# POLITECNICO DI TORINO

DIPARTIMENTO DI INGEGNERIA GESTIONALE E DELLA PRODUZIONE

Corso di Laurea in Ingegneria Gestionale Classe L8 – Ingegneria dell'Informazione


Relazione dell'Esame Finale

## Sviluppo di un'applicazione per simulazione delle consegne di buste leggere con droni in città USA

**Relatore**
*Prof. Fulvio Corno*

**Candidato**
*Marco Pulita s242664*

A.A. 2019/2020

# Contents

# Proposta di progetto

### Studente proponente

s242664 Marco Pulita

### Titolo della proposta

Planner logistico per ritiro e consegna di buste leggere con droni

### Descrizione del problema proposto

Implementazione di un pianificatore per il ritiro e la consegna di buste leggere tra clienti di una piccola città tramite l'uso di droni.

Ogni drone è equipaggiato con un portadocumenti automatico che ritira presso il cliente A una busta di un formato specifico, la registra tramite QRCODE e si avvia per la consegna. La pianificazione dei ritiri e delle consegne è gestita dalla centrale che invia le coordinate di ritiro e consegna al singolo drone. In tempo reale lo scheduler della centrale, conoscendo la posizione dei singoli droni, assegna ai droni in modo ottimale gli ordini di ritiro e consegna basandosi sulla posizione del drone e sulla percentuale di batteria restante. Ogni drone ha un'autonomia delle batterie tale da percorrere 50km, consuma quindi il 2% di batteria ogni km.

La città ha un raggio di 25km e la società di spedizioni ha la sua sede nel centro della città. Il drone può accettare nuovi ordini di lavoro solo se l'autonomia della batteria stimata a consegna completata è almeno del 30%, in caso contrario deve fare rientro e può riprendere le attività il giorno successivo.

Le consegne vengono programmate ogni 30 minuti dalle 9 alle 17 (16 time slot totali). Se un drone non ha consegne in programma per il time slot successivo, dopo aver atteso fermo nel punto dell'ultima consegna del time slot corrente, riceve l'ordine di ritorno in sede ma a differenza del drone con batteria scarica, se la batteria permette ancora consegne può essere riutilizzato pianificando la consegna successiva dalla sede.

Si possono aggiungere o rimuovere droni (ad esempio se necessitano di riparazione) Per semplicità il tempo di percorrenza non viene considerato e le posizioni sono gestite come dati discreti aggiornati ad ogni time slot di 30 minuti.

### Descrizione della rilevanza gestionale del problema

L'applicazione che si vuole implementare soddisfa uno dei più classici problemi di logistica, la consegna.
Qui il vincolo del "carburante" (consumo della batteria) si aggiunge alla ricerca del cammino ottimo ed e' molto stringente, il drone infatti non può fare rifornimento in modo agevole. Bisognerà quindi studiare un algoritmo dinamico che gestisca le consegne su zona, a cluster, basandosi su cammino più breve e durata della batteria.

## Descrizione dei data-set per la valutazione

Il dataset può essere costruito da zero, consisterà principalmente in una serie di coordinate geografiche relative ai punti di ritiro e consegna. Avremo dati di controllo quali le informazioni sui droni inclusa la loro posizione (che per semplicità viene gestita come dato discreto, un grafo)

## Descrizione preliminare degli algoritmi coinvolti

L'applicazione farà uso di grafi per la gestione delle posizioni. Andrà studiato un algoritmo per la ricerca del cammino ottimale nei grafi che si adatti alla dinamicità del problema; si possono infatti ricevere ordini di ritiro in qualunque time slot della giornata e il percorso ottimale va quindi ricalcolato ogni 30 minuti. Il funzionamento dell'applicazione viene valutato con dataset di simulazione (ordini di consegna).

## Descrizione preliminare delle funzionalità previste per l'applicazione software

L'applicazione sarà principalmente una dashboard di monitoraggio in cui sarà possibile visualizzare il percorso effettuato dal drone, la % di batteria rimanente e il percorso programmato ancora da completare. Sarà possibile aggiungere nuovi droni, rimuoverli per manutenzione ed inserire la programmazione delle consegne.

# Descrizione del problema affrontato

Oggi le consegne automatizzate con droni sono sempre più una realtà.

Servizi più o meno sperimentali automatizzati, sia con robot che si muovono a terra che con droni in volo, stanno sorgendo un po' ovunque e in alcune città USA sono già una realtà concreta.

È in questo scenario che nasce la proposta presentata che punta a sviluppare un modello non deterministico del problema del commesso viaggiatore – TSP basata appunto sull'utilizzo di droni in volo sulle città USA.

Nel progetto presentato, si è optato insieme al docente per l'introduzione di fattori non deterministici nella modellazione del problema. Questo approccio consente l'esecuzione di numerose analisi differenziabili dando origine ad una rappresentazione del mondo abbastanza congruente con il reale.

L'esecuzione delle simulazioni non porterà quindi mai allo stesso risultato, se non quello ottimale di evasione di tutte le consegne.

