POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Coverage Planning for Multiple Unmanned Aerial Vehicles with Guaranteed Deadlines

Supervisors

Prof. Massimo CANALE

Candidate

Lucia CORDESCHI

October 2020

Summary

The present work will present possible navigation algorithms for a fleet of UAVs inside a known region.

After a series of consideration about the requirements necessary to achieve a satisfactory result, like for example the complete coverage and the warranty of covering the spots when their priority overcomes a certain value, a set of algorithms able to satisfy the requirements will be developed. In the following will be considered both centralized and decentralized algorithms.

The work flow starts from a centralized approach, that will base its functioning on the search of the maximum value of the priority function at each step, in such a way to assure the visiting of them in the minor possible time. Since in this way the high priority spots in the vehicles neighborhood are considered only if one of them has in absolute the highest priority inside the region, creating an unwanted zig-zag trend and overpositions, two solutions will be proposed: the first is simply not consider just the greater value but a set of the high values (for example the n highest priorities) and among them choose the closest to the vehicle as next destination of the navigation. The second solution is to change entirely approach and to search for each step of the vehicle the position that minimizes a certain function that would take into account both the global state of the region and the local one, through a MPC-based algorithm.

Considering the demands of a real-life application and the impossibility to have a constant connection, a decentralized algorithm will be developed. The region will be divided for this purpose in sub-regions of interest and a meeting will be imposed every period of time, in such a way to perform an update of the information. The navigation inside the respective sub-region in the time instants between the two meeting will be performable with the preferred navigation method, that in this case will be the maximum search algorithm described above.

Acknowledgements

There are so many people to include here that I really don't know where to start. So Daniele, Federica, Stefano, Alessandra, Sara, my parents and my sister and all the relatives around me will forgive me, but I will never be good at this.
So the only thing I feel to share here is:
To people who was with me and to the ones who will remain.

LC

Table of Contents

List of Figures

List of Algorithms

Acronyms

UAV
: Unmanned Aerial Vehicle

MPC
: Model Predictive Control

TSP
: Travel Salesman Problem

Chapter 1

Introduction

The final purpose of the present work is to create a reliable motion planning algorithm for a fleet of Unmanned Aerial Vehicles (UAVs) that performs the monitoring of a known area, inside which different levels of visiting priority are present, with the ultimate purpose to detect fires.
This goal creates a series of preliminary objectives to guarantee:

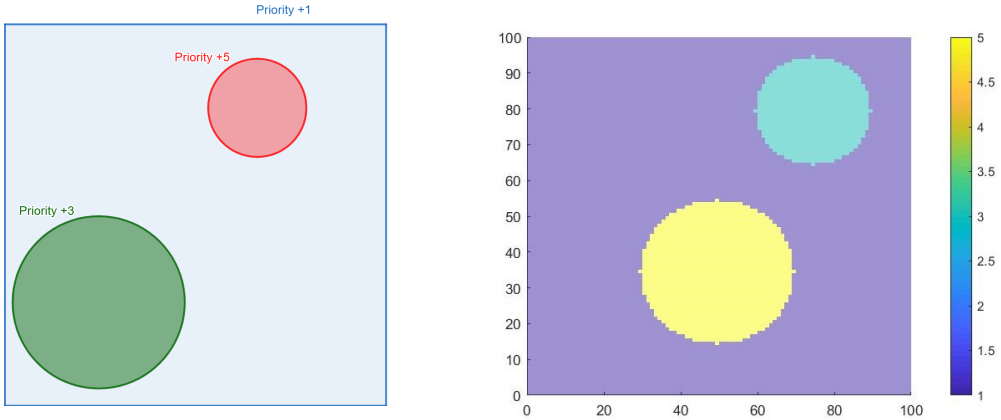
- It is strongly intuitive that a fire individuated in a early phase is easier to extinguish, so to create an efficient and actually useful motion control methodology it is necessary to impose a deadline on the maximum reachable priority value of each point.

- The whole considered region has to be visited, single spots cannot be left without visit.

- The algorithm has to be able to modify instantaneously the behaviour of the vehicle as soon as it gets to know the change in the priority distribution inside the region. This is the main difference between the current approach and the already developed coverage algorithms, that refer to uniform areas for which a predefined path is the most efficient.
  This necessity raises from the inquiry [1]'s results that underlines as 85% of the wildfires are caused by humans, so a wooden region where human presence has been detected (for example a camping) will be more likely to have a starting fire and it should be monitored more often. Since the presence of human, or of other risk factors, may change over the time, it is possible to have the necessity to modify the priority of the points.

Addressing the above goals requires to have a reliable automatic methodology to choose the next position, whichever would be the current states of the region and of the UAVs.

To have a better understanding of what is happening, from a priority point of view, in the evolution of the monitoring it is useful to make a clarification: the initial state of the priority map is uncorrelated with the evolution in time of the priority map future state. Once a point is monitored, in fact, it is reset too, and if the update function of the priority is homogeneous the vehicle will tend to observe the points at regular intervals. In this application this behaviour is unwanted and eliminates the adaptability requirement, so a non-uniform update has to be applied. An example of the described updated priority distribution and of an update matrix in Matlab are represented in 1.1.



(a) Non-uniform priority update

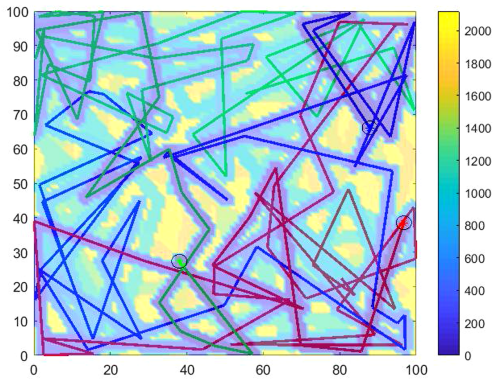
(b) Update matrix generated in Matlab

Figure 1.1: Non-uniform update applied in Matlab

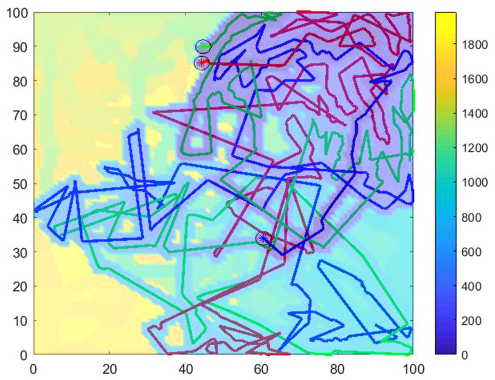
In this perspective, at first an algorithm that follows the policy of closest maximum reaching has been applied. The obtained motion is not optimal since the successive points are reached travelling along a straight line generating the necessity to overlap a recently seen point to cover some eventual spots left unseen before. The described tendency improves with time but, as can be observed in 1.2 also after more than 130.000 iterations (assuming that 1 iteration corresponds to 1 second it would be equivalent to more than 36 hours) the number of overlapping is huge in the central area of the region, creating an inefficient behaviour.

To overcome this problem it is possible to extend the list of the searched point, trying not considering only the points with absolute maximum priority but including the k greater values and imposing the closest as next destination. This approach limits a bit the overlapping and zig-zag problems but only in a partial way. Moreover increasing too much the value of k the adaptability of the algorithm is reduced, getting with the limit of k equal to the number of points of the region the behaviour in 1.3

The proposed algorithm chooses on-line a destination to reach inside the region and



(a) Closest Maximum Search after 2950 iterations



(b) Closest Maximum Search after 132950 iterations

Figure 1.2: Closest Maximum Search Behaviour

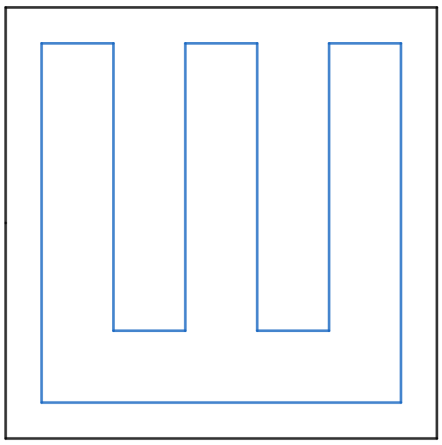


Figure 1.3: Boustrophedon path

once reached it computes the next one and so on, connecting the set of successive destinations with a sequence of straight lines. As explained above this behaviour often left behind unseen spots really close to the followed trajectory due to the shape of the motion, creating the necessity to come back through an already observed territory to check a single spot. A possible solution for this problem is not to reach the next destination through a straight line motion but through a broken line. In this case to be chosen are the single next steps, in such a way to create a generic path that conduct to a high priority spot, covering at the same time the more urgent spots that are in the neighborhood of the travelled path. A comparison between the two path is illustrated in 1.4.

To obtain the Broken Line path an optimization algorithm, based on a MPC, is

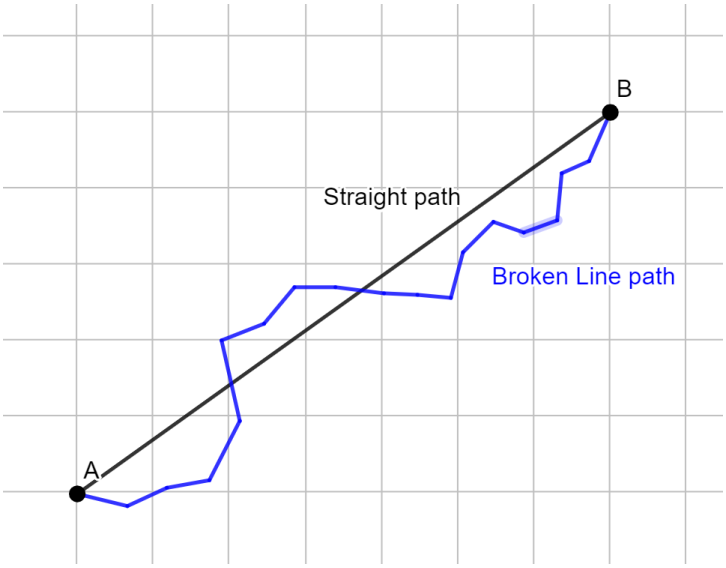


Figure 1.4: Possible paths to reach A from B

applied with the purpose to get closer to the higher priority points covering during the journey as much urgent spots as possible.
The described reasoning is translated into a series of algorithms that have been used to verify the validity of the considered solutions, which can be divided into two parts:

1. Maxima Search Algorithm
   The Maxima Search Algorithm chooses as next vehicle's destination the closest among the points with a greatest values of the α, the point-wise priority function. In this way it is assured the coverage of the highest priority spots. Once developed the basic algorithm the choice for the next destination has been extended to the points with the k greater α values to assure not to leave behind points with a priority high but inferior to the absolute maximum. After the creation of a reliable algorithm for navigation it is applied to a Decentralized case, so to a case where the vehicles have not the information on what the other vehicles are doing at any moment, but to obtain that they have to meet with the other UAVs. Please note that in this case the happening of a *meeting* does not indicate to have all the vehicles at the same location, but that they have a reciprocate distance minor than the communication radius, ρ_c.

2. MPC-based Algorithm
   The MPC-based algorithm is based on the optimization of an *optimization*

function (or objective function) composed by two terms: a global one that assures to have a complexive trajectory that brings the vehicles closer to the maximum and a local one that makes the vehicles cover the high spots along the path. During the implementation process an amount of different optimization functions and optimization variables are considered and a trade-off among accuracy, efficiency and computational cost is necessary. The development in the present work is entirely realized inside the Matlab environment through the use of the function *fmincon*.

A compact representation of the developed algorithms can be found in 1.5.

In the following it is presented an overview about the studies related to the present one and the list of used variables, to make simpler the understanding of the pseudocodes and the reasoning. In Chapter 2 is reported a useful set of theoretical notions used directly or indirectly in this work. In Chapter 3 is present the description of the development procedure of the final algorithms and the algorithms themselves. Finally in Chapter 4 the results of the simulations and a comment to them can be found. In the Conclusion a short section about the future development of this work is placed.

1.0.1 Related Work

The realization of the present work has required the combination of a series of widely spread concepts, among which the main ones are coverage, communication and vehicle routing, with a particular focus on a DTSP(Dynamic Travel Salesman Problem) since the monitored region where vehicles are moving is modeled as a discrete grid. Despite the huge amount of studies none of them can be applied exactly at the present case, even if it is undeniable their usefulness both as background and for an adapted solution of specifics problems.

The faced coverage methods of a give region are analyzed in a complete way in [2] where the different methodologies are cataloged on the basis of both the region's structure and the information about the area to cover. Since in this case a 2-D continuous region without obstacles is considered a particular attention has been given to the boustrodephon path, represented in 1.3, being this the best solution to cover a uniform region, also in case of a non-regular region, as shown in [3].

In case in the coverage are involved more than one vehicle also the exchange of information on the already seen spots is an important element to take into account, so in order to increase the efficiency of the algorithm the coverage methods have to be combined with a connectivity among the sensors, the connection can be either maintained or guaranteed over the time, so after a period when the vehicles cannot exchange information they meet again and update their state. Works that analyze

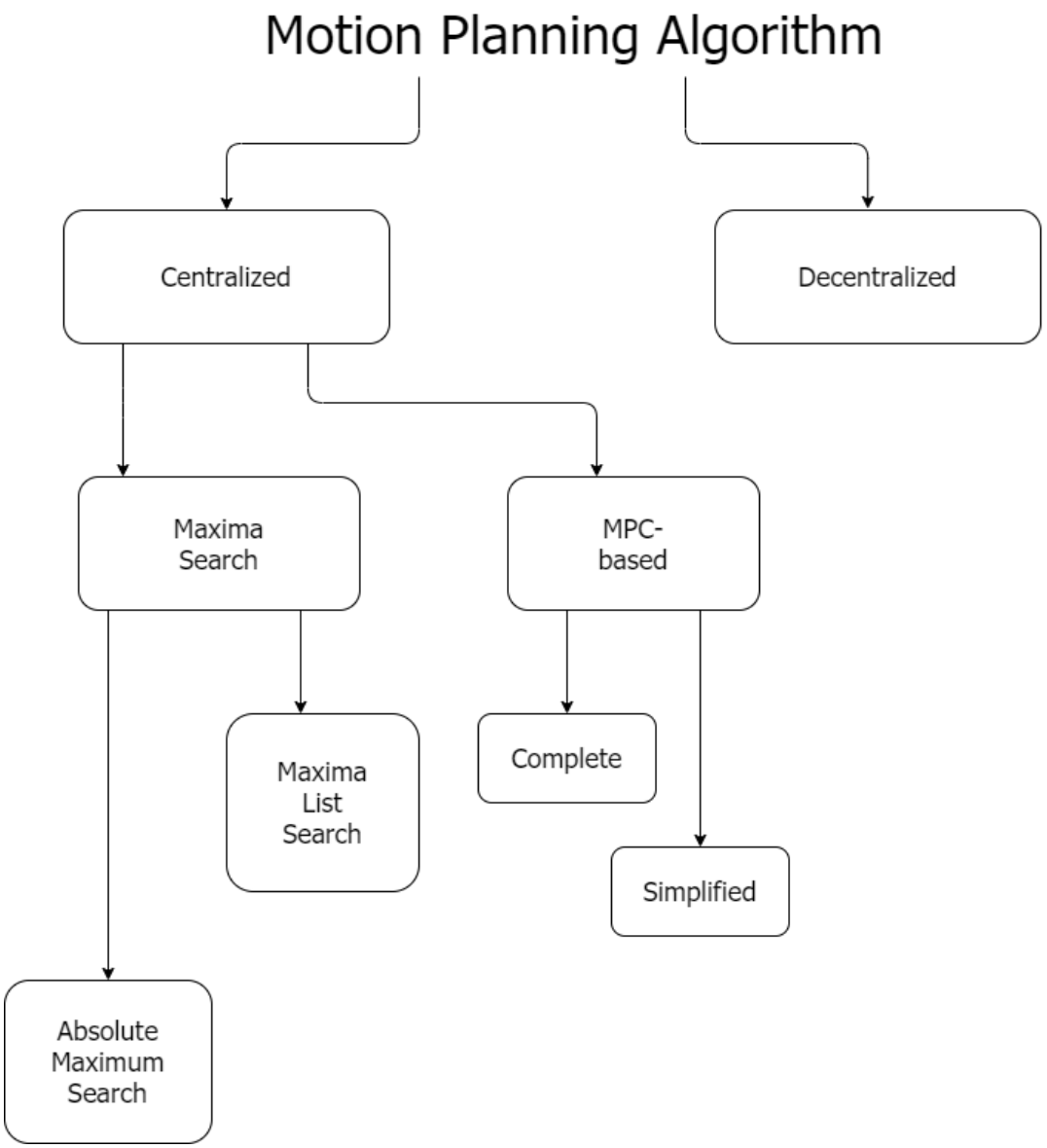


Figure 1.5: Development of the algorithms

the relation between coverage and connectivity in a sensor network usually refers to a static network or to a semi-dynamic network, in the sense that the nodes that compose the network itself have the ability to adapt to changes, in the sense that in case of damage of a part of the network, it is able to recreate a complete coverage of the considered area, different possibilities are in [4] and a particular application can be found in [5]. On the other hand considering a group or fleet of mobile robots that

have to explore an area without guaranteeing at any time instant the converage of the whole area you can find mainly the approach to move the fleet together in such a way to cover a greater area without losing connectivity at all, as in [6]. In [7] the solution is still different since the exploration of an unknown area happens imposing that two connected vehicles separate their paths to overcome an obstacle and then meet again to exchange the gained information, this meeting concept has been adapted and applied in the realization of the decentralized algorithm.
A vehicle routing problem is generally based on the Travel Salesman Problem (TSP) that wants to find the more efficient way to cover a set of position, with a different cost required to cover each of them, the first important result in an euclidean region is the Bertsimas' one in [8]. Starting from the Bertsimas work an amount of specif studies have been published covering specific requirements, as the Travel Repairman Problem (TRP), where to the reaching time for a given position is added the time to perform a specific task in that position, and the dynamic version of TSP and TRP, so respectively the analysis of DTSP and DTRP, [9],[10],[11]. A complete survey on the different developed vehicles routings is [12]. Another class of problems not directly exploited in the present study is the Dubins vehicles navigation, where the solutions for the vehicle routing and the coverage problems are combined with the constraints on the possible motions of the sensors (like the lower bound on the curvature radius) is such a way to have an acceptable model of the motion of an aerial vehicle, [13] and [14].

