UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE

A NetLogo Simulation Tool for UAV-based Secure Location Verification in Crowd Sensing

MSc. Computer Engineering

Candidate:
Marco RASORI

Supervisors:
Prof. Gianluca DINI
Ing. Pericle PERAZZO

Academic Year: 2015/2016

Abstract

In the last decade, *Unmanned Aerial Vehicle (UAV)* production and interest is observing a continuous growth that appears not to decline. Meanwhile, thanks to the increase in the use of personal mobile devices and their onboard sensors, which is becoming more and more widespread, a new data collection technique, named *crowd sensing*, has emerged.

Unfortunately, security remains a relevant issue, chiefly the integrity, i.e. the assurance that the information reported is trustworthy and accurate, still remains unsolved. The information the participant declares could be inaccurate or even counterfeit, due to flaws or fraud. Current literature shows no efficient solutions to the security problem, hence the arising need to point in this direction.

The idea of this thesis came from the merging of the aforementioned mobile technologies. The aim is to fill the security gap in the crowd sensing process through UAVs employment, to prove trustworthiness and accuracy of sensorial data.

The project presumes UAVs expedition in swarms where the data is originated, the authenticity of which could be promptly and directly verified thanks to the onboard sensors and, possibly, through interaction with other close sensors.

Through the deployment of a simulator, written in the NetLogo language, it has been possible to reproduce a crowd sensing system and investigate the trustworthiness gap.

We proposed and compared two different decision criteria to reveal attacks, named *Dictatorship* and *Majority*, both based on distance evaluation through radio frequency communication with the participant. In Dictatorship, it is sufficient that one UAV detects an inconsistency to warn an attack. In Majority, the half plus one of UAVs must detect an inconsistency in order to warn an attack. With regard to that, Dictatorship criterion showed certainly a better performance than Majority one.

We further focused on participants' waiting time reduction acting on the algorithms to schedule swarms missions. A *First Come First Served (FCFS)-like* routine and an *Insertion* heuristic have been deployed. Since there are no statistical differences between the two for the tests we performed, the former scheduling algorithm is preferable due to its deterministic nature.

Keywords: Secure location verification, UAV, crowd sensing

Table of Contents

List of Tables

List of Figures

List of Acronyms

UAV Unmanned Aerial Vehicle

SCIADRO SCIAme di DROni

I.d.S. Ingegneria dei Sistemi

CNR Centro Nazionale di Ricerca

GPS Global Positioning System

RSSI Received Signal Strength Indicator

VRP Vehicle Routing Problem

FCFS First Come First Served

GIS Geographic Information System

PDF Probability Density Function

TSP Traveling Salesman Problem

RNG Random Number Generator

Chapter 1

Introduction

1.1 Unmanned Aerial Vehicles (UAVs)

The word UAV, more commonly drone, refers to an aircraft with no pilot on board that can be remotely controlled either by a human pilot by means of a radio controller or by a system at the ground control station, otherwise it can autonomously navigate preplanned flight missions.

The UAVs manufacturing is a worldwide blossoming phenomenon started in the recent past and, to date, still gaining increasing interest along with conspicuous funding. A 2015 market survey lead by Teal Group[1] states that this branch will produce, on the whole, $93 billion over the next ten years. Such valuation takes into account military, civil and commercial areas. Big tech companies like Google, Facebook and Amazon feel the need to be part of the business and hence they have already started to invest in this sector. Market analysts also expect UAV employment in civil and commercial environments will represent an ever-growing share of the market.

UAVs differ broadly in size and capacity. Their design and the onboard sensors they can be furnished with, can be adapted accordingly to the specific field they are supposed to be used. For instance, the most used UAV by United States Armed Forces, the RQ-11B Raven®[2] is employed in reconnaissance and surveillance missions, so it comes with a high resolution, day/night camera and a thermal imager as main features, which are more generally called *payload*. It is also equipped with a high capacity battery to have an endurance up to 90 minutes. This UAV is a *fixed-wing* aircraft (Fig. 1.1(b)), even though most of the people consider a drone as a *quadcopter* (Fig. 1.1(a)). This is because consumer drones, like AR.Drone by

Parrot, and SOLO by 3D Robotics, are becoming everyday devices, mostly employed for aerial filming and photography, and more and more used by hobbists. This kind of aircrafts is less performing than the one used for military purposes.

(a) 3D Robotics SOLO. (b) AeroVironment RQ-11B Raven®.

Figure 1.1: Two examples of UAV airframes. A quadcopter and a fixed-wing aircraft.

Payload usually includes only a camera and the battery capacity guarantees no more than 20 minutes of flight endurance.

1.2 Crowd Sensing

Nowadays, the use of personal mobile devices, coupled with the many onboard sensors that they are equipped with (e.g. accelerometer, microphone, magnetometer, etc.), is observing a sharp increase. This development has lead to the rise of a new data collection technique, named crowd sensing.

This can be classified accordingly to type of phenomenon measured or to the the participant's involvement in such a process[3].

On the basis of the former criterion, we can discriminate three sub-categories of crowd sensing: *environmental*, *infrastructure*, and *social*. Environmental crowd sensing is used for monitoring environmental conditions and promptly detect abnormal or unwanted changes that may occur[4]. Infrastructure crowd sensing commonly regards data collection from urban and extra-urban areas to control roads traffic and maintenance[5]. Social crowd sensing embrace all aspects of social life, such as the most popular places or the most attended events, allowing individuals to anonymously share information through direct declarations or through the sensors on their portable devices.

The other criterion of classification focuses on the participant, who can actively collaborate, communicating sensed data to a server, whenever he decides to (*participatory crowd sensing*). Otherwise participants can be indirectly involved

in the automatic dispatch of data that are acquired by sensors on their devices when a specific event occurs (*opportunistic crowd sensing*).

The flexibility related to this mechanism leads to the design of highly versatile platforms that are able to replace the typical sensing infrastructures, facilitating the launch of new applications.

1.3 The SCIADRO Project

SCIADRO stands for SCIAme di DROni (*swarm of drones* is the english translation), a project funded by Regione Toscana in 2016 and proposed by the company Ingegneria dei Sistemi (I.d.S.). The duration estimation of this work is two years and it presumes the engineerization of a system composed by commercial UAVs swarms and a participatory crowd sensing platform. The objective is to give to a community of participants the opportunity of immediatly reporting catastrophic events, due to natural disasters (such as earthquakes) or fraudulent episodes (e.g. arsons). A swarm of UAVs that receives such a report, is entrusted to head to the participant's location and there, thanks to their onboard sensors, to confirm the extent of the event proving report's accuracy (intended as position and data truthfulness) and ultimately participant's trustworthiness.

To develop this system, many challenging aspects have to be analyzed and solved. Therefore it is necessary the employment and the collaboration of many research groups, which currently belong to Centro Nazionale di Ricerca (CNR) of Pisa, to department of Information Engineering of Pisa, and to I.d.S.. Researches are focusing on the deployment of a data exchange protocol, both among UAVs belonging to the same swarm, and between these UAVs and the ground control station, on image processing and obstacles avoidance algorithms in respect of UAVs, on the creation of a crowd sensing framework, and on the investigation of its security-related problems.

1.4 Aims and Objectives

This thesis investigates security-related issues concerning the SCIADRO project. Specifically, in a crowd sensing system, these problems refer to a lack of a suitable mechanism able to guarantee that the information the participant communicates is accurate and trustworthy. Indeed, it is not unusual that a malicious partici-

pant succeeds in falsifying a sensor measurement or the GPS position where that measurement is declared to have been registered. Likewise, sensor malfunctions are plausible, but a traditional crowd sensing system is not able to identify them. Differently, the proposed system is inclined in tracking down and becoming aware of both these cases.

