Alma Mater Studiorum · Università di Bologna

Scuola di Ingegneria e Architettura

Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione
"Guglielmo Marconi" – DEI

Corso di Laurea in Ingegneria dell'Automazione

Agricultural Field Detection and Path Planning for an Unmanned Aerial Vehicle

Progetto di Laurea

Relatore:
Chiar.mo Prof. Lorenzo Marconi

Correlatore:
Prof. Nicola Mimmo

Presentato da:
Michael Rimondi

II Sessione
Anno Accademico 2017/2018

Todo list

Abstract

Maecenas accumsan dapibus sapien. Duis pretium iaculis arcu. Curabitur ut lacus. Aliquam vulputate. Suspendisse ut purus sed sem tempor rhoncus. Ut quam dui, fringilla at, dictum eget, ultricies quis, quam. Etiam sem est, pharetra non, vulputate in, pretium at, ipsum. Nunc semper sagittis orci. Sed scelerisque suscipit diam. Ut volutpat, dolor at ullamcorper tristique, eros purus mollis quam, sit amet ornare ante nunc et enim.

Keywords: One, Two

Abstract (italiano)

Phasellus fringilla, metus id feugiat consectetuer, lacus wisi ultrices tellus, quis lobortis nibh lorem quis tortor. Donec egestas ornare nulla. Mauris mi tellus, porta faucibus, dictum vel, nonummy in, est. Aliquam erat volutpat. In tellus magna, porttitor lacinia, molestie vitae, pellentesque eu, justo. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Sed orci nibh, scelerisque sit amet, suscipit sed, placerat vel, diam. Vestibulum nonummy vulputate orci. Donec et velit ac arcu interdum semper. Morbi pede orci, cursus ac, elementum non, vehicula ut, lacus. Cras volutpat. Nam vel wisi quis libero venenatis placerat. Aenean sed odio. Quisque posuere purus ac orci. Vivamus odio. Vivamus varius, nulla sit amet semper viverra, odio mauris consequat lacus, at vestibulum neque arcu eu tortor. Donec iaculis tincidunt tellus. Aliquam erat volutpat. Curabitur magna lorem, dignissim volutpat, viverra et, adipiscing nec, dolor. Praesent lacus mauris, dapibus vitae, sollicitudin sit amet, nonummy eget, ligula.

Keywords: Uno, Due

Contents

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 problems of *agricultural field detection and representation* and *Coverage Path Planning*

1 Bambi Project

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]. The BambiSaver project was born aiming to provide an autonomous, fast and user friendly device able to search and locate, living creatures in agricultural areas. 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 a UAV (Unmanned Aerial Vehicle) [2] and more precisely a quad-copter equipped with a thermal camera. An aerial vehicle is, in fact, 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. Ever since the 1950s, the importance of silage meadows has increasingly taken precedence over the traditional hay harvest.

1.3 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

1.4 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.

Another practical approach is the one adopted in a German wildlife rescue project which consists in deploying small aerial drones to find young deer hiding in tall grass.

2 UAV Usage for Wildlife Research

During last years, the increasing interest and development of UAS (Unmanned Aircraft System) 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 we reviewed recorded minimal or no visible behavioral responses to UAS; however, UAS 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

In the following section it is briefly explained the goals and design guidelines of the project.

3.1 Project Goals

The scope of the Bambi project is to design an integrated system capable to autonomously handle every steps of the mission so that it can be operated also by untrained personals. This must be guaranteed even in adverse environments, especially in 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. All this guidelines have been taken into account during the design and implementation stages (i.e. ROS as framework for the on-board computer).

????the idea was developed having the mountain of Trentino Altoadige in mind????

sinonimo per dire idee alla

3.2 Software Architecture concept

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 five nodes implementing every mission component as an independent module. Figure I.2 displays all the nodes and

the related topics used to let them communicate.
In the following chapter the thesis will focus on 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.

Da fare una intro ai due problemi

introdurre la parte che e' invece discussa nell'altra tesi??

