Appunti di cognitive robotics

Antonio

November 29, 2025

Contents

1 Perception

Classificazione di sensori:

- Sensori **proprio**-cettivi, sensori che misurano valori interno al robot, ad esempio:
  - angolazione di un giunto
  - velocit di un motore
  - voltaggio batteria
- Sensori **estero**-cettivi, sensori che misurano valori esterni al robot, ad esempio:
  - misuratori di distanza
  - misuratori di intensit luminosit
  - telecamera e microfoni (?)
- Sensori **passivi**, che non rilasciano energia nell'ambiente, misurano solo. Termometri, microfono, camere.
- Sensori **attivi**, che rilasciano energia nell'ambiente, come lidar, che usano un impulso laser per misurare la distanza di un oggetto dal robot.

TIPI DI SENSORI

- **Sensori ruota/motore**: Misurano la velocit e la posizione delle ruote/motori. Gli **encoder** sono un esempio comune, convertono il movimento in un segnale elettrico. Gli **encoder ottici** utilizzano luce e un disco modellato per determinare posizione, direzione e velocit.
- **Sensori di direzione**: Determinano l'orientamento e l'inclinazione del robot. Possono essere propriocettivi (giroscopi, inclinometri) o esterocettivi (bussole). I giroscopi misurano l'orientamento e la velocit angolare, sfruttando il principio di conservazione del momento angolare. Esistono giroscopi meccanici e ottici.
- **Beacon terrestri**: Consentono al robot di identificare la sua posizione con precisione, interagendo con beacon nell'ambiente. Il GPS un esempio, utilizza i segnali di almeno 4 satelliti per determinare la posizione. Dato che i satelliti mandano continuamente sengali in maniera sincronizzati, il robot pu inferire la sua posizione. Adatto per veicoli che lavarona all'esterno (non c' molto disturbo sui segnali)
- **Telemetria attiva**: Misura la distanza dagli oggetti utilizzando il tempo di volo di un segnale. Esempi: sensori ad ultrasuoni, laser rangefinder. Il principio di funzionamento si basa sul calcolo del tempo impiegato da un segnale (sonoro o luminoso) per raggiungere un oggetto e tornare indietro (lidar luce, radar radio, sonar suono).
- **Sensori basati sulla visione**: Utilizzano telecamere per acquisire informazioni visive. La visione fornisce una quantit enorme di informazioni sull'ambiente. Le telecamere di profondit consentono di recuperare informazioni sulla profondit utilizzando due sensori e confrontando le immagini (Stereo camera).

2 ROS

2.1 Publisher-Subscriber

PUBLISHER CODE

```
import rospy
from std_msgs.msg import * # import topic types

rospy.init_node('node_name')

pub = rospy.Publisher('topic_name', String, queue_size=10)
# la queue_size serve nel caso vengono prodotti pi  messaggi di quanti ne riesco a mandare e come
# gestire la coda che si viene a creare.
# queue_size = 1, esempio di un sensore in cui solo l'ultimo valore letto    importante,
# se ci sono letture precedenti che non sono state mandate possono essere scartate

pub.publish('hello')
```

SUBSCRIBER CODE

```
import rospy
from std_msgs.msg import *

def callback(data):
    rospy.loginfo(data)

rospy.init_node('node_name')

rospy.Subscriber('node_name', String, callback )

rospy.spin()
```

2.2 SERVICE

1. si crea un srv file 'SumSrv' (nome generico) che definisce cosa prende input il servizio e cosa restituisce il servizio. Esempio di srv:

```
SumSrv.srv
int32 a
int32 b
---
int32 sum
```

2. si aggiungono le dependency per la generazione di messaggi
3. si aggiunge nel cmakelist il srv file in add_service_files, ora nel codice si possono usare i tipi SumSrvRequest e SumSrvResponse creati dopo aver fatto catkin build

SERVER CODE

```python
import rospy
from pkg.srv import * # srv cartella dove sta SumSrv.srv

# data sar  un istanza di SumSrvRequest
def handle(data):
    return SumSrvResponse(data.a + data.b)

rospy.init_node('sum_service')

s = rospy.Service('SumService', SumSrv, handle)

rospy.spin()
```

