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

Presentato da:
Michael Rimondi

Correlatore:
Prof. Nicola Mimmo

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,it will be describe the specific subject of the thesis.

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.

1.1 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.

1.2 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. 5de0697a03

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.

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 State of the art

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 BambiSaver 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 bass

Software Design concept

The raspberry Pi 3 it is used as on-board computer, it runs ROS 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.1 displays all the nodes and the related topics used to let them communicate.
In the following chapter the thesis will focus, exclusively on two problems:

- *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 Innovation

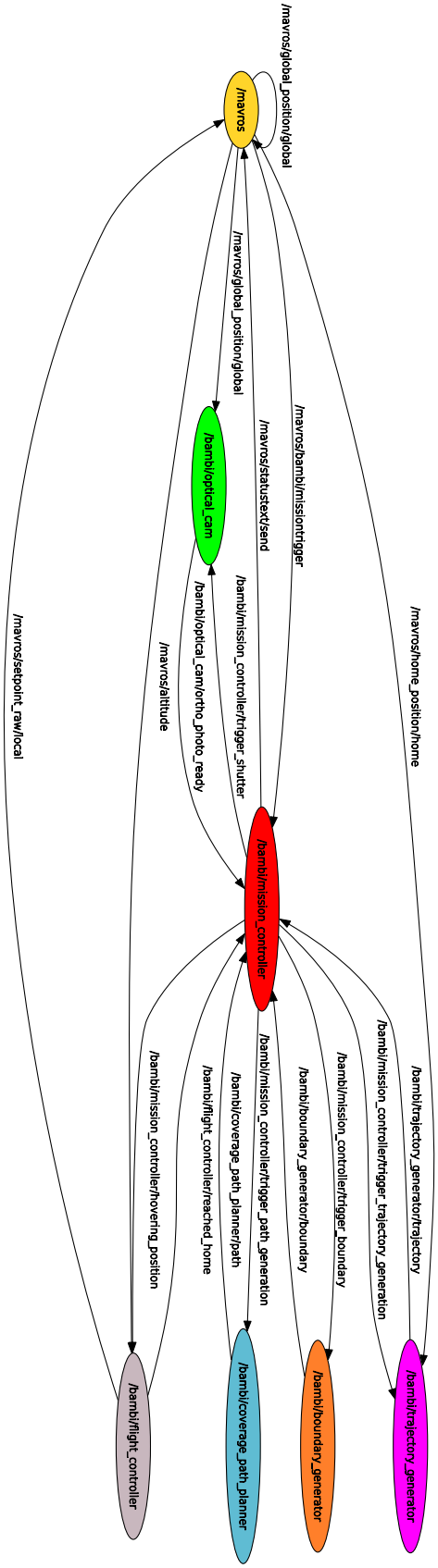

Figure I.1: Nodes and Topics graph

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

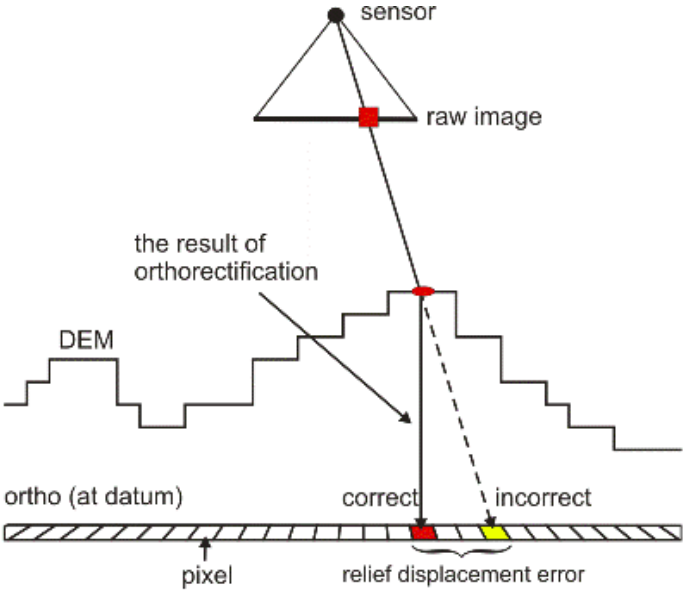

Figure II.1: Orthorectification

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

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

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

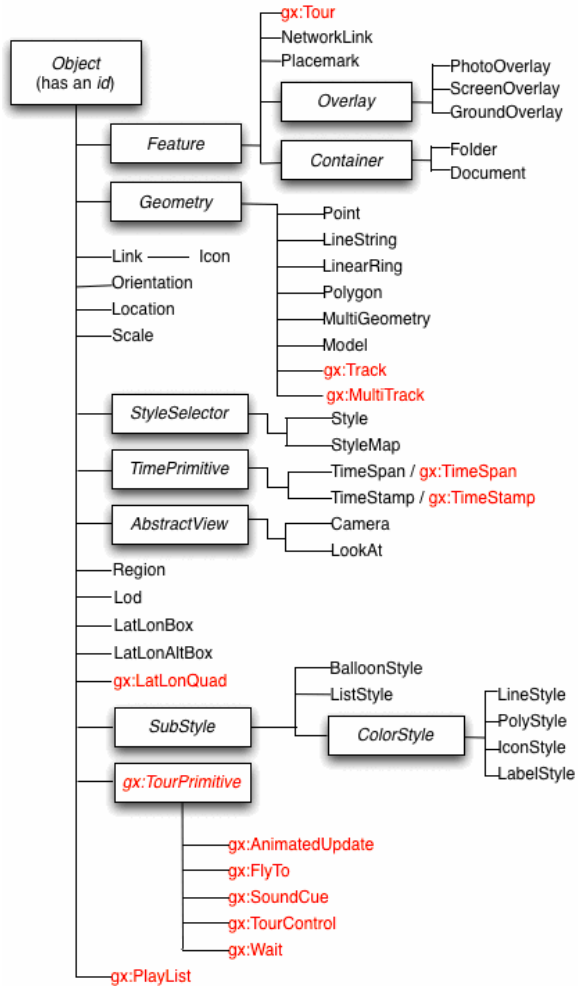

Figure II.2: KML Object Hierarchy Diagram

```
4     <name>Shapes</name>
5     <Style id="thickLine">
6       <LineStyle>
7         <width>2.5</width>
8       </LineStyle>
9     </Style>
10    <Style id="transparent50Poly">
11      <PolyStyle>
12        <color>7fffffff</color>
13      </PolyStyle>
14    </Style>
15    <Placemark>
16      <name>Waldpeter Field Boundary</name>
17      <description>Field Border</description>
18      <LineString>
19        <coordinates>
20          11.491609399895651,46.453469568513725,0
21          11.491755810622863,46.45338389393535,0
22          <!-- OMITTED COORDINATES -->
23           11.49140555201052,46.45356658266721,0
24           11.491609399895651,46.453469568513725,0
25        </coordinates>
```

```
26      </LineString>
27      <styleUrl>#thickLine</styleUrl>
28    </Placemark>
29  </Document>
30 </kml>
```

Using the *name* and *description* tags KML gives the possibility to add meta-data to the geographic feature. This comes in handy to classify the field and creating a database of the place already visited. The idea is to use the data already collected when repeating mission in a previously scanned field.

Spiegare come e' fatto il codice KML

add image of the field contour

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 subsection name

Figure II.3 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.

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.