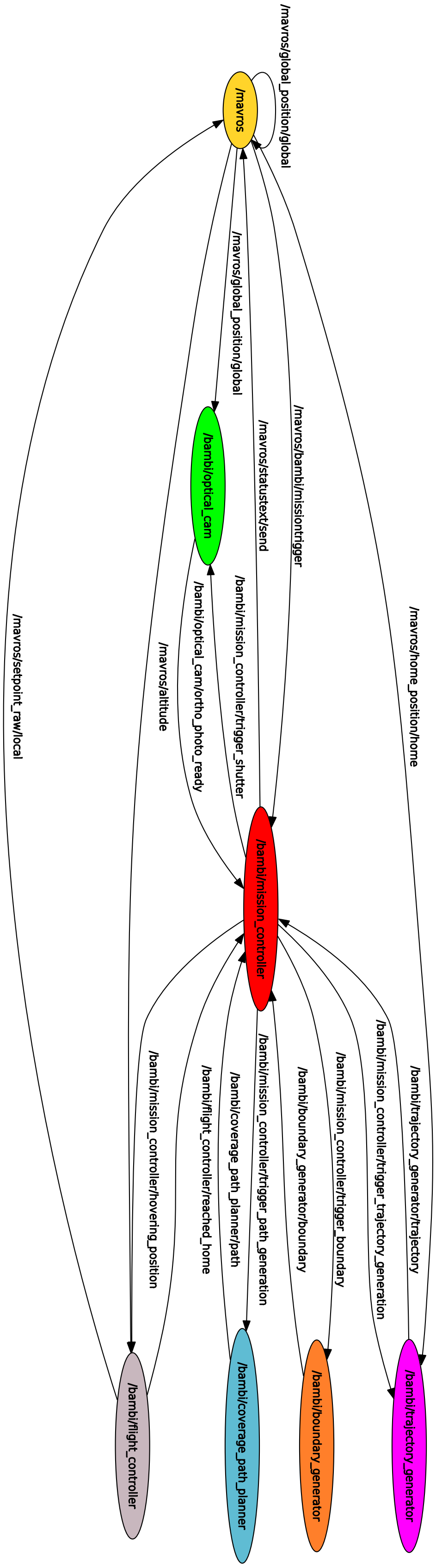

/mavros
/mavros/global_position/global
/mavros/global_position/global
/bambi/optical_cam
/mavros/statustext/send
/mavros/bambi/missiontrigger
/mavros/home_position/home
/mavros/setpoint_raw/local
/mavros/altitude
/bambi/optical_cam/ortho_photo_ready
/bambi/mission_controller/trigger_shutter
/bambi/mission_controller
/bambi/mission_controller/hovering_position
/bambi/flight_controller/reached_home
/bambi/coverage_path_planner/path
/bambi/mission_controller/trigger_path_generation
/bambi/boundary_generator/boundary
/bambi/mission_controller/trigger_boundary
/bambi/mission_controller/trigger_trajectory_generation
/bambi/trajectory_generator/trajectory
/bambi/flight_controller
/bambi/coverage_path_planner
/bambi/boundary_generator
/bambi/trajectory_generator

Figure I.2: Nodes and Topics graph

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) and it is designed for positioning anywhere on Earth.

better explain WGS84 ref system???

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

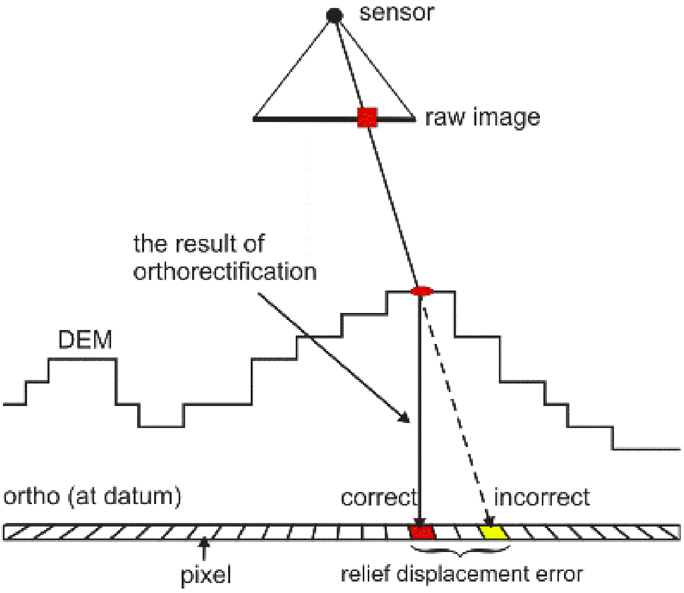


Figure II.1: Orthorectification

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 module. .

sinonimo di module???

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, including hyperlinks and embedded images
- Organize 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.

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 a 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.

Listing II.1: Border in KML

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
```

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

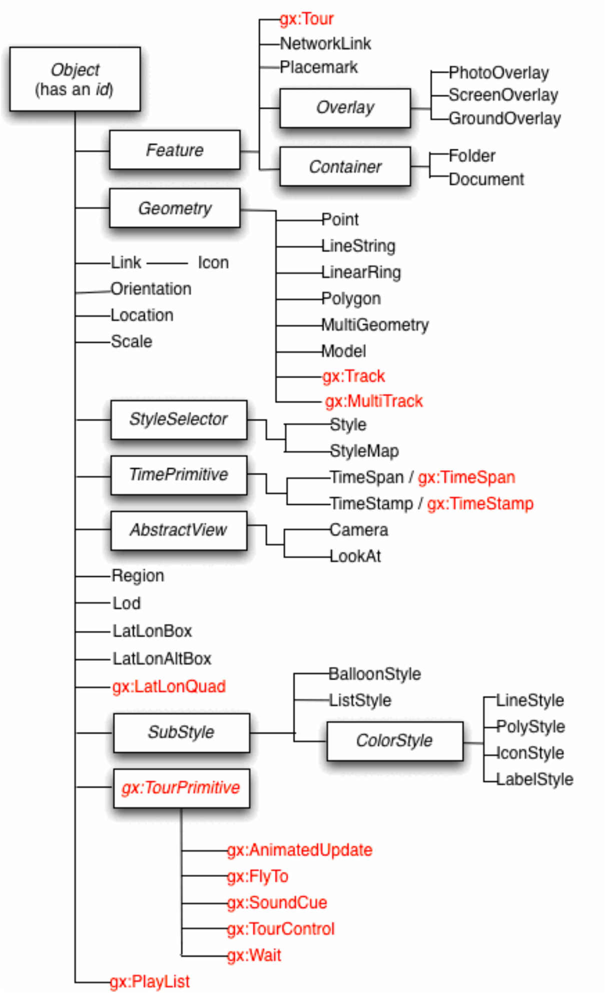


Figure II.2: KML Object Hierarchy Diagram