Chapter 2

System and Adversary Model

System is organized as follows: UAVs are grouped into swarms, each one waiting for a request at the assigned base, or *depot*, within an urban area. Participants walk on the roads and, reached a junction, they randomly choose the next one to follow, if there exists at least another road, otherwise (cul-de-sac) they travel backwards. They hold a constant speed and whenever they sense an *event* they report it and wait there for a swarm to come. Hence, a selected swarm goes in the proximity of the place where the report originated (*waypoint*), with the aim of proving data authenticity and participant's position. Participants can act either *honestly*, and therefore they correctly report ongoing events, or *maliciously*, that means they report a counterfeit position and a non existent event. On the other hand, events are supposed to spawn at random places and random times, and they are characterized by an arbitrary duration and a certain extension, beyond which participants cannot sense them. UAVs are battery-powered and they can travel and serve participants until their battery level reaches a safety threshold beyond which the way back to their depot is mandatory. Once there, the entire swarm must wait until all its UAVs have their batteries fully charged before being allowed to leave again the depot for a new mission.

2.1 Position Verification Method

The project entails that UAVs move toward the point where the report is generated, with the aim of promptly and directly proving its authenticity thanks to their onboard sensors. By means of radio frequency communications, GPS position from where the participant stands, and from which the report has been sent,

can be proved[6]. UAVs are trusted devices and their evaluations are known to be reliable, therefore a UAV single unit or a swarm can be headed to the specific point of interest where the untrustworthy participant has declared the event took place, with the aim of verifying both the position and the event itself. UAVs will arrange themselves randomly around the participant, in a way their positions are not predictable by the participant, with whom each one will communicate with. In our model, we considered a maximum communication range of $250\,m$. After the execution of an authentication protocol, each drone will check if participant's declared position is consistent with the one it independently computes by means of Received Signal Strength Indicator (RSSI).

Although each swarm can be considered as a single entity, it is composed of a number of UAVs that act as individual units. For this reason, it arises the need of a unitary criterion to decide if the declared position is false or not, in the chance UAVs do not provide a unanimous opinion. A possible approach to deal with this issue is to blindly trust any UAV that affirms to be under attack, in a way a single individual can decide the fate of the swarm's opinion. Since there is a kind of similarity to the well-known authoritarianism form of government, from now on, we will refer to this criterion as *Dictatorship*. Another common approach, antithetical to the above mentioned one, could be to let the majority of the UAVs decides the final verdict. Thus, at least half plus one the number of drones, should endorse the attack to determine it as swarm's opinion. Henceforward, we will call this last *Majority*.

2.2 Swarms Coordination

A swarm spends time moving from the depot or from a waypoint to another waypoint, or from a waypoint back to the depot. Therefore pending participants have to wait for UAVs for a certain amount of time. Assuming the presence of more than one swarm in the system, the determination of the one to dispatch to a participant is crucial to determine his waiting time. Hence, we could try to reduce the average waiting time, and serve participants on the basis of the actual position of the swarms along with the evaluation of their capabilities. In this regard, a Vehicle Routing Problem (VRP) algorithm, introduced by Solomon et al.[7] and called *Insertion heuristic*, seems to be a good compromise between the algorithm complexity and the waiting time reduction, tailored for this kind of

systems. It will be compared to a FCFS-like algorithm to understand which one performs better in our case of study.

2.3 Adversary Model

In our model a specific participant can act both honestly and maliciously during a simulation. Each participant who senses an event, always reports it correctly and, obviously, he is admitted to report the same event just once. He will act fairly until the communication with the UAVs ends.

The same participant, at a certain time instant, can decide to become an adversary falsifying his position during the report. He dishonestly claims he stands $50\,m$ far from his real position, in any direction. The swarm will move toward his declared position and will try to confirm it by means of the position verification method mentioned in Section 2.1. The probability a participant becomes an adversary follows an exponential random variable which will determine the instant it happens.

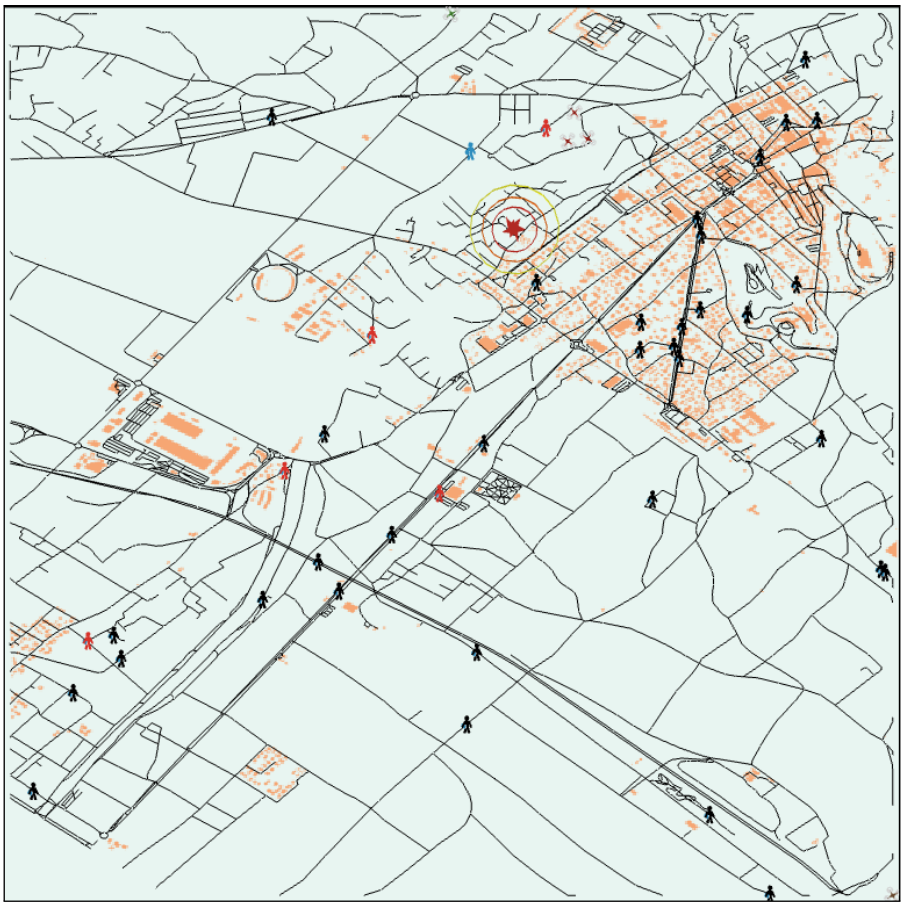
Chapter 3

Evaluation Method

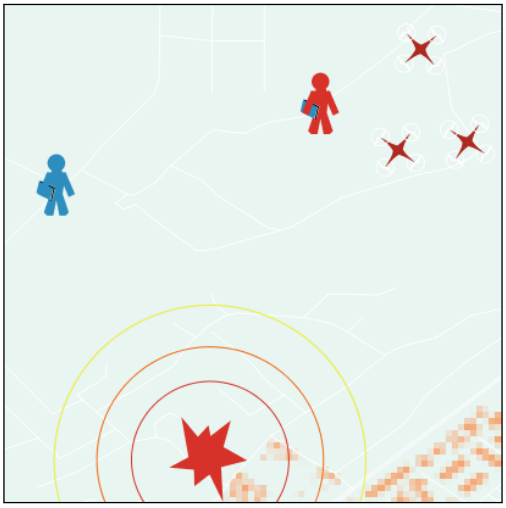
3.1 NetLogo Simulator

In order to study security-related problems affecting the SCIADRO project, and especially the secure location verification method, a simulator has been developed by means of NetLogo. This programming language has been chosen due to its agent-based approach, highly suitable for our purpose. NetLogo offers the opportunity to create and program agents, or *turtles*, able to produce specific actions; since we needed different turtles with different behaviors, we created five kinds of customized turtles, called in this language *breeds*. *Drones*, *participants*, *swarms*, *events*, and *vertices* have peculiar attributes identifying their roles, e.g. *drones* are characterized by a flying speed and a battery level, while *participants* have a walking speed and a positive waiting time (if they are expecting to be reached by a swarm), and so on and so forth. A map $5 \times 5\, km$ sized, and representing the Italian city of Massa, has been loaded thanks to an extension called GIS (Geographic Information System) in order to let *participants* walk on the roads, which are defined by a graph through the use of *vertices* breed. They represent junctions among the roads and own, as customized parameter, a list of other neighbour *vertices* they are linked to.

