GUIDA ALL'USO DEI CODICI SVILUPPATI NELLA TESI "REINFORCEMENT LEARNING PER IL CONTROLLO DI UN QUADRICOTTERO"

GIORGIO DI LIBERI

1 INTRODUZIONE

Questo documento contiene una guida per l'uso della cartella "ReinforcementLearning" contenete tutti i codici sviluppati per il lavoro di tesi "Reinforcmente Learning per il controllo di un quadricottero". Le informazioni contenute nel documento spiegano come usare i codici ma non richiamano la teoria spiegata nella tesi; i codici sono ampiamente commentati per favorire la lettura e la modifica.

2 CONFIGURAZIONE E PACCHETTI NECESSARI

Si raccomanda l'uso di un sistema operativo basato su Linux, in particolare nel lavoro è stato usato Ubuntu 20.04 LTS, è conveniente sfruttare l'interfaccia del terminale o, alternativamente, l'editor VS code implementa diverse funzioni molto utili, tra cui il terminale integrato. I codici sono interamente disponibili nella repository Git-hub dell'autore scaricabili installando il software git sul proprio pc e usando il comando *git clone*.
Il frame-work principale utilizzato è *Stable Baselines 2* (SB2)
(https://stable-baselines.readthedocs.io/en/master/index.html), alcuni codici sono stati sviluppati anche per *Stable Baselines 3* anche se nel momento in cui è stato svolto il lavoro questa versione non era ancora matura e non è stata usata in maniera definitiva.
I programmi richiesti sono *python 3.7.x* e *Tensorflow* (per il lavoro è stato usato *python 3.7.9* e *TF 1.15*) si raccomanda l'uso di un ambiente virtuale per python 3.7.x (creato con Conda o con il comando venv).
La versione SB3 gira anche con *python 3.8*.

3 ORGANIZZAZIONE DELLA CARTELLA

La cartella principale contiene diverse sotto cartelle:

* *Ingegneria Aeronautica, Università di Roma "La Sapienza"*

- *Stable_ Baselines2_ Frame*: contiene i codici con gli ambienti per eseguire l'allenamento usando il framework SB2; questa cartella è stata la più aggiornata durante il lavoro. L'ambiente di Quad implementato in tutti i codici di questa cartella prevede effetti aerodinamici lineari rispetto a velocità e velocità angolare; si raccomanda l'uso di questi codici per allenare delle *policies* ed ottenere leggi di controllo per un quadricottero;
- *Stable_ Baselines3_ Frame*: contiene i codici per eseguire l'allenamento usando SB3; come anticipato questo frame-work non è completamente maturo al momento dello svolgimento della tesi e dunque non è stato usato massicciamente, tuttavia alcuni codici sono implementati e possono essere modificati qualora venga aggiornata la libreria;
- *Hummingbird*: l'ambiente di quad implementato in questa cartelle prevede gli effetti di flappeggio ed influsso sui rotori. Questo modello non è risultato ideale per eseguire le procedure di allenamento, è stato usato per validare le prestazioni delle *policies* allenate con i codici nella cartella "*Stable_ Baselines2_ Frame*";
- *Test_ codes*: contiene alcuni codici per test e prove sulle funzioni base di python e delle librerie;

4 STABLE BASELINES2 FRAME

Questa cartella contiene i codici per l'allenamento di *policies* nel controllo di un quadricottero per la risoluzione di diversi problemi. In particolare:

- *QuadEnvTest_ 6DOF*: contiene i codici per allenare e testare una *policy* al controllo di posizione di un quadricottero, le caratteristiche di questa rete sono riportate nel capitolo 4 della tesi;
- *QuadEnvTest_ 6DOF_ Att*: contiene i codici per allenare una *policy* al controllo di assetto (vedi ch4 tesi);
- *QuadEnvTest_ 6DOF_ NoisyBuild*: contiene i codici per ottenere un controllore per il controllo di posizione con dei file di configurazione che inizializzano i parametri costruttivi in modo diverso ad ogni reset; questi codici sono pensati per allenare una rete di tipo LSTM che sia robusta allo scattering dei parametri; tuttavia l'allenamento non è stabile e dopo alcuni aggiornamenti delle reti il run viene ucciso automaticamente per esaurimento della memoria disponibile;
- *QuadEnvTest_ 6DOF_ Vectorial*: contiene i codici per ottenere un controllore per la navigazione vettoriale.

4.1 uso della cartella

Ognuna delle cartelle contiene gli stessi codici, la differenza tra le cartelle risiede nell'interfaccia tra rete neurale e ambiente controllato. E' necessario aprire la cartella che si desidera usare nel terminale e lanciare i codici con il comando

```
python <nome_codice>.py
```

quadcopt_ 6DOF.py

Il codice *quadcopt_ 6DOF.py* implementa l'ambiente del quadricottero con effetti aerodinamici lineari rispetto alla velocità e velocità angolare. L'ambiente è implementato nella forma di una classe. Nella funzione di inizializzazione sono impostati

tutti i parametri dell'ambiente, questi parametri restano memorizzati e possono essere richiamati dalle funzioni della classe o dall'esterno. Nella classe sono implementate tutte le funzioni necessarie e realizzare l'ambiente di quadricottero.

La funzione *step()* implementa l'integrazione delle equazioni del moto, ricevendo in ingresso le azioni e restituendo in uscita lo stato dell'ambiente al passo temporale successivo. Inoltre la funzione valuta le osservazioni che costituiscono l'input per la rete attore.

La funzione *reset()* valuta lo stato iniziale dell'ambiente, viene richiamata all'inizio di un episodio o di una simulazione.