```
  <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>
```

Using the *name* and *description* tags gives 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.

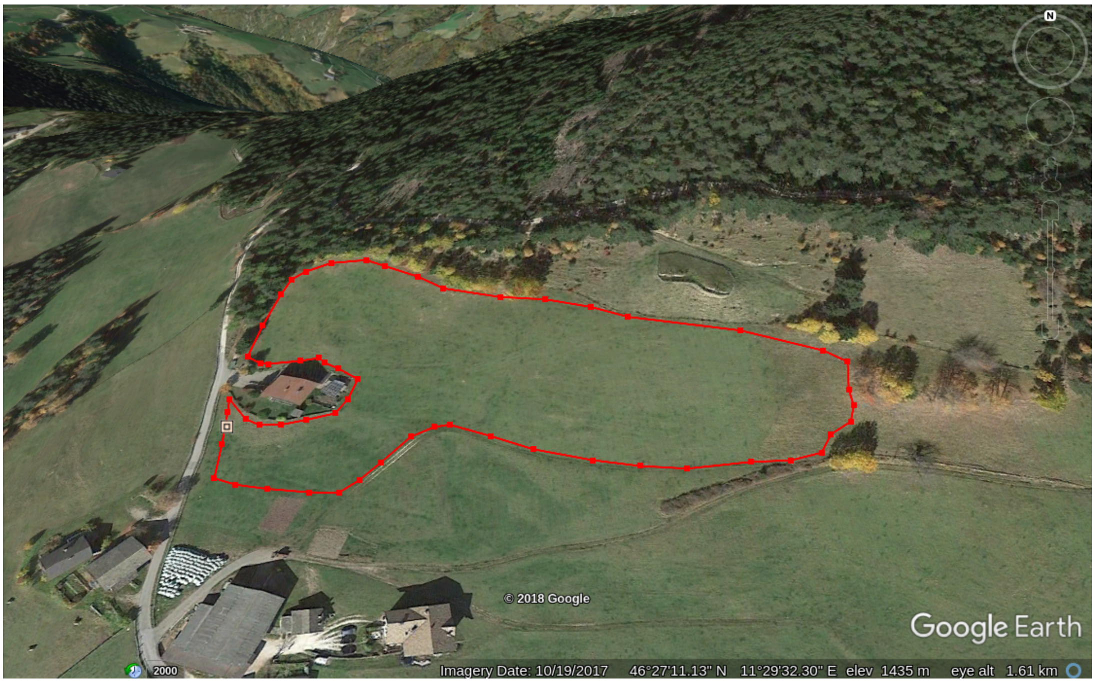

N
© 2018 Google
Google Earth
2000
Imagery Date: 10/19/2017 46°27'11.13" N 11°29'32.30" E elev 1435 m eye alt 1.61 km

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

3 Implementation

As it was accennato in 1.2 although the source of the orthophoto used in the software implementation is Google Earth, in defining the mission it has been taken into account the possibility to ??shut?? a photo of the whole field and georeference it (that is the goal of */optical_cam* node). This would provide a ??more?? up-to-dated 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 */mission_controller*, */boundary_generator* and), 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.

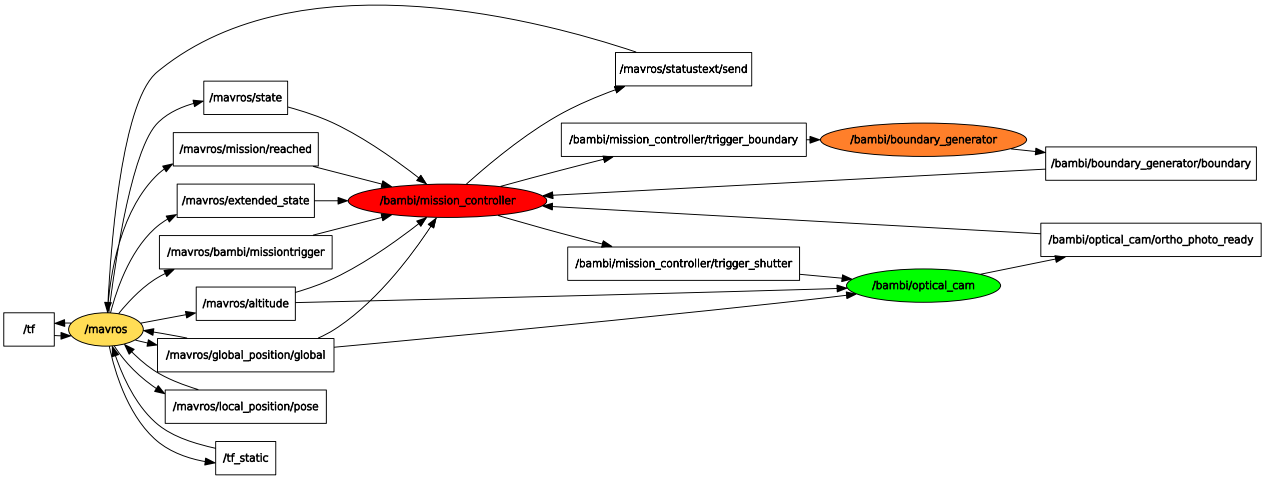

/mavros/statustext/send
/mavros/state
/bambi/mission_controller/trigger_boundary
/bambi/boundary_generator
/bambi/boundary_generator/boundary
/mavros/mission/reached
/mavros/extended_state
/bambi/mission_controller
/bambi/optical_cam/ortho_photo_ready
/mavros/bambi/missiontrigger
/bambi/mission_controller/trigger_shutter
/bambi/optical_cam
/mavros/altitude
/tf
/mavros
/mavros/global_position/global
/mavros/local_position/pose
/tf_static

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

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 the receiving (??reception??) of the *trigger_shutter* message from 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. 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

QUI INSERIRE LISTING DELLA DEFINIZIONE DEI MESSAGGI USATI DAI DUE NODI

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.2. 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 ??leaved?? to future developments.

Listing II.2: take_photo() function in python

