ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Ingegneria dell'Energia Elettrica e dell'informazione "Guglielmo Marconi"

Corso di Laurea Triennale in Ingegneria dell'Automazione

TESI DI LAUREA

in
Automatic Controls T-2

Agricultural Field Detection and Coverage Path Planning for an Unmanned Aerial Vehicle

CANDIDATO
Michael Rimondi

RELATORE:
Prof. Lorenzo Marconi

CORRELATORE:
Dott. Nicola Mimmo

Anno Accademico 2017/18

Sessione II

Abstract

Environment representation and path planning are two major issues in any navigation systems. This thesis discuss those issues in the specific use case regarding search and rescuing operation over agricultural area.

First, it is analyzed the process of image orthoretification and KML (Keyhole Markup Language) file format for representing geographical features, i.e the field boundary. Orthorectification is the process of correcting geometrical distortions of an image such that the scale is uniform and it can be use to obtain environment-specific information which are later encoded in KML.

Successively, the focus moves on Coverage Path Planning (CPP) that is the operation of finding a path covering all the points of a specific area. This task is fundamental to many robotic applications such as cleaning, painting, mine sweeping, monitoring, searching and rescue operations. The proposed approach is implemented in Robot Operating System (ROS) and it is based on approximate cellular decomposition of the environment. The coverage path is calculated using distance transform which propagates around the goal point in a wavefront. The algorithm could be adapted to optimize different path features (e.g. number of turns, path length), an important aspect especially in UAV application where the limited battery life requires the trajectory to be as much efficient as possible.

The results obtained in a simulated environment are satisfactory and even if the outcome path is not always optimal, it shows the versatility of the algorithm when applied to different field geometries.

Keywords: Georeferencing, Orthophoto, Coverage Path Planning (CPP), Approximate Cellular Decomposition, Unmanned Air Vehicles (UAV), copters

Abstract (italiano)

La rappresentazione dell'area di lavoro e la pianificazione del percorso sono due aspetti importanti in qualsiasi sistema di navigazione. Questa tesi si propone di discutere tali questioni nello specifico ambito delle operazioni di ricerca e salvataggio della fauna selvatica su suolo agricolo.

Inizialmente, viene analizzato il processo di ortorettificazione di immagini e presentato KML (Keyhole Markup Language) quale formato standard di rappresentazione di dati geografici. L'ortorettificazione consiste nel correggere distorsioni geometriche dell'immagine così da rendere la scala uniforme e usarla per ottenere informazioni specifiche sul campo di lavoro da codificare successivamente in KML.

Proseguendo, l'attenzione si sposta sulla Pianificazione del Percorso di Copertura (CPP), ovvero su quell'operazione che consente di trovare il percorso che copra tutti i punti di un'area designata e che per tale aspetto risulta fondamentale in differnti applicazioni in robotica quali pulizia, pittura, disinnesco mine, monitoraggio, e operazioni di ricerca e salvataggio. L'approccio proposto è implementato su ROS (Robot Operating System) e si basa sulla decomposizione approssimata dell'area in celle. Il tracciato di copertura è calcolato tramite una trasformazione della distanza che si espande dal punto d'arrivo in un fronte d'onda. L'algoritmo può essere adattato per ottimizzare diverse caratteristiche della traiettoria generata (e.g. numero di curve, lunghezza del percorso), aspetto importante specialmente per applicazioni riguardanti UAV dove la limitata durata della batteria necessita che la traiettoria sia la più efficiente possibile.

I risulatati ottenuti in ambiente simulato sono soddisfacenti e, nonostante il percorso generato non sia sempre quello ottimo, diversi test svolti su campi di forma e dimensione diversi dimostrano la versatilità dell'algoritmo.

Keywords: Georeferenziazione, Ortofotografia, Pianificazione Percorso di Copertura (CPP), Aeromobile a Pilotaggio Remoto (APR)

Contents

Acronyms

CPP Coverage Path Planning. 5, 19, 36, 37, 41, 42

DEM Digital Elevation Model. 8

GPS Global Positioning System. 3, 8

KML Keyhole Markup Language. 5, 10–13, 16, 41

MAVLink Micro Air Vehicle Link. 53, 54

QGC QGroundControl. 35, 36, 54

QML Qt Modeling Language. 36

ROS Robot Operating System. 4, 13, 16, 17, 19, 27, 35, 41

SITL Software-In-The-Loop. 35

UAV Unmanned Aerial Vehicle. iii, v, 1, 3, 4, 35, 36, 38, 39, 41

UTM Universal Transverse of Mercator. 29

WGS84 World Geodetic System of 1984. 8, 9, 29

XML eXtensible Markup Language. 10, 16, 54

Chapter I

Introduction

In this introduction it is first briefly described the motivation, objective and scope of the project this thesis has been developed in. Once the background has been well set and explained, this written work will elaborate in details the specific subject of *georeferencing an agricultural field* and *Coverage Path Planning*.

1 Bambi Project

Bambi Project aims to make the coverage flight over agricultural fields completely automatic for the purpose of wildlife rescuing.

1.1 Motivation

Deers gives birth to their offspring during April and May [1], often choosing meadows as they consider it a safe spot. This period is unfortunately the same in which meadows are cut. The results is that every year a great number of young deers fall victim of combine harvesters cutting hay. Germany counts about 100000 death every harvest season [1].

It is, as a matter of fact, difficult to locate small animals hidden in vast grasslands especially if they freeze when they feel under threat. For this specific reason the proposal design is based on an Unmanned Aerial Vehicle (UAV)[2] and more precisely a quad-copter equipped with a thermal camera. An aerial vehicle is able to efficiently cover large surfaces way faster then any other ground alternatives and guarantees the best viewpoint for the specific kind of wildlife research. Moreover, it has been chosen a copter rather than a fixed wing model for its holonomic properties which turns out to be very useful in upland regions as well as for small and irregular fields. It is basically why in search and rescue operations helicopters are often adopted instead of planes.

1.2 Importance for Wildlife and for Agricultural

In early hunting literature from as far back as the mid-19th century, references can be found to significant losses of breeding partridges and pheasants from the use of sickle and scythe. Due to the fierce competition in the agricultural sector, developments in agricultural technology have brought about a tremendous acceleration in mowing techniques, with tendency still rising. Today, mowing speed can even exceed 15 km/h, while at the same

time ever-wider mowers are used. Nesting birds, young hares and fawns are regular victims of such mowers and even full-grown wild animals cannot always escape.

1.2.1 Affected Species

Grassland is used by countless species of wildlife as food, cover and reproduction habitat. Apart from leverets, fawns and various field birds, small mammals, amphibians and insects all fall victim to the practice of early and more frequent mowing. Formerly reliable survival strategies proven successful over thousands of years have a devastating effect in mowing situations. The instinct of the brooding partridge hen to sit tight on her nest, or of the hare or fawn to freeze motionless, now prove fatal. The optimized patterns of predator avoidance behavior which wild animals have evolved can no longer keep up with the developments in modern cultivation techniques.

Figure I.1: Fawn hiding in meadow

1.3 Measure to Reduce Wildlife Losses

The most important factor influencing wildlife mortality is without doubt the time of mowing. On the other hand, economic considerations make this a crucial factor for the farmer, too. A late mowing is good for wildlife but not ideal for the farmer from the point of view of yield and quality. Yet there are some other factors in the mowing of arable land which offer potential for reducing wildlife losses [1]:

- *Cutting height*: the higher the cut, the lower will be the losses suffered by crouching animals and nesting birds.
- *Mowing direction*: mowing the field from the center outwards gives fully-grown wild animals the opportunity to escape.
- *Mowing date*: late cuts,from mid-July onwards, reduce the losses to wildlife during the nesting and rearing period.
- *Mowing strategy*: mowing of partial areas, leaving edge strips unmown.
- *Mowing frequency*: a longer interval between first and second cuttings reduces the mortality rate, especially for ground-nesting birds.

- *Mowing technology*: cutter-bar mowers cause less harm to wild animals than rotary mowers.

A practical approach which do not impact mowing strategy is the one adopted in a German wildlife rescue project consisting in deploying small aerial drones to find young deers hiding in tall grass.

2 UAV Usage for Wildlife Research

During last years, the increasing interest and development of Unmanned Aerial Vehicle (UAV) makes them affordable and suitable for many different applications. Specifically, in wildlife research, drones are frequently use as they are less expensive, quieter, and safer than traditional manned aircraft. Most studies recorded minimal or no visible behavioral responses to UAV; however, UAV are capable of causing behavioral and physiological responses in wildlife when observing at close range. [3].

For the case under-study, the "Flying Wildlife Finder", represents the state-of-the-art. The project, developed by the German Aerospace Center (DLR), is an application system which prevents accidents by detecting animals hidden in tall grass during the hay harvest. The "Flying Wildlife Finder", a remotely controlled aerial drone equipped with sensors and a GPS link, is sent on a reconnaissance flight before mowing starts. A high resolution thermal imaging camera detects the temperature of animals hidden in the grass, which is higher than the ambient temperature of the field. Once the animal is located a search party is led to the fawn's resting place with the help of GPS. This solution proves to be good as it is less time-consuming then using trained dog, while maintaining even higher "hit-rate", but it has the main drawback of requiring at least two specialists: one pilot and one camera operator that must be focused on the thermal video stream during the whole mission duration.

3 Proposal Solution

Bambi project consists in a standard UAV equipped with a thermal camera and an onboard computer. The mission is carried out by an operator and an assistant who is entitled to rescue the animal once it has been located. The vehicle automatically scan the entire field and inform the operator when thermal irregularity is detected sending the GPS position. At this point the rescuing team is guided toward the animal with the help of the thermal video signal.
This procedure requires that the vehicle passes autonoumusly through the following steps:

1. Takeoff and flight to a position where having a complete view of the field so to shout a photo of the whole field.

2. Generate a georeferenced photo of the field, identify the field boundary and store it as a geographic polygon (set of GPS coordinates).

3. Calculate the *coverage path*.

4. Generate a *timed dependent trajectory* taking into account dynamic constraints of the vehicle.

5. Make the drone to follow the computed trajectory while avoiding obstacles.

6. Interrupt the flight when the thermal camera sense a spot having an higher temperature with respect to the field and guide the user to the target location.

7. When the field has been completely covered return to home position.

3.1 Project Goals

The scope of the Bambi project is to ease a lot wildlife rescuing using UAV so that it can be handled also by untrained people. This must be guaranteed even in adverse environments, especially for uneven fields as it is in upland regions. Along with this, it is of particular interest to keep modularity in every hardware and software components. This is to permit easier implementation of new features or improvements to the existing ones. All this basic concepts have been used as guidelines during the design and implementation stages (e.g. the choice of ROS as framework for the whole software).

3.2 Software Architecture

The raspberry Pi 3 it is used as on-board computer, it runs ROS (see appendix A) and manages the transition between the different mission's phases. The software develops according to the usual ROS philosophy [4], thus it is composed of six nodes implementing every mission component as an independent module. Figure I.2 displays all the nodes and their communication scheme. The mission_controller node, in the center of the graph, handle all the mission procedures and timing implementing a FSM (*Final State Machine*).

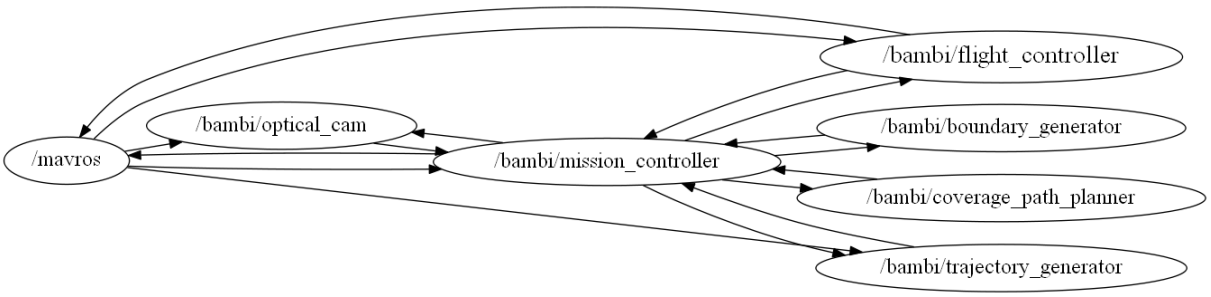


Figure I.2: Nodes and Topics graph

In the following chapter the thesis will examine in particular two parts of the project:

- *Geo-referencing the mission's environment and get the field boundary.* This is of great importance, as it define the mission's range which is fundamental for every successive tasks.

- *Planning the coverage path.* It regards planning the geometric path so that the whole field of interest is covered by the sensor footprint.

4 Geo-referencing Mission's Environment

Chapter II develops the subject of Georeferencing the mission's environment dividing it into two different tasks:

1. Get an orthorectified photo of the workplace, thus an uniform scaled pictures of the field with the same lack of distortion as a map and having geo-position information encoded into it.

2. Detect the field border and store it in a standard format that is commonly used to encode geographic features.

In first place, it is explained what is orthorectification and the process to achieve it. Then free orthophoto database are presented with a special care for Google Imagery database and Google Earth application. Google Earth provides a toolbox to trace path over the map. This is used to define the field boundary easily directly on the orthophoto and later to save it in Keyhole Markup Language (KML). The main features of KML is therefore illustrated as well as how to handle it in Python scripts.

5 Coverage Path Planning

In chapter III the topic of Coverage Path Planning (CPP) is explained providing a basic knowledge of the existent approaches to the problem.

Coverage Path Planning is the task of determining a path that passes over all points of an area or volume of interest while avoiding obstacles. This task is integral to many robotic applications, such as vacuum cleaning robots, painter robots, autonomous underwater vehicles creating image mosaics, demining robots, lawn mowers, automated harvesters, window cleaners and inspection of complex structures. That is why it is an important problem in *mobile robot navigation* and even if the research about this issue developed rapidly, the governing algorithm to efficiently find the optimal path has not been found yet.

Chapter II

Georeferencing the mission's environment

In this chapter it is explained in details how the device obtain the information of the field boundary in form of a list of geographic coordinates. This important task will define the mission range of action which is used as input to the Coverage Path Planning module. It is thus important that the gathered information are rather precise and consistent or it will greatly impact the successive steps of the mission.
The process could be divided in two tasks:

- Obtain a georeferenced photo displaying the entire meadow.
- Detect, trace and store the contour of the field from the georeferenced photo.

1 Georeferencing a Photo

Before an aerial image can be used to support a site-specific application it is essential to perform the geometric corrections and geocoding. This is commonly called *georeferencing* which enables the assignment of ground coordinates to the different features in the datasets. If the map projection [1] (and map projection parameters) of the ground coordinates are known, equivalent geographic coordinates can be produced which enables positioning the features of the coverage into a World context. [6]. For this specific use case the final result is an orthophoto.

1.1 Theory Background

"An orthophoto or orthoimage is an aerial photograph or image geometrically corrected ("orthorectified") such that the scale is uniform: the photo has the same lack of distortion as a map."[7]

Unlike an uncorrected aerial photograph, an orthoimage can be used to measure true distances, because it is an accurate representation of the Earth's surface, having been adjusted for topographic relief, lens distortion, and camera tilt. After the photo has been

[1] A map projection is a way to represent the curved surface of the Earth on the flat surface of a map [5]

orthorectified it is georeferenced by transforming the local reference frame of the photo to the desired geographic coordinate system.