1.0.2 List of variables

Here they are collected the variables used in the present work.

- n: Dimension of the region.
- E: Extension of the region in the linear direction. Since the region is assumed to be square it is the same for all the dimensions.
- v: Number of vehicles.
- ρ_v: Vision radius of the vehicles. Constant.
- δ: Spatial discretization step. Constant.
- Δt: Temporal discretization step. Constant.
- u: Velocity. Constant.
- $Pos_{t,i}$: Position of the ith vehicle at the time t.

- α: Pointwise priority function.
- A: Matrix of priority function in a discrete n-D region.
- $Dest$: Desired destination of the vehicle to reach.
- dir_u: Direction of the velocity.
- d_t: Distance of [x,y] and Pos_t.
- h: Prediction horizon for the MPC algorith.
- V_i: Boolean matrix with same dimension of A. Element = 1, that point belong to the Voronoi region of the ith vehicle.
- s: Dimension of the grid gotten discretizing the region. $s = \frac{E}{\delta} + 1$.
- $Bound_{-/+}$: Inferior/superior bound of vision interval.
- k: Number of maxima considered as next target.
- $kmax$: Vector containing the locations of the points with the $k - th$ higher priorities.
- Th: A value threshold. If $\max A$ is greater its position is the next $Dest$.
- ρ_p: Radius of the circle containing the observable points during the prediction.
- Lim_{Dim}: Limit dimension of the Voronoi region to have the correspondent vehicle searching for the next position inside it.
- w_i: Weight of the $i - th$ region.
- U_{matrix}: Priority update matrix.

Chapter 2

Theoretical Background

In this chapter, some background concepts useful for the understanding of the study are summarized. In particular. the topics are either used directly in the algorithms development, such as the Lloyd algorithm, or represent a starting point for the navigation planning with time constraints in a dynamic environment, such as the Dynamic Travel Salesman Problem.

2.1 Lloyd's Algorithm

One of the most spread method to organize the motion of elements inside a quantized region is undoubtedly the Lloyd's Algorithm, [15]. In this algorithm S.P. Lloyd divides the considered region in n convex sub-regions. Roughly speaking if we consider an area with n sensors the algorithm creates n regions, each one containing a sensors, and then the sensors are sent toward the centroid of the respective region and both, regions and centroids, are recomputed. It is demonstrated that this algorithm leads to a convergence of the sensors' positions.
More formally the Lloyd Algorithm con be divided into three phases:

1. Voronoi Region Computation.
   The first step is the subdivision of the region in n Voronoi regions, $V \in \mathbb{R}$, it is based on the distance between points and sensors. In general the distance can be computed in different ways, in the present case it is used the Euclidean distance.
   The Euclidean distance for two two-elements vectors $a = [a_1, a_2]$ and $b = [b_1, b_2]$

$$\|a - b\| = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} \tag{2.1}$$

   The rule to perform the division in the n regions is:

$$V_i = q \in Q : \|q - p_i\| \leq \|q - p_j\|, \forall j \neq i \tag{2.2}$$

Where:

- V_i: i-th region;
- q: point of the discretized region Q;
- p_i: position of the i-th sensor;
- p_j: position of the j-th sensor;

2. Centroid Computation.
Once the regions are formed their centroid is computed. The generalized centroid, or center of mass, C_{V_i} of a region V_i is defined:

$$C_{V_i} = \frac{1}{M_{V_i}} \int_{V_i} q\rho(q)dq \tag{2.3}$$

Where:

- $\rho(q)$: mass density function;
- M_V: generalized mass of the region computed as

$$M_V = \int_V \rho(q)dq \tag{2.4}$$

It is straightforward that in a discrete case the integrals becomes a sum, so the previous equations (3.3 and 3.4) become respectively:

$$C_V = \frac{1}{M_V} \sum_V q\rho(q)dq \tag{2.5}$$

$$M_V = \sum_V \rho(q)dq \tag{2.6}$$

3. Motion Toward Centroid.
The last step is the computation of the new position of the sensor according to a chosen motion law. The most immediate control law is obtained but the position-velocity kinematic equation:

$$p_{i,t+1} = p_{i,t} + \frac{C_V - p_{i,t}}{\|C_V - p_{i,t}\|} u\Delta t \tag{2.7}$$

Where:

- $p_{i,t}$: position of the i-th sensor at the time t;
- u: absolute value of the velocity of the sensor;
- Δt: length of the time interval;

This procedure is ideally repeated until $p_i = C_{V_i}$. In case of a real application the condition becomes $|p_i - C_{V_i}| \leq \epsilon$, with ϵ a small number. In this way the motion of the sensors is directed toward the most representative spot of the region and the above procedure assures the convergence.

2.2 Model Predictive Control - MPC

A Model Predictive Control is a method for the control of a system. It is based on the optimization of a function (*objective function* or *cost function*) over a certain horizon of time h. The ultimate objective is to choose the control action (*optimization variable* value) to apply at the time $t+1$, predicting the behaviour of the system between the current time t and the instant $t+h$ and finding the next value of the control variable that assures the minimum of the objective function over the prediction horizon.
The objective function can have different characteristics, according the specific requirements of the solver used to resolve the minimization problem, but the more general MPC does not require a particular structure, in other words the cost function can be either linear or non-linear. Nevertheless it is essential to guarantee the function dependence from the temporal sequence of its values, in such a way to be able to exploit its connection with the behaviour of the system over the chosen prediction horizon h. Generally, to assure this relationship, the objective function is the sum of the values assumed by a determined function of the optimization variable at each time t or the average of the function's values over the prediction horizon.
More formally a MPC can be described as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{h}\sum_{i=0}^{h-1} f(x(t+i|t), y(t+i|t)) \\ \text{subject to} \quad & x_{min} \leq x(t+i|t) \leq x_{max} \\ & y_{(t+i)} = g(y(t+i-1)) \end{aligned} \tag{2.8}$$

where:

- f indicates the optimization function (or cost function);
- g indicates the dynamical equations of the system;
- x indicates the optimization variable, in the present case also the control variable;
- x_{min} and x_{max} indicates respectively the minimum and maximum allowed value for the control variable;

- y indicates the state of the system, that has to be updated at each time instant following the dynamical equations of the system;
- h indicates the prediction horizon;

2.3 Connectivity, Graph Theory and Coverage

Approaching the decentralized control problem for a fleet of mobile robot the problem to guarantee connectivity at least over a period of time. The concept of connectivity is strongly related to the graph theory and in this case it has to be analyzed also in the light of coverage, so it is useful to underline some key-concepts. A graph is a deterministic and schematic method to represent uniquely any network. Reporting the definition given in *Graph Theory* [16]:

> A *graph* G is an ordered pair $(V(G), E(G))$ consisting of a set $V(G)$ of *vertices* and a set $E(G)$ of *edges*, together with an *incidence function* ψ_G that associates with each edge of G an unordered pair of vertices of G. The relation between the edge e and the vertices u and v *joint* by it. It is expressed as:
>
> $$\psi_G(e) = \{u, v\} \tag{2.9}$$

Sometimes it is possible that the complete description of the system requires additional information, as a weight w associated to each edge e. This is for example the case when to study the motion in a network it is necessary to know the travelling time between two nodes or the required energy to reach it. The graph gotten is called *weighted graph* and it is denoted as (G, w).
Graphs are widely used in information theory because they allow to represent information networks in a intuitive way and their properties have been deeply analyzed and studied, so also the problem of connectivity in mobile sensors networks has been addressed multiple times.
Trivially two vertices (in the following called also sensors or nodes) are connected if they can communicate, so if their distance is inferior than a communication radius ρ_c linked to the sensors' characteristics. According to graph theory two vertices, representing the sensors, are connected if they are joint from an edge e.
In a monitoring network is it essential to guarantee the exchange of information between the node of the graph, but, to assure it, it is not necessary to have a full connection among the vertices, or, in other words, it is not crucial to guarantee that at each time every vertex can communicate with all the other vertices. The kind of property to assure to guarantee connectivity, according to the $path - definition$ is that whichever couple of nodes N_a and N_b belonging to the network there exists at

least one path that travels along the edges and connects N_a and N_b.
The assurance of connectivity can be achieved through one of the following, according to the network structure:

- The network is connected at every moment.
  This condition can be realized in static network, or in dynamic networks where either only a limited motion of the nodes is allowed or the number of nodes is great, with a suitable disposition of the agents inside the considered region.

- The connection of the network is guaranteed over a period of time.
  It is particularly useful in the discussed case because in case of mobile agents over a region to monitor it may not be possible to cover the whole area in any instant. As consequence it is necessary both to assure an efficient coverage planning to respect deadline and specific region characteristics and to assure connectivity to guarantee the exchange of information between the agents.

In the current analysis the second methodology is the most convenient. In fact in general the considered case consists of a number of vehicles with a total covered area at each instant greatly minor than the total surface to monitor. It creates the necessity for the vehicles to move around the area independently but exchanging information often enough to improve the overall results.
In a centralized algorithm the nature of the algorithm itself provides that at each moment all the sensors have all the updated information about the region, the priority function and the positions of the other member of the fleet, while in a decentralized algorithm this is not true. The problem can be overcome in different ways for example choosing a number of vehicles big enough, imposing that the vehicles during the motion maintain a distance smaller than ρ_c or,like in this case, imposing a meeting where the connectivity of the whole network is restored every Δt, in such a way to guarantee updated information.

2.4 Travel Salesman Problem and variations

2.4.1 Algorithm complexity

The complexity of an algorithm is a measure of the basic computational steps required for its execution. Usually a low-complexity algorithm is preferred, so when it is possible an algorithm is wanted to be optimized, or in other words the computational time wants to be minimized.
Since the computation of the complexity of an algorithm is not always immediate, it has been created a distinction in classes in such a way to define an upper bound

to the required steps number. The class that contains the less complex polynomial-time algorithm is said *P-class*. Its upper bound is a polynomial function and according to the polynomial order is possible to distinguish as linear, quadratic, cubic and so on. This class is deterministic, in the sense that being the solution given by a sequence of polynomial computation, it will be necessarily the expected one, if the algorithm is correct.
The other big class is the *NP-class*, non-deterministic polynomial time. The non-deterministic factor is generated by a required guessing inside the algorithm. The guessed element creates the necessity to check the correctness of the gotten solution. It is demonstrated that is a single *NP-class* problem con be reconducted to a *P-class* one, then all *NP* problems are *P* problems, but until now it has not been found this correspondence. The initial passage from *NP* to *P* is wanted to be a polynomial reduction, so the first algorithm can be converted into the second one. If it exists then the solution for the first one can be converted into a solution for the second algorithm.
In the context of *NP-class* problems, they have to be defined also the concepts of *NPC* (Non-deterministic Polynomial-time Complete) and *NP-hard class* problems. A problem P belongs to *NPC-class* if it both belongs to *NP-class* and $\forall P' \in NP \, \exists$ *polynomial reduction* $Pr : Pr(P') = P$. While *NP-hard* problems are by definition problems at least as hard as any problem in *NP-class*. One of the most spread *NP-hard* problem is the TSP, Travel Salesman Problem.

2.4.2 Travel Salesman Problem definition and Variations

A widely studied problem that searches for the most efficient navigation inside a weighted graph, with the requirement to visit each node once in the minor possible time and then return back at the starting point, is the Travel Salesman Problem (TSP). It is at least a $NP - hard$ problem and its possible global optimal solution has been investigated many times, but the dependence from the network structure makes difficult to find a general solving.
A particular typology of TSP are the Dynamical Travel Salesman Problem (DTSP) where the graph representing the problem changes through the time.
In case the requirement of the problem is not only to pass through a specific node but also to perform a certain kind of task that requires a determined time, a Travel Repairman Problem is faced, existing in both the static and Dynamic version.

Chapter 3

Coverage Planning Algorithm Development and Implementation

The present chapter contains the preliminary analysis of the problem and the work-flow in the creation of motion planning algorithms. The reasoning and the development are described and then the pseudo-codes are reported.

3.1 Problem Definition and Issues

The final purpose of this work is to create a base to solve the problem of monitoring a known region with a non-uniform visiting priority through an autonomous motion planning applied to a fleet of UAVs.
Before to analyze in details the behaviour of a vehicle inside the region or the considered algorithms it is important to understand why the update priority distribution has a greater relevance than the current region priority state and, as consequence, than the initial priority distribution inside the region.
Trivially considering two points if the first has an update rate equal to +1 and the other one an update rate equal to +3, it means that the second spot has to be visited three times more often than the first one and since the value of the point is set to zero once it is visited, this is true whichever is the initial value of the two points. From this example the importance of the update with respect to the current state is clear and it can be deduced considering the reset of the priority value in a specific point once it is seen from the vehicle. After the visit the point priority function will depend on the update rate and on the trajectory of the vehicle and

not on its starting value.
The correctness of this deduction is then confirmed by the simulations where the same steady behaviour is obtained.
The classical coverage problem approach in 1.3 creates the best motion in case all the points of the region have same priority, but in case the urgency of a part of the area is different the performance strongly degrades, as it is observable from the comparison between 3.1 and 3.2, where the non-uniform update is the one in Figure 1.1b.