```python
#! /usr/bin/env python
# encoding: utf-8
import os, re, sys, time, socket, urllib, datetime
from settings import camaddr
from settings import camport
import rospy
def take_photo():
  # socket initialization
  srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  srv.settimeout(5)
  rospy.loginfo(’Trying to connect with yiCam @ ’ + str(camaddr) + ’:’ + str(camport))
  srv.connect((camaddr, camport))
  # sending access token
  srv.send(’{"msg_id":257,"token":0}’)
  # receiving data and looking for token number to be use in next request messages
  data = srv.recv(512)
  rospy.loginfo(’Got first data from yiCam, looking for "rval": ’ + str(data))
  if "rval" in data:
      token = re.findall(’"param":\s*(.+)\s*}’,data)[0]
  else:
   data = srv.recv(512)
   rospy.loginfo('Got second data from yiCam, looking for "rval": ' + str(data))
   if "rval" in data:
       token = re.findall('"param":\s*(.+)\s*}',data)[0]
# send shutter command
tosend = '{"msg_id":769,"token":%s}' %token
srv.send(tosend)
#receiving response message
data = srv.recv(512)
# parsing response message looking for photo file name
findCollection = list()
maxTries = 3
i = 0
while len(findCollection) == 0 and i < maxTries:
 data = srv.recv(512)
 rospy.loginfo('Got data from yiCam: ' + str(data))
 findCollection = re.findall('"param":"/tmp/fuse_d/(.+)"}', data)
 ++i
if len(findCollection) == 0:
 rospy.logwarn("Yi cam photo ERROR")
 return ''
yiCamFileName = findCollection[0]
# preparing URL to fetch image
url = "http://" + str(camaddr) + "/" + yiCamFileName
fileName = '$BAMBI_OWNCLOUD_HOME/yi-cam-pic-' + datetime.datetime.now().strftime('%Y-%m-%d-%H:%M:%S') + '.jpg'
fileName = os.path.expandvars(fileName)
rospy.loginfo(url)
# retrieve image from URL
urllib.urlretrieve(url, filename=fileName)
return fileName
```

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.3.
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. It is based on the *lxml.objectify* API [4] which provides access to XML documents

[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.[11]

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 . At this point it is just 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 completing the node's job.

assicurarsi si possa riferirsi cosi'

Listing II.3: boundary generator node in python

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import roslib
5  roslib.load_manifest('bambi_msgs')
6  import rospy
7  from bambi_msgs.msg import OrthoPhoto, Field, GeoPosition2D
8  import sys
9  # parse kml files
10 from pykml import parser
11 class BoundaryGeneratorNode():
12
13    def __init__(self):
14       self.m_boundaryPublisher = rospy.Publisher("~boundary", Field,
            queue_size=5)
15       rospy.Subscriber('/bambi/mission_controller/trigger_boundary',
            OrthoPhoto, self.cb_boundary_trigger)
16       rospy.spin()
17
18    def cb_boundary_trigger(self, OrthoPhoto):
19       field = Field();
20       rospy.loginfo("Trying to read file %s", OrthoPhoto.
            filenameWithFullPath)
21       with open(OrthoPhoto.filenameWithFullPath) as f:
22          root = parser.parse(f).getroot()
23          coordinates = root.Document.Placemark.LineString.coordinates.text.
               split()
24          for c in coordinates:
25             splitted = c.split(',')
26             lon = splitted[0]
27             lat = splitted[1]
28             pos = GeoPosition2D();
29             pos.latitude = float(lat)
30             pos.longitude = float(lon)
31             field.boundary_path.append(pos)
32       rospy.loginfo("Publishing field with %d coordinates", len(field.
            boundary_path))
33       self.m_boundaryPublisher.publish(field);
34 # Main function.
35 if __name__ == '__main__':
36    # Initialize the node and name it.
37    rospy.init_node('boundary_generator')
38    rospy.loginfo("BoundaryGenerator STARTUP")
39    # Go to class functions that do all the heavy lifting. Do error checking.

40    try:
41       boundaryGeneratorNode = BoundaryGeneratorNode()
```

```
    except rospy.ROSInterruptException: pass
```

.