During the simulations, pending *participants* are coloured in blue in case they have reported a real event and their right location, whereas they become red if they have reported a counterfeit position. Otherwise, when they are just walking on the roads in absence of an event to report, they are represented in black (Fig. 3.1).



(a) Graphical Interface. Pending malicious participants are in red, while honest in blu. All the others, which are not pending, are coloured in black. Events are represented with a red burst and circles giving the idea of their perceived extension.



(b) A detailed view.

Figure 3.1: A frame of the graphical interface during a simulation.

In Figure 3.1 is reported NetLogo interface during a simulation where *drones*, *participants*, and *events* are depicted. The other two breeds are hidden: a *swarm* is just a concept (with attributes, though) and there is no need to point out *vertices* even if they are essential to build the road map.

Figure 3.2 shows another part of the interface where the user can configure the system, acting on *sliders*, *choosers*, *inputs*, and *switches*. Then *setup* routine have to be called to let desired configuration take place. After that, *go* routine is invoked and loops till the *stop condition*, in our case the *number of measurements* to be done, is not met. Right column, coloured in beige, cannot be directly edited by the user, instead it refers to *computed parameters* that are determined during the ongoing run.

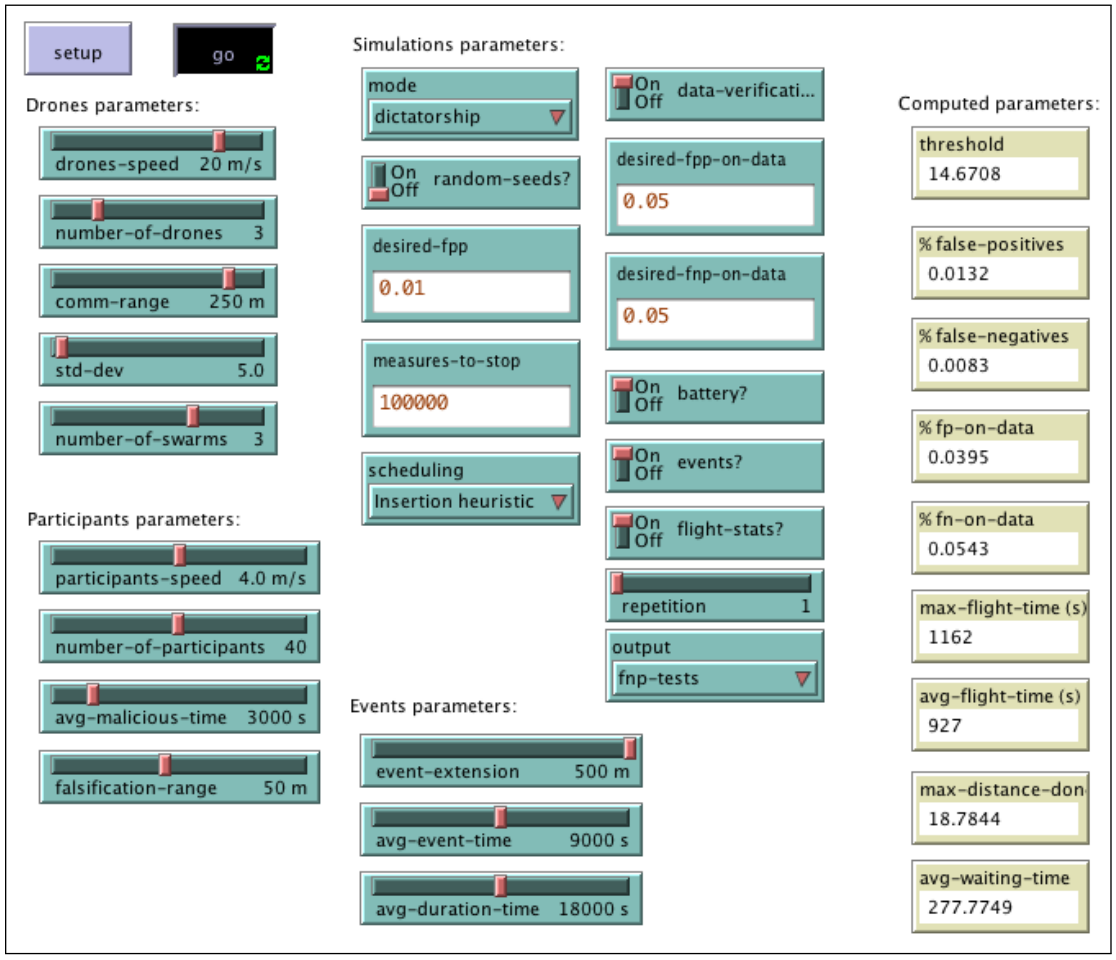


Figure 3.2: Part of graphical interface.

Interface elements are divided into groups. *Drones parameters* include five sliders, so the user can choose UAVs speed (*drones-speed*), the number of UAVs in each swarm (*number-of-drones*), the maximum possible distance at which a UAV can stand from the participant's declared position (called *communication-range*), the amount of swarms (*number-of-swarms*), and another technical parameter we will introduce later (*std-dev*). *Participants parameters* include their walking speed (*participant-speed*), the quantity of participants in a simulation (*number-of-participants*), the measure in which a participant will falsify his position in a circumference of radius equal to *falsification-range*, and *avg-malicious-time*. The last is given as parameter λ of an exponentially distributed random variable in order to have, on the average, one malicious participant every *avg-malicious-time*. The breed *events* is characterized by a size (*event-extention*), by an average time of appearance (*avg-event-time*), and by an average time of persistence (*avg-duration-time*). These last two are modeled in the same way in which *avg-malicious-time* is designed. Among the *Simulation parameters*, we should cite *desired-fpp* that is important for the secure location verification dynamics. It refers to the ratio of false positive we intend to tolerate.

3.2 Implemented Algorithms

3.2.1 Attack identification and false negative probability estimation

We want to find out the false negative ratio using Dictatorship and Majority criteria imposing the false positive ratio. Thresholds beyond which a UAVs declares to be under attack, will vary accordingly with the number of drones in a swarm and the criterion adopted.

Threshold computation for Dictatorship and Majority criteria

Since a simulator has been created, the distance evaluation (ranging) error, committed by radio frequency device onboard of UAVs, has been simulated as well. Such error has been modeled as a random variable with a Gaussian distribution with standard deviation $\sigma = 5\,m$. On this basis, it has been possible to compute an appropriate *threshold*, establishing an a priori false positive ratio. Assume now a scenario in which all participants are honest.

We now consider the two principles introduced in Section 2.1, called *Majority* and *Dictatorship*. Compute the threshold for the latter is straightforward. Recall that in Dictatorship, if at least one UAV believes to be under attack, the swarm will claim to be under attack. A UAV states it if the difference between computed distance from its position and the one declared by the participant (d_{GPS}), and the distance computed by the mean of radio frequency communication (d_{RF}) is greater than the decided threshold.

$$|d_{GPS} - d_{RF}| > threshold \tag{3.1}$$

Assuming d_{GPS} a unbiased estimate, threshold is computed accordingly to the error a UAV can commit during d_{RF} estimation. Using Dictatorship criterion, a swarm claims not to be under attack (event $\overline{attack}$) if and only if each UAV judges itself not to be under attack (event $\overline{d\text{-}attack}$).

$$P(\overline{attack}) = P(\overline{d\text{-}attack})^n \tag{3.2}$$

where n is the number of UAVs. Recalling we are assuming a honest scenario, summing the preset false positive probability (fpp) to the probability to state not to be under attack ($P(\overline{attack})$) we obtain 1 as stated by the normalization condition, therefore:

$$P(\overline{attack}) = 1 - fpp \tag{3.3}$$

Imposing fpp, the probability a UAV will assert not to be under attack ($P(\overline{d\text{-}attack})$) is obtained as follows:

$$P(\overline{d\text{-}attack}) = \sqrt[n]{P(\overline{attack})} \tag{3.4}$$

We can now compute the fpp related to the single UAV ($d\text{-}fpp$) as

$$d\text{-}fpp = 1 - P(\overline{d\text{-}attack}) \tag{3.5}$$

This one corresponds to a portion of the area under the Probability Density Function (PDF) of the Gaussian distribution that identifies ranging error committed by the single UAV. Computing the inverse function we can obtain the threshold value to achieve the desired fpp value.