Among all the existing coordinate system the common WGS84 has been chosen as it is the standard in use by GPS (Global Positioning System). It consists of a three-dimensional Cartesian coordinate system and an associated ellipsoid, therefore WGS84 positions can be described as either XYZ Cartesian coordinates or latitude, longitude and ellipsoid height coordinates. The origin is the Geocentre (the centre of mass of the Earth), it is designed for positioning anywhere on Earth and as any other geographic coordinate system uses decimal degree as unit of measurement.

Spatial datasets, like any type of data, are prone to errors. Thus, three fundamental concepts have to be kept in mind: precision, bias and accuracy. Precision refers to the dispersion of positional random errors and it is usually expressed by a standard deviation. Bias, on the other hand, is associated with systematic errors and is usually measured by an average error that ideally should equal zero. Accuracy depends on both precision and bias and defines how close features on the map are from their true positions on the ground [8]. So, despite being frequently confused concepts, high precision does not necessarily mean high accuracy. But both depend greatly on the map scale. All maps have inherent positional errors, which depend on the methods used in the construction of the map. The scale is the ratio between a distance on the map and the corresponding distance on the ground. The maximum acceptable positional error (established by cartographic standards) is determined by the map scale.

1.2 Ortho-rectification

The goal of the ortho-rectification is to resample the pixels of an aerial photo to a coordinate system that is interpreted in a selected horizontal surface. In georeferencing an aerial photo two distinct distortion effect should be corrected:

- The perspective distortion, which is the result of the geometry of the photograph taking.
- The distortion effect of the relief and/or the surface.

The first one is addressed through camera calibration, which involves finding the focal length of the camera, principal point coordinates and lens distortion of each photo. The second distortion effect can be corrected exploiting Digital Elevation Model (DEM[2]) as shown in Figure II.1. The overall quality of the orthorectified image is therefore directly related to how accurate is the camera model and to the fidelity of the DEM. For this practical case the accuracy requirements of the georeferenced image is not so tight and so it has been decided, at first place, to use the ready-to-use Google Earth imagery database. Anyway in designing the mission work-flow it was predisposed the possibility to take an aerial photo of the entire site to georeference it.

1.3 Google Earth

"Google Earth is a virtual globe, map and geographical information program that was originally called Earth Viewer 3D, and was created by Keyhole, Inc, a Central Intelligence

[2]The DEM is a raster of terrain elevations

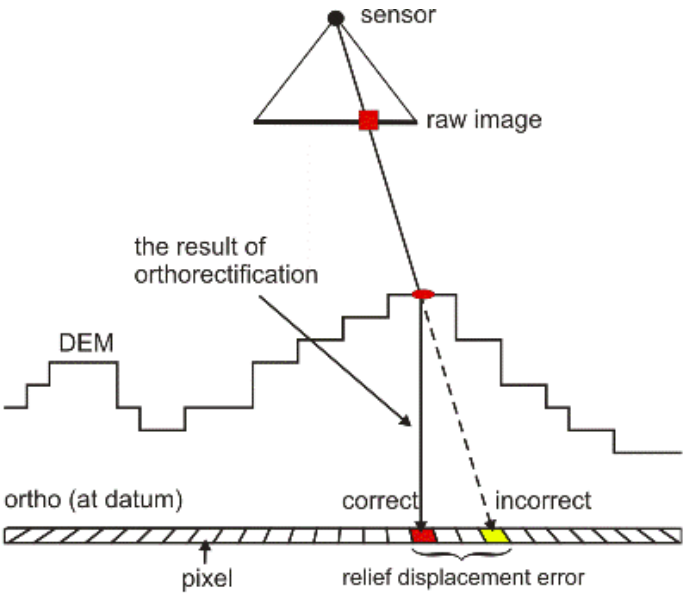


Figure II.1: Orthorectification

Agency (CIA) funded company acquired by Google in 2004. It maps the Earth by the superimposition of images obtained from satellite imagery, aerial photography and GIS 3D globe. Google Earth uses digital elevation model (DEM) data collected by NASA's Shuttle Radar Topography Mission (SRTM). The internal coordinate system of Google Earth is geographic coordinates (latitude/longitude) on the World Geodetic System of 1984 (WGS84) datum i.e., the same datum that used by GPS." [5].

Google Earth shows the earth surface as it looks from an elevated platform such as an airplane or orbiting satellite. The projection used to achieve this effect is called the General Perspective. This is similar to the Orthographic projection. Most of the high resolution imagery in Google Earth Maps is the Digital Globe Quickbird which is roughly 65cm pan sharpened. Google is actively replacing this base imagery with 2.5 m SPOT Image imagery and several higher resolution datasets.

The application is multi-platform and has an user friendly interface which among other provides tools to trace geographic feature such as shape, polygons and path. This turns out to be an essential feature in representing the field border which is after all, the final aim of this software component.

The position accuracy of Google Earth was analyzed by Ahmed Ghazi whose selected 16 points in Khartoum town. As of October 2012 The root mean square of the error results to be about $1.80m$ for horizontal position and $1.773m$ for height estimation.[5] These results, while referring to only a specific region of the earth, are representative of the goodness of the data, especially considering they are available for free.

2 Border Representation and Detection

In order to obtain a digital representation of the border it is, first of all, important to define how to store the data in a file. Google Earth file format is the Keyhole Markup Language (KML).

2.1 KML: Google Earth file format

KML is an XML language focused on geographic visualization, including annotation of maps and images. It became an international standard maintained by the Open Geospatial Consortium, Inc. (OGC).
KML can be used to:

- Annotate the Earth
- Specify icons and labels to identify locations on the surface of the planet
- Create different camera positions to define unique views for KML features
- Define image overlays to attach to the ground or screen
- Define styles to specify KML feature appearance
- Write HTML descriptions of KML features into hierarchies
- Locate and update retrieved KML documents from local or remote networklocations
- Define the location and orientation of textured 3D objects

From a technical point of view, KML uses a tag-based structure based on XML standard [9]. All tags are case-sensitive and must be taken from the "KML Reference"[3]. Figure II.2 shows the most important attribute of a KML object. There are many object types, but for the application under interest some basic types like placemarks/points and lines are enough.

[3]available at `https://developers.google.com/kml/documentation/kmlreference`

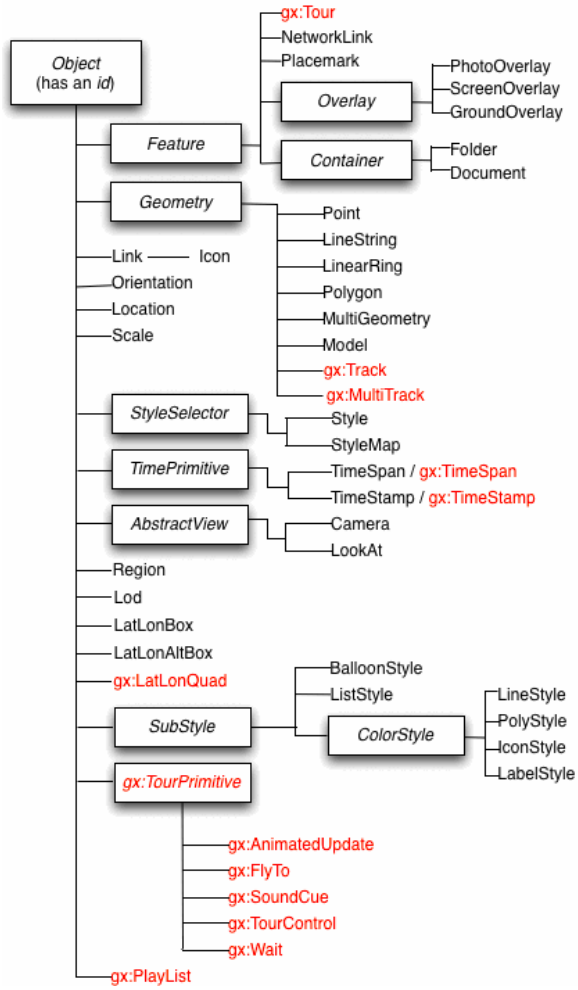


Figure II.2: KML Object Hierarchy Diagram

2.2 Field Border in KML

A border it is basically a close path having an undefined shape and therefore it cannot be represented using one polygon or other predefined geometric shapes. In KML such kind of geometric path should be defined through the object LineString. LineString creates a path from a set of geographic points. Each adjacent point is connected with a line and finally if the ending point coincide with the starting one a close path is obtained. Clearly, this approach required an adequate number of points to make the resulting shape smooth. Fortunately, the generation of the points is automatically done by Google Earth which provides an handy toolbox to graphically create the path object directly drawing it over the map. An example of a field contour encoded in KML code is reported in Listing II.1, where the complete list of coordinates has been omitted as it would be too long.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <name>Shapes</name>
    <Style id="thickLine">
      <LineStyle>
        <width>2.5</width>
      </LineStyle>
    </Style>
    <Style id="transparent50Poly">
      <PolyStyle>
        <color>7fffffff</color>
      </PolyStyle>
    </Style>
    <Placemark>
      <name>Waldpeter Field Boundary</name>
      <description>Field Border</description>
      <LineString>
        <coordinates>
          11.491609399895651,46.453469568513725,0
          11.491755810622863,46.45338389393535,0
          <!-- OMITTED COORDINATES -->
           11.49140555201052,46.45356658266721,0
           11.491609399895651,46.453469568513725,0
         </coordinates>
      </LineString>
      <styleUrl>#thickLine</styleUrl>
    </Placemark>
  </Document>
</kml>
```

Listing II.1: Border in KML

The name and the description tags give the possibility to add meta-data to the KML geographic feature. This comes in handy to classify the field and creating a database of the place already visited. The idea is then to use the data already collected when repeating mission in a previously scanned field. Before entering in the implementation details an example of how the contour appears in Google Earth is displayed in Figure II.3.

3 Implementation

As it was mentioned in 1.2 although the source of the orthophoto used in the software implementation is Google Earth, in defining the mission it has been considered the possibility to take a photo of the whole field and georeference it. This would provide an

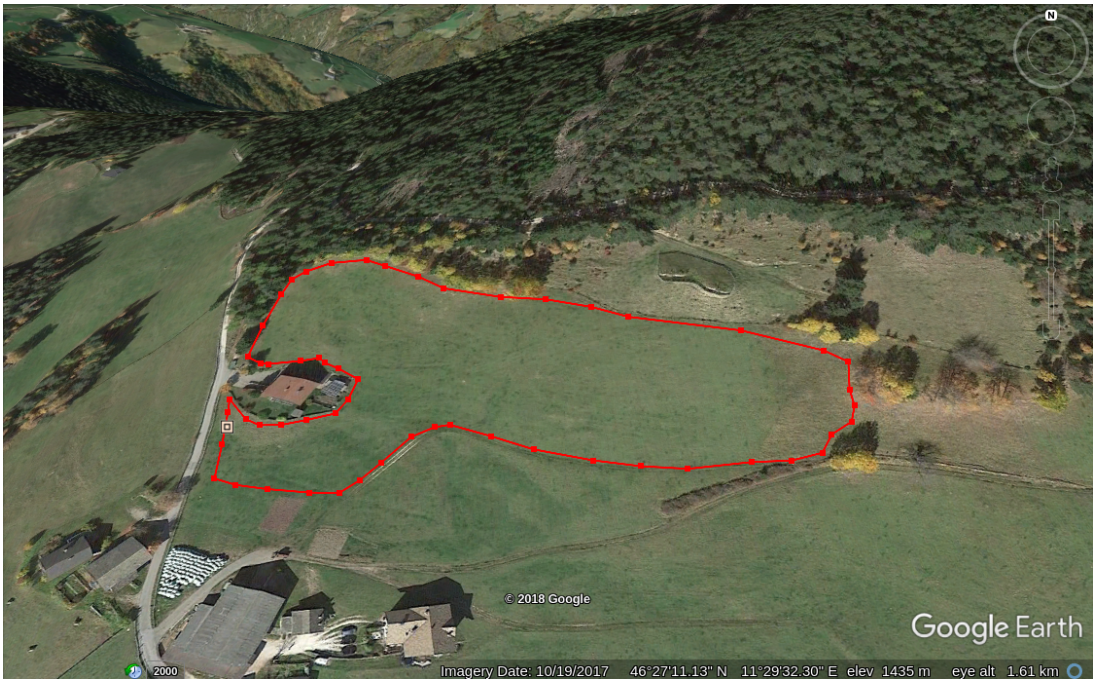


Figure II.3: KML LineString geographic feature visualize on Google Earth

up-to-date and eventually more accurate image, but at the same time the orthophoto generation is computationally heavy especially for a mini PC. This in addition to the limited flight time of the quadcopter suggest to use Google imagery database.

In the following section it is explained how this part of the mission has been implemented in ROS. The most relevant part concerns the representation of the field contour as ROS message and the way to import it given the respective KML file.

3.1 ROS Nodes Architecture

Figure II.4 shows how the three ROS nodes (mission_controller, optical_cam and boundary_generator), displayed as ovals, communicate together through the four topics represented inside rectangles. The node named mavros (for more details see appendix D) on the left side of the diagram, manages everything regarding the communication with the aircraft. It basically provide a ROS interface to read the vehicle status variables and to change the flight controller behaviors.

Nodes specs:

mission_controller It is the node which contains the core of the mission. Through the implementation of a finite state machine it handle every mission steps and the transition among them. For what concern the scope of this chapter the main task of this node is to trigger the other two node, which really carry out the desired tasks.

optical_cam Upon receipt of the trigger_shutter message issued by the mission controller node it triggers the camera shutter and saves the exact GPS position required to geo-tag the photo of the field. At this point the node, in case the it has been chosen to georeference the image locally, will generate the orthophoto. The orthophoto algorithm has not been implemented yet as explained at the beginning of this section.

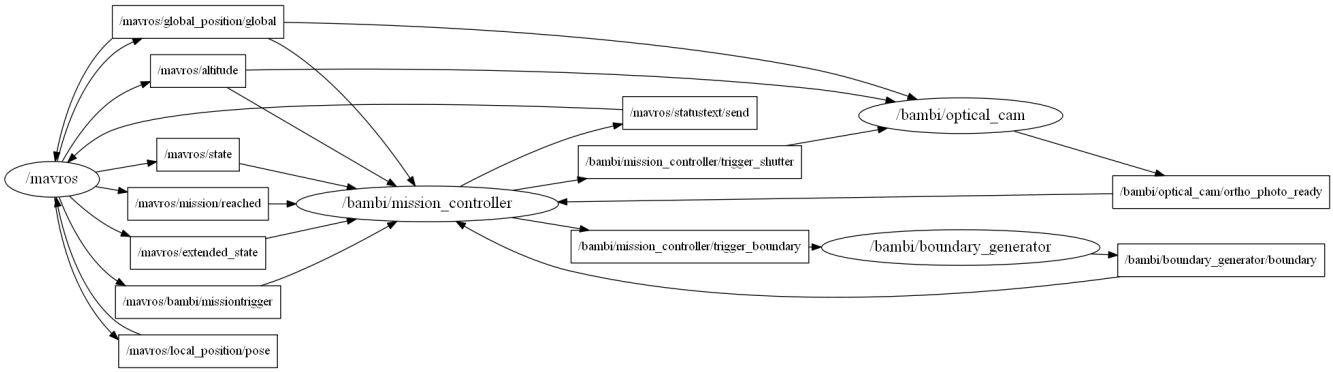


Figure II.4: Nodes and Topics concurring in generating the field contour

The node finish its job once it publishes the message containing the position of the orthophoto back to the /mission_controller node over the orthophoto_ready topic.

boundary_generator This node when triggered is suppose to elaborate the field boundary and sand it to the mission controller node. The communication happens through the two topics trigger_boundary and boundary as shown in the nodes graph.

3.1.1 Relevant message definitions

The definition of relevant messages is given in Listing II.2.