The performance worsening shows the necessity of adaptability of the algorithm

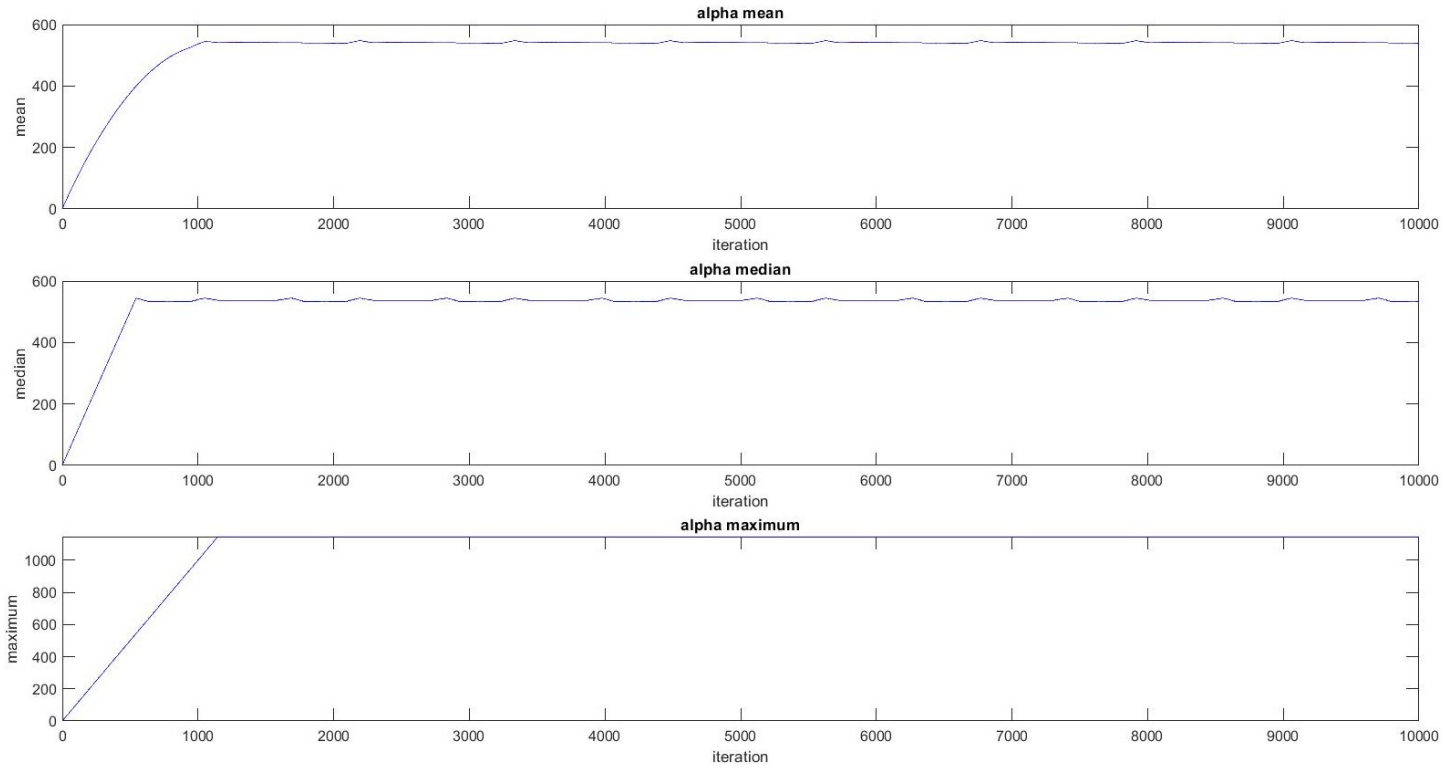


Figure 3.1: Performance of limit algorithm with uniform update

in order to satisfy the warranty of coverage within a given deadline or equivalently a visiting frequency proportional to update rate. The analysis of this objective points out the necessity to define a way to describe the priority state of the region at each instant of time. So in each instant it is necessary to know how much time spent since when a particular spot has been visited and which is the urgency to visit that point. In case of uniform update of the priority these two requirements coincide, but in general it is not true, in fact if one area of the main region has a

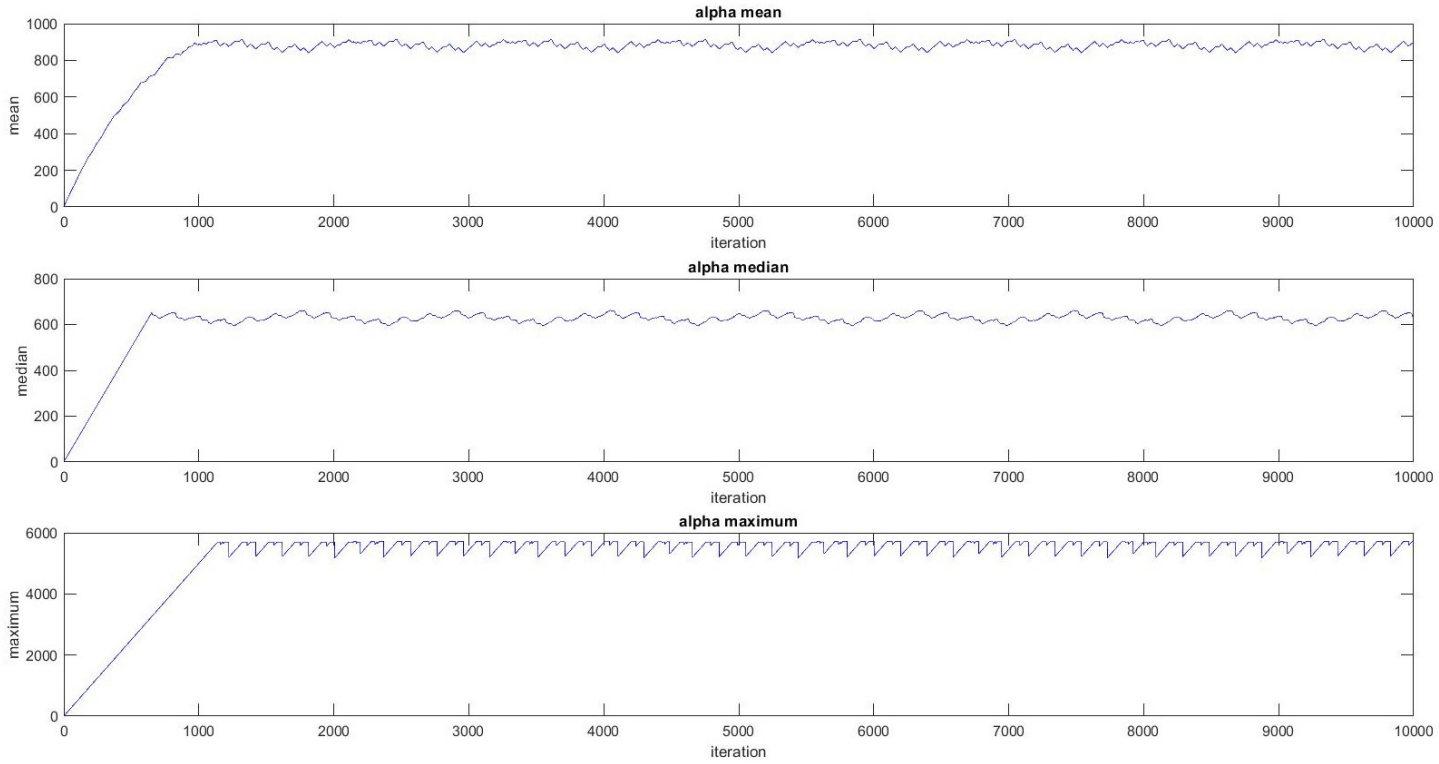


Figure 3.2: Performance with a non-uniform update

priority which increases in a different rate that the rest of the region's points the correspondence between time and priority decays. The indirect relation generated in this case has to be taken into account not only for the description of the region state, but also in assigning a unit of measurement to the priority values. Since it is not possible to choose the seconds [s] the choice is to define the priority as an dimensionless parameter.
In order to define a way to outline the visiting urgency inside the region the first step is to discretize the considered region, in such a way to describe it as a set of equidistant point, each describable through a set of coordinates. Then it is possible to associate to each point specific functions to take into account the priority to visit it again and the time spent since when it has been visited. In motion planning greater is the priority function value, greater is the urgency to visit the point again. The adaptability requirement makes unsuitable the classical coverage method in 1.3 nevertheless, being demonstrated that it is the best possible behavior in a uniform update, it is a good comparison to test the "*goodness*" of an algorithm in case the simulation is properly initialized and the update of A is non-uniform over the

region.
The development was an incremental process, so in the realized model there was a gradual increasing of the difficulty. Initially a 1-D, 1 vehicle, model was developed then a second vehicle was added. In this simple setting the most suitable navigation rule was searched for, starting from a Lloyd's algorithm, that showed to be ineffective due to the vehicles convergence. So a Maxima Search algorithm was implemented. After that the environment was extended to a 2-D square one. Again the first simulation was realized with a single vehicle and then other was added. In this second environment the problem of realizing a "*good*" coverage appears and brings the discussion to the possibility to develop a different motion planning control.

3.1.1 How to judge a "good" Coverage?

The "*goodness*" of whichever algorithm depends mainly on its purpose, focusing on a coverage path planning algorithm with guaranteed deadlines it can be decomposed in two overlapped problems with different requirements: a spatial one and a temporal one.
In the case of a generic coverage problem without any temporal constraint the only goal is to cover completely the target region without leaving unseen spots behind. The quality of the algorithm can be linked to the length of the total path traveled or to the energy consumption. A good example of application of this kind of problem is an automatic floor-cleaner. In this case the most important thing is to cover the entire room with the minor possible energy consumption, the time of usage or the total trajectory length are the only important parameters to minimize. The temporal requirement is, for its own nature, linked to a certain kind of urgency like a maximum time imposed to cover the whole area or like the case the coverage has to be completed more consecutive times, as the monitoring purpose presented in this work. In the case of a uniform updated visiting priority the best behaviour is the boustrophedon path since it is demonstrated to minimize the navigation time required to cover each point of the region. On the other hand in a dynamic context, so in a case when it is necessary to visit some points more often than others, its effectiveness decreases.
Putting together these demands the conditions to take into account to give a definition of "*goodness*" are:

— Completeness of the coverage, that consists of the avoidance to leave unseen spots of interest behind.

— Warranty of the presence of a limit on the maximum time spent between two successive visits in a specific point, hence the respect of a deadline.

To efficiently respect the above requirements an intuitive condition can be made out: to get a trajectory that guaranteeing the complete monitoring of the region avoiding as much as possible the overlaps. The last characteristic arises from the fact that an overlapping enlarges the time required to visit the current maximum since part of the time elapsed between two successive visits is spent to cover again a spot recently seen.
The first and most important mathematical parameter to consider to evaluate the performances of the planning algorithm with temporal constraints is the maximum value of the priority function over the region, $\max(\alpha)$, in such a way to check the presence of an upper bound, indicating the limit among two successive visits. A deadline can be either imposed explicitly or included implicitly in the used algorithm, but whichever choice is made it has to take into account the eventual modifications of the update rate in some subareas and the topography of the region, modifying its value accordingly.
Together with the maximum it is necessary to have some information about the general state of the region and to obtain this information it is useful to look at the average and at the median of the priority function over the points of the region, respectively $\overline{\alpha}$ and $\tilde{\alpha}$. They have generally an oscillating trend, with peaks correspondent to the overlapping.
The combination of them gives a good information on the behaviour of the sensors inside the region, but it corresponds unequivocally to the temporal trend only if the update is uniform, so if after 1 s the priority of all the points of the region increases of the same value. Otherwise it is necessary to create a variable to keep track of the time elapsed between two successive visits. It would not correspond anymore to the $\max(\alpha)$ and to find the maximum time elapsed starting from the maximum value of priority would not be trivial, hence it is useful to add an index of the time spent from the visit of at each point of the region in every instant and keep track of its maximum.
In case $\overline{\alpha}$ and $\tilde{\alpha}$ are close to the maximum it means that the path presents an amount of overlaps or that cover only a small part of the whole area, on the contrary if there are a big distance between the values at least one point in the region is visited fewer than the others. If the values to be distant are $\overline{\alpha}$ and $\tilde{\alpha}$ it means that there are two regions with values deeply different and it can be assumed that a part of the region is extensively visited while the other is almost not seen at all. So they are a good source of information on the level of coverage of the area; basically they answer to the questions: are all the points covered? Is the coverage completed in the required time for all the points? Is given to the whole region the same importance (in case of uniform update of α) or an importance proportional to its own priority (otherwise)?
The previous questions are the ones implicitly asked through the requirements enlisted above. These three elements give a pretty complete overview about the

planning behaviour, in case of some unexpected problems or strange trend in the α's performances can be useful to have a representation of the coverage process with a movie showing the successive state of the vehicles, with appearence similar to the frames represented in 1.2. Outside a debugging context the four described parameters give a complete idea about the algorithm performances, its respectfulness of the deadlines and the quality of the path automatically generated by the algorithm, and consequently representing a good yardstick of the quality of the algorithm.

3.2 1-D Implementation

In the 1-D implementation the validity of the Lloyd's algorithm and of the Maxima Search algorithm is tested. The Optimization algorithm is not considered in this phase because at each step the vehicle can choose only between two possible directions and necessarily covers all the points placed between its current position and its destination. In the case of 1-D implementation the considered A matrix update is only the uniform one.

3.2.1 Lloyd's Algorithm

The procedure repeated at each iteration in the realization of the Lloyd's-based algorithm, represented in the Algorithm 1, is the following: at the beginning of each iteration the priority matrix is updated as $A = A + 1$,then, given the current position of the vehicles, the points covered by them are computed considering the interval $[Pos_{t,i} - \rho_v, Pos_{t,i} + \rho_v]$ and then setting to 0 the α of the points belonging to the interval.
Once updated the configuration the classical Lloyd's algorithm is applied, computing Voronoi regions, finding the centroid and then updating the positions. With the current initialization this is not necessary to consider the motion of the vehicle in updating A since ρ_v is greater than the maximum step travelled in each iteration. In other words no spot is left unseen, considering only the sequence of positions without the motion between them. In case $u > \rho_v$ the second condition of line 7 of Algorithm 1 has to be added. It considers the path traveled among the two positions and resets all the points in the middle of them.

The results of the simulation, in 3.3 it refers to 50 iterations, show that after some iterations the vehicles, both in case $v = 1$ and $v = 2$, get stuck at a intermediate position, making the algorithm useless for our purposes.

Algorithm 1: 1D Lloyd's Algorithm

while *True* **do**
  $t = t + 1$
  $A_{t+1} = A_t + update$
  for $i = 1 : v$ **do**
    $Bound_- = Pos_{t,i} - \rho_v$
    $Bound_+ = Pos_{t,i} + \rho_v$
    if $x \in [Bound_-, Bound_+] \wedge x \in [Pos_{t-1,i}, Post, i]$ **then**
      $A(x) = 0$
    end
  end
  Computation of Voronoi Regions V_i
  Computation of the centroids c_i
  $Pos_{t+1,i} = Pos_{t,i} + \frac{c_i - Pos_{t,i}}{\|c_i - Pos_{t,i}\|} u \Delta t$
end

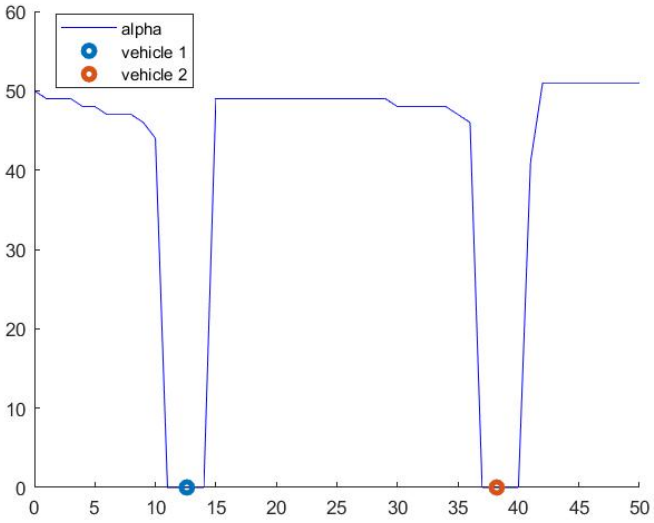


Figure 3.3: Lloyd-based navigation

3.2.2 Maximum Search Algorithm

The necessity to cover with greatest urgency the points with greater priority function brings up an alternative algorithm that follows the maximum among A values. Following this rule the only change is the choice of the destination for the vehicle to

reach. In case of a single vehicle it simply travels to reach that point, while in case of $v = 2$ at each time Voronoi regions are computed and the maxima destinations inside the respective subregions are reached.
This algorithm gives the hoped results with the vehicles that oscillate in the region

Algorithm 2: 1D Maxima Search Algorithm

```
while True do
    t = t + 1
    A_{t+1} = A_t + update
    for i = 1 : v do
        Bound_- = Pos_{t,i} - ρ_v
        Bound_+ = Pos_{t,i} + ρ_v
        if x ∈ [Bound_-, Bound_+] ∧ x ∈ [Pos_{t-1,i}, Post, i] then
            A(x) = 0
        end
    end
    Computation of Voronoi Regions V_i
    x_i = x ∈ V_i | A(x_i) = max_{x∈V_i} A(x)
    c_i = x_i
    Pos_{t+1,i} = Pos_{t,i} + ((c_i - Pos_{t,i}) / ||c_i - Pos_{t,i}||) u Δt
end
```

and cover it uniformly.

3.3 Main Obstacles to Straightforward Implementation

Even though the basic idea of the Maxima Search Algorithm developed in the 1-D case is applicable in the 2-D case the addition of the second dimension takes some issues to deal with to adapt the motion planning. In particular they are linked to the distribution of the value of A, that it is now a matrix, and its update. Considering that the coverage is performed by aerial vehicles, the area covered from a single vehicle at a given instant is a circle around the projection of the position of the vehicle on the soil. As consequence in a 2-D environment is not sufficient to consider only a bounded interval to update A but it has to be considered the circle centered in the vehicle with radius ρ_v and also the motion of the vehicle from Pos_{t-1} to Pos_t that creates a stripe of reset points. So considering $K = Pos_t$ and

$L = Pos_{t+1}$, the area covered at each motion is represented in 3.4.

