AUTONOMA BEZPILOTA DRONU LIDOJUMA PLĀNOŠANA, IZMANTOJOT MODIFICĒTU ĪSĀKĀ CEĻA ALGORITMU AR LAIKA IEROBEŽOJUMIEM

AUTONOMOUS UNMANNED DRONES FLIGHT PLANNING, USING A MODIFIED SHORTEST PATH ALGORITHM WITH A LIMITED TIME FRAME

Authors: **Marks SONDORS**, e-mail: ms20094@edu.rta.lv
Scientific supervisors: **Mg.sc.comp. Ilmārs APEINĀNS**, e-mail: ilmars.apeinans@rta.lv
doc. Dr.sc.ing. Sergejs KODORS, e-mail: sergejs.kodors@rta.lv
Rēzeknes Tehnoloģiju Akadēmija
Atbrīvošanas aleja 115, Rēzekne, Latvija

Abstract. *The aim of this work is to develop an algorithm to find the shortest path for drone flight planning with a limited time frame. Author used the local search shortest path algorithm to find the most efficient algorithm to use for further modification to apply to a drones flight calculation. The algorithm was modified to use the distance between points as a unit of time to limit the flight path length depending on the drone's maximum flight time. As a result of the work, an algorithm was created which, upon receiving an array of points, finds the shortest distance between the points, but when it reaches the limit of the flight duration, it returns to the drone station to charge, and resumes flight once it's done charging.*

Keywords: *Algorithm, Drones, Python, Traveling Salesman's Problem, Local search.*

Ievads

Ceļojošā pārdevēja problēma[1] ir algoritmiska problēma, kuras uzdevums ir atrast īsāko maršrutu starp punktiem un vietām, kas jāapmeklē. Ņemot vērā pilsētu kopu un attālumu starp katru pilsētu pāri, problēma ir atrast īsāko iespējamo maršrutu, kas apmeklē katru pilsētu tieši vienu reizi un atgriežas sākuma punktā. Šajā raksta tiks pielietots šīs algoritmiskās problēmas risinājums, lai izveidotu īsākā ceļa algoritmu, kuram tiks padoti noteikti punkti un tāpat arī drona sākuma atrašanās vieta(drona stacija), kas atradīs lidojuma maršrutu, kuru var izlidot drons pirms viņš izlādējas un ja maršrutu nevar izlidot vienā lidojumā drons atgriežas atpakaļ, lai varētu uzlādēties un tad turpināt savu lidojumu.

Šī pētījuma **mērķis** ir izstrādāt algoritmu īsākā ceļa noteikšanai, kas būs spējīgs noteikt optimālāko lidojuma maršrutu, lai izlidotu doto punktu masīva koordinātes ar laika ierobežojumiem.

Materiāli un metodes

Darbā tika pielietots Python uz Jupyter Notebook [2] vidē.

Tehnoloģijas

Lai varētu izveidoto algoritmu pielietot ir nepieciešamas tādas tehnoloģijas:

- *Python 3.9* – programmēšanas valoda;
- *Anaconda* [3] – *Python* programmēšanas valodas un saistīto rīku atklātā pirmkoda izplatīšana zinātniskajai skaitļošanai un datu zinātnei;
- *Numpy* - *Python* bibliotēka zinātniskai skaitļošanai un datu analīzei, kas nodrošina atbalstu lieliem, daudzdimensiju masīviem un matricām, kā arī lielu matemātisko funkciju kolekciju, lai darbotos ar šiem masīviem;
- *Pandas* - atvērtā koda *Python* bibliotēka, ko izmanto datu analīzei un manipulācijām. Tas nodrošina datu struktūras lielu un sarežģītu datu kopu efektīvai glabāšanai un apstrādei, kā arī rīkus datu tīrīšanai, transformācijai, apkopošanai un analīzei.

https://doi.org/10.17770/het2023.27.7375

- *Matplotlib* [4] - *Python* bibliotēka, kuru pielieto lai izveidotu augstas kvalitātes datu vizualizācijas rīks priekš diagrammām, histogrammu u.t.t.
- *Haversine* [5] - *Python* bibliotēka, lai noteiktu distanci starp diviem punktiem izmantojot platuma un garuma grādus.
- *Python-tsp* [6] - *Python* bibliotēka ceļojošā pārdevēja problēmas (TSP) risināšanai;
- *Shapely* - *Python* bibliotēka, ko izmanto, lai apstrādātu un analizētu ģeometriskus objektus, piemēram, punktus, līnijas un daudzstūrus.