```
############################## FILE: Field.msg ##############################
bambi_msgs/GeoPosition2D[] boundary_path

######################### FILE: GeoPosition2D.msg ########################
float64 latitude
float64 longitude

########################### FILE: NavSatFix.msg ##########################
# Navigation Satellite fix for any Global Navigation Satellite System
# Specified using the WGS 84 reference ellipsoid
uint8 COVARIANCE_TYPE_UNKNOWN=0
uint8 COVARIANCE_TYPE_APPROXIMATED=1
uint8 COVARIANCE_TYPE_DIAGONAL_KNOWN=2
uint8 COVARIANCE_TYPE_KNOWN=3
std_msgs/Header header
sensor_msgs/NavSatStatus status
float64 latitude
float64 longitude
float64 altitude
float64[9] position_covariance
uint8 position_covariance_type

########################### FILE: OrthoPhoto.msg #########################
string filenameWithFullPath
```

Listing II.2: Relevant ROS message definition

Now that a briefly introduction of each node has been done, a deeper analysis of the optical_cam node and boundary_generator node is carried out.

3.1.2 optical_cam

This node has been written in python and it is responsible of the communication between the action camera (Xiaomi Yi Cam) and the other nodes running on the companion computer (raspberry Pi 3b). The camera communicate through WIFI in a server-client like fashion using the TCP protocol. For this reason, in python, the communication is managed using socket library. Once the socket is connected it is possible to send a several different command (in the form of *token*) to remotely control the camera.
The first task of the node is to trigger the camera shutter when requested by the mission controller. The image is fetched and downloaded to the internal storage of the Raspberry, as soon as the camera sends back the message which tells it has successfully taken the photo. All this logic is implemented in the function take_photo listed in Listing II.3. Most of the credits goes to Res Andy for its great effort in reverse engineering the Yi Cam remote control protocol and to provide sample scripts in python. [10] Along with the trigger command, the node continuously listen to the topic mavros/global_position/global where mavros node publishes the global position communicated by the flight controller (GPS fix). In this way the node has the necessary information to geo-tag the photo. Once the photo is geotagged and made available locally on the onboard computer the optical_cam node would be in charge of generating the orthophoto. This part, at first place, has been omitted and left to future developments.

```
1  #! /usr/bin/env python
2  # encoding: utf-8
3  import os, re, sys, time, socket, urllib, datetime
4  from settings import camaddr
5  from settings import camport
6  import rospy
7  def take_photo():
8    # socket initialization
9    srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10   srv.settimeout(5)
11   rospy.loginfo('Trying to connect with yiCam @ ' + str(camaddr) + ':' + ↩
       str(camport))
12   srv.connect((camaddr, camport))
13   # sending access token
14   srv.send('{"msg_id":257,"token":0}')
15   # receiving data and looking for token number to be use in next request ↩
       messages
16   data = srv.recv(512)
17   rospy.loginfo('Got first data from yiCam, looking for "rval": ' + str(↩
       data))
18   if "rval" in data:
19        token = re.findall('"param":\s*(.+)\s*}',data)[0]
20   else:
21        data = srv.recv(512)
22        rospy.loginfo('Got second data from yiCam, looking for "rval": ' + ↩
             str(data))
23        if "rval" in data:
24             token = re.findall('"param":\s*(.+)\s*}',data)[0]
```

```
25   # send shutter command
26   tosend = '{"msg_id":769,"token":%s}' %token
27   srv.send(tosend)
28   #receiving response message
29   data = srv.recv(512)
30   # parsing response message looking for photo file name
31   findCollection = list()
32   maxTries = 3
33   i = 0
34   while len(findCollection) == 0 and i < maxTries:
35     data = srv.recv(512)
36     rospy.loginfo('Got data from yiCam: ' + str(data))
37     findCollection = re.findall('"param":"/tmp/fuse_d/(.+)"}', data)
38     ++i
39   if len(findCollection) == 0:
40     rospy.logwarn("Yi cam photo ERROR")
41     return ''
42   yiCamFileName = findCollection[0]
43   # preparing URL to fetch image
44   url = "http://" + str(camaddr) + "/" + yiCamFileName
45   fileName = '$BAMBI_OWNCLOUD_HOME/yi-cam-pic-' + datetime.datetime.now().↩
       strftime('%Y-%m-%d-%H:%M:%S') + '.jpg'
46   fileName = os.path.expandvars(fileName)
47   rospy.loginfo(url)
48   # retrieve image from URL
49   urllib.urlretrieve(url, filename=fileName)
50   return fileName
```

Listing II.3: take_photo() function in python

3.1.3 boundary_generator

As for optical_cam node, python has been chosen as programming language because of the libraries available. The boundary_generator's goal is, when requested, to publish ROS message ("Field.msg") over the topic boundary containing the array of 2D coordinates representing the field contour. In the following implementation the boundary information is taken from a KML file containing the geographic path manually generated through Google Earth. It has been decided to make a separated for this simple parsing task so that in future it will be ready to hold any image processing algorithm to autonomously extrapolate the field outline from the orthophoto. The code of this node is integrally reported in Listing II.4.

First of all the node subscribes to mission_controller/trigger_boundary in such a way that upon the arrival of a trigger message the callback function cb_boundary_trigger is called. The callback function get the absolute path of the KML file from the message attribute filenameWithFullPath. Now it comes to play the pyKML library.

pyKML is a Python package for creating, parsing, manipulating, and validating KML documents [11]. It is based on the lxml.objectify API[4] which provides access to XML documents in python program. Once the file has been opened it is possible to access to KML child member in the way shown at line 23 of Listing II.4. At this point it is just

[4]It aims to hide the usage of XML behind normal Python objects, sometimes referred to as data-binding. It allows you to use XML as if you were dealing with a normal Python object hierarchy.[12]

a matter of parsing each geographic point as it was a string in the form "<latitude>, <longitude>" using the function split(','). Latitude and longitude coordinate for each point of the contour is then packed in a geoPosition2D object which is push back on the boudary_path vector. Once every point has been parsed and the vector filled the Field ROS message is published and the node finishes its job.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import roslib
roslib.load_manifest('bambi_msgs')
import rospy
from bambi_msgs.msg import OrthoPhoto, Field, GeoPosition2D
import sys
# parse kml files
from pykml import parser
class BoundaryGeneratorNode():

    def __init__(self):
        self.m_boundaryPublisher = rospy.Publisher("~boundary", Field, queue_size=5)
        rospy.Subscriber('/bambi/mission_controller/trigger_boundary', OrthoPhoto, self.cb_boundary_trigger)
        rospy.spin()

    def cb_boundary_trigger(self, OrthoPhoto):
        field = Field();
        rospy.loginfo("Trying to read file %s", OrthoPhoto.filenameWithFullPath)
        with open(OrthoPhoto.filenameWithFullPath) as f:
            root = parser.parse(f).getroot()
            coordinates = root.Document.Placemark.LineString.coordinates.text.split()
            for c in coordinates:
                splitted = c.split(',')
                lon = splitted[0]
                lat = splitted[1]
                pos = GeoPosition2D();
                pos.latitude = float(lat)
                pos.longitude = float(lon)
                field.boundary_path.append(pos)
        rospy.loginfo("Publishing field with %d coordinates", len(field.boundary_path))
        self.m_boundaryPublisher.publish(field);
# Main function.
if __name__ == '__main__':
    # Initialize the node and name it.
    rospy.init_node('boundary_generator')
    rospy.loginfo("BoundaryGenerator STARTUP")
    # Go to class functions that do all the heavy lifting. Do error checking.
    try:
        boundaryGeneratorNode = BoundaryGeneratorNode()
    except rospy.ROSInterruptException: pass
```

Listing II.4: boundary generator node in python

.

Chapter III

Coverage Path Planning

This chapter begins with a theory digression about the topic of Coverage Path Planning. Then the algorithm chosen for the application in this thesis is explained in details and finally the ROS implementation of the algorithm is presented and discussed.

1 Problem Definition

The area coverage problem can be abstractly described as follow:

Given a convex or non convex shaped area $A \subset \mathbb{R}^2$ decomposed approximately by a finite set of cells $C = \{c_1, .., c_n\}$ such that, $A \approx \bigcup_{i=1}^{n} c_i$.

Find a coverage trajectory P as a finite set of continuous way-points p, which can be written as $P = \{p_1, .., p_n\}$. Where each way-point corresponds to the centroid of a corresponding cell, thus $dim(P) = dim(C)$; Considering that valid solutions of P should not visit a way-point twice, the variable of interest to minimize is typically the path length and the number of changes in directions.

2 Theory Background

Coverage Path Planning (CPP) is the problem of finding a trajectory for a mobile robot such that a target area is completely swept by the sensor footprint. In the following section it is first presented the conventional CPP algorithms in use for mobile robots. Later, the focus moves more specifically over aerial application showing some previous work regarding this topic.

2.1 Coverage Path Planning for Mobile Robots

The problem of finding an optimal coverage path, even for a simple polygon, is classified as NP-hard [1] [13]. Hence, existing approaches try to find an approximate solution which fits at best the specific application requirements. For 2D coverage, some methods decompose the target area into simpler polygons and for each compute the coverage path. Other

[1] NP-hard problems are problems for which there is no known polynomial algorithm, so that the time to find a solution grows exponentially with problem size. Although it has not been definitively proven that, there is no polynomial algorithm for solving NP-hard problems, many eminent mathematicians have tried and failed.

methods use a grid-based representation which leads to an approximate coverage. Finally, closed-loop control methods avoid the needs of an a priori representation of the target region.

2.1.1 Exact Cellular Decomposition

One of the main approach in area coverage path planning is based on the divide-and-conquer strategy. In this method the target area is decomposed in simple regions called cells. Since all cells have a simple structure, each can be covered with simple motions such as back-and-forth motion as in Figure III.1. This kind of motion is called *Lawnmower pattern*. Once the robot visits all cell, coverage is achieved.

Trapezoidal decomposition is the most popular cell decomposition. This decomposition relies heavily on the polygonal representation of the planar configuration space [14]. Cells are in fact obtained simply by sweeping a vertical line,termed as *slice*, through the 2D plane and, upon reaching each vertex of the environment polygon, the required edges are added to create trapezoids (see Figure III.2). Two cells sharing a common boundary are defined as adjacent and accordingly an adjacent graph is produced. At this point the strategy consist in finding the exhaustive walk which visits all cells and minimizes the cost of traveling between them. An important factor in finding an efficient path is the choice of the slice direction when decomposing the target area as analyze by Oksane [15]. In his work, he performed a local optimization to find the direction for the sweeping line which minimize trajectory length and the number of turnings.

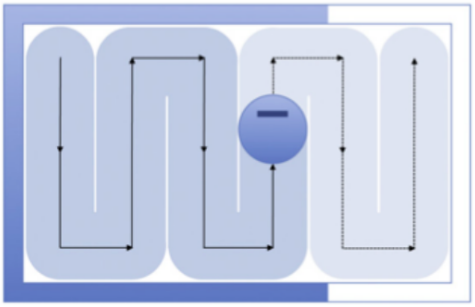


Figure III.1: Lawnmower pattern

In trapezoidal approach many neighbor cells could be merged together into one cell resulting in a shorter coverage path. In the left hand side of Figure III.3 the robot needs to make one additional lengthwise motion to cover the remaining portion of the trapezoidal cell. Following this idea, Choset proposed in [16] an enhancement of the trapezoidal decomposition method, the *Boustrophedon cellular decomposition* designed with the aim to minimize the path length by generating bigger cells. The boustrophedon decomposition is formed similarly to trapezoidal, but by considering only the vertices at which the slice can be extended both up and down in the free space (see Figure III.4). Such vertices are called *critical points*. As before, by visiting all the nodes in the adjacency graph the area is covered completely.

In summary, exact cellular decomposition guarantees a complete coverage of the target area. A bigger drawback to the trapezoidal method is that it fundamentally requires a polygonal workspace, which is not a realistic assumption for many applications. To overcome this limitation the boustrophedon decomposition was designed.

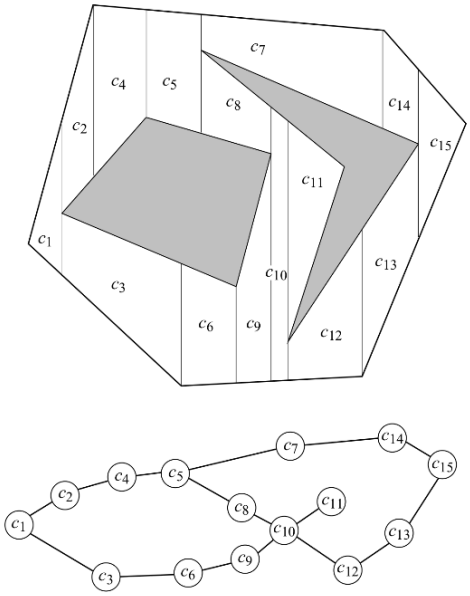


Figure III.2: Trapezoidal decomposition and adjacent graph [14]

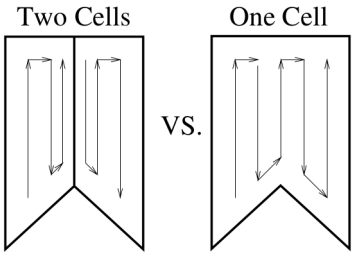


Figure III.3: Fewer cells is better

2.1.2 Grid-based Methods

Another largely used method in coverage path planning divides the environment in a collection of grid cells of the same shape. Each cell is marked as empty or occupy according to the presence of obstacle in that location. This will produce an *approximate cellular decomposition* as the obstacles are represented by grid cells. The algorithms using this grid-based representation should plan a path which visit all the empty cells of the grid minimizing the traveling cost. Primitive criteria could be to avoid revisiting cells and to move from adjacent cells only. Two interesting algorithms have been developed for this scope: The *Wave-front* method [17] and *Minimum-Spanning-Tree* (MST) method [18].

The *Wave-front* method is based upon distance transform (DT) path planning methodology. This approach considered the task of path planning to finding paths from the goal location back to the start location. The path planner propagates a distance wavefront through all free space grid cells starting from the goal cell. The distance wave front flows around obstacles and eventually through all free space in the environment.
"For any starting point within the environment representing the initial position of the mobile robot, the shortest path to the goal is traced by walking down hill via the steepest descent path. If there is no downhill path, and the start cell is on a plateau then it can be concluded that no path exists from the start cell to the goal cell i.e. the goal is unreachable" [17].

