SVS - Assignment 2

Andrea Serafini, Alessandro Martignano, Angelo Tinti, Piero Sanchi

March 2023

1 Requisiti di sistema

L'assignment consiste nell'addestrare due squadre di droni quadrirotore ognuno con la facoltà di sparare cinque sfere nel corso di una sessione di simulazione, con lo scopo di sconfiggere la squadra opponente.

Vincoli di sistema:

- i droni devono comparire in posizioni fisse ed essere rivolti verso la squadra nemica;
- i droni posso sparare le sfere con un rateo di fuoco massimo di una per secondo.
- le sfere sono 5 per ogni drone ed hanno dimensione fissa;
- la policy deve essere in grado di utilizzare informazioni sia dal drone a cui è applicata sia da tutti i droni nella stessa squadra;
- la battaglia deve iniziare tre secondi dopo l'avvio della simulazione per evitare spawnkill.

2 Architettura proposta

Dopo una prima fase di ricognizione del codice abbiamo suddiviso il lavoro in diverse fasi, brevemente descritte nei seguenti due paragrafi.

2.1 Impostazioni dell'ambiente

La prima fase è stata quella di set up dell'ambiente, in cui abbiamo modificato il codice allo scopo di operare con un numero pari variabile di velivoli, con l'obiettivo di lanciare training ad otto droni fin da subito. Una volta uniformati gli indici dei quadricotteri allo scopo di suddividerli correttamente in due squadre e far sì che fossero rivolti alla squadra nemica, abbiamo iniziato a concentrarci sulle sfere. Per quanto riguarda queste ultime abbiamo modificato il codice esistente così che venissero create al di sotto del drone. La scelta ci ha consentito di incrementare la precisione della palla lanciata ed evitare autocollisioni. Le sfere vengono espulse a velocità costante sull'asse x e direzionate verso il campo visivo del drone grazie a velocità variabili applicate agli altri assi. Abbiamo imposto i vincoli di sparo con rateo massimo di una sfera al secondo e punito ogni tentativo di sparo nei primi 3 secondi di esecuzione come da specifiche.

Lo spazio delle osservazioni era stato in precedenza definito nel codice e quindi non è stato modificato ulteriormente in quanto combinava già i valori dell'osservazione di tipo KIN e l'immagine ripresa dalla telecamera del drone dopo essere stata opportunamente rielaborata. Allo stesso modo è stata fatta la scelta di utilizzare lo spazio delle azioni preimpostato, composto dal dizionario contente le azioni in VEL e l'azione di SHOOT tipica del nostro ambiente.

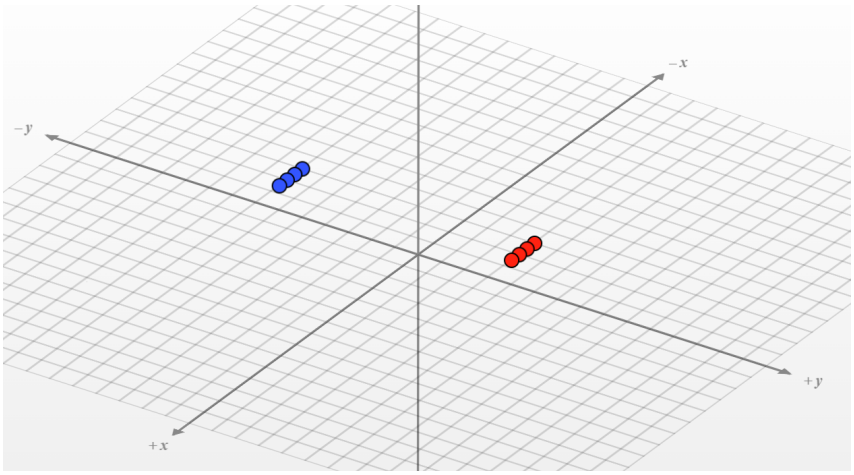

Figura 1: Disposizione dei droni.