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 regular 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] [12]. 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 [13]. 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 [14]. 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.

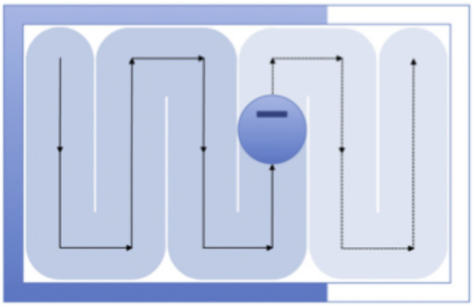


Figure III.1: Lawnmower pattern

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 [15] 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.

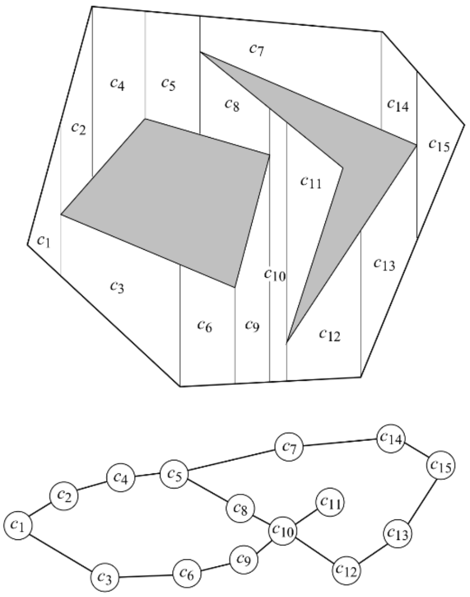


Figure III.2: Trapezoidal decomposition and adjacent graph [13]

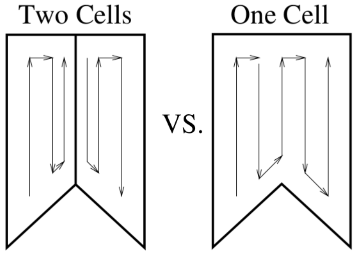


Figure III.3: Fewer cells is better

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 [16] and *Minimum-Spanning-Tree* (MST) method [17].

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" [16].

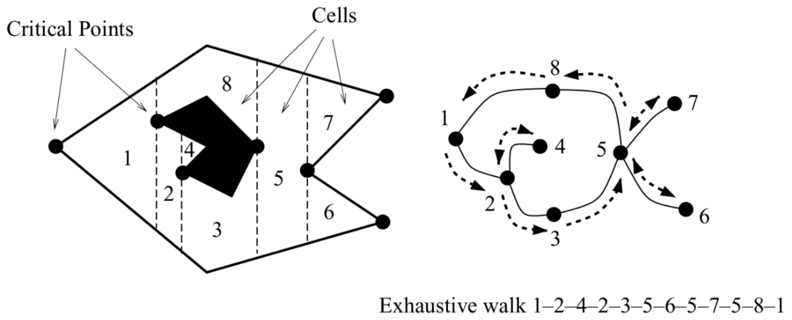
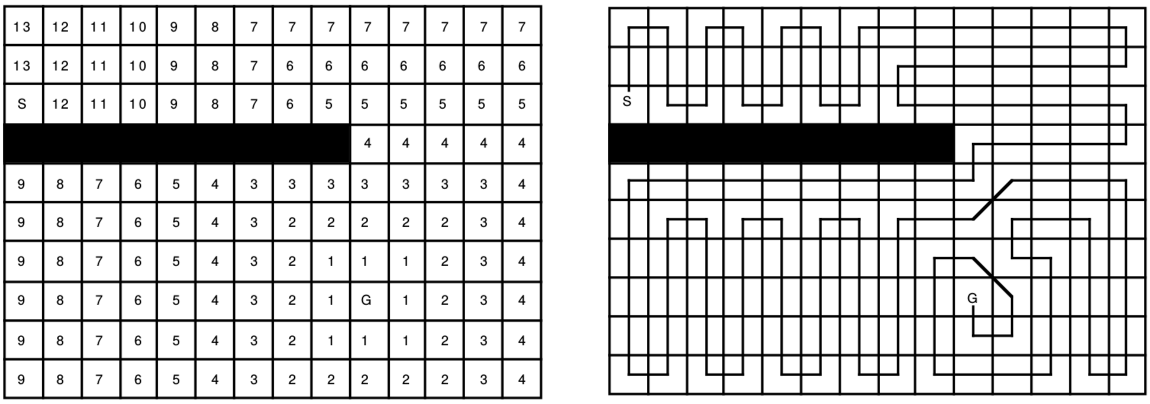


Figure III.4: Fewer cells is better

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 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 to this problem as will be presented in the next sections, could be to take into account other factors than only the distance from the goal.