Scopo della simulazione è quindi la determinazione di un cammino che tocchi tutti i punti dove effettuare una consegna, pescati in modo randomico a partire da una lista di punti presenti nel database. Il vincolo principale inserito del modello è la lunghezza del cammino che non potrà superare la distanza massima percorribile da un drone nell'effettuare le consegne includendo la partenza e il ritorno al magazzino. Un secondo punto chiave da considerare nella simulazione consiste nel fatto che il tempo per la carica delle batterie non è trascurabile. A differenza di un furgone, per il quale fare il pieno di carburante costa pochi minuti, la carica delle batterie costa ore che vengono sottratte al possibile tempo di consegna. Differentemente da quanto si potrebbe pensare, l'impatto di questo vincolo non è un problema ma aiuta anzi nell'ottimizzazione del percorso effettuato. Si verifica infatti che la carica delle batterie è concentrata in un intervallo più ampio a fronte però di un maggiore volume di consegne da distribuire al volo successivo. Il risultato è che viene "sprecato" meno tragitto per tornare al magazzino che si traduce in meno batteria da caricare nell'intero ciclo delle consegne con una conseguente ottimizzazione anche dei tempi.

## *Simulazione – parametri stocastici*

La generazione di un modello stocastico del problema si basa su tre punti principali:

- Una quantità sufficientemente ampia di dati di consegna
- Un fattore di campionamento sull'insieme dei dati
- Un fattore di variazione sul numero di consegne per timeslot

## *Fattore di campionamento*

Consente di estrarre casualmente una percentuale di dati relativi alle consegne. Questo fattore introduce una dinamicità nei dati proposti dall'applicativo ed aiuta a generare simulazioni abbastanza differenziate tra loro in termini di punti da raggiungere. Si ha inoltre la possibilità di variare la dimensione del campione utilizzato.

## *Fattore di variazione del pickup (e quindi disponibilità ai vari intervalli)*

Viene calcolato un valore medio del numero di consegne da effettuare ad ogni intervallo temporale sul quale è applicata una percentuale di variazione.

Questo parametro varia quindi il numero di spedizione nuove che arrivano al magazzino ad ogni singolo intervallo di tempo per essere assegnate ai droni.

Con l'uso di questi parametri, le simulazioni effettuate danno riscontro di una buona casualità nella generazione delle situazioni modellate.

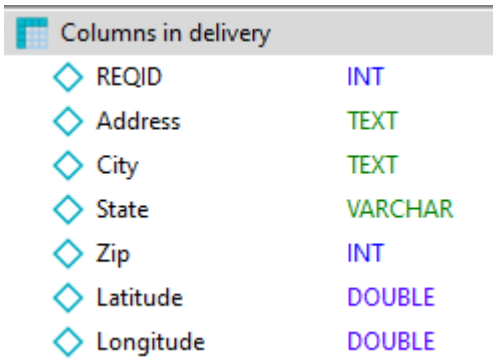
# Descrizione del data-set

Il dataset su cui si basa l'universo dei dati è disponibile **qui**

Si tratta di una lista di spedizioni effettuate nelle maggiori città USA da uno dei principali corrieri a livello mondiale. I dati non sono sottoposti a licenza, sono di pubblico dominio e ad uso pubblico.

Nello specifico i dati consistono in una tabella contenente le informazioni reperibili su una spedizione e chiaramente anonimizzate. Le parti fondamentali della tabella sono l'identificativo della spedizione e le coordinate geografiche, utilizzate per il calcolo della distanza tra i vari vertici del grafo usato per il calcolo del ciclo di Hamilton.

Per quanto il dataset sia semplice, per poter essere utilizzato sono stati necessari alcuni accorgimenti.




## *City Warehouse*

Il problema in oggetto richiede la presenza di un "magazzino di consegna e partenze dei droni". La scelta del punto in cui posizionare il magazzino da cui partiranno tutte le consegne è cruciale. Un magazzino posizionato distante rispetto al cluster delle consegne porterà inevitabilmente a problemi di efficienza dell'applicativo. È sempre importante tenere presente che finito il "carburante", il drone deve tornare indietro e non può ricaricarsi in un punto intermedio. Una soluzione immediata ed automatica è consistita nella generazione dei magazzini tramite una select con cui si è estratto il valor medio di latitudine e longitudine. Questo metodo ha consentito di scegliere un punto strategico centrato rispetto alle consegne da effettuare presenti nel database.

```
SELECT city, '000 Warehouse Street' AS Address, state, '999999', AVG(latitude),AVG(longitude) FROM delivery GROUP BY city;
```

## *Utilizzabilità parziale dei dati.*

Per quanto i dati siano relativi a città degli Stati uniti non tutte queste hanno una dimensione e una quantità di dati sulle spedizioni utilizzabile per lo scopo preposto.

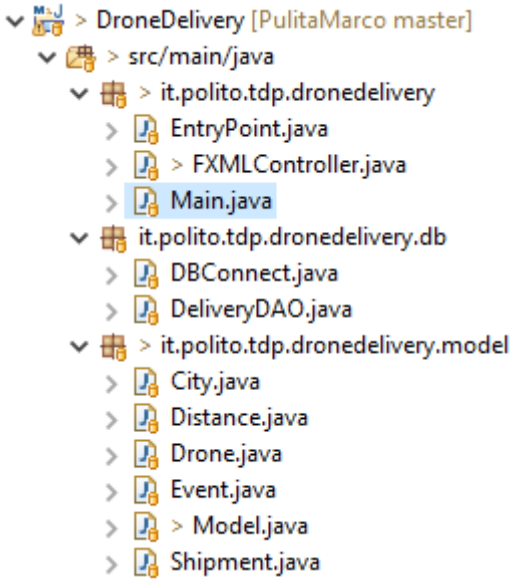
Le query implementate nell'applicativo faranno quindi una scrematura delle città disponibili per la simulazione fornendo le prime 20 in termini di quantità di dati elaborabili, tutte con un numero di spedizioni maggiore di 100.