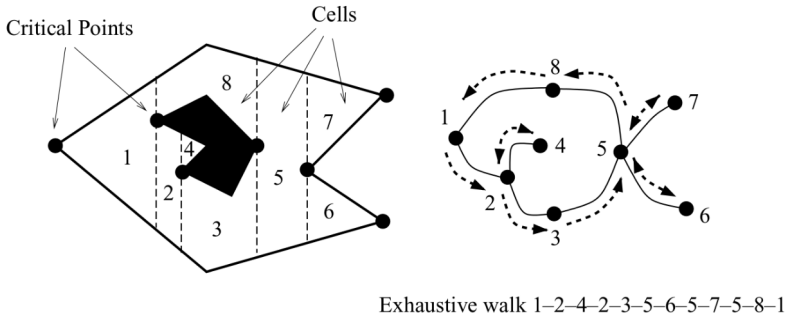
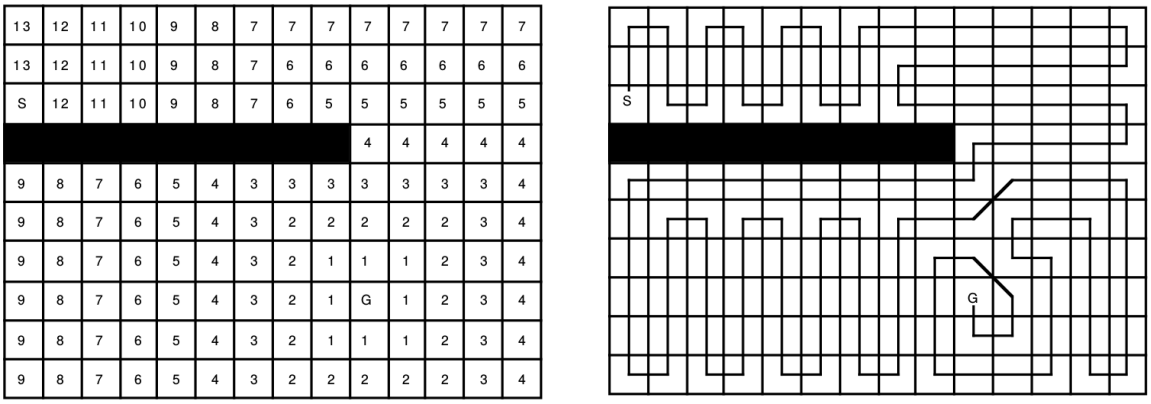


Figure III.4: Fewer cells is better

One significant advantage that distance transform path planning has over other path planning methods is that it can easily be induced to exhibit different types of robot navigation behaviors. In addition to the "optimum" i.e. shortest path behavior it is possible to have the "complete coverage" behavior. To achieve the complete coverage path planning behavior, instead of descending along the path of steepest descent to the goal, the robot follows the path of steepest ascent. In other words the robot moves away from the goal keeping track of the cells it has visited. The robot only moves into a grid cell which is closer to the goal if it has visited all the neighbouring cells which lie further away from the goal. An example of the complete coverage path is shown in Figure III.5. The



(a) Distance waveform grid

(b) Path of complete coverage

Figure III.5: Wavefront method [17]

pseudocode to obtain such behavior is listed in algorithm 1.

A nice feature of this approach is that it is possible to set both starting and goal position. This comes in handy in cleaning or lawn mowing application. However, in some case this method can not avoid revisiting cells. Another shortcoming it is worth to highlight is the high number of turn in the coverage path. This is because the path follows the "spiral" of the distance transform wave front that radiates from the goal. A solution could be to take into account other factors than only the distance from the goal.

The *Minimum-Spanning-Tree* (MST) method only considers the cells that are completely unoccupied by obstacles [74]. First, a grid with cells 4 times the robot sensor footprint is constructed. Then a graph is created by representing each cell with a node and connecting two nodes if they are neighbor cells. The minimum spanning tree of this graph then is computed. In order to achieve complete coverage of the environment the robot

Algorithm 1: CPP algorithm based on Distance Wavefront

input : A matrix M representing the grid
output : A matrix M filled with Distance Transformation

```
startCell ← currentCell;
forall cells of M do
    cell ← NotVisited;
repeat
    Find unvisited Neighboring cell with highest DT;
    if No neighborCell found then /* Goal reached */
        currentCell ← Visited;
        stop ← true;
    if neighborCell DT ≤ currentCell DT then
        currentCell ← Visited;
    currentCell ← neighborCell;
until stop is true;
```

circumnavigates the spanning tree visiting quadrants of the cells as shown in Figure III.6. Unlike the wave-front method, this algorithm never revisits a cell and hence produces an optimal solution in terms of path length.

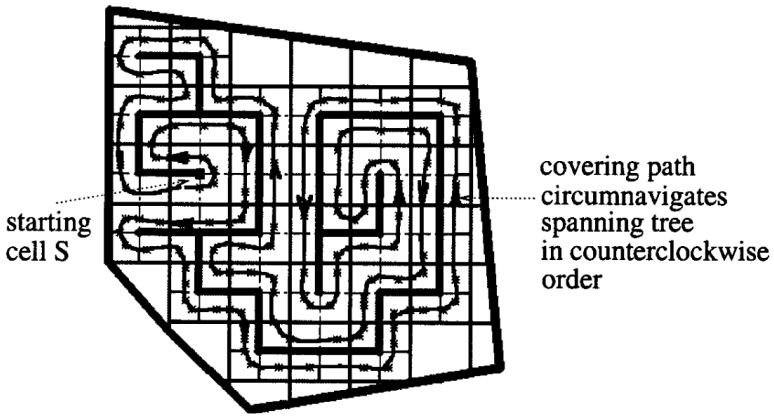


Figure III.6: Minimum Spanning Tree method

2.1.3 Close-loop Control methods (Online)

Online approaches compute the coverage path *in situ* based on sensor information, therefore unlikely offline counterpart does not require an a priori knowledge of the environment. These methods usually formulate the coverage task as an optimization problem in which the goal is to minimize (e.g., sensor overlap) or maximize (e.g., sensing of uncovered area) a metric. Howard in [19] developed a method in which a *continuous potential field* is used to accomplish a coverage task using a team of mobile sensors. Sensors are repelled by obstacles and other team members and are consequently spread out in the area. This is a static and somehow primitive approach for CPP as it do not guarantee neither efficiency

nor complete coverage, its aim is in fact to rapidly find hazards in a scenario involving hazardous materials leak in a damaged structure.

Other approach consist in a control law composed of two components: i) a *local component* that is dependent on the coverage status of a local neighborhood, designed to direct the robot towards regions with high local coverage benefit and ii) a *global component* that depends on the global coverage status and directs the robot towards a point from where the robot can cover uncovered regions [20].

A recent work by Song in [21] present the *ε-algorithm* (that stands for ε-STAR or "ε-coverage via structural transitions to abstract resolutions"), where ε refers to the cell resolution. As shown in Figure III.7, the algorithm utilizes an Exploratory Turing Machine (ETM), that consists of a two dimensional multilevel tape formed by Multiscale Adaptive Potential Surfaces (MAPS). The ETM stores and updates the information corresponding to unexplored, explored, and obstacle-occupied regions, as time-varying potentials on MAPS. In essence, it takes advantage of both the potential field-based and sensor-based planning methods by incrementally building the MAPS using real-time sensor measurements. The ETM acts as a supervisor to the autonomous vehicle and guides it with adaptive navigation commands.

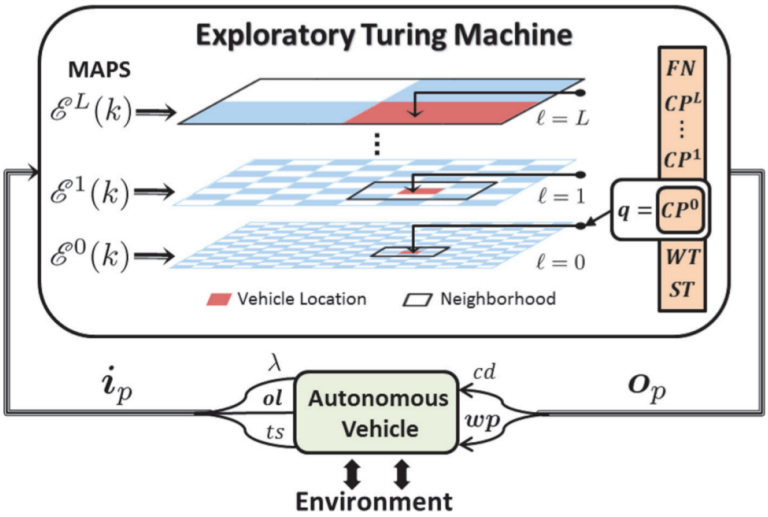


Figure III.7: ETM as a supervisor of the autonomous vehicle [21]

2.2 Aerial Coverage using UAVs

In aerial coverage the CPP methods presented in subsection 2.1 have been largely adopted (e.g. Moon *et al.* in [22] used the Boustrophedon approach for crop dusting with multicopter). In this application the environment is specified as a grid or a polygon using GPS coordinates for polygon nodes or for the center of the cells. Obstacles are generally neglected as the flight altitude is usually high enough to avoid unwanted collision. Anyway in aerial coverage there are often uninterested regions which is out of interest in the coverage task but flight over them is allowed. For example for the specific coverage required in BambiSaver a lake is of no interest as no animals hide there. An important factor in aerial coverage using UAV is the limited flight time of the vehicle. For this reason generating an efficient path is of great importance. Along with shortening the path limiting as much as possible sensor overlap, it is relevant to minimize the number of turns in the flight

trajectory. Reducing the number of turnings will produce trajectory that consist of long straight stripes. This kind of trajectory let the robot to maintain as long as possible the cruise speed, reducing acceleration and consequently power consumption.

3 Bambi Project CPP

The proposal implementation is based on the *Wavefront* algorithm presented in subsubsection 2.1.2. This algorithm has been chosen among the other for the following reasons:

- The environment is known *a-priori* thus an offline solution is preferable.
- The wavefront algorithm proved to be robust and quite efficient for aerial coverage application [23].
- This approach provide the possibility to implement different cost function, thus it is possible to obtain different behaviors (i.e. take into account of the ground elevation profile).

3.1 Workplace Sampling

The workplace is decomposed through approximate cellular decomposition. In this method as explained in subsubsection 2.1.2, the target area is divided in a grid of squared cell and a point is placed in the center of each rectangle. It has been decided to use squared shaped cells even though most of the available thermal sensor have a rectangular FOV to simplify the problem. In fact under this assumption it is possible to neglect the camera orientation and, in addition, this will produce an overlap in the recorded images which improves coverage performance compensating for the presence of small inaccuracy in the tracking of the flight trajectory. Cells dimension is chosen according to the required image resolution and it is dictated by the thermal sensor resolution. Once the cell dimension has been defined the next step is to compute the flight altitude which guarantee that the cell is completely covered by the FOV (Field of View) of the sensor when the UAV reach the point in the center of the cell. This height is maintained during the whole mission to ensure a uniform sampling of the whole field.

The thermal camera chosen to make the computation is the Flir Duo R a radiometric dual-sensor thermal camera specifically made for drone [24]. The main specification of the camera are listed in Figure III.8.

In the following analysis the camera is supposed to be mounted on a gimbal and facing downwards. In this way the gimbal compensate for UAV movements and keep the camera always parallel to the horizontal plane making the FOV projection easier to be computed. The required FOV dimension τ_d is calculated so that one pixel of the image correspond at least to the required resolution:

$$\tau_d = I_r \cdot p_{min}$$
$$I_r = I_t \cdot \gamma$$

Where:

I_t: target dimension

Overview	FLIR Duo R
Thermal Imager	Uncooled VOx Microbolometer
Sensor Resolution	160 × 120
Lens	57° × 44°
Spectral Band	7.5 – 13.5 µm
Thermal Frame Rates	7.5 Hz (NTSC); 8.3 Hz (PAL)
Thermal Measurement Accuracy	+/-5°C
Visible Camera Resolution	1920 × 1080
Visible Camera FOV	90°

Figure III.8: Flir Duo R thermal camera specs [24]

γ: safe factor ($\gamma < 1$)
I_r: required resolution
p_{min}: minimum sensor resolution (pixels)

Each cell must be smaller then the FOV, thus: $d \leq \tau_d$.

The flight height τ_h is then compute from the definition of FOV:

$$\tau_d = 2\,\tau_h \cdot \tan\left(\frac{\alpha}{2}\right)$$
$$\Rightarrow \quad \tau_h = \frac{\tau_d}{2 \cdot \tan\left(\frac{\alpha}{2}\right)}$$

Where:

τ_d: FOV minimum dimension
α: minimum degree of the camera lens

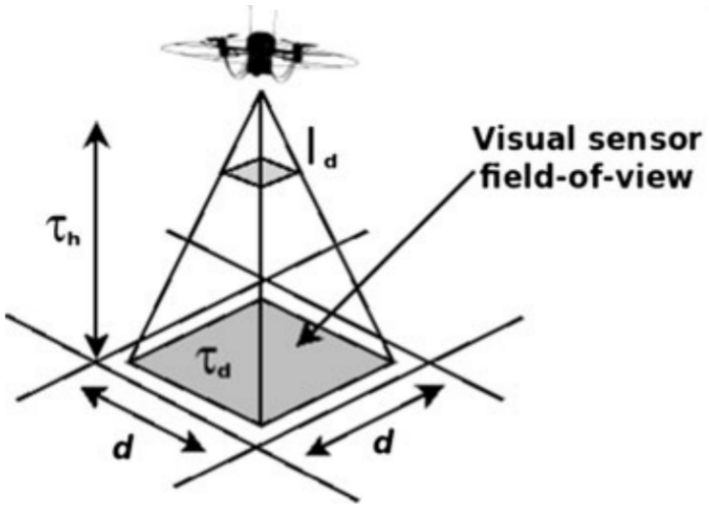


Figure III.9: UAV altitude and the image requirements

Now, supposing to model the target of the searching mission as a square with sides of 0.2 cm (a little fawn is generally much bigger) and a safe factor $\gamma = 0.5$, then the above formulas produce the following results:

$\tau_d = 12\,m$
$\tau_h = 14.85\,m$

Therefore it has been decided to go for $10m \times 10m$ cells and fly at an altitude of $14\,m$ above the ground to have an adequate safe margin.

3.2 ROS Node Architecture

The coverage path planning algorithm is implemented in the ROS node named coverage_path_planner. The graph in Figure III.10 display the nodes and topics related to the CPP procedure. The scope of the other nodes that appear in the graph are:

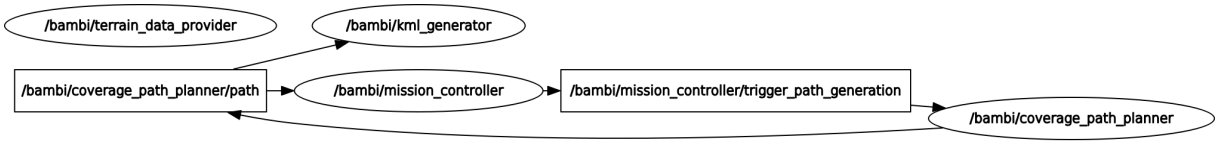


Figure III.10: ROS-graph of CPP related nodes and topics

mission_controller It handles all the mission operations as discussed in chapter II

terrain_data_provider It provides a service (this is why in the graph it is isolated) which allow the CPP node to request terrain data. It access the Google elevation databases using Google elevation API. The inquiry is done in the form of an URL request through the python library urllib [25].

kml_generator It generates a KML file from the CPP generated path. This task is not needed during the mission but it is of great help in debugging and for better display the CPP output.

