

Drone flyrute planlægning og skedulering

Drone flight planning and scheduling

Lea Fog-Fredsgaard - lefog14 - 281177
Danny Rene Jensen - danje14 - 010187

Vejleder: Peter Schneider-Kamp

Syddansk Universitet, Odense
Institut for Matematik og Datalogi

1 januar 2019

Contents

1	Resume på engelsk	4
2	Introduktion	5
3	Specifikation	5
4	Systemdesign	6
5	Routing	7
5.1	OpenStreetMap - OSM	7
5.1.1	Points	7
5.1.2	Design	7
5.1.3	Implementation	9
5.1.4	Test	9
5.2	Open Source Routing Machine - OSRM	10
5.2.1	OSRM opbygning	11
5.2.2	OSRM profiler	11
5.2.3	Setup funktion	12
5.2.4	Implementation	12
5.2.5	Test	14
5.2.6	Fejl og mangler	18
6	3D flyruter	21
6.1	Design	21
6.1.1	Beregning af center punkter mellem 2 master	21
6.1.2	Beregning af center punkter for kabelbuen	23
6.1.3	Beregning af kabelmonteringpunkter	25
6.2	Implementation	30
6.2.1	Center punkter mellem 2 master	32
6.2.2	Centerpunkter for kabelbuen	34
6.2.3	Kabelmonteringpunkter	35
6.3	Test	39
6.3.1	Center punkter mellem 2 master	39
6.3.2	Center punkter for kabelbuen	41
6.3.3	Kabelmonteringspunkter	43
7	Visualisering	46

7.1	Design	46
7.1.1	AirSim	46
7.1.2	Blender	46
7.2	Implementation	47
7.3	Test	50
8	System Integration	51
9	Fremtidige udvidelser	52
10	Konklusion	53
11	Kilder	54
12	Bilag	55
13	Appendix	57

1 Resume på engelsk

In this project a number of programs are being prepared, for the purpose of flying a drone near high voltage cables to scan for errors. To get the right locations for the drone, OSRM(Open Source Routing Machine) is used with a map from OSM(Open Street Map). When the right coordinates are found, another program is then tasked to find the specific coordinates of each cable along the route.

These programs also calculate the z value for the drones flightpath. These programs calculate the angle of the towers, the offset of the cable from the center of the towers and also the z value of each cable.

This means that the program needs a lot of data to be handed to it. It needs at least the x and y coordinates of 3 towers and the z value of the last 2 towers. The x and y coordinates is found using OSRM.

When all the calculations are over and done with, the flight of a drone is simulated using AirSim. AirSim is given the x, y, z coordinates from the other programs and then needs to fly near high voltage power lines, which was created in a 3D modeling program called Blender and put into AirSim, that runs in Unreal Engine.

There have been a lot of problems in this project, like programs not installing correct and computers malfunctioning, which has resulted in a lot of wasted time trying to fix.

2 Introduktion

I dette bachelorprojekt arbejdes med droners anvendelse til inspektion af højspændingskabler i højspændingsnettet.

Via kort over højspændingsnettet i Danmark, arbejdes med planlægning og udvikling af droners flyruter i 3D, samt visualisering af dette.

I Danmark er der forskellig regelsæt for flyvning med droner. Et for private personer og et for professionelle/virksomheder. Flyvning i bymæssigt område kan kun foretages af professionelle/virksomheder.¹.

Regler for professionel flyvning med droner kan læses i ”Bekendtgørelse om flyvning med droner i bymæssigt område” på Trafik-, Bygge og Boligstyrrelsens hjemmeside på retsinformation.dk². Regler og bekendtgørelser vil ikke være en del af dette projekt.

Reglerne er restriktive omkring flyvning af droner i nærheden af boligbebyggelse. Men da der er meget lidt boligbebyggelse i nærheden af højspændingskabler, skulle lovgivningen give mulighed for, at bruge droner til at inspicere højspændingskabler.

3 Specifikation

Med udgangspunkt i OpenStreetMap (OSM) skal der udvikles en metode, der via Open Source Routing Machine (OSRM), henter information i OSM, om højspændingsnettet i Danmark. Denne information skal danne grundlag for udvikling af et program, til beregning og modellering af 3D flyruter for droner. En visualisering af dronens inspektion af el-kabler skal udvikles i AirSim.

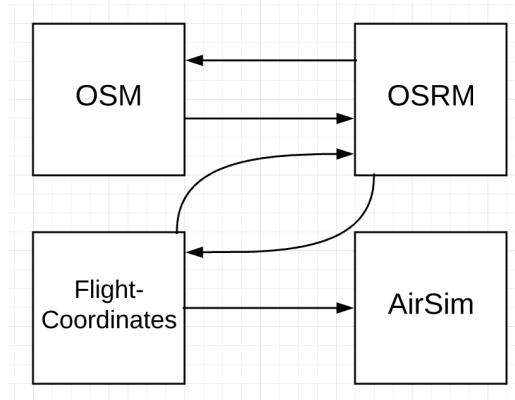
¹<https://www.trafikstyrelsen.dk/da/droneregler>

²<https://www.retsinformation.dk/Forms/R0710.aspx?id=194964>

4 Systemdesign

Projektets formål er at skrive et program der generere 3D koordinater, droner kan anvende, til flyrute langs el-kabler for at kunne inspicere disse.

OpenStreetMap (OSM) stiller detaljeret kortdata til rådighed. Det er blandt andet muligt at finde længde- og breddegrads koordinater for el-masters placering. Open Source Routing Machine, OSRM, er et ruteplanlægnings program som bruger OSM-kortet til at rute på.



Figur 1

Ved hjælp af et program, her kaldet *FlightCoordinates*, skal hentes information omkring højspændingsnettet og planlægges ruter til inspektion af el-kabler. Visualiseringen af dronens inspektion af el-kabler vil udføres i AirSim. AirSim er et program til f.eks at simulere biler eller droners bevægelse gennem et animeret 3D område.

Det er udviklingen af et program, der muliggøre dette, der arbejdes med i dette projekt.

5 Routing

5.1 OpenStreetMap - OSM

OpenStreetMap, OSM, er et detaljeret kort over det meste af verden. OpenStreetMap er et open source produkt. Kortet er genereret af et fællesskab af frivillige kortlæggerer, der via luftfotografier, GPS-enheder og lokalkendskab, bidrager med data til OSM kortet. Data kan være veje, stier, fodboldbaner, jernbanestationer, højspændingmaster og andre “points of interest” (Points).

OpenStreetMap indholder større mængde data, end kort som Google Maps. I OpenStreetMap indlæses al data, og ikke kun det der er relevant for en rute mellem to steder.

5.1.1 Points

De frivillige der bidrager til OpenStreetMap, skal tilføje data til kortet. Den data der tilføjes kortet, tilføjes “tags”, der definere hvilken type det er. For at det bliver brugbart for alle, bliver det nød til at være standardiseret. Guidelines til dette kan findes online.³

5.1.2 Design

Det er ikke muligt at tilgå enkelte data af OSM kortet. For at finde information om specifik data, som for eksempel en højspændingsmasts *id*, bliver hjemmesiden *overpass-turbo.eu*⁴ brugt. Det er muligt at tilgå den relevante data via denne side.

json kommandoer, bruges for at hente data fra kortet. På hjemmesiden trykkes på ”Geodata” på højre side af skærmen. På venstre siden erstattes ”node ... out;” med teksten fra Listing 1, derefter søges med ”Søg”. Output bliver som i Figur 2.

Listing 1: json til Node 341686702

³<https://taginfo.openstreetmap.org/>

⁴<https://overpass-turbo.eu>

```

1 [out:json];
2 (
3   node(341686702);
4 );
5 (_;>););
6 out;

```

I Figur 2 er det muligt at se output, svarende til en *node*. Den information der er interessant for dette projekt er *lat* , breddegrad, og *lon*, længdegrad, på højspændingsmasten, med *id*: 341686702.

Processen med at åbne hjemmeside og manuelt indtaste data, samt kopiere svaret til videre brug, er besværlig og tidskrævende. Denne proces skal automatiseres. *overpass – turbo.eu* har en *API* som skrives i python.

```

{
  "type": "node",
  "id": 341686702,
  "lat": 55.3462730,
  "lon": 10.3367190,
  "tags": {
    "addr:city": "Odense SV",
    "addr:country": "DK",
    "addr:housenumber": "105",
    "addr:municipality": "Odense",
    "addr:place": "Dyrup",
    "addr:postcode": "5250",
    "addr:street": "Vibeægvænget",
    "osak:identifier": "0a3f5089-e531-32b8-e044-0003ba298018",
    "source": "AWS Web API"
  }
}

```

Figur 2: json output

For hver el-mast, *node*, skal bruges informationerne *id*, længde- og breddegrad.

Ved hjælp af den *API* *overpass – turbo.eu* stiller til rådighed, er det muligt at lave et *script*, der henter det vigtige data fra *noderne*. *Scriptet* tager *id* fra *noderne* og output vil være længdegrad, breddegrad og *id* for hver *node*.

5.1.3 Implementation

For at kunne hente information om noderne, skal der laves en *json* forespørgsel (*query*). Denne forespørgsel er en database forespørgsel.

Forespørgslen skrives i en linje, opdelt af semikolon, der indikere linjeskift, som i java.

I linje 7 i Listing 2 ses et eksempel på en *json* forespørgsel, med 1 forespørgsel for hver node. Det ville være muligt at optimeres lidt, da alle noder kan lægges sammen i en forespørgsel. Programmet tager under et sekund, så det ændres ikke.

Alle forespørgslerne sendes på en gang, i linje 9 og gemmes i en liste: *result*. *result* består af af elementer, hvilket kan ses i Listing 2. De relevante informationer i elementerne er *id*, *lon* og *lat*. I linjerne 12 og 13 i Listing 2 printes de relevante informationer ud, til videre brug.

Listing 2: getNode.py

```
1 import overpy
2 import sys
3 api = overpy.Overpass()
4
5 stringNodes = ""
6 for no in range(1, len(sys.argv)):
7     stringNodes = stringNodes + "node(" + str(sys.argv[
        no]) + ");out;"
8
9 result = api.query(stringNodes)
10
11 for no in range(1, len(sys.argv)):
12     print(str(sys.argv[no]))
13     print(result.nodes[no-1].lon, result.nodes[no-1].lat
        )
```

5.1.4 Test

Listing 3: Node 341686702

```
1 <node id="341686702" visible="true" version="7"
  changeset="59472909" timestamp="2018-06-01T21:48:06Z
```

```

    " user="autoAWS" uid="8209541" lat="55.3462730" lon=
    "10.3367190">
2  <tag k="addr:city" v="Odense SV"/>
3  <tag k="addr:country" v="DK"/>
4  <tag k="addr:housenumber" v="105"/>
5  <tag k="addr:municipality" v="Odense"/>
6  <tag k="addr:place" v="Dyrup"/>
7  <tag k="addr:postcode" v="5250"/>
8  <tag k="addr:street" v="Vibeægvænget"/>
9  <tag k="osak:identifier" v="0a3f5089-e531-32b8-e044
   -0003ba298018"/>
10 <tag k="source" v="AWS Web API"/>
11 </node>

```

Listing 4: Node 831295159

```

1  <node id="831295159" visible="true" version="2"
      changeset="14490570" timestamp="2013-01-01T18:53:28Z
      " user="nimapper" uid="36080" lat="55.3480100" lon="
      10.4158238">
2  <tag k="design" v="y-frame"/>
3  <tag k="power" v="tower"/>
4  </node>

```

I Listing 3 ses et tilfældigt punkt (*point*) på en vej. Hvert *point*, eller *node* har *tags*, der indeholder både generel information om området, samt information om netop dette specifikke punkt. Det er muligt at se, at punktet befinner sig i byen, Odense SV, i bydelen Dyrup, vejnavne og nummer. Det er også muligt at se punktets longitude og latitude.

En el-masts *node* indeholder ikke så meget information. Som det ses af Listing 4 har el-mast *noden* ikke så mange *tags*. Den indeholder information om at det er en el-mast, *power tower*, longitude og latitude, samt *nodens id*. Det er muligt, at bruge disse *node id*'er til, at finde en specifik *node*.

5.2 Open Source Routing Machine - OSRM

OSM er et kort og kan kun vise data. Ruteplanlægningen skal derfor ske via et andet program. I dette projekt bruges Open Source Routing Machine,

OSRM. OSRM er et open source program, derfor er det laveligt at ændre i programmet, så det passer til andre behov, end det var designet til.

Det er muligt at hente kort for et specifik område, hvis ruteplanlægningen kun skal udføres i et begrænset område. For eksempel er det muligt at downloade⁵ et kort over Danmark, derved nedsættes beregningstiden væsentligt.

5.2.1 OSRM opbygning

For at starte med at bruge OSRM, skal hentes OSRM backend fra github hjemmesiden⁶. Inde i denne mappe, ligger en mappe som hedder profiles. I profiles ligger alle profiler OSRM bruger.

For at starte OSRM på en Ubuntu maskine, skal Docker hentes og installeres⁷. Yderligere information, om hvordan OSRM opstartes, findes i Test afsnittet.

5.2.2 OSRM profiler

OSRM bruger profiler alt efter om det er cykel, bil eller fodgænger, der skal ruteplanlægges for. Disse profiler indeholder data om hvilke “veje” OSRM må planlægge ruten på og hvilke der ikke må ruteplanlægges på.

En bil må for eksempel ikke rute på en cykelsti, eller en fodgænger må ikke have mulighed for at gå på en motorvej. Derfor skal disse muligheder udelukkes. En OSRM profil bliver brugt til at hive alle de noder ud, som er mulige at ruteplanlægge på, med den profil man vælger.

Listing 5: Highway

```
1 highway = way:get_value_by_key('highway')
```

I profilen skal vejene som skal bruges, defineres i en “local data” liste. Vejene bliver brugt til at udtrække, de noder der opfylder, de stillede krav. I Listing 5 ses et eksempel på, at det er muligt at køre på alle “highways”. Det vil sige, at alle de veje der har *tag’et* “highway”, vil blive “hevet” ud af kortet og gemt i en anden fil. Denne fil vil blive brugt til at ruteplanlægge på. Det er

⁵<https://download.geofabrik.de/>

⁶<https://github.com/Project-OSRM/osrm-backend>

⁷<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

muligt at bruge ORSM til ruteplanlægning på andet end veje, i den normale fortolkning. Hvis noder er forbundet vil de opfattes som veje i OSRM. Derfor vil det være muligt at ruteplanlægge på højspændingsnettet i OSRM.

5.2.3 Setup funktion

I setup funktionen, inde i profilen, ligger der en masse data, som bliver brugt til selve rutningen. I starten skal specificeres hvad der vægtes højest: “duration”, den hurtigste vej, “distance”, den korteste vej eller “routability”. Hvor den hurtigste vej følges, men med nogle begrænsninger på. Det kan for eksempel være at betalingsveje skal udelukkes.

Der er blacklists og whitelists af “tags” som bruges til at finde ud af, om det er muligt at rute en bestemt vej.

Da det ikke skal være muligt for dronen, at følge andet end højspændingskabler og tårne, behøver der ikke være noget i blacklisten.

Blacklist’en bruges til at begrænse, hvor det er muligt at bevæge sig hen. For eksempel skal det ikke være muligt at planlægge en rute for en bil, gennem en privat indkørsel. Private indkørsler skal således “blacklistes”.

5.2.4 Implementation

Der findes ikke en profil for droner i OSRM. Derfor skal en sådan oprettes. For ikke at starte fra bunden af kopieres profilen for fodgænger. Profilen for fodgængere er filen *foot.lua*. Denne danner base til den nye profil.

Listing 6: Tower og power

```
1 tower = way:get_value_by_key('tower'),  
2 power = way:get_value_by_key('power')
```

For at få rutning på højspændingskablerne, bruges “power” tagget i data listen. For højspændingstårnene skal bruges “tower” tagget som kan ses i Listing 6. Disse ”tags” skal skrives ind i listen i profilen, men der skal også ændre i ”setup” i selve profilen.