Questo garantirà una buona flessibilità dell'algoritmo ma al contempo evitando l'inserimento di problemi banali.

# Struttura applicazione e algoritmi

L'applicazione ha un'organizzazione basata su Maven ed è corredata da due package principali: db e model. Utilizza, come visto a lezione, entrambi i pattern DAO e MVC.

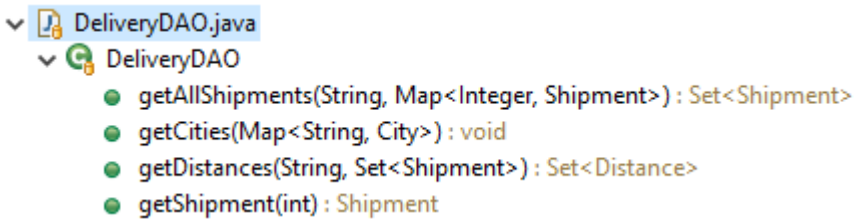
La classe model include metodi di gestione della simulazione ad eventi discreti nonché metodi per la gestione dei grafi necessari all'elaborazione del problema del commesso viaggiatore.




## *Package DB: DAO*

Come visto a lezione l'utilizzo del pattern DAO consente di astrarre alle classi che ne fanno uso, la logica di accesso al database ottimizzandone nel contempo le connessioni e le interrogazioni. L'applicazione implementa quindi le classi *DBConnect*, per la gestione e ottimizzazione delle connessioni al database e la classe DAO.

La classe *DeliveryDAO* espone quattro metodi:



Di maggiore nota nella classe è il metodo *getDistances,* delegato all'interrogazione del database per la generazione dei vertici delle distanze tra tutti i punti del grafo. I risultati vengono restituiti in un Set, contenitore di oggetti *Distance*. La peculiarità dell'oggetto *Distance* è che include la distanza tra agli oggetti *Shipment* sorgente e destinazione cioè il peso dell'arco del grafo tra due vertici. Con questo metodo si "scarica" l'applicazione dalla ricerca dei vertici abbassando la complessità dell'algoritmo di elaborazione.

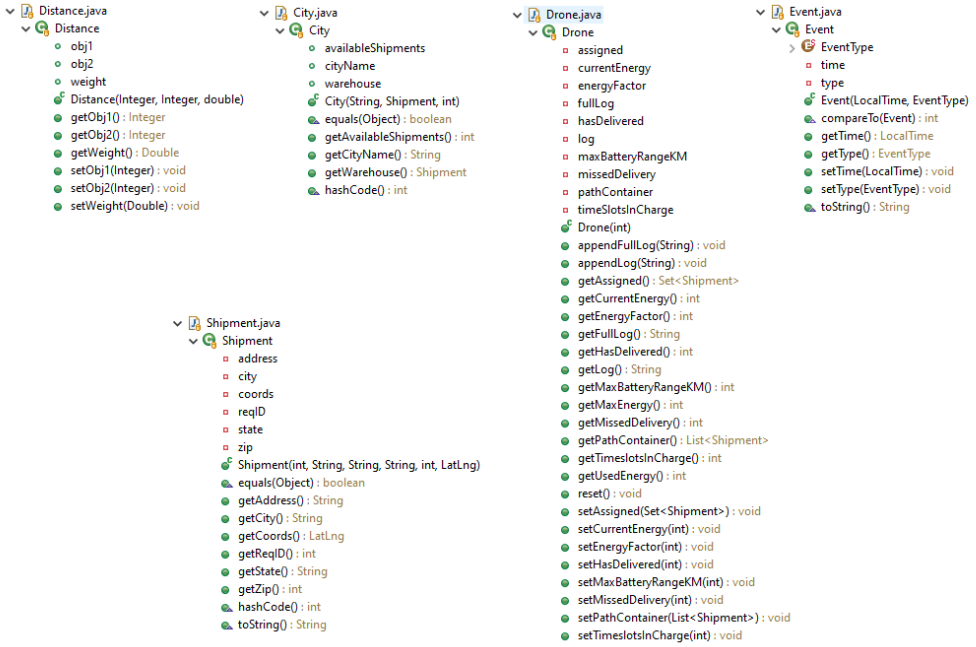
```
String sql = "SELECT d1.REQID P1, d2.REQID P2, d1.city P1city, d1.address P1Address, " +
                  "d2.address P2Address, d1.Latitude la1, d2.Latitude la2, d1.Longitude lo1, d2.Longitude lo2 " +
                  "FROM delivery d1, delivery d2 " +
                  "WHERE d1.city = ? AND d2.city = ? " +
                  "AND d1.REQID < d2.REQID AND d1.REQID IN ("+inStr+") AND d2.REQID IN  ("+inStr+")";

Set<Distance> distances = new HashSet<Distance>();

try {
    Connection conn = DBConnect.getConnection();
    PreparedStatement st = conn.prepareStatement(sql);
    st.setString(1, city);
    st.setString(2, city);
    ResultSet rs = st.executeQuery();
    while (rs.next()) {
        LatLng pc1 = new LatLng(rs.getDouble("la1"), rs.getDouble("lo1"));
        LatLng pc2 = new LatLng(rs.getDouble("la2"), rs.getDouble("lo2"));
        Double distP1P2 = LatLngTool.distance(pc1, pc2, LengthUnit.KILOMETER);
        Distance d = new Distance(rs.getInt("P1"), rs.getInt("P2"), distP1P2);
        distances.add(d);
    }
    st.close();
    conn.close();
}
```

## *Package Model*

Più complesso è invece il contenuto del package model. In questo Package troviamo oltre alla classe principale *model* classi accessorie, come la classe *Distance* utilizzate per modellare i vari oggetti della simulazione.