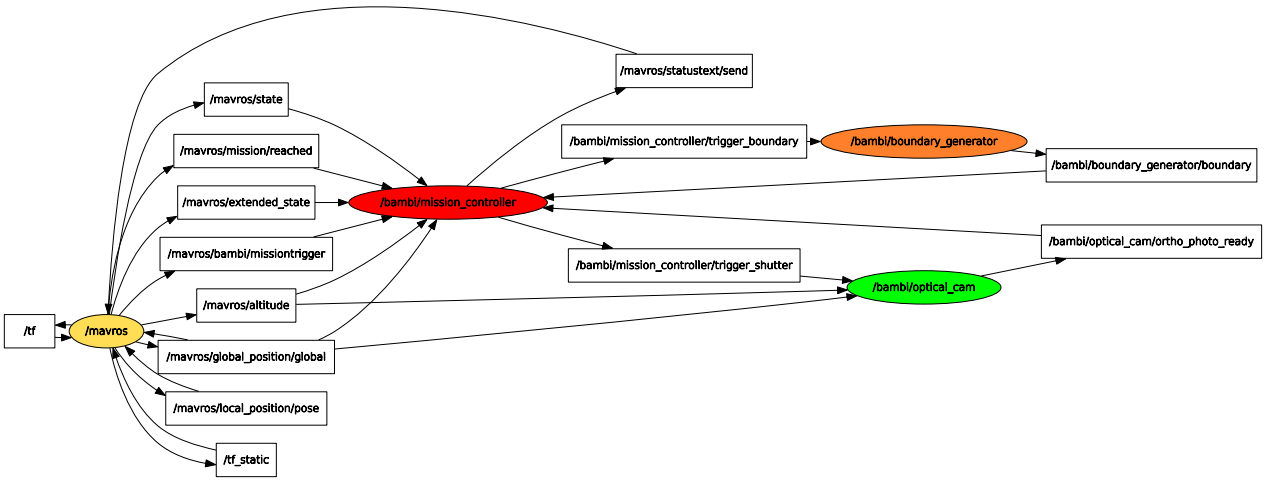

Figure II.3: Nodes and Topics concurring in generating the field contour

- */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.

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.

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

```
#! /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
```

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 in python program. Once the file has been opened it is possible to access to KML child member in the fashion 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:
```

[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]