Computing threshold for Majority criterion required the creation of an extension for NetLogo[8]. The probability a swarm considers itself under attack is composed of a subset of outcomes: any subgroup of $\lfloor \frac{n}{2} \rfloor + 1$ of UAVs, and so on for any group until n. We refer to the following equation related to a binomial distribution:

$$P(X \geq k) = \sum_{k=maj}^{n} \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k} \tag{3.6}$$

Where $p \in [0, 1]$, $n \in N$ represents the number of independent experiments, and maj is the lower bound in the probability evaluation. Specifically, for the Majority criterion, maj is the number of votes claiming an attack to reach the majority (in our case $\lfloor \frac{n}{2} \rfloor + 1$), and so the Equation 3.6 can be expressed as:

$$P(attack) = \sum_{k=\lfloor \frac{n}{2} \rfloor + 1}^{n} \binom{n}{k} \cdot P(d\text{-}attack)^k \cdot (1 - P(d\text{-}attack))^{n-k} \tag{3.7}$$

The extension includes a method that takes as inputs binomial distribution parameters, such as n (number of drones), maj, and the false positive probability we desire; the output it returns, corresponds to the fpp related to the single UAV ($d\text{-}fpp$). Recalling that ranging error is distributed as a Gaussian random variable, we proceed in the computation of threshold value as in the previous case.

3.3 Average Waiting Time reduction

An important aspect we shall take into account is the time between a participant request and the UAVs measurements for that participant. In general, such time interval is called *average waiting time* and it is desirable to be as short as possible.

In literature, VRP is a branch of problems in operations research that derives from the Traveling Salesman Problem (TSP) and that examines this aspect. It refers to a fleet of vehicles, in our study the swarms, which moves from a depot and have to visit a given number of customers before coming back to the depot[9]. We do not consider time constraints but obvious restrictions are imposed by UAVs' battery capacity. The challenge is to minimize the total route cost and therefore the paths covered by the set of vehicles complying with our constraints. Since participants can report events at any time, our optimization problem becomes dynamic and so the scheduling algorithm needs to recompute and optimize routes on these happenings[10]. The FCFS-like algorithm consists in a decision based on the distance a swarm should cover to visit the participant, who just requested the

service, from the last participant already scheduled but not visited if the swarm is performing a mission. A slightly different case occurs when a swarm is moving toward the depot; in this circumstance, distance is the one that is between swarm actual position and new participant's position. Clearly, loads constraints have to be satisfied, so every single UAV belonging to the swarm needs to verify its own battery restrictions for the swarm to be eligible. If a swarm is waiting at the depot, and the batteries of all its UAVs are fully charged, such a swarm is eligible as well and its cost for the new request coincides with the distance from the participant to the depot. Finally, a suitable swarm with the lowest cost is chosen and the request queued to its *service list* like in FCFS algorithm. So far, a method to decrease just the total distance traveled by swarms has been taken into account. If we want to reduce the participants' waiting time, we could let a swarm serve a new incoming request next to its position and then proceed with its mission, if this change in direction respects total route cost minimization and obeys system constraints. This is exactly Insertion heuristic goal; during the scheduling phase, each swarm takes its own *service list*, which includes participants positions already scheduled but not visited, and computes costs of insertion for the incoming request. At each iteration, it is located between a pair of items in the service list and the swarm route cost for that configuration is computed afterwards. Finally, the minimum cost is taken and compared to the costs estimated by the other eligible swarms. Again, the minimum cost is chosen and the request assigned to the relative swarm. This algorithm's first objective is still the minimization of the total route costs but, obliquely, it tries to reduce participants' average waiting time. Statistical analysis has been performed on these methods and will be discussed further.

```
repeat
    select a swarm from the pool;
    if swarm is eligible then
        compute and store cost for the swarm;
    end
    remove the swarm from the pool;
until the pool is not empty;
select the swarm with the lowest cost;
update swarm service list and total cost properly;
```

Figure 3.3: Scheduler pseudocode.

Figure 3.3 shows pseudocode for the scheduler; *compute and store cost for the swarm* is the part where the two algorithms differ. Figs. 3.4, 3.5 contain pseudocode both for the FCFS-like algorithm and Insertion heuristic. *Service list* is an ordered list of agents; it is sorted according to service order, so there are participant agents and the last element is the depot. $d(i, i+1)$ stands for the spatial distance between the ith and the $(i+1)$th element of the list. *Sum of distances* is a variable that contains partial costs during the iteration and finally represents total route cost after the inclusion of the new participant.

```
make a copy of the service list;
insert unrouted participant to the list before the last entry;
if swarm is going to the depot then
    insert swarm current position to the list as the first entry;
end
foreach element i < service list length do
    compute d(i, i + 1);
    update sum of distances;
end
if battery constraints are still respected then
    compute net cost for the inclusion;
    return net cost;
else
    return an error of scheduling not feasible;
end
```

Figure 3.4: Part of the FCFS-like routine pseudocode.

In Insertion heuristic *minimum sum of distances* is the minimum among all *sum of distances*. Battery constraints have been estimated taking general civil UAVs data sheets ; by means of such specs we were able to find out the maximum distance a UAV can travel before a certain battery level, used as safety threshold (5% in this study). Clearly, a schedule is feasible for a swarm only if every UAV can afford the new potential mission. Since a swarm is a set of UAVs located at different coordinates, *swarm current position* is considered to be the one of the farthest UAV from the subsequent element of the list. In the same way, every time the distance between two consecutive participants is computed, we examine the worst case, i.e. the most distant position a UAV could get after running the method to chose a random position, within communication range limit, around the participant. Whenever the swarm finishes to deal with a participant, these new coordinates are computed by each UAV starting from participant's declared position, so there is not any a priori knowledge about this and we always need to suppose a worst case reasoning.

```
make a copy of the service list;
for j ← 1 to service list length − 1 do
    insert unrouted participant after jth element of the list;
    insert swarm current position as first element of the list;
    foreach element i < service list length do
        compute d(i, i + 1);
        update sum of distances;
    end
    if sum of distances < minimum sum of distances then
        update minimum sum of distances;
    end
end
if battery constraints are still respected then
    compute net cost for the inclusion;
    return net cost;
else
    return an error of scheduling not feasible;
end
```

Figure 3.5: Part of the Insertion routine pseudocode.

Chapter 4

Tests and Results

4.1 Majority and Dictatorship comparison with regard to false negative probability

An essential ambition of this study is to maximize the chances of detecting an attack to increase system trustworthiness.

In order to do that, evaluation and minimization of false negative probability were crucial points. Taking advantage of the algorithms explained in Chapter 3, simulation campaigns have been performed; the goal was to establish which behaves better between Majority and Dictatorship criteria.

NetLogo offers the opportunity to run parallel simulations to speed up data collection. We gathered 32 repetitions for each scenario, changing Random Number Generators (RNGs) seeds every repetition, in order to have sufficient data sets to work with. For each different scenario, we used a Gaussian random variable to make statistical inference because the number of independent samples was large enough. Consequently we computed *sample means* and 95% *confidence intervals* after *truncation points* evaluation. Data manipulation has been carried out using MATLAB, importing `csv` output files provided by NetLogo.

Simulations have been run changing the number of UAVs composing the swarm and therefore threshold (see Section 3.2.1) is the only parameter influenced either by swarm dimension (size) and used criterion. Table 4.1 shows threshold values (in metres) for our scenarios.