Der skal ændres i ”access_tag_whitelist” listen, på den måde bliver det muligt

at rute på ”tags’ne” ”power” og ”tower”. ”Tags’ne” skal også skrives i listen ”access_tags_hierarchy” som afgører hvilken prioritet et ”tag” har, det vil sige hvilket der først kigges efter i en node.

Alle hastighederne skal sættes til de rigtige værdier. Da fodgænger profilen danner grundlag for droneprofilen, er ”default_mode” sat til ”mode.walking”. Denne mode er en *integer* der kommer fra ”include/extractor/travel_mode.hpp” og bruges i ”profiles/lib/way_handlers.lua” til at sætte ”forward_mode” og ”backward_mode”. Det skal bruges til nogle sammenligninger i den lua fil, som ikke er relevant for dette projekt.

Mange lister kan efterlades tomme, for eksempel ”access_tag_blacklist”. ”blacklist” bruges til at fortælle profilen, at hvis et ”tag” ligger i ”blacklisten”, så er det ikke muligt for denne profil at rute på den vej der er ”blacklistet”.

Koordinater og punkter

Med OSRM er det muligt at få dronen til det område, der skal inspiceres. Herfra kan OSRM ikke længere bruges, da det ikke understøtter 3D koordinater.

Højspændingsmasternes koordinater skal hentes, så de kan bruges til at arbejde videre med. Dette gøres med java programmet *getLoc.java*.

getLoc.java er i stand til at hente højspændingsmasternes koordinater. I Listing 7 i linje 7, kører programmet en *curl* kommando, for at hente informationer fra OSRM. (mere om dette i Test afsnittet). I linjerne 11 til 13 trimmer programmet dataen, så der kun er oplysningen om *id*, fra de højspændingsmaster på ruten der skal skannes. I linje 14 erstattes alle kommaer, med mellemrum. Det ændres til en streng, da det er det input *getNodes.py* bruger. *getNodes.py* henter og returnere nodernes *id* og længde- og breddegrader.

Listing 7: koordinat hentning

```
1 public class getLoc {
2
3     public static void main(String args[]) {
4         String s = null;
5         try{
6             ...
7             Process p0 = Runtime.getRuntime().exec("curl
8                 http://127.0.0.1:5000/route/v1/walking/
9                 + point1lon + point1lat + point2lon +
```

```

    point2lat + "?steps=true&annotations=
    nodes");
8
9     ...
10    s = stdInput0.readLine();
11    ...
12    int begining = s.indexOf("nodes");
13    int end = s.lastIndexOf("steps");
14    String nodes = s.substring(begining+8, end
15        -4);
16    nodes = nodes.replace(", ", " ");
17    Process p = Runtime.getRuntime().exec("python3 getNodes.py " + nodes);
18
19    BufferedReader stdInput = new BufferedReader(
20        new
21            InputStreamReader(p.getInputStream())));
22
23    while ((s = stdInput.readLine()) != null) {
24        System.out.println(s);
25    }
26
27    ...

```

5.2.5 Test

For at teste OSRM skal backend serveren startes op af OSRM. Denne backend startes op med hjælp af ”Docker”. Først skal kortet pakkes ud, så OSRM kan bruge det, det sker med kommandoen i Listing 8. Kortet til dette projekt er et kort over hele Danmark. Det er muligt at få kort over større byer, hvis det ønskes.

Kortet udpakkes og den valgte profil, i dette tilfælde, den ændrede *foot.lua* profil (droneprofilen). Når kortet udpakkes med en profil, sker der det at det kun er alle de punkter, som det vil være muligt at rute på, der kommer med og alle andre bliver ignoreret.

Listing 8: Udpakning af kort

```

1 sudo docker run -t -v $(pwd):/data osrm/osrm-backend
    osrm-extract -p /data/profiles/foot.lua /data/denmark
    -latest.osm.pbf

```

Når kortet er udpakket, skal det partitioneres i mindre filer, hvor al dataen er sorteret. På den måde er det hurtigere, at finde de data, OSRM har brug for, når den skal lave en rute. Koden til dette kan ses i Listing 9.

Listing 9: partitionering af kort

```
1 sudo docker run -t -v $(pwd):/data osrm/osrm-backend
  osrm-partition /data/denmark-latest.osrm
```

Den sidste ting der skal gøres inden backend serveren kan startes, er at køre en ”customize” kommando på kortet. Dette sker i Listing 10.

Listing 10: customize af kort

```
1 sudo docker run -t -v $(pwd):/data osrm/osrm-backend
  osrm-customize /data/denmark-latest.osrm
```

Backend serveren startes op med kommandoen som ses i 11.

Listing 11: cserver start

```
1 sudo docker run -t -i -p 5000:5000 -v $(pwd):/data osrm/
  osrm-backend osrm-routed --algorithm mld /data/
  denmark-latest.osrm
```

I dette tilfælde åbnes den lokale server på port 5000. Det vil sige at for at få en rute, skal den køres på port 5000. Dette gennemgåes længere nede i rapporten.

For at køre backend serveren, bliver en ”curl” brugt, for at få data ud i terminalen, som kan ses i Listing 12.

De 4 lange kommatal i Listing 12 er længde- og breddegrader på det punkt der rutes fra og til det punkt der rutes til.

steps = true betyder at alle sving printes ud, når der bedes om ruten.

annotations = nodes er en meget vigtig kommando, da det er den der vil printe alle højspændingsmasternes *id* ud. Denne *id* skal bruges til at finde højspændingsmastens længdegrad og bredgrad.

Listing 12: cserver start

```
1 curl "http://127.0.0.1:5000/route/v1/walking
  /10.327102,55.346126;10.330245,55.342476?steps=true&
  annotations=nodes"
```

```
→ osrm-backend git:(master) ✘ curl "http://127.0.0.1:5000/route/v1/walking/10.327102,5
5.346126;10.330245,55.342476?steps=true&annotations=nodes"
{"code":"Ok", "routes": [{"geometry": "csxpI{t_-@pQi[", "legs": [{"annotation": {"nodes": [831
300406, 831300506, 831300392, 831300438]}], "steps": [{"intersections": [{"out": 0, "entry": [true
e], "bearings": [139], "location": [10.32542, 55.3453]}]}, {"driving_side": "right", "geometry": "csxpI{t_-@|@{A|HcNtEiI", "mode": "walking", "duration": 52.4, "maneuver": {"bearing_after": 13
9, "type": "depart", "modifier": "left", "bearing_before": 0, "location": [10.32542, 55.3453]}, "weight": 52.4, "distance": 437.4, "name": ""}, {"intersections": [{"in": 0, "entry": [true], "bear
ings": [319], "location": [10.32995, 55.342329]}]}, {"driving_side": "right", "geometry": "q`xpIe
q`~@", "mode": "walking", "duration": 0, "maneuver": {"bearing_after": 0, "type": "arrive", "modi
fier": "left", "bearing_before": 139, "location": [10.32995, 55.342329]}, "weight": 0, "distance
": 0, "name": ""}, {"duration": 52.4, "summary": "", "weight": 52.4}], "distance": 437.4, "duration": 52.4, "weight_name": "duration", "weight": 52.4}], "waypoints": [{"hint": "UgAAgDEAAIA1AAAAJQEAAL1AAAC1dgAAUSM0QoD2c0PlVfhGhDDRjUAAAAlAQAAvUAAALV2AAASAAA
rI2dAJSATAM-lJ0AzoNMAzEA_xPUhIHV", "name": "", "location": [10.32542, 55.3453]}, {"hint": "UgAAgDEAAI
AxAAAAvwAAALU-AAAnQA6Ds1QjckH0OoifG-4PRHrjEAAC_AAAatT4AACd5AAASAAAAXp-dApI0TAOf0J0Aj
HVMAY8A7xPUhIHV", "name": "", "location": [10.32995, 55.342329]}]}]}
→ osrm-backend git:(master) ✘
```

Figur 3: Curl test

Som det fremgår i figur 3 skriver OSRM serveren, at det lykkedes at finde en rute ved at sige, ”code” : ”Ok” efterfulgt af en del som ikke er relevant for dette projekt.

Det næste i testen som er relevant er ”annotation” : {”nodes” : [, for listen efter dette er fyldt med alle højspændingsmaternes *id*’er.

Næsten i slutningen af test printet, står *duration* som fortæller lidt om hvor lang tid turen vil tage. I denne test er der kun lidt over 400 meter, som også står beskrevet i test outputet som *distance*.

OSRM har en frontend version, som vil gøre det nemmere for personer, der ikke har kendskab til kode at bruge.

Denne frontend skal startes op efter backenden er startet op. Frontenden opstartes med koden i Listing 13.

Listing 13: Server start

```
1 sudo docker run -p 9966:9966 osrm/osrm-frontend
```

Som det ses i Listing 13 så startes frontend, i dette tilfælde, op på port 9966. Derfor skal åbnes et browserwindow på denne port. Dette kan gøres i commandprompt med koden som ses i Listing 14

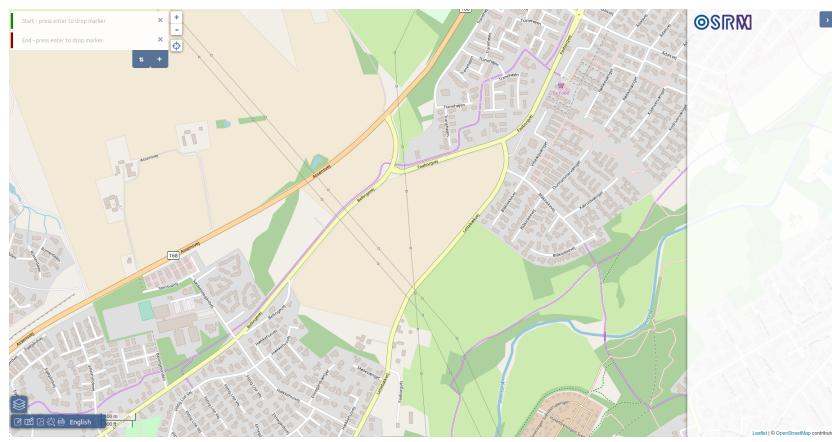
Listing 14: Side åbning

```
1 xdg-open 'http://127.0.0.1:9966'
```

Det er også muligt at indtaste `http://127.0.0.1:9966` i søgefældet på ens browser.

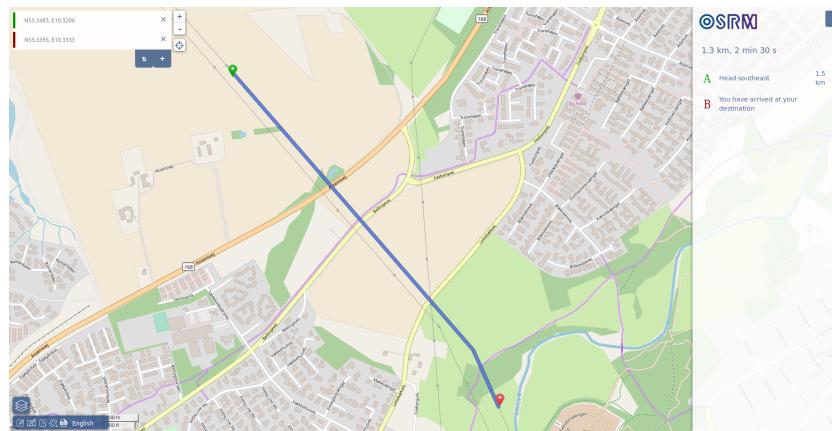
127.0.0.1 er localhost.

Frontend til OSRM ses i Figur 4



Figur 4: OSRM

Det er muligt at finde en rute mellem to steder ved at trykke to steder i OSRM frontend. Dette ses i Figur 5.



Figur 5: OSRM rute

Test af `getLoc.java`

Det er desværre ikke muligt at vise test af `getLoc.java`.

Computeren som OSRM var installeret på, brød desværre sammen. Der skulle derfor geninstalleres styresystem på computeren. Ubuntu har i mellemtiden, udgivet en stor opdatering og derfor har Docker også opdateret deres system. Docker bruges til at køre OSRM. Desværre er der, med de fleste store opdateringer, altid nogle små *bugs* som skal udbedres, og dette tilfælde er ikke anderledes. Docker vil bruge en ny version af et *libc* bibliotek, men det kan ikke findes selvom det tydeligt er i systemet. Det er blevet testet med 3 forskellige versioner af Ubuntu, 4 forskellige versioner af Docker og 3 forskellige versioner af OSRM og fejlen opstår ved alle. Det menes at kerneprogrammet i Docker leder efter den nyeste version af *libc* i Ubuntu og der er sket en fejl med stien til dette.

Men `getLoc.java` er blevet testet grundigt inden system nedbruddet. Desværre blev det ikke dokumenteret. Men testes med noden med *id* 341686702, bliver output:

```
341686702
10.3367190
55.3462730
```

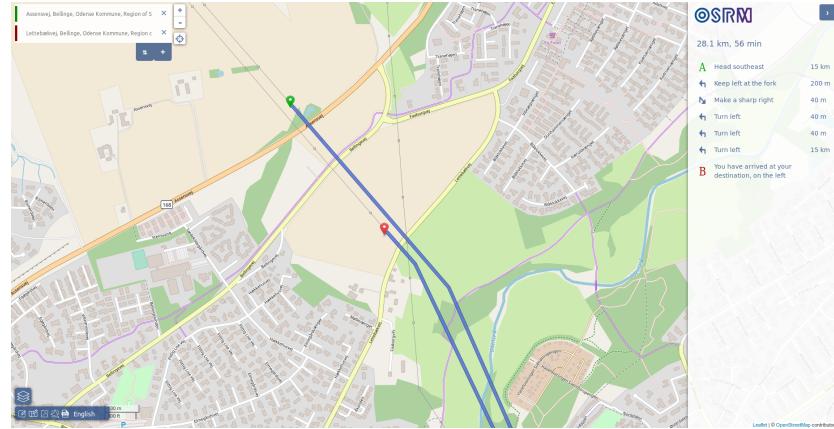
Det første er *id* efterfulgt af længde- og bredegrad. Dette vil ske for hver *id* som koden bliver givet.

5.2.6 Fejl og mangler

OSRM er meget omfattende og det tager lang tid bare at sætte sig ind i koden til en profil. Dokumentationen af koden til OSRM, er ikke så omfattende. Dette har besværliggjort processen meget. OSRM kodes i programmeringssproget *lua*. Et sprog der ikke er ikke er en del af pensum eller bliver undervist i på Syddansk Universitet. Det har derfor været en meget tidskrævende process, at sætte sig ind i. En simpel ting, som at få OSRM til at rute langs højspændings kablerne, har taget utrolig lang tid. Derfor har det ikke været tid nok, til at løse følgende udfordringer.

Det er ikke muligt for dronen at ”springe” fra et kabel til et andet, selv ikke

engang hvis disse kabler krydser hinanden. I i Figur 6 og 7 ses resultatet af hvis, det forøges.

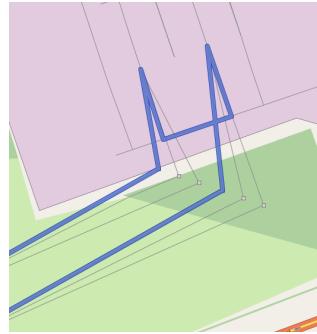


Figur 6: OSRM rutnings problem 1

OSRM vil rute dronen hele vejen hen til en omformer station, som ses i Figur 7, hvor disse to kabler mødes, som ses i Figur 8.



Figur 7: OSRM rutnings problem 2



Figur 8: OSRM rutnings problem 3

Hvis kablerne ikke går til samme omformer station, så prøver OSRM at løse opgaven ved, at prøve på kabler der går længere væk. Hvis to kabler har en afstand på f.eks. 20 meter, men ikke krydser hinanden eller ikke forbindes ved sammen transformerstation, kan resultere i dronen vil flyve hele landet rundt, inden den kan komme til sin destination.

Dronen kan ikke flyve væk fra kablerne i OSRM, dette ville ellers have løst problemet med at flyve fra et kabel til et andet, der ikke er forbundet.

