University of Padova

Department of Information Engineering

École Polytechnique Fédérale de Lausanne

Mobile Communications Laboratory

Master Degree in

Telecommunication Engineering

Machine learning techniques for geolocalization of Wi-Fi devices using flying Unmanned Aerial Vehicles

Professors

Bixio Rimoldi

Andrea Zanella

Candidate

Mattia Carpin

Academic Year 2014/2015

Abstract

The purpose of this work is to develop and test a localization algorithm that can be used by flying Unmanned Aerial Vehicles to locate Wi-Fi devices on the ground. Assuming the target to be uncooperative, we measure the received signal strength on the packets that are periodically and spontaneously broadcasted by the Wi-Fi cards, and we use this information to estimate their position. Gaussian Processes allowed us to reinforce the system against the noise that affects the measurements and the possible lack of useful data in certain positions. Bayesian optimization finally proved to be an efficient approach for collecting data in crucial locations, thus improving the accuracy of the estimation while saving time.

Ringraziamenti

Al termine di questo percorso, avventuroso ma ricco di soddisfazioni, desidero ringraziare tutte le persone che in un modo o nell'altro mi hanno accompagnato e seguito fino a qui. Senza di loro, questo importante traguardo non avrebbe avuto lo stesso significato. Anzitutto ringrazio la mia famiglia, che mi ha sempre incoraggiato e sostenuto in tutte le mie iniziative. Anche nei momenti di fatica, la loro fiducia e il loro supporto non sono mai venuti a mancare. Ringrazio poi i miei professori universitari, in particolar modo Andrea Zanella, e gli insegnanti di tutte le scuole che ho frequentato. Mi hanno trasmesso, con grande passione e carisma, le competenze che ora possiedo, e sono stati consiglieri preziosi in tutti i momenti importanti della mia vita. Infine ringrazio i miei amici, in gran parte scout, e i miei straordinari compagni di Università. Per le mille avventure affrontate insieme, per gli splendidi campi estivi, per il disinteressato e costante supporto reciproco, per avermi aiutato a sdrammatizzare in tutte le difficoltà. La strada condivisa insieme a voi mi ha aiutato a diventare la persona che ora sono.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview and motivation

Over the last years, with the advancement of the technology, we witnessed the birth, the development and the growth of many new devices. Smartphones, smartwatches, tablets and laptops now belong to our everyday life, and through the Internet network they keep us in continuous connection with each other. In this new florid technological environment, Unmanned Aerial Vehicles (UAVs) are also experiencing a tremendous expansion. Rather than being employed exclusively for military purposes, nowadays UAVs (also known as drones) are used for several civilian and scientific applications, and the industry is also producing low-cost UAVs for entertaining intents.

Due to the speed and ease with which drones can be deployed in large areas, one possible application for UAVs is the localization of people on the ground. On the one hand, after catastrophic events when all the network infrastructure is unavailable, drones could be used to locate survivors and to direct the emergency teams. On the other hand, rather than confining the usage to emergency situations, geolocalization using drones is also gathering interest in the scientific and industrial community. For example, Amazon [4] is now developing a new technology for the delivery of packages using UAVs, where the localization of the user clearly plays a crucial role. In other projects, where the objective is to provide internet connection using aerial devices, localizing people on the ground and moving the aerial access point accordingly could potentially yield an improvement in the offered Quality-of-Service (QoS).

Localization of people on the ground could potentially be performed using bird's-eye view camera mounted on a UAV [5]. However, in order to acquire and analyze the images, the UAV needs to fly at a low altitude and at low speed, thus limiting the

dimension of the search area. Furthermore, the approach is confined to applications where the people are clearly visible from the onboard camera.

In this work we consider a different approach to the problem, and we estimate the location of people on the ground monitoring the activity of their Wi-Fi devices. Due to the massive wide spread of Wi-Fi-enabled devices, locating the device often corresponds to locating its owner. This approach allowed us to design a system that can be used to scan efficiently areas that can be potentially vast. Our idea is to monitor the Wi-Fi activity using a UAV that carries a Wi-Fi monitor, with an active-sensing approach where the sensor can be quickly moved in different locations.

One key assumption for this work is to consider the target devices to be uncooperative and unaware of the localization process, meaning that such devices do not send specific packets that contain their location and do not run any particular software. Since the natural application of this work is for rescue operations, we want to limit our assumptions on the devices as much as possible. Even if the target is uncooperative, as specified in the 802.11x standards, device periodically broadcasts a management frame, called probe-request frame (PRF), to actively scan the environment and find the available access points (APs). Any Wi-Fi enabled device operating in monitor mode (the UAV, in our case) can capture and decode the unencrypted PRF. By measuring the intensity of this received signal, we can estimate the location of the device on the ground. Intuitively, if the signal strength is high it means that the UAV is flying close to the source, if the signal is weak the UAV is far from the transmitting device. We will measure the signal strength using the Received Signal Strength Indicator (RSSI), that is a measure performed by the receiving wireless card that indicates the power level of the received signal, and it is usually employed by the operating system to roughly evaluate the quality of the connection. The RSSI is an indication of the power level being received by the antenna, and therefore the higher the RSSI number, the stronger the signal. More details on the IEEE 802.11 standard and on the RSSI will be given in Section 2.2.

1.2 System requirements

The requirements of the system, and the design choices, are here briefly summarized.

- **Robustness:** the usage of the RSSI as a metric for locating devices has always been very controversial in the scientific community, mainly because this indicator was not designed for this specific localization purpose. Therefore the algorithm should be robust and able to deal with noisy measurements and potential missing

data, since the RSSI can be measured only if the UAV is close enough to the source. We will use Gaussian Processes (GPs) to estimate the device's location from the noisy measurements, mainly for their flexibility and robustness.

- **Autonomous and real-time:** to be completely autonomous, and perform a real-time estimation, the UAV carries a small embedded computer that is able to process the incoming data and drive the UAV throughout the localization process. Furthermore, the onboard computer should be able to communicate with a ground station to report step-by-step the evolution of the mission and the current location estimate.

- **Energy efficiency:** an exhaustive scan of the area would be extremely expensive both in terms of battery's energy an time. For this reason, we use a Bayesian-optimization driven search algorithm. In a nutshell, based on the observations collected so far, we decide if it is better to explore areas of where the estimate exhibits a high uncertainty, or to stick with the the current estimation and refine it.

- **Search area:** the target to be located is assumed to be inside a pre-defined search area. To cover a potentially vast areas, we used a fixed-wing drone that can easily fly at high speed.

- **Non cooperative target:** differently from what is usually assumed in the literature, the target to be located is assumed to be uncooperative. This is why, relying on the 802.11x standard, we perform the localization using only the PRFs that are periodically and spontaneously transmitted from the device.

1.3 Outline

The remainder of this thesis is organized as follows.

We describe in Chapter 2 the UAV that we used to conduct the experiments and the onboard computer that processes the incoming data. In the same chapter, we also present some relevant aspects of the IEEE 802.11 standard, with particular focus on the network architecture, the management frames and the scanning techniques. The results of a preliminary feasibility analysis are also reported there, to prove that, in our environment, the RSSI can be used for localization purposes. At the beginning of Chapter 3 we give the formal statement of the problem and its formulation as a regression problem. We also explain why a traditional approach could potentially yield

poor performance. Chapter 4 describes how Gaussian Processes are used in our work to solve the regression problem. The mathematical core of the thesis lies in Chapter 5, where we describe how the uncertainty of our estimation can guide us through an efficient exploration of the search area. The localization algorithm is then fully described in Chapter 6. To provide some reliability measures we designed a small MATLAB simulator that is described in Chapter 7, along with the simulated and experimental results. Finally, in Chapter 8, we present the conclusions of the project and the possible future extensions.

Chapter 2

Background

2.1 Unmanned Aerial Vehicles

A Unmanned Aerial Vehicle (UAV) (also known as drone or Remotely Piloted Aircraft (RPA)), is an aircraft without a human pilot onboard. The initial and natural development field for drones has been the military industry, where the absence of a human pilot onboard allowed to conduct riskier missions without the threat of casualties. However, with the advancement of technology and driven by a rising demand, the development of UAVs has become more and more relevant also for civilian and commercial applications. Nowadays, beyond the military applications of drones, UAVs are used for aerial surveying [6], search and rescue operations [7], aerial photography [8], climate observation [9], wildlife counting [10], forest fire detection [11] and lot more [12]. UAVs have also been proven to be useful for deploying self-organizing wireless networks to support rescue operations in case of natural disasters or catastrophes, as shown in [13]. Small UAVs can be divided in 2 categories:

- Rotary-blade UAVs, commonly known as mini-copters or multi-copters, in which lift and thrust are supplied by rotors. This allows them to take off and land vertically, and to stay in a specific position in the air. Rotary-blade UAVs are very stable and they usually move slowly, thus making them suitable for aerial photography, but not convenient when the area to be monitored is vast.

- Fixed-wing UAVs, that are moved forward from a propeller, so that the wings can generate the necessary lift to keep the plane in the air. Fixed-wings drones usually fly at a relatively high speed and are mostly used for covering huge areas.

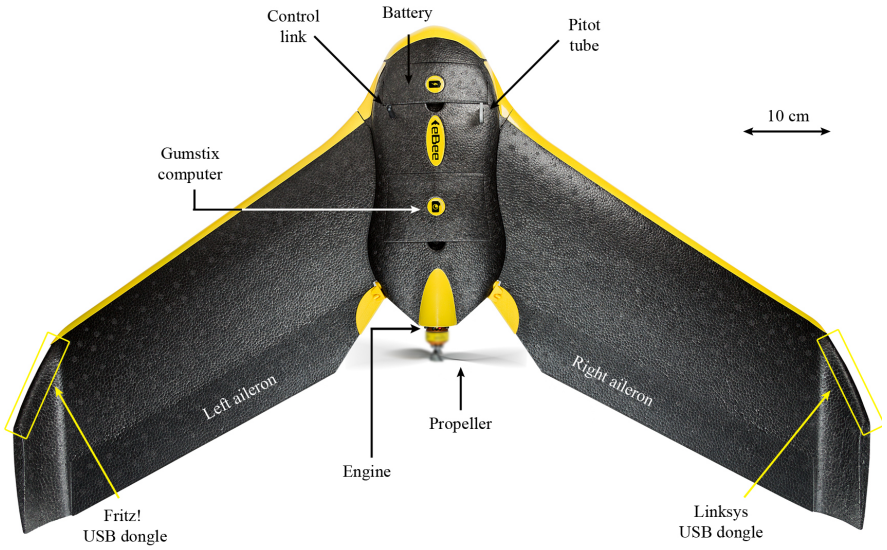


Fig. 2.1: SenseFly eBee UAV.

For this project we used the fixed-wing *eBee* UAV developed by SenseFly, a Swiss company that produces aerial imaging drones for professional applications [8]. The UAV is shown in Figure 2.1. The plane wingspan is 96 cm and the body is made of expanded polypropylene, a light material that makes eBee harmless in case of unexpected crash. This is the reason why, in lot of countries including Switzerland, eBee can be flown without any particular permission. The battery can provide an autonomy of about 45 minutes of flying time and the engine can push the plane to a speed of 60 km/h withstanding wind flurries up to 30 km/h.

The plane comes with some useful instruments onboard: a pitot tube that measures the macroscopic speed of the plane, a barometer, an inertial measurement unit, a GPS receiver and a ground proximity sensor that is used to determine the absolute altitude with respect to the ground. All those data are processed by the onboard autopilot, that is responsible for flying the plane and is in control of the engine and all the plane's ailerons. To assign a specific flight path to the plane, and monitor all the parameters, we can use the provided Windows software, named eMotion, that makes use of an external 2.4 GHz USB radio modem to communicate with the onboard autopilot. Using the GUI provided by eMotion, we can easily setup a mission for the UAV, monitor its progress and eventually land. After the mission has been initially specified using eMotion, the UAV can be flown autonomously by the onboard autopilot, since it has access to all the sensors.

SenseFly produces eBees for aerial imaging professional applications, and that's why, in the central part of the main body, a standard camera can easily be arranged. Instead of a digital camera, we housed inside the compartment a mini ARM-based Gumstix Overo Tide board, a small embedded computer developed by Gumstix Inc [14]. Gumstix runs Yocto (kernel version 3.5.0) [15], a tiny Linux distribution that is especially designed for embedded devices. The Gumstix board is connected with the onboard autopilot through a serial port, and can take control of the mission flying the plane to specific locations using a C++ library directly provided by SenseFly. Furthermore, using the same library, Gumstix can fetch all the vital information from the UAV, such as the latitude, the longitude and the altitude of the plane. All the software and the algorithms we developed, and that will be described later, run on this Gumstix computer.

The Gumstix computer is connected also to an expansion board, that has been designed and produced at EPFL to fit the compartment minimizing the occupied space. Two different USB Wi-Fi cards are connected to the expansion board (see Fig. 2.2):

- A WLAN Fritz! USB card [16], configured in monitor mode to operate on a frequency of 2.4 GHz, used to perform packet sniffing.

- A WLAN Linksys AE3000 USB card [17], set to work in the 5 GHz unlicensed band. Using this interface we can create an ad-hoc network that gives us the access to the data onboard and allows to give instructions to Gumstix realtime from any station on the ground. This is an high data-rate link, that works only on short distances (typically some hundreds meters if we are in Line-of-sight propagation). It may seem unreasonable to choose the 5 GHz band for this card, since at higher frequencies electromagnetic signals undergo an higher attenuation. However, as seen from previous experiments, the less crowded 5GHz bands gives better results in terms of data rate and covered distance.

2.2 IEEE 802.11 standard

In this section we introduce the IEEE 802.11 standard, focusing our analysis on three different aspects. After a brief history of the standard, we present the network architecture, that is needed to better understand the involved devices and their role. We then move to the management frames used by the standard to provide the basic functions and to the active scanning techniques, that are of utmost importance in our

Fig. 2.2: WLAN Fritz! USB card (left side) and Linksys AE3000 USB card (right side).

work. Finally, we describe a software framework that can be used to parse and extract information from received 802.11 packets.

2.2.1 Introduction

Wireless local area networking has experienced an incredible growth in the last decades. Since 1971, when the first packet-based wireless network was created by researchers at the University of Hawaii [18], a tremendous improvement in the performance of wireless networks has been achieved, both in terms of performances and deploying and maintenance costs. This large and sustained growth is in large part due to the benefits that Wireless Local Area Networks (WLANs) offer over their wired counterpart. In fact, providing wireless network connectivity in houses or enterprises, usually involves the installation of simple access points that can be easily deployed to serve even a large area. Furthermore, the proliferation of laptops, smartphones and handheld devices, has meant that people desire to be connected wherever they are despite of where the network connection is located, thus encouraging even more the deployment of wireless networks [1].

The Institute of Electrical and Electronics Engineers (IEEE) started in 1990 the development of an international WLAN standard identified as IEEE 802.11. The purpose was to design a Medium Access Control (MAC) and Physical Layer (PHY) specification for wireless fixed, portable and moving stations within a local area. With an increased commercial interest on wireless networks, the principal manufacturer formed the Wi-Fi Alliance in 1999, to certify interoperability between IEEE 802.11 devices. Ideally, users of wireless networks will want the same services and capabilities that they have commonly come to expect with wired networks. However, to fulfill those expectations, the wireless community faced certain challenges that were not imposed on

the wired counterpart. Interference management, reliability, security issues, power consumption, mobility are a few examples of problems that had to be properly addressed to achieve the good level of service we are nowadays used to [19]. The commercial interest has pushed the scientific community to propose solutions to those problems, bringing WLANs to an undeniable level of diffusion and utilization. Shipment of Wi-Fi certified integrated circuits are expected to exceed 2.5 billion units by 2016 [20].

2.2.2 The network architecture

As a member of the 802 family of local area networking (LAN) standards, 802.11 inherits the 802 reference model (based on the OSI reference model) and the 48-bit universal addressing scheme. The Basic Service Set (BSS) is the basic building block of any 802.11 WLAN. Stations that remain within a certain coverage area and are somehow associated form a BSS [1].

The basic form of association is an ad-hoc network, where stations communicate directly with one another and form an independent BSS (IBSS). Usually, however, devices associate with a central station (referred as access point (AP)) that is in charge of managing the BSS. A BSS that is built around an AP is called an infrastructure BSS, and it may be interconnected with other BSSs via their AP trough a distribution system (DS). The BSSs interconnected by a DS form and extended service set (ESS), and devices belonging to the same ESS can address each other directly at the MAC layer. An illustration of both the association schemes is shown in Figure 2.3. Each BSS is identified using a 24 bit long unique code, named BSS IDentification (BSSID), that is the formal name of the BSS.

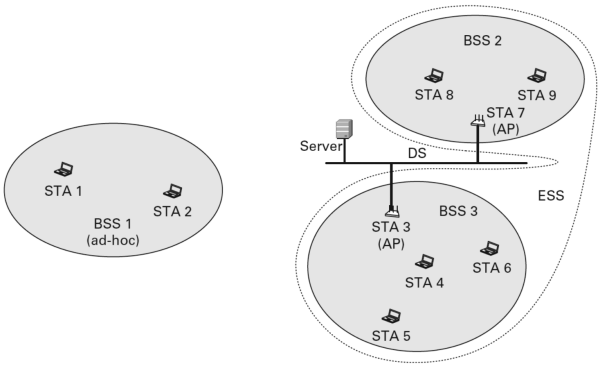


Fig. 2.3: On the left side an independent BSS, where devices form an ad-hoc network. On the right side, two infrastructure BSSs connected using a DS trough their APs to form an ESS [1].

2.2.3 The MAC layer

The MAC layer is responsible for channel allocation procedures, packet data unit (PDU) addressing, frame formatting, error checking, fragmentation and reassembly [19]. The wireless transmission medium can operate exclusively in contention mode, thus requiring all stations to contend for accessing the channel for each packet transmission. Influenced by the huge market success of Ethernet, the 802.11 MAC adopted a variant of a simple distributed access protocol: carrier sense multiple access with collision avoidance (CSMA/CA). With this access technique, if a station sees the medium to be busy, it uses a more conservative approach following the medium and waiting for a random backoff time. The performance using this conservative approach improves, since on wireless networks collisions cannot be directly detected and need to be inferred through the lack of the acknowledgement. Therefore, the penalty factor for a collision in WLAN is higher with respect to the wired LAN, thus justifying this more conservative approach.

The standard IEEE 802.11 frame format is illustrated in Figure 2.4. We focus our attention on the Frame Type and Frame Subtype fields, that are respectively 2 and 4 bits long. The frame type bits specify the nature of the frame as either control, management or data, while the frame subtype field further contributes to identify the particular role of the frame. Table 2.1 shows the type and subtype bits for three management frames that are relevant in our analysis, since their role is related to the scanning techniques that are now explained.

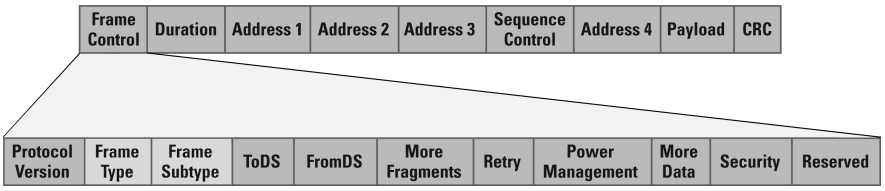


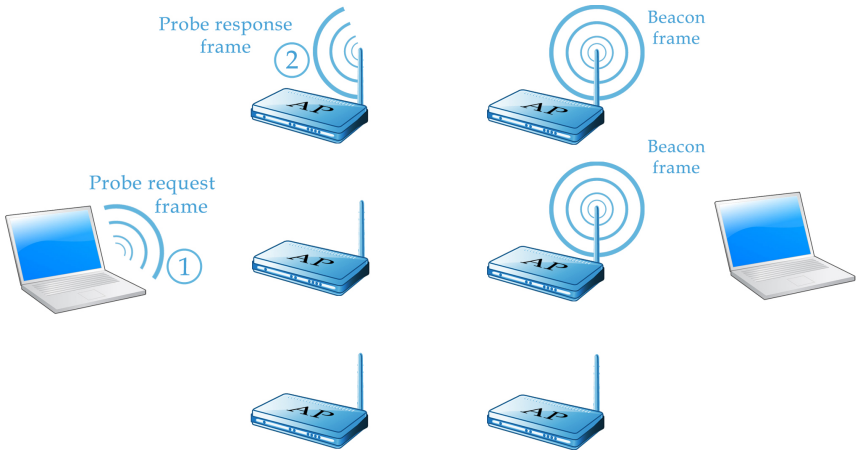
Fig. 2.4: Standard IEEE 802.11 frame format.

Type	Subtype	Frame name
00	0100	Probe request
00	0101	Probe response
00	1000	Beacon

Table 2.1: Type and subtype bits for three relevant management frames.

2.2.4 Scanning techniques

Usually, when a device wants to connect to a wireless network, it uses a process that is called scanning. This procedure can be active, where the node that wants to connect is directly involved in the transmission of a join request, or passive, where the node relies on the information that are periodically distributed from the available BSSs.