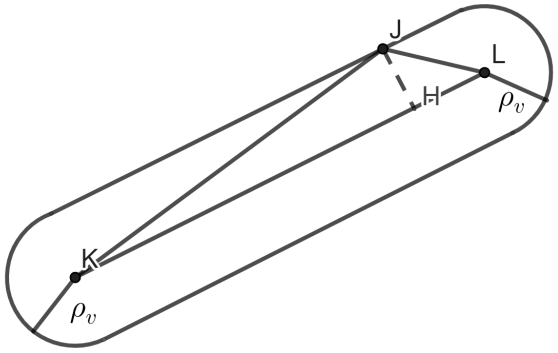


Figure 3.4: 2-D Alpha coverage during the motion

To correctly update α a geometrical relation to describe the set of points $[x, y]_{seen}$ inside the stripe is required. To simplify the description the points seen can be split in two groups:

- Points seen at the beginning or at the end of the step.
  They are representable as the ones inside the circles of radius ρ_v and center the position of the vehicle, in the figure represented from points K and L. They can be described simply with the disequations extrapolated from the equation of a circle in the 2-D plane:
  $$\begin{aligned}(x - x_K)^2 + (y - y_K)^2 \leq \rho_v \\ (x - x_L)^2 + (y - y_L)^2 \leq \rho_v\end{aligned} \tag{3.1}$$
- Points covered during the motion of the sensor.
  Again the geometry of the system has to be exploited, even if the extrapolation of the condition is less straightforward. It can be noticed that for all the points J covered in the path from K to L the projections onto the direction $\overline{KL}$ of $\overline{KJ}$ and $\overline{LJ}$ have sum equal to $\overline{KL}$ itself. Moreover since $\forall[x, y] \in stripe\, JH \leq \rho_v$ a belonging condition can be found in the following way:
  $$KH + HL = KL \tag{3.2}$$
  From Pythagoras' theorem:
  $$JH = \sqrt{KJ^2 - KH^2} \tag{3.3}$$

$$JH = \sqrt{JL^2 - LH^2} \tag{3.4}$$

Rearraging 3.3 and 3.4, since $JH \leq \rho_v$

$$\rho_v \leq \sqrt{KJ^2 - KH^2} \quad \rightarrow \quad KH \leq \sqrt{KJ^2 - \rho_v^2} \tag{3.5}$$

$$\rho_v \leq \sqrt{LJ^2 - LH^2} \quad \rightarrow \quad LH \leq \sqrt{LJ^2 - \rho_v^2} \tag{3.6}$$

At this point summing 3.5 and 3.6 the 3.2 is obtained and a condition with no unknowns:

$$KL = KH + HL = \sqrt{KJ^2 - \rho_v} + \sqrt{LJ^2 - \rho_v} \tag{3.7}$$

3.7 is the second condition we need to identify the points seen during the motion.

At this point the update of A at each step can be performed verifying the validity of the disequations 3.1 and 3.7 for the points of the region.

3.4 2-D Implementation

Although in the 2-D motion control of vehicles inside the target region $T = [0, E]\text{x}[0, E]$ different algorithms are applied, there are some common functions used in all the developed algorithms.

- An action necessary in each algorithm is for sure the A update. According to the relations found above the algorithm to update it after the motion of the sensor is the following one.
  The U_{matrix} is the update matrix to add to A. It can be a uniform unitary matrix or a non-uniform one according to the specific application considered. In the present implementation at first it is considered a uniform update to test and compare the results with the boustredopheon motion and then a non-uniform one, to create a situation closer to the one considered in the study, so to an environment with different priority areas.

- Another process present in the algorithms is the division in Voronoi Regions. In fact for both algorithms the next position is searched inside the vehicle's closest points. Starting from a set of v binary matrices $V \in \mathbb{R}^{s,s}$, with the '1' positions that represent the points closest to the $i-th$ vehicle. The region associated to the $i-th$ UAVs is defined as:

Algorithm 3: A Update

$A = A + U_{matrix}$
for $[x, y] \in T$ **do**
  $d_t = ||Pos_t - [x, y]||$
  $d_{t+1} = ||Pos_{t+1} - [x, y]||$
  $path = ||Pos_{t+1} - Pos_t||$
  if $d_t \leq \rho_v \wedge d_{t+1} \leq \rho_v$ **then**
    $A(x, y) = 0$
  end
  if $\sqrt{d_t^2 - \rho_v^2} + \sqrt{d_{t+1}^2 - \rho_v^2} \leq path$ **then**
    $A(x, y) = 0$
  end
end

Algorithm 4: V_i Region Computation

while *True* **do**
  $A = A + update$;
  for $[x, y] \in \mathbb{R}^{s,s}$ **do**
    for $i = 1 : v$ **do**
      for $j = 1 : v$ **do**
        if $\|[x, y] - Pos_i\| \leq \|[x, y] - Pos_j\|$ **then**
          $V_i(x, y) = 1$
        else
          $V_i(x, y) = 0$
        end
      end
    end
  end
end

3.4.1 Maxima Search Algorithm

The Maxima Search algorithm is a straightforward extension of the method applied in the 1-D case. It is developed in two versions: the first one that addresses as next target the absolute maximum of the Voronoi region V_i associated to the $i-th$ vehicles and the second one that chooses the next position among the k maxima values of the region, collected in the vector $kmax$, computing for each one the distance between it and the current position and moving toward the closest one.

Absolute Maximum Search

This is nothing more than the extension in the 2-D case of the Algorithm 2 with the only difference that in case of more spots with the same α the closest one is chosen as $Dest$. The only exception is the case when the points belonging to V_i are less that a certain value Lim_{Dim}. This additional condition allows to avoid to have a vehicle oscillating in a really small area, decreasing the overall efficiency of the monitoring. It is important to underline that in case searching for the positions of the maximum sometimes there are more points with the same value, x and y are two vectors and not two single values.

Algorithm 5: 2-D Single Maximum Search

```
while True do
    Apply Algorithm 3
    Apply Algorithm 4
    for i = 1 : v do
        if sum_{(x,y)∈T} V_i(x,y) ≤ Lim_Dim then
            A_{t,i} = A
        else
            A_{t,i} = A * V_i
        end
        [x,y]_max : A(x,y) = max A
        if length(x) > 1 then
            for j = 1 : length(x) do
                dist(j) = |[x(j), y(j)] − Pos_t|
            end
            J : dist(J) = min dist
            Dest = [x(J), y(J)]
        else
            Dest = [x, y]
        end
    end
end
```

List of Maxima

Despite the acceptable performances of the first version of the Maximum Search some spots are left unseen reducing the overall efficiency of the algorithm and requiring a longer time to reach a steady navigation condition. This issue can be overcame replacing the search for the absolute maximum of the V_i with the closest one among the k maxima, collected in the vector $kmax \in \mathbb{R}^{k,1}$. This expedient creates a more compact monitoring of the region.
The algorithm has the same structure of Algorithm 5, but for the creation of $[x, y]$ lists that refer to the positions of the $k-th$ maxima values. To make more linear the pseudocode, Algorithm 6, is introduced the notation $\max_k$ to indicate one of the k maxima of A.
Obviously even if the Algorithm 6 reduces the unseen spots it does not eliminate them completely.

3.4.2 MPC-Optimization Algorithm

The previously described algorithms assure the covering of the α maxima inside a temporal window, but to get a stable motion they require a long iteration time since their purpose is not to choose the best next step but to reach as fast as possible the closest maximum. Following this procedure the neighborhood of the vehicle is not consider at all and so the eventual points around it are left behind creating the necessity to come back in a second moment.
To overcome this phenomenon an optimization problem is created in such a way to optimize a global term, that considers the maximum inside the sub-region associated to the vehicle, and a local term, that considers the neighborhood values. This composite objective function is inserted inside a MPC-based algorithm in such a way to choose the single next step that allow you to minimize the objective function value over a chosen prediction horizon, h. To optimize the objective function it is used the Matlab function $fmincon$ that finds the local minimum of the function itself through a gradient-based method.
After defining the neighborhood of the vehicle as the circle with radius equal to the maximum distance that the UAV can monitor during the prediction phase, $\rho_p = \rho_v + h$, and centered in the initial position of the vehicle, the optimization variable has to be chosen. The initial decision about it was computing directly the next position of the vehicle inside the region.
Then the composition of the objective function was faced. Being the choose of the objective function crucial for the correct functioning of the motion planning an amount of tries was done, considering simpler expressions or more complex ones. The considered functions are a combination of all or part of the following elements, each one representing a temporal or spatial characteristic of the area:

Algorithm 6: 2-D List Maxima Search

while *True* **do**
  Apply Algorithm 3
  Apply Algorithm 4
  for $i = 1 : v$ **do**
    $kmax \in \mathbb{R}^{k,1} = \max_k A$
    $[x, y]_k : A(x, y) = \max_k A$
    if $(length(x) \geq k) \wedge ((x(1), y(1)) \geq Th)$ **then**
      for $j = 1 : length(x)$ **do**
        $dist(j) = |[x(j), y(j)] - Pos_t|$
        $positions(j) = [x(j), y(j)]$
      end
    else
      while $r = 1 : k$ **do**
        $[x_r, y_r] = [x, y] \in \mathbb{R}^{s,s} : A(x_r, y_r) = kmax(r)$
        for $j = 1 : length(x_r)$ **do**
          $dist(r) = |Pos_t - [x_r(j), y_r(j)]|$
          $positions(r) = [x_r(j), y_r(j)]$
          $r = r + 1$
        end
      end
    end
  end
  $J = \{J \in \mathbb{R} : dist(J) = \min dist\}$
  $Dest = positions(J)$
end

- $\overline{\alpha}$: average of the priority function.
  It was considered a local element to minimize and pushes the vehicle to cover with the next steps the greater values inside the neighborhood. The drawback is that in a situation of symmetry priority distribution with respect to the position of the vehicle it begins to oscillate since it is surrounded from equally convenient directions and in successive steps may two opposite directions are chosen in such a way to have the UAV stuck there. This undesirable behaviour it is possible also because even if the priority of the neighbor unseen points increase, there is small region just observed from the vehicle, where it is always 0 or 1 and that as consequence takes down the value of $\overline{\alpha}$.

- $\tilde{\alpha}$: median of the priority function.

It was considered as both local and global element and it pushes the vehicle to cover points that have priority greater at least than half of the values. The problem in this case is that it does not take into account directly the values of the covered points, for example considering a region with $\tilde{(\alpha)} = 50$ to pass across a point with priority 60 or across one with priority 90 is equivalent in minimizing the median.

- $\sum_k max(\alpha)$: sum of the k maxima value of α.
  It was considered both as a local and a global term. It makes UAV to follow the highest values without considering the distance between that points and the current position of the vehicle, so in case of a global function if the greater priority points are further ρ_p all the directions are equivalent. In case of local function the effect is similar to the median.

- $\sum(A - A_{update})$: sum of the priorities of the points observed in the following step.
  The effect is similar to the $\overline{\alpha}$ in fact, it tends to maximize the total sum of the seen spots' priority, that creates automatically a decreasing of the average of the local area, eliminating the oscillatory behaviour. Since this is a quantity to maximize and not to minimize in the final objective function has to be considered with negative sign.

- d: distance of the vehicle from the sub-region maximum.
  It is a global term to minimize.

- $e^{\frac{d}{\sqrt{2}E}}$: coefficient whose maximum value is e, it is used as weight coefficient to increase the importance of one of the term to minimize.
  In case it is used as coefficient for a term to maximize the exponential is replaced with a logarithm and the coefficient becomes $\log \frac{\sqrt{2}E}{d}$.

All these terms are at the beginning recomputed inside the optimization function and in the projection of the behaviour of the system in the MPC, then only the local terms are recomputed at each projected step while the global ones are estimated only at the beginning of the projection to decrease the computation time.
The initial tries are based on the minimization of the variables considered in the evaluation of the performance, so $\overline{\alpha}$, $\tilde{\alpha}$ and d, with median as global and mean as local variable, in the following shape:

$$f(\alpha) = \tilde{\alpha} * \overline{\alpha} + d \tag{3.8}$$

This initial formulation generates some oscillations so the exponential term is added as coefficient of the d term to push the vehicle toward the further maximum, but also this action still did not solve the oscillations problem. So the $\sum(A - A_{update})$

is added to the previous equation and the $de^{\frac{d}{\sqrt{2}E}}$ term is changed in $\log\frac{\sqrt{2}E}{d}$, in such a way to have anyway a measure of the distance of the current position and the maximum point. The new objective function is:

$$f(\alpha) = \tilde{\alpha} * \overline{\alpha} - \sum(A - A_{update}) * \log(\frac{\sqrt{2}E}{d}) \tag{3.9}$$

With the adding of the new element the swinging is eliminated, the performances are quickly comparable with the boustredopheon path for a uniform update and the problem of the spots left behind is overcome.
The drawback of the yet described settings is the computational cost of the implementation, in fact its entity makes the iteration time in a complex setting too long to be acceptable, hence the necessity to simplify the expression is created.
The first step was to change the optimization variable, that becomes the velocity direction. In fact it has a value contained in a smaller interval, $[-1,-1]$x[1,1], that makes faster its choice. Then the computation of the global terms was performed only once and not at each step of the MPC projection. Last action was the simplification of the optimization function itself from whom they are eliminated part of the elements composing it, in particular the average and the median. These modifications decreases the performances of the motion planning, increasing the presence of fluctuations in the maximum values of α, but reducing dramatically the time of computation for a complex initialization.
The fluctuations of α depends on some unseen spots that explode with the spending of iterations and since the straight trajectory is not the more convenient in minimizing the (3.9) the vehicle has to be in the neighborhood of the point to cover for sure it.
This issues is also linked to the fact that the path found with the optimization method is a track that the vehicles tend to follow, but for external modifications in the priority distribution. 3.5, represents the performance of a 10 prediction horizon MPC with a single vehicle in a $20x20$ region, used to perform preliminary tests on the behaviour of this algorithm due to the huge time required to simulate it in the bigger region. Referring to it, it is clear how the tendency to follow the track worsen the performance, in fact without the overlappings circled in red the covering time would be reduced reaching the ideal trajectory for a uniform region with a uniform update. The final algorithm, Algorithm 7, has a similar structure of Algorithm 5 with the difference that in this case the next position is explicitly computed solving the nMPC problem instead of selecting the step with length that takes the vehicle closer to the final destination.