The *Coverage Path Planner* node task can be further divided in five sub-operations: (1) Approximate cellular Decomposition; (2) Choosing Starting Position (3) Wave-front Propagation Algorithm; (4) Coverage Path Generator; (5) Spline Interpolation.

In the following sections, first the definition of relevant ROS messages is given, then each sub-task is analyze separately. For parts where coding has required additional steps and consideration the implementation is analyze more in details, otherwise only the pseudocode is given for clarity and readability reasons. Anyway the whole code is accessible on the project GitHub page [26].

3.2.1 Relevant Messages Definition

Ros nodes communicate using messages over different topics. There are many build-in messages already defined in ROS, anyway it has been necessary to define custom messages for the specific use case. The definition of relevant messages is given in Listing III.1

```
1 #-----------------------------Bambi_msg-----------------------------#
2 ############################FieldCoverageInfo#############################
3
4 float32 thermal_camera_ground_footprint_width
5 float32 thermal_camera_ground_footprint_height
```

```
# Scanning Altitude
float32 relative_altitude_scanning
# Altitude to use when going back home
float32 relative_altitude_returning
bambi_msgs/GeoPositionWithRelativeAltitude current_position
bambi_msgs/GeoPosition2D home_position
bambi_msgs/Field field

##################################Field###################################

bambi_msgs/GeoPosition2D[] boundary_path

############################GeoPosition2D#############################

float64 latitude
float64 longitude

###################GeoPositionWithRelativeAltitude###################

bambi_msgs/GeoPosition2D geopos_2d
#Altitude is in meters
float32 altitude_over_ground

################################Path#################################

bambi_msgs/GeoPositionWithRelativeAltitude[] geometric_path

############################TerrainData#############################

float64 altitudeInMeters
```

Listing III.1: Relevant message definition for CPP node

The *coverage path planner* node receives the FieldCoverageInfo message subscribing to the topic /bambi/mission_controller/trigger_path_generation. The message contains the field contour data in the member type bambi_msg/Field and additional information such as *sensor footprint*, scanning altitude, return to home altitude, current global position and home global position. Once the path is generated the node publish a path message on the topic /bambi/coverage_path_planner/path

3.3 Approximate Cellular Decomposition

In this section it is analyze how to decompose the workplace so that it can be represented as a grid of squared cells.
The cells dimension has been already computed, thus the grid dimensions are simply obtained measuring the maximum field dimensions along the horizontal and vertical axis and dividing by the cell dimension. The result of both divisions is rounded up in order to guarantee that in any case the field is smaller than the grid. The cells of the rectangular matrix so obtained, have to be marked depending on whether they are part of the field or not. The strategy adopted is to represent both the field and the cell of the grid as polygons in the same coordinates reference system. To check if a particular cell belongs to the workplace the intersection between the two polygons is performed. If the intersection

exists the cell under consideration is part of the field, otherwise it is not. The *approximate cellular decomposition* of the target area is so achieved. The algorithm 2 explain formally how to construct the grid representing the working field.

Algorithm 2: Approximate cellular decomposition of the field in a grid of squared cells

input : A polygon fieldPoly representing the target area; The cellDimension

output : A matrix M which correspond to the cellular decomposition of the field (i.e. where each cell is marked as "Field" or "Empty" according to whether it is part of the field or not)

```
/* border members of the polygon object are the East or
   North (UTM) coordinates of the polygon's extremities   */
width ← fieldPoly.rightBorderE − fieldPoly.leftBorderE ;
height ← fieldPoly.topBorderN − fieldPoly.bottomBorderN ;
/* Computes the matrix dimensions                         */
nE ← ceil(width/cellDimension) ;
nN ← ceil(height/cellDimension) ;
Define M as a nN × nE-matrix;
forall x ∈ M do
    fieldPoly ← getPolygonFromMatrixCell(fieldPoly.bottomBorderN,
     fieldPoly.leftBorderE, x, cellDimension) ;
    if fieldPoly ∩ cellPoly > 0 then /* cell x intersect fieldP   */
        M (x) ← Field;
    else /* cell x do not intersect fieldP                      */
        M (x) ← Empty;
```

3.3.1 Implementation

The implementation of the cellular decomposition in C++ begins with the conversion of field border geo-points from WGS84[2] to UTM[3] coordinate. This is required to obtain X (easter coordinate in UTM) and Y (norther coordinate in UTM)) coordinates of each point in meters and construct the polygon representing the field. In practice the conversion is achieved using the geodesy ROS package (see [27]).

The polygon is represented with the C++ type boostFieldBorderPolygon which is part of the Boost.Geometry library [28]. To define it is just required to add point-by-point the outer contour. A snapshot of the code which perform this first part of the decomposition is given in Listing III.2.

The code implementing the algorithm 2 is not listed because it do not add anything relevant to what already seen in the pseudocode except the specific C++ syntax and definition. An example of the output matrix is displayed in Figure III.11 where the cells which are part of the field are marked with "XX" otherwise with "-2".

[2]See section 2 of chapter II

[3]Universal Transverse of Mercator (UTM) conformal projection uses a 2-dimensional Cartesian coordinate system to give locations on the surface of the Earth. The units used are meters

```
polygon_t boostFieldBorderPolygon;
   for (auto point : fieldCoverageInfo.field.boundary_path) {
      geographic_msgs::GeoPoint geoPoint = geodesy::toMsg(point.latitude, ↩
         point.longitude);
      geodesy::UTMPoint utmPoint((geoPoint));
      boost::geometry::append(boostFieldBorderPolygon.outer(), point_t(↩
         utmPoint.northing, utmPoint.easting));
   }
```

Listing III.2: Coordinates transformation and polygon construction in C++

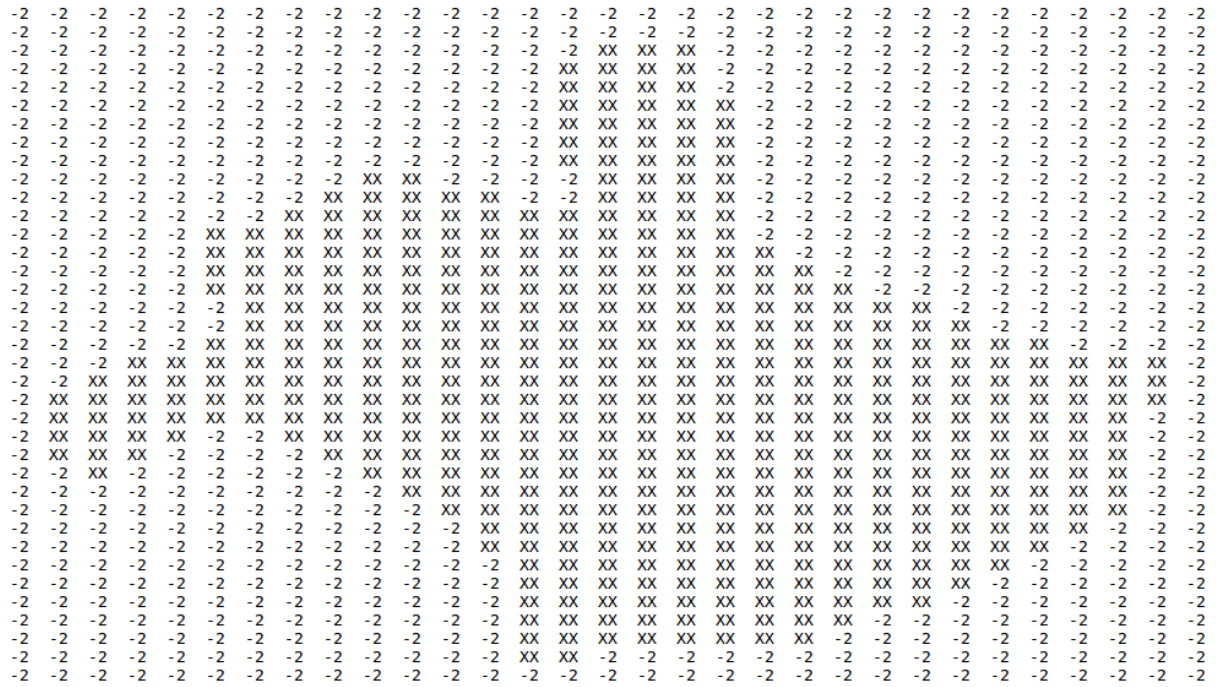


Figure III.11: Example of Cellular Decomposition Matrix

3.4 Choosing Starting Position

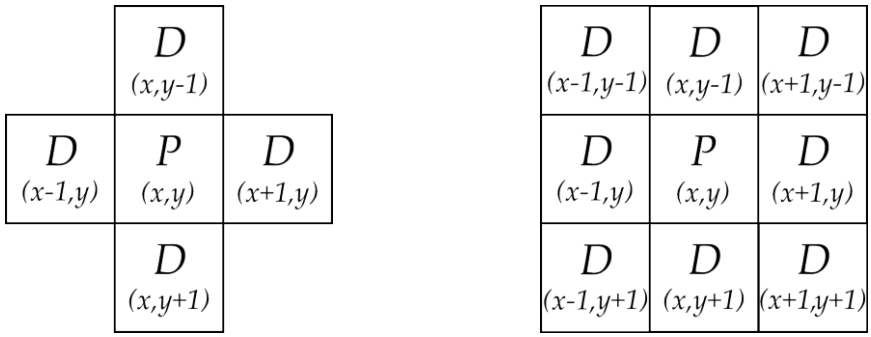
For this specific use case there is no constraint in the choice of the starting position other that it should be inside the target area. Hence it makes sense to select a smart starting position so that the path planner could generate an efficient path. After some test it seems that a good starting position is the one corresponding to the most isolated cell. Isolated in the sense that it is the cell with the lowest number of neighbors having the same cost. Among all the candidate cells, it is selected the one having the higher cost value (i.e the furthest from the goal cell).

3.5 Wave-front Propagation Algorithm

Being G the grid graph, where each element represents a cell through its centroid. Consider the navigation function: $\phi : G \rightarrow [1, \infty]$ which has the properties as a potential function. For the goal cell, the navigation cost $\phi(c_G) = 1$ and the further the cell position to the

goal cell the larger the navigation cost[4]. Once the cost has been assigned to each cell the planner algorithm will find the path which guarantees that all cells having an higher value then the goal cell are visited before the robot reaches the goal.

In order to construct a navigation function, we must consider the type of *cell connectivity* based on the maneuverability of robot. In grid-based path planning there are two types of connectivity: the Von Neumann neighborhood (Figure III.12a) and Moore neighborhood (Figure III.12b). In a Von Neumann neighborhood, the robot turning angle is limited to ±90. In Moore neighborhood the robot will be able to turn ±45, ±90, or ±135. A quad-rotors UAV, being an *holonomic* robot, it can turn at any direction rotating to any yaw angle value by changing the velocity of each motors. Therefore, it is straightforward the choice of using Moore neighborhood.



(a) (b)

Figure III.12: III.12a Von Neumann Neighborhood III.12b Moore Neighborhood

Now using the distance transformation (as cost function) the matrix is filled starting from the goal cell. This is carried out through wave-front propagation algorithm listed in algorithm 3. The initial wavefront is W_0, which represents the states where the current cell is the *goal cell*. The algorithm can immediately assign an optimal cost-to-go value of 1 to every state that can be reached in one stage from any state in W_0. The states that receive cost 1 can be organized into the wavefront W_1. The unexplored neighbors of W_1 are assigned cost 2. This process repeats inductively from i to $i + 1$ until all reachable states (cells) have been reached. In the end, all the cells are filled in $O(n)$ time, in which n is the number of reachable grid states. In Figure III.13 are represented, using 3D surface plot in Matlab, the value associated to each cell. The figure is taken from the coverage path in Figure IV.6c as being the goal position in the center (the minimum point of the plot) of the field, it produce a easier to understand graph.

[4]In the actual implementation only distance transformation has been taken into account as cost function.

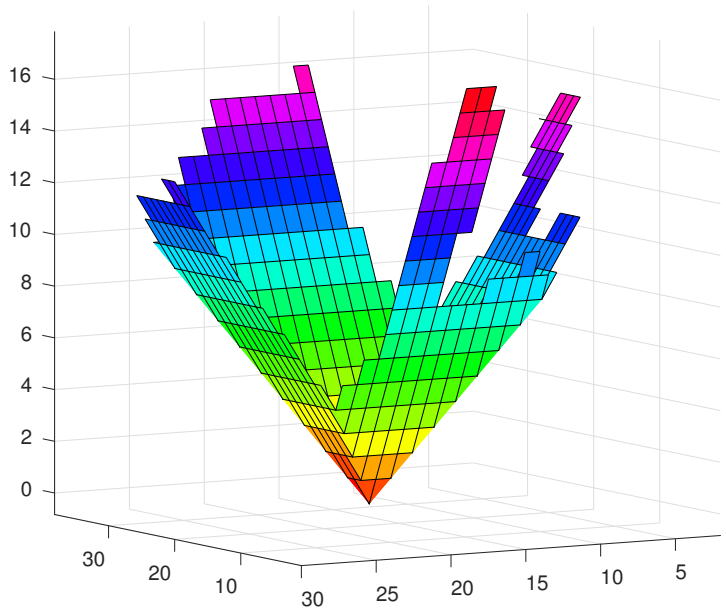


Figure III.13: Field 3 Wavefront propagation (goal position at the center)

Algorithm 3: Filling matrix using Wavefront propagation starting from the *goal* cell

input : A matrix M representing the grid:;the goalCell
output : A matrix M_o filled with Cost Function

$W_0 \leftarrow$ goalCell;
i $\leftarrow 0$;
repeat
  $W_{i+1} \leftarrow$ empty ;
  forall x $\in W_i$ **do**
    M_o(x) $\leftarrow$ i;
    $W_{i+1} \leftarrow$ `getUnexploredNeighbors(M, x);`
  i $\leftarrow$ i $+1$;
  if W_{i+1} *is empty* **then** `/* No more cells to expand */`
    stop $\leftarrow$ true;
until stop *is true*;

3.6 Coverage Path Generator

Once the elements matrix has been filled according to the cost function, the last step is to generate the path which assure that each cell is visited. This is done through a gradient ascending searching algorithm from the start cell. All the unvisited neighbors of the current cell are evaluated and for each is given a *priority*. The neighbor having the higher priority is appointed and selected as the current cell for the next step of the iteration. The process continues until there are no more unvisited neighbors.