While thresholds for Dictatorship mode ascend monotonically when increasing the number of UAVs, these ones for Majority tests decrease wavering. Such behaviour is clearly observable in Figure 4.1 and it is ascribable to binomial distri-

Criterion	Number of UAVs							
	3	4	5	6	7	8	9	10
Majority	9.4446	7.3628	8.0905	6.812	7.3369	6.4335	6.8455	6.1548
Dictatorship	14.6708	15.111	15.4452	15.7138	15.9379	16.1298	16.2975	16.4463

Table 4.1: Threshold values (in metres) computed with $fpp = 0.01$.

bution discrete nature. Let n be the number of UAVs belonging to a swarm. By definition, more than half the votes is needed to reach the majority. If n is an even number, swarms of $n + 1$ and n UAVs will require the same number of votes to achieve the majority condition. As consequence, binomial extension will compute different *d-fpp* and hence thresholds values, resulting in a higher value for the odd number when using (3.1). As n grows larger, the difference between n and $n + 2$ grows smaller.On the other end, referring to Dictatorship, there is a direct proportionality between thresholds and n. We can simply explain this result with an example: imagine we are dealing with a honest participant and the swarm is composed by 3 UAVs; if at least one of them gets a measurement wrong, that participant is flagged as an attacker. Imagine now we have a 100 UAVs swarm, all UAVs needs to remain within the threshold when making their evaluation to reach the right verdict. Straightforwardly, it is essential to increase threshold value in order to achieve the same false positive probability of the previous case.

Figure 4.1 shows the false negative probability varying the number of UAVs composing a swarm both for Dictatorship and Majority criteria. The former rule performs clearly better than Majority one starting from the very minimum number of UAVs. This is another metric we would like to keep as low as possible since UAVs are a cost to the system.

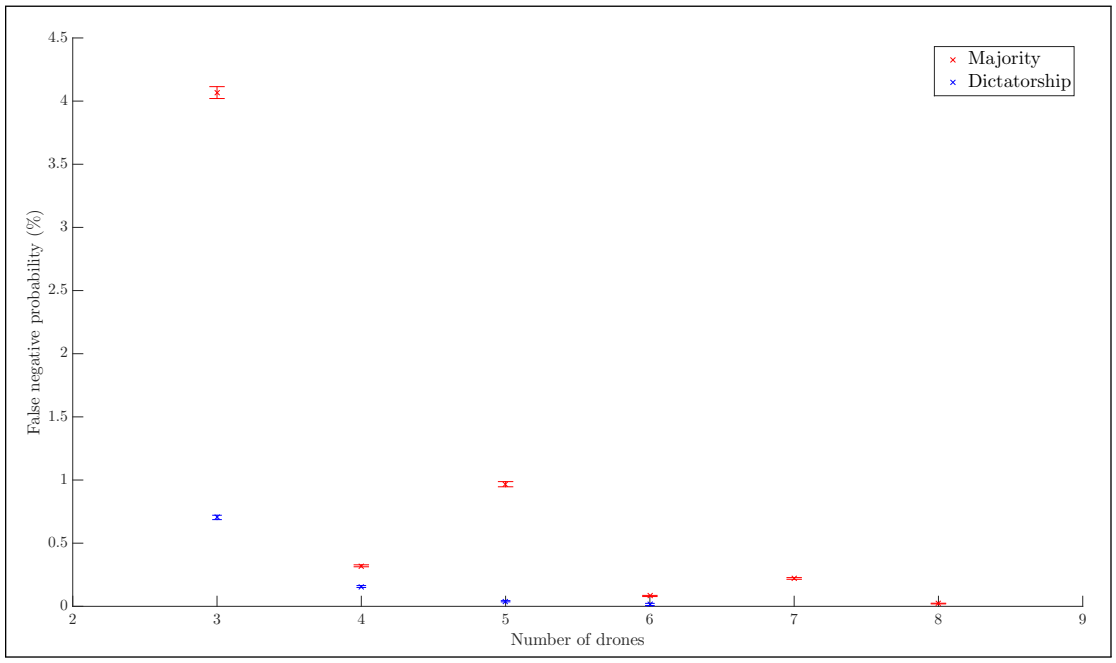


Figure 4.1: Comparison of Majority and Dictatorship criteria in terms of false negative probability (%) for different numbers of UAVs in a swarm.

The cause of this behaviour is clear after observing the arrangement of the drones around the participant in cases where Majority rule was deceived by the attacker while Dictatorship revealed indeed the attack. As shown in Figure 4.2, d_1 is the only one that perceives the attack; the other UAVs are unfortunately disposed in a way such that the distance between them and the attacker is approximately the same as the one between UAVs and his false declared position.

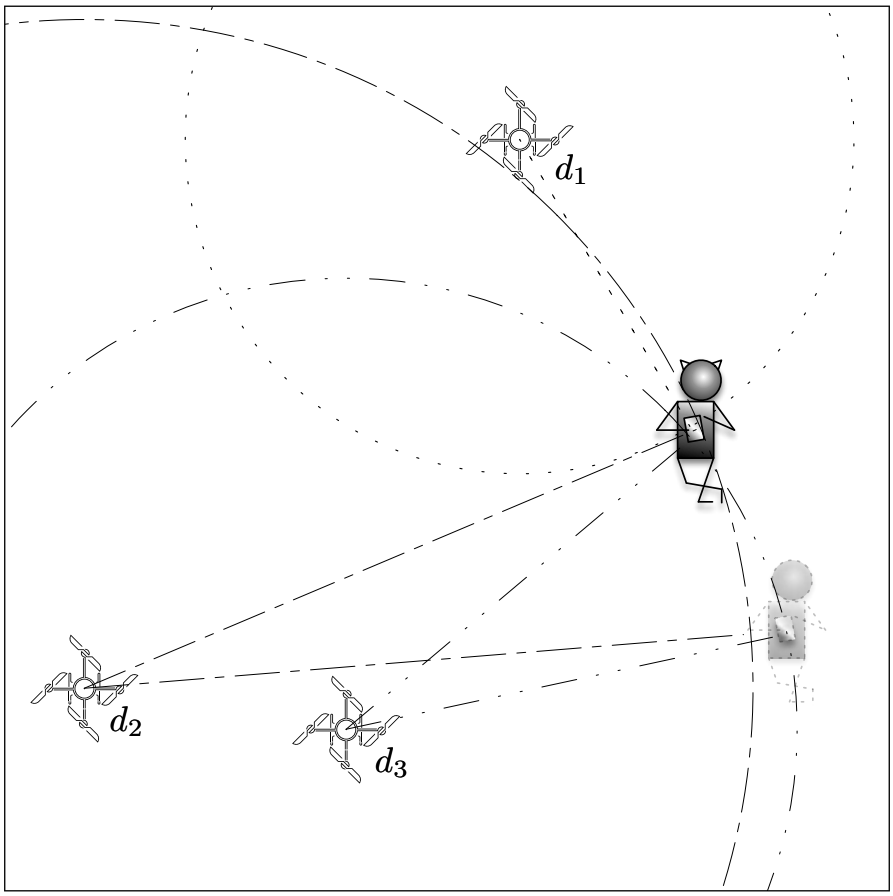


Figure 4.2: A configuration in which Majority criterion does not reveal the attack.

In such circumstances, attacker succeeds in cheating the system. This is not true with Dictatorship criterion where, as a consequence of a higher threshold value, is less likely for a UAV to detect an attack, nevertheless if there is at least one drone declaring an attack, the rule establishes that one as the whole swarm judgement.

4.2 Scheduling algorithms evaluation

A number of simulation campaigns have been executed in order to understand which scheduling algorithm achieves better results in terms of participants' average waiting time among all the served requests. Another connected metric we have taken into account is the average number of not served participants. Let us recall that a participant is not served when no swarms are eligible at the time a request occurs; that is either a swarm is recharging batteries at its depot, or it is flying

but battery is not enough to handle the new request. Concerning these tests, we decided to change the number of swarms in each scenario, and to compare the two adopted techniques. We started from 1 swarm and then we increased that number up to a maximum of 4, so we had four different scenarios for every algorithm. Furthermore, we decided to test the system under different load conditions with regard to the events rate, and so varying *avg-event-time* parameter. Table 4.2 shows and names these cases.