6 3D flyruter

6.1 Design

Det er muligt at finde længdegrad- og breddegrads-koordinatsæt for el-masterne i OSM. Disse længdegrad- og breddegrads-koordinatsæt er toppunktet af el-masten. Ved hjælp af masternes længdegrad- og breddegrads-koordinatsæt, er det muligt at beregne de eksakte længdegrad- og breddegrads-koordinatsæt, for hvor kablerne er monteret på masterne.

I dette projekt, er der taget udgangspunkt i den masttype, der har 6 kabler monteret, 3 på hver side. Dronen skal inspicere højspændingskablerne ved at flyve langs kablerne, i den bue, der dannes når kablerne udspændes mellem masterne. Et kabel af gangen.

Afstanden fra toppunktet af el-masten ud til hvor kablerne er monteret fremgår af bilag 2.

Dronen skal flyve langs et kabel fra monteringspunktet, på den ene mast til monteringspunktet på den næste mast. Dronen skal have koordinatsæt bestående af x, y og z koordinater. Udregningerne deles op, så der beregnes på ruten set ovenfra (x, y) og derefter beregnes på højden, buen på kablet (z), og de to samles. Ved hjælp af masternes længdegrad- og breddegrads-koordinatsæt, er det muligt at beregne de eksakte længdegrad- og breddegrads-koordinatsæt, for hvor kablerne er monteret på masterne. Ved at beregne differencen mellem mastens længdegrad- og breddegrads-koordinat og koordinatsættene for de 6 kabelmonteringspunkter, kan beregningen anvendes til alle master af denne type.

6.1.1 Beregning af center punkter mellem 2 master

Der tages udgangspunkt i centerlinjen mellem masterne i den første beregning. I beregningen ”ses” masterne ovenfra og her beregnes (x, y) koordinatsæt mellem 2 master, med en givet afstand, (*intervaldistancen*), f.eks. pr. 1/2 meter.

Masternes placering i forhold til hinanden vil danne en vinkel i forhold til x-aksen, denne skal bruges for at kunne regne ud om den beregnede længde

skal ligges til eller trækkes fra koordinaterne for mastens centerpunkt. Den beregnede længde afgører hvormeget der skal lægges til/trækkes fra i x og y koordinaterne, med en givet vinklen.

Vinklen til x-aksen beregnes ved hjælp af masternes x og y værdier og tangens⁸. Figur 9a

$$\tan(v) = \frac{\text{modst  ende}}{\text{hosliggende}} \rightarrow v = \tan^{-1}\left(\frac{|\text{Mast2}y - \text{Mast1}y|}{|\text{Mast2}x - \text{Mast1}x|}\right)$$

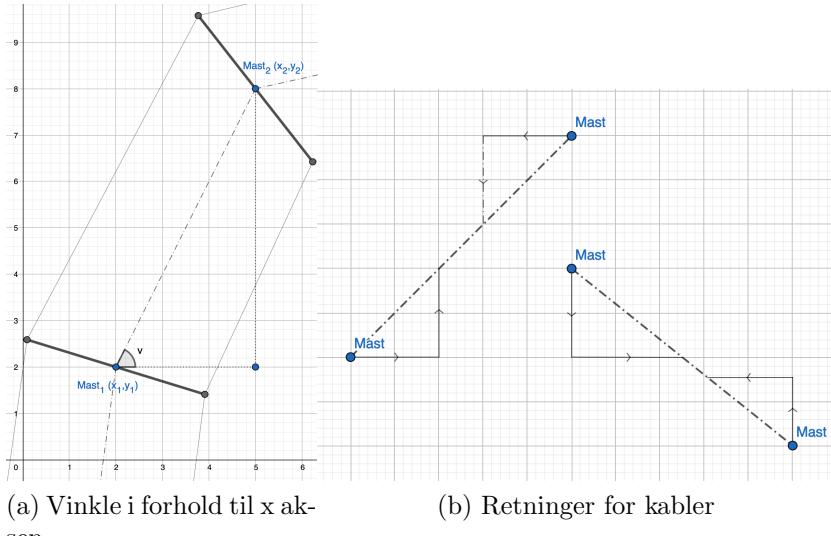


Figure 9

Beregningen af den længde, der skal ligges til/trækkes fra x koordinaten udregnes:

$$\cos(v) * \text{intervaldistance}$$

L  ngden der skal lægges til/trækkes fra y koordinaten udregnes:

$$\sin(v) * \text{intervaldistance}$$

Masternes placering i forhold til hinanden, og dermed retningen af kablerne, giver 4 muligheder for, om der skal lægges til eller tr  kkes fra koordinaterne for den mast dronen begynder sin inspektion ved. Se figur 9b.

⁸Adams og Essex (2014), Calculus Eighth Edition, siden 55

De beregnede x, y koordinatsæt, på centerlinjen mellem de to master, gemmes i en liste.

6.1.2 Beregning af center punkter for kabelbuen

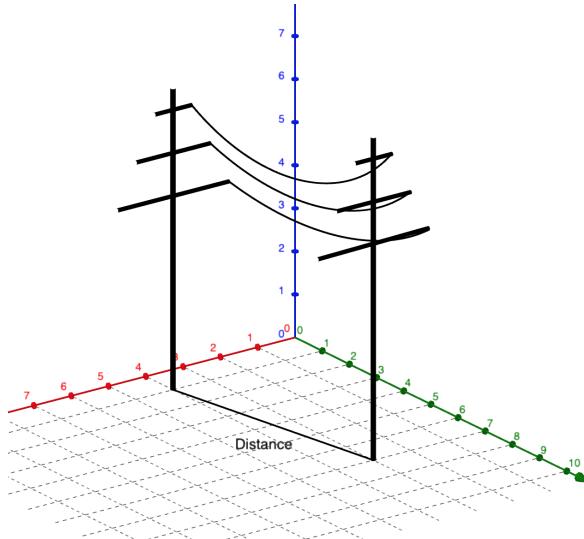
Dronen skal følge højspændingskablerne i den bue, der dannes når kablerne udspændes mellem masterne. I beregningen ovenfor findes x, y koordinatsæt, til centerlinjen mellem masterne. Nedenfor findes parablen, for kabelbuen mellem masterne. I beregningerne ses masterne fra siden og er angivet med x, z koordinatsæt. Til sidst samles centerpunkter koordinaterne med z koordinaterne.

Ligningen for parablen⁹:

$$z = ax^2 + bx + c$$

Kablerne er udspændt mellem masterne B og C. Mast B er den mast dronen påbegynder inspektionen. Afstanden mellem disse master udregnes ved hjælp af masterne x og y koordinater, hvor masterne ”ses fra oven”. Da beregningen af kabelbuen foretages 2 dimensionelt, ”set fra siden” vil afstanden mellem de to master ikke være korrekt. Derfor sættes Mast B’s x-koordinat til 0, og afstanden mellem masterne vil være *distancen* udregnet ved hjælp af x, y koordinaterne. Se Figur 10

⁹Matematisk Formelsamling, Højt Niveau, (1993), s. 8



Figur 10

$$Distancen = \sqrt{(x_3 - x_1)^2 + (y_2 - y_1)^2} \text{ } ^{10}$$

Mast B's x koordinat sættes til 0, indsættes dette i ligningen for parablen, vil $z_1 = a * 0^2 + b * 0 + c \rightarrow z_1 = c$.

Ved at indsætte dette i ligningen, kan b beregnes:

$$z_3 = ax_3^2 + bx_3 + c \rightarrow z_3 = ax_3^2 + bx_3 + z_1 \rightarrow b = -\frac{ax_3^2 + z_1 - z_3}{x_3}$$

Toppunktets (lavpunktet) koordinatsæt (x_2, z_2) for en parabel er givet ved:

$$(-\frac{b}{2a}, -\frac{d}{4a}) \text{ hvor diskriminanten } d = b^2 - 4ac \text{ } ^{11}$$

z-koordinaten beregnes:

$$z_2 = -\frac{d}{4a} \rightarrow z_2 = -\left(\frac{b^2 - 4ac}{4a}\right) \rightarrow z_2 = \left(-\frac{b^2}{4a}\right) + \left(\frac{4ac}{4a}\right) \rightarrow z_2 = \left(-\frac{b^2}{4a}\right) + c$$

¹⁰Adams og Essex (2014), Calculus Eighth Edition, siden 12.

¹¹Matematisk Formelsamling, Højt Niveau, (1993), s. 8

a beregnes ved at indsætte b og c i ovenstående formel:

$$z_2 = -\frac{(-\frac{ax_3^2+z_1-z_3}{x_3})^2}{4a} + z_1 \rightarrow a = \frac{z_1-2z_2+z_3+2\sqrt{-(z_2-z_3)(z_2+z_1)}}{(x_3)^2}$$

Formlen for parablen:

$$z = \left(\frac{z_1-2z_2+z_3+2\sqrt{-(z_2-z_3)(z_2+z_1)}}{(x_3)^2} \right) x^2 + \left(-\frac{ax_3^2+z_1-z_3}{x_3} \right) x + z_1$$

Mast B's koordinater, (x_1, z_1) svare til $(0, Mast_{Bz})$

Mast C's koordinater, (x_3, z_3) svare til $(Distancen, Mast_{Cz})$

Lavpunktet for kablet, (x_2, z_2) svare til $(lowpoint_x, lowpoint_z)$

Indsat i formlen:

$$z = \left(\frac{Mast_{Bz}-2*lowpoint_z+Mast_{Cz}+2\sqrt{-(lowpoint_z-Mast_{Cz})(lowpoint_z+Mast_{Bz})}}{(Distancen)^2} \right) x^2 + \left(-\frac{\left(\frac{Mast_{Bz}-2*lowpoint_z+Mast_{Cz}+2\sqrt{-(lowpoint_z-Mast_{Cz})(lowpoint_z+Mast_{Bz})}}{(Distancen)^2} \right) * (Distancen)^2 + Mast_{Bz}-Mast_{Cz}}{Distancen} \right) x + Mast_{Bz}$$

Dette er parablen for højspændingskablet, udspændt mellem 2 master, dronen skal følge. Ved at angive en afstand (x) med et bestemt interval f.eks. pr. 1/2 meter, kan z koordinaterne udregnes, med formlen. z koordinaterne tilføjes listen over x, y koordinater, så de kommer til at ligge i en samlet liste, med x, y og z koordinater for centerlinjen.

Dronen skal flyve langs kablerne, derfor skal monteringspunkterne til kablerne beregnes. Monteringspunkterne regnes ud som en forskel mellem centerpunktet og kabelmonteringspunktet, så den kan anvendes på alle ledninger både på højresiden og venstresiden af masten.

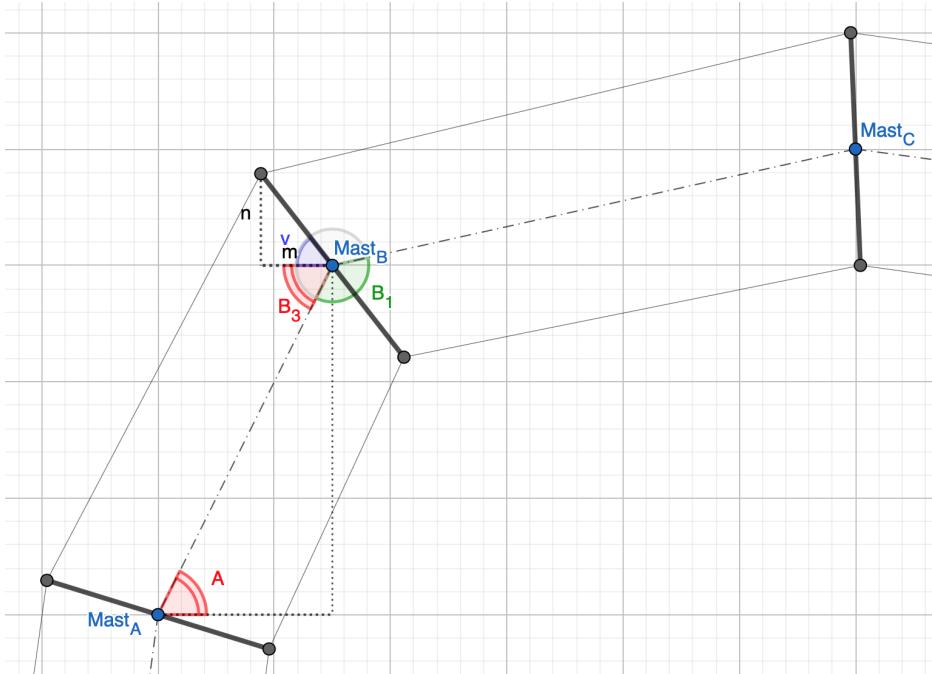
6.1.3 Beregning af kabelmonteringspunkter

I beregningen af kabel monteringspunkterne ses masterne "ovenfra" og er angivet som x, y koordinatsæt.

For at beregne kabel monteringspunkterne til en given mast, benyttes den foregående og den efterfølgende mast's koordinatsæt, samt afstanden ud til de

enkelte kabel monteringspunkter. Masternes koordinatsæt samt længderne ud til monteringspunkterne er kendte parametre.

Se de indsatte figurer som visualiserer beregningerne.



Figur 11: Mast A, B og C set ovenfra

”Hældningen” på kablerne mellem masterne beregnes, for at finde vinklen mellem dem, vinkel B_1 . Ved at trække denne vinkel fra $2 * pi$, findes den ydre vinkel, vinklen mellem kablerne. Mastens bærearm står midt mellem de to kabler, derved halveres vinklen af bærearmen, vinkel B_2 .

Hældningen på kabel mellem to master: $a = \frac{|y_2 - y_1|}{|x_2 - x_1|}$ 12

Sammenhængen mellem hældning på en linje og vinklen linjen danner med vandret: $a = \tan(v) \rightarrow v = \tan^{-1}(a)$ ¹³

¹²Matematisk Formelsamling, Højt Niveau, (1993), s. 7

¹³Matematisk Formelsamling, Højt Niveau, (1993), s. 7

Vinklen mellem linjerne, $B_1 = pi - |\tan^{-1}(a_1) + \tan^{-1}(a_2)|$

Halvdelen af den ydre vinkel, $B_2 = \frac{2*pi - B_1}{2}$

Vinklen v skal udregnes for at kunne beregne koordinaterne til monteringspunkterne. Vinklen v , bærearmens vinkel, er vinklen $B_2 - B_3$. Vinklen B_3 svare til vinklen A . Vinklen A beregnes ved hjælp af $\tan(A) = \frac{\text{modstående}}{\text{hostiggende}}$ ¹⁴

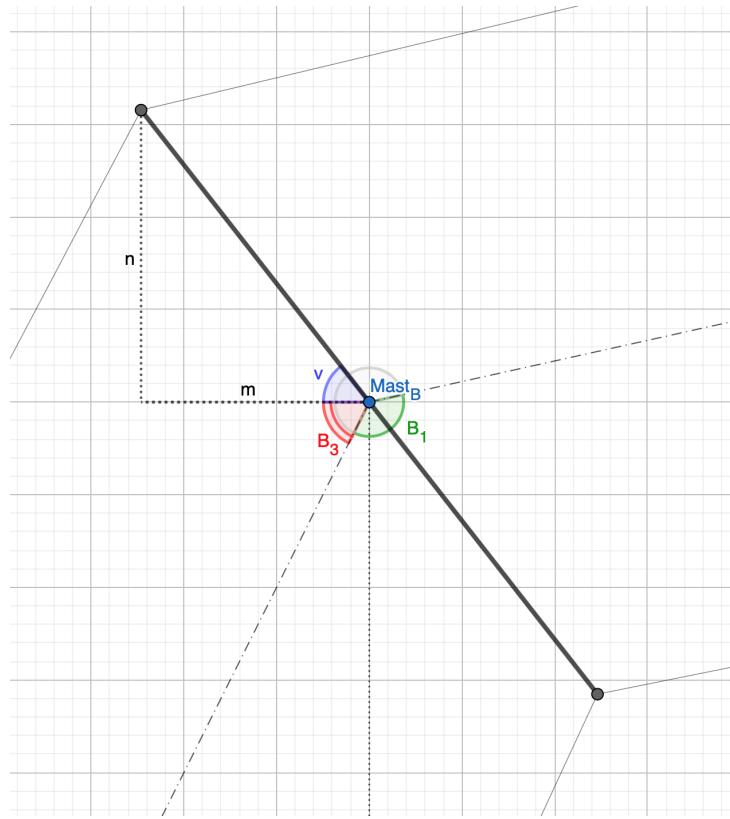
Vinkel $B_3 = A = \tan^{-1}\left(\frac{|y_2 - y_1|}{|x_2 - x_1|}\right)$

Vinkel $v = B_2 - B_3$

$v = \frac{2*pi - pi - |\tan^{-1}(a_1) + \tan^{-1}(a_2)|}{2} - \tan^{-1}\left(\frac{|y_2 - y_1|}{|x_2 - x_1|}\right)$

Når vinklen v er beregnet, kan koordinatsæt til monteringspunkt på Mast B findes ved hjælp af linjestykkerne n og m som ses i Figur 12.