Di seguito una descrizione degli oggetti usati nella simulazione e modellati da classi del package

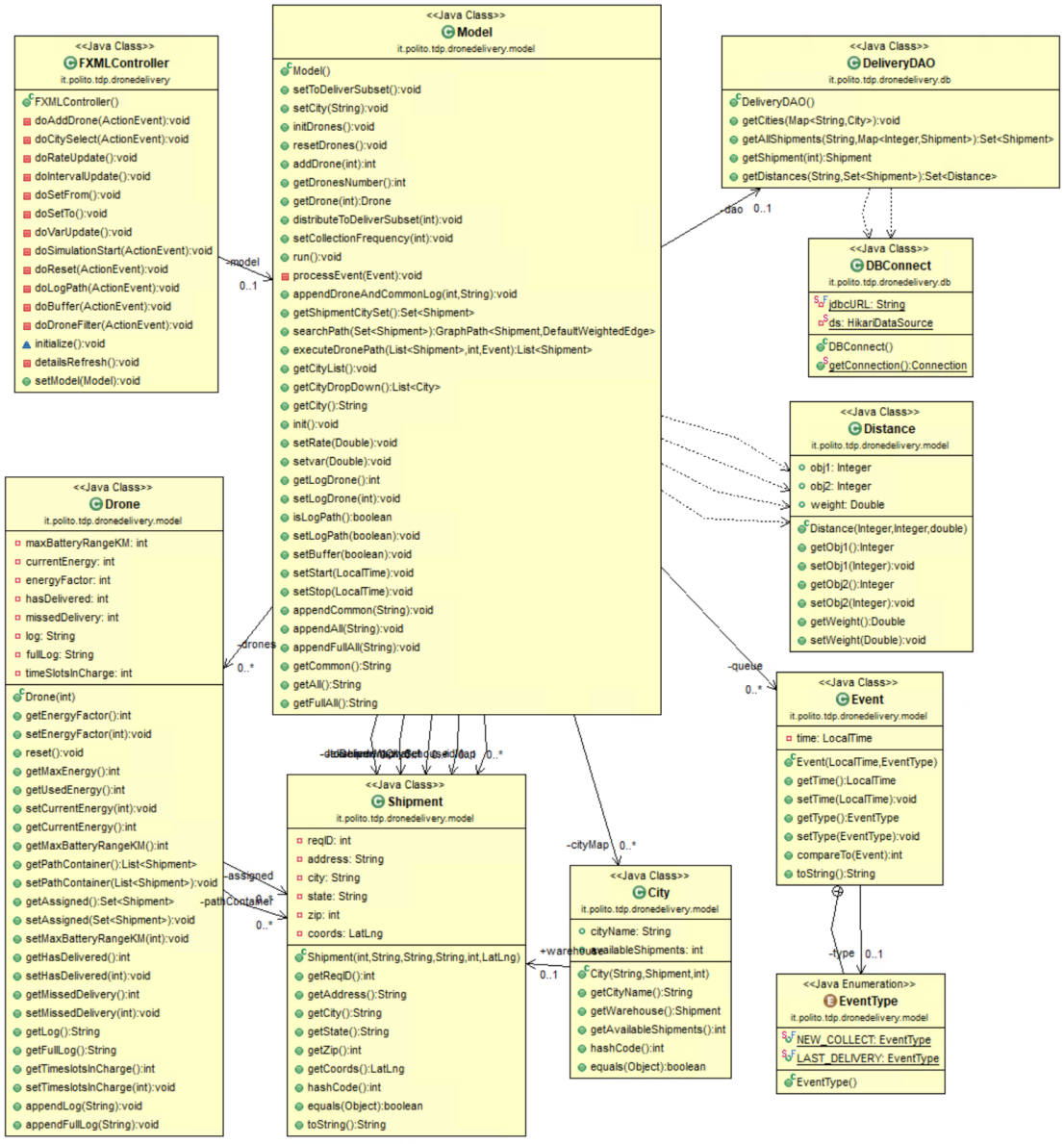
- *City*: rappresenta un oggetto città. Tramite *getAvailableShipements()* restituisce il numero di oggetti *Shipment* contenuti nel database.
- *Distance*: è l'oggetti che modella un vertice del grafo. Il peso è espresso proprio tramite la distanza tra i due oggetti di tipo *Shipment* come oggetto *Latlng*.
- *Drone*: è la classe principale del simulatore, modella un drone e le sue proprietà
  I parametri principali che modellano un drone sono:
  - *assigned*:La lista si spedizioni assegnate
  - *currentEnergy*: Il valore attuale di carica della batteria. L'algoritmo ricarica la batteria del singolo drone di una frazione ad ogni timeslot **tenendo fermo il drone**
  - *maxBatteryRange:* Il valore massimo di percorrenza per quel drone.
- *Event:* Modella gli eventi del mondo simulato (EVENT_COLLECT, LAST_DELIVERY)
- *Shipment:* È l'oggetto spedizione, il vertice del grafo.

## *Algoritmi utilizzati*

L'applicazione sviluppata modellando la simulazione tramite grafi. Si sono utilizzate le librerie java disponibili per la gestione delle strutture dati focalizzandosi maggiormente sullo sviluppo dell'algoritmo per la risoluzione del problema. L'algoritmo di risoluzione del TSP è il NearestNeighborHeuristicTSP con complessità in classe $O(V^2)$. È richiesto che il problema si espresso con un grafo completo (connesso con il numero massimo – tutti – di vertici), richiesta soddisfatta individuando tutte le distanze tra ogni singolo vertice.