Uno dei problemi con cui ci siamo dovuti interfacciare in questa fase è stato la gestione della sconfitta di un drone. Non essendoci possibile terminare l'esecuzione per un singolo drone, abbiamo deciso di sperimentare varie tecniche per aggirare il problema, per ultima è stata scelta quella di non considerare "morto" un drone, bensì di penalizzarlo fortemente in caso vengano letti dei valori, a nostra discrezione, indicativi del suo abbattimento, come ad esempio una prossimità al suolo maggiore di quella di partenza o una velocità lineare su uno degli assi troppo elevata per essere giustificabile da uno spostamento volontario.

2.2 Metodologia di lavoro

Per lo svolgimento del progetto abbiamo cercato di seguire le best practices suggerite dai docenti, nello specifico:

- **Use automatic hyper-parameters search framework**: È stato utizzato ray tune.
- **Start with default hyper-parameters**
- **Start simple**: Si è iniziato addestrando solamente 2 droni e con una policy minimale, per poi estenderla e aggiungere droni.
- **Normalize the observation space**: I valori delle osservazioni sono stati normalizzati.
- **Start with a continuous action space normalized than rescale the value if needed and consider discrediting the space**: Abbiamo adottato uno spazio delle azioni continuo.
- **Consider multiple reward functions**: Abbiamo adottato diverse reward, sia di tipo sparse che shaped, come dettagliato in seguito.
- **Termination condition**: Sono stati impostati dei timeout per la fine degli episodi. Non è stato tuttavia possibile implementare una condizione di early-stopping relativa alla morte di tutti i droni.

2.3 Meccanismo di Reward

Le prime operazioni svolte sulla reward di partenza sono state quelle per gestire lo spawnkill dei droni. Dai primi training abbiamo subito notato che con otto droni i tempi risultavano molto lunghi e non ci permettevano di raggiungere risultati tangibili. Abbiamo così deciso, almeno in una prima fase, di trainare il modello con due soli droni allo scopo di poter testare più reward possibili.

In una seconda fase ci siamo concentrati sulla mira. Abbiamo correlato la reward al numero di pixel presenti nel campo visivo di un drone colorati con i colori della squadra avversaria con l'obiettivo di addestrarli ad avvicinarsi nel tentativo di aumentarli ed incrementare così la precisione del colpo e le capacità di movimento del velivolo.

La strategia adottata inizialmente prevedeva l'utilizzo di un "cannone" fisso verso il centro del campo visivo del drone, lasciando ad esso l'onere di muoversi per orientarlo verso il nemico. Era stata prevista in questo scenario una reward incrementale, basata sulla prossimità del drone nemico al centro dell'inquadratura. Dopo svariati training senza risultati tangibili abbiamo ipotizzato che il problema risultasse troppo complesso per permettere apprendimenti in tempi così brevi.

A questo punto abbiamo recuperato la logica precedentemente implementata nel codice e creato una sorta di "cannone" mobile che applica la velocità alla sfera in base alla posizione dei pixel nemici nel campo visivo dell'attaccante per alleggerire il carico di elaborazione.

Listing 1: Reward definitiva

```
def _computeReward(self):
    rewards = {}
    states = np.array([self._getDroneStateVector(i) for i in range(self.NUM_DRONES)])

    for i in range(self.NUM_DRONES):
        if states[i, 2] < 0.5 or states[i, 3] > 0.5 or
                states[i, 4] > 0.5 or states[i, 5] > 0.5: # sono stato atterrato
            rewards[i] = -1
        elif self.step_counter/self.SIM_FREQ < 3 and array_target[i][4]: # sparo nei primi 3s
            rewards[i] = -1
        else:
            rewards[i] = 0
            kills = 0
            for avv in range(self.NUM_DRONES):
                if i%2 != avv%2 and
                        (states[avv, 2] < 0.5 or states[avv, 3] > 0.5 or
                        states[avv, 4] > 0.5 or states[avv, 5] > 0.5):
                    kills = kills + 1
            rewards[i] = rewards[i] + (1*kills)
            rewards[i] = rewards[i] + (0.05*RESULT_MASK[i]) # n pixel nemici visti

            if MAX_SPHERES[i] == 0 and kills == 0:
                rewards[i] = -0.5

    return rewards
```