¹⁴Adams og Essex (2014), Calculus Eighth Edition, siden 55.



Figur 12: Mast B samt linjestykkerne m og n

Med udgangspunkt i figur ovenfor vil beregningen være som følger:

kabel_B's koordinatsæt: $(x_B - m, y_B + n)$

$$\sin(V) = \frac{\text{modstående}}{\text{hypotenusen}}^{15} \rightarrow \sin(V) = \frac{n}{\text{længde}_{Mast-Kabel}} \rightarrow n = \sin(V) * \text{længde}_{Mast-Kabel}$$

$$y_{kabelB} = y_B + n$$

$$y_{kabelB} = y_B + \sin(V) * \text{længde}_{Mast-Kabel}$$

$$\cos(V) = \frac{\text{hosliggende}}{\text{hypotenusen}}^{16} \rightarrow \cos(V) = \frac{m}{\text{længde}_{Mast-Kabel}} \rightarrow m = \cos(V) * \text{længde}_{Mast-Kabel}$$

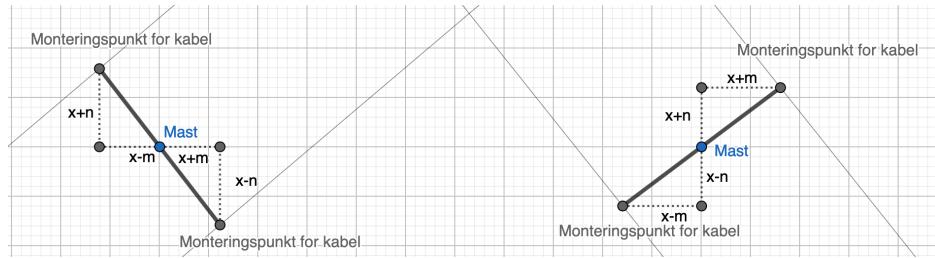
¹⁵Adams og Essex (2014), Calculus Eighth Edition, siden 55.

¹⁶Adams og Essex (2014), Calculus Eighth Edition, siden 55.

$$x_{kabelB} = x_B - m$$

$$x_{kabelB} = x_B - \cos(V) * længde_{Mast-Kabel}$$

Alt efter hvor kablerne er monteret og hvordan masterne er placeret afgøre om m og n skal lægge til eller trækkes fra i beregningen. Se figur 13.



Figur 13: Mast og retning af kabler

x_B og y_B er mastens centerpunkt og m og n er den beregnede afstand ud til kabel monteringspunkterne. Ved at anvende de beregnede afstande til centerpunktet, kan genereres (x, y, z) koordinater langs kablerne mellem masterne, både på højresiden og venstresiden af masten. Disse koordinatsæt kan dronen anvende som flyrute.

6.2 Implementation

Med udgangspunkt i design afsnittet, implementeres koden i java. De parametre der er tilgængelige er masternes længde- og breddegrads koordinater, afstanden fra mastens centerpunkt og ud til hvor de 6 kablerne er monteret og hvormeget kablet hænger ned mellem to master.

Metoden *calcCable* samler alle metoder i koden, og køres fra main, *calcCable* beskrives i det følgende. Alle metoder nævnt i den kommende afsnit vil blive gennemgået i større detaljer i efterfølgende.

Metoden *calcCable* tager følgende argumenter: Afstand mellem ”inspektionsspukterne” *intervalDistance*, 3 på hinanden følgende master, den første mast bruges kun x, y koordinater og for de to næste master x, y og z koordinater, lavpunkter på kablet (z koordinaten) *lowpoint* samt to lister med afstandende fra masternes centerpunkt lodret ned til kabelmonteringspunkterne, *zDiff*, og vandret ud til kabelmonteringspunkterne, *xyDiff*.

calcCable bruger metoden *calcCenterPoints* til at beregne x, y koordinater lange centerlinjen mellem to master, med et givet interval. De returneres i listen *centerXY*.

calcParabola beregner z koordinaten til de x, y koordinater fundet med *calcCenterPoints*. Det hele samles i listen *centerCoords*.

calcOffsets beregner offsettet ud til kablemonteringspunkterne, det vil sige, i hvilken retning og med hvilken afstand kablerne sidder. Disse oplysninger gemmes i listen, *offsetsL* og i *for – løkken* findes højresiden og gemmes i listen *offsetsR*.

I en for-løkke beregnes x, y og z koordinater, med metoden *offsetPoints*, først for kablerne på venstresiden af masten og derefter på højresiden, de tilføjes listen *XYZcoords*.

Resultatet af *calcCable* er en liste, *XYZcoords*, bestående af to lister, med de kabler der sidder på henholdsvis højresiden og venstre siden af masten. I dette tilfælde vil højresidelisten og venstresidelisten indeholde 3 lister hver, en for hvert kabel på hver side. I de lister vil ligge x, y og z koordinater, for punkter på de enkelte kabler. Der kan tilføjes flere kabler i højre- og venstresidelisterne.

[[[xyzxyz...][xyzxyz...][xyzxyz...]..][[xyzxyz...][xyzxyz...][xyzxyz...]..]]

Listing 15: calcCable

```
1  public static ArrayList<ArrayList<ArrayList<Double
2      >>> calcCable(double intervalDistance, double
3      point1x, double point1y, double point2x, double
4      point2y, double point2z, double point3x, double
5      point3y, double point3z, double lowpoint,
6      ArrayList<Double> zDiff, ArrayList<Double>
7      xyDiff){
8
9      ArrayList<ArrayList<ArrayList<Double>>>
10     XYZcoords = new ArrayList<ArrayList<
11         ArrayList<Double>>>();
12
13     XYZcoords.add(new ArrayList());
14     XYZcoords.add(new ArrayList());
15
16     ArrayList<Double> centerXY = calcCenterPoints(
17         intervalDistance, point2x, point2y, point3x,
18         point3y);
19
20     ArrayList<Double> centerCoords = calcParabola(
21         point2x, point2y, point2z, point3x, point3y,
22         point3z, lowpoint, intervalDistance,
23         centerXY);
24
25     ArrayList<Double> offsetsL = new ArrayList();
26     ArrayList<Double> offsetsR = new ArrayList();
27     //tånet tegning på tavlen - spørg Danny igen
28     offsetsL = calcOffsets(point1x, point1y,
29         point2x, point2y, point3x, point3y, zDiff,
30         xyDiff);
31
32     for(int i = 0; i < offsetsL.size()-2; i+=3){
33         offsetsR.add(-1*offsetsL.get(i));
34         offsetsR.add(-1*offsetsL.get(i+1));
35         offsetsR.add(offsetsL.get(i+2));
36     }
37
38     for(int i = 0; i < offsetsL.size(); i += 3){
39         XYZcoords.get(0).add(offsetPoints(
40             centerCoords, offsetsL.get(i), offsetsL.
41             get(i+1), offsetsL.get(i+2)));
42         XYZcoords.get(1).add(offsetPoints(
43             centerCoords, offsetsR.get(i), offsetsR.
```

```

24         get(i+1), offsetsR.get(i+2)));
25     }
26 }

```

6.2.1 Center punkter mellem 2 master

Som beskrevet i designafsnittet tages der udgangspunkt i centerlinjen mellem masterne ”ses” ovenfra og her beregnes (x, y) koordinatsæt mellem 2 master, løbende med en givet afstand, *intervalDistance*. Metoden *calcCenterPoints* tager to masters x og y koordinater og *intervalDistance* som argumenter.

Vinklen til x-aksen, *angle*, beregnes ved hjælp af masternes x og y værdier og tangens.

Med vinklen, *angle*, og distancen, *intervalDistance*, beregnes den længde, der skal ligges til/trækkes fra for at finde næste punkt.

opposite beregner længden der skal lægges til/trækkes fra y koordinaten.
adjacent beregner for x koordinaten.

Masternes placering i forhold til hinanden, afgøre om længderne *opposite* og *adjacent* skal lægges til, eller trækkes fra x og y koordinaterne. De 4 muligheder beskrevet i designafsnittet er implementeret i en *if/elseif – statement*, hvor de 2 første muligheder er vist i koden i Listing 16. Hele koden kan ses i appendix. I *whileløkken* lægges til/trækkes fra de forgående koordinater, og de tilføjes til listen *Xycoords*. Koordinaterne for den første mast tilføjes til listen inden *if – statementen* og koordinatesættet for ”slut” masten tilføjes til listen til sidst i *whileløkken*.

Listing 16: calcCenterPoint

```

1  public static ArrayList calcCenterPoints(double
2      intervalDistance, double a_x, double a_y, double
3      b_x, double b_y){
4      double angle = Math.atan(Math.abs(b_y-a_y)/Math.
5          abs(b_x-a_x));
6      double opposite = Math.sin(angle)*
7          intervalDistance;

```

```

4     double adjacent = Math.cos(angle)*
5         intervalDistance;
6     double x = a_x;
7     double y = a_y;
8
9     ArrayList<Double> Xycoords = new ArrayList();
10    Xycoords.add(a_x);
11    Xycoords.add(a_y);
12
13    if(a_x < b_x && a_y <= b_y){//up right and right
14        while (x+adjacent<=b_x && y+opposite <= b_y)
15        {
16            x += adjacent;
17            y += opposite;
18            Xycoords.add(x);
19            Xycoords.add(y);
20        }
21        //add last point
22        if(x < b_x || y < b_y){
23            Xycoords.add(b_x);
24            Xycoords.add(b_y);
25        }
26    }else if(a_x >= b_x && a_y < b_y){//up and up
27        left
28        while (x-adjacent >= b_x && y+opposite < b_y
29        ) {
30            x -= adjacent;
31            y += opposite;
32            Xycoords.add(x);
33            Xycoords.add(y);
34        }
35
36        ...
37
38        return Xycoords;
39    }

```

6.2.2 Centerpunkter for kabelbuen

Buen kablerne danner mellem masterne, er en parabel, nedefor vises implementationen af denne. Masterne ”ses” fra siden og er angivet med (x, z) koordinatsæt.

Metoden *calcParabola* tager (x, y, z) koordinaterne for de to master, *intervalDistance*, z koordinaten til lavpunktet på parablen, *lowpoint_z*, og listen *Xycoords*.

Afstanden, *distance*, mellem masterne beregnes udfra masternes *x* og *y* koordinater, dette giver mast C's x koordinat. Mast B's x koordinat sættes, som beskrevet i designafsnittet, til 0.

a , b , og c , i ligningen for parablen $z = ax^2 + bx + c$, beregnes som angivet i designafsnittet.

I et *for-loop* beregnes z koordinaterne for buen mellem de to master. Startende ved den første masts x koordinat, 0, til den anden masts x koordinat, distancen, med samme interval som i centerkoordinaterne, *intervalDistance*. Da de z koordinaterne skal tilføjes de tilsvadende x,y koordinater til de samme punkter. Hentes x,y koordinaterne med *XYcoords.get()*; og tilføjes en ny liste, *XYZcoords*, sammen med z-koordinaterne. Den sidste del af koden tilføjer den sidste koordinat, svarende til punktet på den sidste mast. Hele listen returnes til sidst.

Listing 17: calcParabola

```
1  public static ArrayList<Double> calcParabola(double
2      piont2x, double piont2y, double piont2z, double
3      piont3x, double piont3y, double piont3z, double
4      lowPoint_z, double intervalDistance, ArrayList<
5      Double> XYcoords){
6
6      double z, distance, temp_a1, temp_a2, a, b, c;
7      distance = Math.sqrt(Math.pow((piont3x-piont2x),
8          2)+Math.pow((piont3y-piont2y), 2));
9
10     temp_a1 = (piont2z-2*lowPoint_z+piont3z);
11     temp_a2 = -(lowPoint_z-piont3z)*(-lowPoint_z+  
12         intervalDistance/2);  
13     a = temp_a1*temp_a1-4*temp_a2;  
14     if (a<0){  
15         XYcoords.add(piont2y);  
16         XYcoords.add(piont3y);  
17         return XYcoords;  
18     }  
19     b = temp_a1*piont2y+temp_a2;  
20     c = temp_a2*piont2y;  
21     double discriminant = Math.sqrt(b*b-a*c);  
22     double x1 = (-b+discriminant)/(2*a);  
23     double x2 = (-b-discriminant)/(2*a);  
24     if (x1<lowPoint_z){  
25         XYcoords.add(piont2y);  
26         XYcoords.add(piont3y);  
27         return XYcoords;  
28     }  
29     if (x2>lowPoint_z){  
30         XYcoords.add(piont2y);  
31         XYcoords.add(piont3y);  
32         return XYcoords;  
33     }  
34     XYcoords.add(piont2y);  
35     XYcoords.add(piont3y);  
36     return XYcoords;  
37 }
```

```

8         piont2z);
9         a = (temp_a1+2*Math.sqrt(temp_a2))/Math.pow(
10            distance, 2);
11         b = -(a*Math.pow(distance, 2)+piont2z-piont3z)/(
12            distance);
13         c = piont2z;
14
15         ArrayList<Double> XYZcoords = new ArrayList<
16             Double>();
17         double x;
18         int adding= 0;
19         for(x = 0.0; x<distance; x+=intervalDistance){
20             XYZcoords.add(XYcoords.get(adding));
21             XYZcoords.add(XYcoords.get(adding+1));
22             XYZcoords.add(z = a*(Math.pow(x, 2))+b*x+c);
23             adding += 2;
24         }
25         x = distance;
26         XYZcoords.add(XYcoords.get(adding));
27         XYZcoords.add(XYcoords.get(adding+1));
28         XYZcoords.add(z = a*(Math.pow(x, 2))+b*x+c);
29
30         return XYZcoords;
31     }

```

6.2.3 Kabelmonteringspunkter

I beregningen af kabel monteringspunkterne ses masterne ”ovenfra” og er angivet som (x, y) koordinatsæt.

Implementationen af kabel monteringspunkterne, sker i *calcOffsets*, metoden tager 3 på hinanden følgende masters x og y koordinater, samt afstanden ud og ned til de enkelte kabel monteringspunkter. De horisontale afstande ligger i listen *xyDiff*, de vertikale i listen *zDiff*.

Først beregnes vinklen *angleV*, med *calcAngle*, se Listing 19. Denne vinkel er bæreammens vinkel.

I *if/elseif – stantmenten* er defineret de 4 forskellige retninger på kablerne og alt efter retningen på kablerne, vælges den rette if/elseif. I *for – løkken* tilføjes, differencen mellem centerpunktet på masten og x og y koordinaterne

for kabelmonteringpunktene. Punkterne udregnes med cos, x koordinaten eller sin, y koordinaten, til bærearamens vinkel, ganget med den horisontale afstand, hentet med, $xyDiff.get(i)$, fra listen $xyDiff$ ud til kabelmonteringpunktet.

x og y differencen tilføjes listen $listOfDist$ sammen med afstanden verticalt hentet fra listen $zDiff$.