CLIENT CODE

```python
import rospy
from pkg.srv import *

rospy.init_node('client_node') # non    necessario che sia un nodo per usare service

rospy.wait_for_service('sum_service')

sum_proxy = rospy.ServiceProxy('sum_service',SumSrv)

somma = sum_proxy(10,10)
```

2.3 LAUNCHER

```
<launch>
    <arg name='arg_name' default='def_value'/>
    <node name='node_name' type='python_script_name'
    output='screen' args=$(arg arg_name) type/>
</launch>
```

3 Conversation AI

Pu essere:

- **chit-chat**, conversation for conversation sake, non ha uno scopo preciso.
- **TOD**, task oriented conversation, c' uno scopo alla conversazione, serve per aiutare l'user a conseguire uno specifico task. Esempio: conversazione per prenotare un appuntamento.

Durante la conversazione ci sono 3 step principali:

- **NLU, natural language understanding**
- **DM, dialogue management**
- **NLG, natural language generation**

3.0.1 NLU, capire

In una frase presentata da un user, si cerca di capire l'**INTENT** e le **ENTITY** contenute nella frase.

L'intent legato ai **verbi**, cosa l'user vuole fare. Pu essere visto come un task di classificazione.

Le entity sono i **nomi**, informazioni legati a cosa l'user vuole fare. Problema di detezione.

Esempio:

COME SARA' IL TEMPO DOMANI A SALERNO?
INTENT: RICHIESTA PREVISIONE DEL TEMPO
ENTITA': SALERNO, DOMANI

Soluzione basata sul deep learning: due modelli per entity recognition e un per intent classification, o un singolo modello.

Il vantaggio del singolo modello la condivisione dei parametri, quindi sar pi veloce.

Il modello per interagire con una frase deve prima tokenizzarla. Possibile metodo per tokenizzare usare gli spazi bianchi.

Dai token poi si deve passare a un vettore numerico per ogni token, necessario un embedding layer, successivamente architetture comuni per il task sono i **biderectional recursive neural network e i transformers**.

3.1 Concetti fondamentali per RASA

- **ENTITIES**
- **INTENTS**
- **ACTIONS**, risposte che il chatbot produce agli input degli utenti, in Rasa possono essere
  - Responses, utterances, semplici frasi, che possono essere in aggiunta a immagini e/o bottoni
  - Forms, per chiedere in risposta un set di informazioni
  - Azioni, codici python che vengono eseguiti in risposta, per esempio per fare query ad API o base di dati.
- **STORIES**, servono per addestare il modello, sono tipici flussi di dialogo tra il chatbot e un utente. Dalle storie il modello impara a gestire le conversazioni. Vengono descritte solo con intents, entities e actions.
- **DOMAIN**, descrive il campo in cui il chatbot opera, quindi tutti gli intent,entities, azioni, risposte e slots.
- **SLOTS**, key-value pairs che sono la memoria del chatbot, vengono riempiti durante la conversazione e la possono influire. Ad esempio se chiedo 'Come sar il tempo domani?' ed ho gi settato lo slot citt, posso direttamente rispondere con il tempo di domani a Salerno.
- **RULES**, regole che il chatbot deve sempre seguire, come fare un saluto dopo che una frase stata classificata come greet, o come chiedere conferma prima di fare una prenotazione.

3.2 Esempio RASA

Creaimo un chatbot basato su Rasa per assistere gli utenti di una pizzeria per fare ordinazioni. **domain.yml**, file che contiene il dominio del chatbot

```yaml
version: "3.1"

intents:
  - greet
  - goodbye
  - get_menu
  - order_pizza
  - specify_pizza_type
  - confirm_order
  - cancel_order

actions:
  - action_delivery

responses:
  utter_greet:
  - text: "Hey! Do you want a pizza?"

  utter_get_menu:
  - text: "What kind of pizza would you like? We have Margherita, Pepperoni, Vegetarian, and Hawaiian."

  utter_confirm_order:
    - text: "You want a {pizza_type} pizza. Is that correct?"

  utter_order_placed:
    - text: "Great! Your {pizza_type} pizza has been ordered!"

  utter_cancel_order:
    - text: "Your order has been canceled."

  utter_goodbye:
  - text: "Bye"

  utter_specify:
  - text: "What would you like to do?"

entities:
- pizza_type

slots:
  pizza_type:
    type: text
    influence_conversation: false
    mappings:
    - type: from_entity
      entity: pizza_type

session_config:
  session_expiration_time: 60
  carry_over_slots_to_new_session: true
```