13	12	11	10	9	8	7	7	7	7	7	7	7	7
13	12	11	10	9	8	7	6	6	6	6	6	6	6
S	12	11	10	9	8	7	6	5	5	5	5	5	5
■	■	■	■	■	■	■	■	■	4	4	4	4	4
9	8	7	6	5	4	3	3	3	3	3	3	3	4
9	8	7	6	5	4	3	2	2	2	2	2	3	4
9	8	7	6	5	4	3	2	1	1	1	2	3	4
9	8	7	6	5	4	3	2	1	G	1	2	3	4
9	8	7	6	5	4	3	2	1	1	1	2	3	4
9	8	7	6	5	4	3	2	2	2	2	2	3	4

(a) Distance waveform grid

(b) Path of complete coverage

Figure III.5: Wavefront method [16]

valutare se mettere l'algoritmo qui o dopo

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 $DT \leq$ 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.

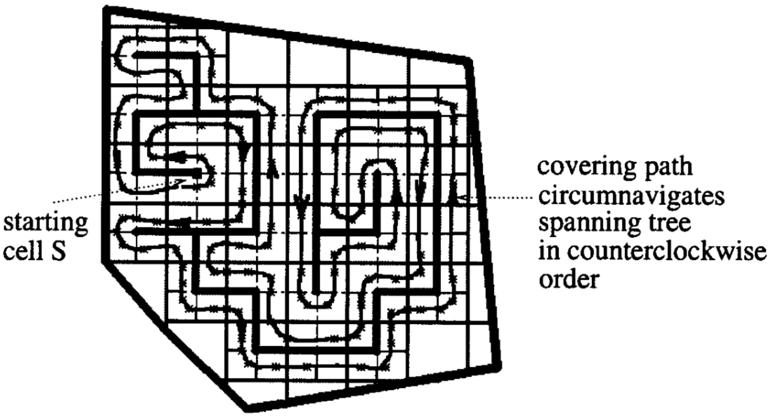


Figure III.6: Minimum Spanning Tree method

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 [18] 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 [19].
DA FAREREEEEEEJGFJDIJDIJFIJDIJIJD

A recent work by Song in [20] 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 Fig. 1, the algorithm utilizes an Exploratory Turing Machine (ETM), that consists of a twodimensional 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. While, by default, the ETM uses the lowest level of MAPS for generating the coverage path online, it switches to higher levels as needed to escape from a local extremum. The ETM acts as a supervisor to the autonomous vehicle and guides it with adaptive navigation commands.

2.2 Aerial Coverage using UAVs

In aerial coverage the CPP methods presented in subsection 2.1 have been largely adopted. 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 [21].
- This approach provide the possibility to implement different cost function, thus it is possible to obtain different behaviors (i.e. Taking care of the ground elevation

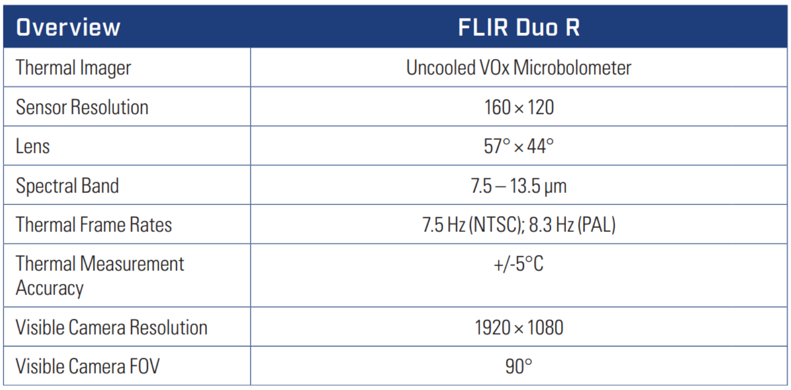
profile).

3.1 Workplace Sampling

The workplace is decomposed through approximate cellular decomposition. In this method as explained before, 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 [22]. The main specification of the camera are listed in Figure III.7.

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:



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.7: Flir Duo R thermal camera specs [22]

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

Where:

I_t: target dimension
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

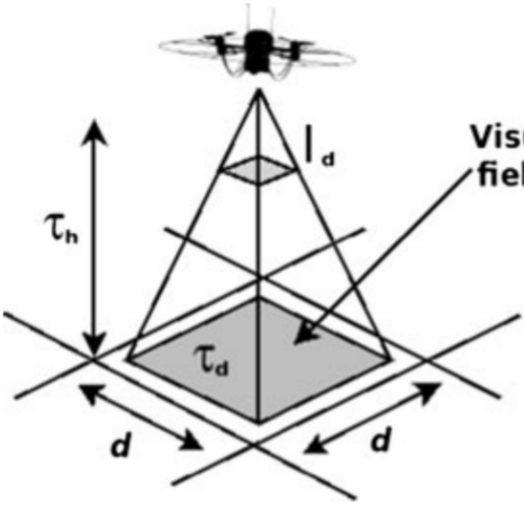


Figure III.8: UAV altitude and the image requirements

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.9 display the nodes and topics related to the CPP procedure. The scope of the other nodes that appear in the graph are:

/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 to Google elevation databases using the Google elevation API. The inquiry is done in the form of an URL request through the python library *urllib* [23].