Eksperimenta process

Pēc izstrādes vides sagatavošanas, bija nepieciešams sagatavot masīvu ar punktiem un drona sākuma atrašanās vietu. Masīvs ar punktiem un drona sākuma atrāšanās vieta tika sagatavot pielietojot projekta ietvaros izveidoto tīmekļa vietni *droni.rta.lv* [7]. dotās koordinātes tika nosūtītas caur *API*, kurš nosūta datus par punktiem un drona stacijas atrāšanās vietu (skatīt 2.1. att.).

2.1. attēls. Punktu masīva vizuālā reprezentācija vietne *droni.rta.lv*.

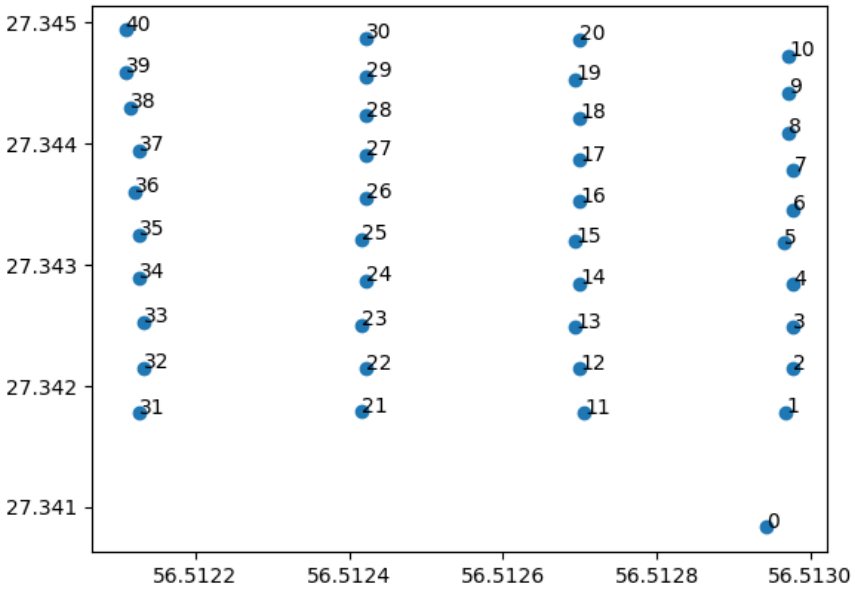
Algoritms pielieto lokālās meklēšanas heiristisko metodi [8], lai noteiktu īsāko ceļu. Tāpat arī lai noteiktu distanci starp punktiem tiek pielietota *Haversine* formula [9] izmantojot *Haversine Python* bibliotēku, kas nosaka un vēlāk ievieto noteiktās distances masīva, kuras tiks pielietotas paša algoritma. Pēc dota masīva un drona sākumpunkta iegūšanas, tika palaists algoritms, kurš izveido lidojuma plānus, pielietojot īsākā ceļa algoritma principus (skatīt 2.2. att.).

```
flight_path_points = []
new_distance = 0
for i in range(len(permutation)-1):
# check if distance to and back from point is not over the flight limit
    if((new_graph[permutation[i]][permutation[i+1]]+new_graph[permutation[i+1]][permutation[0]]) <
(flight_distance_limit - critical_flight_limit)):
        new_distance += new_graph[permutation[i]][permutation[i+1]] *1000
# check if curent segment path plus distance home is within flight distance of drone
        if(((new_distance) + (new_graph[permutation[i+1]][permutation[0]]*1000))
            flight_path_points.append(permutation[i])
        else:
            new_distance = 0
            flight_path_points.append(0)
            new_distance += new_graph[permutation[i]][permutation[i+1]] *1000
            flight_path_points.append(permutation[i])
flight_path_points.append(0)
```

2.2. attēls. Algoritma daļa, kur izveido lidojuma plānus ar laika ierobežoju.