(a) Active scanning (b) Passive scanning

Fig. 2.5: Scanning techniques for IEEE 802.11 standard.

In active scanning (see Fig. 2.5a), the scanning station transmits a particular management frame, called probe request frames (PRFs). Inside the PRF, the node can specify the SSID of the BSS he wants to connect to, or choose to leave this field empty, meaning that he is willing to connect with any available BSS. The unencrypted PRF, that contains the MAC address of the source, is then broadcasted. If there is a BSS that matches the SSID contained in the PRF, that BSS replies by sending a probe response frame to the scanning station. This approach can minimize the scanning time if the node knows which channel should be used for the transmission, but scanning all the available channels may take time and be expensive in terms of consumed energy. Note that the PRFs can be captured and interpreted from any device that is in the range of the transmitting scanning station, that can be uniquely identified by the mean of its MAC address. Therefore, any station performing active scanning can be detected by monitoring the transmitted PRFs.

In passive scanning (see Fig. 2.5b), instead, the scanning node listens for beacons that are periodically transmitted from the APs. This operation can be repeated for different channels until the node is connected with a BSS. This approach is conservative in terms of energy, but may take long time to be completed, since the beacon frames

are transmitted only at predetermined instants by the APs.

2.2.5 The radiotap header

If we analyze any received management frame, coming from a generic wireless card using a suitable application, such as Wireshark [21], we will observe that the frame is composed of two different parts: first the radiotap header, and then the content of the management frame (similar to what is shown in Figure 2.4). The radiotap header is a flexible mechanism that supplies additional information about frames from the drivers of the wireless card to the userspace applications. All the details and the information contained in the radiotap header can be found at [22]. We will make an extensive use of the Received Signal Strength Indicator (RSSI, or simply SSI), that is an information coded inside the radiotap header. The RSSI is a measure performed by the receiving wireless card that indicates the power level of the received signal, and it is usually employed by the operating system to roughly evaluate the quality of the connection. RSSI is an indication of the power level being received by the antenna, and therefore the higher the RSSI number, the stronger the signal. It is important to know that there is no standardized relationship of any particular physical parameter to the RSSI reading. The 802.11 standard does not define any relationship between RSSI value and power level of the received signal, and vendors and chipset makers use their own accuracy, granularity, and range.

2.2.6 Scapy and Libcrafter

To decode, interpret and capture packets from a network card, several options are available. Some applications, for example Wireshark, provide also a Graphical User Interface (GUI) to perform a visual inspection of the packets. Other tools, such as tcpdump, can monitor an interface logging all the received data into a file that can then be parsed and interpreted. For this work we need an application that can run on the onboard Gumstix computer, and that is capable of receiving packets, decoding them, and extracting the RSSI. This needs to be performed real-time, since the new RSSI measurements affect the estimation of the location, and therefore the actions of the UAV.

Scapy is python module that allows to capture, decode, forge, dissect and send packets complying with a huge variety of protocols [23]. Scapy recognizes all the fields of 802.11 management frames, and easily allows for their extraction. Nonetheless, despite its user-friendly programming style, Scapy is not a suitable tool when the device

operates under heavy load conditions. In fact, the module doesn't have a proper buffer to store unprocessed packets: if a new packets comes when the previous is still being processed, the latter is simply discarded. This limitation is also mentioned on the developer website [23]. We implemented a basic sniffer in Scapy and run it on our onboard Gumstix computer confirming that, under heavy load conditions, around 20% of the packets were dropped and were not properly inspected and logged.

For those reasons, we moved to Libcrafter [24], an high level C++ library that has an interface very similar to Scapy. We managed to run Libcrafter on the Gumstix computer, and we tested the script under heavy load conditions, seeing that all packets were correctly received and interpreted. A small piece of the basic code is reported in the following box. In particular, in the main routine, we define the interface to be monitored and the filter, that follows the same syntax used in tcpdump. The instruction *sniff.Capture()* blocks the execution of the code: all the lines that follow that instruction in the main file will never be reached. Each time a packet that matches the specified filter is received, the function *PacketHandler* will be invoked. The *PacketHandler* function is responsible for obtaining the payload of the packet, extracting the RSSI and saving it into a file, along with the current position of the UAV. According to the structure of the radiotap header, the RSSI is a 2 bytes-long field contained in the 14th and 15th bytes, and it is expressed in ones' complement notation. Finally, if the packet length is different from what it is declared in the header, the packet is marked as malformed and dropped, to avoid an inconsistent measures of the RSSI.

```
void PacketHandler(Packet* sniff_packet, void* user)
{
  int packetSize=sniff_packet->GetSize();
  byte* buffer = new byte[packetSize];
  int readData=sniff_packet->GetData(buffer);
  short RSSI=(short)(buffer[14])-256;
  getPosition();        //Get from GAPI the current position
  logPacket();          //Log the RSSI and the position on a file
}
int main(int argc, char** argv)
{
  string iface = "mon.ap0";                    //Interface's name
  string filter = "type mgt subtype probe-req"; //Filter for packets
  //Start sniffing packets on the interface.
  Sniffer sniff(filter, iface, PacketHandler);
  sniff.Capture();
  return 0;
}
```

The application, to work properly, requires to set the interface to Radio Frequency MONitor (RFMON) mode, so that it can monitor all the traffic received from the wireless network. Unlike promiscuous mode, which is also used for packet sniffing, monitor mode allows packets to be captured without having to associate with an access point or ad hoc network first. In our UAV, the WLAN Fritz! USB card is set to operate in monitor mode and is used to collect the RSSI measurements.

2.3 RSSI-based ranging

The underlying idea to perform RSSI-based localization is simple: the signal strength should be greater when the monitor approaches the source, and should decade as the monitor moves away. Therefore, by measuring the RSSI in certain locations, one should be able to say where the signal is stronger an where it is weaker, and therefore locate the source. Unfortunately, the RSSI measurements depend not only on the distance, but also on the noise introduced at the received and the changing channel conditions. In fact, the usage of the RSSI as an indicator for locating devices has been very controversial in the scientific community.

Some authors have shown that the RSSI can be used in real situations to perform an estimation of the relative positions between the transmitting and the receiving devices. For example, in [25], authors have presented a RSSI-based framework for estimating the relative position of mobile robots in an unknown environment. Gaussian Processes (GPs) are used to fit the experimental data while local gradient ascent is used to guide the exploration of the unknown area. In [26], authors use a basic k-nearest signal space neighbour matching algorithm for location estimation, applying the algorithm in a real experiment and presenting some results for the achieved precision, showing that under some conditions the estimation can be quite accurate. Finally, the accuracy of RSSI-based ranging can be increased by averaging RSSI samples collected on different RF channels, as shown in [27].

On the contrary, in [28], authors concluded that RSSI is a poor distance estimator when the nodes are inside a building, since reflection, scattering and other physical properties have a too strong impact on the RSSI measurement. Lot of work has been done by several authors to develop models that can capture the behaviour of the noisy RSSI, but each environment presents different features that are difficult to be captured and generalized by a single model. It is therefore not easy to predict beforehand if an RSSI-based location estimation can be performed in our scenario, which has some peculiarities that make it different from what can be found in the scientific literature.

In particular

- The UAV is forced to move forward to generate the necessary lift to fly, and therefore it cannot stand in the same position to collect multiple RSSI measurements from the same spot.
- The UAV flies at a relatively high speed, thus the received signal undergoes a non negligible Doppler shift that affects the measured RSSI. However, we assume the source to be in an open-area, so the signal propagates in Line-of-Sight conditions.
- The source is assumed to be not cooperative, i.e., the generation and emission of the probe packets are agnostic of the localization agent.

Under those conditions, is it possible to use the RSSI to estimate the location of the source? In particular, this question boils down to see if there is a correlation between the RSSI and the distance between the monitor and the Wi-Fi device. If such relation exists, it can be used to estimate the location. We describe here the first experiment designed to answer this preliminary question. The source to be located is a Sony Vaio Pro 13 Ultrabook, and the Wi-Fi monitor is carried by the eBee UAV designed by SenseFly, previously described in Section 2.1.

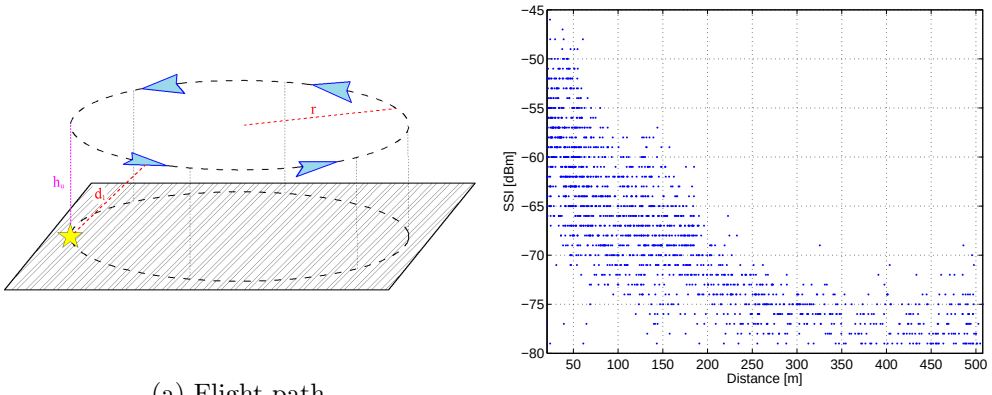
At this very preliminary stage, we assume the source to be transmitting PRFs at a constant rate over time. Forging and transmitting artificial PRFs can be challenging, although some libraries as Scapy and Libcrafter could provide support for such process. However, we noticed that each time a Wi-Fi interface is activated under Ubuntu 14.04, some PRFs are generated to scan for the available networks. Therefore, we designed a simple application that turns on the Wi-Fi interface, waits for the PRFs to be sent, switches off the interface and repeats the operation again. Using this simple approach, the notebook will emit PRFs at an almost constant rate. Those packets are then received by the WLAN Fritz! USB card onboard the UAV, the RSSI is extracted from the radiotap header and recorded together with the instantaneous position provided by the onboard GPS. In this experiment, the position of the source is known by the UAV, so for each received packet the RSSI can easily be associated with the distance between the target and the UAV. In particular, let h_u denote the height of the UAV with respect to the the take-off position, as measured from the GPS, and h_s be the height of the source with respect to the same reference altitude. Then $\Delta h = h_u - h_s$. If we denote with (λ_u, φ_u), (λ_s, φ_s) the longitude and the latitude of the UAV and the source, respectively, the distance can be computed as

$$d = \sqrt{\mathrm{dist}\big((\lambda_u, \varphi_u), (\lambda_s, \varphi_s)\big)^2 + \Delta h^2}, \tag{2.1}$$

where $\text{dist}(a, b)$ is the distance between the two coordinates computed using the harvesine formula (see equation (9.1b) in Section 9.1).

2.3.1 Experiment design

To collect RSSI measurements from a wide range of distances, the plane is instructed to fly at a constant altitude following a circular path of radius r, with the source positioned to lie on the ground projection of this circular trajectory. Figure 2.6a shows the designed flight path that allows us to collect packets with distances in the range $d \in \left[\Delta h, \sqrt{\Delta h^2 + 4r^2}\right]$.



(a) Flight path.

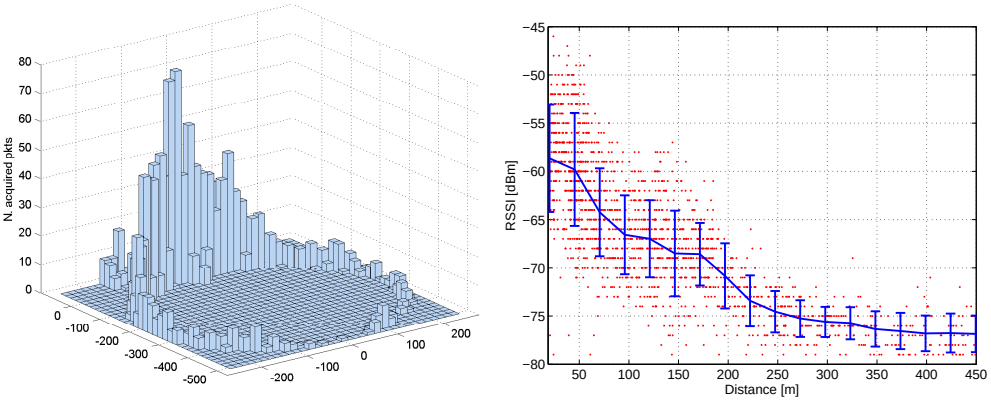
(b) Scatter plot of the raw data.

Fig. 2.6: Preliminary experiment.

Figure 2.6b shows the scatter plot of the raw dataset, composed of 1800 RSSI samples obtained with $r = 250$ m and $\Delta h = 20$ m.

A clear and strong inverse relation between the distance and the RSSI can be observed, although the measurements are corrupted by some noise. When the distance between the monitor and the target is reduced, the RSSI reflects the increase in the strength of the received signal, thus showing that in our scenario, the RSSI has the potential to be used as a metric for location estimation. We can also observe that packets can be received by the monitor onboard the UAV even at relatively high distances. In particular, some packets are received even when the UAV is furthest away from the source, at $d = 500$ m. However, the packets reception rate at different distances is not constant. Figure 2.7a shows the spatial distribution of the received packets over the circular ring, where the two axis represents the distance in meters between the source and the UAV (the source is positioned on the origin of such system). As we can see, the majority of packets are captured when the monitor is close to the source. In fact, while

for short distances we are able to capture almost all packets, the amount of received packets drops quickly as the UAV moves away.



(a) Spatial distribution of the captured packets during the mission.

(b) k-points Nearest-Neighbour fitting.

Fig. 2.7: RSSI measurements obtained during the preliminary mission.

2.3.2 Nearest neighbour technique

The scatter plot in Figure 2.6b shows a correlation between the distance and the measured RSSI. If we were to fit the empirical data, to obtain an analytic model that given the distance predicts the RSSI, averaging would probably be the first and easiest thing to do. We will describe in Chapter 4 more refined techniques to learn such relation, but we could now think of reducing the noise by averaging all the measures that exhibit a similar distance. This corresponds to applying a k-points Nearest-Neighbour method [29], where the observations that are close in the input space are averaged to obtain the estimation. Formally, the estimation of the RSSI at distance d is computed as

$$\mathrm{RSSI}(d) = \frac{1}{k}\left(\sum_{\mathrm{RSSI}_i \in N(d)} \mathrm{RSSI}_i\right), \tag{2.2}$$

where $N(d)$ is the neighbourhood of d, defined by the k closest points to d in the dataset. Figure 2.7b shows the results of this basic technique on the data we collected, where the blue curve is the pointwise averaged RSSI using the k-points nearest-neighbour method plus and minus the standard deviation, for a value of $k = 80$. The curve confirms the inverse relation between the RSSI and the distance.

With the results of this preliminary analysis we can now move to more refined techniques for handling the noisy measurements and perform the localization of the

Wi-Fi node, as discussed in the next chapters.

Chapter 3

System model

In this chapter we provide the formal statement of the problem and its formulation as a maximization problem.

3.1 Mathematical formulation

In this work, our goal is to localize multiple Wi-Fi devices in a given rectangular search area $\mathcal{X} \subset \mathbb{R}^2$. Let us say that we have M devices whose fixed positions are denoted by $\mathbf{x}_i^* \in \mathcal{X}$, where $i = 1, 2, \ldots, M$. We emphasize that, in our scenario where $\mathcal{X}$ could potentially be vast, it is crucial to develop an algorithm that is capable of collecting measurements where they really matter. Otherwise we may end up trying to perform an exhaustive scan of $\mathcal{X}$, that is not feasible given the constraints in terms of the UAV's flight autonomy. For the sake of simplicity, in the following we consider the case of a single device, i.e., $M = 1$. Chapter 6 provides the extension to multiple devices.

We can formally think of measuring the RSSI at a certain location $\mathbf{x} \in \mathcal{X}$ as drawing a sample from a random variable $Y : \mathbb{R}^2 \to \mathbb{R}$. More precisely, the measurement y_i at location $\mathbf{x}$ depends on the distance between the UAV and the device (as shown in the preliminary analysis), on the environment (buildings, vegetation, weather conditions), on the channel (fading and shadowing) and on the noise introduced at the receiver. In any case, we can think the measurement to be composed of two parts: the first that depends only on the distance (that is where the information we are looking for is hidden) plus some kind of noise, that accounts for everything mentioned before.

This drives us to formulate the problem as a supervised-regression problem, where we are required to learn the input-output mapping from empirical continuous data. In particular, the i-th RSSI measurement y_i taken at location $\mathbf{x}$ with a single target in

position $\mathbf{x}^*$ can be written as

$$y_i = f\big(\mathbf{x}|\mathbf{x}^*\big) + \varepsilon_i \tag{3.1}$$

where $f(\mathbf{x}|\mathbf{x}^*)$ is a function (to be estimated) that maps each location $\mathbf{x}$ into the corresponding expected RSSI value measured from a transmitter in $\mathbf{x}^*$. The term ε_i is the noise that accounts for the changing environment, the fluctuations on the channel, and the noise introduced at the receiver. Since the position of the target is fixed and does not change over time, we will omit $\mathbf{x}^*$ from the notation, so that $f(\mathbf{x}|\mathbf{x}^*) = f(\mathbf{x})$ from now on.

In this framework, $f(\mathbf{x})$ can be seen as a black-box function to be estimated from the noisy response y_i obtained at corresponding location $\mathbf{x}_i$. We will model the noise $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$ as an i.i.d. zero-mean Gaussian random variable with variance σ_n^2, independent of the location.

A crucial assumption, that will allow us to reduce the localization problem to a simple maximization problem, is that $f(\mathbf{x})$ exists and is a monotonic decreasing function of the distance, so that

$$f(d_1) > f(d_2) \Leftrightarrow d_1 < d_2. \tag{3.2}$$

The monotonic decreasing assumption is justified by the well-known Friis transmission equation. Such equation relates the power received by an antenna (P_r) with the distance d from the transmitting antenna and with the transmit power P_t. In particular we have

$$P_r \propto P_t - \beta \cdot \log(d), \tag{3.3}$$

where β is a parameter that depends on the medium and it is clear that $P_r(d)$ is a monotonic decreasing function of the distance. In fact, $f(\mathbf{x})$ should model the received power at a certain distance, so we can expect it to be similar in shape to (3.3), and to inherit its decreasing monotonicity. However, the free-space propagation model does not account for a number of non-idealities that are common in a real environment, where lot of reflections are involved. Therefore we should ensure not to tighten the shape of the function $f(\mathbf{x})$ too much to this model. Furthermore, different weather conditions and different environments may lead to very different model parameters, so that using a static model may be a very short-sighted and unsatisfactory strategy.

Consider now l received packets. The measured RSSIs values are stored in the vector $\mathbf{y}_l = [y_1, y_2, \ldots, y_l] \in \mathbb{R}^l$, while the corresponding locations are stored in the matrix $\mathbf{X}_l = [x_1, x_2, \ldots, x_l] \in \mathbb{R}^{l\times 2}$. The vectors $\mathbf{y}_l$ and $\mathbf{X}_l$ together form the input training dataset, denoted as $\mathcal{D}_l = \{\mathbf{y}_l, \mathbf{X}_l\}$.

How to efficiently and robustly estimate the function $f(\mathbf{x})$ given $\mathcal{D}_l$ is the subject

of Chapter 4, where we propose the well-known Gaussian Processes (GPs) as a tool to solve the regression problem. For the time being, assuming the function $f(\mathbf{x})$ to be available, and since $f(\mathbf{x})$ decreases as the distance between the source and the receiver increases, we can estimate the location of the source as

$$\hat{\mathbf{x}}_i = \arg\max_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}). \tag{3.4}$$

Once estimated the location $\hat{\mathbf{x}}_i$, we can either stop the localization process and return $\hat{\mathbf{x}}_i$ as an estimate of the device location, or ask for more measurements to perform a more refined localization. In the latter case, the question is where to collect those new measurements. Chapter 5 will answer this question, and we will propose a well-known machine learning technique to decide which point to sampled next.

As a final remark, it is important to point out that standard non-machine learning techniques will yield poor results in this scenario. For example, since we assumed zero-mean noise, one could think of collecting several measurements in the same location, and average them to obtain $f(\mathbf{x})$. Then, one would run a standard numerical-optimization method such as stochastic-approximation gradient-descent to find the location of the maximum of $f(\mathbf{x})$. Unfortunately, this approach is not suitable under our assumptions. In fact, measurements are received asynchronously and at unpredictable time, because, on the one hand, the source is assumed to be non-collaborative and transmits packets at random instants. On the other hand not all the transmitted packets are correctly received. Therefore it is challenging to collect enough packets in the same position, and perform a good estimation without wasting a lot of precious time.

Chapter 4

The regression problem

In the previous chapter we formulated our localization problem as a regression problem, where the observed inputs and outputs were related by an unknown function f as for (3.1). How to estimate such function, starting from partial and noisy observations, is the subject of this chapter.

