Documentazione Modulo `unimi-crop-sensing`

September 16, 2025

Introduzione

Questo modulo ha l'obiettivo di fornire strumenti per:

- configurare una telecamera ZED e acquisire immagini e mappe di profondità;
- identificare e segmentare piante a partire da immagini RGB;
- calcolare bounding box 2D e 3D;
- comunicare i risultati ad un cobot tramite ROSBridge e WebSocket.

Installazione

Il pacchetto è distribuito tramite `pip`, il gestore ufficiale dei pacchetti Python che permette di scaricare e installare librerie dal Python Package Index (PyPI).
Per installare:

```
pip install unimi-crop-sensing
```

Requisiti:

- Python 3.9 o superiore
- ZED SDK
- Le librerie python citate su pypi
- ROS2 su un altra macchina per i moduli di comunicazione su ROSbridge

Per maggiori dettagli e aggiornamenti: `https://pypi.org/project/unimi-crop-sensing/`

Configurazione ZED

zed_init(pose=0) *richiede stereocamera zed collegata

Questa funzione inizializza la telecamera ZED con una configurazione predefinita (HD2K, profondità `NEURAL_PLUS`, sistema di coordinate `right handed z up`) e imposta una trasformazione in base alla posa del cobot. L'approccio seguito consiste nel preparare i parametri di inizializzazione, verificare che la camera si apra correttamente e quindi calcolare la trasformazione tramite traslazione e quaternione di orientazione, derivati dall'oggetto `pose`. Nel caso si volesse usare il sistema di riferimento di default, l'oggetto pose passato deve avere ogni valore uguale a 0.

Parametri:

- pose (object): oggetto contenente posizione (x, y, z) e orientazione (x, y, z, w) (default = ogni valore a 0)

Ritorna:

- sl.Camera: oggetto camera inizializzato e pronto all'acquisizione

Eccezioni:

- SystemExit: se la camera non si apre correttamente

Esempio di un oggetto pose:

```
class Pose:
    class Position:
        def __init__(self, x=0, y=0, z=0):
            self.x = x
            self.y = y
            self.z = z

    class Orientation:
        def __init__(self, x=0, y=0, z=0, w=0):
            self.x = x
            self.y = y
            self.z = z
            self.w = w
```

—

get_zed_image(zed, save=False) *richiede stereocamera zed collegata

La funzione acquisisce un singolo frame dalla telecamera ZED e ne estrae diverse rappresentazioni utili all'elaborazione: immagine RGB, mappa di profondità, mappa delle normali e nuvola di punti. Durante l'acquisizione viene utilizzato un ciclo di cattura che termina non appena si ottiene un frame valido. Per motivi di debugging e testing, tutte le acquisizioni possono venir salvate nella sottocartella `/data`.

Parametri:

- zed (sl.Camera): oggetto camera inizializzato
- save (bool, opzionale): False come defalut, se True salva ogni acquisizione nella sottocartella `/data`

Ritorna:

- tuple: (image, depth_map, normal_map, point_cloud)

—

Localizzazione delle piante

filter_plants(image, ...)

Questa funzione individua le aree verdi presenti in un'immagine RGB applicando l'indice *Excess Green*. Dopo aver calcolato l'indice, l'immagine viene filtrata tramite una soglia ottenuta con il metodo di Otsu (o forzata al valore minimo indicato dall'utente) e successivamente ripulita da rumore attraverso operazioni morfologiche di erosione e dilatazione.

Parametri:

- image (np.ndarray): immagine RGB di input
- default_T (int, opzionale): soglia minima da usare se Otsu restituisce un valore troppo basso
- kernel_dimension (int, opzionale): dimensione del kernel usato nelle operazioni morfologiche (default = 1)
- cut_iterations (int, opzionale): numero di iterazioni di erosione e dilatazione (default = 1)
- save_mask (bool, opzionale): se True salva un'immagine mascherata in `data/excess_green.png`

Ritorna:

- np.ndarray: maschera binaria con le aree verdi rilevate

—

segment_plants(mask, n_plants)

Data una maschera binaria in cui le piante sono rappresentate da pixel con valori non nulli, questa funzione le suddivide in un numero definito di cluster utilizzando l'algoritmo K-Means.