Load	*avg-event-time* (s)
Light	18000
Moderate	9000
Heavy	4500

Table 4.2: Simulations scenarios names and parameters.

When a participant has been served, its waiting time is stored in an output file. During the last step of the simulation, number of not served participants is saved as well. Again, we manipulated data using MATLAB and plotted sample means for these two quantities together with 95% confidence intervals for both scheduling algorithms. Simulations have been run in a *honest scenario*, so no attacks were possible and therefore the entire run length was governed by trustworthy event reports. This means load conditions are affected only by the parameter in Table 4.2 and by another one that is related to events persistence, which in our case has been considered as a constant (*avg-duration-time* = 18000 s). Another way to diversify the load could have been obtained by acting on the number of participants, but we decided to keep it unvaried in these simulation campaigns, where *number-of-participants* was equal to 20.

In Figure 4.3 are plotted results for the first set of simulations. In a lightly loaded context, we cannot infer any statistical consideration because confidence intervals in both the graphs are overlapped for all the scenarios.

This is because, with such a load, requests are sporadic and swarms serve few participants during every mission. Particularly, in this scenario all the pending participants are close to the ongoing event, which is usually unique due to the low frequency they appear in the simulation, and their duration. Indeed, an event manifests every 1/18000 seconds, and the period it lasts is an average of 18000 seconds (recall that *avg-event-time* and *avg-duration-time* are both modelled by exponential random variables). Figure 4.4 shows how, with Insertion heuristic,

a UAV travels less distance then the one it would cover with the other algorithm proposed. Nevertheless, considering what we just evidenced, the average waiting time is not lowered enough to assert that an algorithm performs better than the other.

However, in Fig. 4.3 we can notice a ratio about 0.5 using a single swarm. It is unacceptable to miss one request over two done. This value will grow up more and more increasing the load, so we can exclude the possibility one swarm is sufficient to guarantee a tolerable service in such a system. Even two swarms miss a huge amount of requests, specifically more than one out of every four for both the algorithms, though a consistent decrease of the average waiting time is noticeable and is around $20\,s$. Either the average waiting time and the not served ratio get a more evident shrink using 3 and 4 swarms. In particular, the average waiting time is lowered more than the 25% with regard to the case with 1 swarm. Additionally, the not served participants ratio assumes passable values (lower than 0.05).

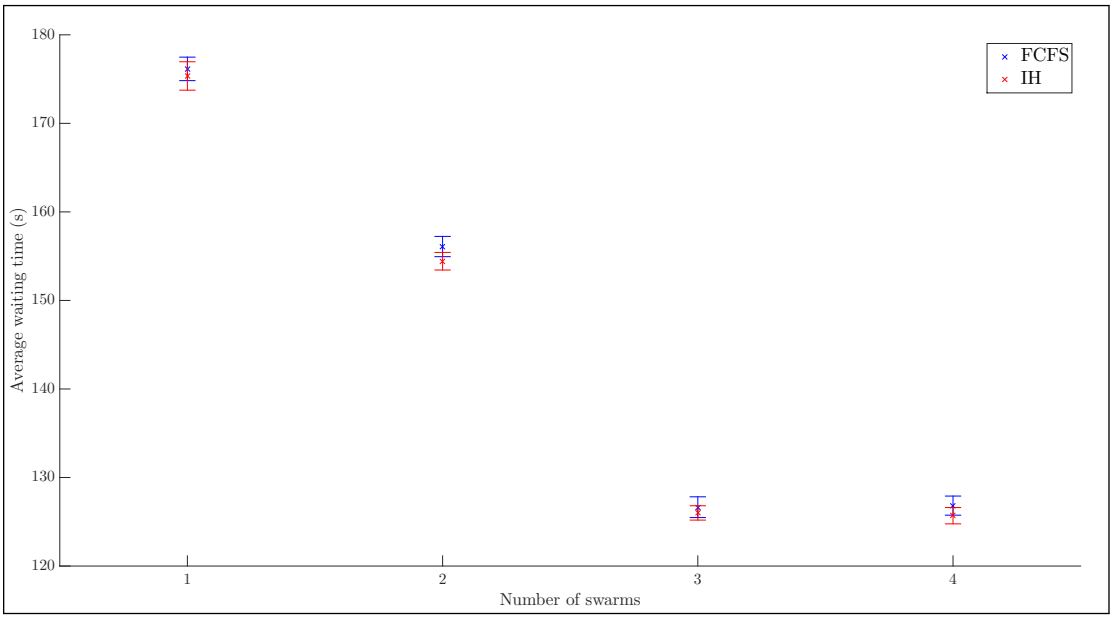
In the second set of simulations (*moderate* load) the number of participants and the event duration have been left unchanged, but the event rate has been doubled. Graphic results connected with the two metrics we are taking into account are observable in Fig. 4.5. It is easy to see in Fig. 4.5(b) that all the ratios regarding the not served participants increased with respect to the previous case, and all the arguments we exposed for that still remain valid. These simulation results are useless since they do not further augment our knowledge with regard to which scheduling algorithm employs a lower average time to serve the requests. However, all the sample mean values related to Insertion heuristic are lower than the other ones, so we decided to increase system's load again to understand if we can draw a conclusion about the algorithms.

For our last set of simulations, *avg-event-time* is four times lower than the original one, with events happening on average every $4500\,s$. Due to this heavy load condition, average waiting times for both the algorithms are higher and participants generally have to wait up to $10\,s$ more than the *moderated* load configuration. Here for the first time, we can affirm that Insertion heuristic works better than the other algorithm considering the single swarm scenario.

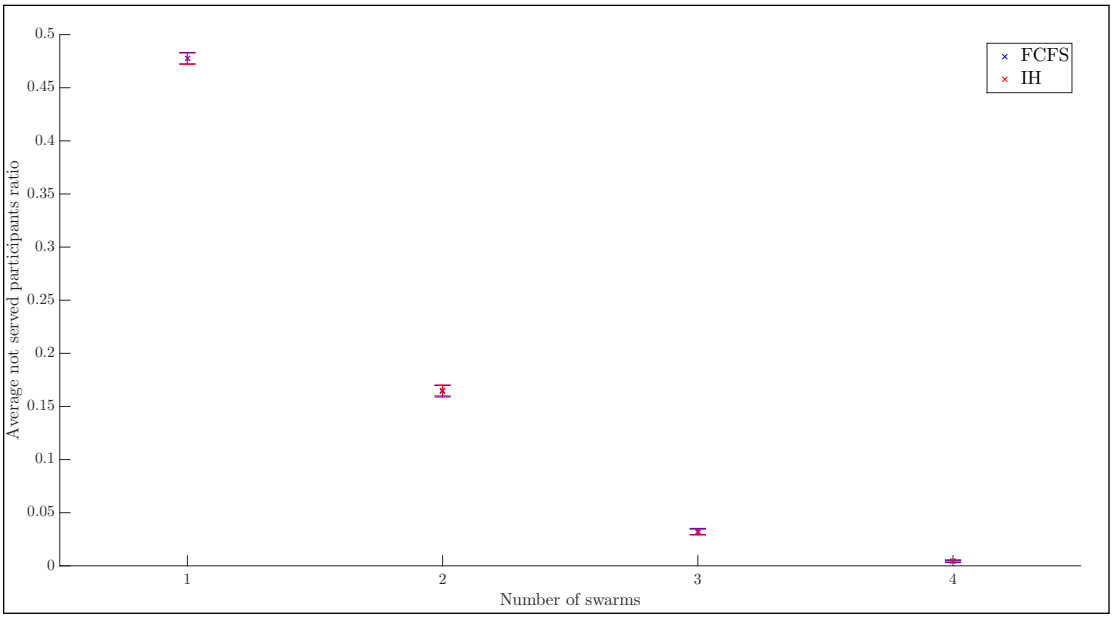
For every load condition, we noticed that the results of average waiting times related to a single swarm performing evaluations remains almost unchanged. This behaviour is reasonable due to battery constrains which generally affect swarms. On the other hand, differences among increasing event frequencies are detectable

in the not served ratio, where the number of not served participants raises with the loads for every the number of swarms employed, even for the case of a single swarm. The explanation for this trend lies on the higher number of events which lead to an higher number of requesting participants.