4.1 Introduction

Generally speaking, regression solves the problem of learning input-output mapping from empirical continuous data. In particular, in supervised regression, we observe some inputs $\mathbf{x}_i$ and some corresponding outputs y_i, and we assume that $y_i = f(\mathbf{x}_i)$ for an unknown function f. Given a set of potentially noisy observations, we want to predict the value measured at a new candidate-location[1] $\mathbf{x}_*$ that we have not yet visited. To this end, we need to estimate $y_* = f(\mathbf{x}_*)$. This requires us to move from a finite set of observed values to a general regression function f that makes prediction for all possible input values.

To solve this problem two different approaches can be used. In the first one we only consider a particular class of functions, for example those linear with the inputs. All the functions belonging to such class will depend on a set of parameters, that can be estimated minimizing a certain loss function. The obvious drawback of this method is that, if the function does not model the data in a proper way, the prediction will be very poor. Furthermore, if the function becomes too complex, more parameters need to be estimated, and this could result in a poor reusability of the model. In fact one may be tempted to increase the flexibility of the approach by enriching the class of functions,

[1] Not to be confused with the real location of the device that we indicated as $\mathbf{x}^*$.

but this approach may generate overfitting problems, thus loosing the capability of the model to generalize the prediction beyond the training set.

The second approach consists in giving a prior probability to every possible function, where higher probability is given to functions that we consider to be more likely. For example, we could say that peeky functions are more likely than smooth functions, or that our regression function is likely to be continuous with zero-mean. Clearly, the set of possible functions is uncountable and infinite, so we need to properly explain how we can compute such probabilities in a finite time. Furthermore we need to say how to mathematically describe functions, i.e., how to translate the adjectives "smooth" or "peeky" into a mathematical quantity. In this work we will use this second approach, and to derive this prior probabilities over functions we will use Gaussian Processes (GPs), that are introduced. Detailed information on GPs can be found in [2].

4.2 Mathematical formulation of Gaussian Processes

A GP, in the context of supervised regression, defines a prior distribution over all the possible regression functions. Once some data is observed, this prior can be converted into a posterior distribution [30] to perform the prediction for any new candidate location. It is often useful to think of a GP as similar to a function, that instead of returning a scalar value for an arbitrary input value $\mathbf{x}_*$, returns the mean and the variance of a normal distribution that models the likelihood of all the possible values of f at $\mathbf{x}_*$ [3]. We use GP regression since it is robust, well known and it can learn and model non-linear relations as the one between the RSSI and the distance from the device.

More formally, a GP is a collection of random variables, any finite number of which have a joint Gaussian distribution [2]. If the prior on the regression function is a GP, then we write

$$f(\mathbf{x}) \sim \mathrm{GP}\Big(m(\mathbf{x}), \kappa(\mathbf{x}, \mathbf{x}')\Big) \tag{4.1}$$

where $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$ is the mean and $\kappa(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))^T]$ is the kernel or covariance function.

Since the prior on the regression function is a GP, such prior contains all the functions with mean $m(\mathbf{x})$ and covariance between points that follows $\kappa(\mathbf{x}, \mathbf{x}')$. In other words, all the functions that belong to the GP prior are different, but share the same mean and the same covariance function (that somehow reflects in the smoothness of the function).

4.2.1 Noise-free regression

We consider to begin with the simpler noise-free problem, where we collect l observations into the vector $\mathbf{y}_l$ and the corresponding l locations in the matrix $\mathbf{X}_l \in \mathbb{R}^{l\times 2}$. Under the noise-free assumption we have

$$y_i = f(\mathbf{x}_i), \quad i = 1, 2, \ldots, l. \tag{4.2}$$

Then, given some candidate-locations contained in the matrix $\mathbf{X}_* \in \mathbb{R}^{n_*\times 2}$ we want to predict the function outputs in such locations, denoted here as $\mathbf{y}_* \in \mathbb{R}^{n_*}$. By definition of GP, the joint distribution of the training outputs $\mathbf{y}_l$ and the test outputs $\mathbf{y}_*$ is Gaussian, i.e.

$$\begin{bmatrix}\mathbf{y}_l \\ \mathbf{y}_*\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu} \\ \boldsymbol{\mu}_*\end{bmatrix}, \begin{bmatrix}\mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**}\end{bmatrix}\right) \tag{4.3}$$

where $\boldsymbol{\mu} = m(\mathbf{X})$, $\boldsymbol{\mu}_* = m(\mathbf{X}_*)$, $k_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{K} = \kappa(\mathbf{X}_l, \mathbf{X}_l) \in \mathbb{R}^{l\times l}$, $\mathbf{K}_* = \kappa(\mathbf{X}_l, \mathbf{X}_*) \in \mathbb{R}^{l\times n_*}$ and $\mathbf{K}_{**} = \kappa(\mathbf{X}_*, \mathbf{X}_*) \in \mathbb{R}^{n_*\times n_*}$.

To get the posterior distribution over all the possible functions, i.e., the prediction at the candidate-locations $\mathbf{X}_*$, we restrict the prior to contain only those functions that agree with the observed data points $\mathbf{y}_l$. We can intuitively think of this operation as drawing functions from the prior rejecting the ones that disagree with the observations. In probabilistic terms, this operation corresponds to conditioning the joint distribution on the observations (see Section 9.3). Denoting with $\mathcal{R}(\mathbf{y}_*|\mathbf{X}_*, \mathbf{X}_l, \mathbf{y}_l)$ such a conditional distribution, we hence have

$$\mathcal{R}(\mathbf{y}_*|\mathbf{X}_*, \mathbf{X}_l, \mathbf{y}_l) = \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}), \tag{4.4}$$

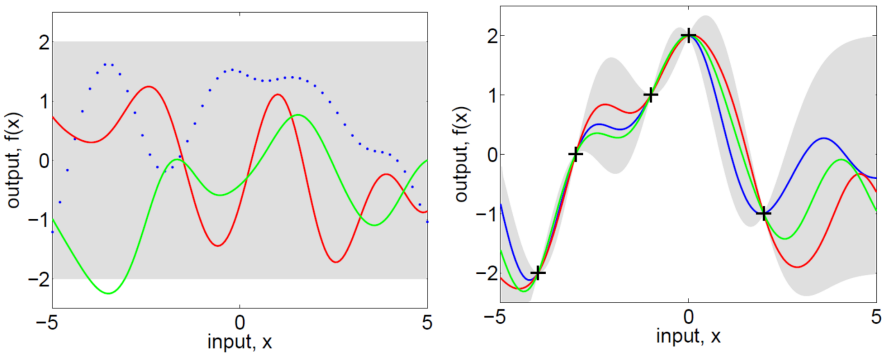
where

$$\hat{\boldsymbol{\mu}} = m(\mathbf{X}_*) + \mathbf{K}_*^T\mathbf{K}^{-1}(\mathbf{y}_l - m(\mathbf{X})), \tag{4.5a}$$

$$\hat{\boldsymbol{\Sigma}} = \mathbf{K}_{**} - \mathbf{K}_*^T\mathbf{K}^{-1}\mathbf{K}_*. \tag{4.5b}$$

The outcome of the prediction is therefore a vector of jointly Gaussian random variables, where the mean of each variable is contained in the vector $\hat{\boldsymbol{\mu}}$, while the correlation between each pair of variables is described by $\hat{\boldsymbol{\Sigma}}$. This means that, for each point $\mathbf{x}_{*,i} \in \mathbf{X}_*$, the result of the prediction is a Gaussian random variable with mean $\hat{\boldsymbol{\mu}}_i$ and variance $\hat{\boldsymbol{\Sigma}}_{i,i}$.

A pictorial example of the posterior probability derivation is shown in Figure 4.1. More specifically, Figure 4.1a shows three functions drawn at random from the prior, while Figure 4.1b shows three functions drawn from the posterior, i.e., the prior conditioned on the 5 observations. It is interesting to note that, under the noise free assumption, GP acts as an interpolator, i.e., if we ask the prediction for a value $\mathbf{x}_i$ that has already been seen, GP returns y_i with no uncertainty. In both plots the shaded area represents the pointwise mean plus and minus twice the standard deviation (95% confidence interval).



(a) Prior distribution (b) Posterior distribution

Fig. 4.1: Example of functions' drawing from the prior and posterior distribution [2].

4.2.2 Noisy regression

In a typical scenario we have only access to a noisy version of the measurements, $y_i = f(\mathbf{x}_i) + \varepsilon_i$. If we assume the noise to be modeled as a Gaussian independent identically distributed i.i.d. random variable, $\varepsilon_i \sim \mathcal{N}(0, \sigma_n^2)$, the conditional covariance of the observations given the location vector $\mathbf{X}_l$ is given by

$$\text{cov}(\mathbf{y}_l|\mathbf{X}_l) = \mathbf{K} + \sigma_n^2\mathbf{I} = \mathbf{K}_y, \tag{4.6}$$

where $\mathbf{I}$ is the identity matrix of size $l \times l$. Equation (4.4) still holds, but the mean vector and covariance matrix that account for the noise are computed as

$$\hat{\boldsymbol{\mu}} = m(\mathbf{X}_*) + \mathbf{K}_*^T\mathbf{K}_y^{-1}(\mathbf{y}_l - m(\mathbf{X})), \tag{4.7a}$$

$$\hat{\boldsymbol{\Sigma}} = \mathbf{K}_{**} - \mathbf{K}_*^T\mathbf{K}_y^{-1}\mathbf{K}_*. \tag{4.7b}$$

It is common, though not necessary, to consider GPs as a zero mean functions. Note that this is not a drastic limitation, since the mean of the posterior process is not confined to be zero. We will follow this approach considering a zero-mean prior, leaving the use of explicit basis functions as mean functions for future developments.

4.2.3 Kernel function description

So far, the kernel function $\kappa(\mathbf{x}, \mathbf{x}')$ has not explicitly been defined, although the predictive performance of GP depends mostly on the chosen kernel [30]. The kernel function $\kappa(\mathbf{x}, \mathbf{x}')$ is a mathematical model that describes the correlation exhibited from different observations taken at locations $\mathbf{x}$ and $\mathbf{x}'$.

Several different functions can be used, but the Squared Exponential (SE) kernel, shown below, is probably the most widely employed. If we consider $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^D$, the multidimensional SE kernel has the form

$$\kappa_{SE}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\Big(-(\mathbf{x} - \mathbf{x}')^T \mathbf{M} (\mathbf{x} - \mathbf{x}')\Big), \tag{4.8}$$

where, assuming the kernel to be isotropic, $\mathbf{M} \in \mathbb{R}^{D \times D} = l^{-2} \cdot \mathbf{I}$. In our case, since $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^2$, equation (4.8) reduces to

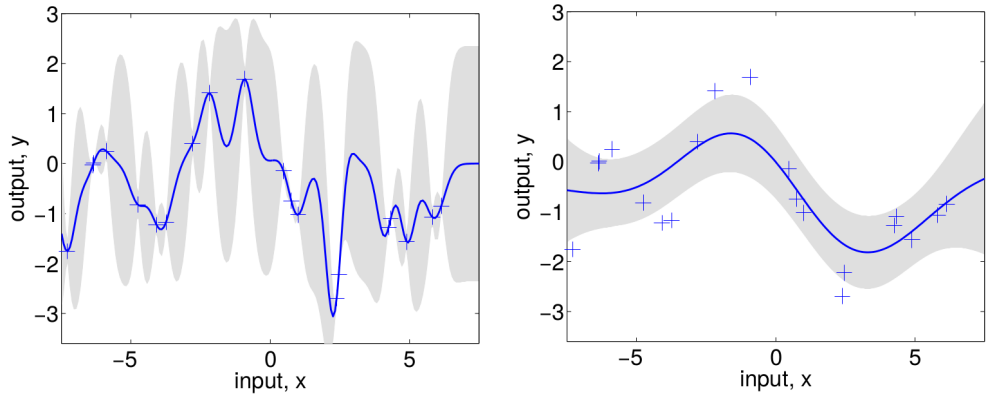
$$\kappa_{SE}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|}{l^2}\right). \tag{4.9}$$

The SE kernel depends on two different parameters that need to be properly estimated to obtain a good prediction. The characteristic horizontal length scale, l, defines the scale at which correlation between points can be observed. On the one hand, when $l \to 0$ points will be almost uncorrelated also if they are very close one another, and the resulting function will peeky and bumpy, i.e., similar to what is shown in Figure 4.2a. On the other hand, if $l \gg 1$, points will exhibit a strong correlation also if they are far away, and the function will look smoother, as in Figure 4.2b.

4.2.4 Hyperparameters estimation

To compute the posterior distribution, three parameters need to be estimated, namely $\boldsymbol{\theta} = \{\sigma_f, l, \sigma_n\}$. We chose to perform such optimization offline, relying on a dataset acquired in an initial experiment. Such dataset is composed of m RSSI measurements $\mathbf{y} = [y_1, y_2, \ldots, y_m]$ collected at corresponding locations $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_m]$.

We recall that for a multivariate Gaussian distribution with mean vector $\mathbf{m}$ and



(a) Small length scale l.

(b) Large length scale l.

Fig. 4.2: Effect of the characteristic length scale l on the GP predicted function with SE kernel [2].

covariance matrix $\mathbf{\Sigma}$, the joint probability density function is given by

$$p(\mathbf{x}) = (2\pi)^{n/2}|\mathbf{\Sigma}|^{-1/2} \cdot \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{m})\right). \tag{4.10}$$

The vector of hyperparameters $\boldsymbol{\theta}$ is estimated using a Bayesian approach that maximizes the marginal log-likelihood, defined as

$$p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) = \int_{\mathbf{f}} p(\mathbf{y}|\mathbf{f},\mathbf{X},\boldsymbol{\theta})p(\mathbf{f}|\mathbf{X},\boldsymbol{\theta})d\mathbf{f} \tag{4.11}$$

where $\mathbf{f}$ is the prior Gaussian vector that has been marginalized out. According to the covariance model expressed in (4.9), and having assumed a zero mean prior, we can express the PDF of $\mathbf{f}|\mathbf{X},\boldsymbol{\theta}$ as

$$\log p(\mathbf{f}|\mathbf{X},\boldsymbol{\theta}) = -\frac{1}{2}\left(\mathbf{f}^T\mathbf{K}^{-1}\mathbf{f} + \log|\mathbf{K}| + n\log(2\pi)\right). \tag{4.12}$$

Similarly, having assumed the noise affecting the RSSI measurements to be i.i.d. with variance σ_n^2, we can write the PDF of $\mathbf{y}|\mathbf{f},\mathbf{X},\boldsymbol{\theta}$ as

$$\log p(\mathbf{y}|\mathbf{f},\mathbf{X},\boldsymbol{\theta}) = -\frac{1}{2}\left(\frac{\|\mathbf{y}-\mathbf{f}\|}{\sigma_n^2} + \log(n\sigma_n^2) + n\log(2\pi)\right). \tag{4.13}$$

Equation (4.11) can now be rewritten using (4.12) and (4.13) and is finally given by [2]

$$\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2}\left(\mathbf{y}^T\mathbf{K}_y^{-1}\mathbf{y} + \log|\mathbf{K}_y| + n\log(2\pi)\right). \tag{4.14}$$

Three terms come into play in equation (4.14):

1. The first term, $\mathbf{y}^T\mathbf{K}_y^{-1}\mathbf{y}$, is a data fit term: the better the model fits the data, the smaller this value will be, and the likelihood will consequently improve (note the minus sign in front of the expression).

2. The second term, $\log|\mathbf{K}_\mathbf{y}|$, accounts for the model complexity: if the model is richer, the matrix will be almost diagonal and the term will increase, thus decreasing the likelihood [2]. Simple models are instead encouraged by this term, thus encouraging GP not to overfit the data (see Section 4.2.5 for more details on overfitting)

3. The third term, $n\log(2\pi)$, is just a normalization factor.

Equation (4.14), given the observations $\mathbf{y}$ measured at corresponding locations $\mathbf{X}$, only depends on $\boldsymbol{\theta}$, which can now be estimated as

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}}\Big(\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})\Big). \tag{4.15}$$

To perform such maximization, we could use a grid search approach, where we define, for each θ_i a set of values, and we compute the log likelihood for each tuplet of values that lies in the Cartesian product of the sets. We can then choose as the optimum hyperparameters the vector that maximizes (4.14). Due to the curse of dimensionality, the required number of evaluations grows exponentially with respect to the number of parameters. Given d hyperparameters (for us $d = 3$) and assuming a set with p points along each parameter's covariate, the number of evaluations required is

$$n = p^d, \tag{4.16}$$

which can rapidly become infeasible. Furthermore, once the optimization is complete, grid search does not naturally allow for adding new evaluations. New evaluations would require to either redefine the grid and lose the past evaluations, or fill the gaps from the previous grid and typically lose the regular structure of the grid.

On the other hand, we can choose a more efficient gradient ascent approach that

requires to derive the gradient of (4.14) with respect to $\boldsymbol{\theta}$. Using the properties

$$\frac{\partial}{\partial\theta}\mathbf{K}^{-1} = -\mathbf{K}^{-1}\left(\frac{\partial\mathbf{K}}{\partial\theta}\right)\mathbf{K}^{-1}, \tag{4.17a}$$

$$\frac{\partial}{\partial\theta}\log|\mathbf{K}| = \mathrm{tr}\left(\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\theta}\right), \tag{4.17b}$$

the derivative of (4.14) with respect to θ_i can be expressed as

$$\frac{\partial}{\partial\theta_i}\log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) = \frac{1}{2}\cdot\mathrm{tr}\left((\boldsymbol{\alpha}\boldsymbol{\alpha}^T - \mathbf{K}_y^{-1})\frac{\partial}{\partial\theta_i}\mathbf{K}_y^{-1}\right), \tag{4.18}$$

where $\boldsymbol{\alpha} = \mathbf{K}_y^{-1}\mathbf{y}$ and $\mathrm{tr}(\mathbf{A})$ is the trace of the matrix $\mathbf{A}$. Using the SE kernel, we can easily compute the partial derivative of $\mathbf{K}_y$ with respect to each hyperparameter. In particular

$$\frac{\partial\mathbf{K}_y}{\partial{\sigma_f}^2} = 2\sigma_f\begin{bmatrix} 1 & \exp\left(\frac{-\|x_1-x_2\|^2}{\theta_2^2}\right) & ... & \exp\left(\frac{-\|x_1-x_n\|^2}{\theta_2^2}\right) \\ \exp\left(\frac{-\|x_2-x_1\|^2}{\theta_2^2}\right) & 1 & ... & \exp\left(\frac{-\|x_2-x_n\|^2}{\theta_2^2}\right) \\ ... & ... & ... & ... \end{bmatrix}, \tag{4.19}$$

and deriving with respect to the length scale we obtain

$$\frac{\partial\mathbf{K}_y}{\partial l^2} = \frac{2\sigma_f^2}{l^3}\begin{bmatrix} 1 & \exp\left(\frac{-\|x_1-x_2\|^2}{\theta_2^2}\right) & ... & \exp\left(\frac{-\|x_1-x_n\|^2}{\theta_2^2}\right) \\ \exp\left(\frac{-\|x_2-x_1\|^2}{\theta_2^2}\right) & 1 & ... & \exp\left(\frac{-\|x_2-x_n\|^2}{\theta_2^2}\right) \\ ... & ... & ... & ... \end{bmatrix}\cdot\mathbf{U}, \tag{4.20}$$

where the symbol $\cdot$ represents the inner product between matrices, and

$$\mathbf{U} = \begin{bmatrix} 0 & \|x_1-x_2\|^2 & ... & \|x_1-x_n\|^2 \\ \|x_2-x_1\|^2 & 0 & ... & \|x_2-x_n\|^2 \\ ... & ... & ... & ... \end{bmatrix}. \tag{4.21}$$

We are now able, using gradient ascent or Newton methods, to find the value of the hyperparameters $\boldsymbol{\theta}_*$ that maximizes the marginal log likelihood expressed in (4.11).

4.2.5 A cross-validation approach to prevent overfitting

According to the maximization process described in the previous section, we may be temped to use all the data in our possession to find the optimal value for the hyper-

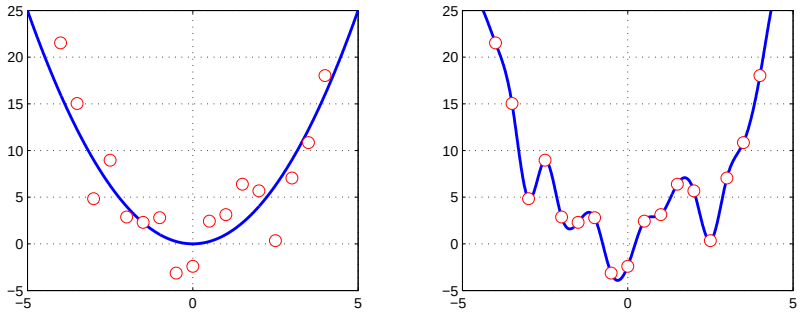


Fig. 4.3: Example of correct fitting (left) and overfitting (right).

parameters. Using this strategy, however, we may overfit the data, i.e., our statistical model may describe random errors or noise instead of the underlying relationship between variables.