Listing 18: calcOffsets

```

1  public static ArrayList<Double> calcOffsets(
2      double Mast_A_x, double Mast_A_y, double
3      Mast_B_x, double Mast_B_y, double Mast_C_x,
4      double Mast_C_y, ArrayList<Double> zDiff,
5      ArrayList<Double> xyDiff){
6      ArrayList<Double> listOfDist = new ArrayList<>()
7          ;
8
9      double angleV = calcAngle(Mast_A_x, Mast_A_y,
10         Mast_B_x, Mast_B_y, Mast_C_x, Mast_C_y);
11
12      if(Mast_B_x < Mast_C_x && Mast_B_y <= Mast_C_y){
13          //up right and right
14          for(int i = 0; i < xyDiff.size(); i++){
15              listOfDist.add(-Math.cos(angleV)*xyDiff.
16                  get(i));
17              listOfDist.add(Math.sin(angleV)*xyDiff.
18                  get(i));
19              listOfDist.add(zDiff.get(i));
20          }
21      }else if(Mast_B_x >= Mast_C_x && Mast_B_y <
22          Mast_C_y){//up and up left
23          for(int i = 0; i < xyDiff.size(); i++){
24              listOfDist.add(-Math.cos(angleV)*xyDiff.
25                  get(i));
26              listOfDist.add(-Math.sin(angleV)*xyDiff.
27                  get(i));
28              listOfDist.add(zDiff.get(i));
29          }
30      }
31
32      ...
33  }
```

```

20
21      }
22      return listOfDist;
23  }

```

Beregningen af vinklen v , derfineret i designafsnittet, er implementeret i funktionen calcAngle. Argumenterne er x og y koordinaterne til de 3 master. Først beregnes ”hældningen” på de to kabler, $a1$ og $a2$. Derefter beregnes vinklen v , jævnføre beregningerne i designafsnittet. Beregningen er udført i en enkelt linje. Vinklen v returneres.

Listing 19: calcAngle

```

1  public static double calcAngle(double point1x,
2      double point1y, double point2x, double point2y,
3      double point3x, double point3y){
4
5      double a1, a2, v1;
6      a1 = Math.abs(point1y-point2y)/Math.abs(point1x-
7          point2x);
8      a2 = Math.abs(point2y-point3y)/Math.abs(point2x-
9          point3x);
10
11     v = (Math.PI*2 - (Math.PI - Math.abs(Math.atan(
12         a1) - Math.atan(a2))))/2-(Math.atan(Math.abs(
13         point1y-point2y)/Math.abs(point1x-point2x)));
14
15     return v;
16  }

```

calcOffsets gav offsettet ud til kablemonteringspunkterne, det vil sige, i hvilken retning og med hvilken afstand kablerne sidder. Disse gemmes i to lister, *offsetsL* og *offsetsR*.

Metoden *offsetPoints* tager listen af centerkoordinater *centerCoords* og x, y og z offsetkoordinaterne for kablerne fra listerne *offsetsL* eller *offsetsR*. For hver koordinat i listen *centerCoords*, her kaldet *points*, ligges offsettet til i x , y retningen og trækkes fra i z retningen. Dette gøres for alle punkterne i listen, *offsetsPoints*, det vil sige langs hele kablet.

Listing 20: offsetPoints

```
1  public static ArrayList offsetPoints(ArrayList<
2      Double> points, double x, double y, double z){
3      ArrayList<Double> offsetPointList = new
4          ArrayList<Double>();
5      for(int i = 0; i < points.size(); i+=3){
6          offsetPointList.add(points.get(i) + x);
7          offsetPointList.add(points.get(i+1) + y);
8          offsetPointList.add(points.get(i+2) - z);
9      }
10     return offsetPointList;
11 }
```

Resultatet returneres til *calcCable*, med *for-løkken* sørger for at beregne på kablerne på venstre siden af masten. Resultatet af *calcCable* bliver således en liste, bestående af to lister, med de kabler der sidder på henholdsvis højresiden og venstre siden af masten. Højre og venstre side listerne indeholder lister med x, y og z koordinater til alle punkter på hvert kabel.

6.3 Test

Test udføres på alle metoder gennemgået i implementations afsnittet. Der vil testes med test data, hvor kablerne går i fire forskellige retninger, samt en test hvor kablet løber i en 90 graders vinkel.

De 5 test udføres ud fra følgende data:

Interval distancen	Testdata 1			Testdata 2			Testdata 3			Testdata 4			Testdata 5		
	1			1			1			1			1		
	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
Mast 1	5	3		13	3		20	20		1	14		5	1	
Mast 2	7	4	54.49	10	7	64.49	12	12	64.49	5	7	64.49	5	5	54.49
Mast 3	13	7	64.49	2	12	54.49	5	5	64.49	13	4	64.49	10	5	64.49
Lavpunkt på parablen (lowpoint)			44.49			44.49			44.49			67.0			44.49
zDiff	Liste			Liste			Liste			Liste			Liste		
xyDiff	Liste			Liste			Liste			Liste			Liste		

Figur 14: Testdata til 5 tests

	Top cable	Middle cable	Low cable
zDiff	8.69	8.69+9.02	8.69+9.02+8.98
xyDiff	2.05	4.00	2.00

Figur 15: Data i zDiff og xyDiff listerne

6.3.1 Center punkter mellem 2 master

Testen af centerpunkter mellem to master, tager udgangspunkt i to masters x og y koordinater og beregner x og y koordinater mellem disse to master, løbende med en afstand på $intervaldistancen$.

I Figur 16a ses test udført med testdata 1, det vil sige der beregnes på kablet mellem mast 2 med koordinaterne (7, 4) og mast 3 med koordinaterne (13, 7) med et interval på 1. Centerlinjen mellem mast 2 og 3, løber fra venstre mod højre, stigende x-værdi, og op, stigende y-værdi. Hvilken Figur 16a bekræfter.

Figur 16b viser test med testdata 2, Mast 2 (10, 7) og Mast 3 (2, 12) og intervaldistance på 1. Centerlinjen løber fra højre mod venstre og op. Det vil sige faldende x-værdi og stigende y-værdi, hvilket bekræftes af testen.

```
calcCenterPoints, center line x and y coordinates:
x: 7.0 y: 4.0
x: 7.894427190999916 y: 4.447213595499958
x: 8.788854381999831 y: 4.894427190999917
x: 9.683281572999748 y: 5.341640786499875
x: 10.577708763999665 y: 5.788854381999833
x: 11.472135954999581 y: 6.236067977499792
x: 12.366563145999498 y: 6.68328157299975
x: 13.0 y: 7.0

calcCenterPoints, center line x and y coordinates:
x: 10.0 y: 7.0
x: 9.152001695994912 y: 7.52999894000318
x: 8.304003391989824 y: 8.05999788000636
x: 7.456005087984735 y: 8.58999682000954
x: 6.608006783979647 y: 9.11999576001272
x: 5.760008479974559 y: 9.6499947000159
x: 4.9120101759694705 y: 10.179993640019081
x: 4.0640011871964382 y: 10.709992580022261
x: 3.216013567959294 y: 11.239991520025441
x: 2.368015263954206 y: 11.769990460028621
x: 2.0 y: 12.0
```

(a) Fra venstre mod højre og op

(b) Fra højre mod venstre og op

Figure 16: Centerpunkts test

Figur 17a tester med testdata 3, Mast 2 (12, 12) og Mast 3 (5, 5) og intervaldistance på 1. Centerlinjen løber fra højre mod venstre og ned. Det vil sige faldende x-værdi og faldende y-værdi, hvilket fremgår af testen.

I Figur 17b testes med testdata , Mast 2 (5, 7) og Mast 3 (13, 4) og intervaldistance på 1. Centerlinjen løber fra venstre mod højre og ned. x-værdi er stigende og y-værdi faldende, hvilket bekræftes af testen.

```
calcCenterPoints, center line x and y coordinates:
x: 12.0 y: 12.0
x: 11.292893218813452 y: 11.292893218813452
x: 10.585786437626904 y: 10.585786437626904
x: 9.878679656440356 y: 9.878679656440356
x: 9.171572875253808 y: 9.171572875253808
x: 8.46446609406726 y: 8.46446609406726
x: 7.757359312880713 y: 7.757359312880713
x: 7.050252531694165 y: 7.050252531694165
x: 6.343145750507617 y: 6.343145750507617
x: 5.636038969321069 y: 5.636038969321069
x: 5.0 y: 5.0

calcCenterPoints, center line x and y coordinates:
x: 5.0 y: 7.0
x: 5.9363291775690445 y: 6.648876558411608
x: 6.872658355138089 y: 6.297753116823216
x: 7.808987532707134 y: 5.946629675234824
x: 8.745316710276178 y: 5.595506233646432
x: 9.681645887845223 y: 5.24438279205804
x: 10.617975065414267 y: 4.893259350469648
x: 11.554304242983312 y: 4.5421359088812565
x: 12.490633420552356 y: 4.191012467292865
x: 13.0 y: 4.0
```

(a) Fra højre mod venstre og ned

(b) Fra venstre mod højre og ned

Figure 17: Centerpunkts test

Den sidste test der udføres er center linjen vandret. Testen udføres med testdata 5, mast 2 (5, 5), mast 3 (10, 5), intervaldistance på 1 og resultatet bekræftes i Figur 18.

```
calcCenterPoints, center line x and y coordinates:  
x: 5.0 y: 5.0  
x: 6.0 y: 5.0  
x: 7.0 y: 5.0  
x: 8.0 y: 5.0  
x: 9.0 y: 5.0  
x: 10.0 y: 5.0
```

Figur 18: Centerpunkts test

6.3.2 Center punkter for kabelbuen

Centerpunkterne til kabelbuen, parablen, mellem to master, beregnes udfra masternes (x, y, z) koordinater, *intervaldistancen* og listen af (x, y) koordinater mellem masterne. Der beregnes z-koordinater til de tilsvarende (x, y) koordinater fundet ovenfor. Resultatet er en liste af x, y og z koordinater til centerlinjen mellem de to master.

Det er valgt kun at udskrive z-koordinaterne, da det er nemmere at se resultatet. De høje master, i testdataen, er 64.49 meter høje, de lave 54.49. Lavpunktet for kablet ligger på 44.49 meter, med mindre andet er angivet.

Der udføres 5 tests på parablen. Første test er med testdata 1, fra lav mast til høj mast, i Figur 19a ses resultatet hvor z-koordinaterne starter ved den lave masts toppunkt og følger buen ned til lavpunktet og op til toppen af masten.

I test 2 følges buen fra høj til lav mast, hvilket ses i Figur 19b. Her testes med testdata 2.

```

calcParabola, center line z coordinates:
z: 64.49
z: 57.906761176209784
z: 52.63328125685692
calcParabola, center line z coordinates:
z: 54.49
z: 48.58741184416382
z: 45.27523574377039
z: 44.553471698819706
z: 46.42211970931177
z: 50.88117977524659
z: 57.93065189662416
z: 64.49000000000001
z: 46.01559813146323
z: 44.671394925422405
z: 44.636950623818926
z: 45.91226522665279
z: 48.497338733924
z: 52.39217114563255
z: 54.490000000000016

```

(a) Fra lav mast til høj mast

(b) Fra høj til lav mast

Figure 19: Centerpunkter for kabelbuen

Test 3 tester med to lige høje master, Figur 20a, her følges buen også fint mellem masterne ned til lavpunktet og op igen. Testen udføres med testdata 3.

I test 4 ligger lavpunkt højere end masterne, hvilket det ikke gør i virkeligheden, da kablerne ikke vil danne en bue opad. Dronen vil flyve vandret mellem disse to master, hvilket kan ses af Figur 20b.

```

calcParabola, center line z coordinates:
z: 64.49
z: 57.22510617419456
z: 51.59286540961361
z: 47.593277706257155
z: 45.22634306412518
z: 44.492061483217704
z: 45.390432963534714
z: 47.92145750507621
z: 52.0851351078422
z: 57.88146577183268
z: 64.49000000000001
calcParabola, center line z coordinates:
z: 64.49

```

(a) To lige høje master

(b) Lavpunkt højere end masterne

Figure 20: Centerpunkter for kabelbuen

Nedenstående test, Figur 21, viser at der ret faktisk ligger x, y, z koordinater i den endelige liste og at det er de rigtige x, y og z koordinater der sættes sammen. Testen er udført med testdata 1.

```
calcParabola, center line x,y,z coordinates:
7.0 4.0 54.49
7.894427190999916 4.447213595499958 48.58741184416382
8.788854381999831 4.894427190999917 45.27523574377039
9.683281572999748 5.341640786499875 44.553471698819706
10.577708763999665 5.788854381999833 46.42211970931177
11.472135954999581 6.236067977499792 50.88117977524659
12.366563145999498 6.68328157299975 57.93065189662416
13.0 7.0 64.49000000000001
```

Figur 21: Centerpunkter for kabelbuen, x,y,z koordinater

Test udført med *calcParabola*, viser at det er muligt at finde z-koordinaterne mellem to master og at de følger kabelbuen.

6.3.3 Kabelmonteringspunkter

calcOffsets beregner forskellen i afstanden mellem center af masterne og kabelmonteringspunkterne, offsettet.

Beregningen bruger *calcAngle* til at beregne vinklen mellem mastens bærearmlængde og x-aksen. I Figur 22, ses test af vinklen med testdata 1-5. De første 4 linjer viser resultatet med testdata 1-4, her ses at det giver forskellige vinkler. Vinklerne er i radianer. Test med testdata 5 i linje 5, skiller sig lidt ud. Her løber kablerne med et vinkel mellem sig på 90 grader. Da den fundne vinkel er bærearmlængdens vinkel i forhold til vandret, og bærearmlængden er placeret midt mellem ledningerne, skulle denne vinkel gerne blive 45 grader. 0.7853981633974483 svarer til 45 grader.

```
calcAngle: 1.1071487177940904
calcAngle: 0.8278490601223093
calcAngle: 0.7853981633974483
calcAngle: 0.8655858853854237
calcAngle: 0.7853981633974483
```

Figur 22: Test af calcAngle

Test af *calcOffset* viser offsettet til de tre kabelmonteringspunkter, på venstre side af maseten. Figur 23a viser test med testdata 1, Figur 23b bruger testdata 4. Resultatet er x, y, z værdier til de tre kabelmonteringspunkter. z -værdierne ændre sig ikke, da det altid vil være i den samme højde. x og y -værdierne ændres da det er i forhold til vinklen på bærearmlængden.

calcOffsets, differences from center to the 3 cables, x y z coordinates:	calcOffsets, differences from center to the 3 cables, x y z coordinates:
-1.1180339887498951	1.6204852244839218
2.23606797749979	1.903687904365969
8.69	8.69
-1.7888543819998322	2.592776359174275
3.5777087639996634	3.04590064698555
17.71	17.71
-0.8944271909999161	1.2963881795871375
1.7888543819998317	1.522950323492775
26.69	26.69

(a) Forskelle mellem center og 3 kabler (b) Forskel mellem center og 3 master

Figure 23: Test af calcOffsets

Testen i Figur 24b viser højresiden af kabelkonteringspunkterne. Figur 24a viser venstresiden. z koordinaterne skulle være ens. x og y skulle være med modsat fortægn, da bæreammens kabelmonteringspunkter vil ligge i modsat retning i x og y retningerne. Testen bekræfter dette.

offsetsL, differences from center	offsetsR, differences from center
-1.1180339887498951	1.1180339887498951
2.23606797749979	-2.23606797749979
8.69	8.69
-1.7888543819998322	1.7888543819998322
3.5777087639996634	-3.5777087639996634
17.71	17.71
-0.8944271909999161	0.8944271909999161
1.7888543819998317	-1.7888543819998317
26.69	26.69

(a) Forskel mellem center og 3 kabler på venstre siden (b) Forskel mellem center og 3 kabler på højre siden

Figure 24: Test af calcOffsets

Den sidste test, er en test af hele programmet. Resultatet skulle give en liste, bestående af to lister, venstre og højresiden, og i hver af dem ligger 3 lister, med de tre kablers koordinater langs kablerne.

Testdata 1 er anvendt i testen som ses på Figur 25.