Avendo svolto la maggior parte dei training con due droni per poter raggiungere un numero soddisfacente di iterazioni senza che il tempo necessario crescesse esponenzialmente è stata poi tenuta la suddivisione delle due policy anche dopo aver aumentato il numero di componenti della squadra. L'assegnazione delle policy avviene quindi con il seguente criterio:

```
drone_index % 2 == 0 → blue team (policy 0)
drone_index % 2 == 1 →  red team (policy 1)
```

3 PPO e tuning degli iperparametri

Come algoritmo per l'addestramento degli agenti si è scelto di utilizzare PPO (Proximal Policy Optimization).

PPO è un algoritmo di apprendimento per rinforzo (RL) utilizzato per addestrare agenti di intelligenza artificiale a compiere decisioni in un ambiente dinamico. PPO si basa sul concetto di politiche (policy) nel RL, che rappresentano la strategia dell'agente per scegliere le azioni in base allo stato dell'ambiente. L'obiettivo dell'apprendimento per rinforzo è di trovare la politica ottimale che massimizza la ricompensa accumulata nel tempo. L'algoritmo PPO è stato sviluppato per superare alcuni dei limiti degli algoritmi precedenti come TRPO e DDPG. Utilizza una tecnica chiamata "clipping" per evitare grandi aggiornamenti alle politiche durante l'addestramento e limitare la variazione tra le vecchie e le nuove politiche. Ciò rende PPO molto stabile e adatto per l'addestramento di agenti di intelligenza artificiale in ambienti dinamici e complessi, dove la politica dell'agente deve essere aggiornata frequentemente per adattarsi a nuove situazioni.

Per valutare gli iperparametri migliori con cui configurare PPO si è optato per una strategia automatizzata, fornita da Ray Tune, componente della libreria RLlib.

Sono stati presi in considerazione i seguenti iperparametri:

- **entropy_coeff**: questo parametro controlla il contributo della "entropy loss" nell'aggiornamento del modello durante l'addestramento. L'entropia è una misura dell'incertezza della distribuzione di probabilità sulle azioni del modello, e l'aggiunta di un termine di entropia può aiutare a evitare di cadere in minimi locali e migliorare la diversità delle azioni generate dal modello. Il valore di entropy_coeff controlla l'importanza dell'entropia rispetto alla "policy loss" nell'aggiornamento del modello. Nel caso della configurazione proposta, entropy_coeff è scelto da una distribuzione loguniforme con valori possibili compresi tra 0.00000001 e 0.1.

- **lr**: questo parametro rappresenta il "learning rate" utilizzato durante l'aggiornamento dei pesi del modello. Il "learning rate" controlla la dimensione del passo che il modello compie durante la discesa del gradiente, e deve essere scelto in modo da garantire che l'aggiornamento dei pesi del modello sia stabile e converga rapidamente. Nel caso della configurazione proposta, "lr" è scelto da una distribuzione loguniforme con valori possibili compresi tra 5e-5 e 1.

- **sgd_minibatch_size**: questo parametro rappresenta la dimensione del minibatch utilizzato durante l'aggiornamento dei pesi del modello. Il minibatch è un sottoinsieme del dataset di addestramento utilizzato per aggiornare i pesi del modello, e la dimensione del minibatch può influire sulla stabilità e sulla velocità di convergenza del modello. Nel caso della configurazione proposta, "sgd_minibatch_size" può assumere valori discreti tra 32, 64, 128, 256 e 512.