Figure 4.3 visually exemplifies such problem. In the example we draw some samples from the function $y = x^2$ adding a small amount of Gaussian noise. On the left side we plot the real function $y = x^2$ and the samples that have been used to find the optimum values of the hyperparameters. The right side of the pictures shows an example of overfitting. The inferred function fits perfectly the observed data, but it is modelling the noise instead of the underlying relation between the variables. If we now ask the model to perform predictions for new points using such regression function, it will yield poor results. A model that has been overfitted will generally have poor predictive performance, as it can magnify minor fluctuations in the data.

To avoid overfitting, we find the optimal hyperparameters using a subset of the available data $\mathcal{T} \subset \mathcal{D}$, and we then measure the prediction error on the remaining test set $\mathcal{D} \setminus \mathcal{T}$. This approach is called cross-validation, and its aim is to ensure that the parameters that maximize the likelihood also yield a small prediction error. We describe here how the hyperparameters were estimated in our work and how we ensured that data was not overfit.

We equally and randomly divided the m available observations in two parts: a training set $\mathcal{T}$, $|\mathcal{T}| = m/2$ and a test set $\mathcal{D} \setminus \mathcal{T}$. We now define the three sets $\mathcal{L} = \{l_1, l_2, \ldots, l_p\}$, $\mathcal{S} = \{\sigma_1, \sigma_2, \ldots, \sigma_p\}$ and $\mathcal{Q} = \{q_1, q_2, \ldots, q_p\}$. We assume each set to contain p possible values for each hyperparameter: $l \in \mathcal{L}$, $\sigma_f \in \mathcal{S}$ and $\sigma_n \in \mathcal{Q}$. For each triplet $\{l, \sigma_f, \sigma_n\}$ in the Cartesian product $\mathcal{L} \times \mathcal{S} \times \mathcal{Q}$, using the training set $\mathcal{T}$, we compute the marginal log likelihood according to (4.14). Then, using the data contained in the test set $\mathcal{D}$, we compute the test error using the Mean Squared Error

(MSE) i.e.,

$$\text{MSE}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}(y_i(\mathbf{x}_i) - \hat{y}_i)^2 \tag{4.22}$$

where $\hat{y}_i$ is the estimation given by the GP at location $\mathbf{x}_i$. If the triplet $\boldsymbol{\theta}$ that maximizes the likelihood corresponds to a small MSE, than we know that we are not overfitting the data.

The outcome of such process for our observations is shown in Figure 4.4, where for graphical convenience the hyperparameter σ_n^2 has been fixed. From Figure 4.4 we observe that choosing $l = e^{-1.6} \simeq 0.20$ and $\sigma_f = e^{1.2} \simeq 3.32$ we are able to maximize the log likelihood while keeping the corresponding MSE low. That choice of hyperparameters is therefore reasonable and acceptable having used (4.14) as objective function to be maximized, and according to the available observations, overfitting is not occurring.

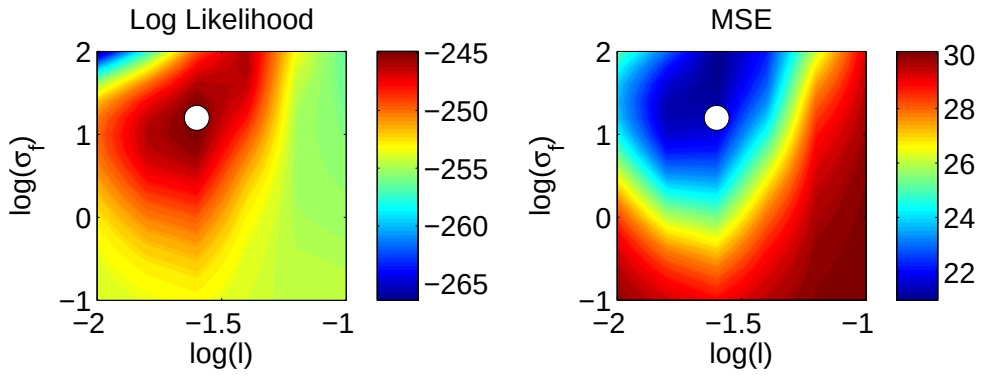


Fig. 4.4: Marginal log-likelihood and MSE for different values of the hyperparameters (l, σ_f). The white circle in both plots denotes the maximum value of the likelihood.

4.2.6 Numerical issues and an example of application

As shown in equations (4.7a) and (4.7b), the derivation of the posterior distribution using a GP depends on the computation of $\mathbf{K}_y^{-1}$. For reasons of numerical stability, it is discouraged to directly invert the matrix $\mathbf{K}_y$. We can instead exploit the fact that $\mathbf{K}_y$ is a covariance matrix, therefore symmetric and positive definite to factorize it using the Cholesky decomposition. The matrix $\mathbf{K}_y$ can be decomposed into a product of a lower triangular matrix $\mathbf{L}$ and its transposed,

$$\mathbf{K}_y = \mathbf{L}\mathbf{L}^T, \tag{4.23}$$

where $\mathbf{L}$ is called the Cholesky factor. This is particularly useful when we are asked to solve linear systems in the form $\mathbf{x} = \mathbf{K}_y^{-1}\mathbf{b} \Leftrightarrow \mathbf{K}_y\mathbf{x} = \mathbf{b}$, because we can first solve the triangular system $\mathbf{L}\mathbf{y} = \mathbf{b}$ by forward substitution for $\mathbf{y}$. Then we solve the triangular system $\mathbf{L}^T\mathbf{x} = \mathbf{z}$ by backward substitution (see Section 9.2 for the details). Both those operations require $n^2/2$ operations, while the Cholesky decomposition takes time $n^3/6$. We will denote this procedure with the notation $\mathbf{x} = \mathbf{L}^T \setminus (\mathbf{L} \setminus \mathbf{b})$. Using this approach equations (4.7a) and (4.7b) can be rewritten as

$$\hat{\boldsymbol{\mu}} = m(\mathbf{X}_*) + \mathbf{K}_* \left[\mathbf{L}^T \setminus \left(\mathbf{L} \setminus (\mathbf{y}_l - m(\mathbf{X})\right)\right], \tag{4.24a}$$

$$\mathbf{U} = \mathbf{L} \setminus \mathbf{K}_*, \tag{4.24b}$$

$$\hat{\boldsymbol{\Sigma}} = \mathbf{K}_{**} - \mathbf{U}^T\mathbf{U}. \tag{4.24c}$$

where the mean vector and covariance matrix are computed without explicitly inverting $\mathbf{K}_y$.

It is also important to observe that, using (4.24c), we are actually computing the entire covariance matrix, thus getting more information than what we usually need. In particular, the cross correlation between all the variables is not of particular interest for us. The important information resides is the variance $\boldsymbol{\sigma}_*^2$, that could in principle be obtained extracting the diagonal entries from $\hat{\boldsymbol{\Sigma}}$. This approach, correct in principle, leads to very high computational complexity, especially for the computation of the matrix product $\mathbf{U}^T\mathbf{U}$, that requires $O(n \cdot n_*^2)$ operations. To implement the algorithm in a more efficient way, using the SE kernel, we can reduce the computation of the variance to

$$\sigma_{*,i}^2 = \sigma_f^2 - \sum_{j=1}^{n} \mathbf{U}(j,i)^2, \quad i = 1, 2, ..., n_*, \tag{4.25}$$

thus saving both memory and time.

To conclude the chapter, we present here a very short toy example of prediction using a GP. We draw 400 samples from the function $f(\mathbf{x}) = 10 - 10\log(\|\mathbf{x}\| + 0.1)$, $f : \mathbb{R}^2 \to \mathbb{R}$, that are shown in the left-hand side of Figure 4.5. Then we add some Gaussian noise, with variance $\sigma_n^2 = 9$ (the noisy measurements are shown in the central part of the Figure). Finally, after the hyperparameters estimation using a grid search approach, we perform the prediction using a GP. The estimated mean value for the function, $\hat{\boldsymbol{\mu}}$ according to (4.24a), is shown in the right-hand side of Figure 4.5.

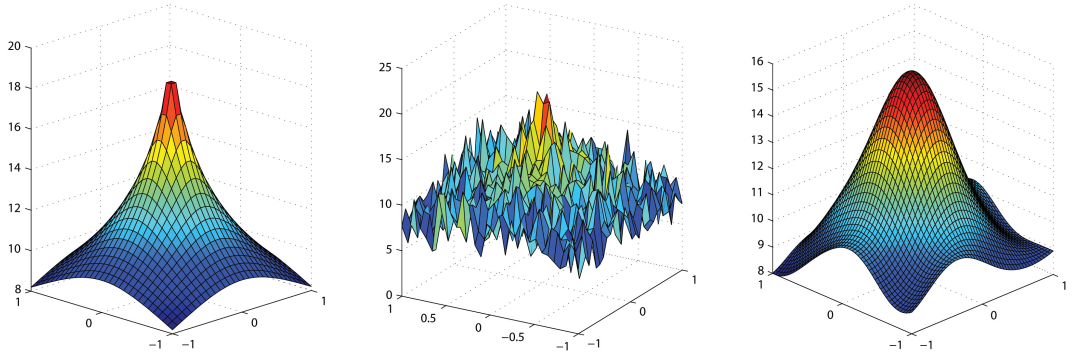


Fig. 4.5: Example of bidimensional prediction using GP.

Chapter 5

Bayesian Optimization

5.1 Motivation and Introduction

In the previous chapter we introduced Gaussian Processes, a regression tool that can be used to learn the input-output relation between variables starting from some noisy observations. The solution of the regression problem, using a GP, is a Gaussian random variable for each candidate location $\mathbf{x}_*$. Such random variable describes the likelihood of each possible value for the function f at location $\mathbf{x}_*$.

According to (3.4), to estimate the location of the source, once the regression problem has been solved, one simply needs to find the argument that maximizes $\hat{\mu}(\mathbf{x})$. Such location clearly changes when new observations are available, according to (4.7a) that depends on $\mathbf{y}_l$. In a nutshell, the RSSI y_{l+1} measured from a new packet at location $\mathbf{x}_{l+1}$ changes our prediction and consequently our estimation of the source's location. The question now is whether or not there is a specific way to collect RSSIs in particular locations so that we can localize the source in a faster and more precise way. Consider m measurements $[y_1, y_2, \ldots, y_m]$ collected at corresponding locations $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m]$. Using this dataset, we can compute the posterior distribution and correspondingly estimate the source location. Since the UAV with the onboard monitor is free to move to any location, we now need to decide the position $\mathbf{x}_{m+1}$ where the drone is sent to collect the next measurement y_{m+1}. How to choose such $\mathbf{x}_{m+1}$ in order to improve the accuracy of our estimation and guarantee a fast convergence is the subject of the next sections. At this stage it is important to observe that the variance of the posterior distribution reflects our confidence in the estimation for $f(\mathbf{x})$. If the variance is small, then our estimation is expected to be precise. On the contrary, a high variance generally associated with high uncertainty in the prediction. In the next sections we will see how

such uncertainty measure can be used to speed up the localization process and to make it more precise, i.e., how it can be used to determine $\mathbf{x}_{m+1}$. Using such uncertainty for making the localization process efficient can be considered the main contribution of this work.

5.2 Localization using Bayesian Optimization

Bayesian Optimization (BO) is a robust, reliable, well-known machine learning technique that tries to efficiently maximize a function that is costly to evaluate. A detailed explanation of BO can be found in [3].

Generally speaking, if the objective is to determine the shape of $f(\mathbf{x})$, the uncertainty of the posterior distribution can guide the sampling process. Following this intuition, more samples should be collected from areas of high uncertainty to reduce the variance and improve the accuracy of the prediction. However, the sampling process can be made more efficient if our attention is focused on the localization of the maximum of $f(\mathbf{x})$. In fact, the peak of $f(\mathbf{x})$ is likely observed in correspondence of the actual location of the source device, and therefore achieving a good precision in the estimation of the maximum is crucial for an accurate prediction of the source's position.

A lot of scientific literature has been devoted to the general problem of maximizing a non linear function over a compact set. Usually, the function is assumed to be convex or at least cheap to evaluate. Bayesian optimization is instead a strategy for finding the extrema of functions that are expensive to evaluate [3]. For us, measuring the RSSI at a location $\mathbf{x}$ is costly since the UAV needs to physically move there and wait until a new packet is received, and this is expensive in terms of both time and energy.

Recently, Bayesian optimization appeared in numerous applications as a solution for the efficient maximization of black box functions as $f(\mathbf{x})$. In [31], authors used Bayesian optimization to perform the estimation of a set of parameters for maximizing the speed of a Sony robot. In [32], instead, Bayesian optimization is used to learn a policy for simulated driving tasks. In [33] the authors use Bayesian optimization to select a set of sensors in a dynamic system. The objective is to find a set that reduces the MSE of the predictions made by the sensor network. Gaussian processes and Bayesian optimization are jointly used in [34] to solve a difficult optimization problem - finding weights in a neural network controller - using an order of magnitude fewer samples than reported elsewhere. Those few examples show that Bayesian optimization can become a very useful technique to restrain the cost of a function estimation.

5.3 The acquisition function

For notation simplicity, in this chapter we will restrict ourselves to the localization of a single device. An extension to multiple devices is discussed later. Let us recall that $\mathbf{y}_l$ denotes the vector that contains l received measurements, while $\mathbf{x}_l$ is the current position of the UAV. Note that $l \in \mathbb{N}$ is increased by one unit each time a new packet is received. To sample efficiently, Bayesian optimization uses an **acquisition function** $\beta(\mathbf{x})$ to determine the next point $\mathbf{x}_{l+1}$ to be sampled, where

$$\mathbf{x}_{l+1} = \arg\max_{\mathbf{x}\in\mathcal{X}} \beta(\mathbf{x} \,|\, \mathbf{y}_l, \mathbf{X}_l). \tag{5.1}$$

The function $\beta(\cdot)$ should act as a reward function and represents a trade-off between the following two competing goals:

1. **Exploration**: it should encourage sampling of areas of high uncertainty where the maximum could potentially be in spite of the fact that the current estimation of $f(\mathbf{x})$ is low.

2. **Exploitation**: it should encourage the sampling of zones that previous measurements have indicated as potential global maximum.

To understand the acquisition function trade off, we can consider the analogy with the well known multi-armed bandit problem, where a gambler stands in front of a row of slot machines and has to decide in which order to play them and how many times to play each machine. When played, each machine provides a random reward from a distribution specific to that machine. The objective of the gambler is to maximize the sum of rewards earned through a sequence of lever pulls [35]. Similarly to that problem, we have a set of points and we can choose to sample the function in one of those points. What we need to decide is whether to stick with good machines that we know better (i.e., points with high mean and low variance, **exploitation**) or explore others machines hoping to get a better reward (i.e., points with high variance, **exploration**).

Figure 5.1 shows a typical run of Bayesian optimization on a simple one-dimensional toy problem. At the beginning, the dataset counts $l = 6$ observations. The black line is the mean predicted value for the function, while the grey shaded area is the pointwise mean plus and minus two times the standard deviation of the prediction. We can observe some areas of higher uncertainty (around $x = 6$), and other zones where the variance is small ($x \leq 1$). The acquisition function $\beta(x)$ is computed for each $x \in \mathcal{X}$ and the next point to be sampled is the argument that maximizes $\beta(x)$. A

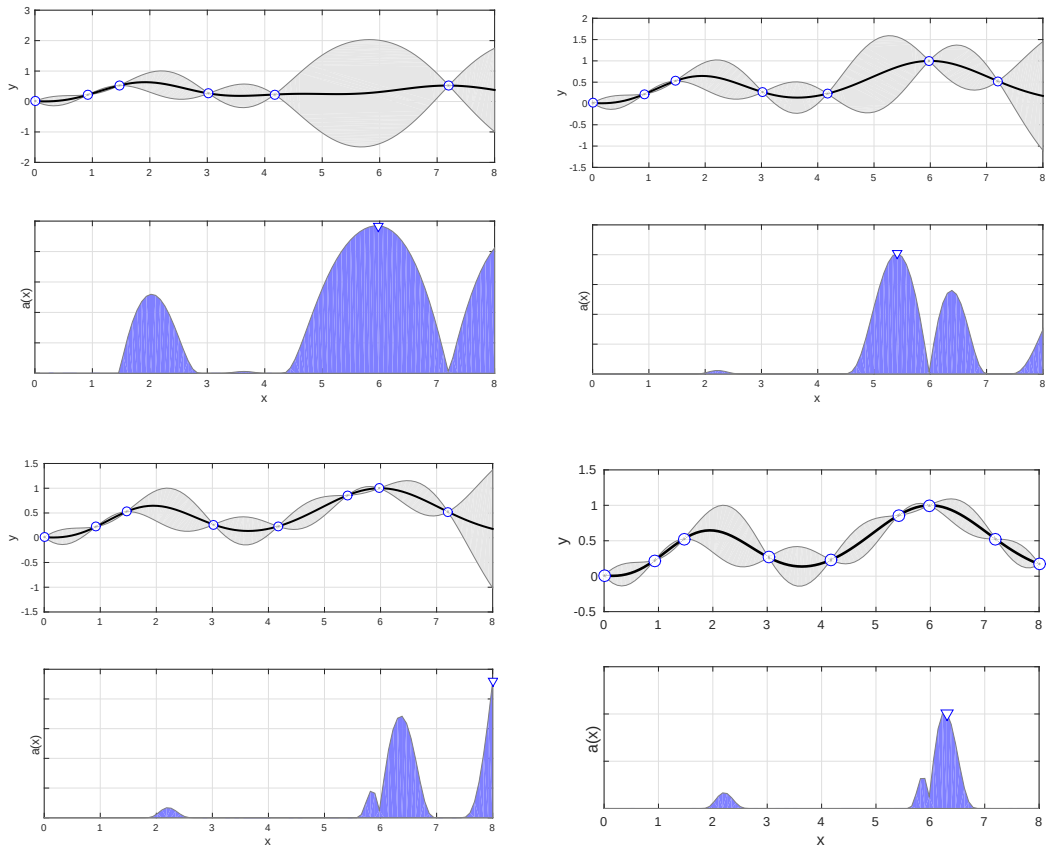


Fig. 5.1: Toy example of a Bayesian optimization process.

possible example of such function is shown in the lower plot for each step (purple shaded function). The new sample is then collected and added to the dataset, a new estimation is performed and the whole procedure is repeated again. The general algorithm for Bayesian optimization is shown in Algorithm 1.

Algorithm 1 Bayesian Active Learning

1: Update the prediction computing the mean and the variance for each $\mathbf{x} \in \mathcal{X}$
2: Choose $\mathbf{x}_{l+1} = \arg\max_{\mathbf{x}\in\mathcal{X}} \beta(\mathbf{x} \,|\, \mathbf{y}_l, \mathbf{X}_l)$
3: Sample the function at $\mathbf{x}_{l+1}$, i.e., collect $y_{l+1} = f(\mathbf{x}_{n+1})$
4: Augment the dataset $\mathbf{y}_{l+1} = \mathbf{y}_l \cup \{y_{l+1}\}$; $\mathbf{X}_{l+1} = \mathbf{X}_l \cup \{\mathbf{x}_{l+1}\}$
5: $l = l + 1$ and go to step 1.

As one can easily argue, two are the things that lie at the core of Bayesian optimization.

1. How we estimate the function given some observation, and in particular its mean

and variance at different candidate-locations. To address this, we use GPs, that have been described in the previous chapter. Note that we use GPs in this framework, although other methods such as neural networks can also be used with Bayesian optimization [36].

2. How we choose the function $\beta(\cdot)$, which decides the next point to be sampled based on the current estimation. On the one hand, if $\beta(\cdot)$ pushes too much toward exploration we may end up having a non precise estimation. On the other hand, if we focus our attention only on the current estimated maximum, i.e., we care more about exploitation, we may end up not recognizing the real peak of the function that may be hidden in areas of high uncertainty.

We show now different approaches for choosing the acquisition function $\beta(\cdot)$.

5.3.1 Improvement based acquisition function

Let $\mu^+ = \max_{\mathbf{x}\in\mathbf{X}_l}\big[\hat{\boldsymbol{\mu}}(\mathbf{x})\big]$, where we recall that, according to (4.7a), $\hat{\boldsymbol{\mu}}(\mathbf{x})$ is the posterior mean as computed from the GP. The first and easiest acquisition function aims at maximizing the probability of improvement over μ^+. The probability of improvement itself can be used as an acquisition function and is expressed for each point $\mathbf{x} \in \mathcal{X}$ as

$$\mathbf{PI}(\mathbf{x}|\mathbf{y}_l, \mathbf{X}_l) = P(f(\mathbf{x}) \geq \mu^+) = \Phi\left(\frac{\hat{\mu}(\mathbf{x}) - \mu^+}{\hat{\sigma}(\mathbf{x})}\right) \quad \mathbf{x} \in \mathcal{X}, \tag{5.2}$$

where $\Phi(\cdot)$ is the Cumulative Distribution Function (CDF) of the Gaussian distribution with zero mean and unit variance. Figure 5.2 shows an example of such computation for a toy function.