Figur 25 er en printvenlig version, så det er nemt at se resultaterne. Resultatet er som forventet, en liste af x , y og z koordinater, for de 6 kabler.

```

Left side cable 0:
x: 5.881966011250105 y: 6.23606797749979 z: 45.800000000000004
x: 6.776393202250021 y: 6.683281572999748 z: 39.89741184416382
x: 7.670820393249937 y: 7.130495168499706 z: 36.58523574377039
x: 8.565247584249853 y: 7.577708763999665 z: 35.86347169881971
x: 9.45967477524977 y: 8.024922359499623 z: 37.732119709311775
x: 10.354101966249686 y: 8.472135954999581 z: 42.191179775246596
x: 11.248529157249603 y: 8.91934955049954 z: 49.240651896624165
x: 11.881966011250105 y: 9.23606797749979 z: 55.80000000000001

Left side cable 1:
x: 5.211145618000168 y: 7.577708763999663 z: 36.78
x: 6.105572809000083 y: 8.024922359499621 z: 30.877411844163817
x: 6.9999999999999 y: 8.47213595499958 z: 27.565235743770387
x: 7.894427190999916 y: 8.919349550499538 z: 26.843471698819705
x: 8.788854381999833 y: 9.366563145999496 z: 28.71211970931177
x: 9.68328157299975 y: 9.813776741499455 z: 33.17117977524659
x: 10.577708763999667 y: 10.260990336999413 z: 40.22065189662416
x: 11.211145618000169 y: 10.577708763999663 z: 46.78000000000001

Left side cable 2:
x: 6.105572809000084 y: 5.7888543819998315 z: 27.8
x: 7.0 y: 6.23606797749979 z: 21.897411844163816
x: 7.894427190999916 y: 6.683281572999748 z: 18.585235743770387
x: 8.788854381999831 y: 7.130495168499706 z: 17.863471698819705
x: 9.683281572999748 y: 7.577708763999665 z: 19.73211970931177
x: 10.577708763999665 y: 8.024922359499623 z: 24.191179775246592
x: 11.472135954999581 y: 8.472135954999581 z: 31.24065189662416
x: 12.105572809000083 y: 8.788854381999831 z: 37.80000000000001

Right side cable 0:
x: 8.118033988749895 y: 1.7639320225002102 z: 45.800000000000004
x: 9.012461179749812 y: 2.2111456180001685 z: 39.89741184416382
x: 9.906888370749726 y: 2.658359213500127 z: 36.58523574377039
x: 10.801315561749643 y: 3.105572809000085 z: 35.86347169881971
x: 11.69574275274956 y: 3.5527864045000435 z: 37.732119709311775
x: 12.590169943749476 y: 4.00000000000002 z: 42.191179775246596
x: 13.484597134749393 y: 4.44721359549996 z: 49.24065189662416
x: 14.118033988749895 y: 4.76393202250021 z: 55.80000000000001

Right side cable 1:
x: 8.788854381999831 y: 0.4222912360003366 z: 36.78
x: 9.683281572999748 y: 0.8695048315002949 z: 30.877411844163817
x: 10.577708763999663 y: 1.3167184270002532 z: 27.565235743770387
x: 11.47213595499958 y: 1.7639320225002115 z: 26.843471698819705
x: 12.366563145999496 y: 2.21114561800017 z: 28.71211970931177
x: 13.260990336999413 y: 2.658359213500128 z: 33.17117977524659
x: 14.15541752799933 y: 3.1055728090000865 z: 40.22065189662416
x: 14.788854381999831 y: 3.4222912360003366 z: 46.78000000000001

Right side cable 2:
x: 7.894427190999916 y: 2.2111456180001685 z: 27.8
x: 8.788854381999831 y: 2.658359213500127 z: 21.897411844163816
x: 9.683281572999748 y: 3.105572809000085 z: 18.585235743770387
x: 10.577708763999665 y: 3.5527864045000435 z: 17.863471698819705
x: 11.472135954999581 y: 4.00000000000002 z: 19.73211970931177
x: 12.366563145999498 y: 4.44721359549996 z: 24.191179775246592
x: 13.260990336999415 y: 4.89442719099918 z: 31.24065189662416
x: 13.894427190999917 y: 5.2111456180001685 z: 37.80000000000001

```

Figur 25: Test af calcCable

7 Visualisering

7.1 Design

7.1.1 AirSim

AirSim er valgt som simuleringprogram, til at simulere dronens rute gennem et animeret 3d område. AirSim bruger Unreal Engine. Unreal Engine skal startes op for at kunne flyve med en drone. I Unreal Editor kan skabes eller ændres i det aktuelle område. Unreal Editor bliver brugt af mange til at lave baner og verdener til computer spil. Unreal er en af de største Engines, til computer spil i dag.

I dette projekt skal bruges højspændings master med dertilhørende kabler. Derfor skal disse tilføjes den ”verden”, som dronen skal flyve i. Der findes imidlertid ikke templates for højspændingsmaster eller kabler, derfor var det nødvendig at kreere disse.

7.1.2 Blender

Til at modellere højspændings master, skal der bruges et 3d modelerings program. I dette tilfælde Blender.

Blender blev valgt, da det er et gratis 3d modellerings program. Blender er et af de bedste, kun overgået af dyre programmer som Cinema4d og AutoCad. At modellere med Blender er ikke helt nemt. Det er svært at modellere efter specifikke mål og vinkler, da Blender mest er fokuseret på at lave skulptere og lignende.

Modellering med Blender foregår mest via ”shortcuts” på keyboardet. Det har derfor været en tidskrævende process at sætte sig ind i.

Blender fungerer ved at finde en model, der minder mest om det resultat det skal ende ud med. I dette tilfælde blev en kasse valgt. Det der kan modelleres på denne kasse er enten en ”edge”, vertex eller flader. En edge på start kassen er hjørnerne. En vertex er en linie mellem to edges, disse bliver brugt til at skære gennem flader, så de kan bukkes.

Når højspændingsmaster med kabler er modelleret, skal disse indsættes i det område som dronen er i, i Unreal.

7.2 Implementation

AirSim styres via et Python program. Dette Python program skal starte med at forbinde til klienten (AirSim). Dette ses i Listing 21 i linje 1 og 2. I linje 3 og 4 bliver dronen armeret, det vil sige at Python programmet får lov til at styre dronen.

Listing 21: start af AirSim

```
1 client = airsim.MultirotorClient()
2 client.confirmConnection()
3 client.enableApiControl(True)
4 client.armDisarm(True)
```

I listing 22 linje 2 ses en normal *move* funktion. Denne funktion tager værdierne x, y, z, v . Hvor v er velocity (hastigheden). I dette tilfælde vil dronen flyve med en hastighed på 5 km/t. z -værdien er angivet som negativ værdi, det vil sige at jo længere op dronen skal, desto større negativt værdi skal bruges.

I listing 22 får linje 1, dronen til at lette, så den svæver 1 meter over jorden. Linje 3 og 4 i Listing 22 specificerer at dronen skal flyve 10 meter op i luften og derefter skal programmet ”sove” i 6 sekunder. Disse linjer er egentlig overflødige og ikke noget der bliver brugt i projektet. Men slettes disse to linjer, drejer dronen 180 grader om sin x akse, altså den ville flyve på hovedet og så flyve ned igennem jorden. Så disse linjer bibeholdes. Det er kun i AirSim de to linjer skal bruges, ikke når det skal implementeres med en rigtig drone.

Listing 22: Takeoff AirSim

```
1 client.takeoffAsync().join()
2 client.moveToPositionAsync(0, 0, -10, 5).join()
3 time.sleep(6)
```

I Listing 23 positioneres dronen, til begyndelsen af det første kabelmonteringspunkt på det kabel der skal scannes.

De samme koordinater gentages lige efter. Hvis ikke det gøres *glider* dronen ind i kablerne i AirSim. Dronen i AirSim stopper ikke helt på alle dens akser i tide. Disse linjer er specifik for AirSim og skal undlades når det skal implementeres med en rigtig drone.

Listing 23: AirSim start position

```
1 client.moveToPositionAsync(float(sys.argv[1]), float(sys
   .argv[2]), -float(sys.argv[3]), 5).join()
```

I Listing 23 i linje 3 bruges en *move* funktion, hvor den henter værdierne fra *argv*, dette sker i Listing 24.

Da det er et java program, der skal starte et python program op, er det svært at videregive en liste fra java til python.

Det ville have været lettere at kode det hele i python. Men da det tog lang tid at få AirSim til at virke, og beslutningen om at kode i java blev truffet længe inden det virkede.

Python programmet tager her alle *x, y, z* værdier som *argv*, selvom denne liste af *argv* bliver utrolig lang.

For at loope rigtigt over alle disse *argv* input, indsættes det i et *while loop* som ses i linie 2 i Listing 24. While løkken starter fra 1, hvilket ses i linie 1, grundet til det er at selveprogrammet ligger på plads 0. While løkken øges med 3 hver gang, for at få de rigtige *x, y, z* værdier til at ligge på de rigtige pladser.

Listing 24: AirSim kabel løber

```
1 i = 1
2 while i < len(sys.argv):
3     client.moveToPositionAsync(float(sys.argv[i]), float
       (sys.argv[i+1]), -float(sys.argv[i+2]), 5).join()
4     i = i + 3
```

Efter de 6 kabler er blevet inspiceret, så holder dronen 6 sekunders pause, og derefter bliver dronen nulstillet. Det vil sige den lander det samme sted den lettede fra og kontrollen over dronen bliver frigivet. Dette er kun en visualisering og har ingen betydning program mæssigt.

Listing 25: AirSim afslutning

```
1 time.sleep(6)
2 client.armDisarm(False)
3 client.reset()
4 # that's enough fun for now. let's quit cleanly
5 client.enableApiControl(False)
```

For at det er muligt, at kunne køre et andet program i java programmet, bliver *Runtime exec* brugt. Dette gør at der skrives ud til commandolinjen og køres derefter. Dette kræver dog at det bliver puttet ind i en *try catch*.

For at få dronen til at flyve den ene vej på det første kabel og den anden vej på det andet kabel, bliver der brugt en if-statement med en *int*, der skifter alt efter hvilken vej dronen skal flyve. Dette kan ses i Listing 26 i linjerne 6 til 17. Inde i disse for-løkker bliver *cont* strengen opbygget, med alle *x, y, z* værdierne med mellemrum (*whitespace*) i mellem. På den måde bliver det klar til at sendes til Python programmet.

Listing 26: Main i java filen

```
1 try{
2     String cont= " ";
3     int forBack = 0;
4     for(int k = 0; k < coords.size(); k++){
5         for(int j = 0; j < coords.get(k).size(); j++){
6             if (forBack == 0){
7                 for(int i = 0; i < coords.get(k).get(j).
8                     size(); i+=3){
9                     ...
10                }
11                forBack = 1;
12            }else{
13                for(int i = coords.get(k).get(j).size()
14                    -1; i >2; i-=3){
15                    ...
16                }
17            }
18        }
19    }
20 }
```

I Listing 27 bliver de koordinater tilføjet, der skal sørge for at dronen kan flyve hen over tårnene, når kablerne på den anden side af tårnet skal scannes.

Disse koordinater kan ikke rigtigt tilføjes andre steder, da det ikke vides om der vil blive tilføjet flere koordinater, til listen inden dette punkt.

Listing 27: Main i java filen

```
1
2     if (k < coords.size()-1){
```

```
3         if (forBack == 1){
4             ...
5         }else{
6             ...
7         }
8     }
9 }
10 Runtime.getRuntime().exec("python droneflight2.py "
11     + cont);
12 System.exit(0);
13 }catch(IOException e){}
```

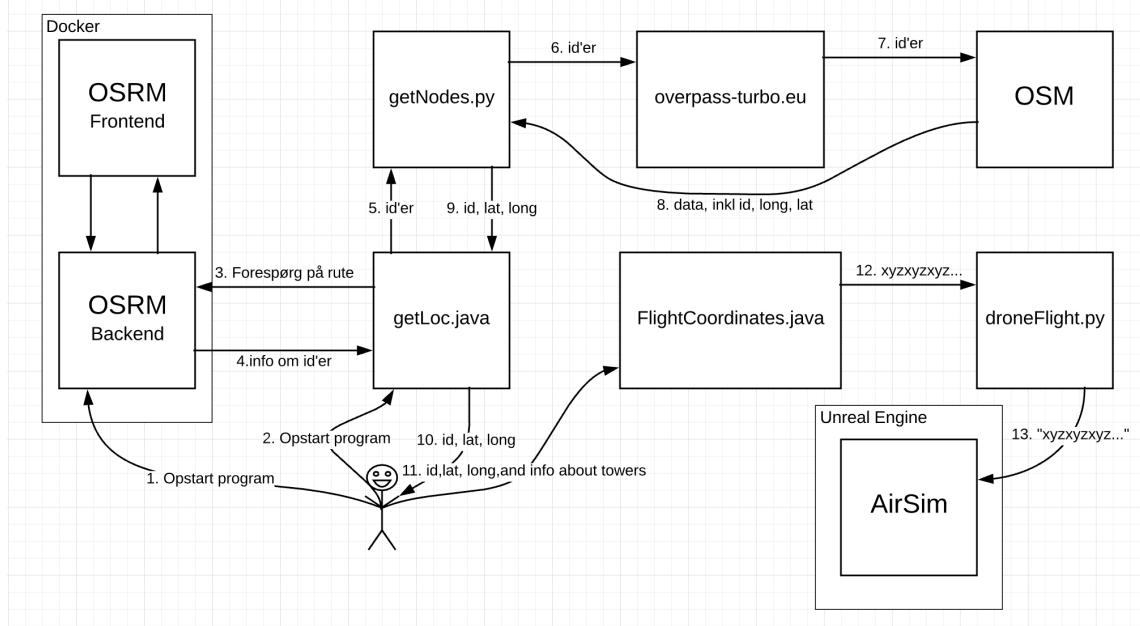
7.3 Test

Test af om dronen følger kablerne, kan ses i følgende video:

<https://youtu.be/gNSa1P36jq8>

I denne test flyver dronen med 5km/t langs kablerne. Tårn 1 står direkte på jorden, det vil sige at toppen af tårnet er 54.49 meter oppe. Tårn 2 står på en 10 meter høj blok dermed er dens højde 64.49 fra jorden af.

8 System Integration



Figur 26

Ovenstående figur visualiserer designet og kommunikationen i systemet. Det er ikke alle processer, der er automatiseret, der skal således udføres manuelle indtastninger. Dette er indikeret med ”tændstikmanden”.

Docker og OSRM opstartes som beskrevet i OSRM testafsnit (5.2.5). Derefter køres *getLoc.java*, der henter information om højspændingsmasternes *id'er* på den aktuelle rute. *getLoc.java* kører *getNodes.py* med de fundne *id'er*. Via *overpass-turbo.eu* hentes data på højspændingsmasterne i OSM. Med *getNodes.py* hentes information om masternes *lat* og *lon* og dette returneres til *getLoc.java*. *lat*, *lon* og masternes *id* printes til brugeren. *lat* og *lon* svarende til masternes *x* og *y* koordinater.

Ved manuelt at indtaste, *x*, *y* og *z* – *koordinater* for 3 på hinanden følgende master, samt *z* – *koordinaten* for lavpunktet mellem de sidste to master i *FlightCoordinates.java*, beregnes 3D koordinater til dronens flyrute.

z – koordinaterne for masterne, er masternes højde, eksempler på disse kan findes på Bilag 1 og 2. Eksempler på *z – koordinaten* for lavpunktet findes på Bilag 1. Ønskes at beregnes på andre typer af master end, de i projektet brugte (Bilag 2), skal listerne med afstande ud til kabelmonteringspunkterne ændres, i *zDiff* og *xyDiff* i *FlightCoordinates.java*.

FlightCoordinates.java kører filen *droneFlight.py* der opstarter AirSim i Unreal Engine, med *3D – koordinaterne*. AirSim simulerer dronens inspektion af højspændingskablerne.