La funzione *isDone()* valuta il termine dell'episodio in base allo stato dell'ambiente.

La funzione *getReward()* restituisce il guadagno ad ogni passo temporale dato lo stato dell'ambiente.

Le restanti funzioni implementano le azioni del quadricottero, mentre la funzione *equationsOfMotion()* calcola la derivata temporale delle variabili di stato a partire dalle forze e dai momenti.

actionTest.py

Questo codice permette di testare alcune azioni definite dall'utente attraverso la variabile *action*.

EnvTest.py

Questo codice permette di scrivere e testare le caratteristiche della classe implementata in quadcopt_ 6DOF.

Training.py

Questo codice consente di effettuare l'allenamento di una *policy* per il problema implementato nella classe quadcopt. Se ad esempio nella classe è impostata una rete per il controllo di posizione, attraverso la definizione del guadagno, delle osservazioni e del reset, allora attraverso l'allenamento si otterrà una *policy* per il controllo di posizione.
Il codice è lanciato con il comando:

```
python Training.py
```

Nel codice è possibile modificare i parametri dell'algoritmo per l'allenamento. La variabile *LearningTimeSteps* è il numero totale di passi temporali impostati come *budget* per l'allenamento, la variabile *cpu* deve essere impostata pari al numero di *thread* del processore che svolge il calcolo e sarà pari al numero di ambienti paralleli. Gli altri parametri sono relativi all'algoritmo, il set di questi va impostato in relazione al problema che si intende risolvere.
La cartella */tensorboardLogs/* contiene i file di tensorboard utili per osservare a posteriori l'andamento del *training* attraverso l'osservazione dell'andamento di alcuni parametri, ad esempio la somma dei guadagni per un episodio.
La cartella */EvalClbkLogs/* contiene la *best_ model* ovvero la *policy* che durante l'allenamento ha totalizzato la massima somma dei guadagni in un episodio.
La cartella */Policies/* contiene la *policy* finale ottenuta e salvata; Attraverso il file salvato può essere richiamata la rete attore salvata per utilizzarla come controllore.

simulator.py

Questo codice consente di valutare le prestazioni di una rete attore salvata, attraverso una simulazione. In particolare la parte iniziale del codice inizializza le variabili per il salvataggio e per la realizzazione dei grafici. Il ciclo *for* implementa la simulazione vera e propria, all'inizio di ogni passo del ciclo vengono impostati i valori di riferimento mentre le righe:

```
action, _state = model.predict(obs, deterministic=True)

obs, reward, done, info = env.step(action)
```

sono rispettivamente la valutazione delle azioni e l'elaborazione dello stato del quad al passo successivo date le azioni. La variabile info contiene lo stato al passo temporale in esame e viene usata per riempire i vettori con i dati per il salvataggio e i plot.

Infine la cartella */SimulationResults/* contiene i grafici salvati a valle di una simulazione.

5 HUMMINGBIRD

Questa cartella contiene i codici per l'implementazione dell'ambiente quad basato sull'Asc.tech. Hummingbird. Il modello di simulazione in questa cartella superano gli effetti aerodinamici lineari rispetto alla velocità, considerando effetti di influsso e flappeggio sui rotori. Questi modelli non hanno mostrato un buon comportamento per gli allenamenti mentre sono stati usati per effettuare le simulazioni di validazione delle *policies* allenate con i codici in *Stable Baselines2 Frame*. La procedura dunque è stata di allenare delle reti attraverso i codici in *Stable Baselines2 Frame* ed in seguito copiare il file relativo alla *policy* nella relativa cartella in *Hummingbird* e lanciare il codice *simulator.py*.
Le cartelle presenti sono:

- *Hummingbird_ env*: implementa l'ambiente per la simulazione del controllore di posizione allenato con l'ambiente nella cartella *QuadEnvTest_ 6DOF*;
- *Hummingbird_ env_ Att*: implementa l'ambiente per la simulazione del controllore di assetto allenato in *QuadEnvTest_ 6DOF_ Att*;
- *Hummingbird_ env_ Vref*: implementa l'ambiente per la simulazione del controllore per la navigazione vettoriale senza controllo sull'angolo di rotta;
- *Hummingbird_ env_ Vref_ Psiref*: implementa l'ambiente per la simulazione del controllore per la navigazione vettoriale allenato in *QuadEnvTest_ 6DOF_ Vectorial*;
- *Hummingbird_ env_ diffParams*: implementa l'ambiente per la simulazione del controllore di assetto su un ambiente con parametri modificati.

5.1 Uso della cartella

L'uso di ogni cartella è analogo a quello spiegato nella cartella precedente, tuttavia in questo caso il codice più importante è *simulator.py*. E' possibile richiamare il codice aprendo una delle cartelle nel terminale e digitando il comando:

```
python simulator.py
```

Il codice può essere modificato nella parte iniziale del ciclo *for* per modificare i riferimenti e realizzare diverse tasks.

La cartella *Hummingbird_ env_ Vref_ Psiref* implementa un controllore proporzionale che genera i riferimenti vettoriali per la *policy* in base all'errore di posizione. Nel codice *simulator.py* della cartella viene impostata una lista di *waypoints* per la realizzazione di una missione di volo.

Nella cartella *Hummingbird_ env* e *Hummingbird_ env_ diffParams* è implementato il controllo di posizione a partire da conzioni iniziali casuali. Il codice *MonteCarlo.py* consente di relaizzare una serie di simulazioni con condizione iniziale casuale, salvare le traiettorie e tracciare i grafici delle prestazioni del controllore nelle simulazioni. Il resto dei codici hanno un funzionamento analogo alla cartella sopra descritta.