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & f(\alpha)_{t+1} \\ \text{subject to} \quad & |u| = 1, \\ & Pos_{t+1,i} = Pos_{t,i} + u\Delta t \end{aligned} \tag{3.10}$$

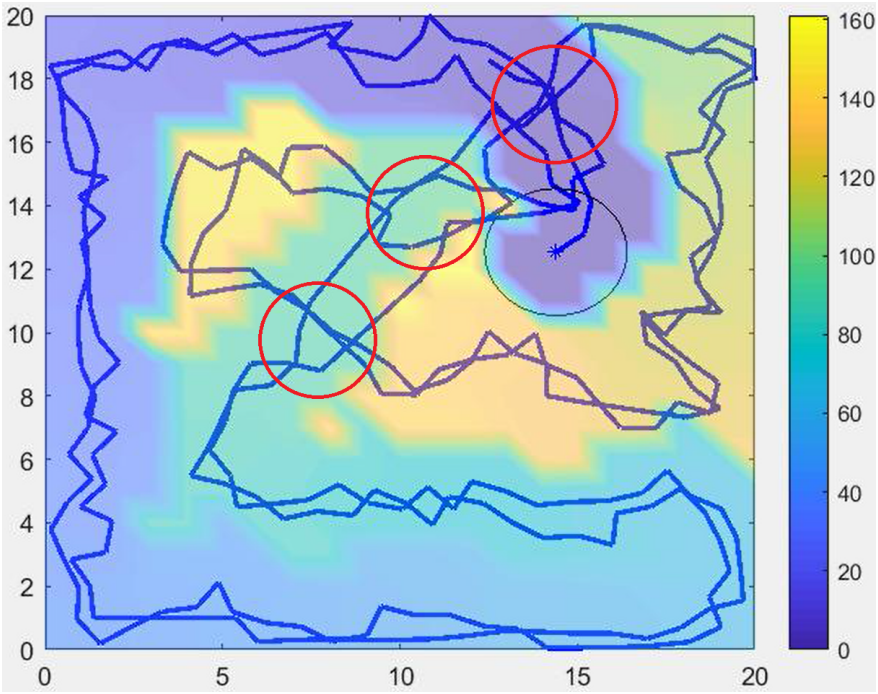


Figure 3.5: MPC-optimization behaviour

Algorithm 7: MPC algorithm

```
while True do
    Apply Algorithm 3
    Apply Algorithm 4
    for i = 1 : v do
        Solve 3.10
        Pos_{t+1,i} = Pos_{t,i} + u_opt Δt
    end
end
```

3.4.3 Decentralized Implementation

After the development of suitable navigation laws in a centralized control context it is relevant from a practical point of view to implement a version of the algorithms where the control is applied in a decentralized way.
The reason behind this choice is that in a real-life scenario it is likely to have a limited communication radius ρ_c, both for technical limitations and energy consumption needs, and as consequence not to have a complete connection of the network at any instant. Despite this, to guarantee the exchange of information

between vehicles over a finite time interval is crucial to the correct functioning of the monitoring fleet and then for the adaptivity of its motion. A complete lack of communication may lead to neglect some areas and overmonitoring others or, in a limit case, to the overposition of the UAVs' trajectories that, if during the simulation is an unwanted behaviour since it affects the efficiency of the coverage planning, in real-life applications it can cause the clash of the vehicles. To avoid this dangerous behaviour without sacrificing the adaptability of the algorithm a Voronoi division of the region is combined with scheduled meeting of the sensors, created specifically to exchange information.
The time interval T value between two meetings is a crucial factor and it is imposed at the beginning of the simulation. Initially it was computed as:

$$T = c\sqrt{2}E \tag{3.11}$$

with $\sqrt{2}E$ length of the diagonal of the square region and c a factor that is changed to analyze heuristically its best value. Since the velocity absolute value is equal to $1\frac{m}{s}$ and the time step at each iteration is supposed to be $1s$ the time T between two successive meetings represents also the length of the path travelled by the vehicle among two meetings, in m.
Then this definition is replaced with:

$$T = \frac{nodes}{vN} \tag{3.12}$$

with *nodes* equal to the total nodes in the grid representing the region to visit, v the number of vehicles and N a coefficient to choose, the modification is performed in such a way to extend the navigation time between two meetings.
Also the selection of the next meeting location was modified along the implementation. Initially it was chosen as the further position from the current one, among the points with a $\alpha \geq max(A) - T(U_{matrix})$, in such a way that at the time of the next information exchange the destination point will have at least a priority value equal to $max(A)$. To assure each vehicle arrives at a position $Pos_M - \rho_c$ a check is added before to update the position, so if $|Pos_{t,i} - Pos_M| \geq T$ the new $Dest_i$ is Pos_M until either $|Pos_{t,i} - Pos_M| \leq \rho_v$ or the whole network is connected. The check on the network connectivity is left but the meeting position choice is modified because selecting it among the further points it was always placed in one of the corner, creating the necessity to travel a long, useless path to reach it. The new meeting point is selected as the centroid of the centers of mass of the regions of interest recomputed for the next navigation period.
In the period between two successive meetings each vehicle moves inside its own region of interest, computed at the moment of the reunion through the Lloyd algorithm. The generators of the regions are v virtual sensors with initial position coincident with the UAVs' ones.

The happening of a meeting is checked through a vector counter, $count \in \mathbb{R}^{v,1}$, whose components are associated to the vehicles. At each iteration, if two vehicles met, so if their distance is less than ρ_c, the correspondent components are incremented of 1. The vector is initialized to zero and reset at each iteration, when the following condition is verified, the network is connected and a meeting has taken place.

$$\sum_{i=1:v} count(i) \geq 2(v-1) \tag{3.13}$$

On the other hand is the sum value is greater than zero it means that some vehicles met but the whole network is not connected and if the sum is equal to 0 there is no contact at all during the current iteration.
Once arrived at the meeting point the exchange of information is performed through a Consensus Algorithm, Algorithm 8, consisting in comparing the A_i of each UAV and then, for each point of the region, in choosing the minimum among the correspondent values prensent in the single matrices. In this manner the priority of each point is updated in such a way to take into account the last visit of whichever vehicle.
The single next position of the vehicles is chosen following one of the previous centralized implementation methods, in particular the List Maxima Algorithm.
As it appears clear from the above explanation in implementing the decentralized control an amount of functions are introduced, in the following they are explained one by one and their pseudocode is provided.

- Consensus Algorithm
  A consensus algorithm is typically used in a context where a number of entities have to communicate to update their information. The final single information of each sensor is established by a general rule linked to the nature of the problem. In this case the relevant data is the priority of each point that is strictly linked to when that point was seen the last time, so the wanted value each UAV has to consider is the one that indicates the last visit in each point, hence the component-wise minimum of the v A_i.

Algorithm 8: Consensus Algorithm

```
for [x, y] ∈ R^{s,s} do
    A_real(x, y) = min(A_i(x, y))
end
for i = 1 : v do
    A_i = A_real
end
```

Algorithm 9: Next Meeting Point Selection. First method

$count = 0$
$A_{map} = A - T$
for $[x, y] \in \mathbb{R}^{s,s}$ **do**
  if $A_{map}(x, y) \geq 0$ **then**
    $A_{map} = 1$
    $count = count + 1$
    $[x, y]_{A_{map}(x,y)>0}(count) = list$
    $dist(count) = |Pos_M - [x, y]|$
  end
end
$index = i : dist(i) = max(dist)$
$Pos_M = [x(i), y(i)]$

- Meeting Point
  The search for the next meeting point is a crucial choice. Initially its computation was performed in a similar way with respect to the Maxima Search. In this case the searched positions are the one with values greater than a threshold, imposed at $max(A) - T$, and then the further among them is chosen. This last decision depends on the nature of the motion planning algorithms, in fact both of them search for the closest maximum or however choose the next position in a limited radius, imposing the meeting point far enough makes the possibility to cover it during the intermediate navigation less likely.
  This method of choosing has two main flaws: since the chosen maximum is the further one the meeting point is probably in one of the corner creating the necessity to travel a long path to reach the it. Moreover since the region of interest of the vehicles is computed with a simple Voronoi division it is frequent the overlap of two UAVs for an amount of iterations, since they may be in the same position at the moment of the sub-regions computation.
  To overcome this problems both the centroid computation and the regions one were modified in such a way to avoid the additional useless path and to have a division of the region that takes into account the current state of the area.

- Region Lloyd Division
  The second described problem, so the overposition of the subregions associated to the vehicles, is solved through a different process for the computation of the regions of interest.
  To apply this method the regions are computed applying the Lloyd algorithm until convergence with generators of the regions v virtual vehicles, whose initial positions coincide with the real vehicles ones.

Algorithm 10: Next Meeting Point Selection. Second and definitive method.

1 New V_is already computed.
2 $w_i = \sum_i V_i$
3 $w_{tot} = \sum_i w_i$
4 $x_M = \sum_i w_i x_i$
5 $y_M = \sum_i w_i y_i$
6 $Pos_M = \frac{[x_M, y_M]}{w_{tot}}$

The final implementation of the decentralized algorithm has the structure represented in 3.6.

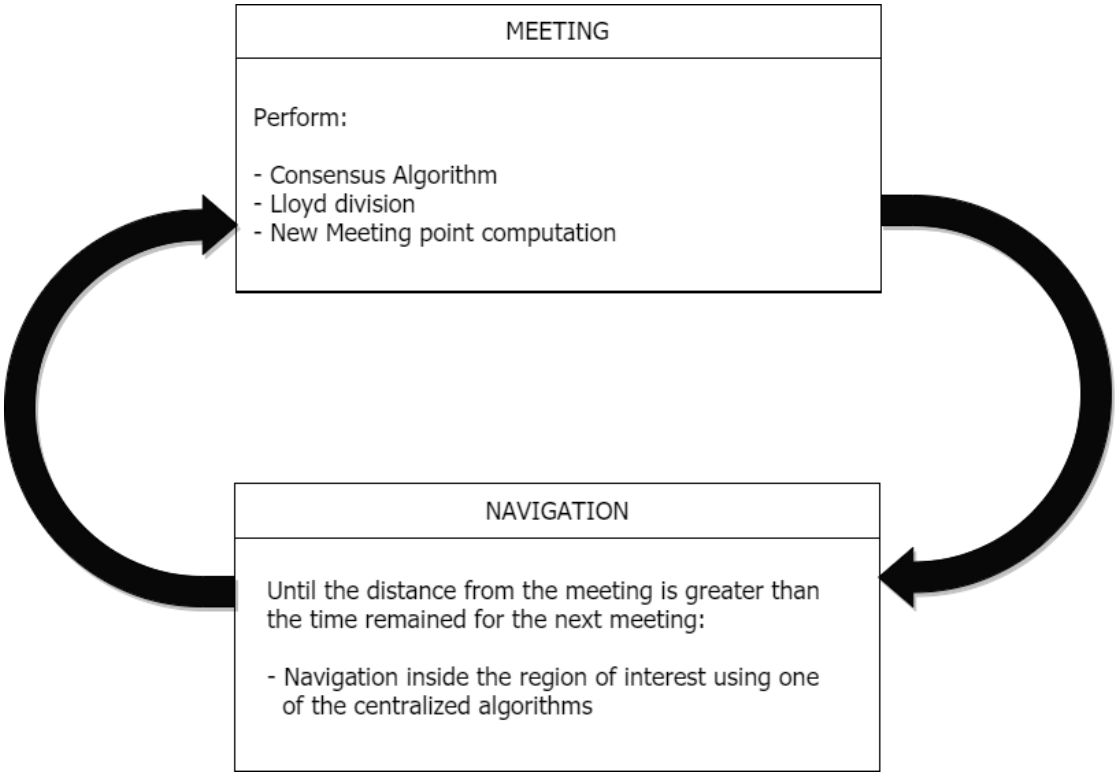


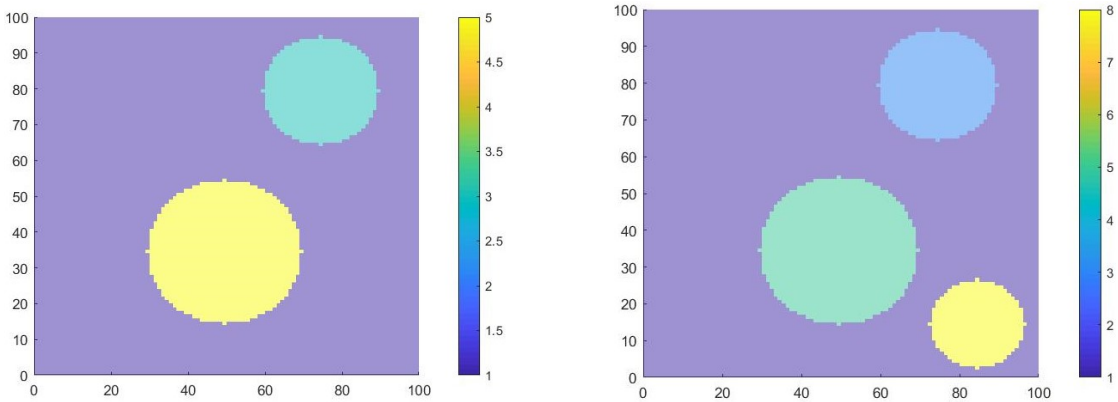
Figure 3.6: Decentralized algorithm

Chapter 4

Simulation results

The developed algorithms are simulated entirely in Matlab, without the usage of any external solver. To test their effectiveness, their behaviour is compared in different update cases. To simplify the reading of the data the comparisons are made between two methods at time. The initialization of the priority function over the region is realized both uniformly and randomly, but not always both results are reported. This choice is due to the little importance of the region initialization in the behaviour of the algorithm, as anticipated in Chapter 3 and as it is shown in the simulation results. The considered updates are a uniform update and two non-uniform ones, in Figure 4.1a and Figure 4.1b.

The time of iteration depends on the time of convergence of the specific considered



(a) First non-uniform update

(b) Second non-uniform update

Figure 4.1: Non-uniform updates applied in Matlab simulations

algorithm, in fact after a initial length equal to 133.500 iterations for the uniform update the centralized algorithm shows a convergence after 20.000 iterations and

so a number of total iterations of 80.000 produces reliable results.
In the following sections they are reported the initializations used for the specific simulation and comparison.

4.1 Centralized Maxima Search Algorithm

For the test of the Centralized Maxima Search Algorithm the variables are initialized as follows:

- $n = 2$;
- $E = 100$;
- $v = 3$;
- $\rho_v = 2$;
- $\delta = 1$;
- $\Delta t = 1$;
- $u = 1$;
- $k = 20$;
- $g = 10$;
- $Pos_{t,i}$ randomly initialized in the region;
- $A = 0^{ExE}$ (uniform case);
- A uniformly random distribution between 0 and 100 (non-uniform case);

4.1.1 Single Maximum Search vs Limit Algorithm

The time of iteration is different: the Limit algorithm immediately converges after a single exploration of the region, 1143 iterations, so a time of iteration of 10.000 is enough while the time of iteration of the Single Maxima Algorithm is 133.500 for the uniform update, in such a way to have a measure of the required time to converge and then in the cases of non-uniform updates it is reduced to 80.000.

- Uniform update
  The case of uniform update with a uniform initialization priority in the region is the one present in the Figure 3.1 used in the introduction as example, so in the following the reported simulation will be a random initialization in such a way to have the occasion to show the lack of differences in the steady state behaviour.

  In condition of uniform update the limit behaviour is clearly superior in

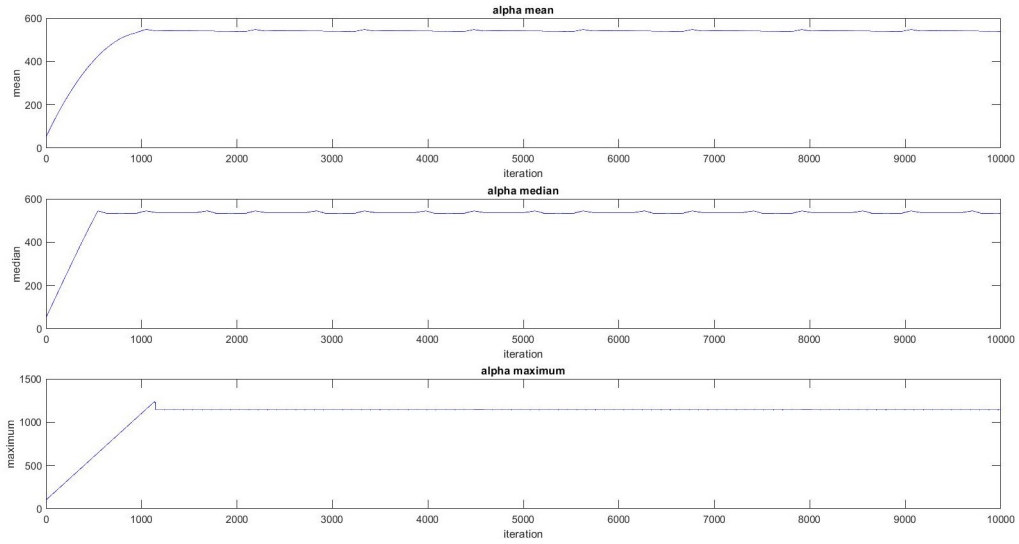


Figure 4.2: Limit behaviour with a random initialization and uniform update

performance to the Absolute Maximum Search algorithm, in fact its steady maximum value is 1143 while for the developed algorithm is about 2000, the time required to see each point is almost twice. Also the median and the mean in case of the comparison algorithm are lower, both around 500, while Figure 4.3 and Figure 4.4 shows respectevely a median of about 650 and a mean of about 800, with not meaningful differences but for a slight delay in convergence in case of a uniform initialization of the priority in the region.

- First non-uniform update
  Since it is demonstrated that a different initialization does not create substantial modification in the motion and performances of the vehicles monitoring, in the following only one of the considered initialization will be used in analyzing the results, in particular in the present case the reported results are referred to a uniform initialization.