The intuitive drawback of such approach is that it is purely exploitative: points that have a high probability of yielding an infinitesimal improvement will be always preferred to points that could give an higher gain with a lower probability. Therefore, if the initial estimation of the function is not enough refined, the algorithm may force the UAV to collect packets from a small area while leaving completely unexplored some other zones that may contain the real peak of the function. To avoid that, we can add the parameter $\xi \geq 0$ and modify (5.2) into the Modified Probability of Improvement (MPI) acquisition function, defined as

$$\mathbf{MPI}(\mathbf{x}|\mathbf{y}_l, \mathbf{X}_l) = P\Big(f(\mathbf{x}) \geq (\mu^+ + \xi)\Big) = \Phi\left(\frac{\hat{\mu}(\mathbf{x}) - \mu^+ - \xi}{\hat{\sigma}(\mathbf{x})}\right) \quad \mathbf{x} \in \mathcal{X}. \tag{5.3}$$

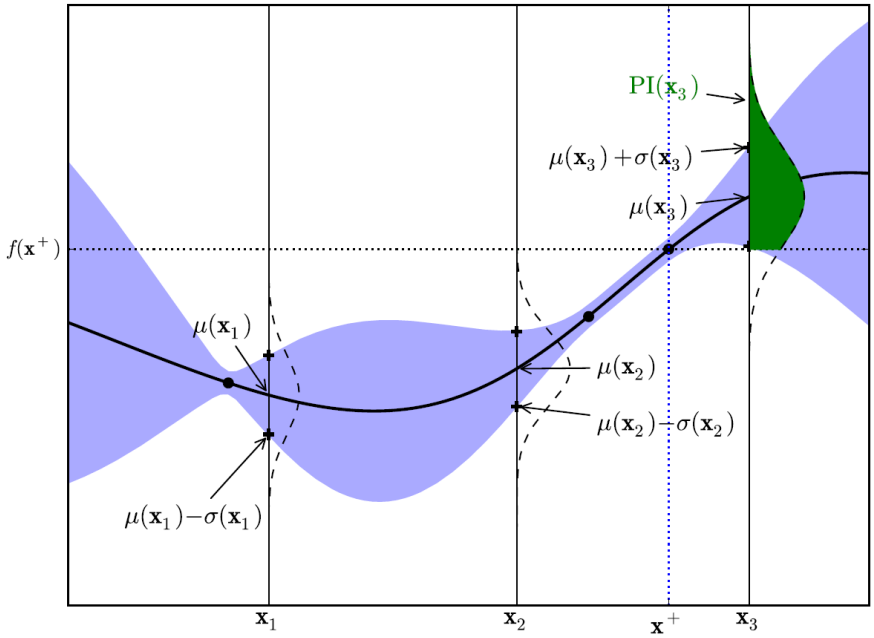


Fig. 5.2: Visual example of PI based acquisition function computation [3].

The idea is to use such parameter to encourage the algorithm choosing points that yield an improvement greater than ξ. Therefore, tuning ξ, we can explicitly force the algorithm to explore ($\xi \gg 1$) or to refine the position of the maximum ($\xi \ll 1$). As suggested in [3], the basic idea is to start with a high value of ξ and explore the whole environment, lowering ξ at each step thus progressively pushing the algorithm toward "exploitation" in order to refine the position of the maximum.

5.3.2 Expected Improvement acquisition function

We consider now a different acquisition function that not only takes into account the probability of improvement, but also the magnitude of the improvement that a point could potentially yield. The idea is to pick a new point $\mathbf{x}_{l+1}$ that minimizes the expected deviation from the true maximum of the mean function, i.e.

$$\mathbf{x}_{l+1} = \underset{\mathbf{x}\in\mathcal{X}}{\operatorname{argmin}} \left[\mathbb{E}\Big(\|\hat{\mu}_{l+1}(\mathbf{x}) - \mu^*\| \,\Big|\, \mathbf{y}_l, \mathbf{X}_l \Big) \right] \tag{5.4}$$

$$= \underset{\mathbf{x}\in\mathcal{X}}{\operatorname{argmin}} \int \|\hat{\mu}_{l+1}(\mathbf{x}) - \mu^*\| \cdot p(\hat{\mu}_{l+1} | \, \mathbf{y}_l, \mathbf{X}_l) d\hat{\mu}_{l+1} \tag{5.5}$$

where $\mu^* = \max_{\mathbf{x}\in\mathcal{X}}\big[f(\mathbf{x})\big]$ is the true maximum of the function. However, due to the computational efforts required to compute (5.5), we can use a simpler approach that consists in maximizing the expected improvement with respect to μ^+. We define the improvement function as

$$\mathbf{I}(\mathbf{x}) = \max\{0, \hat{\mu}(\mathbf{x}) - \mu^+\}, \tag{5.6}$$

i.e., $\mathbf{I}(\mathbf{x})$ is set to zero if the predicted mean in $\mathbf{x}$ does not exceed μ^+, and is set to $\hat{\mu}(\mathbf{x}) - \mu^+$ otherwise. In other words, $\mathbf{I}(\mathbf{x}) > 0$ only for that points that could yield an improvement over μ^+. Then, the new point to be sampled can be obtained maximizing the expected value of the improvement function, that combines somehow the magnitude of the improvement with its probability. If we assume the prediction to be distributed according to a Guassian distribution, then the likelihood of the improvement can be computed as

$$\mathbf{LI}(\mathbf{x}) = \frac{1}{\sqrt{2\pi}\hat{\sigma}(\mathbf{x})} \exp\left(- \frac{\big(\mathbf{x} - \mu^+ - \mathbf{I}(\mathbf{x})\big)^2}{2\hat{\sigma}(\mathbf{x})^2} \right), \tag{5.7}$$

and the corresponding expected value, to be maximized, can be obtained solving the integral

$$\mathbf{EI}(\mathbf{x}) = \int_{I=0}^{I=+\infty} \mathbf{LI}(\mathbf{x}) \cdot \mathbf{I}(\mathbf{x}) d\mathbf{I}. \tag{5.8}$$

The integral can be evaluated analytically using the following equations

$$\mathbf{EI}(\mathbf{x}) = \begin{cases} (\hat{\mu}(\mathbf{x}) - \mu^+)\Phi(Z) + \hat{\sigma}(\mathbf{x})\phi(Z), & \text{if } \hat{\sigma}(\mathbf{x}) > 0; \\ 0, & \text{if } \hat{\sigma}(\mathbf{x}) = 0; \end{cases} \tag{5.9}$$

where $\phi(\cdot)$ is the Probability Density Function (PDF) of the zero mean unit variance Gaussian distribution, and

$$Z = \frac{\hat{\mu}(\mathbf{x}) - \mu^+}{\hat{\sigma}(\mathbf{x})}. \tag{5.10}$$

Finally, it may be useful to control explicitly the trade off between exploration and exploitation. To do that, similarly to what done with the probability of improvement, we can add a parameter $\xi \geq 0$ that balances the two competing objectives. Equation (5.9) can be rewritten to incorporate such parameter as

$$\mathbf{MEI}(\mathbf{x}) = \begin{cases} (\hat{\mu}(\mathbf{x}) - \mu^+ - \xi)\Phi(Z') + \hat{\sigma}(\mathbf{x})\phi(Z'), & \text{if } \hat{\sigma}(\mathbf{x}) > 0; \\ 0, & \text{if } \hat{\sigma}(\mathbf{x}) = 0; \end{cases} \tag{5.11}$$

where

$$Z' = \frac{\hat{\mu}(\mathbf{x}) - \mu^+ - \xi}{\hat{\sigma}(\mathbf{x})}. \tag{5.12}$$

5.3.3 Example of application

Figure 5.3 shows the estimation of a one-dimensional function $u(x)$ guided by the two described acquisition functions: probability of improvement and expected improvement. The function to be estimated is

$$u(x) = \frac{3}{4} \cdot e^{(x-2)^2} + e^{\frac{(x-6)^2}{4}}, \tag{5.13}$$

which has a local maximum in $x = 2$ and a global maximum in $x = 6$. The first and third rows of Figure 5.3 show the real function $u(x)$ (red dashed line) and the pointwise predicted mean (solid black like) plus and minus the standard deviation (gray filled area), using a GP. The second and fourth rows show the acquisition functions that guide the sampling process, computed respectively using (5.2) and (5.9).

Both algorithms have been initialized with a single sample ($x_1 = 2$, $y_1 = u(x) \simeq 0.7$) but quickly follow different sampling trajectories. On the one hand, the PI acquisition function ignores zones of high variance and keeps collecting samples that are close to the local peak in $x = 2$. Eventually the sampling procedure is too much exploitation-oriented and fails in identifying the real maximum. This highlights the huge vulnerability of the PI guided approach: if a local maximum is found at the beginning, the algorithm encourages to sample close to the peak and fails in identifying the real maximum. On the contrary, EI automatically balances between exploration and exploitation, and quickly manages to identify the real peak in $x = 6$.

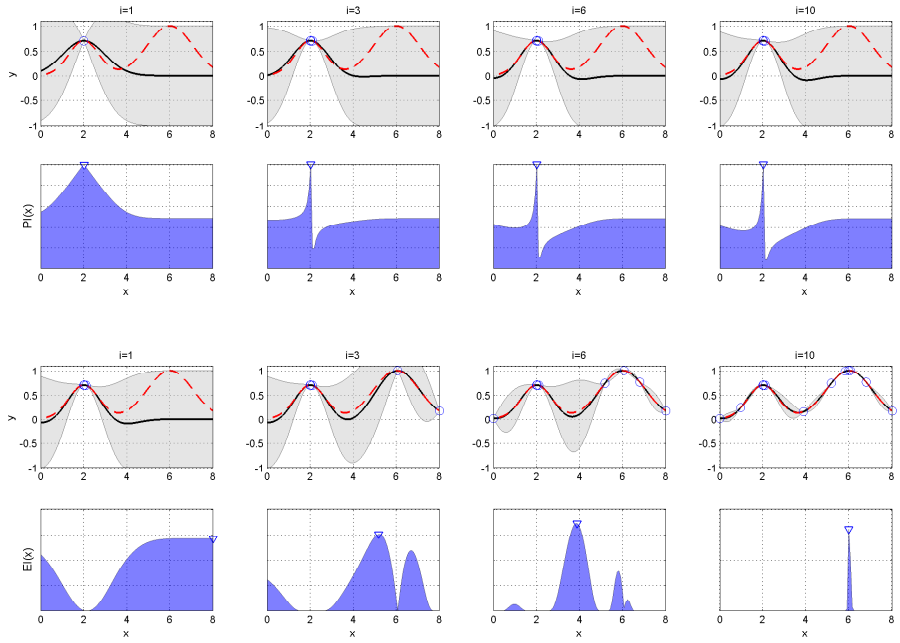


Fig. 5.3: Comparison between probability of improvement and expected improvement for a one-dimensional problem. The underlying objective function is shown with red-dashed line, while the solid black line is the predicted mean. The grey shaded area shows the 95% confidence interval.

Chapter 6

The localization algorithm

In the previous chapters, we described the mathematical tools that are needed to solve the localization problem formulated as a maximization process. In this chapter, we combine such tools into the final localization algorithm that is here described and explained. We also introduce the multi-target extension of the algorithm that allows for a quick and efficient localization of multiple devices.

6.1 Single-target algorithm

The basic localization algorithm for a single target is shown in Algorithm 2, and the main steps are described below.

6.1.1 Quick scan

In the initialization phase the UAV flies with a predetermined trajectory over the search area $\mathcal{X}$. Starting from the closest corner, it reaches the opposite corner bouncing from one edge to the other n_{init} times. Fig. 6.1 shows the trajectory followed by the drone for $n_{init} = 2$ (red plot) and $n_{init} = 3$ (blue plot), where the search area is delimited with a black dashed line. The goal of this phase is to collect an initial number of measurements in order to start the Bayesian optimization algorithm. The average duration of this phase, assuming a square search area with edge L meters long, is $\bar{t}_{init} = \big(L/\bar{v}\big)(1 + n_{init}^2)^{1/2}$, where $\bar{v}$ is the average speed (m/s) of the UAV. In our experiments, $\bar{v} \simeq 13$ m/s, $L = 1$ km and $n_{init} = 3$, so $\bar{t}_{init} \simeq 5$ minutes.

Algorithm 2 Localization algorithm for single Wi-Fi device

Require: Search region $\mathcal{X}$, stopping threshold δ, number of candidate locations n_*.

1: **repeat**
2: Do a quick scan of the search area to get initial measurements $y_1, \ldots, y_m$ from corresponding locations $\mathbf{x}_1, \ldots, \mathbf{x}_m$ as described in Section 6.1.1.
3: **until** $m = 0$
4: Set $l = m$ and initialize $\mathbf{y}_l = [y_1, \ldots, y_m]$, $\mathbf{X}_l = [\mathbf{x}_1, \ldots, \mathbf{x}_m]$.
5: **repeat**
6: Randomly choose a set $\mathbf{X}_*^{(l)}$ consisting of n_* candidate-locations.
7: Add previous packets locations $\mathbf{X}_l$ to the set $\mathbf{X}_*^{(l)}$.
8: Compute the predictive mean $\hat{\mu}(\mathbf{x})$ and variance $\hat{\sigma}^2(\mathbf{x})$ for all $\mathbf{x} \in \mathbf{X}_*^{(l)}$.
9: Compute current estimate of location: $\hat{\mathbf{x}}^{(l)} = \arg\max_{\mathbf{x} \in \mathbf{X}_*^{(l)}} \hat{\mu}(\mathbf{x})$.
10: Choose $\mathbf{x}_{l+1} = \arg\max_{\mathbf{x} \in \mathcal{X}_*^{(l)}} \mathbf{EI}(\mathbf{x} \,|\, \mathbf{y}_l, \mathbf{X}_l)$.
11: Go to $\mathbf{x}_{l+1}$.
12: If a new packet is received before T_{max} seconds, then $\mathbf{y}_{l+1} = \mathbf{y}_l \cup y_{l+1}$, otherwise $\mathbf{y}_{l+1} = \mathbf{y}_l \cup \{-85\}$. $\mathbf{X}_{l+1} = \mathbf{X}_l \cup \mathbf{x}_{l+1}$
13: Compute average interdistance $\alpha^{(l)}$.
14: Set $l \leftarrow l + 1$
15: **until** $\alpha^{(l)} < \delta$

6.1.2 Choosing the candidate-locations $\mathbf{X}_*$

The cardinality of the set $\mathbf{X}_*^{(l)}$, namely n_*, has a twofold consequence on the algorithm. On the one hand, n_* determines the computational complexity of the algorithm, so we would like to keep n_* as small as possible. On the other hand, the estimated target position will be a point of $\mathbf{X}_*^{(l)}$, so lowering n_* corresponds to increasing the granularity error. How to choose the locations in such set is therefore crucial in our problem. The Gumstix computer used in the experiments is able to perform the prediction in a reasonable time for a fixed number $n_* = 400$ of locations. If we draw at random n_* locations from a uniformly spaced grid over $\mathcal{X}$ (see Figure 6.2a), this would result in a high granularity error. The same holds if we consider a 2D-uniform distribution that covers the whole search area $\mathcal{X}$

To generate the set $\mathbf{X}_*^{(l)}$ we use instead a rejection sampling approach. If a packet has been received in a certain location, this means that the source must be located within a range of d_{max} meters. Exploiting this fact, let $\tilde{\mathbf{x}}$ denote a location drawn at random from a 2D-uniform distribution that covers the whole area $\mathcal{X}$. If the distance between $\tilde{\mathbf{x}}$ and all the locations of the other received packets is smaller than d_{max}, $\tilde{\mathbf{x}}$ is added to $\mathbf{X}_*^{(l)}$, otherwise it is rejected. This operation is repeated until $\mathbf{X}_*^{(l)}$ contains n_* points.

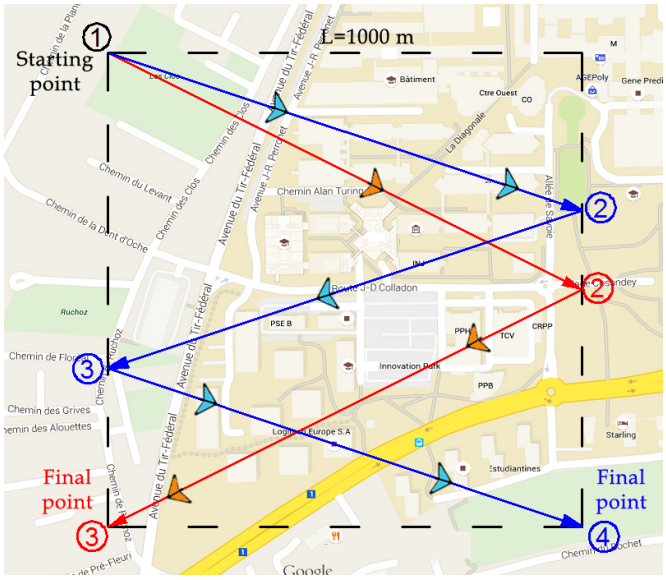
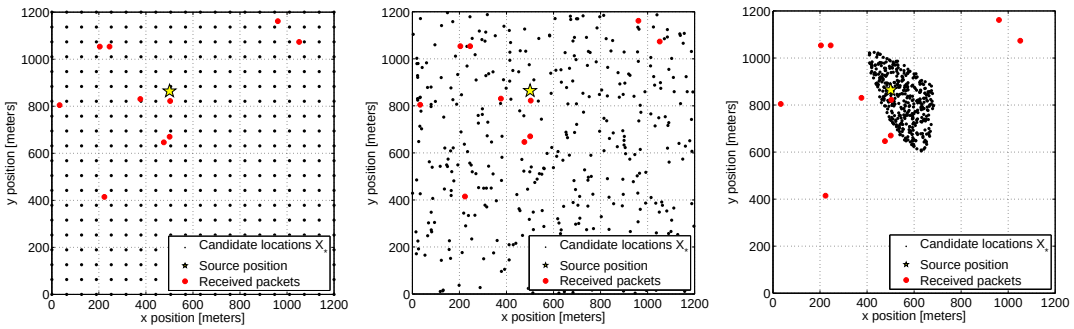


Fig. 6.1: Trajectory followed by the plane during the initial phase for $n_{init} = \{2, 3\}$.

At the very beginning of the algorithm, when no packets are available,no samples are rejected and n_* locations are chosen randomly from $\mathcal{X}$. Each time a new packet is received, the rejection area grows and the acceptance region shrinks down together with the granularity error. Even when a few observations are available, as Figure 6.2c shows, the n_* samples are very clustered around the source.



(a) Bidimensional mesh (b) Uniform random drawing (c) Rejection sampling

Fig. 6.2: Rejection sampling approach for building $\mathbf{X}_*$.

As a final observation, we note that step 7 of the algorithm requires to add to $\mathbf{X}_*^{(l)}$ the locations of the acquired packets $\mathbf{X}_l$. This means that the prediction is performed not only for the candidate locations obtained using the method described so far, but also for each location where a packet has been received. This is necessary because the

computation of the acquisition function requires $\mu^+ = \max_{\mathbf{x}\in\mathbf{X}_l}\big[\hat{\mu}(\mathbf{x})\big]$, that in turn requires $\hat{\mu}(\mathbf{x})$, $\forall \mathbf{x} \in \mathbf{X}_l$.

6.1.3 Collecting a new sample

Once the UAV reaches the location $\mathbf{x}_{l+1}$ as suggested from the acquisition function, it in supposed to collect the new sample y_{l+1}, that is then added to the dataset. There is the possibility that the UAV flies to a point where packets cannot be received. In such case, the posterior distribution and the outcome of the Bayesian Optimization would be always the same, and the UAV would be held in the same location trying to sample the function where no measurement can be performed. To avoid that, the drone waits up to T_{max} seconds to collect a new packet. If the monitor does not receive any packet in such time interval, the value $y_{l+1} = -85$ is added to the dataset. This is the minimum RSSI value, and this discourage the UAV from coming back in the future in the same position.

6.1.4 The stopping rule

The last thing to be formally defined in the algorithm is a stopping rule, i.e., a termination condition. The Wi-Fi device is considered to be correctly located if its estimated location remains stable for q successive iterations. Formally, the algorithm terminates if

$$\alpha^{(l)} = \frac{1}{q}\sum_{i=l-q}^{l} \|\hat{\mathbf{x}}^{(i)} - \hat{\mathbf{x}}^{(i-1)}\| \leq \delta, \tag{6.1}$$

with δ (meters) suitable threshold. Intuitively, a small δ corresponds to an higher accuracy in the localization. Chapter 7 shows the formal relation between the localization error and the threshold δ for the stopping rule.