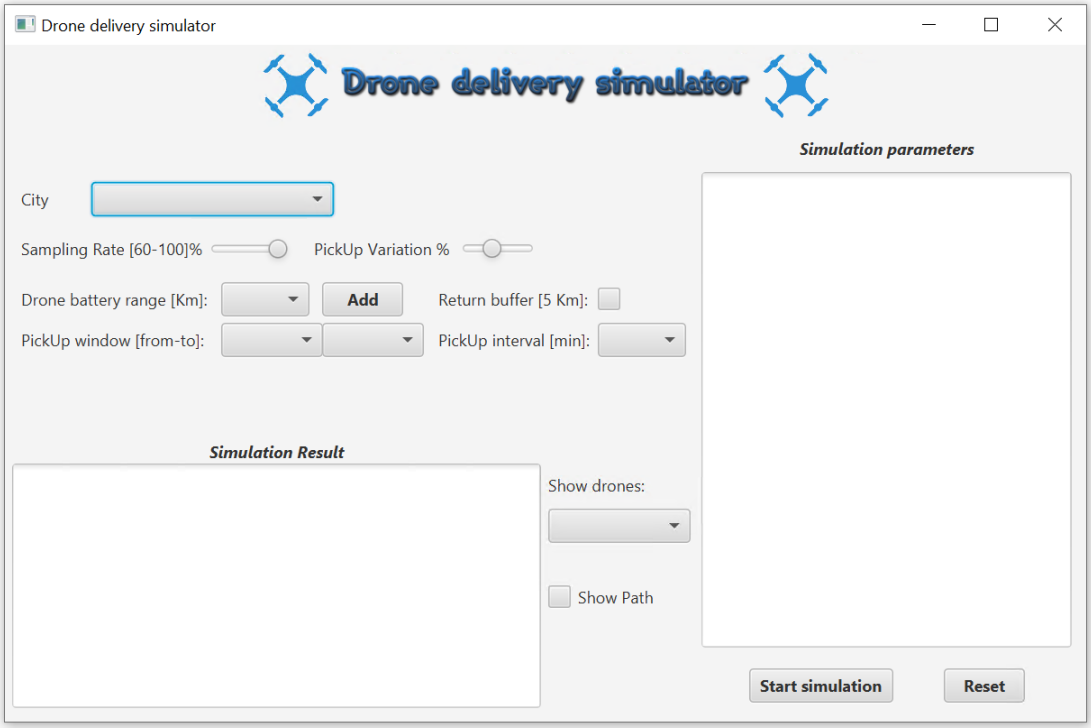
# Diagramma delle classi principali

Come descritto l'applicazione fa uso del pattern MVC nel quale la classe *model* gestisce la logica di elaborazione. Il *model* non fa diretto accesso al database ma delega la restituzione di oggetti "dato" al DAO, in questo caso la classe designata è la *deliveryDAO.* Il set di classi include infine le classi che modellano gli oggetti della simulazione.

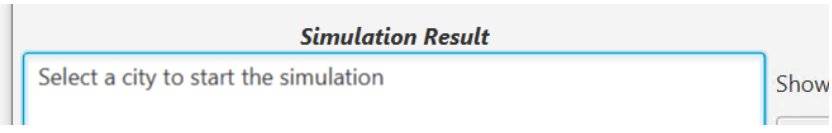


# Esecuzione e funzionamento dell'applicazione

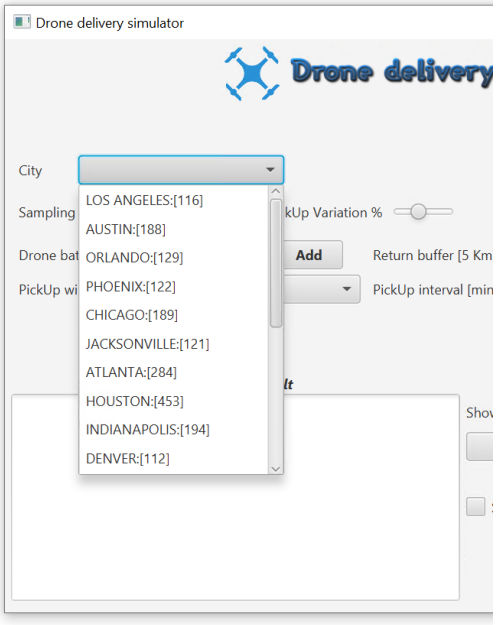
L'esecuzione dell'applicazione avviene in un'unica finestra nella quale si ha la possibilità di inserire i parametri di simulazione. Parametri inseriti e risultati sono riportati in textbox specifiche.




La logica dell'applicazione prevede alcuni valori di default per minimizzare i dettagli di simulazione da inserire. Obbligatori per la simulazione dell'esecuzione sono però le informazioni sulla città da modellare e l'inserimento di almeno un drone. Nel caso queste informazioni siano mancanti, si viene avvisati da un messaggio specifico nel momento in cui si preme il pulsante *Start simulation*.



La dropbox che consente di selezionare la città da subito un'idea del numero di spedizioni presenti nel database indicandole tra parentesi quadre.