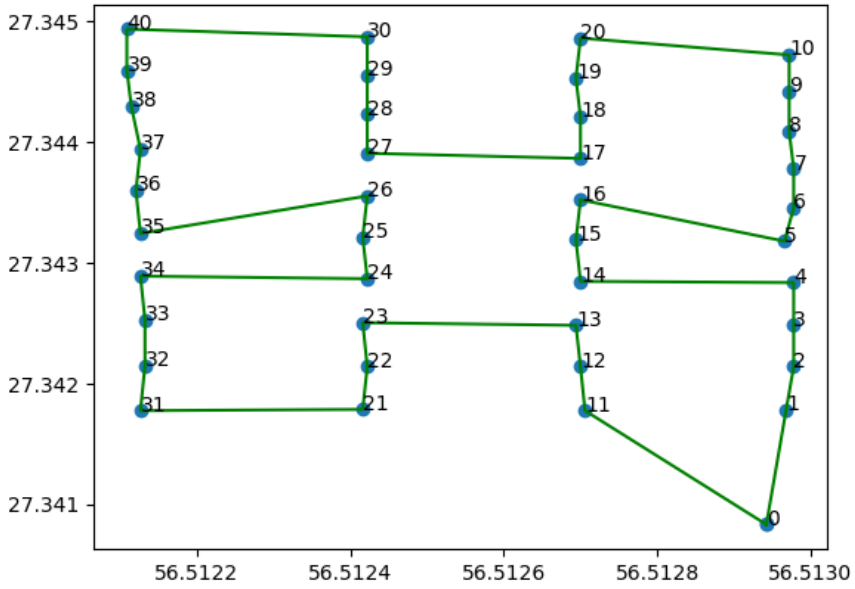
Rezultāti

Kad algoritms tika palaists, tika iegūtas vairākas vizualizācijas un plānu diagrammas. Masīva punktu vizualizāciju var apskatīt 3.1. attēlā. Vizualizācija parāda visu punktu atrašanas vietu.



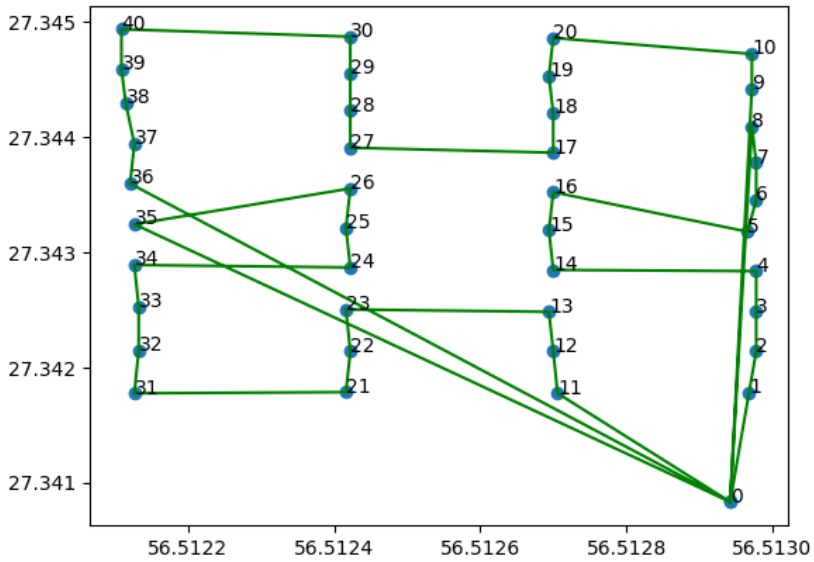
3.1. attēls masīva punkti

Plāna vispārējo diagrammu, jeb labāko variantu, kad drons var izlidot visus punktus un netiek pielietoti laika ierobežojumi(skatīt 3.2. attēls). Šo plānu būs arī jāpielieto, lai noteiktu plānu ar laika ierobežojumiem.