- **lambda**: questo parametro controlla il trade-off tra il "policy loss" e la "value loss" nell'aggiornamento del modello. Il "value loss" è una misura dell'errore nella stima della funzione di valore del modello, mentre il "policy loss" è una misura dell'errore nella stima della politica del modello. Nel caso della configurazione proposta, "lambda" può assumere valori discreti tra 0.1, 0.3, 0.5, 0.7, 0.9 e 1, e controlla il trade-off tra "policy loss" e "value loss" nell'aggiornamento del modello.

Di seguito, il codice di configurazione di ray tune con cui si è implementata la ricerca

```
ppo_params = {
    "entropy_coeff": tune.loguniform(0.00000001, 0.1),
    "lr": tune.loguniform (5e-5, 1),
    "sgd_minibatch_size": tune.choice ([ 32, 64, 128, 256, 512]),
    "lambda": tune.choice ([0.1,0.3, 0.5, 0.7,0.9,1.0])
}
```

In sintesi, questi quattro parametri rappresentano i principali iperparametri dell'algoritmo PPO e controllano aspetti come la regolarizzazione, la velocità di apprendimento e la stabilità dell'algoritmo. La configurazione proposta utilizza una combinazione di distribuzioni continue e discrete per definire l'ipotesi di ricerca per i valori degli iperparametri, al fine di esplorare un'ampia gamma di valori possibili e trovare quelli che funzionano meglio per il problema specifico.

4 Analisi delle performance

Le performance sono state valutate principalmente osservando l'esecuzione della simulazione a due o otto droni e misurando la durata del training. In fasi più avanzate di tuning del meccanismo di reward abbiamo monitorato l'andamento degli addestramenti anche utilizzando i grafici predisposti da tensorboard, brevemente illustrati nella seguente sezione.

4.1 TensorBoard

Per valutare la qualità dell'addestramento ci si è basati sull'andamento di alcune metriche, monitorate tramite tensorboard.

4.2 Reward

La reward è una misura di quanto bene l'agente sta eseguendo la sua attività. Idealmente, si vuole che la reward aumenti nel tempo, poiché ciò indica che l'agente sta imparando a svolgere il suo compito in modo più efficace. Nel nostro caso, la reward aumenta sino a raggiungere una sorta di plateau.

Figura 2: Andamento della reward

4.3 Loss

La loss è una misura di quanto l'algoritmo sta sbagliando nella sua previsione dell'azione da intraprendere. Si vuole che la loss diminuisca nel tempo, poiché ciò indica che l'algoritmo sta imparando a fare previsioni più accurate. Nel nostro caso, la loss cala sino a raggiungere un minimo.