La selezione di una città attiva il caricamento nella finestra dei parametri dei valori di default descritti precedentemente, in particolare:

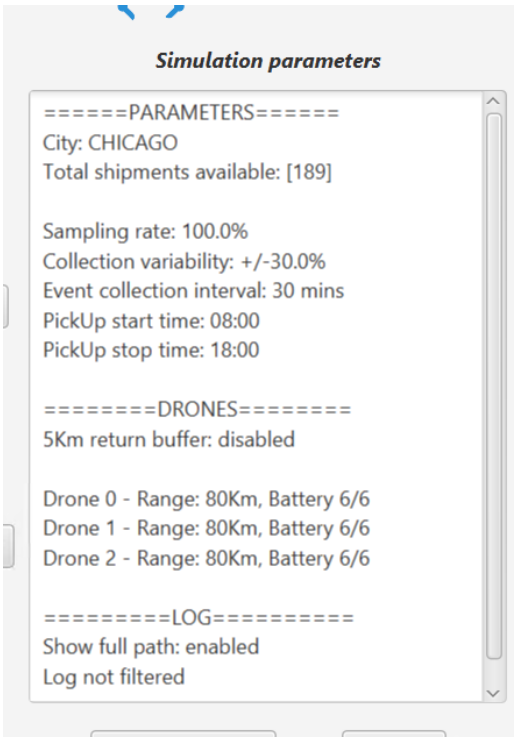
- *Sampling rate*: è il fattore di campionamento con cui viene estratta una certa porzione dei dati delle spedizioni. Il campionamento avviene con selezione randomica degli oggetti del database. Il valore minimo è 60% delle spedizioni, il massimo il 100% cioè tutti i dati disponibili. Da notare che l'uso di un set completo dei dati rimuove la dinamicità della simulazione a livello di variazione dei dati del campione usato.
- *PickUp Variation*: Consente di introdurre variabilità nel numero di spedizioni da gestire al singolo time slot. L'applicazione calcola un valore medio di spedizioni da inoltrare ad ogni istante di tempo dato da

  "Spedizioni da consegnare"/"eventi tempo in coda"

  Il parametro fa variare di +/- var % il numero di spedizioni gestite nello specifico slot di tempo.

- *Return buffer:* Aggiunge complessità al problema inserendo un vincolo che riduce di 5Km la distanza percorribile dal drone.
- *PickUp[from-to]:* Consente di variare la finestra temporale della simulazione
- *PickUp interval:* Aumenta la complessità distanziando maggiormente gli eventi EVENT_COLLECT. La conseguenza è un minor numero di time slots in cui gestire lo stesso numero di spedizioni.

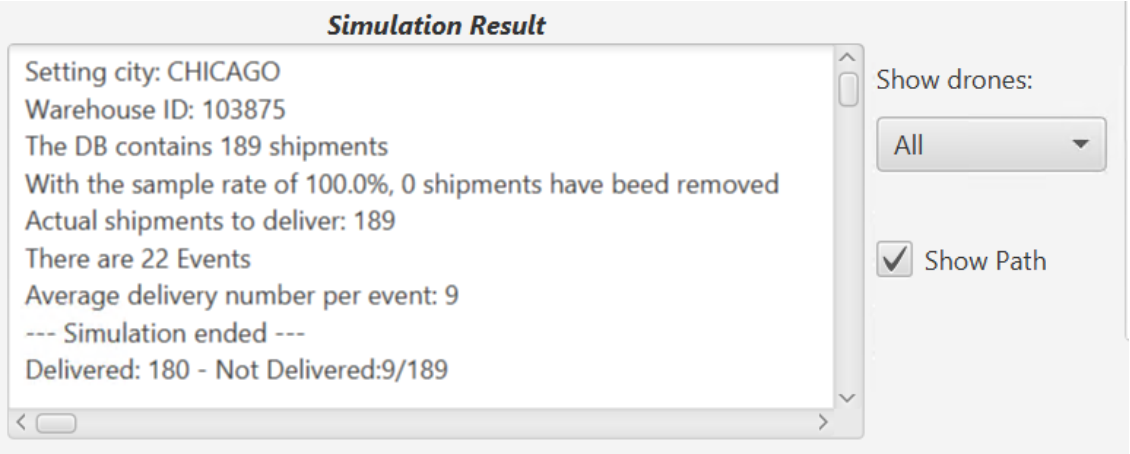
Tutti i parametri, inclusi i droni aggiunti alla simulazione per I quali è richiesta la selezione del battery *range*, sono riassunti nel riquadro *Simulation Parameters* ed aggiornati in modo automatico ad ogni modifica:




Vi sono inoltre due parametri che controllano il logging dell'applicazione:

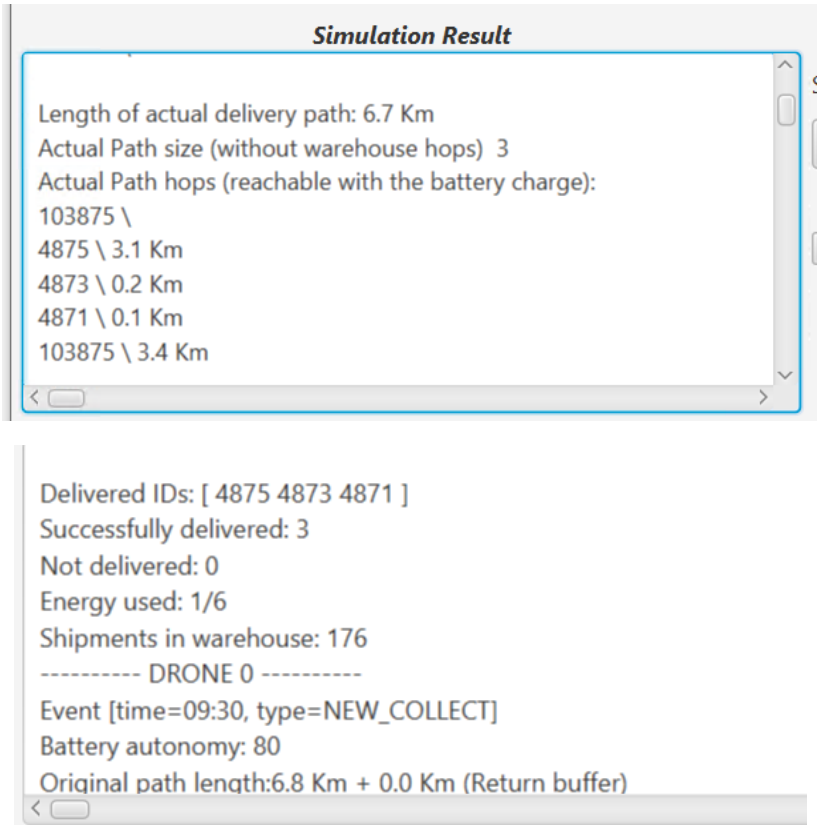
- *Show drones:* Consente di decidere se visualizzare il log del singolo drone o l'intera lista di eventi.
- *Show Path:* Abilita la visualizzazione in forma di testo dei path generati dall'algoritmo TSP.

Qualora l'inserimento dei droni non sia corretto è possibile reimpostare l'applicazione tramite il pulsante *Reset*



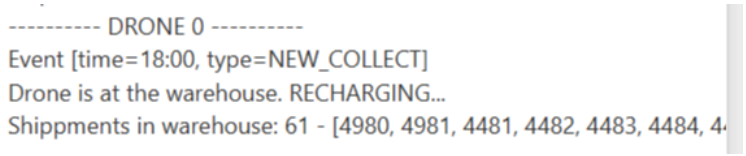
Nella prima parte del log di esecuzione vengono riportate le informazioni condensate della simulazione a livello generale. Si ha l'informazione sul numero di spedizioni consegnate e non consegnate sul totale.

Visualizzando la parte del testo sottostante questa sezione di hanno i dettagli di esecuzione per singolo drone. Sono riportati il path originario proposto e il path effettivo con i nodi del grafo toccati e la relativa distanza. Primo e ultimo nodo sono sempre il magazzino.




Infine, l'ultima parte del log relativo al singolo evento, riporta a livello di drone gli ID spedizione consegnati, l'energia usata e le spedizioni rimaste in magazzino da consegnare, chiaramente suddivise per intervalli di tempo.

I Droni in fase di ricarica non ricevono nuove spedizioni e si è avvisati dello stato.

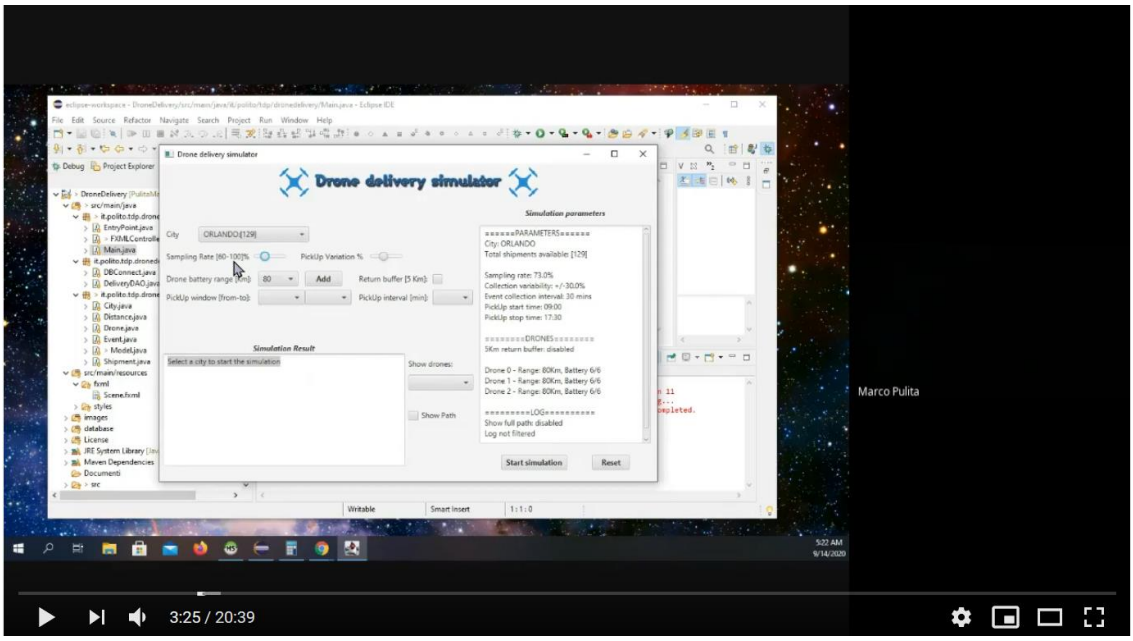


# Presentazione video dell’applicazione

Long (20 mins)
www.youtube.com/watch?v=NpR-x3bFU_o

Short (3 mins)