3.2. attēls. Plāna vispārēja diagramma

Plāna diagramma ņemot vēra drona lidojuma laiku, var redzēt 3.3. attēlā. Šis plāns ir izveidots, lai drons varētu atgriezties atpakaļ uzlādei un turpināt veikt lidojumu no pēdējā punkta, kuru izlidoja.



3.3. attēls. Plāna diagramma, ņemot vēra drona lidojuma laiku

Secinājumi

Šajā pētījumā, tika izveidots algoritms, kurš saņem masīvu ar punktiem un tad izveido optimālāko lidojuma plānu ņemot vērā laika ierobežojumus, līdz ar to seko ka, pētījuma mērķis tika sasniegts. Realizētais algoritms ir pielietojams, bet algoritmu var uzlabot pievienojot papildus nosacījumus, piemēram kā, aizliegtās robežas, kur drons nedrīkstētu lidot, tā pat arī būtu arī jāpievieno papildus topogrāfiskie dati, kā altitūda, lai lidojums notiktu noteiktos augstumos. Tomēr, izmantojot realizēto algoritmu, lai autonomi izlidotu noteiktus punktus var, bet kā iepriekš minēts noteiktus nosacījumus un uzlabojumus vajadzētu implementēt, lai lidojuma precizitāte un sekmība būtu augstāka.

Acknowledgement

Funding institution: Latvian Council of Science
Funding number: lzp-2021/1-0134
Acronym: FLPP-2021-1
Funding text: This research is funded by the Latvian Council of Science, project "Development of a decision-making system for smart horticulture using autonomous drones", project No. lzp-2021/1-0134

Summary

In this study, an algorithm was created that receives an array of points and then creates the optimal flight plan taking into account time constraints, therefore, the research objective was achieved. The implemented algorithm is applicable, but it can be improved by adding additional conditions, such as restricted boundaries where the drone should not fly, and also additional topographic data such as altitude should be included to ensure flight at specific heights. However, using the implemented algorithm, it is possible to autonomously fly to specified points, but as previously mentioned, certain conditions and improvements should be implemented to increase flight precision and success.

Literatūra

[1] *Travelling Salesman Problem using Dynamic Programming* [tiešsaiste], [atsauce uz 05.03.2023]. Pieejams: https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/
[2] *The Jupyter Notebook* [tiešsaiste], [atsauce uz 05.03.2023]. Pieejams: https://jupyter-notebook.readthedocs.io/en/stable/notebook.html

[3] *What is Anaconda for Python & Why Should You Learn it?* [tiešsaiste], [atsauce uz 05.03.2023]. Pieejams: https://blog.hubspot.com/website/anaconda-python
[4] *Matplotlib: Visualization with Python* [tiešsaiste], [atsauce uz 05.03.2023]. Pieejams: https://matplotlib.org/
[5] *Haversine - Python* bibliotēka [tiešsaiste], [atsauce uz 05.03.2023]. Pieejams: https://pypi.org/project/haversine/
[6] *python-tsp - Python* bibliotēka [tiešsaiste], [atsauce uz 05.03.2023]. Pieejams: https://pypi.org/project/python-tsp/
[7] *droni.rta.lv* vietne [tiešsaiste], [atsauce uz 05.03.2023]. Pieejams: https://droni.rta.lv/
[8] *Local Search Algorithm* [tiešsaiste], [atsauce uz 06.03.2023]. Pieejams: https://www.sciencedirect.com/topics/computer-science/local-search-algorithm
[9] Distance on a sphere: The Haversine Formula [tiešsaiste], [atsauce uz 05.03.2023]. Pieejams: https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128