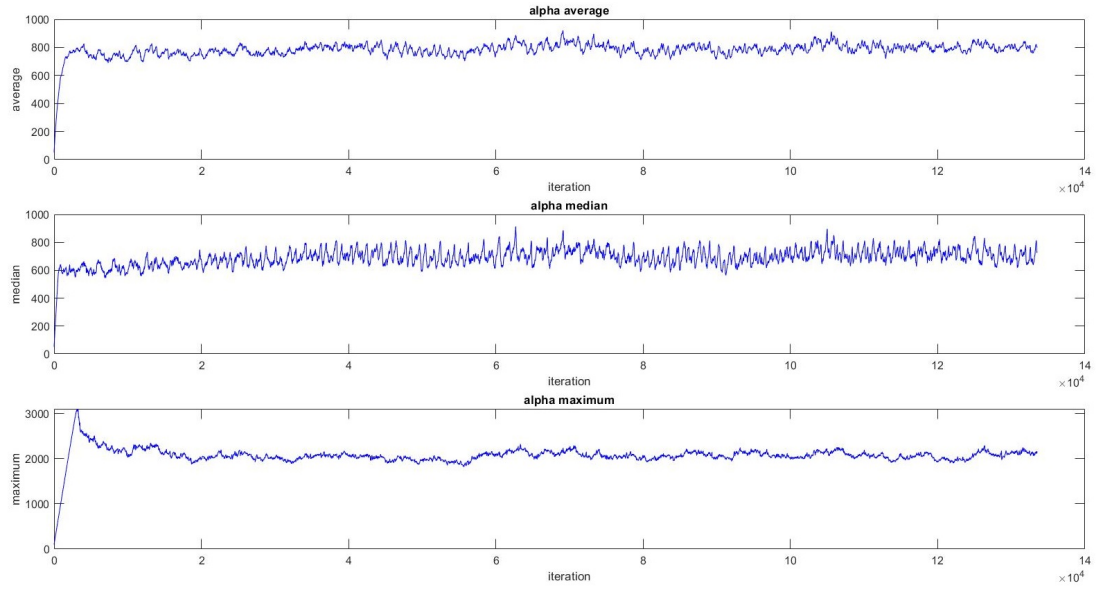


Figure 4.3: Absolute Maximum Search algorithm with random initialization and uniform update

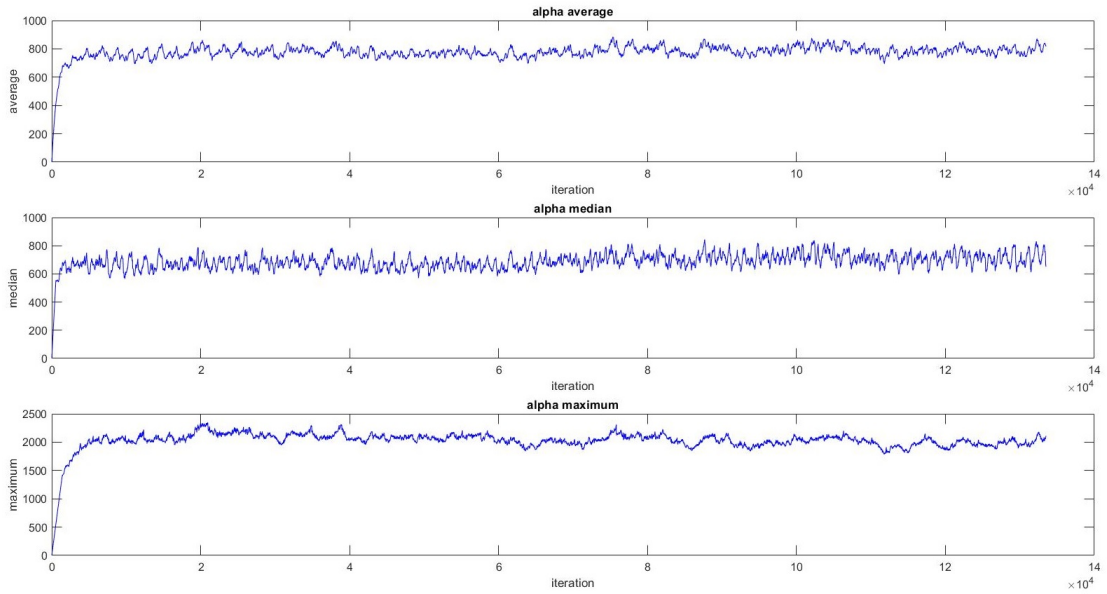


Figure 4.4: Absolute Maximum Search algorithm with uniform initialization and uniform update

The length of the limit algorithm simulation is still 10.000 iterations while for the Absolute Maximum Search algorithm it is shorten to 83.500. In this

case a the non-uniform initialization of Figure 4.1a is applied for the whole considered time.

Immediately after a consideration about the general worsening of the perfor-

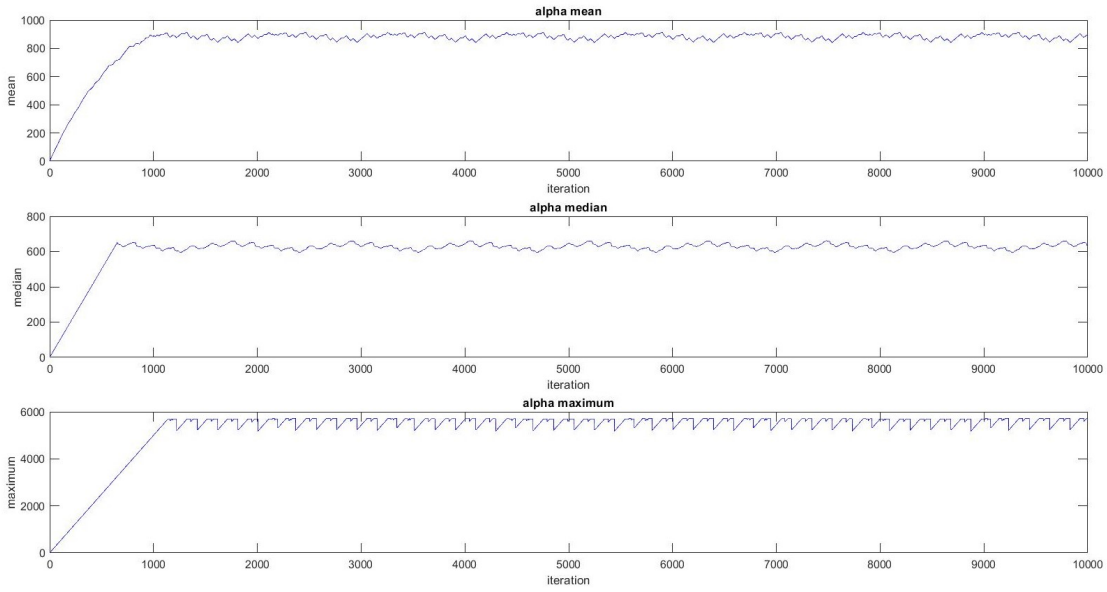


Figure 4.5: Limit algorithm with uniform initialization and first non-uniform update

mance, translated in the increasing of all the tracked variable, it appears clear that the priority is better taken under control in the case of the Maximum Search algorithm. Indeed despite the greater values of average and median of α the maximum value at each instant becomes about of 3700 in the case of AMS algorithm while for the limit algorithm it reaches almost the 6000.

- Second non-uniform update
  The last test performed consists of modifying the priority update throughout the simulation, passing from the update described by Figure 4.1a to the one in Figure 4.1b. The duration of the simulation is the same of the previous case and the change is done about at half of the test, at 5000 for the limit algorithm and at 40.000 for the AMS algorithm.

  The results in this case are similar to the previous one. In fact after the modification the performances degrade in both cases, but while the AMS is able to contain the modification and to adapt to it, reaching a maximum oscillating around 5000 with a pretty quick response, in the case of the limit algorithm its uncapability to model its behaviour on the modifications of the region makes the maximum explode to almost 10.000.

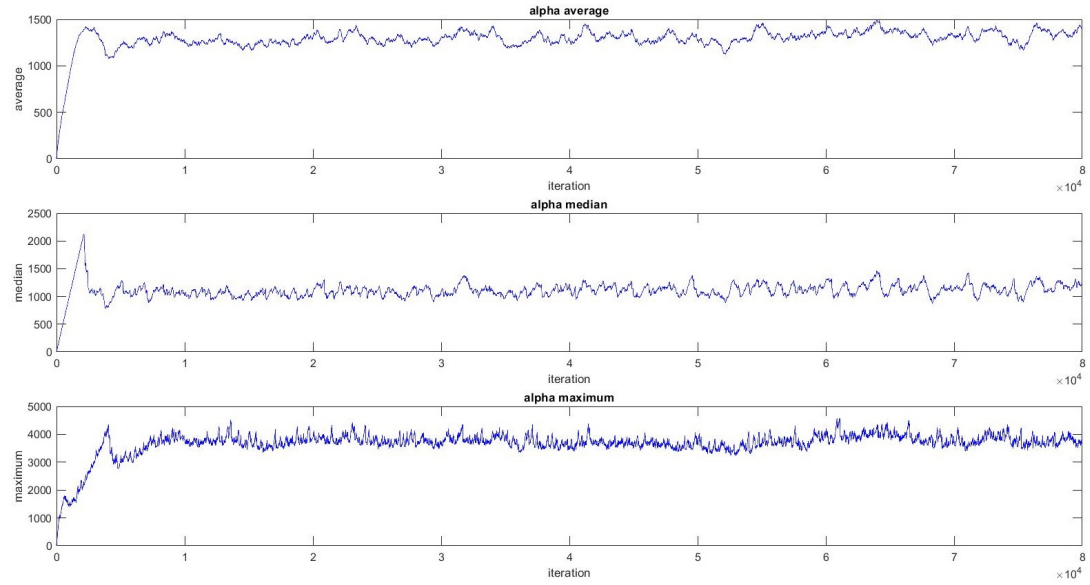


Figure 4.6: Absolute Maximum Search algorithm with uniform initialization and first non-uniform update

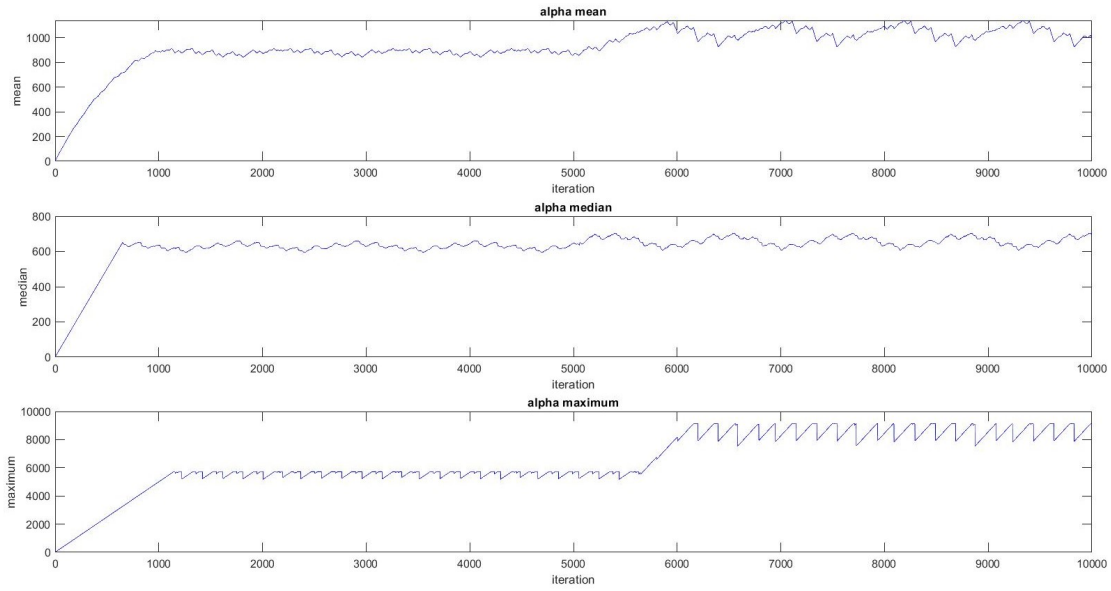


Figure 4.7: Limit algorithm with uniform initialization and changing non-uniform update

A clear point in this discussion is that despite the undeniable superiority of the boustrodopheon path in a uniform setting as soon as the update is not uniform

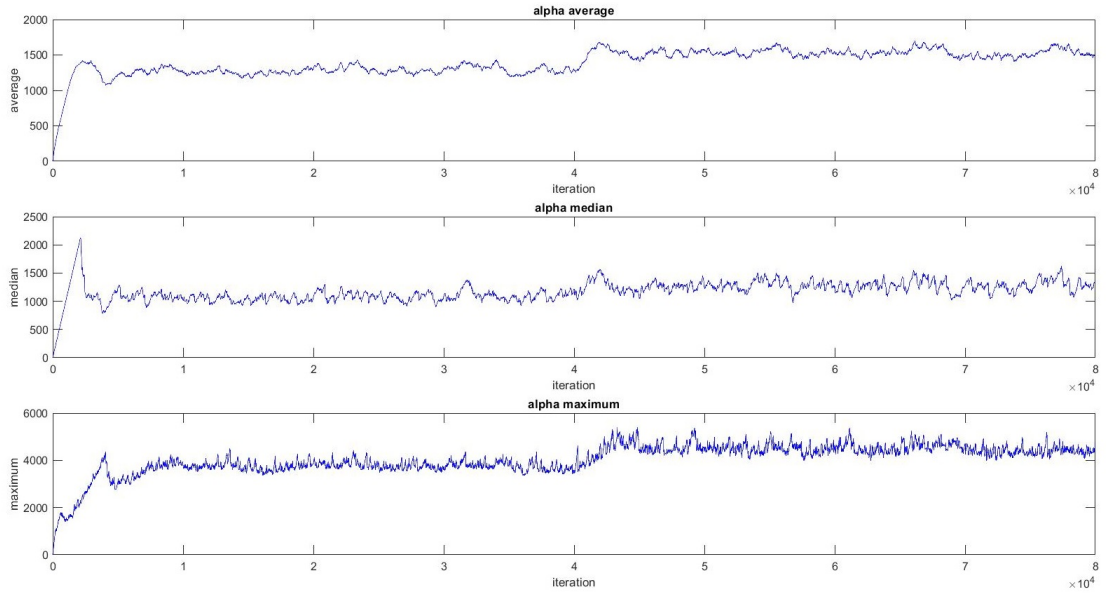


Figure 4.8: Absolute Maximum Search algorithm with uniform initialization and changing non-uniform update

anymore its performances, in particular the maximum priority inside the region, degrade dramatically. The same tendency can be observed also in the Absolute Maximum Search algorithm but in a relevantly minor measure.

4.1.2 Absolute Maximum Search vs List Maxima algorithm

The iteration time in the two cases is the same of the AMS in the previous comparison. In the following the reported results referred to the uniform initialization of the region.

- Uniform update
  Figure 4.9 shows than in this case the convergence time is longer than the AMS algorithm with a final maximum α smaller, from about 2100 to about 1900. Looking at $\overline{\alpha}$ the values trend in the LM algorithm has a steady-state lower, about 700, value and minor amplitude in the oscillations and the same modification is observable also in $\tilde{\alpha}$.
  The median and the mean have a similar behaviour in the steady-state but the initial peak is relevantly minor. Instead the maximum values are almost the same, with again a smaller amplitude in this second case, that does not represent a meaningful improvement strictly from a priority point of view. So this first test shows a slight improvement in the performance.

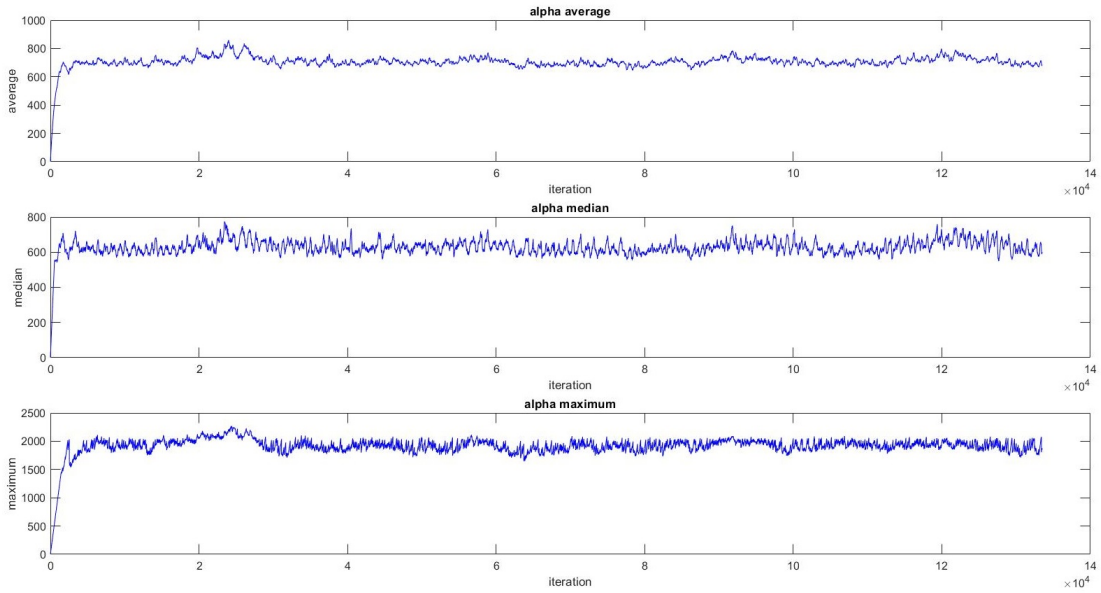


Figure 4.9: List Maxima algorithm with uniform initialization and uniform update

- First non-uniform update
  Also in this case the priority region initialization is uniform, but here the differences from a temporal point of view arise. In fact there is not a direct correlation between the time elapsed between two successive visits in the same point and the α value in that point. To observe the motion from a temporal perspective it is useful to map the time elapsed since the last visit for each of the grid nodes and observe the trend of the maximum one.
  Comparing it for the two algorithms through Figure 4.11 and Figure 4.12 the behaviour is also in this case very similar but for the initial peak that in case of the List Algorithm is limited.