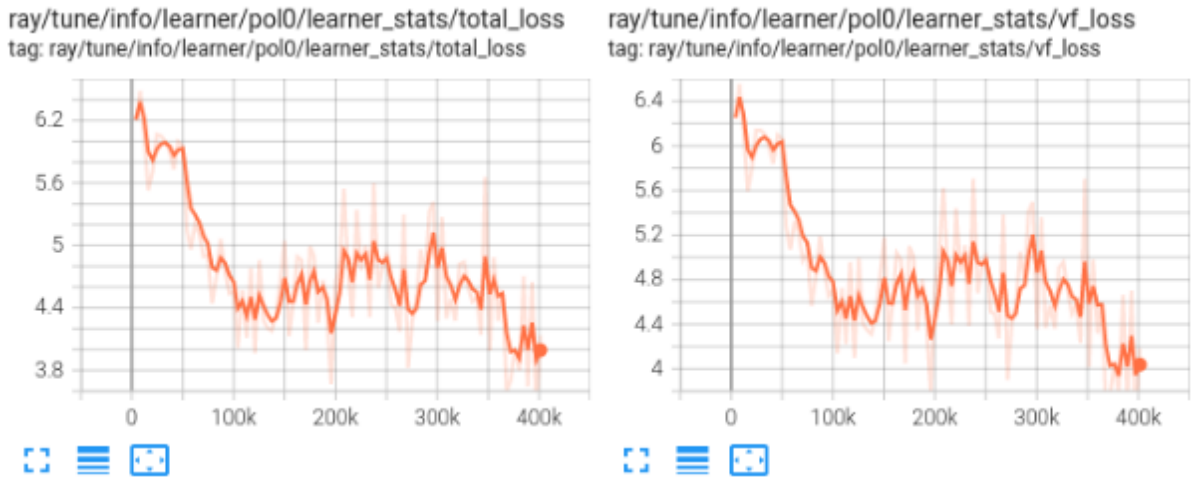

Figura 3: Andamento della loss

4.4 Entropy

L'entropia è una misura della diversità delle azioni che l'agente sta eseguendo. In generale, si vuole che l'entropia sia alta all'inizio dell'addestramento e poi diminuisca nel tempo, poiché questo indica che l'agente sta esplorando diverse azioni per trovare la strategia migliore, ma sta diventando sempre più sicuro della sua scelta. Nel nostro caso, l'entropia decrementa nel tempo ma tende ad aumentare nuovamente una volta che la reward raggiunge il plateau. Questo è probabilmente dovuto dal fatto che l'agente, una volta raggiunta la massima reward, tenta modi sempre più "creativi" per superarsi.

Figura 4: Andamento dell'entropia

4.5 KL (Kullback-Leibler) divergence

È un concetto importante nel reinforcement learning e in particolare nell'algoritmo PPO (Proximal Policy Optimization). La KL divergence è una misura della differenza tra due distribuzioni di probabilità. Nel contesto di PPO, una delle distribuzioni di probabilità è la politica attuale dell'agente, mentre l'altra è la politica che viene aggiornata durante l'addestramento. La KL divergence viene utilizzata per regolare la quantità di aggiornamento della politica per assicurarsi che le nuove politiche siano abbastanza vicine alla vecchia politica per evitare aggiornamenti troppo grandi, che potrebbero causare un'instabilità dell'algoritmo. Nel nostro caso la kl segue una parabola discendente.

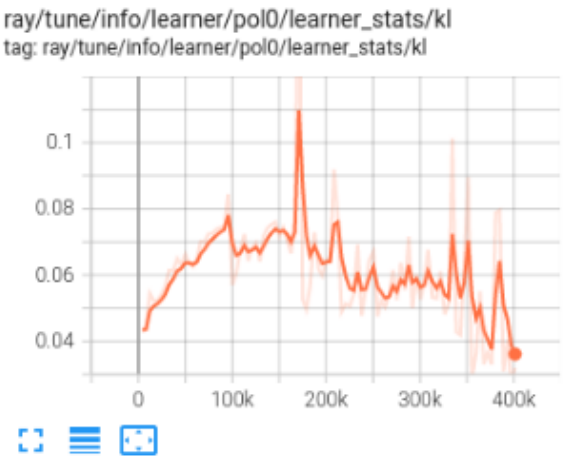

Figura 5: Andamento della kl-divergence

4.6 Valutazione del modello

Un altro metodo che si voleva utilizzare per migliorare la scelta del modello e per poterlo analizzare più approfonditamente è l'utilizzo dei checkpoints. Questi, data una frequenza relativa al numero di iterazioni avvenute in una simulazione, avrebbero dovuto salvare lo stato in quell'istante permettendo di visionare i diversi dati della simulazione come i livelli di reward (minima, massima, media ecc.) e anche i valori della loss function. Purtroppo, quello che si è riusciti ad ottenere è stato un unico checkpoint per simulazione contenente l'ultimo stato raggiunto. Ciò ha impedito di analizzare più approfonditamente i dati citati. Cercando nella documentazione online e controllando diverse *issues* presentate nei diversi siti, non è stato comunque raggiunto il risultato sperato.
Si è riusciti, ad ogni modo, ad aumentare il numero di simulazioni per addestramento grazie alla configurazione passata a all'elemento `tune.run` della libreria `ray` utilizzata nel progetto. Così facendo è risultato più semplice osservare come, su diverse simulazioni, variava il livello raggiunto dal modello (ricordando che ad ogni simulazione la libreria `gym` crea un ambiente casuale portando quindi ad ogni simulazione possibili risultati differenti).