9 Fremtidige udvidelser

I dette afsnit beskrives områder, der desværre ikke har været tid til at implementere i dette projekt.

Kabler udspændt mellem højspændingmaster udvider sig betragteligt i varmt vejr. Der kan være op til flere meter i forskel på længden, på kablerne fra den koldeste til varmeste dag. Dette har indvirkning på den bue kablet danner, det vil sige at lavpunktet ikke er stationært, som det antages i dette projekt.

Et fremtidigt projekt kunne derfor indholde, at dronen skal kunne hente vejr data. Disse data skal indgå i beregningen af lavpunktet af et kabel.

Vejr data kan også bruges til at stoppe dronen i at flyve i dårligt vejr. Hvis det for eksempel blæser for meget, vil det have indvirkning på dronens kurs.

Et andet fremtidigt projekt, kunne være at implementere at dronen kan flyve mellem kabler, der ikke hænger sammen, men er i umiddelbar nærhed af hinanden. Dette ville gøre, at dronen ikke skal ud på nogle unødig lange ruter, for at scanne et kabel, der løber langs med det sidst scannede. Dette problem står også beskrevet i Fejl og mangler i OSRM afsnittet.

Det er ikke implementeret i dette projekt at dronen skal oplade. Der skal laves en udvidelse til programmet, der gør dette muligt. Denne udvidelse er meget vigtig, inden dronen reelt set kan bruges. Denne udvidelse skal være i dronens program og ikke i dette.

Hvis dronen også skal ”sige til” når den skal lade op, i stedet for med et givet

tidsinterval, skal der indhentes data, om hvordan dronen giver besked om at den mangler strøm.

Når dronen skanner kablerne, flyver dronen ret tæt på meget kraftige højspændings kabler. Disse kan forstyrre dronens sensorer. Derfor kunne det på længere sigt måske implementeres, at i stedet for kun at lade dronen flyve efter x, y, z koordinater, så også få billedgenkendelse med til at styre dronen.

Det vil også være smart at udvide med at dronen ikke kun behøver at flyve ved højspændings kablerne, men at den også kan flyve lidt væk fra dem for at skanne andre ting end kun master og kabler.

10 Konklusion

Med programmet, *getLoc.java*, udviklet i dette projekt, er det muligt at finde information om en rute i Open Source Routing Machine. Med *id*'er for masterne i ruten, er det muligt, med *getNodes.py*, at finde længde- og breddegradskoordinater for disse master, via *overpass-turbo.eu* i OpenStreetMap. Indtastes denne information, sammen med information om masternes højde og afstande ud til kabelmonteringspunkterne, i *FlightCoordinates.java*, beregnes 3D koordinater langs kablerne mellem 2 master. Med disse koordinater er det muligt at få en drone til at flyve langs kablerne, og inspicere disse. Dette er i denne opgave visualiseret i AirSim.

Det er således muligt at få x, y og z -koordinater, der skulle gøre det muligt at få en drone til at flyve langs højspændingskabler, med henblik på inspektion af disse.

Det har været et projekt, hvor meget af tiden er brugt til at sætte sig ind i nye programmeringssprog og installere programmer.

11 Kilder

Brugt i afsnittene OpenStreetMap og Open Source Routing Machine:
Luxen, D., and C. Vetter. 2011. Real-time routing with OpenStreetMap data. In Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 513–516. New York: Association for Computing Machinery.

Haklay, M., Weber, P. (2008). Openstreetmap: User-generated street maps. IEEE Pervasive Computing, 7(4), 12-18

Om OpenStreetMap:

<https://www.openstreetmap.org/about> Tilgå information på OpenStreetMap via hjemmesiden:
<https://overpass-turbo.eu>

”Tag” information til OpenStreetMap:

<https://taginfo.openstreetmap.org>

Download kort over Danmark:

<https://download.geofabrik.de>

Open Source Routing Machine backend (til Ubuntu) hentes på hjemmesiden:
<https://github.com/Project-OSRM/osrm-backend>

Docker (til ubuntu) hentes via hjemmesiden:

<https://docs.docker.com/install/linux/docker-ce/ubuntu>

Brugt i afsnit med matematiske udregninger:

Adams og Essex (2014), Calculus Eighth Edition, Pearson
Matematisk Formelsamling, Højt Niveau, (1993), Undervisningsministeriet, Gymnasieafdelingen.

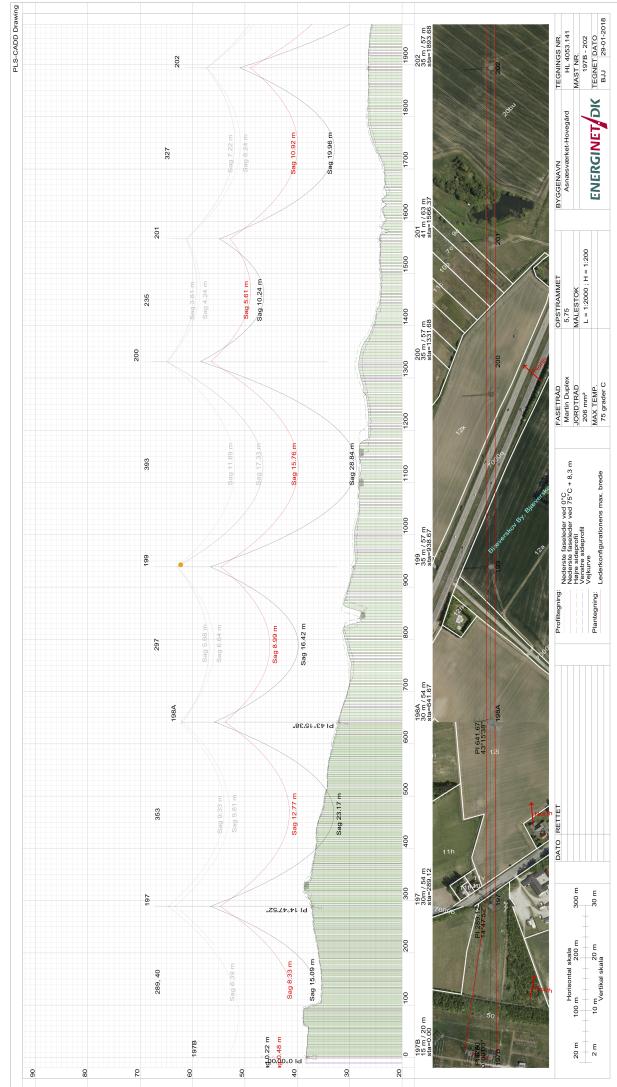
Regler for droneflyvning:

<https://www.trafikstyrelsen.dk/da/dronereggle>

<https://www.retsinformation.dk/Forms/R0710.aspx?id=1949>

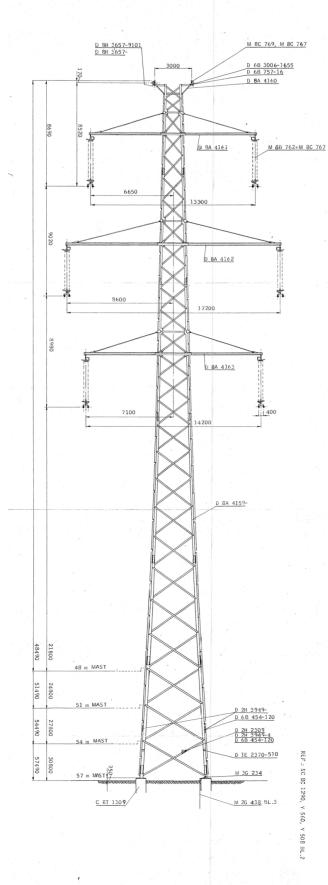
12 Bilag

Kilde: bilag 1: mail modtager fra Energinet Eltransmission A/S, Tonnekjærsvæj 65, 7000 Fredericia.



Bilag 1

Kilde Bilag 2: mail modtager fra Energinet Eltransmission A/S, Tonnekjærsvæj 65, 7000 Fredericia.



Bilag 2

13 Appendix

getLoc.java

Listing 28: getLoc.java

```
1 import java.io.*;
2
3 public class getLoc {
4
5     public static void main(String args[]) {
6         String s = null;
7         try{
8             double point1lon = 10.327102;
9             double point1lat = 55.346126;
10            double point2lon = 10.330245;
11            double point2lat =55.342476;
12            Process p0 = Runtime.getRuntime().exec("curl
13                 http://127.0.0.1:5000/route/v1/walking/"
14                 + point1lon + point1lat + point2lon +
15                 point2lat + "?steps=true&annotations=
16                 nodes");
17            BufferedReader stdInput0 = new
18                BufferedReader(new
19                    InputStreamReader(p0.getInputStream())));
20                ;
21
22            s = stdInput0.readLine();
23            //System.out.println(s);
24
25            int begining = s.indexOf("nodes");
26            int end = s.lastIndexOf("steps");
27            String nodes = s.substring(begining+8, end
28                -4);
29            nodes = nodes.replace(", ", " ");
30            //System.out.print(nodes);
31
32            Process p = Runtime.getRuntime().exec("python3
33                getNodes.py " + nodes);
34
35            BufferedReader stdInput = new BufferedReader
36                (new
```

```
28             InputStreamReader(p.getInputStream()));
29
30         while ((s = stdInput.readLine()) != null) {
31             System.out.println(s);
32         }
33         System.exit(0);
34     }catch(IOException e){}
35 }
36 }
```

getNodes.py

Listing 29: getNodes.py

```
1 import overpy
2 import sys
3 api = overpy.Overpass()
4
5 stringNodes = ""
6 for no in range(1, len(sys.argv)):
7     stringNodes = stringNodes + "node(" + str(sys.argv[
        no]) + ");out;"
8
9 result = api.query(stringNodes)
10
11 for no in range(1, len(sys.argv)):
12     print(str(sys.argv[no]))
13     print(result.nodes[no-1].lon, result.nodes[no-1].lat
        )
```

foot.lua

Listing 30: foot.lua

```
1 -- Foot profile
2
3 api_version = 2
4
5 Set = require('lib/set')
6 Sequence = require('lib/sequence')
7 Handlers = require("lib/way_handlers")
8 find_access_tag = require("lib/access").find_access_tag
9
```

```

10  function setup()
11      local walking_speed = 30
12      return {
13          properties = {
14              weight_name                 = 'duration',
15              max_speed_for_map_matching = 25/3.6, -- kmph ->
16              m/s
17              call_tagless_node_function = false,
18              traffic_light_penalty     = 0,
19              u_turn_penalty             = 0,
20              continue_straight_at_waypoint = false,
21              use_turn_restrictions     = false,
22          },
23          default_mode                = mode.walking,
24          default_speed               = walking_speed,
25          oneway_handling             = 'specific', -- respect
26          'oneway:foot' but not 'oneway'
27          barrier_blacklist = Set {
28
29      },
30
31      access_tag_whitelist = Set {
32          'tower',
33          'power'
34      },
35
36      access_tag_blacklist = Set {
37          'foot',
38          'highway'
39      },
40
41      restricted_access_tag_list = Set { },
42
43      restricted_highway_whitelist = Set { },
44
45      construction_whitelist = Set {},
46
47      access_tags_hierarchy = Sequence {
48          'tower',
49          'power'

```



```

89         platform          = walking_speed
90     } ,
91
92     amenity = {
93         parking          = walking_speed ,
94         parking_entrance= walking_speed
95     } ,
96
97     man_made = {
98         pier            = walking_speed
99     } ,
100
101    leisure = {
102        track           = walking_speed
103    }
104 },
105
106    route_speeds = {
107        ferry = 5
108    },
109
110    bridge_speeds = {
111    },
112
113    surface_speeds = {
114        fine_gravel = walking_speed ,
115        gravel = walking_speed ,
116        pebblestone = walking_speed ,
117        mud = walking_speed ,
118        sand = walking_speed
119    },
120
121    tracktype_speeds = {
122    },
123
124    smoothness_speeds = {
125    }
126 }
127 end
128
129 function process_node(profile, node, result)
130     -- parse access and barrier tags

```

```

131  local access = find_access_tag(node, profile.
132      access_tags_hierarchy)
133  if access then
134      if profile.access_tag_blacklist[access] then
135          result.barrier = true
136      end
137  else
138      local barrier = node:get_value_by_key("barrier")
139      if barrier then
140          -- make an exception for rising bollard barriers
141          local bollard = node:get_value_by_key("bollard")
142          local rising_bollard = bollard and "rising" ==
143              bollard
144
145          if profile.barrier_blacklist[barrier] and not
146              rising_bollard then
147              result.barrier = true
148          end
149      end
150
151      -- check if node is a traffic light
152      local tag = node:get_value_by_key("highway")
153      if "traffic_signals" == tag then
154          result.traffic_lights = true
155      end
156
157      -- main entry point for processsing a way
158      function process_way(profile, way, result)
159          -- the intial filtering of ways based on presence of
160          -- tags
161          -- affects processing times significantly, because all
162          -- ways
163          -- have to be checked.
164          -- to increase performance, prefetching and intial tag
165          -- check
166          -- is done in directly instead of via a handler.
167
168          -- in general we should try to abort as soon as
169          -- possible if the way is not routable, to avoid doing
170          -- unnecessary work. this implies we should check

```

```

        things that
167  -- commonly forbids access early, and handle edge
      cases later.

168
169  -- data table for storing intermediate values during
      processing
170  local data = {
171      tower = way:get_value_by_key('tower'),
172      power = way:get_value_by_key('power')
173  }
174
175  -- perform an quick initial check and abort if the way
      is
176  -- obviously not routable. here we require at least
      one
177  -- of the prefetched tags to be present, ie. the data
      table
178  -- cannot be empty
179  if next(data) == nil then      -- is the data table
      empty?
180      return
181  end
182
183  local handlers = Sequence {
184      -- set the default mode for this profile. if can be
          changed later
185      -- in case it turns we're e.g. on a ferry
186      WayHandlers.default_mode,
187
188      -- check various tags that could indicate that the
          way is not
189      -- routable. this includes things like status=
          impassable,
190      -- toll=yes and oneway=reversible
191      WayHandlers.blocked_ways,
192
193      -- determine access status by checking our hierarchy
          of
194      -- access tags, e.g: motorcar, motor_vehicle,
          vehicle
195      WayHandlers.access,
196

```

```

197      -- check whether forward/backward directons are
198      -- routable
199      WayHandlers.oneway,
200      -- check whether forward/backward directons are
201      -- routable
202      WayHandlers.destinations,
203      -- check whether we're using a special transport
204      -- mode
205      WayHandlers.ferries,
206      WayHandlers.movables,
207      -- compute speed taking into account way type,
208      -- maxspeed tags, etc.
209      WayHandlers.speed,
210      WayHandlers.surface,
211      -- handle turn lanes and road classification, used
212      -- for guidance
213      WayHandlers.classification,
214      -- handle various other flags
215      WayHandlers.roundabouts,
216      WayHandlers.startpoint,
217
218      -- set name, ref and pronunciation
219      WayHandlers.names,
220
221      -- set weight properties of the way
222      WayHandlers.weights
223  }
224
225  WayHandlers.run(profile, way, result, data, handlers)
226 end
227
228 function process_turn (profile, turn)
229   turn.duration = 0.
230
231   if turn.direction_modifier == direction_modifier.
232     u_turn then
233     turn.duration = turn.duration + profile.properties.