Priority function takes into account different features of the neighbors cells which are in order of relevance:

1. The cost function value associated to that cell
2. How many adjacent cells have the same value, favoring the neighbor which has the lower number of same value adjacent cells. This is to avoid as much as possible the risk of reaching a local minimum which could cause the algorithm to stack.
3. The Von Neumann neighbors are preferred, i.e the diagonal neighbors have less priority because they are a little farther ($\sqrt{2}\,l$ instead of just l as it is for Von Neumann neighbors)

The pseudocode which plan the coverage path is listed in algorithm 4.

Algorithm 4: Generating Coverage Path

input : the grid matrix M filled with Distance Transformation values; the startCell
output : A vector P of waypoints representing the coverage path

```
currentCell ← startCell;
reachedEnd ← false ;
while not reachedEnd do
    reachedEnd ← true ;
    bestPriority ← -1 ;
    N ← getUncoveredNeighbors(M, currentCell) ;
    forall x ∈ N do
        reachedEnd ← false ;
        priority ←
         M(currentCell) ∗ 100 + (8 − nSameValuedNeighbors(M, x) ∗ 10 + 1;
        if isDiagonalCell(currentCell, x) then /* It is a diagonal
         cell                                                    */
            priority ← priority − 1 ;
        if priority > bestPriority then /* find highest priority cells */
            bestPriority ← priority;
            bestChoice ← x;
    if not reachedEnd then
        M(currentCell) ← Covered;
        P.push_back(getCentroidCoordinates(currentCell)) ;
        currentCell ← bestChoice;
```

3.7 Spline Interpolation

The resulting coverage path is a piecewise function in the local reference system made up of segment connecting the center of the cells in the order provided by the algorithm shown in subsection 3.6. Therefore the outcome path is really sharply and edgy. Before sending the trajectory to the vehicle it is then required to smooth it. This is done using spline interpolation. Spline interpolation is a form of interpolation where the interpolant is a special type of piecewise polynomial called a spline. This subject goes outside the scope of the thesis, thus only the implementation is given.

3.7.1 Implementation

The implementation in C++ relay on the *cpp-spline* Library provided by Xianshun Chen who make it available on GitHub [29]. Figure III.14 shows the path before and after the interpolation. The smoothing effect is evident since the algorithm produce a piecewise function imposing continuity up to second order derivative. This positive features comes at the expenses of accuracy, because the resulting spline does not exactly pass through every given point as evidenced in Figure III.15. Fortunately this inaccuracy has a little impact on coverage performance due to sensor overlap considered in the computation of the flight altitude (see subsection 3.1).

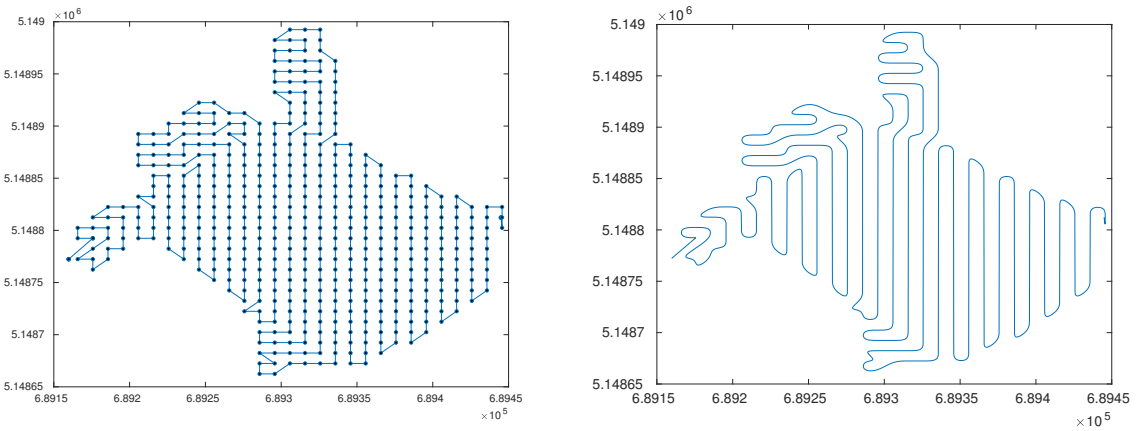


Figure III.14: CPP path (a)before and (b)after interpolation

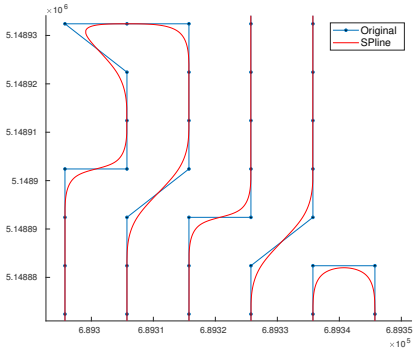


Figure III.15: Interpolation error

Chapter IV

Simulation Results

The PX4 flight stack firmware offers a great Software-In-The-Loop (SITL) simulation environment which was fundamental in develop the software. In fact, it provides a real-time, safe and convenient way to test the software implementation without all the risks related to a real flight.

1 Simulation Environment

The simulation environment consist of:

- **PX4 SITL**: The PX4 simulated hardware which reacts to the simulated given input exactly as it would react in the reality and issues the output as a percentage of the total thrust that every rotor has to provide.
- **Gazebo**: The dynamics simulator that is used by the SITL. This software reads the PX4 output and, by elaborating the modeled dynamics, provides the simulated input to the PX4. This allows for a quite accurate simulation of the real model behavior during flight.
- **ROS**: the robotics middleware over which the Bambi Project software (in the form of package) runs.
- **QGC**: the ground control station used to send the mission start/stop (see appendix C) command and to monitor all the UAV parameters during flight.

The different parts of the system (Figure IV.1) are connected via UDP, and can be run on either the same computer or another computer on the same network. The communication protocol is MAVLink (see appendix C).
The following simulations has been carried out on a single PC running Ubuntu 16.04 LST.

Gazebo offers the possibility to load a satellite map through the StaticMap plugin[1]. In this way the simulation environment (Figure IV.2) represents quite accurately the real world scenario. The vehicle used in simulation environment is the 3DR Iris (Figure IV.3), because its model was already created by PX4 team along with different sensors like the 2D 360° lidar that was used to test object avoidance.

[1] documentation at `http://gazebosim.org/tutorials?tut=static_map_plugin&cat=build_world`

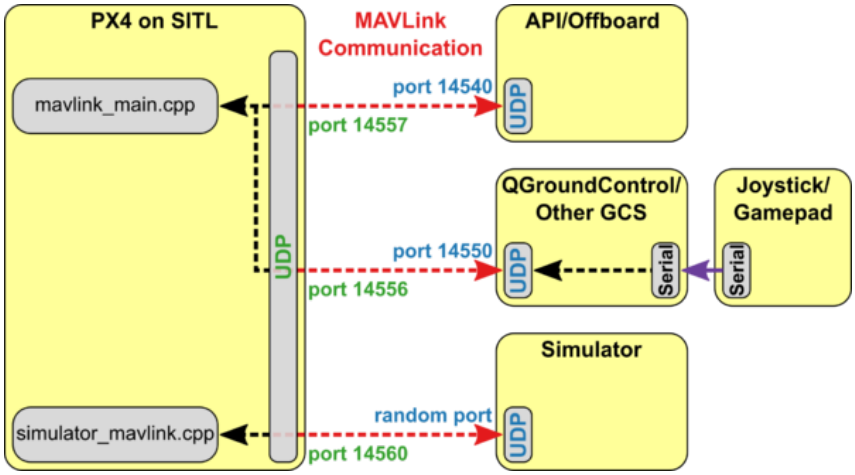


Figure IV.1: SITL with Gazebo architectural scheme

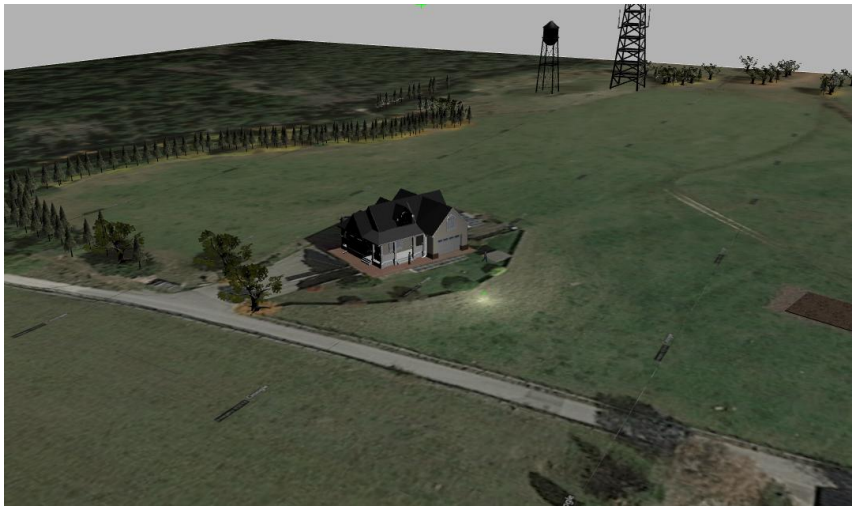


Figure IV.2: Gazebo satellite ground plane

2 Starting Bambi Mission

Mission start and stop command are issue using the QGC custom command widget displayed in Figure IV.4. This widget was written in Qt Modeling Language (QML) an user interface markup language providing an easy way to write simple GUIs [30].
The button BAMBI MISSION START automatically send the mission start message while BAMBI MISSION STOP send the mission abort message which according to the current mission state causes the UAV to return to the home position or to immediately land at the current position.

3 Testing CPP algorithm

In order to test the Coverage Path Planning algorithm it would be necessary to run through the whole mission steps and until the mission controller issue the FieldCoverageInfo message triggering the CPP node. Therefore to save the time required by the drone to perform the first mission tasks it is convenient to publish a "fake" message directly over

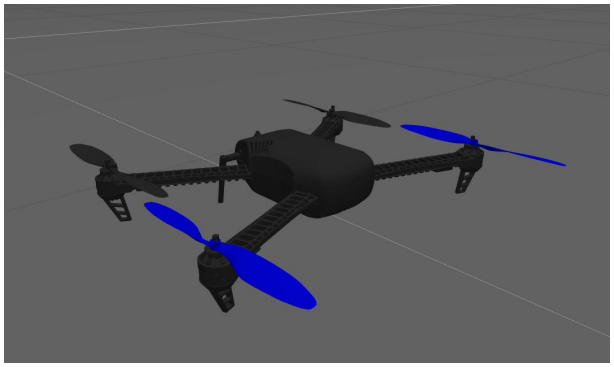


Figure IV.3: Gazebo 3DR iris model

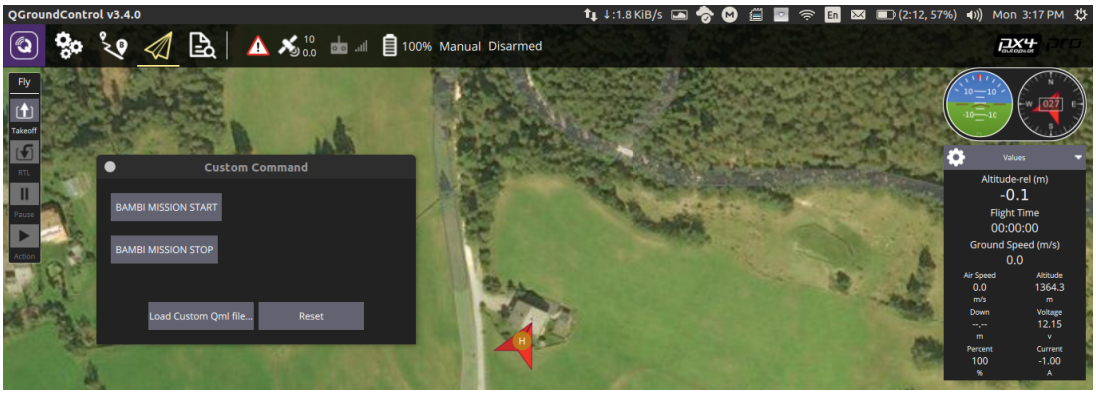


Figure IV.4: QGC Bambi's custom command widget

the /bambi/mission_controller/trigger_path_generation topic. This is performed executing a simple bash script calling the rostopic pub command.

The algorithm was tested using different sample fields. The results obtained for three of them are displayed in Figure IV.5. The cellular decomposition consist of 10×10 cells. Table IV.1 reports the path length compared to the field area which as expected is greater for larger fields.

Field	Area	Path Length
1	$15444\,m^2 = 1.5444\,ha$	$2.17\,Km$
2	$50665\,m^2 = 5.0665\,ha$	$5.56\,Km$
3	$41586\,m^2 = 4.1586\,ha$	$4.66\,Km$

Table IV.1: Comparison between field extension and coverage path length

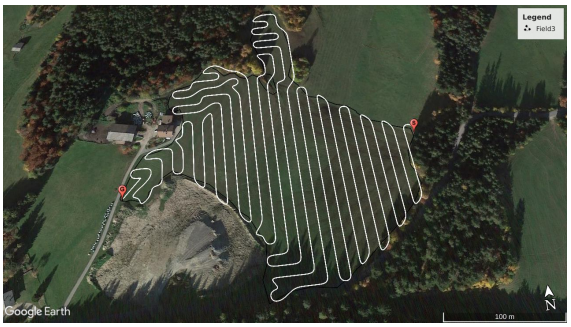
3.1 Different Starting and Goal Positions

During the design of the CPP node several starting and goal position were tested. This helped to understand how the algorithm behaves under different condition and to implement a smart selection of the starting cell. Figure IV.6 shows the outcome path under three different situations.

When the starting point is in the middle of the field, as it is in Figure IV.6b, the coverage trajectory starts as a straight line toward the furthest point from the goal position,

(a) Field 1

(b) Field 2



(c) Field 3

Figure IV.5: CPP output path applied to different shaped fields

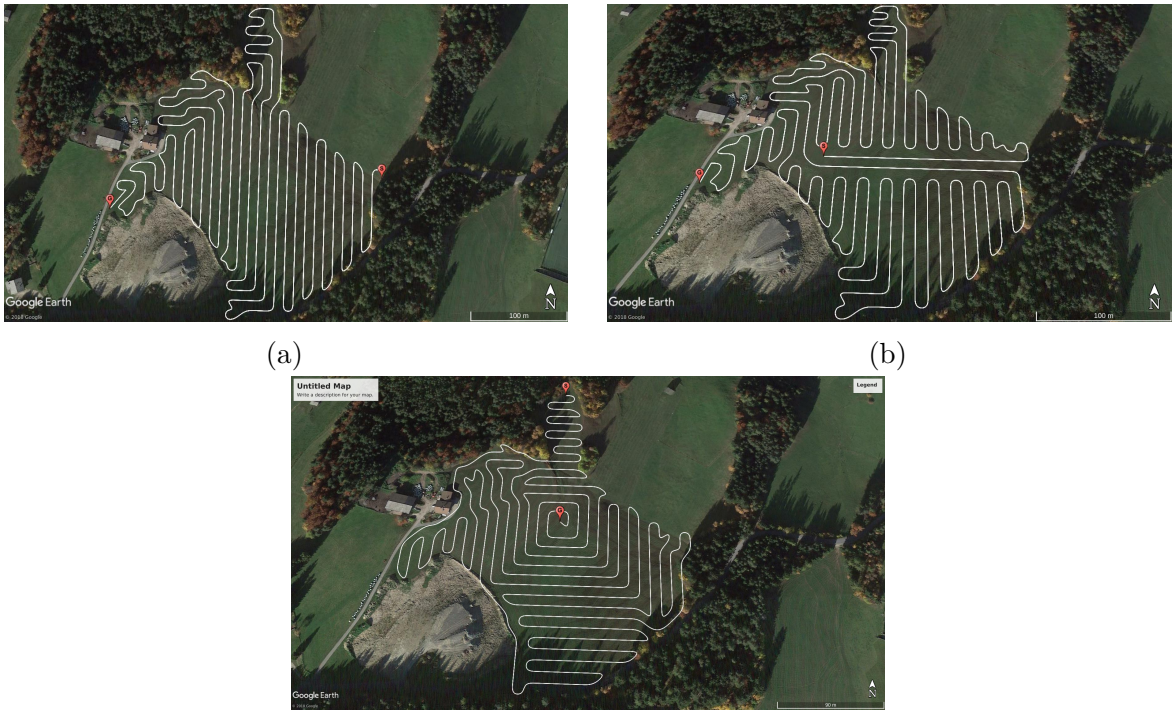
thought it is the path having the steepest ascending gradient. The cells visited in this transition are marked as visited and therefore avoided in the successive steps. This leads to have such kind of two halves partitioning in the trajectory.