- Changing non-uniform update
  Same initialization than the previous case.
  The way α changes has not particular modification with the implementation of the List Maxima algorithm but for the limitation of the initial peak and a steadier trend, in particular for the mean and the median. On the other hands considering the changings in the maximum time elapsed at each instant in the graphs in Figure 4.14 and Figure 4.15, they describes as the region with this second algorithm is monitored in a more homogeneous way since its values oscillate around a smaller steady-state range with a minor variance.

The improvement in using a List Maxima algorithm is, in particular, in the initial phases since this method allows to avoid useless back and forth motions, covering

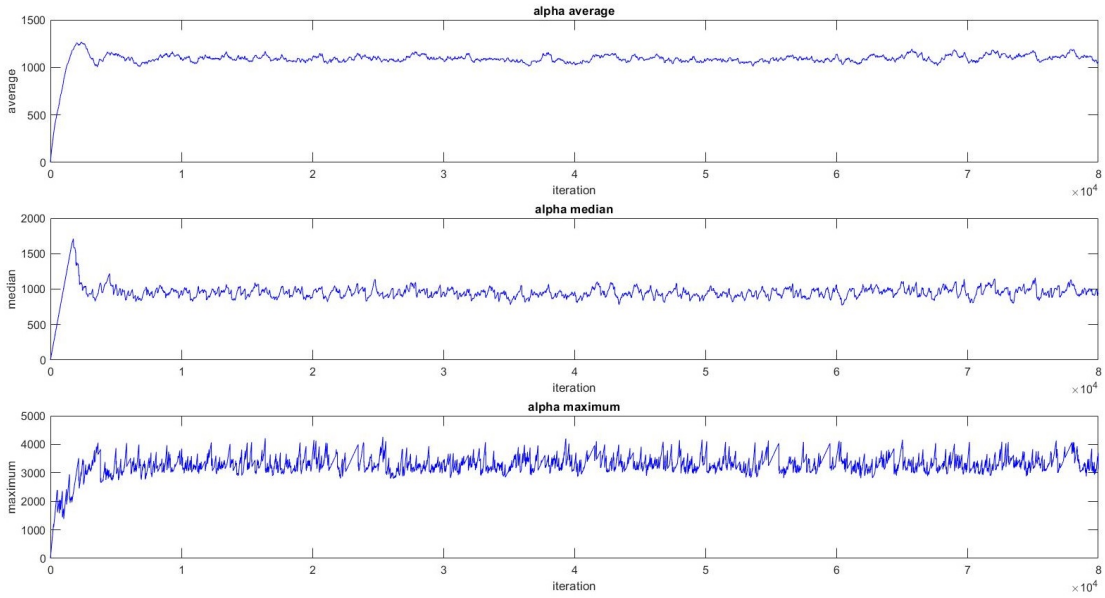


Figure 4.10: List Maxima algorithm with uniform initialization and non-uniform update

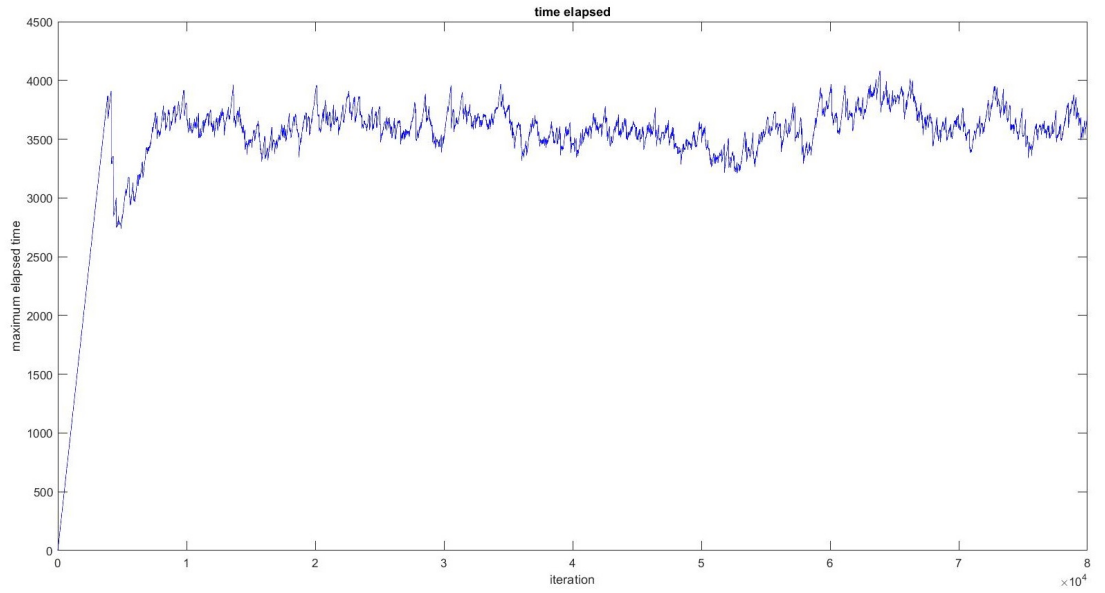


Figure 4.11: Time elapsed in Absolute Maximum Search algorithm with uniform initialization and non-uniform update

from the beginning the close high priority spots, creating in a strongly non-uniform

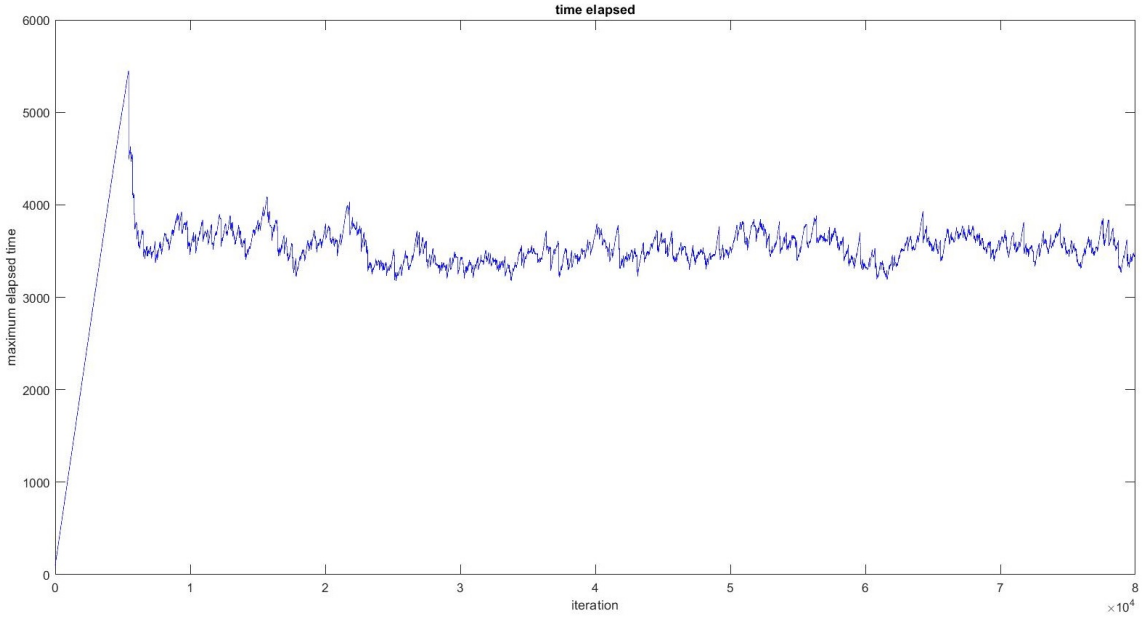


Figure 4.12: Time elapsed in List Maxima algorithm with uniform initialization and non-uniform update

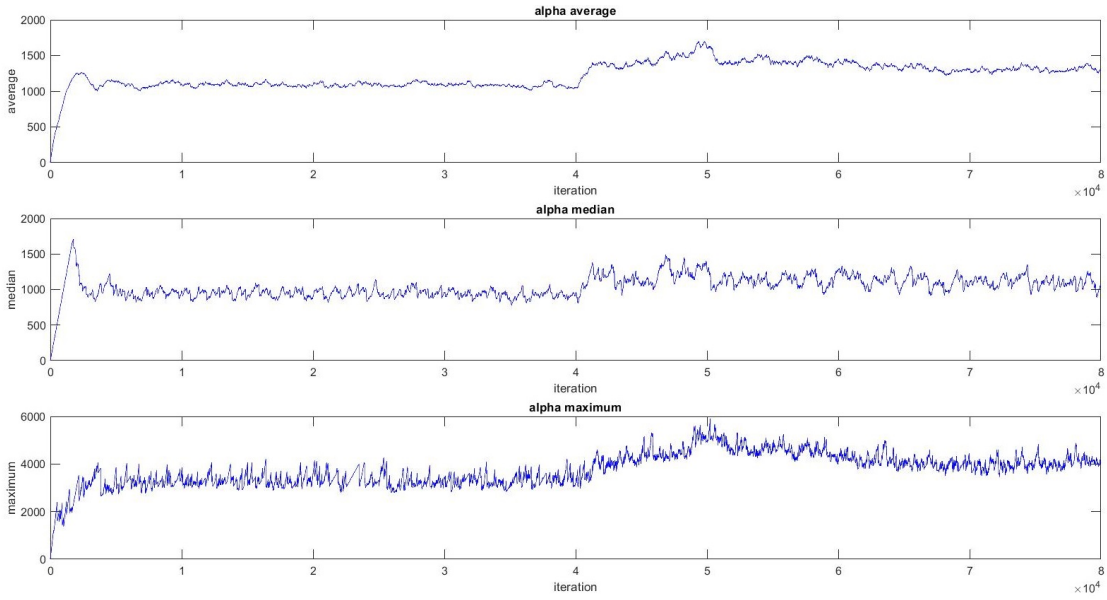


Figure 4.13: List Maxima algorithm with uniform initialization and changing non-uniform update

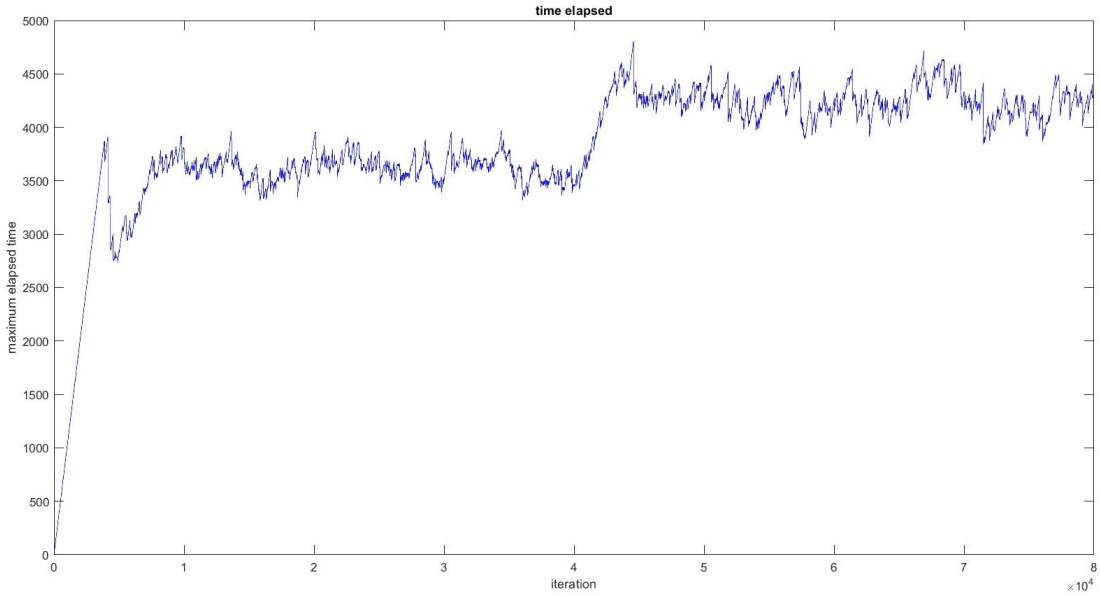


Figure 4.14: Time elapsed in Absolute Maximum Search algorithm with uniform initialization and changing non-uniform update

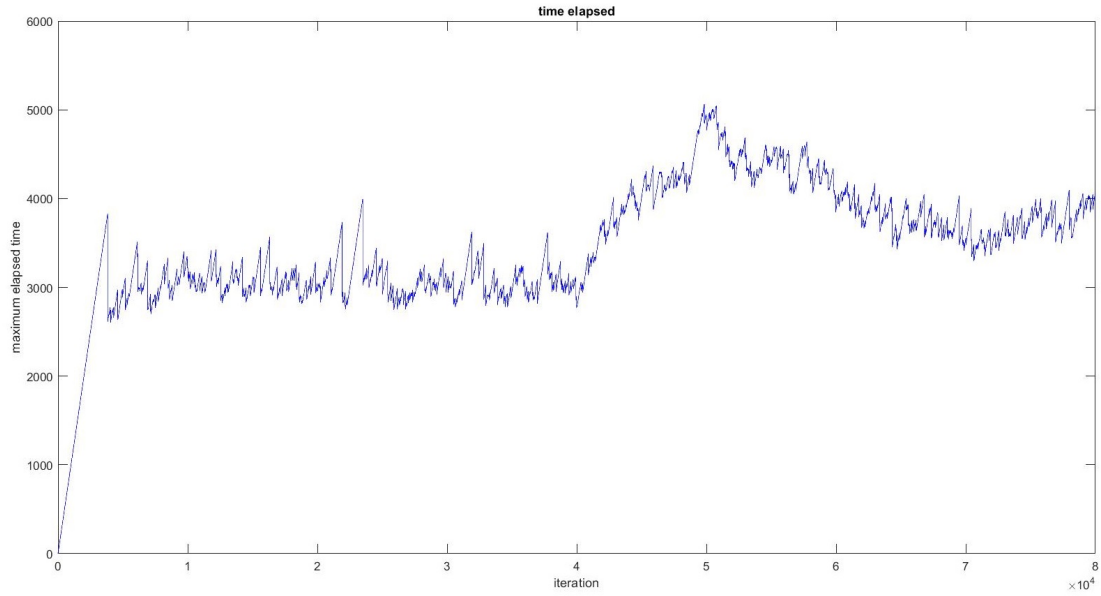


Figure 4.15: Time elapsed in List Maxima algorithm with uniform initialization and changing non-uniform update

update a more homogeneous monitoring without sacrificing the priority performances.

4.2 MPC Algorithm

The Model Predictive Control algorithm is tested in a smaller region both for the necessity to test in a simpler environment the algorithm and for being by its own nature more complex than a Maximum Search algorithm. The computational complexity creates also a longer time required to perform a single iteration. For this reason the duration of the simulation in this case is 3500 iterations.
The prediction horizon $h = 3$ is chosen for two main reasons: first of all increasing the prediction horizon extend the iteration time and also this value it gives good results. The second reason is linked to the way the algorithm is designed, the next position is not predicted with the same problem for the whole fleet, but singularly for each vehicle. So extending too much the prediction horizon the motion of the remaining vehicles is less taken into account in the optimization, creating worse results.