To better understand the chain of events which develops within our system with a growing complexity, we must draw attention to the nature of the *FCFS-like* algorithm. The fact that we named it *-like* derives from a compromise we made to reach the best possible results in serving the higher number of pending participants along with system costs optimization. Hence, our algorithm is not a *pure-FCFS*, instead when a participant reports an event he is assigned to the swarm for which his evaluation costs less. Therefore a participant who reports an event subsequent to other participants already pending for a swarm to come, could be served before them if he is assigned to a different swarm. The algorithm presumes the FCFS method is limited within each swarm, where there is a well-established order to be followed. Thus, we can affirm there is a certain similarity with the Insertion heuristic in its behaviour, but also in the results obtained for the scenarios we tested.

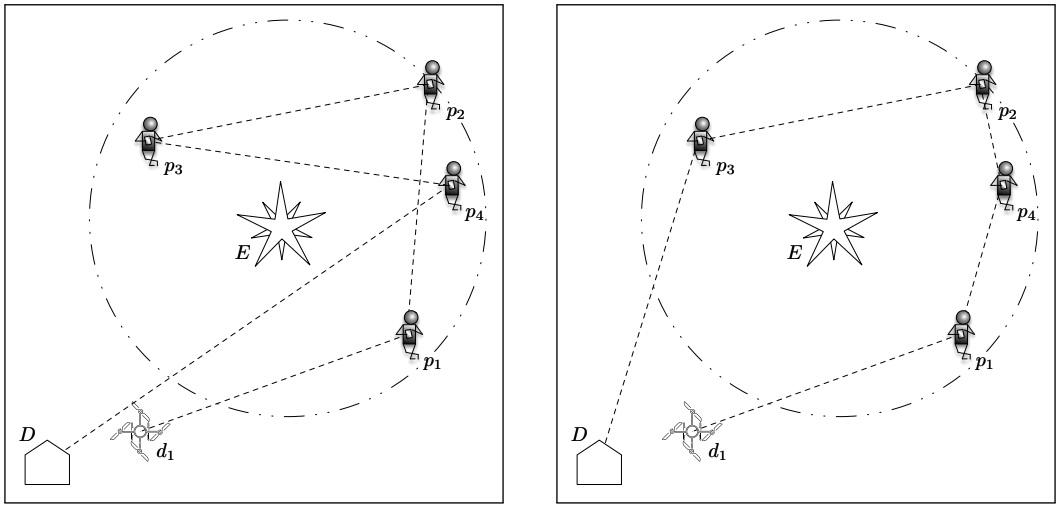


(a) Average waiting time.



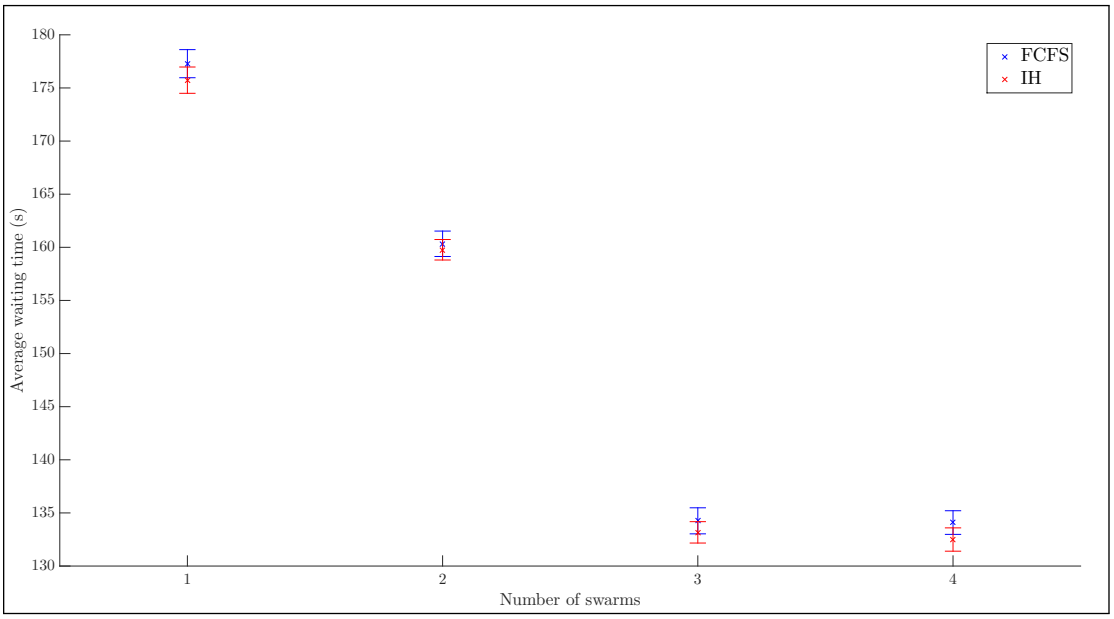
(b) Average not served participant ratio.

Figure 4.3: Graphical results for *light* load simulations with both the scheduling algorithms, varying the number of swarms. In the graphs are shown average values and 95% confidence intervals.

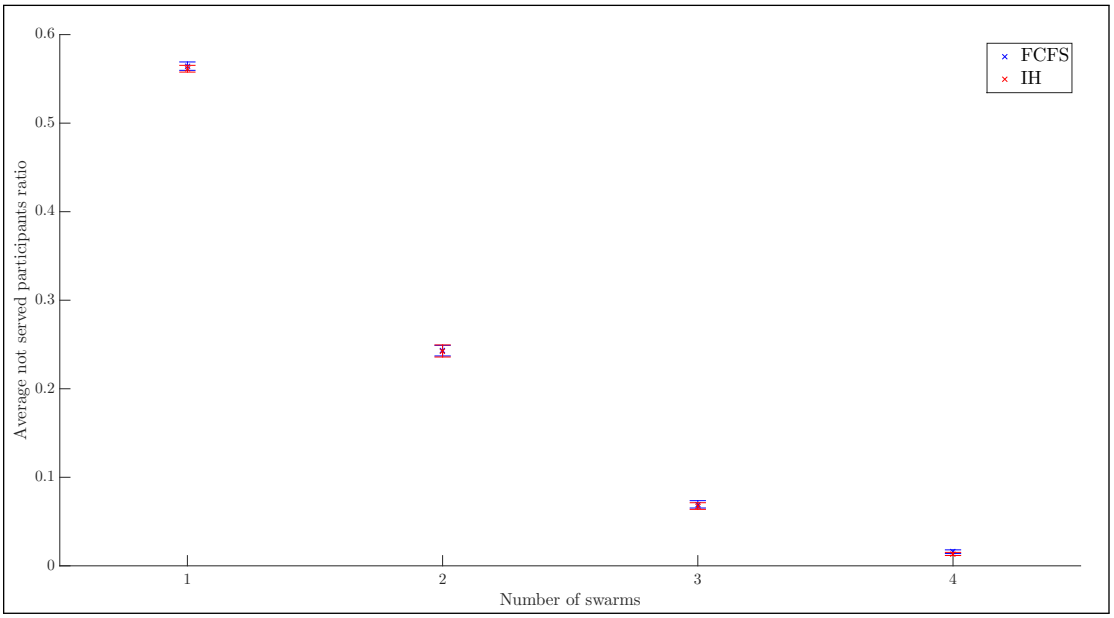


(a) FCFS-like routine behaviour. (b) Insertion heuristic behaviour.

Figure 4.4: A case in which Insertion performs better than the FCFS-like routine.