6.2 Multi-target algorithm: the sequential approach

The sequential extension of the algorithm described in the previous section is quite straightforward. In particular, after having located the first target, we repeat the Bayesian driven search for each other target, while the initialization phase is performed only once at the beginning. It is important to observe that during the localization of the first target, the monitor can potentially collect packets coming from any other target. Therefore the algorithm has an advantage when the estimation of the second target starts, because the dataset is already rich. This can potentially speed-up the localization

of any successive target. The sequential version of the multi-target algorithm is shown in Figure 6.3.

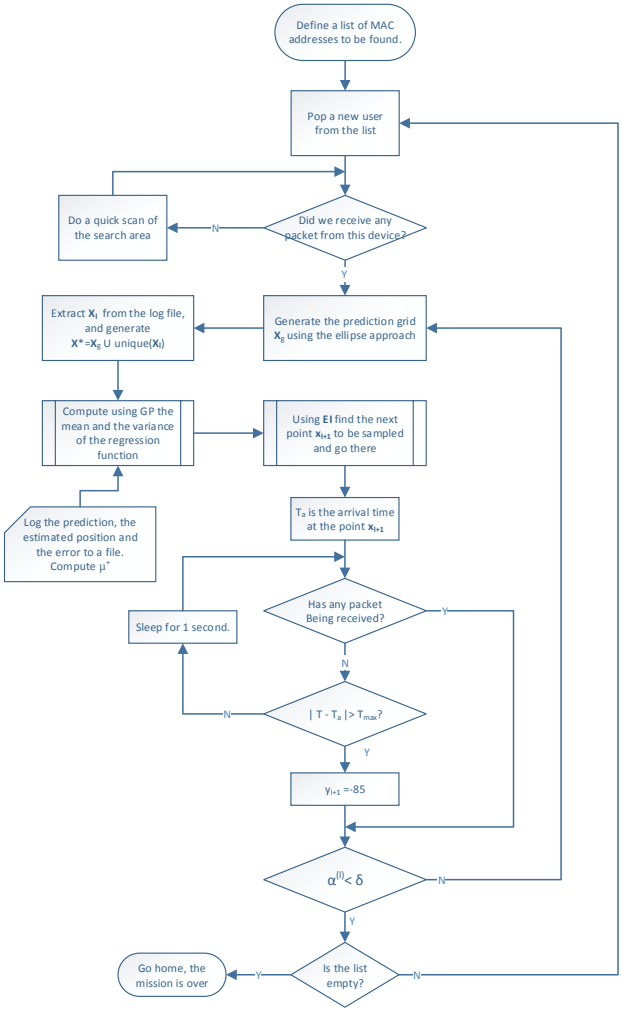


Fig. 6.3: Flowchart of the sequential multi-target localization algorithm.

Algorithm 3 Parallel localization algorithm for multiple Wi-Fi devices

Require: Search region $\mathcal{X}$, stopping threshold δ, number of locations n^*, number of devices M.

repeat
  Do a quick scan of the search area to get initial measurements $y_1, \dots, y_m$ from corresponding locations $\mathbf{x}_1, \dots, \mathbf{x}_m$ as described in Section 6.1.1.
until $m = 0$
Set $l = m$ and initialize $\mathbf{y}_l = [y_1, \dots, y_m]$, $\mathbf{X}_l = [\mathbf{x}_1, \dots, \mathbf{x}_m]$. Create the empty set of found targets $\mathcal{F} = \emptyset$.
repeat
  Randomly choose a set $\mathbf{X}_*^{(l)}$ consisting of n_* candidate-locations.
  Add previous packets locations $\mathbf{X}_l$ to the set $\mathbf{X}_*^{(l)}$.
  Compute the predictive mean $\hat{\mu}_i(\mathbf{x})$ and variance $\hat{\sigma}_i^2(\mathbf{x})$, $\forall \mathbf{x} \in \mathbf{X}_*^{(l)}$, $\forall\ i \notin \mathcal{F}$
  Compute current estimate of location: $\hat{\mathbf{x}}_i^{(l)} = \arg\max_{\mathbf{x} \in \mathbf{X}_*^{(l)}} \hat{\mu}_i(\mathbf{x})$, $\forall\ i \notin \mathcal{F}$
  Compute average interdistance $\alpha_i^{(l)}$, $\forall\ i \notin \mathcal{F}$
  If $\alpha_i^{(l)} < \delta$ then $\mathcal{F} = \mathcal{F} \cup \{i\}$
  Define $\tilde{\mu}(\mathbf{x}) = \max_{i \notin \mathcal{F}}(\hat{\mu}_i(\mathbf{x}))$ and $\tilde{\sigma}^2(\mathbf{x}) = \max_{i \notin \mathcal{F}}(\hat{\sigma}_i^2(\mathbf{x}))$, $\forall \mathbf{x} \in \mathbf{X}_*^{(l)}$.
  Go to $\mathbf{x}_{l+1} = \arg\max_{\mathbf{x} \in \mathcal{X}_*^{(l)}} \mathbf{EI}(\mathbf{x} \,|\, \mathbf{y}_l, \mathbf{X}_l)$ using $\tilde{\mu}(\mathbf{x})$ and $\tilde{\sigma}^2$ as objective function.
  Set $l \leftarrow l + 1$.
until $|\mathcal{F}| < M$

6.3 Multi-target algorithm: the parallel approach

We consider here a possible extension of the Bayesian Optimizator for multiple devices. The idea is to design a specific solution for the multi-target problem capable of performing better with respect to the simple sequential approach. Our proposed solution, here described, is the summarized in Algorithm 3.

In a nutshell, the Bayesian Optimizator flies the plane to collect measurements that can be useful for all the M devices, not just for a single one. In particular, instead of using the objective function of a single device at time, we combine such functions together to give a single aggregate objective function to be maximized.

However, Bayesian optimization is designed to identify just a single global maximum of a certain objective function, and this limitation need to be properly addressed to allow the localization of heterogeneous devices. Three 1D toy functions are shown in Figure 6.4, and we can think of them as representing the received power from 3 different devices at different locations x, where the transmitting devices use a different electric power. How to aggregate the three functions together to give a single objective

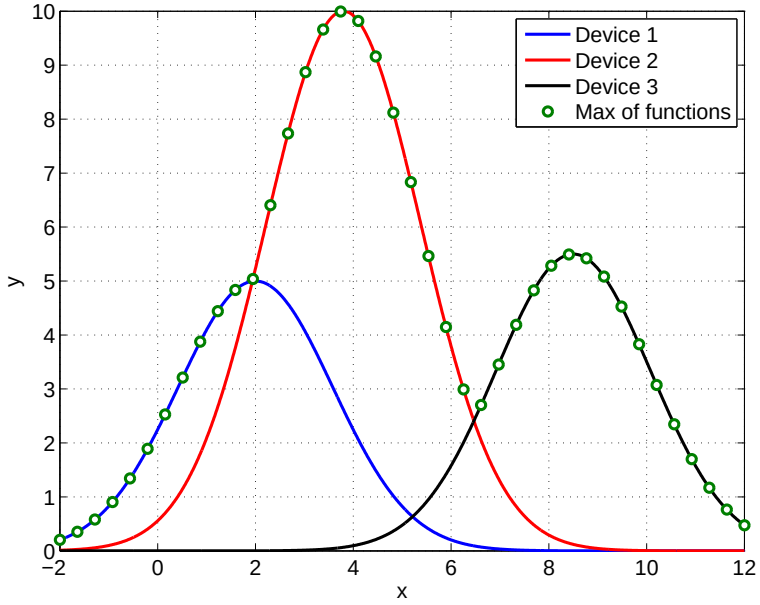


Fig. 6.4: Toy example for deriving the general objective function in the parallel localization algorithm.

function is an open choice, but we used the pointwise *max* operator in order not to loose or smooth down the peaks, that are crucial for a precise estimation. The pointwise maximum of the three functions is shown with the green dots. If we use such function to guide the sampling, the expected improvement will push the sampling around $x = 4$, where the global maximum is, thus leaving unexplored the other two peaks. Therefore, this method would give a high precision for devices with high transmission power, and poor performance for those with low transmission power. To solve this problem, once each device has been located, we remove its objective function from the max operator. In other words, the global function is the pointwise maximum of all the acquisition functions of devices not yet located. The remainder of the algorithm remains basically the same. We just point out that the posterior distribution is derived independently for each device, and then the posteriors are combined together to give the single objective function.

Chapter 7

Results

7.1 Introductive results

In this section we present some of the results obtained during the preliminary analysis, that was carried out before implementing the algorithm. Such results give some insights we used to design and develop the algorithm, and justify some of the design choices we made.

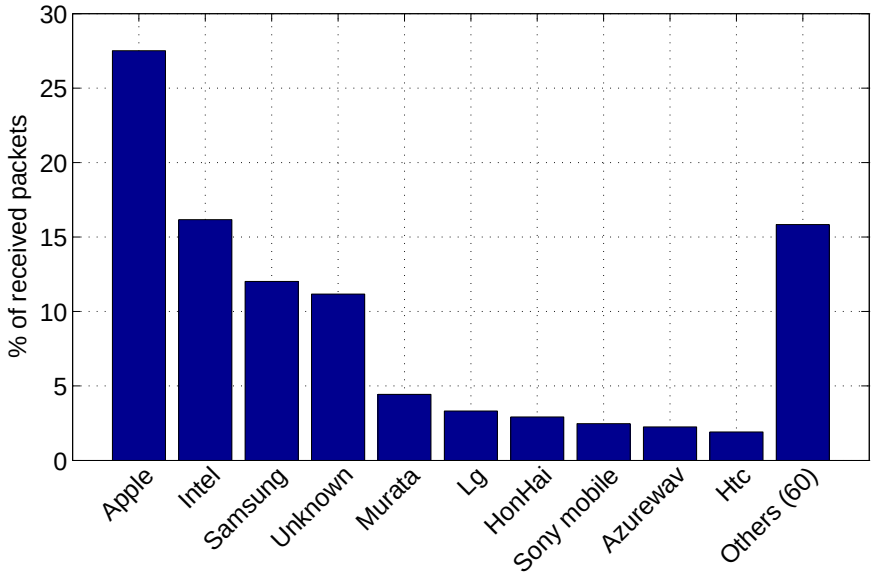


Fig. 7.1: Histogram of the vendors that manufactured the Wi-Fi cards of the observed unique devices.

During one experiment we flew the UAV for around 40 minutes in a confined one-

square kilometer area, recording all the received PRFs. During such period of time, we received around 16000 packets ($\simeq$ 7 packets/second) coming from 1800 individual MAC addresses, i.e., 1800 single devices. The first 3 bytes of the MAC address are used to identify the manufacturer of the card, so we can associate each one of the 1800 device to the corresponding vendor. Figure 7.1 shows the resulting histogram of the vendors. Excluding the unlabeled traffic, we see that around 65% of the devices' Wi-Fi cards are manufactured by Apple, Intel or Samsung. All those companies are very active and involved in developing mobile devices, so the traffic we observed is likely to come from a laptop or a smartphone. The remaining part of the traffic is divided among other smaller manufacturers, that get less than 5% each.

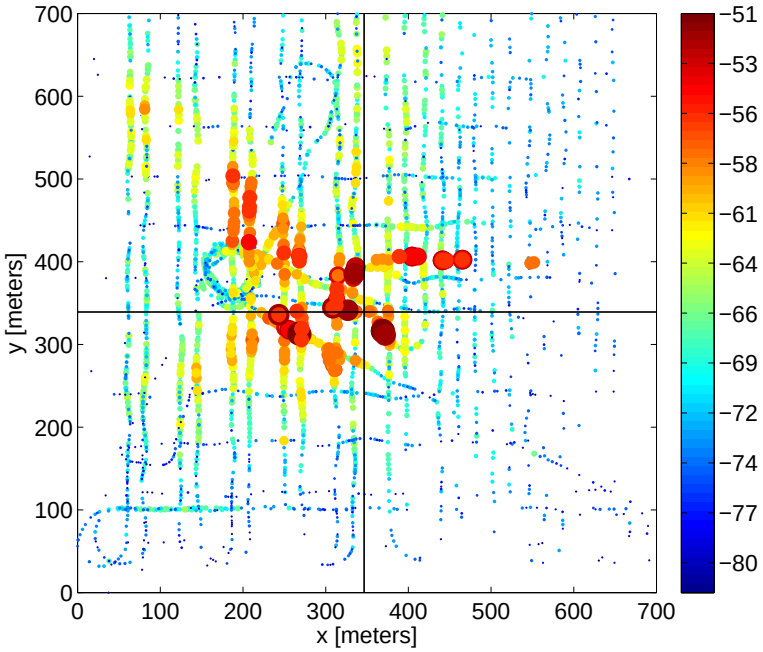


Fig. 7.2: RSSI measured during an exhaustive scan of a bidimensional region. The source is a ultrabook Sony Vaio Pro 13 that lies on the black cross, and is programmed to transmit PRF at the highest possible frequency.

Figure 7.2 shows the RSSI measured during an exhaustive scan of a bidimensional search region. The source, an ultrabook Sony Vaio Pro 13, lies on the black cross and is programmed to transmit PRFs at the highest possible rate. The color and the size of the bubbles on Figure 7.2 depends on the measured RSSI: the highest the RSSI, the largest and hottest the bubble. We observe in Figure 7.2 that the distribution of the bubbles is asymmetric with respect to the source, and that not all the biggest bubbles lie over the cross. This shows the relevance of the noise in our problem: big bubbles

can be observed even at relatively high distances from the source. We can also observe that in the bottom-right corner just a few measurements are available, while in the opposite upper-left corner more packets have been received. Once more, this shows how strongly the environment impacts the measurements. In fact, not all the points, even if they are at the same distance, have the same probability of successful packet reception. The same observations are represented in Figure 7.3, where the measured RSSI is shown as a function of the distance (blue crosses). In the same figure, we also plot the RSSI measured from a Samsung Galaxy Nexus smartphone (red dots).

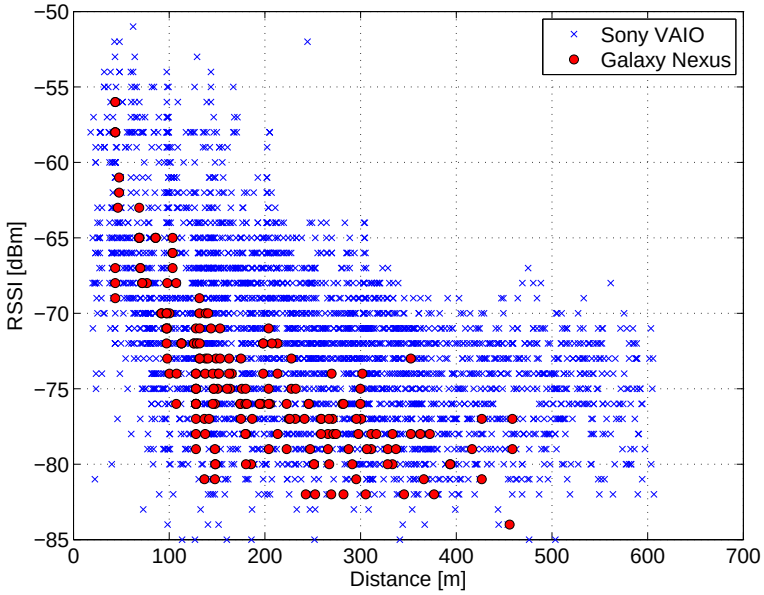


Fig. 7.3: RSSI as a function of the distance for two devices: a laptop (blue) and a smartphone (red).

It is interesting to observe not only the huge amount of noise that affects the measurements, but also that the smartphone behaves differently from the laptop. In fact, on average, the RSSI measurements are generally lower for the smartphone, probably because the energy-constrained phone allocates less energy to the Wi-Fi card. This fact reinforces the need for a flexible approach capable of adapting to different devices with different behavior. Such flexibility, in our approach, is achieved using Gaussian Processes that can easily adapt to different conditions.

7.2 A real experiment

In this section we present the results of a real experiment we conducted to test the single-target algorithm. The device to be located is the smartphone Samsung Galaxy Nexus, where no particular applications have been installed. We just prevent it from going to stand by, thus simulating a normal use of the phone. The search area has an edge of $L = 1000$ meters, and the initial mission has been configured so that the UAV goes through 2 points before reaching the opposite corner of the search area. Figure 7.4 shows the trajectory followed by the UAV during the localization process.

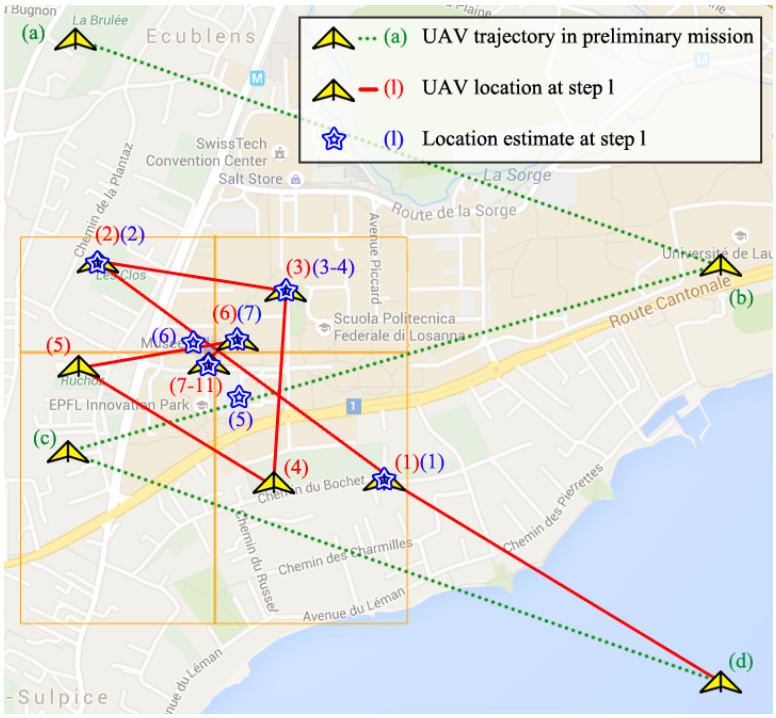


Fig. 7.4: Trajectory followed by the UAV during the single-target localization process.

The green dotted lines show the path followed by the UAV during the preliminary mission that ends after 5 minutes at the point denoted with (d), with $l = 7$ packets. Using such observations, the posterior distribution is derived, as shown in the first row of Figures 7.5 and 7.6. Using the prediction, the UAV estimates the location of the phone, denoted with the blue star (1), and guided by the Expected Improvement function it decides where to move next (red (1)). The UAV then follows the path drawn using the red line. At the first step of the algorithm, the point where the UAV moves (1) corresponds with the estimated location of the phone (1), i.e., with the current global

maximum of the mean posterior (exploitative approach). The same holds for the second and third steps. However, we observe that at the fourth step the point where the UAV samples (4) and the estimated location of the phone (4) are different, thus meaning that the algorithm encourages the exploration. The same holds for the the fifth step. After the sixth step, the maximum of the mean function, and the maximum of the EI function coincide again, thus meaning that the algorithm is back to the exploitative approach. Figures 7.5 and 7.6 show the evolution of the posterior distribution for some steps of the algorithm. For both figures, the first column shows the posterior mean, the second column shows the posterior standard deviation while the third shows the acquisition function, i.e., the EI function based on the current posterior distribution. It is interesting to observe how the peak of the mean function emerges step-after-step. At the first step, the posterior distribution is flat and blurry, no clear maximum can be observed and the uncertainty is high almost everywhere. When we move down row by row, i.e., we follow the evolution of the algorithm step-by-step, we observe that the peak starts to emerge in the mean function, and that the standard deviation at the peak location is considerably diminished. Even at the last iteration the standard deviation at certain locations remains high, but the mean value in such locations is low enough to ensure that the real peak is not hidden there. It is also interesting to observe how the EI function evolves. At the beginning, we can observe a wide region with high values of EI. All those points, since the uncertainty is high, can potentially yield a huge improvement of the mean function. As we get more samples, the hot region of the EI tends to shrink in size, and eventually becomes a single narrow peak around the source location, as shown in the last row of Figure 7.6.