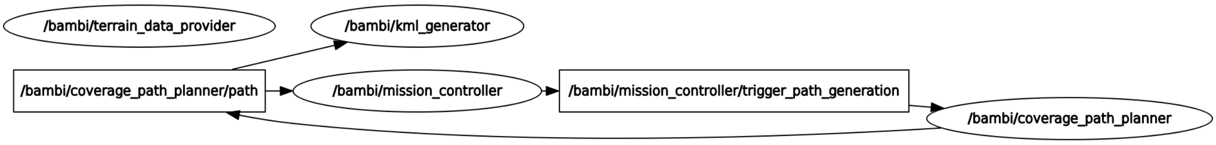

/bambi/terrain_data_provider
/bambi/kml_generator
/bambi/coverage_path_planner/path
/bambi/mission_controller
/bambi/mission_controller/trigger_path_generation
/bambi/coverage_path_planner

Figure III.9: ROS-graph of CPP related nodes and topics

/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) Wave-front Propagation Algorithm; (3) Choosing Starting Position (4) Coverage Path Generator; (5) Spline Interpolation;

3.2.1 Relevant Messages Definition

In the following sections, the specification of each sub-task is explained as well as its implementation.

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 as discussed before, 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 ??living??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.

existing, living??

3.3.1 Implementation

La questione della conversione delle coordinate

3.4 Wave-front Propagation Algorithm

Being G the grid graph, where each element represent 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[2]. Once the cost has been assigned to each cell the

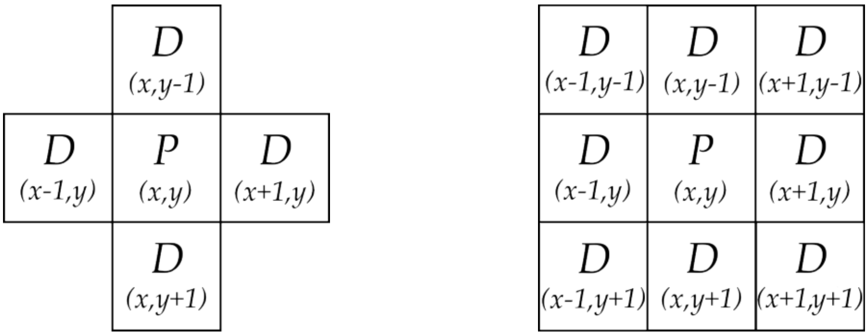
[2] In the actual implementation only distance transformation has been taken into account as cost function.

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 $\leftarrow$ fieldPoly.rightBorderE $-$ fieldPoly.leftBorderE ;
height $\leftarrow$ fieldPoly.topBorderN $-$ fieldPoly.bottomBorderN ;
/* Computes the matrix dimensions                          */
nE $\leftarrow$ ceil(width/cellDimension) ;
nN $\leftarrow$ ceil(height/cellDimension) ;
Define M as a nN $\times$ nE-matrix;
forall x $\in$ M do
    fieldPoly $\leftarrow$ getPolygonFromMatrixCell(fieldPoly.bottomBorderN,
     fieldPoly.leftBorderE, x, cellDimension) ;
    if fieldPoly $\cap$ cellPoly $> 0$ then /* cell x intersect fieldP */
        M (x) $\leftarrow$ Field;
    else /* cell x do not intersect fieldP                 */
        M (x) $\leftarrow$ Empty;
```

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.10a) and Moore neighborhood (Figure III.10b). 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.10: III.10a Von Neumann Neighborhood III.10b 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.

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

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

3.4.1 Implementation

3.5 Choosing Starting Position

The

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 weight :

1. The cost function value associated to that cell
2. How many adjacent cells have the same value, prediligendothe neighbor which have the lower number of same value adjacent cells. This is to avoid as much as possible to reach a local minimum which cause the algorithm to stack.

weight / importance??

trovare termine + capire meglio questione del local mini

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 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;
```

3.6.1 Implementation

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 it is not

3.7.1 Implementation

4 Simulation Results

Qui mostrero' l'output dell'algoritmo rappresentato come tracciato su google earth. verranno messi a confronto diverse scelte di starting e goal point e discutero' delle performance ottenute con relativi problemi da risolvere

Mostrare importanza della scelta della starting position + bezier vs spline interpolation

Chapter IV

Conclusion and Future Work

CONCLUSIONE ANCORA DA FARE All the Bambi Project is available as open source so ware, hosted at github under `https://github.com/BambiSaver` Some documentation is published on the special purpose media-wiki under https://wiki.bambi.florian.world .

1 Future Work

- Automatic field detection through IA image recognition
- improve CPP algorithm so that it takes care of elevation information and minimize turns/path length
- OTHER stuff

Chapter V

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

[12] 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

[13] 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

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

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

[16] 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.

[17] 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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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

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

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

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

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

Appendix A

ROS

TODOTODOTODO

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." [24]

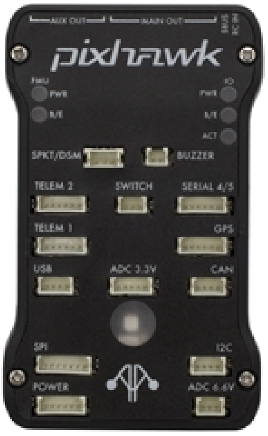


Figure B.1

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.