```
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()
42    except rospy.ROSInterruptException: pass
```

.

Chapter III

Coverage Path Planning

Breve intro al problema

1 CPP Algorithms

2 Proposal Solution

2.1 Flight Altitude

Come abbiamo calcolato l-altitudine a cui volare (required resolutiion + sensor footprint)

Abbiamos celto il wavefront perch[oltre ad essere abbastanza efficente computazionalmente permette di tenere conto di diverse cost function (diverse priorita' come ad esempio l'altitudine, minor number of turns, ecc...)]

3 Implementation in ROS

Chapter IV

Simulation results

The PX4 firmware offer a great Software-In-The-Loop simulation environment which was fundamental for the software development. It guarantee 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.

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).

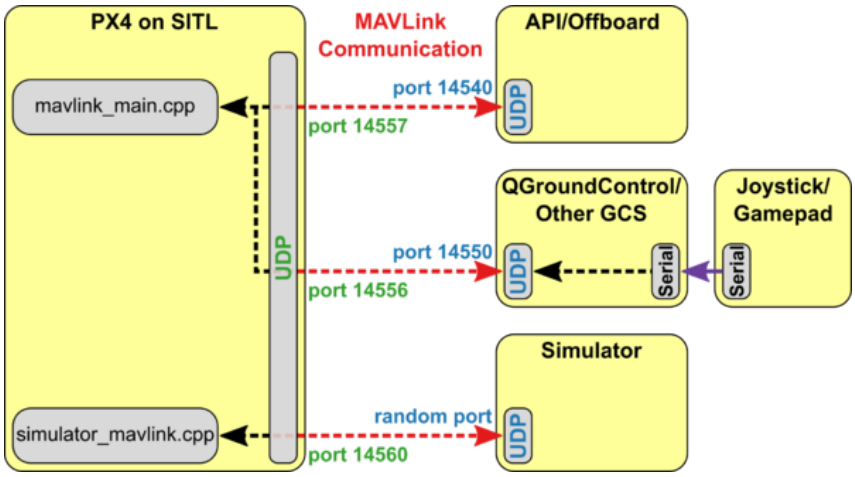

Figure IV.1: SITL with Gazebo architectural scheme

Chapter V

Conclusion

Appendix A

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

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

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." [12]

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.

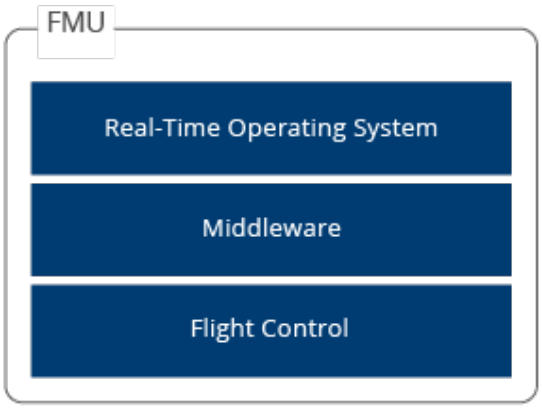

Figure B.2: The architecture of a modern flight controller

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.

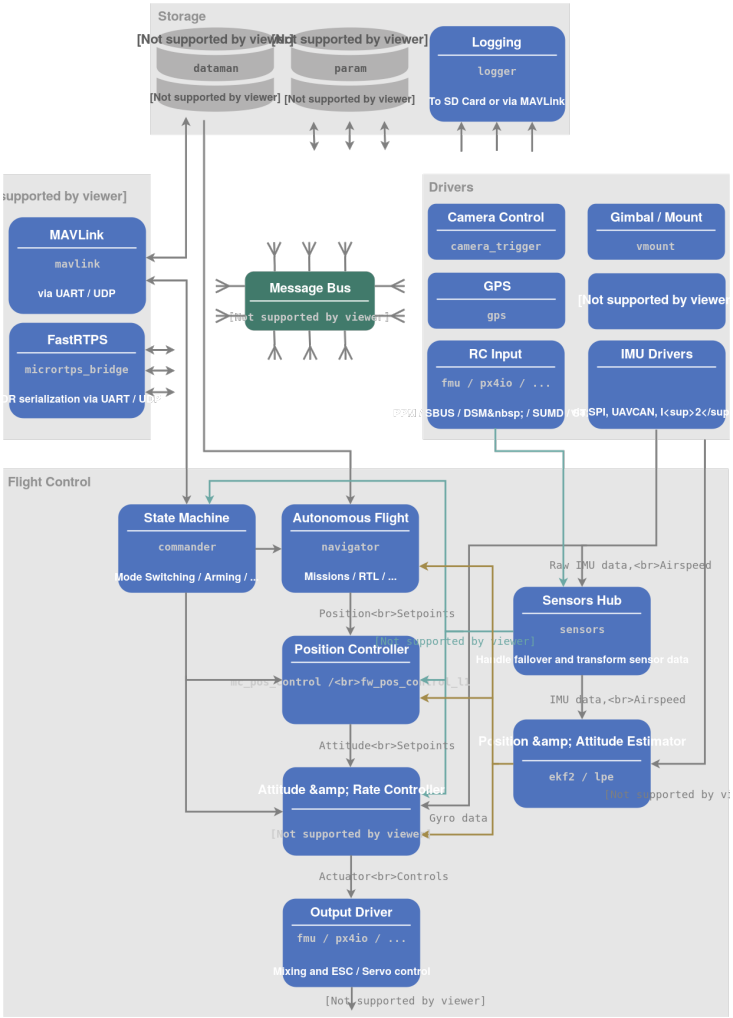

Figure B.3: PX4 Architecture

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." [13]

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).

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'

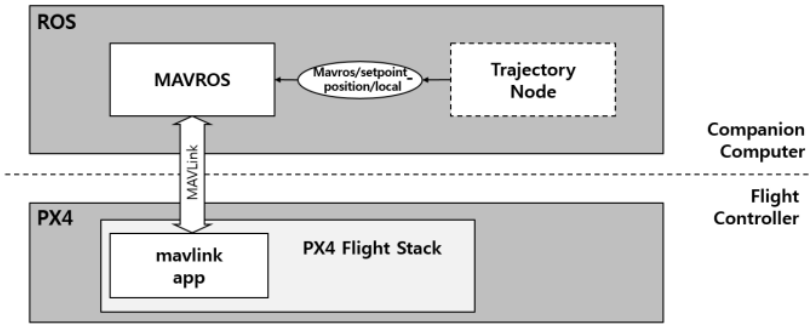

Figure D.1: Mavros as communication gateway

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