The situation radically change if, at the center of the workplace, is placed the goal point. The wave-front propagate all around this position and the generated path assume the spiral like shape in Figure IV.6c. As in the previous situation the path present numerous changes of direction.

Finally in Figure IV.6a the starting cell is chosen looking for the farthest most isolated cell with the method presented in subsection 3.4. The goal position is instead at the extremity of the field and it coincides with the home position, thus the point where the UAV were placed when the mission started. This is reasonable as, in a real situation, where the grass is tall, the mission is suppose to start from the field's border. The resulting path is definitely the best of the three showing a lawnmower like behavior. The prevalence of straight lines limits speed variations, consequently acceleration and power consumption are reduced.

3.2 Errors Analysis

Looking at the three coverage paths in Figure IV.5 one can notice that in some section at the border, the trajectory exceed the boundary (represented as black line). This phenomenon is a drawback of the approximate cellular decomposition (see subsection 3.3. The algorithm, in fact, marks a cell as part of the field when the intersection between the square polygon of the cell and the field polygon exist. This cause the cells locate at the edge of the workplace to be marked as field even if just a small portion of them are actually inside the field. This approximation leads to imperfections in the path as



(a) (b)

(c)

Figure IV.6: Coverage Path generated with Starting and Goal position

shown in Figure IV.7. The larger are the cells, the greater becomes the effect because the approximation gets worse. For instance using $10 \times 10\,m$ cells the trajectory goes outside the border of a distance as high as $9\,m$.

Possible *solutions* could be: (1) Using a grid with smaller cell, thus increasing the sensor overlap; (2) Considering a threshold on the intersecting area under which the cell it is not considered as part of the field. This could lead to a not complete coverage of the target area. (3) In the algorithm handle external cells as particular case and using as waypoint not the center but a point inside the workplace.

4 Final Considerations

Overall, the generated path (assuming a smart selection of start and goal point) is satisfactory for the desired purpose even if, in some section, there are turns which would be nice to avoid. This is accentuated especially when the workplace has a complex shape (Figure IV.5b). This shortcoming is something expected, in fact, at the actual development stage, in the wave-front propagation algorithm, only the distance from the goal point is used as cost function, therefore no optimization for reducing turns has been implemented (see subsection 3.5). The distance transformation used, leads to a spiral like behavior near the goal cell (discussed in subsubsection 2.1.2), thus it is convenient to set the goal position (home position in the mission) at an extremity of the field.

Regarding the issue discussed in subsection 3.2, it could cause real problem if at the border of the field there would be obstacles risking to make the UAV crash. This, at the actual state of development, is partially addressed relying on the *object avoidance* system that has been implemented inside *PX4 Firmware* which, using potential field, try to push

(a) (b)

Figure IV.7: Errors due to Approximate Cellular Decomposition

the drone away from surrounding obstacles.

Finally, considering that the average flight time of a mid-range drone is between 15 to 20 minutes and assuming a flight mean velocity of about $4.5\,m/s$ during the mission duration, the distance it can travel is in the range of $4 - 5.4\,Km$. This distance is enough to cover a $4\,Ha$ field (Table IV.1).

Chapter V

Conclusion and Future Work

The thesis provides an implementation approach to *georeference* and *digitally encode* a geographic bounded area and to intelligently plan a *coverage path* satisfying specific requirements. All this is achieved using ROS as software framework and therefore following at best its programming philosophy (i.e. modularity and distributive computing).

The main objective of the first part (chapter II), is to give basic knowledge regarding georeferentiation and to provide a way to acquire, store and manipulate the field border as geographic feature using the standard Keyhole Markup Language (KML) format. Acquisition is done without particular difficulties thanks to Google Earth application and its graphical tools. At ROS sides it was required to write a simple Python node which, thanks to the PyKML library, parse the geographinc informations inside the KML file and convert them to a ROS message. The implementation proves to work flawlessly without particular computational overhead even for big files. The outcome, an accurate and lightweight representation of the mission environment, is the base point of the entire mission and all other component lean on it.

The second part (chapter III) is related to Coverage Path Planning (CPP), thus generates the flight trajectory for the UAV in a way that, once completed, guarantees the total coverage of the target area by the on-board sensor. Several approaches from the literature were analyzed and at the end it is explain the algorithm implemented in the project. The proposal solution is based on *wave-front* propagation algorithm that is largely used in navigation planning for mobile robot. For the required use case it was required to modify and rethink some parts of the original algorithm so to make it generate a *coverage path*. Despite there are still space to further improvements, the results are satisfactory and the path well fits the physical properties of multirotor drones. Considering, in fact, the performance obtained in simulation environment (see chapter IV), it results that the algorithm is able to generate the trajectory for all the sample fields tested, proving to be quite robust.
Nevertheless, imperfections are still present, as the error caused by the approximate cellular decomposition which leads, in some situation, the trajectory to exceed workplace boundary. Overall, considering that the problem of finding an optimal coverage path for an area of complex shape is a typical NP-hard problem and therefore, at the actual state-of-the-art it is not solvable, the algorithm presented in chapter III do not always give the optimal path, but it gives an admissible path.

This drawback is intrinsic to the chosen area decomposition approach, but it was evaluated less important then the capability of relatively easy optimization for different

path features (i.e. number of turns, terrain elevation, ecc.).

All the Bambi Project is available as opensource software, hosted at github under `https://github.com/BambiSaver`. The project wiki is available at `https://wiki.bambi.florian.world` where some technical documentation and instructions on how to setup the workspace and build the code are provided.

1 Future Works

Many different adaptations, tests, and experiments have been left for the future due to lack of time (i.e. compile the code on the raspberry required 2 hours). Future work concerns analysis of particular mechanisms, improvements, to try different methods and exhaustive on field testing.

Concerning the environment representation a great improvement could be to generate the orthophoto directly on the field. This would guarantee an up-to-dated image and eventually more accurate than Google Earth alternatives. A good starting point for this task could be to use OpenDroneMap, an open source toolkit for processing aerial drone imagery [31], which among other features provides the possibility to generate orthophoto. An important thing to evaluate is if the computing power of the Raspberry is enough to run this kind of images manipulation that is notoriously computationally expensive.

Another interesting feature, concerning image processing, would be to automatically detect the border of the field using a *neural network*. This could be done through what is called *semantic segmentation*[1], thus to label every pixel of the target area as "field pixel". In this way the mission could be completely automated requiring almost no human interaction. Such kind of neural network would require a dataset of at least 1000 images of different fields in order to be trained, therefore an huge amount of time as for each image the field must be manually recognized. After the training stage, the expected inference time on a Raspberry is around $300\,ms$ so perfectly feasible for the specific use case.

For the Coverage Path Planning (CPP), the next step in the algorithm is to modify the cost function so that it takes care of the elevation profile provided by the already implemented terrain_data_provider node and to limit at maximum the number of turns. This requires to analyze and understand how to weight every features in order to build a coherent cost function. Once understand that, the only part that has to be changed in the CPP algorithm is the wave-front propagation, i.e. where the matrix is filled according to the cost function (chapter III subsection 3.5).

[1] Semantic segmentation describes the process of associating each pixel of an image with a class label (such as flower, person, road, sky, ocean, or car).

Chapter VI

Bibliography

[1] D. W. Stiftung, "Mowing mortality in grassland ecosystems," 2011.

[2] in *ICAO. Global air traffic management operational concept*, 2005.

[3] K. S. Christie, S. L. Gilbert, C. L. Brown, M. Hatfield, and L. Hanson, "Unmanned aircraft systems in wildlife research: current and future applications of a transformative technology," *Frontiers in Ecology and the Environment*, vol. 14, no. 5, pp. 241–251. [Online]. Available: https://esajournals.onlinelibrary.wiley.com/doi/abs/10.1002/fee.1281

[4] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[5] M. Nagi Zomrawi, G. Ahmed, and M. Hussam Eldin, "Positional accuracy testing of google earth," vol. 4, pp. 6–9, 01 2013.

[6] A. D. Chapman and J. Wieczorek, *Guide to Best Practices for Georeferencing*. Copenhagen: Global Biodiversity Information Facility, 8 2006.

[7] G. S Smith, "Digital orthophotography and gis," 08 2018.

[8] S. Aronoff, "Geographic information systems: A management perspective," *Geocarto International*, vol. 4, no. 4, pp. 58–58, 1989. [Online]. Available: https://doi.org/10.1080/10106048909354237

[9] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan, "Extensible Markup Language (XML) 1.1 (Second Edition)," http://www.w3.org/TR/2006/REC-xml11-20060816/, W3C - World Wide Web Consortium, W3C Recommendation, September 2006. [Online]. Available: http://www.w3.org/TR/2006/REC-xml11-20060816/

[10] R. Andy. (2015) Xiaomi yi camera control & configure gui and via python scripts. [Online]. Available: https://github.com/deltaflyer4747/Xiaomi_Yi

[11] T. Erickson, "pykml - creating, parsing, manipulating, and validating kml in python," 2018. [Online]. Available: https://pythonhosted.org/pykml/

[12] S. Behnel and H. Joukl. (2018) lxml - xml and html with python. [Online]. Available: https://lxml.de/index.html#introduction

[13] E. M. Arkin, S. P. Fekete, and J. S. Mitchell, "Approximation algorithms for lawn mowing and milling." *Computational Geometry*, vol. 17, no. 1, pp. 25 – 50, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925772100000158

[14] S. H. G. K. W. B. L. E. K. S. T. Howie Choset, Kevin M. Lynch, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, ser. Intelligent Robotics and Autonomous Agents. The MIT Press, 2005. [Online]. Available: http://gen.lib.rus.ec/book/index.php?md5=8A32158BFD69A0E45BED460104C0D4A2

[15] T. Oksanen and A. Visala, "Coverage path planning algorithms for agricultural field machines," vol. 26, pp. 651–668, 08 2009.

[16] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon decomposition," in *International Conference on Field and Service Robotics*, January 1997.

[17] A. Zelinsky, R. Jarvis, J. C. Byrne, and S. Yuta, "Planning paths of complete coverage of an unstructured environment by a mobile robot," in *In Proceedings of International Conference on Advanced Robotics*, 1993, pp. 533–538.

[18] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1, pp. 77–98, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1016610507833

[19] A. Howard, M. Mataric, and G. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," vol. 5, 06 2002.

[20] C. Franco, D. Paesa, G. Lopez-Nicolas, C. Sagues, and S. Llorente, "Hierarchical strategy for dynamic coverage," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 5341–5346.

[21] J. Song and S. Gupta, "$\varepsilon^\star$: An online coverage path planning algorithm," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 526–533, April 2018.

[22] M. Sang-Woo and S. David Hyun-Chul, "Study on path planning algorithms for unmanned agricultural helicopters in complex environment," *International Journal of Aeronautical and Space Sciences*, vol. 2, no. 2, Nov 2009. [Online]. Available: http://dx.doi.org/10.5139/IJASS.2009.10.2.001

[23] L. H. Nam, L. Huang, X. J. Li, and J. F. Xu, "An approach for coverage path planning for uavs," in *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, April 2016, pp. 411–416.

[24] Flir, *Radiometric Dual-sensor Thermal Camera for Drones*, 2017. [Online]. Available: https://www.flir.com/globalassets/imported-assets/document/flir-duo-r-datasheet-en.pdf

[25] P. S. Foundation, "urllib — open arbitrary resources by url," 2018. [Online]. Available: https://docs.python.org/2/library/urllib.html

[26] M. R. Florian Mahlknecht, "Bambi project - an automate solution for wildlife rescue in agricultural land," 2018. [Online]. Available: https://github.com/BambiSaver/bambi

[27] J. O'Quin, "Python and c++ interfaces for manipulating geodetic coordinates," 2017. [Online]. Available: http://wiki.ros.org/geodesy

[28] M. L. A. W. Barend Gehrels, Bruno Lalande, "Boost.geometry (aka generic geometry library, ggl) defines concepts, primitives and algorithms for solving geometry problems," 2017. [Online]. Available: https://www.boost.org/doc/libs/1_65_1/libs/geometry/doc/html/geometry/

[29] X. Chen, "C++ implementation of spline interpolation," 2017. [Online]. Available: https://github.com/chen0040/cpp-spline

[30] S. Behnel and H. Joukl. (2018) Qt qml. [Online]. Available: https://doc.qt.io/qt-5.11/qtqml-index.html

[31] "Opendronemap - an open ecosystem of solutions for processing, analyzing and displaying aerial data and to build strong, self-sustaining communities around them," 2018. [Online]. Available: https://www.opendronemap.org/

[32] A. Hellmund, S. Wirges, C. Bandera, and N. O. Salscheider, "Robot operating system: A modular software framework for automated driving," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2016, pp. 1564–1570.

[33] L. Meier. Pixhawk main page. [Online]. Available: http://pixhawk.org

[34] ——. Mavlink developer guide main page. [Online]. Available: https://mavlink.io/en/

[35] ——. Mavlink packet serialization. [Online]. Available: https://mavlink.io/en/guide/serialization.html

Appendix A

ROS

"The Robot Operating System (ROS) is an opensource software framework supporting the development of complex, but modular systems in a distributed computing environment. While the core components of ROS are highly generic, the primary focus of ROS and its ecosystem is set to the development and research of robots. The performance critical parts of the framework are written in C++, but applications operating on top of the framework may currently be written in C++, Python or Lisp."[32]

1 Concepts

Master It provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.

Nodes are processes that perform computation. ROS is designed to be *modular* at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, one Node provides a graphical view of the system, and so on. A ROS node is written with the use of a ROS client library, such as roscpp or rospy

Parameter Server It allows data to be stored by key in a central location. It is currently part of the Master.

Messages Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs).

Topics Messages are routed via a transport system with publish / subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each others' existence. The idea is to decouple the production of

information from its consumption. Logically, one can think of a topic as a strongly typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages as long as they are the right type.

Services The publish / subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.