Parametri:

- mask (np.ndarray): maschera binaria delle piante
- n_plants (int): numero di cluster (piante) da individuare

Ritorna:

- tuple: contenente
  - maschere (list of np.ndarray): lista di maschere binarie, una per ogni pianta segmentata
  - bounding box 2D (list of tuples): coordinate $(x\_min, y\_min, x\_max, y\_max)$ per ciascuna pianta

—

save_clustered_image(image, bounding_boxes)

Disegna i bounding box 2D sull'immagine originale per visualizzare i cluster delle piante e salva il risultato in `data/clusters.png`.

Parametri:

- image (np.ndarray): immagine RGB originale su cui disegnare i bounding box
- bounding_boxes (list of tuples): elenco di bounding box 2D, ciascuno nella forma `(x_min, y_min, x_max, y_max)`

—

get_3d_bbox(mask, point_cloud)

Questa funzione ricava la bounding box tridimensionale di una pianta estraendo dalla point cloud tutte i punti 3D che corrispondono ai pixel attivi della maschera, e calcolando da essi i valori minimi e massimi lungo ciascun asse `x, y, z`.

Parametri:

- mask (np.ndarray): maschera binaria 2D che definisce la regione di interesse
- point_cloud (sl.Mat): nuvola di punti 3D allineata con l'immagine

Ritorna:

- dict: dizionario contenente le coordinate minime e massime del bounding box tridimensionale

Esempio di struttura bounding box 3D:

```
bbxpts = {
    "min": {"x": x0, "y": y0, "z": z0},
    "max": {"x": x1, "y": y1, "z": z1}
}
```

—

record_and_save(plant_name, frames) *richiede stereocamera zed collegata

Questa funzione utilizza la telecamera ZED per acquisire dati di mappatura spaziale e salvarli come file `.ply`, aumentando il numero di frames da registrare aumenta sia il tempo di registrazione, sia la precisione della nuvola dei punti.

NOTA: Viene inizializzata una nuova istanza della camera con parametri di tracking e mapping, perciò è consigliato chiudere (`zed.close()`) istanze attive+.

Parametri:

- plant_name (str, opzionale): nome del file di output (default = "plant")
- frames (int, opzionale): numero di frame da acquisire (default = 300)

Eccezioni:

- RuntimeError: se la camera non si inizializza correttamente o non riesce ad acquisire i frame richiesti

—

Comunicazione con il cobot

NOTA: Queste funzioni sono usate solo nel caso in cui gli algoritmi di controllo del cobot siano presenti in una macchina differente. Nel caso in cui lo script di controllo della Zed fosse contenuto nella stessa macchina, basterebbe semplicemente chiamare le funzioni direttamente.

get_cobot_pose(linux_ip, timeout)

Recupera la posa attuale del cobot tramite una connessione WebSocket verso un server ROS-Bridge. La funzione si connette al server specificato, si iscrive al topic `/cobot_pose` e attende di ricevere i dati della posa. Se i dati non arrivano entro il tempo massimo indicato da `timeout`, viene restituito un oggetto `Pose` con tutti i campi impostati a zero.

Parametri:

- linux_ip (str): indirizzo IP del server ROSBridge
- timeout (int, opzionale): tempo massimo di attesa in secondi (default = 1)

Ritorna:

- Pose: oggetto contenente
  - position: coordinate (x, y, z)
  - orientation: quaternione (x, y, z, w)

Esempio di oggetto Pose:

```
class Pose:
    class Position:
        def __init__(self, x=0, y=0, z=0):
            self.x = x
            self.y = y
            self.z = z

    class Orientation:
        def __init__(self, x=0, y=0, z=0, w=0):
            self.x = x
            self.y = y
            self.z = z
            self.w = w

    def __init__(self):
        self.position = self.Position()
        self.orientation = self.Orientation()
```

send_cobot_map(linux_ip, bbxpts)

Invia le coordinate di un bounding box al cobot tramite ROSBridge usando WebSocket. La funzione stabilisce una connessione al server ROSBridge, pubblica un messaggio sul topic `/Boing` e trasmette la bounding box 3D delle piante da tracciare.

Parametri:

- linux_ip (str): indirizzo IP del server ROSBridge
- bbxpts (dict): dizionario contenente le coordinate del bounding box con struttura

Esempio di struttura bbxpts:

```
bbxpts = {
    "min": {"x": x0, "y": y0, "z": z0},
    "max": {"x": x1, "y": y1, "z": z1}
}
```