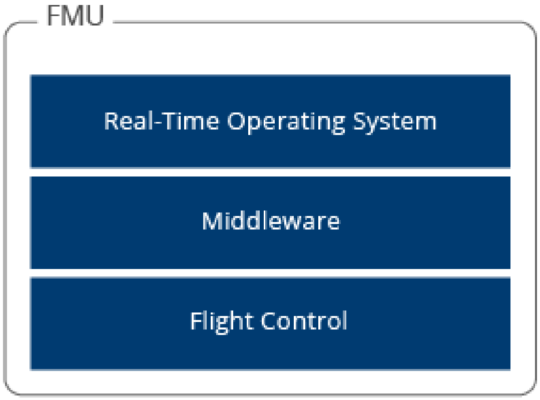


Figure B.2: The architecture of a modern flight controller

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.

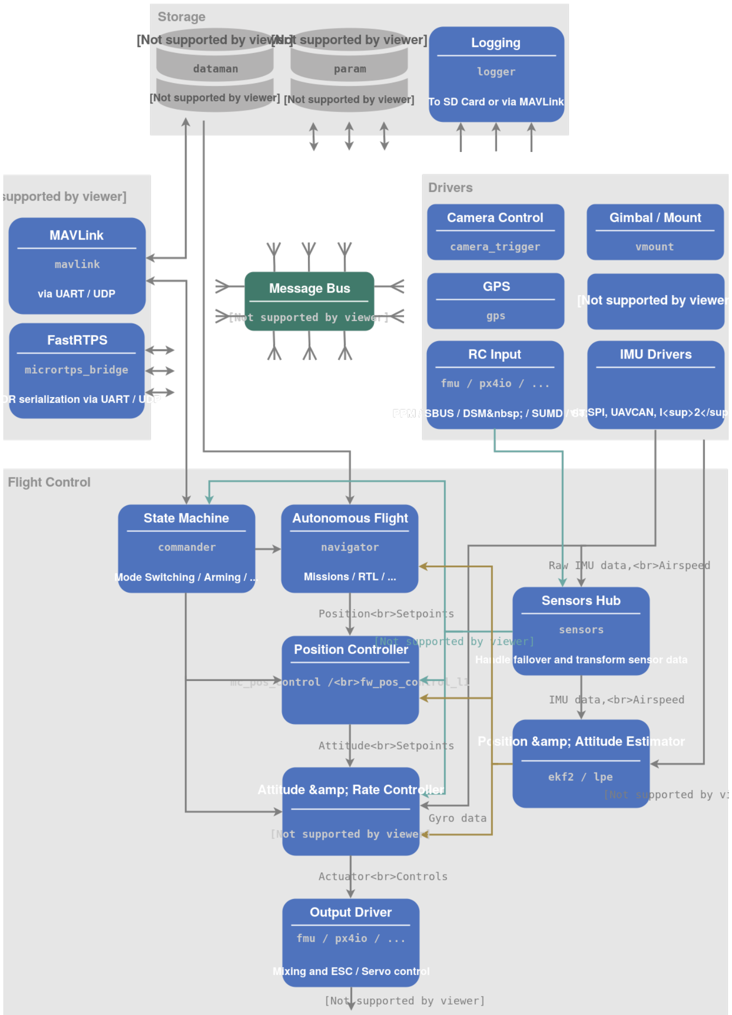


Figure B.3: PX4 Architecture

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.” [25]

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.

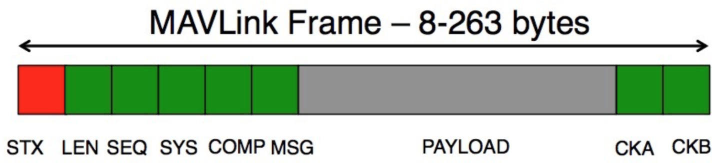


Figure C.1: MAVLink packet [26]

MAVLink messages

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

1. 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.

2. MAVLINK_MSG_ID_COMMAND_LONG
   This message is used to give commands to the autopilot. Several commands are supported.

3. SET_MODE
   It sets the different mode of operations for the drone. Few supported modes for ArduCopter are

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 [26]

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'

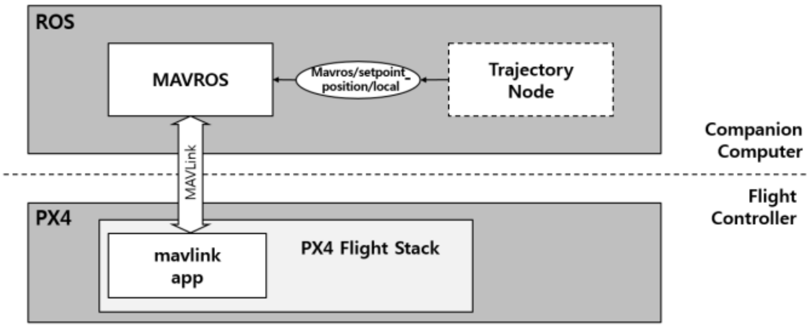


Figure D.1: Mavros as communication gateway

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 part 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.

trovare sinonimo di part

listare i plugin usati con PX4.launch???

1 List of Figures

2 List of Tables

3 Listings

4 Index