Attenzione: il seguente documento è ancora in fase di stesura, pertanto presenta sezioni abbozzate, incorrette ed incomplete.

Tesi (titolo da definire)

Gianluca Mondini

6 settembre 2018

1 Obiettivo

Implementazione dell'algoritmo di Lloyd per l'equidistribuzione di uno sciame di droni su un'area specifica.

È prevista la realizzazione di un modulo nell'ambiente *OpenModelica* che permetta di simulare il suddetto algoritmo; più avanti verranno illustrate parti di codice e verranno fornite informazioni relative alla sua implementazione.

2 Parole chiave

Prima di procedere con la trattazione, è necessario indicare alcune parole chiave presenti in questo documento e nel codice dell'implementazione

Drone Velivolo che, nel nostro caso di interesse, è rappresentato come un punto bidimensionale

Area Poligono convesso che delimita l'algoritmo;

Cella Porzione di spazio alla quale appartiene un drone; la cella relativa ad un drone è l'insieme di punti dell'area per i quali, tra tutti i vari droni, quello appartenente alla cella è il più vicino

3 Il drone e sue componenti

Possiamo schematizzare, ai fini di questa trattazione, un *drone* mediante 3 componenti: il *centro fisico*, il *controllo di volo* e il *controllo della traiettoria*[1]

Inserire qui uno schemino che illustri i tre componenti del drone

3.1 Centro fisico

Il centro fisico è un astrazione che rappresenta il rapporto del drone con il mondo esterno. Vengono quindi qui considerati parametri la massa, la velocità, l'accelerazione, la posizione, la rotazione, la velocità di rotazione dei motori e la potenza che viene fornita a questi.

[1] da rivedere i nomi dei 3 componenti

4 Algoritmo di Lloyd

4.1 Introduzione

L'algoritmo di Lloyd, conosciuto anche con il nome di *iterazione di Voronoi*, è un algoritmo che permette di suddividere un'area in celle convesse uniformemente dimensionate.

L'algoritmo di Lloyd è implementato mediante iterazioni continue dell'algoritmo di Voronoi.

4.2 Descrizione

L'algoritmo esegue ripetutamente i seguenti *step*:

1. Viene generato il diagramma di Voronoi
2. Per ogni cella trovata, viene determinato il *baricentro*
3. Ogni punto viene spostato in corrispondenza del *baricentro* della propria *cella di Voronoi*

4.3 Esempio di applicazione dell'algoritmo

Viene qui presentata l'applicazione dell'algoritmo di Lloyd ad un'area quadrata nella quale sono presenti 5 partizioni.

Le croci rappresentano i *baricentri* delle varie partizioni.

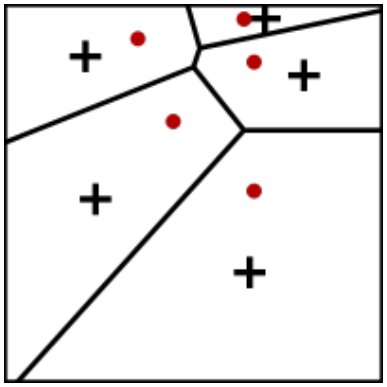

Figura 1: I iterazione

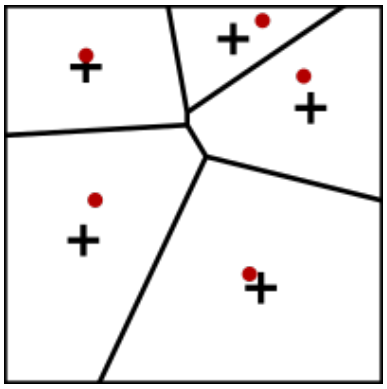

Figura 2: II iterazione

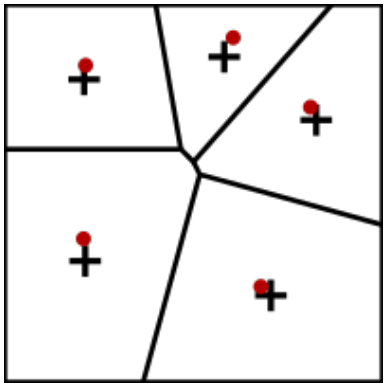

Figura 3: III iterazione

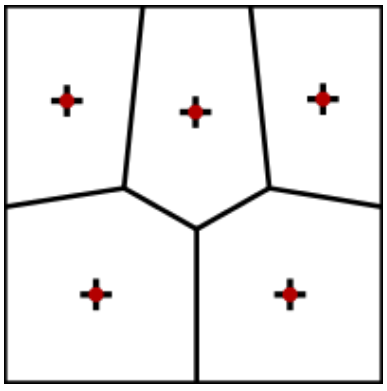

Figura 4: IV iterazione

4.4 Convergenza dell'algoritmo

Intuitivamente, si può dire che l'algoritmo converga in quanto i punti che si trovano a minor distanza tra loro tendono a compiere un movimento più alto, mentre i punti che si trovano a distanze elevate tendono a muoversi meno.

5 Implementazione

5.1 Introduzione

L'implementazione dell'algoritmo di Lloyd richiede una funzione che effettui la tassellazione di Voronoi, la quale a sua volta utilizza una serie di funzioni geometriche che operino nello spazio bidimensionale.

Nell'implementazione che verrà illustrata successivamente, si è scelto di implementare in primo luogo l'algoritmo utilizzando il linguaggio *Python* per avere a disposizione una maggior quantità di strumenti di debugging e testing; successivamente è stato effettuato il porting del codice in *OpenModelica*

5.2 Funzioni e classi geometriche

5.2.1 Point

Per poter operare nello spazio bidimensionale, è necessario definire una classe *Point* rappresentante una tupla *x, y*

```
record Point
  Real x;
  Real y;
end Point;
```

5.2.2 Verifica della sovrapposizione di due punti

Può verificarsi che, in seguito a ..., ci si trovi nella situazione in cui due punti sovrapposti non superino il test di equalità. Ad esempio:

$$P_1 = (4.0, 2.0), P_2 = (4.00001, 1.999999)$$

è chiaro come i due punti siano in realtà sovrapposti, ed è quindi necessario definire una funzione che si occupi di verificarlo

```
function points_are_close
  input Point p1;
  input Point p2;
  output Boolean out;
protected
  parameter Real tolerance = 0.001;
algorithm
  if abs(p1.x - p2.x) < tolerance then
    if abs(p1.y - p2.y) < tolerance then
      out := true;
    else
      out := false;
```

```
    end if;
  else
    out := false;
  end if;
end points_are_close;
```

Il valore *tolerance* può essere variato dipendentemente dalla precisione richiesta e dalla scala di applicazione. Nel caso in cui si operi in una scala metrica, ovvero nel caso in cui *1.0* corrisponda ad *1 metro*, una tolleranza di 0.001 corrisponde ad *1 millimetro*, ed è quindi considerato un valore accettabile dal momento che i droni hanno delle dimensioni nell'ordine di grandezza delle decine di centrimetri.

5.2.3 Edge

Per rappresentare un segmento, viene implementata la classe *Edge*. Ogni Edge è definito come una coppia di punti.

```
record Edge
  parameter Point starting;
  parameter Point ending;
end Edge;
```

Bibliografia

- https://en.wikipedia.org/wiki/Lloyd%27s_algorithm
- github.com/zapateocallisto