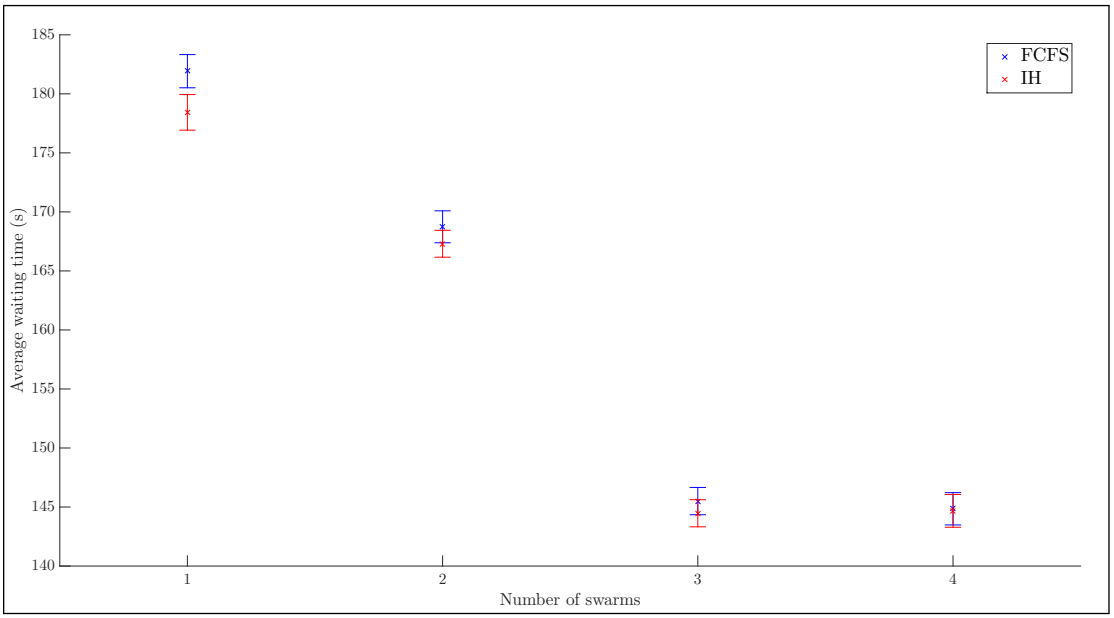


(a) Average waiting time.

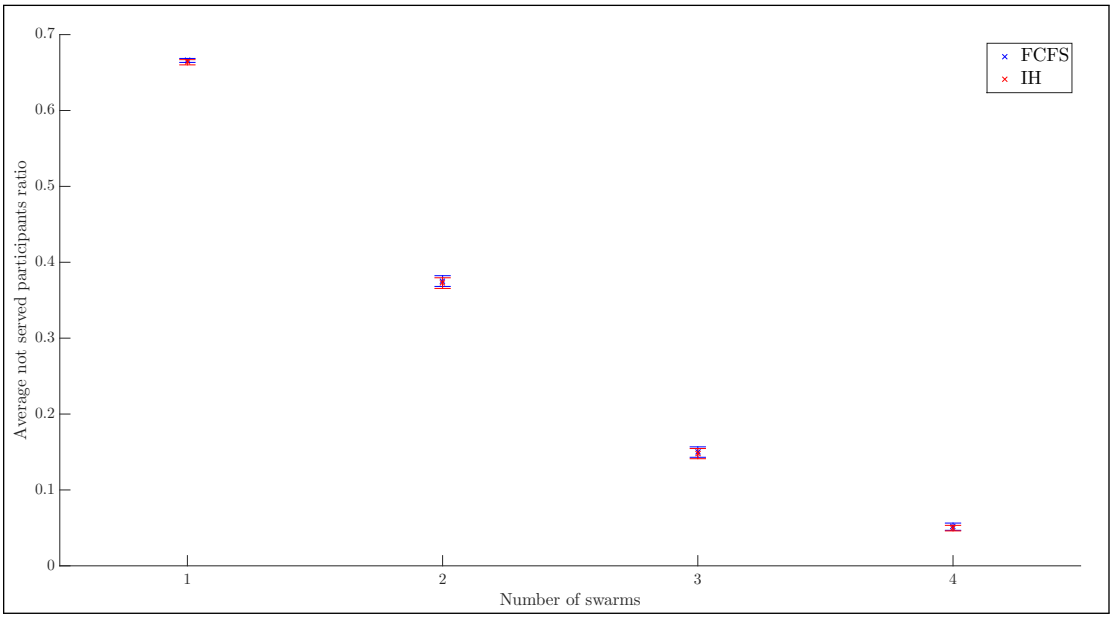


(b) Average not served participant ratio.

Figure 4.5: Graphical results for *moderate* load simulations with both the scheduling algorithms, varying the number of swarms. In the graphs are shown average values and 95% confidence intervals.



(a) Average waiting time.



(b) Average not served participant ratio.

Figure 4.6: Graphical results for *heavy* load simulations with both the scheduling algorithms, varying the number of swarms. In the graphs are shown average values and 95% confidence intervals.

Chapter 5

Conclusions

The key goal of this research was the merging of two technologies, UAV and crowd sensing, and the investigation of security-related problems affecting the latter, through the employment of UAVs swarms. Specifically, we focused on the participant position verification.

In a simulated environment that recreates an urban area, we inserted drones and participants reproducing their movements, speeds, and other peculiar features. We introduced two criteria for the attack identification, called Dictatorship and Majority, and after statistical tests imposing the false positive probability equal to 0.01, we can affirm that Dictatorship performs better than Majority, whatever is the number of UAVs composing a swarm. Moreover, the former criterion showed an average false negative percentage lower than the 1% even with the employment of the minimum number of UAVs per swarm, that was 3. From this investigation, we can affirm that swarms composed by 3 UAVs and using the Dictatorship rule are adequate to reveal more than the 99% of attacks. In addition, the lower the number of UAVs within a swarm, the more effective the costs optimization.

The following question we analyzed, regarded the shortening of participants' average waiting time. Two scheduling algorithms, an FCFS-like routine and an Insertion heuristic, were developed and compared to find out which one performs better. We tested them under increasing load conditions to determine which algorithm achieves preferable results, but they did not show statistically significant differences even with the highest load, although not realistic since catastrophic events are unlikely to happen every hour and a half.

Hence, we can affirm that our system does not significantly benefit from scheduling algorithm choice. Actually, we should prefer the FCFS-like algorithm because it is deterministic. Using Insertion heuristic, neither we can assert, nor communicate to the participant the amount of time to wait for the swarm arrival since other participants can be scheduled and can overtake him in the service list. On the other side, in the alternative algorithm, each swarm never changes its service list order, so a participant can be notified with an estimation of its waiting time and this would be, indeed, a remarkable feature.

Bibliography

[1] Teal Group Corporation. Teal Group Predicts Worldwide UAV Production Will Total $93 Billion in Its 2015 UAV Market Profile and Forecast. `http://www.tealgroup.com/index.php/teal-group-news-media/item/press-release-uav-production-will-total-93-billion`, 2015.

[2] AeroVironment, Inc. RQ-11B Raven specifications. `https://www.avinc.com/uas/view/raven`, 2016.

[3] B. Guo, Z. Yu, X. Zhou, and D. Zhang. From participatory sensing to mobile crowd sensing. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 593–598, March 2014.

[4] Sunyoung Kim, Christine Robson, Thomas Zimmerman, Jeffrey Pierce, and Eben M. Haber. Creek watch: Pairing usefulness and usability for successful citizen science. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2125–2134, New York, NY, USA, 2011. ACM.

[5] Prashanth Mohan, Venkata N. Padmanabhan, and Ramachandran Ramjee. Nericell: Rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 323–336, New York, NY, USA, 2008. ACM.

[6] Kasper Rasmussen, Mani Srivastava, et al. Secure location verification with hidden and mobile base stations. *IEEE Transactions on Mobile Computing*, 7(4):470–483, 2008.

[7] Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

[8] Marco Rasori. Binomial: a netlogo extension for binomial distributions. `https://github.com/Daven00/Binomial-NetLogo-Extension`, 2016.

[9] Suresh Nanda Kumar and Ramasamy Panneerselvam. A survey on the vehicle routing problem and its variants. *Intelligent Information Management*, 4(3):66, 2012.

[10] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.