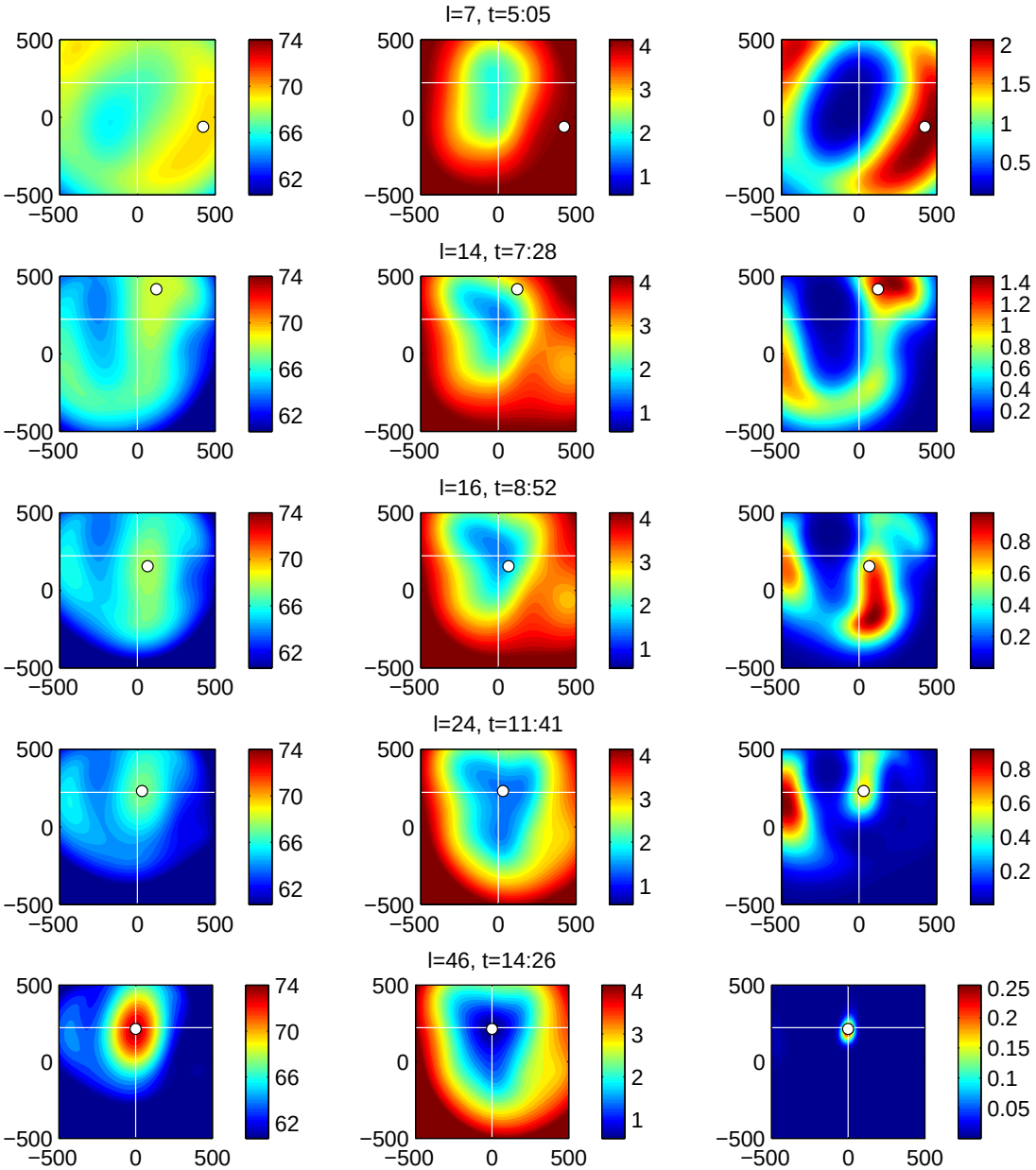


Fig. 7.5: Posterior distribution at successive steps of the algorithm. The first column shows the posterior mean, the central column shows the variance, the third column shows the EI function. The cross marks the real location of the phone, while the white dots is the estimated position at step l.

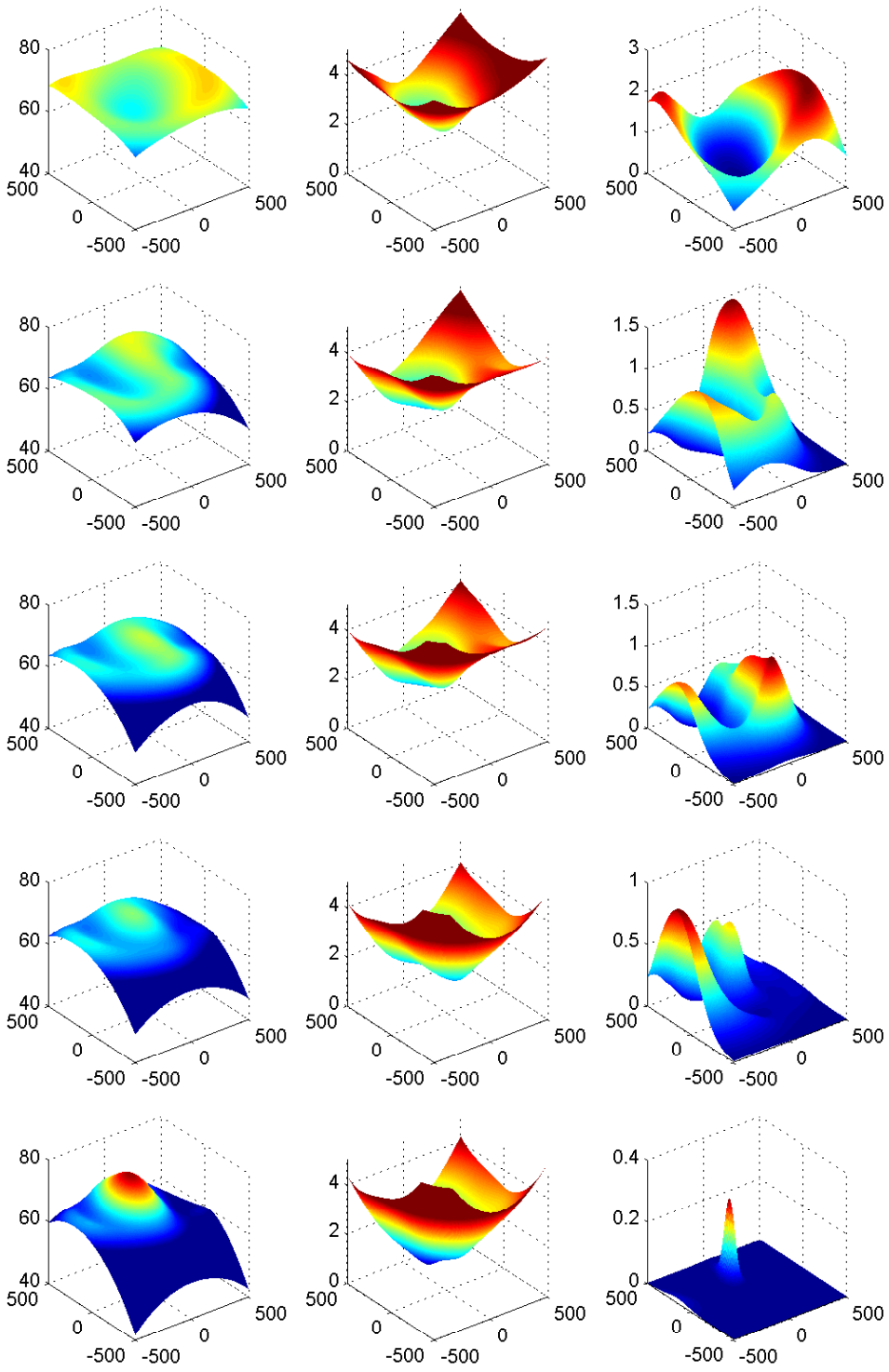


Fig. 7.6: Posterior mean (first column), posterior std. dev (central column) and EI function (third column) at successive steps of the algorithm.

The stopping rule

In Chapter 6 we described the stopping rule of the localization algorithm. We consider the source to be correctly located if

$$\alpha^{(l)} = \frac{1}{q} \sum_{i=l-q}^{l} \|\hat{\mathbf{x}}^{(i)} - \hat{\mathbf{x}}^{(i-1)}\| \leq \delta, \tag{7.1}$$

with δ suitable threshold. In our experiment, we considered $\delta = 5$ meters and $q = 5$. Figure 7.7 shows the evolution of $\alpha^{(l)}$ and of prediction error $\epsilon^{(l)}$ for successive iterations of the algorithm. We emphasize that the algorithm can only make use of $\alpha^{(l)}$ to decide whether to stop or to continue sampling. Figure 7.7 shows a close correspondence between the evolution of the error $\epsilon^{(l)}$ and $\alpha^{(l)}$: the two curves overlap almost perfectly. This confirms that by observing the evolution of $\alpha^{(l)}$ we can understand if our estimation is precise or not. In the Section 7.4 we will present other results to validate the use of (7.1) as a stopping rule.

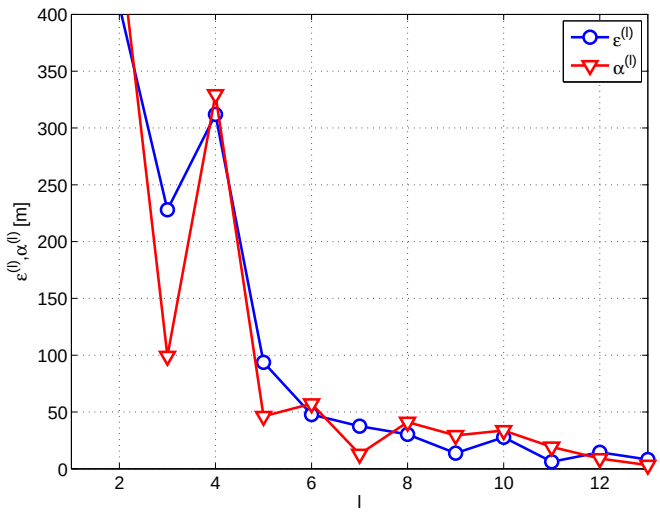


Fig. 7.7: Evolution of the prediction error $\epsilon^{(l)}$ and of the stability of the prediction $\alpha^{(l)}$ at successive iterations of the algorithm.

7.3 A Graphic User Interface to monitor the progress of the mission

Since the UAV is completely autonomous during the localization process, there is in principle no need to develop any particular software to control it from the ground. We used the proprietary eMotion application to manage the landing and to give the control of the UAV to the Gumstix computer. However, to follow the progress of the mission, we developed a Graphic User Interface (GUI) that shows the received packets and the estimated locations of the device. Using the Google Maps API [37] and the Javascript programming language, we developed a dynamic webpage that periodically fetches the data from the UAV and displays it over the map.

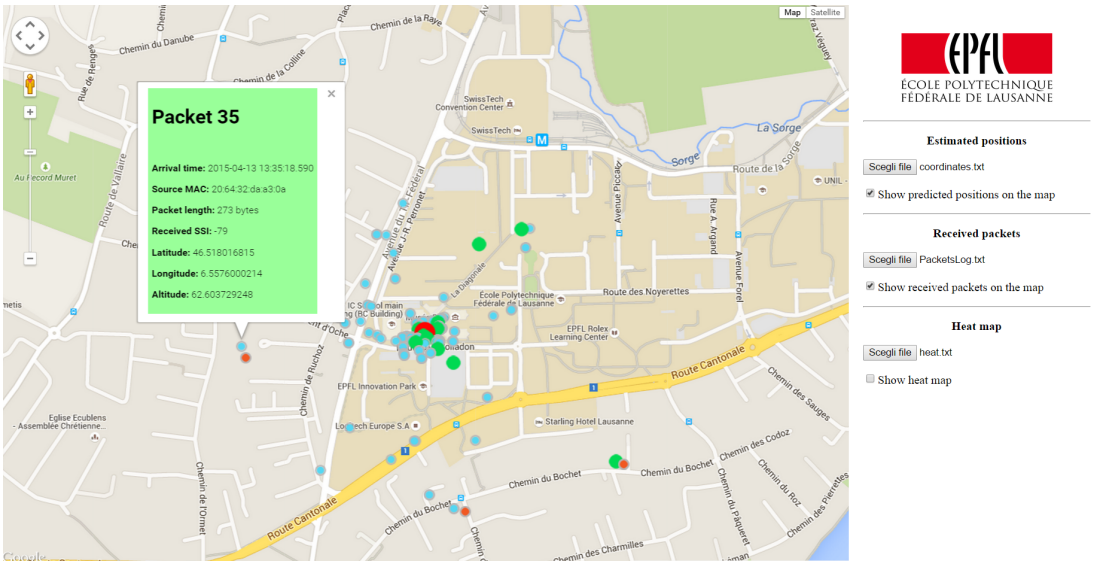


Fig. 7.8: GUI to monitor the progress of the localization algorithm.

The blue and orange circles show the locations of received packets, and clicking over them we can see when the packet has been received and the measured RSSI. The estimated location of the source at each step of the algorithm is shown with a green circle, while the last estimation, i.e., the one that should be more accurate, is shown with a red big circle. By clicking over a green or red circle we can see the corresponding step of the algorithm. The page is automatically refreshed, so that when new measurements are available they are immediately displayed over the map.

7.4 The MATLAB simulator

Field experiments are expensive and require the involvement of people, transportation, and costly equipment. Furthermore, providing some reliability measures can require hundreds of experiments. This called for the development of a suitable environment to emulate the behavior of the UAV during a localization mission. Such MATLAB emulator is illustrated in this section, and is designed to emulate as closely as possible a real flight. The design choices of the simulator are here briefly summarized:

- The search area is a square with L meters side. Each target is randomly positioned in the search area with uniform probability.

- The targets broadcast PRFs independently one another. The i-th target broadcasts the j-th PRF at time $t_{i,j} = t_{i,0} + j \cdot \Delta T + \varepsilon_{i,j}$, where ΔT is the PRFs inter-transmission time; $t_{0,i}$ is an initial random time offset uniformly distributed between 0 and ΔT (i.e., $t_{i,0} \sim \mathrm{U}(0, \Delta T)$ i.i.d. $\forall i$) and $\varepsilon_{i,j}$ is a random time offset representing the jitter in the PRF transmission. The random variables $\varepsilon_{i,j}$ are i.i.d., with uniform distribution between $-\varepsilon_{max}$ and ε_{max}, i.e., $\varepsilon_{i,j} \sim \mathrm{U}(-\varepsilon_{max}, \varepsilon_{max})$.

- The plane flies at constant speed (v m/s) and altitude (h meters).

To obtain realistic results it is crucial to implement a trusty RSSI measurement mechanism. To this end, we built a synthetic model of the channel using the dataset of real observations shown in Figure 7.3. We consider N bins that cover the range $[0, R = 600]$ meters, each one of size r meters, so that $N = R/r$. We denote with $\rho_i = \big[i \cdot r, (i+1) \cdot r\big)$, $i = 0, 1, \ldots, N-1$ the interval of distances of length r meters. To build the model of the channel, we first derive one empirical CDF (ECDF) for each of the N bins, considering only the measurements that have corresponding distance $d \in \rho_i$. We denote such ECDF as $r_i(u)$, with $i = 0, 1, \ldots, N-1$ and $u \in \mathbb{Z}$ RSSI. Figure 7.9 shows an example of such ECDF for four different bins, with $R = 600$, $r = 20$. Then, an RSSI sample at distance d_* is generated from $r_j(u)$, with $j : d_* \in \rho_j$.

Testing the stopping rule

The first simulation is conducted to evaluate the impact of the stopping-rule's threshold δ on the performance of the localization algorithm. The main parameters of the simulation are summarized in Figure 7.10.

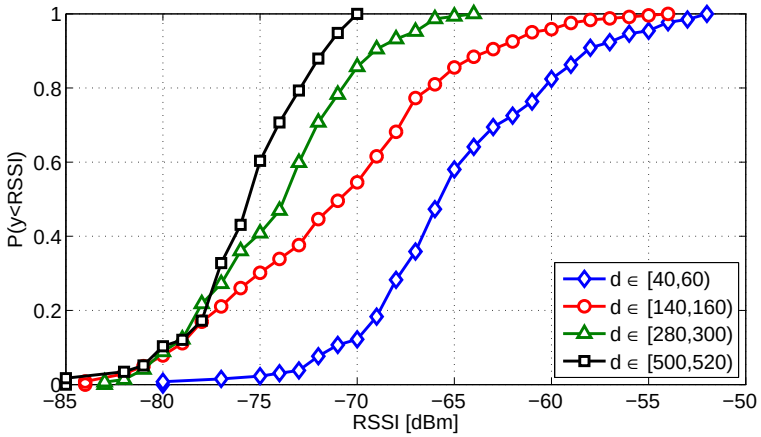


Fig. 7.9: ECDF of the RSSI for four different bins ρ.

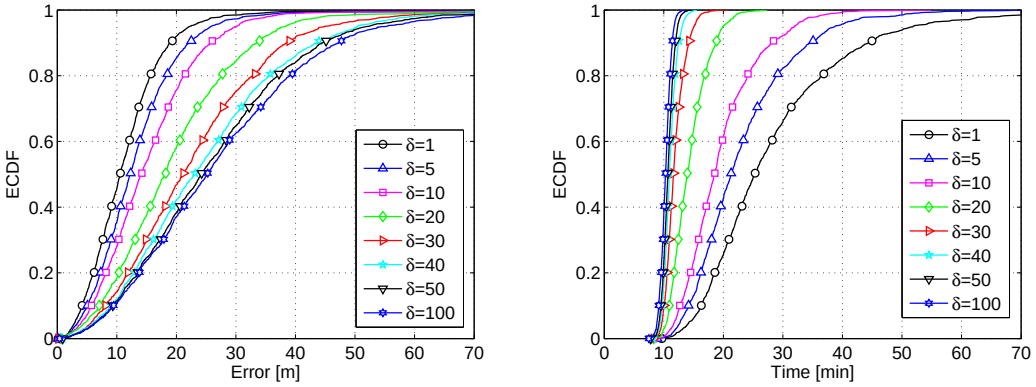
A single target is randomly positioned inside the search area and the UAV runs the single-target algorithm to locate it. When the stopping condition is satisfied, (i.e., the algorithm ends) we record the duration of the mission and the error made by the algorithm when estimating the target's position. Figure 7.11a shows the ECDF of the error obtained from several independent repetitions of the experiment for different values of the threshold δ. The ECDF of the localization time is shown in Figure 7.11b for the same set of values of δ. We observe that, when δ is smaller, i.e., the stopping rule is tighter, the localization error decreases. However, the time required to locate the target increases. Therefore δ should be chosen according to the specific application as a suitable trade-off between accuracy and quickness. We identify $\delta = 10$ as a reasonable value for the threshold, as it gives an error that is smaller than 30 meters in 99% of the cases while keeping the searching time always under 40 minutes, thus allowing the localization of the target with a single flight.

Parameter	Value
ΔT	20 s
ε_{max}	0.5 s
R_{max}	450 m
v	13 m/s
h	40 m
n_{init}	3
n_*	350
q	5

Fig. 7.10: Simulation's parameters.

Sequential approach for multiple devices

In the second scenario, $N = 15$ targets are randomly positioned in the search area and the sequential approach is used to locate them. We denote with τ_i the mean time spent for locating the i-th target. Formally, assuming that the targets are numbered



(a) ECDF of the localization error for different values of δ

(b) ECDF of the localization time for different values of δ

Fig. 7.11: Localization error and time for different values of δ

according with the order in which they are located, if t_i is the time at which target i is located, then

$$\tau_i = \begin{cases} \mathbb{E}[t_i], & i = 1 \\ \mathbb{E}[t_i - t_{i-1}], & i > 1. \end{cases} \tag{7.2}$$

As discussed in Chapter 6, we expect τ_i to decrease as i increases, because the algorithm can benefit from the PFRs detected during the localization of the previous targets. The behavior of τ_i as a function of the target's index is shown by the blue curve of Figure 7.12, where the expectation has been replaced by the mean of averaging over several different independent repetitions. In agreement with what we expected, τ_i decreases from the initial value $\tau_1 \simeq 20$ minutes and settles to $\tau_{lim} \simeq 6$ minutes, thus confirming that collecting packets of other targets is useful to reduce the localization time. From Figure 7.12, we also observe that τ_i does not tend to zero. In fact, the time required to localize i targets using the sequential approach with $i \gg 1$ can be approximated as $t_i \simeq i \cdot \tau_{lim}$. This gives us an insight: even if several packets are available when the algorithm starts, to perform an accurate localization of the target the plane must fly close to the source for a certain time in order to collect packets with high RSSI.

We also considered a clustered scenarios, where the targets are clustered and lie in a circular area of radius R_c. Figure 7.12 shows τ_i for $R_c \in \{50, 200, 400\}$. In this case, we can observe that τ_i decreases drastically as the radius of the cluster, R_c, decreases. This is because while the UAV locates the device i, it also collects packets that are close to the other devices, therefore not affected by lot of noise and extremely useful for the estimation.