nlu.yml, file che contiene esampi per addestrare il modello di nlu. **Importante** che contenga esempi per l'estrazione di entit.

```yaml
version: "3.1"

nlu:
- intent: greet
  examples: |
    - hey
    - hello
    - hi
    - hello there


- intent: goodbye
  examples: |
    - cu
    - good by
    - cee you later
    - good night

- intent: order_pizza
  examples: |
    - I want to order a pizza
    - I'd like a pizza
    - Can I get a pizza?


- lookup: pizza_type
  examples: |
    - Margherita
    - Pepperoni
    - Vegetarian
    - Hawaiian

- intent: specify_pizza_type
  examples: |
    - I want a [Margherita](pizza_type)
    - I'd like a [Pepperoni](pizza_type) pizza
    - Give me a [Vegetarian](pizza_type) pizza
    - Can I order a [Hawaiian](pizza_type)?

- intent: confirm_order
  examples: |
    - yes
    - y
    - indeed
    - of course


- intent: cancel_order
  examples: |
    - no
    - n
    - never
    - I don't think so


- intent: get_menu
  examples: |
    - What pizzas do you have?
    - Can I see the menu?
    - Show me the pizza menu
    - What is on the menu?
```

rules.yml, regole che il chatbot seguir durante la conversazione

```yaml
version: "3.1"

rules:

- rule: Great the user anytime they greet the bot
  steps:
  - intent: greet
  - action: utter_greet

- rule: Say goodbye anytime the user says goodbye
  steps:
  - intent: goodbye
  - action: utter_goodbye

- rule: Get menu when user asks for it
  steps:
  - intent: get_menu
  - action: utter_get_menu

- rule: Order pizza
  steps:
  - intent: order_pizza
  - action: utter_get_menu
```

Stories.yml, per addestare il modello a gestire le conversazioni.

```yaml
version: "3.1"

stories:
- story: happy path
  steps:
  - intent: specify_pizza_type
    entites:
        - pizza_type: "margherita"
  - action: utter_confirm_order
  - intent: confirm_order
  - action: action_delivery

- story: unhappy path
  steps:
  - intent: specify_pizza_type
    entites:
        - pizza_type: "margherita"
  - action: utter_confirm_order
  - intent: cancel_order
  - action: utter_cancel_order

- story: cancel without a reason
  steps:
  - intent: cancel_order
  - action: utter_specify

- story: confirm without a reason
  steps:
  - intent: confirm_order
  - action: utter_specify
```

4 Audio analysis

Nell'analisi dell'audio, si ha un problema di **classificazione**, dare una label a uno stream audio, cio capire cosa contiene, voce umana, genere musicale, emozione della voce, colpi di arma da fuoco. Una traccia audio pu contenere pi di un label.

Si ha anche un problema di **detezione**, stimare quando inizia e finisce una specifica traccia audio, come la voce umana di uno interlocutore.

4.1 Rappresentare l'audio

ADC, analog to digital conversion.

Il segnale audio campionato con un **Sampling Rate sr**, il teorema di Nyquist impone che sr sia almeno il doppio della frequenza pi alta nel segnale audio per evitare aliasing.

L'ampiezza del segnale quantizzata in bit, solitamente float32 o int16.

Per analizzare un segnale audio, viene diviso in frame. Questa divisione ha due parametri:

- K, frame size, quanto grande ogni frame. (solitamente 40ms / 20ms)
- Q, Hop size che definisce quanto dista l'inizio di un frame dall'inizio del frame successivo. Solitamente Q ¡ K cos che ci sia overlap tra i segnali (Q = K/2)

Poi vengono applicati filtri passa-basso per evitare aliasing (hamming window e hann window).

Per estrarre delle feature dai frame audio, viene usata Fourier transform per passare dal dominio del tempo alla frequenza (coefficienti della sommatoria di sinusoidi = feature?). In alternativa short-time-fourier-transform STFT, per usare uno spettrogramma, che contiene informazioni temporali, sulla frequenza e sull'ampiezza dell'onda in una singola immagine.