4.7 Sviluppi futuri

Tra i possibili sviluppi futuri ipotizzati abbiamo analizzato alcuni scenari che potrebbero essere interessanti da approfondire.

Una possibile strada avrebbe potuto prevedere l'introduzione di una reward più correlata al task "secondario" di movimento piuttosto che a quello dell'abbattimento per favorire un apprendimento orientato all'attacco o alla fuga e vedere così un modello più mobile all'interno dell'ambiente. Per ottenere ciò bisognerebbe implementare una reward che punisca la permanenza prolungata di un drone nello stesso punto, e/o premiare movimenti di sufficiente entità.

Nel tentativo poi di comprendere i motivi per cui non si sono evidenziate in fase di evaluation delle soluzioni apprese in risposta al problema del massimizzare il numero di pixel del colore nemico all'interno del campo visivi abbiamo cercato di analizzare i dati a nostra disposizione. Andando a plottare a video le immagini recepite dalla telecamera di un drone è stato possibile identificarne una possibile causa. Come si può vedere anche dalle figure seguenti, osservando il punto di vista da 10 metri, ovvero dalla distanza "regolamentare" definita nelle specifiche, i droni appaiono grandi come un singolo pixel.

Figura 6: Deployment dei droni a 10 metri di distanza

Cambiando il punto di origine di una delle due squadre riducendo così a 5 metri la distanza frontale dagli avversari si può vedere come la dimensione del singolo drone osservato aumenta a soli 2 pixel.

Figura 7: Deployment dei droni a 5 metri di distanza

Abbiamo perciò ipotizzato che una reward definita per crescere in maniera direttamente proporzionale al numero di pixel non potesse essere efficiente con un rapporto spostamento/pixel così sfavorevole. Inoltre si è osservato come la forma prevalentemente assottigliata del drone gli permetta di "sparire" dall'inquadratura se inclinato con la stessa orientazione della camera.

Una possibile soluzione per questo problema potrebbe essere quella di aumentare la risoluzione dell'immagine rilevata dalla telecamera del drone per rendere più fedele un eventuale avvicinamento al bersaglio. Così facendo si andrebbe però ad aumentare il numero di informazioni passate come osservazione per cui potrebbe essere utile prevedere anche un sistema di pre-processing dell'immagine in maniera da ridurre il numero di valori passati, magari sostituendola con degli indici di aggregazione relativi alla posizione del nemico nell'inquadratura e alla sua dimensione.

5 Considerazioni finali

L'assignment complessivamente è stato molto interessante, tuttavia riteniamo di esserci scontrati con un task troppo complesso per le nostre disponibilità, principalmente di tempo e risorse.
Alcuni vincoli ci sono stati imposti dalle limitazioni intrinseche nel simulatore utilizzato, come ad esempio l'impossibilità di terminare l'esecuzione per il singolo drone, discussa precedentemente.

In conclusione ci consideriamo soddisfatti dei risultati ottenuti, consapevoli che il modello addestrato non rispecchia standard qualitativi che sarebbero stati chiesti in un contesto differente, ma ritenendo di aver impostato accuratamente l'ambiente e la reward, necessitando però di più tempo per effettuare il tuning degli iperparametri ed addestramenti sufficientemente lunghi.

L'interfacciamento con questo tipo di tecnologie è stato tuttavia di grande aiuto alla comprensione "under the hood" per problemi definibili di reinforcement learning e riteniamo di aver fatto quanto più possibile per avvicinarci al risultato atteso, con traguardi più che ottimi tenendo conto di modalità e tempi.