https://youtu.be/KFrcnZi_y8Q



# Risultati e conclusioni

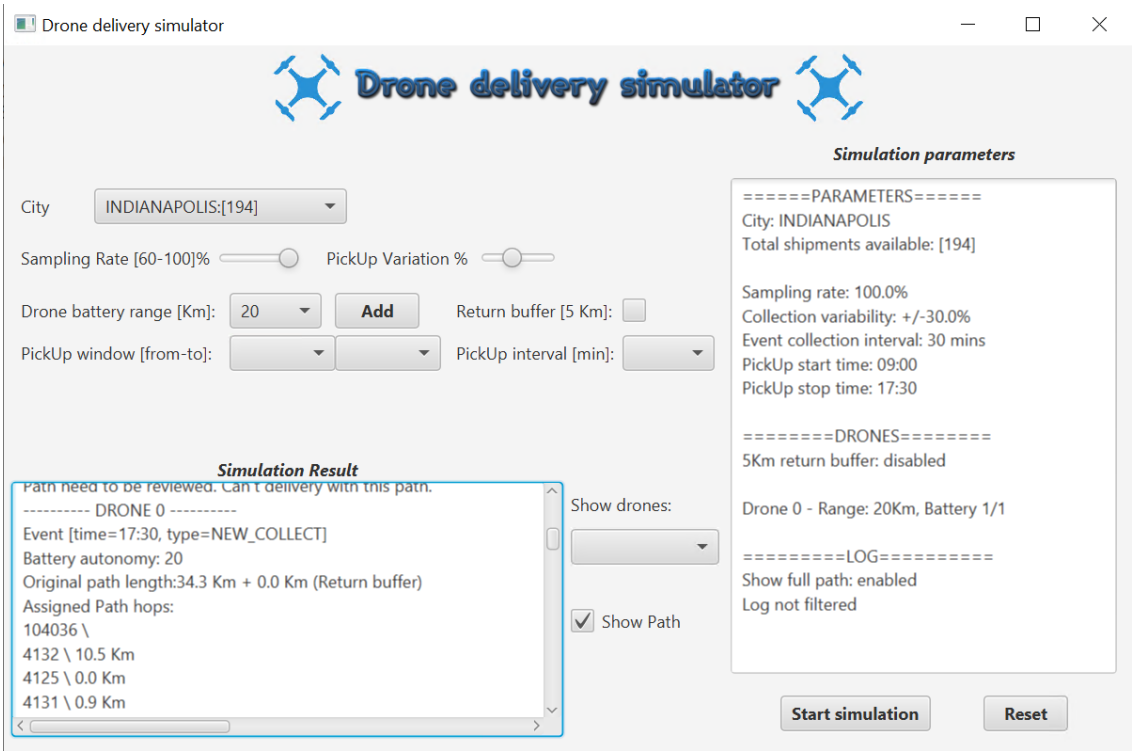
Scopo dell'applicativo è lo studio del modello di spedizioni mettendo in pratica l'algoritmo del cammino minimo. Le prestazioni in termini di tempo di esecuzione sono buone, il risultato della simulazione viene generato pressoché in tempo immediato.

Anche senza variazione dei parametri di input il modello genera risultati sempre diversi ma comunque nella media tra loro.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Orlando [129] 4 x 70Km | | | | | | | | | | AVG |
| 49 | 73 | 51 | 60 | 30 | 44 | 56 | 55 | 63 | 49 | 53 |
| | | | | | | | | | | |
| Orlando [129] 4 x 70 + 2 x 40 | | | | | | | | | | |
| 19 | 27 | 16 | 23 | 7 | 23 | 40 | 27 | 13 | 37 | 23 |
| | | | | | | | | | | |
| Orlando [129] 4 x 70 + 2 x 40 + 3 x 100 | | | | | | | | | | |
| 0 | 5 | 1 | 1 | 2 | 8 | 4 | 5 | 4 | 0 | 3 |

(Simulazione su Orlando con droni: 1) 70,70,70,70 2) 70,70,70,70,40,40 3) 70,70,70,70,40,40,100,100,100

L'algoritmo interno dimostra una discreta robustezza a meno di condizioni estreme che restituiscono risultati con limiti; è il caso della simulazione con un singolo drone con portata 20Km su Indianapolis. Per tutti i cicli viene generato un grafo con il primo nodo distante più della portata del drone causando la mancata esecuzione del path.




Con l'evento *LAST_DELIVERY* viene forzato l'inserimento di tutte le Spedizioni per correggere la varianza che generalmente ci si aspetta sia minima e questo porta a sbloccare il loop consegnando 23 spedizioni.

Riporto un risultato coerente ma che non avevo colto immediatamente nella definizione del modello. Mi aspettavo che fermando I droni in ricarica i numeri delle consegne portate a termine sarebbe drasticamente calato. Un drone con ampia autonomia rischia di dover stare fermo in carica anche un giorno. La mia valutazione era però sbagliata. Tenere fermo il drone consente alle consegne di "accumularsi" e porta ad avere una percentuale di meno strada percorsa per tornare se comparata con viaggi con una sola spedizione, saturando la capacità del drone in modo migliore. La minor strada si traduce in minor batteria da caricare e quindi maggior velocità in fase di carica(meno slot totali per la carica). Il numero di consegne per time slot deve però essere portato completamente a termine e questa condizione si ha prevalentemente con droni con una buona autonomia.

Future implementazioni avrebbero potuto tener conto di ulteriori parametri limitanti come ad esempio la generazione di eventi *WIND_EVENT* o condizioni più favorevoli come il posizionamento di più magazzini nella città.

Ad ogni modo, per vedere concretamente l'algoritmo in azione ed apprezzarne la bontà sarebbe stato cruciale l'uso di librerie grafiche per tracciare su mappa i grafi generati.