Proposta di progetto "Standard"

Questo è il progetto **standard** inerente argomenti trattati a lezione. È Possibile in alternativa svolgere un progetto Avanzato, da concordare individualmente con il docente.

La consegna proposta per il progetto di esame consiste in DUE esercizi.

- La risoluzione del PRIMO esercizio è condizione NECESSARIA per il superamento dell'esame e SUFFICIENTE al raggiungimento di un punteggio base (funzione solo del punteggio ottenuto in scritto e orale). Il primo esercizio è di natura strettamente individuale.
- La proposta di una soluzione per il secondo esercizio è considerata SOLO a patto di avere risolto il primo. È condizione NECESSARIA per ambire a un punteggio superiore.

1. Controllo del sistema Cart-Pole.

Per il primo esercizio si richiede di attenersi scrupolosamente alla consegna (specificata di seguito). E' comunque richiesto che il candidato sia in grado di dimostrare adeguata conoscenza della soluzione proposta e delle tecniche impiegate.

2. Navigazione autonoma di base.

Per il secondo esercizio viene fornita una traccia di progetto. Il candidato è libero di proporre qualunque variazione dalla traccia di esempio, a patto di essere in grado di giustificare le proprie scelte. Il giudizio finale terrà conto della complessità del problema proposto e dalla bontà dell'esecuzione.

CONSEGNA

ESERCIZIO 1.

Si sviluppi un controllore per il sistema fisico cart-pole su ambiente simulato.

- Il sistema Cart-Pole è costituito da un pendolo invertito montato su un carrello mobile. Il carrello è libero di muoversi avanti e indietro lungo un asse fissato (y).
- Il controllore da implementare ha l'obiettivo di stabilizzare il pendolo nella posizione verticale, agendo sul movimento del carrello.
- Impiegare l'ambiente di simulazione fornito (https://github.com/cscribano/gazebo_polecart_ros), basato su Gazebo-classic e ROS2 per la comunicazione con il simulatore.

Si *consiglia* di procedere come segue:

- Studiare scrupolosamente la documentazione fornita per il simulatore (Readme dal repo github).
- Clonare, poi compilare il pacchetto ROS2 fornito utilizzando il sistema colcon, quindi eseguire l'ambiente tramite ROS2 come indicato dalla documentazione.
- Controllare che siano effettivamente visibili i topic ROS2 utilizzati dal simulatore, utilizzando l'interfaccia ROS2 a riga di comando (tip: elencare topic publicati, pubblicare controllo manuale come da documentazione).
- Creare un NUOVO pacchetto ROS2 da utilizzare per l'implementazione della soluzione proposta.
- Implementare la soluzione in C++ utilizzando le API ROS2 viste a lezione.
- Implementare un nuovo nodo ROS con publisher e subscriber per i topic opportuni.
- In particolare:

- Leggere lo stato del pendolo dal topic /cart/pole_state
- Aggiornare il ciclo di controllo
- Pubblicare il valore di controllo sul topic /cart/force
- Implementare il ciclo di controllo per mezzo di un semplice controllore PID

Suggerimenti:

- Lo stato del pendolo, pubblicato sul topic /cart/pole_state come messaggio di tipo JointState, identifica l'angolo (in radianti) rispetto alla posizione verticale del pendolo (come indicato nella documentazione). Il set-point è quindi un valore di 0.0 radianti.
- È possibile muovere il carrello applicando una forza lungo l'asse y (positiva o negativa) pubblicando sul topic /cart/force un messaggio di tipo Wrench (come indicato nella documentazione).
- Prima di iniziare il ciclo di controllo, è consigliato eseguire il reset della simulazione per partire dalla condizione iniziale con il pendolo in verticale. A questo scopo si consiglia di invocare il servizio di reset (indicato nella documentazione). È consigliabile invocare il reset all'avvio del nodo implementato, utilizzando le API C++ (ROS2 service CLIENT).
- Come sottolineato nella documentazione, l'effetto della forza applicata NON svanisce nemmeno dopo il reset del simulatore. È quindi necessario applicare una forza di 0.0 N sull'asse y se si intende annullare gli effetti dei controlli precedenti e arrestare il carrello.
- Il delta di tempo (dt) utile per le operazioni di derivazione e integrazione può essere assunto pari a 50 millisecondi (tenendo in conto di una frequenza di pubblicazione di 20hz).
- È anche possibile (ma non obbligatorio), ricavare dt come differenza fra i Timestamp di due messaggi di stato successivi.