Bags A format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

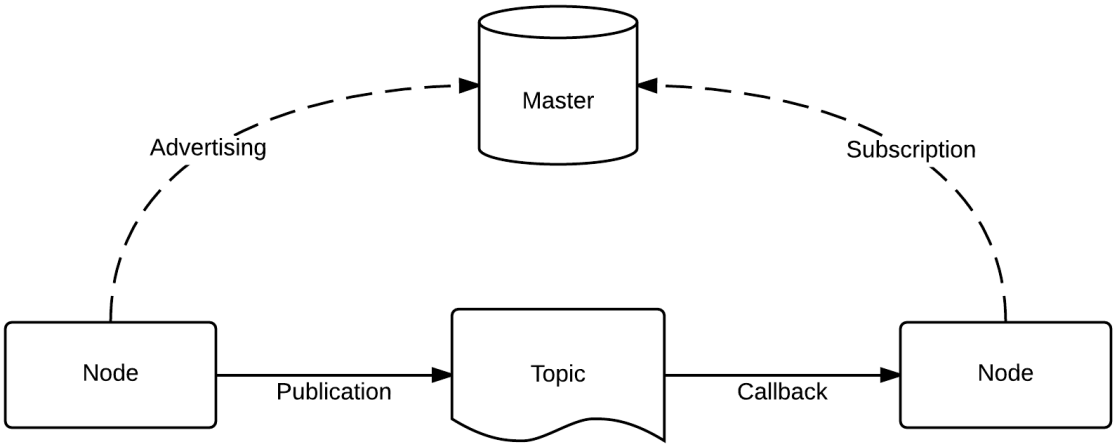


Figure A.1: ROS Nodes communication

Appendix B

Pixhawk Autopilot

1 Hardware

"Pixhawk is an independent, open-hardware project aiming at providing high-end autopilot hardware to the academic, hobby and industrial communities at low costs and high availability. It provides hardware for the Linux Foundation DroneCode project. It originated from the PIXHAWK Project of the Computer Vision and Geometry Lab of ETH Zurich (Swiss Federal Institute of Technology) and Autonomous Systems Lab as well from a number of excellent individuals." [33]

Figure B.1

The Pixhawk hardware weights 38g and it is provided with a 32-bit ARM Cortex M4 core with FPU with 256 KB of RAM and a 32-bit fail-safe co-processor; it is also equipped with a compass, a barometer, an accelerometer and a gyro sensor.

2 Softwere

Modern, sophisticated flight controllers share a commonality in architecture. We can divide their functionality into three distinct layers, illustrated in Figure B.2.

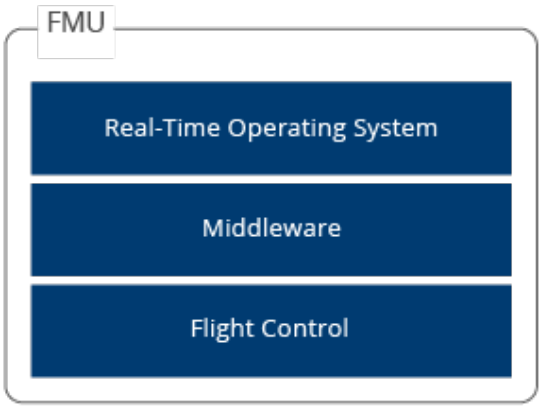


Figure B.2: The architecture of a modern flight controller

Layer 1: Real Time Operating System

The real time operating system is the back bone of the flight firmware, providing basic hardware abstraction and concurrency. Real time systems are critical for flight control performance and safety, as they guarantee that flight control tasks will be completed in a certain amount of time, and are essential for the safety and time-critical performance of UAVs. Luci uses a real time operating system called NuttX, which is highly expansive and configurable.

Layer 2: Middleware

The middleware is a collection of tools, drivers, and libraries that relate to flight control. It contains device drivers that handle sensors and other peripherals. It also contains flight control libraries such as RC protocols, math utilities, and control filters.

Layer 3: Flight Control

The flight control layer is the brains of the operation; this layer contains all of the command and control routines. Things like state estimation, flight control, system calibration, telemetry, motor control, and other flight control aspects reside in this layer.

2.1 PX4 Stacks

The Pixhawk platform can be flashed with two very popular flight stack: ArduPilot and PX4. In this thesis it has been adopted the PX4 software because:

- Its software-in-the-loop (SITL) simulation is much more developed and matured.
- Supports a much larger number of peripherals, including more IMU sensors, lidar, range finders, status indicators, optical flow, and motion capture units. PX4 supports the most advanced sensing peripherals for drones.
- Contains advanced command and control functionality, including things like terrain estimation, and indoor flight correction.
- More ubiquitous and built with advanced drone applications in mind. It can be compiled for POSIX (Linux) systems, and it can also integrate with ROS to run

flight applications in a hybrid system, with some running on an underlying real-time OS, and others running on Linux using ROS to communicate.

The choice was mainly driven by the more developed SITL environment and framework provided by the PX4 community and for the more advanced integration with ROS rather then a matter of performance or features, where ArduPilot stack prove to be good as well. The diagram in Figure B.3 provides a detailed overview of the building blocks of PX4. The top part of the diagram contains middleware blocks, while the lower section shows the components of the flight stack.

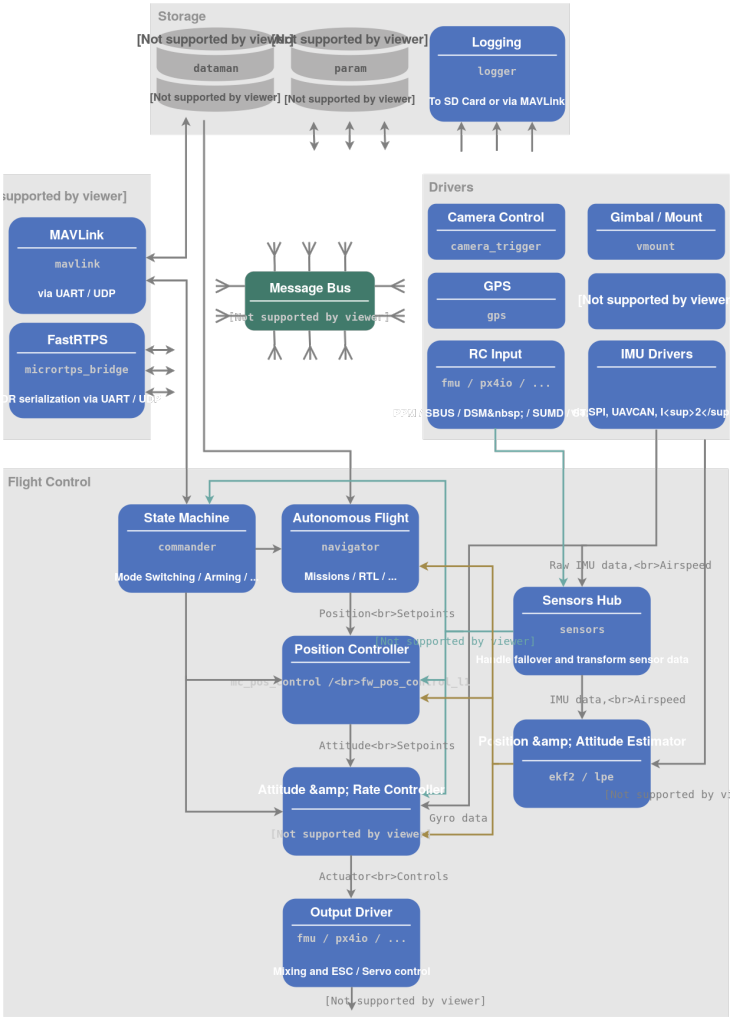


Figure B.3: PX4 Architecture

Appendix C

MAVLink Protocol

"MAVLink is a very lightweight messaging protocol for communicating with drones (and between onboard drone components).
MAVLink follows a modern hybrid publish-subscribe and point-to-point design pattern: Data streams are sent / published as topics while configuration sub-protocols such as the mission protocol or parameter protocol are point-to-point with retransmission.
Messages are defined within XML files. Each XML file defines the message set supported by a particular MAVLink system, also referred to as a "dialect". The reference message set that is implemented by most ground control stations and autopilots is defined in common.xml (most dialects build on top of this definition).
The MAVLink toolchain uses the XML message definitions to generate MAVLink libraries for each of the supported programming languages. Drones, ground control stations, and other MAVLink systems use the generated libraries to communicate. These are typically MIT-licensed, and can therefore be used without limits in any closed-source application without publishing the source code of the closed-source application." [34]

Key Features

- Very efficient. MAVLink 1 has just 8 bytes overhead per packet, including start sign and packet drop detection. MAVLink 2 has just 14 bytes of overhead (but is a much more secure and extensible protocol). Because MAVLink doesn't require any additional framing it is very well suited for applications with very limited communication bandwidth.
- Very reliable. MAVLink has been used since 2009 to communicate between many different vehicles, ground stations (and other nodes) over varied and challenging communication channels (high latency/noise). It provides methods for detecting packet drops, corruption, and for packet authentication.
- Supports many programming languages, running on numerous microcontrollers/operating systems (including ARM7, ATMega, dsPic, STM32 and Windows, Linux, MacOS, Android and iOS).
- Allows up to 255 concurrent systems on the network (vehicles, ground stations, etc.)
- Enables both offboard and onboard communications (e.g. between a GCS and drone, and between drone autopilot and MAVLink enabled drone camera).

In a second version of the MAVLink protocol was release mainly to allow more then 256 message IDs as it was in the first version. In addition MAVLink 2 is backward-compatible and implement package signing (authentication) which greatly improves security. Basic frame is shown in Figure C.1 and each fields is described in Table C.1. There are several MAVLink messages. Understanding them is vital for sending proper commands to the mav.

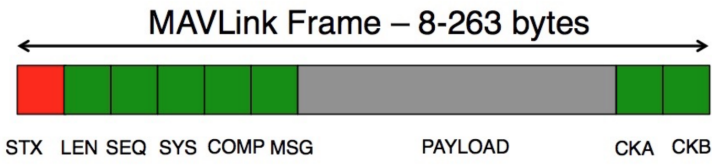


Figure C.1: MAVLink packet [35]

1 MAVLink Messages

It follows a short list of the most commonly used MAVLink messages type:

- MAVLINK_MSG_ID_REQUEST_DATA_STREAM
  This message is used to request the stream of data from the autopilot. Requested data can be sensors, RC channels, GPS position, status or the combination of them.
- MAVLINK_MSG_ID_COMMAND_LONG
  This message is used to give commands to the autopilot. Several commands are supported.
- SET_MODE
  It sets the different mode of operations for the drone. Few supported mode in PX4 flight stack are: Position; Altitude; Manual/Stabilize; Acro; Auto: Takeoff, Land, Hold, Return, Mission, Follow me, Offboard.

In the *Bambi Project* it was necessary to define a custom MAVLink named MAV_CMD_BAMBI_ START_STOP_MISSION whose XML definition is given in Listing C.1. This message is sent by the ground station (QGC) to the aircraft and cause the mission state to change.

```
1  <?xml version="1.0"?>
2  <mavlink>
3  <version>3</version>
4  <dialect>0</dialect>
5    <enum name="MAV_CMD">
6      <description>
7        Commands to be executed by the MAV. They can be executed↩
             on user request, or as part of a mission script.
8      </description>
9      <entry value="2720" name="MAV_CMD_BAMBI_START_STOP_MISSION↩
         ">
```

```
10      <description>
11        Request to start/stop Bambi mission: it also contain ↩
            the coordinate of the first mission point
12      </description>
13      <param index="1">Trigger Mission (0: stop, 1: start)</↩
          param>
14      <param index="2">AltitudeTO (in meters, relative to home↩
          )</param>
15      <param index="3">Latitude of orthophoto (WGS84)</param>
16      <param index="4">Longitude of orthophoto (WGS84)</param>
17      <param index="5">Altitude of orthophoto (in meters AMSL)↩
          </param>
18      <param index="6">Altitude of scanning (in meters, ↩
          relative to ground)</param>
19      <param index="7">Sensor Footprint min Dimension (in ↩
          meters)</param>
20    </entry>
21  </enum>
22 </mavlink>
```

Listing C.1: MAVLink message definition of BAMBI_ START_STOP_MISSION command

Byte Index	**Content**	**Value**	**Description**
0	Start byte of package	0xFD	Protocol-specific start-of-text (STX) marker used to indicate the beginning of a new packet. Any system that does not understand protocol version will skip the packet.
1	Payload length	0 - 255	Indicates length of the following payload section.
2	Incompatibility Flags		Flags that must be understood for MAVLink compatibility (discards packet if it does not understand flag).
3	Compatibility Flags		Flags ignored if not understood (implementation handle packet even if it does not understand flag).
4	Packet sequence number	0 - 255	Used to detect packet loss. Components increment value for each message sent.
5	System ID (sender)	1 - 255	ID of system sending the message. Used to differentiate systems on network.
6	Component ID (sender)	0 - 255	ID of component sending the message. Used to differentiate components in a system (e.g. autopilot and a camera).
7 to 9	Message ID (low, middle, high bytes)	0 - 16777215	ID of message type in payload. Used to decode data back into message object.
10 to (n+10)	Payload		Message data. Depends on message type (i.e. Message ID) and contents.
(n+11) to (n+12)	Checksum (low byte, high byte)		X.25 CRC for message (excluding magic byte). Includes CRC_EXTRA byte.
(n+12) to (n+26)	Signature		(Optional) Signature to ensure the link is tamper-proof.

Table C.1: Mavlink Packet Field Description [35]

Appendix D

Mavros

Mavros package implements a MAVLink extendable communication node for ROS with UDP proxy for Ground Control Station that includes the following features:

- Communication with autopilot via serial port
- UDP proxy for Ground Control Station
- Plugin system for ROS-MAVLink translation
- Parameter manipulation tool
- Waypoint manipulation tool'

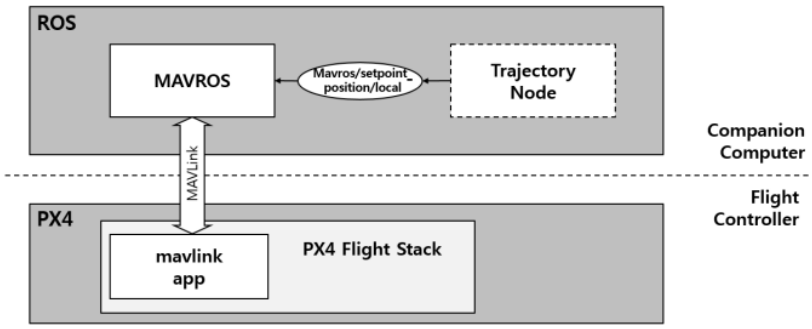


Figure D.1: Mavros as communication gateway

Through mavros it is possible to communicate with the flight controller (Figure D.1) simply publishing over the topic subscribed by mavros or making a call to the services it provides. The package is made up of various plugins, each handling different components of the FCU. Every plugin can be load and configure separately when starting mavros through *launch* file. Inside the package there are already some sample launch files specifically created to configure the communication with PX4 or APM flight stack.

1 List of Figures

2 List of Tables

List of Algorithms

3 Listings

4 Index