft Docs](#)

Risultati ottenuti

L'analisi dei dati dimostra che il vettore dei tempi di volo (TOF) stimanti dai sensori dipende dalla posizione del bersaglio all'interno dell'area di rilevamento e dalle caratteristiche fisiche (geometria, materiale e tipo di superficie) dell'oggetto. Il sistema è quindi potenzialmente in grado di stimare la posizione dell'oggetto all'interno dell'area di rilevamento e di discriminare tra diversi tipi di oggetti.

I test eseguiti con il prototipo del sistema hanno però evidenziato anche alcune importanti criticità:

- Il pattern delle distanze stimate dipende in maniera non facilmente separabile sia dal tipo di oggetto presente nell'area di rilevamento che dalla posizione dell'oggetto rispetto ai sensori. Il problema può essere mitigato scegliendo una opportuna configurazione geometrica (numero e posizione dei sensori).
- L'impossibilità di separare gli effetti della posizione dell'bersaglio da quelli della geometria e delle caratteristiche fisiche dell'oggetto rendono particolarmente complesso il compito di sviluppare un sistema di classificazione degli oggetti indipendente dalla posizione.

I modelli di classificazione addestrati e testati si sono invece rivelati molto precisi se gli oggetti da identificare vengono posizionati in maniera precisa, ricreando una condizione molto vicina a quella in cui sono stati prodotti i dati di addestramento.

- I classificatori addestrati nell'ambito di questo progetto hanno performance molto elevate se gli oggetti vengono posizionati esattamente come durante la fase di training, ma non riescono ad riconoscere l'oggetto in maniera indipendente dalla posizione. Questo problema è affrontato nel progetto

Ultrasonic Object Recognition che ha come obiettivo la realizzazione di un sistema di riconoscimento degli oggetti basato sui misuratori ad ultrasuoni utilizzando tecniche di machine learning e deep learning più avanzate.

- La relazione tra la distanza stimata e quella reale non è sempre lineare. Nel caso di posizionamento dell'oggetto non ideale la misura è affetta da errori anche molto ampi e difficilmente compensabili perché dipendenti dalle caratteristiche fisiche e geometriche del bersaglio.
- La forma ed il materiale degli oggetti ha un effetto importante nell'accuratezza della stima della distanza.
- Per la riflessione delle onde sonore vale la legge di Snell quindi se l'onda sonora colpisce una superficie non parallela al piano frontale del sensore è possibile che l'onda riflessa non raggiunga direttamente il ricevitore (distanza stimata superiore a quella reale, o non lo raggiunga affatto)
- L'impedenza acustica dell'interfaccia tra l'aria (gas) ed un corpo solido bersaglio è solitamente molto elevata e questo in generale genera una buona riflessione. In presenza di superfici fonoassorbenti e l'energia riflessa può essere molto minore e a volte l'eco potrebbe non essere rilevato dal ricevitore
- Nel caso di superfici irregolari l'onda riflessa può essere molto attenuata a causa di fenomeni di diffrazione)

Futuri Sviluppi Ultrasonic - Vision

Connettività ed invio dati verso storage esterno

Aggiungere sistema per invio dei dati acquisiti verso uno storage esterno. Es. IoT Hub, attraverso protocollo MQTT e rete ethernet/wifi.

Trigger automatico

Aggiungere un sistema di trigger per avviare in maniera automatica la misura (es. fotocellula, sensore di prossimità IR, PIR, ...)

Caratterizzazione dei sensori ad ultrasuoni utilizzati

Ricavare (dove non disponibile) una caratterizzazione affidabile dei sensori utilizzati.

Rappresentazione grafica 3D dell'oggetto

Restituire una rappresentazione grafica 3D dell'oggetto a partire dagli echi sonar acquisiti (stima delle distanze)

Ridurre il tempo necessario ad eseguire una misura

Determinare, in base a caratteristiche dell'hardware e alla configurazione geometrica, l'intervallo di tempo minimo tra due impulsi sonar per evitare interferenze misuratori diversi

Modello avanzato per riconoscimento oggetti

Sviluppare ed addestrare un modello avanzato per il riconoscimento degli oggetti in un progetto di ricerca dedicato.

Pipeline di miglioramento continuo del classificatore

Progettare, sviluppare e pubblicare pipeline per addestramento e miglioramento continuo del modello ogni volta che sono disponibili nuovi dati per l'addestramento

Calibrazione del sistema

La velocità propagazione delle onde sonore dipende dal mezzo. In aria il parametro fondamentale è la temperatura che nel caso di applicazioni reali può variare anche in modo significativo. Due possibilità per migliorare l'accuratezza del sistema:

1. Misurare la temperatura e applicare compensazione della temperatura nel calcolo della distanza dalla temperatura
2. Calibrare il sistema (eventualmente anche sensore per sensore) eseguendo delle misure ripetute su bersagli a distanza nota. -> vedi libreria https://github.com/emanbuc/Bluetin_Python_Echo

Nota: nei Raspberry è presente un sensore di temperatura a bordo della scheda che può essere usato per la calibrazione

Versione “Industrial grade” del sistema

Utilizzando un gateway per applicazioni IoT industriali compatibile con Raspberry il sistema può essere trasformato in un prodotto reale utilizzabile in ambi-

ente industriale e portato realmente sul campo con pochissimo lavoro (qualche giorno) senza modificare il software.

Esempi di prodotti facilmente reperibili a basso costo (130-150€) sono:

- PLC / PC industriali basati su Raspberry <https://andino.shop/en>
- Industrial gateway della Hilsher (<https://www.hilscher.com/products/product-groups/industrial-internet-industry-40/netiotnetfield-edge>)

Versione 2.0: rete di sensori distribuiti realizzata con moduli WiFi ESP82266 che scambiano dati tramite MQTT

1 raspberry utilizzato come broker MQTT e eventualmente anche come gateway/access point wifi

N noti ESP8266 gestiscono i sensori ed inviano i dati tramite MQTT e rete Wifi

Nota: i moduli ESP8266 espongono fino a 16 GPIO utilizzabili => ogni modulo può supportare fino a 8 sensori HC-SR04 + sensore di temperatura su ingresso analogico ADC interno per calibrazione.

In questo modo si può aumentare praticamente all'infinito la risoluzione del sistema in termini di numero di sensori.

E si possono estendere le dimensioni dell'area di rilevamento fino ai limiti della copertura delle reti WiFi (In teoria usando dei ripetitori ed una suddivisione a zone si potrebbe coprire anche un intero palazzo o un'area molto grande).

Strumenti e tecnologie utilizzate

- Ambiente di sviluppo
- Jupyter Notebook
- Visual Studio Code
- Python3
- GIT
- Azure Machine Learning
- Automated ML
- Compute
- Pipeline
- SDK
- Service endpoints

- Machine Learning Studio
- Dataset
- Analisi dati e Machine Learning (Python)
- Numpy
- SciPy
- Pandas
- Seaborn
- SciKitLearn
- ONNX
- Documentazione e gestione progetto
- GitHub
- Markdown
- Pandoc
- Raspberry
- Raspberry Pi 3
- Raspberry OS
- GPIO