```

```

                u_turn_penalty
233    end
234
235    if turn.has_traffic_light then
236        turn.duration = profile.properties.
236        traffic_light_penalty
237    end
238    if profile.properties.weight_name == 'routability'
238    then
239        -- penalize turns from non-local access only
239        segments onto local access only tags
240    if not turn.source_restricted and turn.
240        target_restricted then
241        turn.weight = turn.weight + 3000
242    end
243    end
244 end
245
246 return {
247     setup = setup,
248     process_way = process_way,
249     process_node = process_node,
250     process_turn = process_turn
251 }

```

flightCoordinates.java

Listing 31: Main i java filen

```

1 import java.util.*;
2 import java.io.*;
3
4 public class flightCoordinates {
5
6     /*
7      * calcPoints calculates the x and y coordinates for
7      * line between two towers,
8      * with an interval of the length intervalDistance.
9      *
10     * intervalDistance is a length, for the interval
10     * between "inspectionpoints"
11     * point1x is tower 1's x coordinate
12     * point1y is tower 1's y coordinate

```

```

13     * point2x is tower 2's x coordinate
14     * point2y is tower 2's y coordinate
15     * point2z is tower 2's z coordinate
16     * point3x is tower 3's x coordinate
17     * point3y is tower 3's y coordinate
18     * point3z is tower 3's z coordinate
19     * lowpoint is the z coordinate for the lowpoint of
      the parabola between two towers
20     * zDiff is an ArrayList of doubles, with the z
      distances from center of the mast's to the cable
      attachment point
21     * xyDiff is an ArrayList of doubles, with the x, y
      distances from center of the mast's to the cable
      attachment point
22     */
23
24
25     public static ArrayList<ArrayList<ArrayList<Double
      >>> calcCable(double intervalDistance, double
      point1x, double point1y, double point2x, double
      point2y, double point2z, double point3x, double
      point3y, double point3z, double lowpoint,
      ArrayList<Double> zDiff, ArrayList<Double>
      xyDiff){
26         ArrayList<ArrayList<ArrayList<Double>>>
      XYZcoords = new ArrayList<ArrayList<
      ArrayList<Double>>>();
27         XYZcoords.add(new ArrayList());
28         XYZcoords.add(new ArrayList());
29
30         ArrayList<Double> centerXY = calcCenterPoints(
      intervalDistance, point2x, point2y, point3x,
      point3y);
31
32         ArrayList<Double> centerCoords = calcParabola(
      point2x, point2y, point2z, point3x, point3y,
      point3z, lowpoint, intervalDistance,
      centerXY);
33
34         ArrayList<Double> offsetsL = new ArrayList();
35         ArrayList<Double> offsetsR = new ArrayList();
36

```

```
37     offsetsL = calcOffsets(point1x, point1y,
38                             point2x, point2y, point3x, point3y, zDiff,
39                             xyDiff);
40     for(int i = 0; i < offsetsL.size() - 2; i += 3) {
41         offsetsR.add(-1 * offsetsL.get(i));
42         offsetsR.add(-1 * offsetsL.get(i + 1));
43         offsetsR.add(offsetsL.get(i + 2));
44     }
45
46     for(int i = 0; i < offsetsL.size(); i += 3) {
47         XYZcoords.get(0).add(offsetPoints(
48             centerCoords, offsetsL.get(i), offsetsL.
49             get(i + 1), offsetsL.get(i + 2)));
46         XYZcoords.get(1).add(offsetPoints(
47             centerCoords, offsetsR.get(i), offsetsR.
48             get(i + 1), offsetsR.get(i + 2)));
49     }
50
51     return XYZcoords;
52 }
53
54 /**
55 * calcCenterPoints calculates the center coordinates
56 * from one tower to another,
57 * with an interval of the length intervalDistance
58 * intervalDistance is a length, for the interval
59 * between "inspectionpoints"
60 * a_x is tower 2's x coordinate
61 * a_y is tower 2's y coordinate
62 * b_x is tower 3's x coordinate
63 * b_y is tower 3's y coordinate
64 *
65 */
66 public static ArrayList calcCenterPoints(double
67     intervalDistance, double a_x, double a_y, double
68     b_x, double b_y){
69     double angle = Math.atan(Math.abs(b_y - a_y) / Math.
70         abs(b_x - a_x));
71     double opposite = Math.sin(angle) *
72         intervalDistance;
73     double adjacent = Math.cos(angle) *
74         intervalDistance;
75     double x = a_x;
```

```

66     double y = a_y;
67
68     ArrayList<Double> Xycoords = new ArrayList();
69     Xycoords.add(a_x);
70     Xycoords.add(a_y);
71
72     if(a_x < b_x && a_y <= b_y){//up right and right
73         while (x+adjacent<=b_x && y+opposite <= b_y)
74             {
75                 x += adjacent;
76                 y += opposite;
77                 Xycoords.add(x);
78                 Xycoords.add(y);
79             }
80             //add last point
81             if(x < b_x || y < b_y){
82                 Xycoords.add(b_x);
83                 Xycoords.add(b_y);
84             }
85             }else if(a_x >= b_x && a_y < b_y){//up and up
86                 left
87                 while (x-adjacent >= b_x && y+opposite < b_y
88                     ) {
89                         x -= adjacent;
90                         y += opposite;
91                         Xycoords.add(x);
92                         Xycoords.add(y);
93                     }
94             }
95             }else if(a_x > b_x && a_y >= b_y){//left and
96                 down left
97                 while (x-adjacent > b_x && y-opposite >= b_y
98                     ) {
99                         x -= adjacent;
100                         y -= opposite;
101                         Xycoords.add(x);
102                         Xycoords.add(y);
103                     }
104             }

```

```

103             Xycoords.add(b_x);
104             Xycoords.add(b_y);
105         }
106     }else if(a_x <= b_x && a_y > b_y){//down and
107         down left
108         while (x+adjacent <= b_x && y-opposite > b_y
109             ) {
110             x += adjacent;
111             y -= opposite;
112             Xycoords.add(x);
113             Xycoords.add(y);
114         }
115         if(x < b_x || y > b_y){
116             Xycoords.add(b_x);
117             Xycoords.add(b_y);
118         }
119     }
120
121     /* *
122      * calcOffsets calculates the offset to the cable
123      * mounting points
124      *
125      * point1x is tower 1's x coordinate
126      * point1y is tower 1's y coordinate
127      * point2x is tower 2's x coordinate
128      * point2y is tower 2's y coordinate
129      * point3x is tower 3's x coordinate
130      * point3y is tower 3's y coordinate
131      *
132      * zDiff is an ArrayList of doubles, with the z
133      * distances from center of the mast's to the cable
134      * attachment point
135      * */
136
137     public static ArrayList<Double> calcOffsets(double
138             Mast_A_x, double Mast_A_y, double Mast_B_x,
139             double Mast_B_y, double Mast_C_x, double Mast_C_y

```

```

, ArrayList<Double> zDiff, ArrayList<Double>
xyDiff){
136     ArrayList<Double> listOfDist = new ArrayList<>()
;
137
138     double angleV = calcAngle(Mast_A_x, Mast_A_y,
139                               Mast_B_x, Mast_B_y, Mast_C_x, Mast_C_y);
140
141     if(Mast_B_x < Mast_C_x && Mast_B_y <= Mast_C_y){
142         //up right and right
143         for(int i = 0; i < xyDiff.size(); i++){
144             listOfDist.add(-Math.cos(angleV)*xyDiff.
145                           get(i));
146             listOfDist.add(Math.sin(angleV)*xyDiff.
147                           get(i));
148             listOfDist.add(zDiff.get(i));
149         }
150     }else if(Mast_B_x >= Mast_C_x && Mast_B_y <
151             Mast_C_y){//up and up left
152         for(int i = 0; i < xyDiff.size(); i++){
153             listOfDist.add(-Math.cos(angleV)*xyDiff.
154                           get(i));
155             listOfDist.add(-Math.sin(angleV)*xyDiff.
156                           get(i));
157             listOfDist.add(zDiff.get(i));
158         }
159     }else if(Mast_B_x <= Mast_C_x && Mast_B_y >
160             Mast_C_y){//down and down left
161         for(int i = 0; i < xyDiff.size(); i++){
162             listOfDist.add(Math.cos(angleV)*xyDiff.
163                           get(i));
164             listOfDist.add(Math.sin(angleV)*xyDiff.
165                           get(i));

```

```

162             list0fDist.add(zDiff.get(i));
163         }
164     }
165
166     return list0fDist;
167 }
168
169 /**
170 * calcAngle calculates the angle of the supporting
171 * beam
172 *
173 * point1x is tower 1's x coordinate
174 * point1y is tower 1's y coordinate
175 * point2x is tower 2's x coordinate
176 * point2y is tower 2's y coordinate
177 * point3x is tower 3's x coordinate
178 * point3y is tower 3's y coordinate
179 */
180 public static double calcAngle(double point1x,
181                               double point1y, double point2x, double point2y,
182                               double point3x, double point3y){
183
184     double a1, a2, v;
185     a1 = Math.abs(point1y-point2y)/Math.abs(point1x-
186         point2x);
187     a2 = Math.abs(point2y-point3y)/Math.abs(point2x-
188         point3x);
189
190     v = (Math.PI*2 - (Math.PI - Math.abs(Math.atan(
191         a1) - Math.atan(a2))))/2-(Math.atan(Math.abs(
192         point1y-point2y)/Math.abs(point1x-point2x)));
193
194     return v;
195 }
196
197 /**
198 * offsetsPoints findes the offsetpoints
199 *
200 * points is the liste of centerCoords
201 * x is the x coordinate from the offsetslist
202 * y is the y coordinate from the offsetslist

```

```

197     * z is the z coordinate from the offsetslist
198     *
199     * */
200     public static ArrayList offsetPoints(ArrayList<
201         Double> points, double x, double y, double z){
202         ArrayList<Double> offsetPointList = new
203             ArrayList<Double>();
204         for(int i = 0; i < points.size(); i+=3){
205             offsetPointList.add(points.get(i) + x);
206             offsetPointList.add(points.get(i+1) + y);
207             offsetPointList.add(points.get(i+2) - z);
208         }
209         return offsetPointList;
210     }
211     /**
212      * calcParabola first calculates the x value of the
213      * lowpoint of the hanging cable
214      * then uses intervalDistance as the interval to
215      * calculate the z values along the power lines.
216      *
217      * Mast_A_x first mast x value
218      * Mast_A_y first mast y value
219      * Mast_B_x second mast x value
220      * Mast_B_y second mast y value
221      * lowPoint_y is the hanging depth of the cable
222      * intervalDistance is an interval with a length
223      *
224      * */
225     public static ArrayList<Double> calcParabola(double
226         piont2x, double piont2y, double piont2z, double
227         piont3x, double piont3y, double piont3z, double
228         lowPoint_z, double intervalDistance, ArrayList<
229             Double> XYcoords){
230
231         double z, distance, temp_a1, temp_a2, a, b, c;
232         distance = Math.sqrt(Math.pow((piont3x-piont2x),
233             2)+Math.pow((piont3y-piont2y), 2));
234
235         temp_a1 = (piont2z-2*lowPoint_z+piont3z);
236         temp_a2 = -(lowPoint_z-piont3z)*(-lowPoint_z+
237             piont2z);

```

```

229     a = (temp_a1+2*Math.sqrt(temp_a2))/Math.pow(
230         distance, 2);
230     b = -(a*Math.pow(distance, 2)+piont2z-piont3z)/(
231         distance);
231     c = piont2z;
232
233     ArrayList<Double> XYZcoords = new ArrayList<
234         Double>();
234     double x;
235     int adding= 0;
236     for(x = 0.0; x<distance; x+=intervalDistance){
237         XYZcoords.add(XYcoords.get(adding));
238         XYZcoords.add(XYcoords.get(adding+1));
239         XYZcoords.add(z = a*(Math.pow(x, 2))+b*x+c);
240         adding += 2;
241     }
242
243     x = distance;
244     XYZcoords.add(XYcoords.get(adding));
245     XYZcoords.add(XYcoords.get(adding+1));
246     XYZcoords.add(z = a*(Math.pow(x, 2))+b*x+c);
247
248     return XYZcoords;
249 }
250
251 /**
252 * main
253 *
254 *
255 */
256 public static void main(String[] args) throws
257     InterruptedException{
258
258     ArrayList<Double> zDiff = new ArrayList();
259     zDiff.add(8.69);
260     zDiff.add(8.69+9.02);
261     zDiff.add(8.69+9.02+8.98);
262
263
264     ArrayList<Double> xyDiff = new ArrayList();
265     xyDiff.add(5+6.65);
266     xyDiff.add(5+8.6);

```

```

267     xyDiff.add(5+7.1);
268
269     double intervalDistance = 1;
270     double point1x = 13.0;
271     double point1y = 3.0;
272     double point2x = 10.0;
273     double point2y = 7.0;
274     double point2z = 64.49;
275     double point3x = 2.0;
276     double point3y = 12.0;
277     double point3z = 54.49;
278     double lowpoint = 44.49;
279
280     ArrayList<ArrayList<ArrayList<Double>>> coords =
281         calcCable(intervalDistance, point1x, point1y
282             , point2x, point2y, point2z, point3x, point3y
283             , point3z, lowpoint, zDiff, xyDiff);
284
285     try{
286         String cont= "";
287         int forBack = 0;
288         System.out.println("k: " + coords.size());
289         for(int k = 0; k < coords.size(); k++){
290             for(int j = 0; j < coords.get(k).size();
291                 j++){
292                 if (forBack == 0){
293                     for(int i = 0; i < coords.get(k)
294                         .get(j).size(); i+=3){
295                         System.out.println("x: " +
296                             coords.get(0).get(j).get(
297                                 i)+ " y: " + coords.get(k)
298                                 .get(j).get(i+1)+ " z: "
299                                 + coords.get(k).get(j).
300                                     get(i+2));
301                         cont = cont + coords.get(k).
302                             get(j).get(i)+ " " +
303                             coords.get(k).get(j).get(
304                                 i+1)+ " " + coords.get(k)
305                                 .get(j).get(i+2) + " ";
306                     }
307                     forBack = 1;
308                 }
309             }
310         }
311     }
312 }
```

```

295     }else{
296         for(int i = coords.get(k).get(j)
297             .size()-1; i >2; i-=3){
298             cont = cont + coords.get(k).
299                 get(j).get(i-2)+ " " +
300                 coords.get(k).get(j).get(
301                     i-1)+ " " + coords.get(k)
302                     .get(j).get(i) + " ";
303         }
304     }
305     if (k < coords.size()-1){
306         if (forBack == 1){
307             cont = cont + coords.get(k).get
308                 (0).get(coords.get(k).get(0).
309                     size()-3)+ " " + coords.get(k)
310                     .get(0).get(coords.get(k).
311                         get(0).size()-2) + " " +
312                         (point3z+10) + " ";
313             cont = cont + (coords.get(k+1).
314                 get(0).get(coords.get(k+1).
315                     get(0).size()-3))+ " " +
316                     coords.get(k+1).get(0).get(
317                         coords.get(k+1).get(0).size()
318                             -2) + " " + (point3z+10) + "
319                         ";
320         }else{
321             cont = cont + coords.get(k).get
322                 (0).get(0)+ " " + coords.get(
323                     k).get(0).get(1) + " " +
324                         (point2z+10) + " ";
325             cont = cont + coords.get(k+1).
326                 get(0).get(0)+ " " + coords.
327                     get(k+1).get(0).get(1) + " "
328                     + (point2z+10) + " ";
329         }
330     }
331 }
332
333 Runtime.getRuntime().exec("python
334     droneflight2.py " + cont);

```

```
314         System.exit(0);
315     }catch(IOException e){}
316 }
317 }
```

droneflight.py

Listing 32: droneflight.py

```
1 import setup_path
2 import airsim
3 import sys
4 import numpy as np
5 import os
6 import tempfile
7 import pprint
8 import time
9
10 # connect to the AirSim simulator
11 client = airsim.MultirotorClient()
12 client.confirmConnection()
13 client.enableApiControl(True)
14 client.armDisarm(True)
15
16 client.takeoffAsync().join()
17 client.moveToPositionAsync(0, 0, -10, 5).join()
18 time.sleep(6)
19 client.moveToPositionAsync(float(sys.argv[1]), float(sys
    .argv[2]), -float(sys.argv[3]), 5).join()
20 i = 1
21 while i < len(sys.argv):
22     client.moveToPositionAsync(float(sys.argv[i]), float
        (sys.argv[i+1]), -float(sys.argv[i+2]), 5).join()
23     i = i + 3
24
25 time.sleep(6)
26
27 client.armDisarm(False)
28 client.reset()
29
30 # that's enough fun for now. let's quit cleanly
31 client.enableApiControl(False)
```