Pi la finestra grande, pi ho una risoluzione maggiore per le frequenze a discapito della risoluzione temporale.

Pi la finestra piccola, pi ho una risoluzione minore per la frequenza a vantaggio della risoluzione temporale.

4.2 Deep learning

Si pu usare lo spettrogramma di un segnale, che un immagine, come ingresso per una rete profonda basata su CNN per risolvere il problema di detezione e classficazione. Questo approccio ideale in casi in cui i dati sono pochi, dato che diamo in ingresso alla rete dell'handcrafted features.

Si pu invece dare il segnale campionato grezzo, che un vettore 1D, come ingresso ad per un rete profonda basata su CNN (user 1D convolution) per risolvere il problema di detezione e classficazione. Richieder pi dati per imparare una corretta rappresentazione dell'audio.

4.3 Re-Identifcation

Riconoscere gli utenti dalla loro voce. Idea naive:

- CNN addestrata per riconoscere diversi speaker.
- Rimozione dell'ultimo FC layer, ora abbiamo una rete che dato un segnale audio produce un vettore che ne un embedding (idea, due sample di voce della stessa persona saranno vicini nel feature space).
- Usiamo gli embedding per fare le associazioni, similarit basate su cosine similarity. Usiamo threshold per decidere se lo speaker nuovo o gi incontrato. (Simile al problema di artifical vision per bounding box e track matching)

4.3.1 Siamese network

Per imparare una funzione di similarit. Dati due rete neurali che condividono architettura e pesi, vengono addestrate usando la **Pairwise-loss** o **Triplet-loss**.

Una delle due reti riceve un campione "ancora", e l'altra riceve un campione che pu essere positivo (dello stesso tipo dell'ancora) o negativo.

La pairwise-loss computata sugli embedding prodotti dalle due reti, se il campione positivo, la loss sar la distanza tra i due embedding, se negativo sar invece la vicinanza (il massimo tra 0 e m - distanza).

La triplet-loss invece usa 3 reti, di cui una l'ancora, una riceve un campione positivo e l'altra negativo. Principio lo stesso, usano il massimo tra 0 e (m - distanza tra l'ancora e il negativo) + distranza tra l'ancora e il positvo.

La validazione va fatta su voci (classi) che non appaiono proprio nel dataset.

5 Object detection

Data un immagine, riconoscere di un oggetto, la **posizione** (tramite bounding box), la **categoria** (che classe appartiene), livello di **confidenza**.

6 Facial emotion recognition

L'emotion recognition pu essere un task:

- Classificazione / Categorico, classificare l'emozione di una persona tra x possibili emozioni, solitamente: felicit, tristezza, paura, disgusto, rabbia, sorpresa, neutro. (65% +- 5% di accordo tra annotatori)
- Regressione, considerare l'emozione come un vettore di tre variabili: valence (quanto piacevole l'emozione), arousal (quanto forte l'emozione), dominance (quanto controllo si ha sull'emozione)
- Detezione, riconoscere singoli movimenti facciali e dalla lora combinazione capire l'emozione. (tipo raised eyebrows + bocca e occhi spalancati = sopresa)

Problema: gli annotatori umani non sono mai al 100% d'accordo su come classificare l'emozione sulla faccia di una persona (una persona disgustata per un annotatore ed impaurita per un altro).

6.1 Pipeline per emotion recognition

1. Modello (dnn o euristico) per face-detection, che isola le faccie in un immagine
2. Face normalization (frontalization, histogram equalization ...)
3. Feature extraction, HOG (histograms of oriented gradients), LBP (local binary patterns), CNNs
4. Classificazione/Regressione, tramite MLP (multilayer perceptrons) o SVM (support vector machine)

Si pu usare la classificazione di un singolo frame, oppure usare **approcci temporali**:

- Decision level, voting aggregato di x frame
- Feature level fusion, concatenazione delle feature estratte da x frame e classficazione di questa nuova feature.
- Joint spatio-temporal model, modelli che sfruttano anche le informazioni temporali come ad esempio le reti ricorrenti