Note finali:

Il "tuning" del controllore PID può richiedere diversi tentativi per individuare i guadagni ottimali per un controllo stabile. SE non si dovesse riuscire a tunare correttamente il controllore (e per tanto il pendo non sarà del tutto stabile) la soluzione PUO comunque essere accettabile A PATTO CHE LA SOLUZIONE IMPLEMENTATA SIA COMUNQUE TECNICAMENTE CORRETTA.

ESERCIZIO 2.

Di seguito si propone UNA POSSIBILE traccia per il secondo esercizio.

Il candidato può attenersi alle indicazioni della traccia o proporre tutte le variazioni che desidera, a patto che il livello di complessità sia paragonabile o superiore e che si faccia uso delle medesime tecnologie fondamentali.

L'unico requisito fondamentale è che si contempi un problema di navigazione autonoma di un robot in ambiente simulato, utilizzando ROS2 e gli strumenti affini trattati a lezione.

Traccia di ESEMPIO

Viene predisposto un ambiente simulato utilizzando il simulatore (visto a lezione) Turtlebot3 basato su Gazebo. Si vuole realizzare un sistema autonomo per navigare l'ambiente circostante alla ricerca di un marker noto (di tipo ArUco), che si vuole localizzare all'interno della mappa.

Utilizzando ROS2 e lo stack Nav2, si implementi il seguente comportamento:

- Viene avviato l'ambiente simulato, viene se necessario inizializzato lo stack di navigazione Nav2
- Viene posizionato (anche manualmente) in un punto della mappa il cubo con i marker aruco (ref. slide)
- Inizialmente il robot naviga autonomamente per la mappa.
- Simultaneamente viene analizzato il topic della camera simulata per individuare il marker Aruco.
- Quando il marker è individuato, viene stimata la posa (ref. slide), publicata la trasformazione (TF) del marker nel sistema di globale della mappa.

Variazioni suggerite:

- Il robot può navigare visitando in successione una serie di punti fissi forniti (FACILE) o è possibile calcolare dinamicamente il prossimo punto verso cui dirigersi (esempio: utilizzando il laser scanner, PIU AVANZATO).
- Quando viene trovato il marker sarebbe preferibile impostare automaticamente la navigazione in modo da fermarsi nelle vicinanze del marker localizzato. Alternativamente è possibile arrestarsi sul posto o proseguire la navigazione in cerca di altri eventuali marker.

Suggerimenti:

- Si faccia riferimento alle slide del 7/6 per ulteriori dettagli. Il modello del marker cubico da inserire in Gazebo e il codice di esempio per pose estimation del marker sono forniti nella cartella "**starter pack**" (su Moodle)
- Non è necessario l'utilizzo esplicito o creazione di Behaviour Tree (argomento non trattato a lezione).

Note finali:

- La capacità di proporre e risolvere variazioni rispetto allo scenario suggerito verrà valutata positivamente.

PRESENTAZIONE DEL PROGETTO

- I progetti possono essere discussi in una data da concordare.
- Per l'esercizio 1 è sufficiente portare il codice e dimostrarne conoscenza approfondita.
- Per l'esercizio 2 è possibile aiutarsi con una semplice presentazione, in particolare qualora si siano proposte variazioni originali rispetto la traccia.

**** F.A.Q ****

1. è necessario sviluppare esclusivamente in C++?

- Si. Nel caso malaugurato di lacune incolmabili rivolgersi al docente.

2. non ho un computer / il mio computer non è all'altezza.

- In questo caso è possibile concordare un progetto adeguato alle risorse, oppure un progetto che usi le risorse che è possibile fornirvi (es., board jetson...)

3. Ho completato l'esercizio 1, l'esercizio 2 invece è parziale e non funziona benissimo.

- Anche se parziale è comunque più di niente.

Per eventuali dubbi non coperti da questo documento non esitare a contattare carmelo.scribano@unimore.it per ottenere chiarimenti.