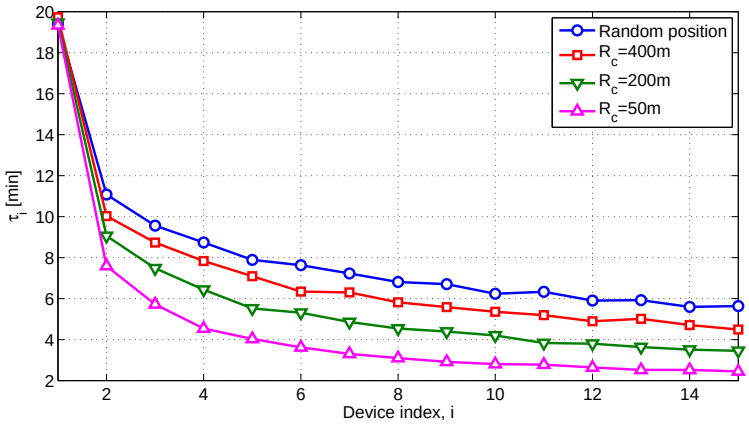


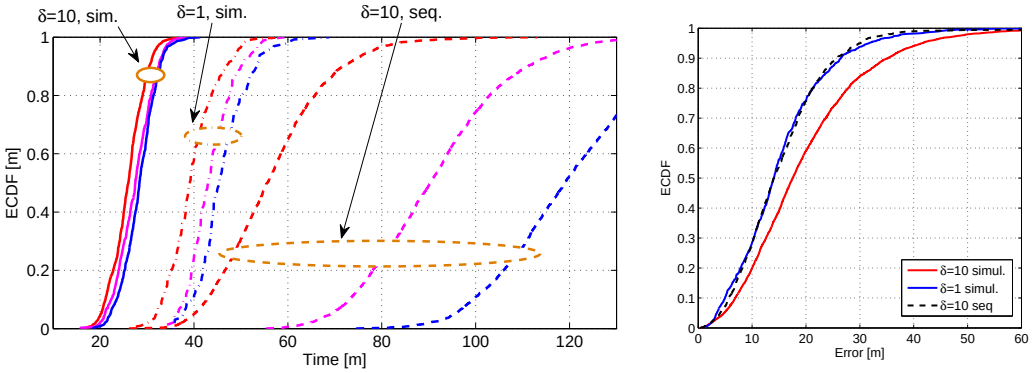
Fig. 7.12: Average time dedicated to the localization of a user for randomly positioned target or clustered targets, as a function of the target's index.

Parallel approach for multiple devices

We show here the results of the parallel approach for the multi-target localization problem. We asked the algorithm to locate $N = \{5, 10, 15\}$ targets randomly positioned in the search area, keeping the same stopping rule as before (i.e., $\delta = 10$, $n_\delta = 5$).

Figure 7.13a shows that the time required to locate the targets is drastically reduced with respect to the sequential localization. Interestingly, the time required to locate 15 targets is just slightly greater than the time needed to locate 5 targets. Simulations results show also that the parallel-approach, with the same stopping rule, is more likely to make a bigger estimation error. This effect is clear from Figure 7.13b, that shows the ECDF of the localization error obtained using the parallel approach (solid red line) against the sequential approach (dashed black line). We see that, using the parallel approach, we obtain an error that is on average 10 meters greater. This is the price we have to pay to use a global objective function that drives the Bayesian optimization.

In order to perform a fair comparison with the sequential algorithm, we reduced the value of δ to 1 meter. In this manner, the parallel approach yields an error with the same distribution of the sequential algorithm (see blue curve in Figure 7.13b). The corresponding time slightly increases, as Figure 7.13a shows, but is still smaller than for the sequential approach, thus confirming the benefit given by the parallel algorithm instead of the sequential one. As a final remark we highlight that, from a computational point of view, the parallel approach is very demanding, since the prediction must be performed independently for each target at each iteration of the algorithm.



(a) ECDF of the time required to locate $N = 5, 10, 15$ targets using the sequential approach for $\delta = 10$ meters and the parallel approach for $\delta = 1, 10$ meters.

(b) Localization error using the parallel approach with $\delta = 1, 10$ and the sequential approach for $\delta = 10$.

Fig. 7.13: Localization time and error using the parallel approach.

Chapter 8

Conclusions and Future Work

In this work, we designed and tested a localization algorithm for UAVs that makes use of the RSSI to estimate the position of a Wi-Fi device. Since we assumed the source to be uncooperative, we can think of using the algorithm for several different applications. In Chapter 1 we mentioned some of them, but the flexibility of the approach open the way to new unexpected applications. Some suggestions for the future works can also be found in the next section.

After having carried out a first rough feasibility analysis, we formulated the problem as a maximization problem. We decided to use Bayesian Optimization to estimate the maximum of a function that is costly to evaluate, and such approach proved very useful to solve the problem quickly and efficiently. In fact, using the Expected Improvement acquisition function, we achieved an automatic trade off between exploration and exploitation, showing that the algorithm can collect useful RSSI measurements and achieve a good precision even with few observations. We built this model on top of a solid understructure: Gaussian Processes, which are powerful, flexible and well-known. We designed the localization algorithm and implemented it on the UAV, performing some tests in a real environment. We showed how the algorithm behaves in normal conditions and proved that, even with uncooperative devices, we can achieve a very good precision when localizing the target. To provide some reliability measures, we designed a MATLAB simulator that has been also very useful to show the benefits of a parallel approach over the sequential one. Implementing such parallel algorithm inside the UAV is the next step of our course of actions.

8.1 Future work

8.1.1 Uncooperative source

In this work, we assumed the source to be uncooperative, i.e., to have no software onboard and to have no particular connection with the UAV. Even under such assumption, we showed how the RSSI can be used to perform the localization with a reasonable precision. Keeping the same approach, we could potentially achieve an improvement of the performance by

- **Multiple antennas**. Increasing the number of antennas at the receiver side could help increasing the number of observations, thus mitigating the effects of the noise.

- **Directive antennas**. Instead of using simple omnidirectional antennas, it is possible to use patch antennas or directive antennas that could give additional information on the direction the signal comes from.

- **Multiple UAVs**. It may be interesting to investigate the results that could be obtained using multiple cooperative UAVs at the same time. The UAVs could use some cooperating technique to share the data and equally split the workload to localize the sources in a quicker way. This approach can be useful especially for vast search areas.

- **Different regression solver**. In our approach we used Gaussian Processes because they are flexible and well-known. More refined techniques, such as neural networks could potentially give benefits to the estimation process.

A different way to tackle the uncooperative problem would be to install a Software Defined Radio (SDR) platform onboard the UAV, such as the one described in [38], that gives us access to the signal at the physical layer. Since the platform is especially designed to work with high frequencies, we can estimate very precisely the arrival time of a packet. Then, using two or more UAVs equipped with such SDR platform, we could estimate the location of the source by evaluating the difference on the arrival time of the same packet. The precision of this approach depends of course on the chosen SDR platform. Using the datasheet provided in [38], the time resolution τ of the board can be computed as the inverse of the maximum clock,

$$\tau = \frac{1}{\nu} \simeq \frac{1}{4} 10^{-9} \simeq 250 \text{ pS}. \tag{8.1}$$

In the ideal scenario, assuming free space Line-of-Sight propagation, the electromagnetic signal in τ seconds covers a distance of

$$d \simeq \tau \cdot (3 \cdot 10^8) \simeq 10 \text{ cm}, \tag{8.2}$$

thus showing that the accuracy that could potentially be achieved is very high. However, real experiments are needed to evaluate the impact of reflections and multipath on the estimation.

8.1.2 Cooperative source

When we assume the target to be cooperative with the localizing agent, the approach and the focus change. The RSSI-based estimation can still be performed, but we can potentially have access to information that the target is now willing to share. If we first perform a rough estimation of the target location, than the UAV can move closer to the Wi-Fi device to ensure a better connection quality. When the UAV is close to the device, it can announce its presence to the target device, and this can open different scenarios:

- If the device does not have the proper application, the drone could provide it. In fact, the smartphone will see the drone as a Wi-Fi hotspot, so it will be able to connect and, if needed, download the application. Then, after the application has been installed, a secure and reliable connection between the UAV and the smartphone can be established.
- The application, which can have access among other things to the GPS sensor, can transmit to the UAV all the available information at the phone side. The UAV can make use of such information to improve the accuracy of the localization.
- The UAV could also transmit to the smartphone messages from the rescue teams. It could also act as a relay to forward messages and requests to the ground station where rescue teams coordinate the operations. Interestingly, in [13] authors have shown that a VoIP call can take place between two nodes on the ground using the UAVs as relays. However, if there is no single or multihop communication with the ground station, the UAV can temporary store messages and other information coming from the device (videos, pictures, recordings) and transmit them to the ground node when the connection becomes available again.

Finally, ultra-wideband (UWB) signals provide accurate positioning capabilities, at a relatively low cost. In principle, the UWB platform estimates the distance between

the modules measuring the average Round Trip Time of consecutive impulses. If we assume the target to be equipped with a UWB module, than we can estimate its location with high precision. Nonetheless, usually UWB modules are designed for indoor environments and for short range. Therefore, to employ such technology to our problem, we still need a preliminary RSSI-based localization, and once we are close enough to the source we can refine the estimation using the dedicated module.

Chapter 9

Appendix

9.1 Distance computation

This subsection briefly describes how the distance between two points can be computed given their coordinates. We define as $(\lambda_1, \varphi_1), (\lambda_2, \varphi_2)$ the longitudes and the latitudes of the two points respectively, while R is the earth's radius. We assume here that the latitude and the longitude are expressed in radiants. Since we deal mostly with relatively small distances, we can ignore the ellipsoidal effects and assume earth to be spherical. We use the haversine formula to calculate the great-circle distance between two points, also because it remains particularly well-conditioned for numerical computation even at small distances, so that

$$a = \sin^2(\Delta\varphi/2) + \cos(\varphi_1)\cos(\varphi_2)\sin^2(\Delta\lambda/2), \tag{9.1a}$$

$$d = 2R \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{1-a}), \tag{9.1b}$$

where $\Delta\varphi = \varphi_2 - \varphi_1$, $\Delta\lambda = \lambda_2 - \lambda_1$, $R = 6371009$ meters and $\operatorname{atan2}(\cdot,\cdot) \in [-\pi, \pi]$ is the four quadrant inverse tangent. The inverse procedure can be particularly useful if we want to specify the center of the research area and its edge expressed in meters. It is therefore necessary to convert a distance expressed in meters into the corresponding range in geographical coordinated. We consider in general a point (λ_1, φ_1) and we want to compute the coordinates of the point (λ_2, φ_2) such that the distance along the longitude axis is h meters and the distance along the latitude axis is v meters. This translates directly into the problem of determining $\Delta\varphi = \varphi_2 - \varphi_1$ and $\Delta\lambda = \lambda_2 - \lambda_1$.

$$\alpha_h = \tan^2\left(\frac{h}{2R}\right), \quad \alpha_v = \tan^2\left(\frac{v}{2R}\right); \tag{9.2}$$

$$a_h = \frac{\alpha_h}{1+\alpha_h}, \quad a_v = \frac{\alpha_v}{1+\alpha_v}; \tag{9.3}$$

$$\Delta\varphi = 2 \cdot \text{asin}\left(\sqrt{\frac{a_h}{cos^2(\lambda_1)}}\right), \quad \Delta\lambda = 2 \cdot \text{asin}\big(\sqrt{a_v}\big). \tag{9.4}$$

9.2 Forward and Backward substitution

We describe here an efficient technique for solving a system of linear equations when the the coefficient matrix $\mathbf{A}$ is decomposed using the Cholesky method. In particular, suppose that $\mathbf{A} \in \mathbb{R}^{n\times n}$ is the matrix of the coefficients, $\mathbf{B} \in \mathbb{R}^{n\times h}$ is the matrix of the constant known values, while $\mathbf{X} \in \mathbb{R}^{n\times h}$ is the matrix of the unknown values to be determined. The three matrices satisfy the following relation

$$\mathbf{AX} = \mathbf{B}. \tag{9.5}$$

We consider the Cholesky decomposition of the matrix $\mathbf{A}$ into $\mathbf{A} = \mathbf{LL}^T$, where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is the lower-triangular Cholesky factor. The original (9.5) can be rewritten as

$$\mathbf{AX} = (\mathbf{LL}^T)\mathbf{X} = \mathbf{L}(\mathbf{L}^T\mathbf{X}) = \mathbf{B}. \tag{9.6}$$

Now, since the matrices $\mathbf{L}$ and $\mathbf{B}$ are known, we can solve the system and obtain $\mathbf{Y} = \mathbf{L}^T\mathbf{X}$. Then, since $\mathbf{Y}$ and $\mathbf{L}^T$ are known, we can finally obtain $\mathbf{X}$. The original problem therefore decomposed into two easier steps:

1. Solving for $(\mathbf{L}^T\mathbf{X})$ from $\mathbf{L}(\mathbf{L}^T\mathbf{X}) = \mathbf{B}$.
2. Solving for $\mathbf{X}$ from $\mathbf{Y} = \mathbf{L}^T\mathbf{X}$.

The advantage is that, given the triangular structure of the matrix $\mathbf{L}$, the computation can be performed very efficiently. In fact, consider first the *forward substitution* to solve

$$\mathbf{LY} = \mathbf{B}, \tag{9.7}$$

that can be expanded as

$$\begin{bmatrix} l_{11} & 0 & ... & 0 \\ l_{21} & l_{22} & ... & 0 \\ \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & ... & l_{nn} \end{bmatrix} \begin{bmatrix} y_{11} & y_{12} & ... & y_{1h} \\ y_{21} & y_{22} & ... & y_{2h} \\ \vdots & \vdots & \vdots & \vdots \\ y_{n1} & y_{n2} & .... & y_{nh} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & ... & b_{1h} \\ b_{21} & b_{22} & ... & b_{2h} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n1} & b_{n2} & .... & b_{nh} \end{bmatrix}. \tag{9.8}$$

We can solve one column of the matrix $\mathbf{Y}$ at time, thus obtaining the following formulation that is valid for each columns:

$$\begin{cases} b_1 = l_{11} \cdot y_1 \\ b_2 = l_{21} \cdot y_1 + l_{22} \cdot y_2 \\ ... \\ b_n = l_{n1} \cdot y_1 + l_{n2} \cdot y_2 + ... + l_{nn} \cdot y_n \end{cases} \tag{9.9}$$

The formulation shows that, from the first equation, we can immediately obtain y_1, then plug the value into the second obtaining y_2 and so on. In conclusion,

$$y_i = \frac{1}{l_{ii}} \left[b_i - \sum_{j=1}^{i-1} l_{i,j} y_j \right]. \tag{9.10}$$

Once we have solved the first step, we can apply a similar approach to solve for $\mathbf{X}$ from $\mathbf{Y} = \mathbf{UX}$, by backward substitution. The resolvent equation is

$$x_i = \frac{1}{l_{ii}} \left[y_i - \sum_{k=i+1}^{n} l_{k,i} x_k \right]. \tag{9.11}$$

9.3 Marginalization of Gaussian vectors

Let $\mathbf{x}$ and $\mathbf{y}$ be two jointly Gaussian random vectors

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{bmatrix}, \begin{bmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^T & \mathbf{B} \end{bmatrix} \right). \tag{9.12}$$

Then the conditional distribution of $\mathbf{x}$ given $\mathbf{y}$ are

$$p(\mathbf{x}|\mathbf{y}) \sim \mathcal{N}\Big(\boldsymbol{\mu}_x + \mathbf{C}\mathbf{B}^{-1}(y - \boldsymbol{\mu}_y), \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^T \Big) \tag{9.13}$$

Bibliography

[1] E. Perahia and R. Stacey, *Next Generation Wireless LANs: 802.11n and 802.11ac*. New York, NY, USA: Cambridge University Press, 2nd ed., 2013.

[2] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[3] E. Brochu, V. M. Cora, and N. de Freitas, "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," *CoRR*, vol. abs/1012.2599, 2010.

[4] "Amazon "Prime Air" Webpage." `http://www.amazon.com/b?node=8037720011`. Accessed: 2015-03-27.

[5] A. Symington, S. Waharte, S. Julier, and N. Trigoni, "Probabilistic target detection by camera-equipped UAVs," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 4076–4081, May 2010.

[6] Z. Sun, P. Wang, M. C. Vuran, M. Al-Rodhaan, A. Al-Dhelaan, and I. F. Akyildiz, "Bordersense: Border patrol through advanced wireless sensor networks.," *Ad Hoc Networks*, vol. 9, no. 3, pp. 468–477, 2011.

[7] T. Furukawa, F. Bourgault, B. Lavis, and H. F. Durrant-Whyte, "Recursive Bayesian search-and-tracking using coordinated UAVs for lost targets," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 2521–2526, IEEE, 2006.

[8] "senseFly home webpage." `https://www.sensefly.com`. Accessed: 2015-03-27.

[9] S. d'Oleire Oltmanns, I. Marzolff, K. D. Peter, and J. B. Ries, "Unmanned Aerial Vehicle (UAV) for monitoring soil erosion in Morocco," *Remote Sensing*, vol. 4, no. 11, pp. 3390–3416, 2012.

[10] G. P. Jones, *The feasibility of using Small Unmanned Aerial Vehicles for wildlife research*. PhD thesis, University of Florida, 2003.

[11] C. Barrado, R. Messeguer, J. López, E. Pastor, E. Santamaria, and P. Royo, "Wildfire monitoring using a mixed air-ground mobile network.," *IEEE Pervasive Computing*, vol. 9, no. 4, pp. 24–32, 2010.

[12] A. C. Watts, V. G. Ambrosia, and E. A. Hinkley, "Unmanned Aircraft Systems in Remote Sensing and Scientific Research: Classification and Considerations of Use," *Remote Sensing*, vol. 4, no. 6, pp. 1671–1692, 2012.

[13] S. Rosati, K. Kruzelecki, G. Heitz, D. Floreano, and B. Rimoldi, "Dynamic routing for flying ad hoc networks.," *CoRR*, vol. abs/1406.4399, 2014.

[14] "Gumstix Inc. home webpage." `https://www.gumstix.com/`. Accessed: 2015-03-27.

[15] "Yocto Project home webpage." `https://www.yoctoproject.org/`. Accessed: 2015-03-27.

[16] "FRITZ!WLAN USB Stick N v2 Overview." `http://fritzbox.com.au/product-fritz!wlan-usb-stick-n.html#1`. Accessed: 2015-03-27.

[17] "Linksys AE3000 N900 Dual-Band Wireless-N USB Adapter." `http://www.linksys.com/us/support-product?pid=01t80000003K7a3AAC`. Accessed: 2015-03-27.

[18] N. Abramson, "The aloha system: another alternative for computer communications," in *Proceedings of the November 17-19, 1970, fall joint computer conference*, pp. 281–285, ACM, 1970.

[19] Widjaja, J. G. Kim, and Sakai, "IEEE 802.11 Wireless Local Area Networks," *IEEE Communications Magazine*, pp. 116 – 126, September 1997.

[20] ABI research, "Wireless Connectivity Market Data," March 2012.

[21] "Wireshark web page." `https://www.wireshark.org/`. Accessed: 2015-03-27.

[22] "Radiotap header fields." `http://www.radiotap.org/defined-fields`. Accessed: 2015-03-27.

[23] "Scapy project home webpage." `http://www.secdev.org/projects/scapy/`. Accessed: 2015-03-27.

[24] "Libcrafter project home webpage." `https://code.google.com/p/libcrafter/`. Accessed: 2015-03-27.

[25] J. Fink and V. Kumar, "Online methods for radio signal mapping with mobile robots.," in *ICRA*, pp. 1940–1945, IEEE, 2010.

[26] M. Saxena, P. Gupta, and B. N. Jain, "Experimental Analysis of RSSI-based Location Estimation in Wireless Sensor Networks," in *In Proc. Int. Conf. Communication System Software and Middleware*, pp. 503–510, 2008.

[27] A. Zanella and A. Bardella, "RSS-Based Ranging by Multichannel RSS Averaging," *Wireless Communications Letters, IEEE*, vol. 3, pp. 10–13, February 2014.

[28] Benkic, Malajner, and Planinsic, "Using RSSI value for distance estimation in wireless sensor networks based on ZigBee," in *In Proc. of 15th International Conference on Systems, Signals and Image Processing, 2008*, IEEE, 2008.

[29] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001.

[30] K. P. Murphy, *Machine learning: a probabilistic perspective*. Cambridge, MA: The MIT Press, 2012.

[31] D. Lizotte, T. Wang, M. Bowling, and D. Schuurmans, "Automatic gait optimization with gaussian process regression," in *in Proc. of IJCAI*, pp. 944–949, 2007.

[32] V. M. Cora, "Model-Based Active Learning in Hierarchical Policies," Master's thesis, Simon Fraser University, 2008.

[33] Garnett, Roman and Osborne, Michael A. and Reece, Steven and Rogers, Alex and Roberts, Stephen J., "Sequential Bayesian Prediction in the Presence of Changepoints and Faults.," *Comput. J.*, vol. 53, no. 9, pp. 1430–1446, 2010.

[34] M. Frean and P. Boyle, "Using Gaussian processes to optimize expensive functions," in *AI 2008: Advances in Artificial Intelligence*, pp. 258–267, Springer, 2008.

[35] J. C. Gittins, *Multi-armed Bandit Allocation Indices*. Wiley, Chichester, NY, 1989.

[36] M. D. Richard and R. P. Lippmann, "Neural network classifiers estimate Bayesian a posteriori probabilities," *Neural computation*, vol. 3, no. 4, pp. 461–483, 1991.

[37] "Google Maps API for developers." `https://developers.google.com/maps/`. Accessed: 2015-8-26.

[38] "Blade RF product homepage." `http://www.nuand.com/bladeRF`. Accessed: 2015-6-26.