- $n = 2$;
- $E = 20$;
- $v = 2$;
- $\rho_v = 2$;
- $\delta = 1$;
- $h = 3$;
- $\Delta t = 1$;
- $u = 1$;
- $g = 10$;
- $Pos_{t,i}$ randomly initialized in the region;
- A uniformly random distribution between 0 and 100 (non-uniform case);

4.2.1 MPC algorithm vs List Maxima algorithm

Firstly, to understand if this ulterior step is meaningful it is useful to compare the behaviour of the MPC with the one of the List Maxima with the same initialization and duration.
It is observable from Figure 4.16 and Figure 4.17 that even if the mean and the median oscillate between the same values, despite the fluctuations of the MPC are more regular, from the point of view of the trend of $max(\alpha)$ the superiority of the predective algorithm is undeniable. It becomes steadier and maintains an almost constant behaviour from the iteration 500, while the Maxima List does not present any convergence in the considered interval. As consequence, it clearly appears that from a simulation point of view the MPC takes with it a relevant improvement in terms of both performance and speed of convergence.

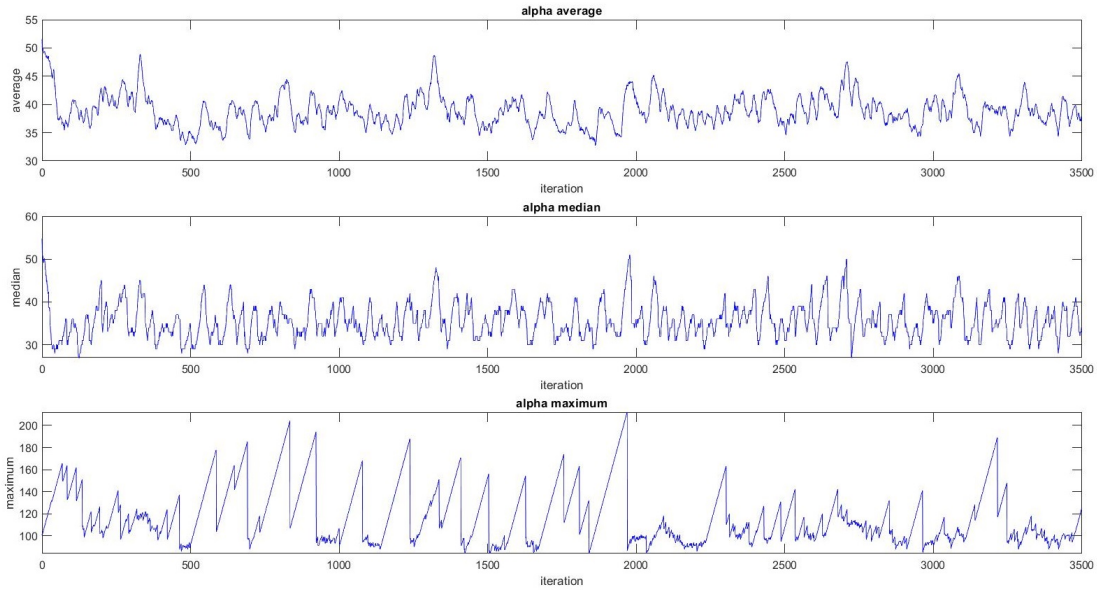


Figure 4.16: Maxima List with random initialization, 3500 iterations

4.2.2 First MPC algorithm vs Simplified MPC algorithm

Unluckily the iteration time required for the simulation of the first algorithm is not sustainable, in fact it is equal to more than 2 minutes for iteration in case of the region with $E = 100$ and $v = 2$ on a PC with 20x Intel Core i9-9900X CPU @ 3.50GHz, so it has to be simplified in such a way to obtain a faster optimization algorithm.
The simulation results for the simplified algorithm are acceptable even if it is

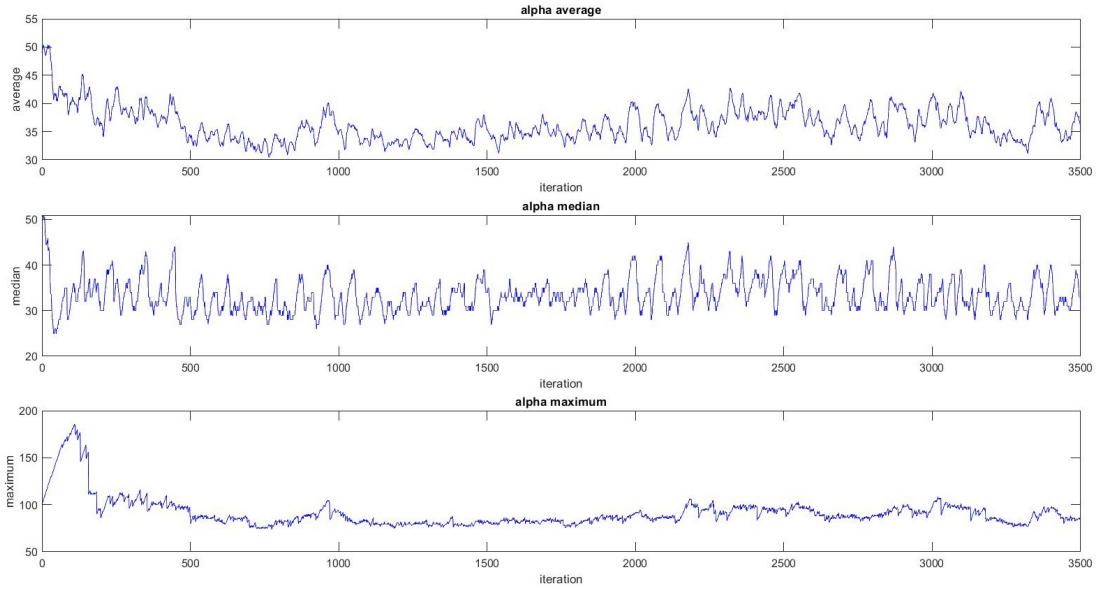


Figure 4.17: MPC with random initialization, 3500 iterations

present a slight degradation of the performance, consisting of a bigger variance and a longer phase required for the convergence.
Seen the interesting and still acceptable results, in Figure 4.18, the simplified algorithm is implemented in a larger region, $E = 100$ with $v = 3$, the time required to perform a single iteration becomes equal to 8 seconds in average, more acceptable than the previous time of more than a minute, but the performance in applying it worsen terribly, as shown in Figure 4.19 for a simulation long 59.000 iterations, where, despite the average and the median present a trend similar to the one of the Maxima Search algorithm, the maximum oscillates constantly among 2000 and 4000.
The explanation for this unwanted behaviour is that during the navigation some spots are left unseen and until one of the vehicle travel again in the neighborhood of that spot or the priority increases a lot and, as consequence, it is convenient from the objective function point of view to cover it, the point remains unseen and its priority function explodes making the algorithm unsuitable.

4.3 Decentralized Maxima Search Algorithm

For the tests of the Decentralized Maxima Search algorithm the variables are initialized as follows:

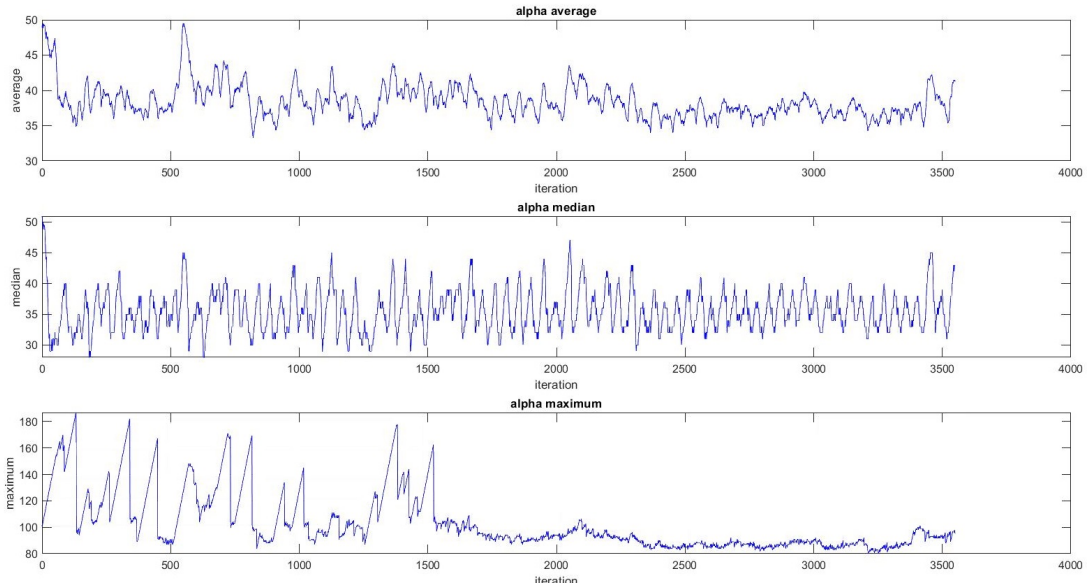


Figure 4.18: MPC simplified with random initialization, 3500 iterations

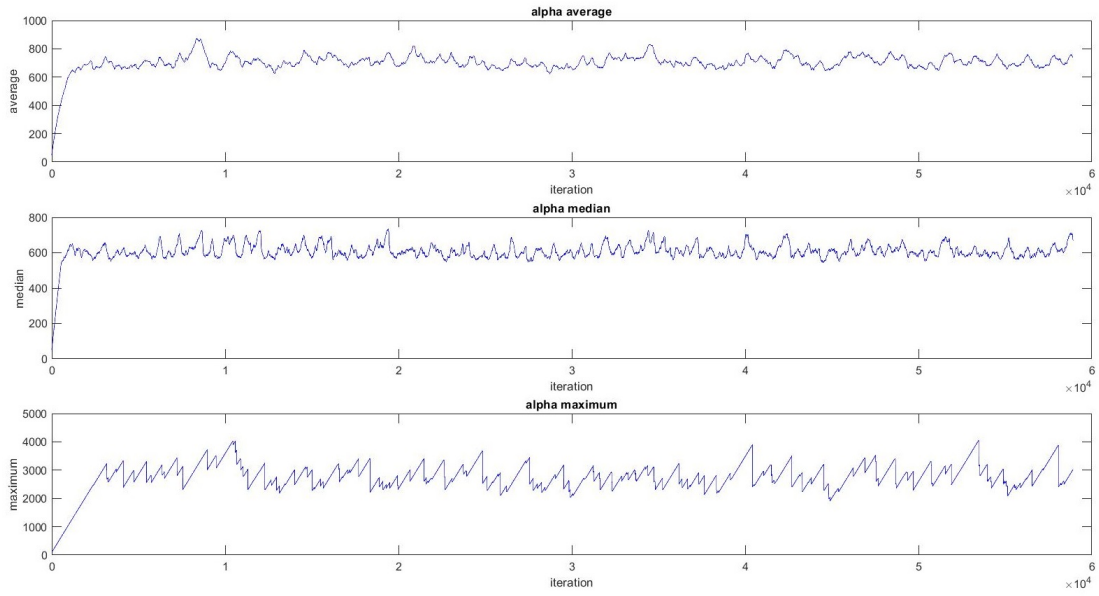


Figure 4.19: MPC simplified with random initialization, 100x100 region with 3 vehicles, 59000 iterations

- $n = 2$;
- $E = 100$;

- $v = 3$;
- $\rho_v = 2$;
- $\delta = 1$;
- $\Delta t = 1$;
- $u = 1$;
- $k = 20$;
- $g = 10$;
- $\rho_c = 10$;
- $Pos_{t,i}$ randomly initialized in the region;
- $A = 0^{ExE}$ (uniform case);
- A uniformly random distribution between 0 and 100 (non-uniform case);
- $T = 850$ or 1500, specified in the simulations;

4.3.1 Period 850 vs Period 1500

The values of the period is a crucial decision in this algorithm since it establishes the time between two meetings. The used values are chosen heuristically considering the total number of nodes to visit present in the grid and dividing them first for the number of vehicles and then for 2, in case of $T = 1500$, and for 4, $T = 850$.
In this case the time of iteration is always equal to 133.500 iterations for both uniform and non-uniform update and some considerations about the difference between a uniform and random priority initialization are pointed out.

- Uniform update
  Before to analyze in particular the differences among the graphs it is interesting notice that fixing the region for a interval of time longer than 1 second creates a trend of the mean and of the average closer to the limit algorithm than to the Maxima Search, although the choice of the next position is based on that algorithm.
  Observing at first the differences in performance between a uniform and a random initialization it can been observed that in both cases performances of a random and uniform initialization are pretty similar, with uniform case that presents slightly lower peaks. In the following the uniform initialization will be used.

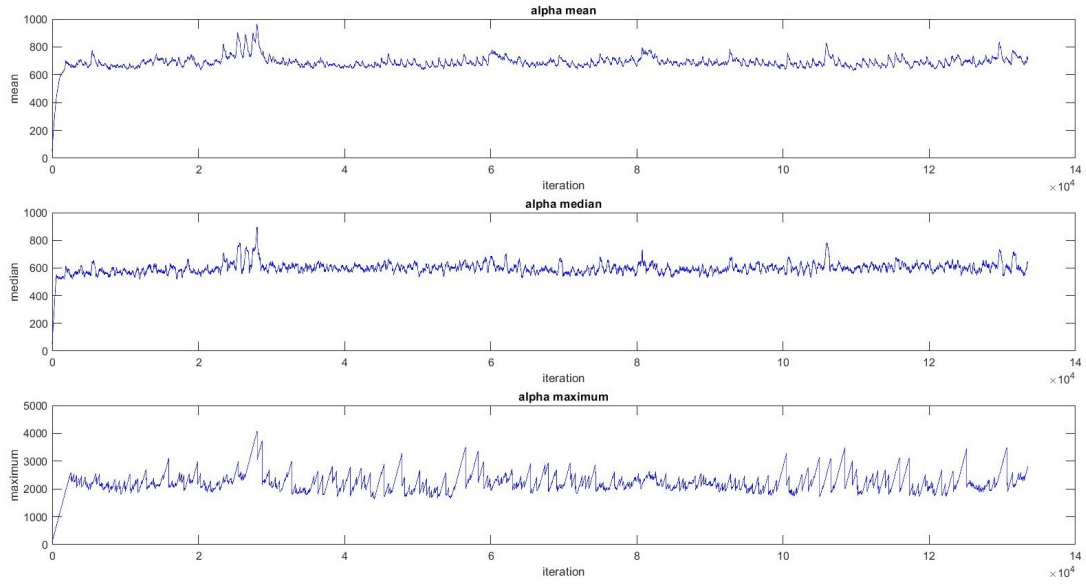


Figure 4.20: Priority with Decentralized algorithm, $T = 850$, random initialization and uniform update

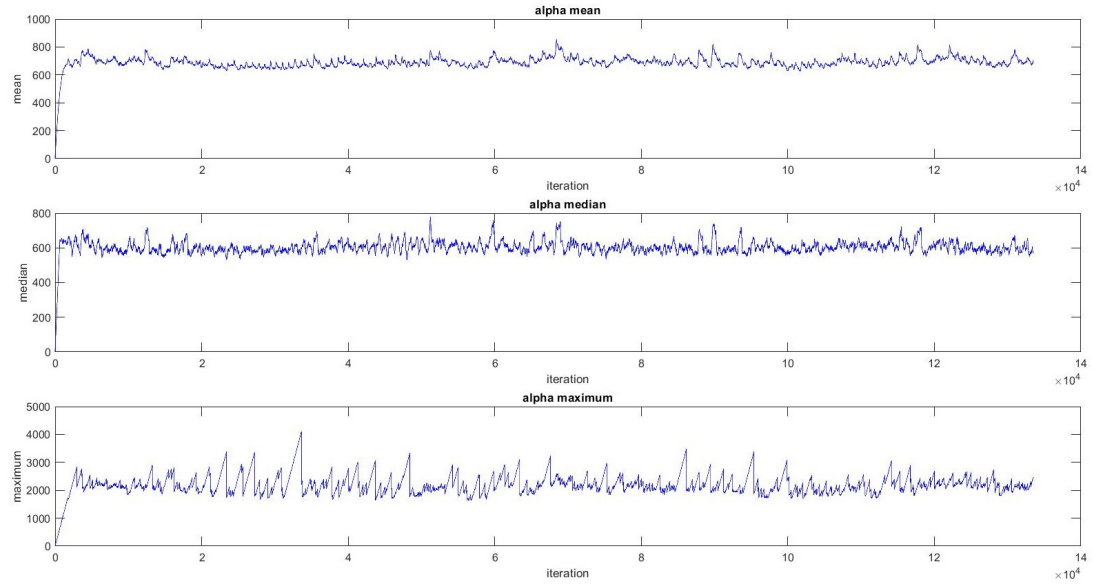


Figure 4.21: Priority with Decentralized algorithm, $T = 850$, uniform initialization and uniform update

Instead, comparing a longer and a shorter period, the spikes for a greater period are both less frequent and lower. This behaviour was expected because

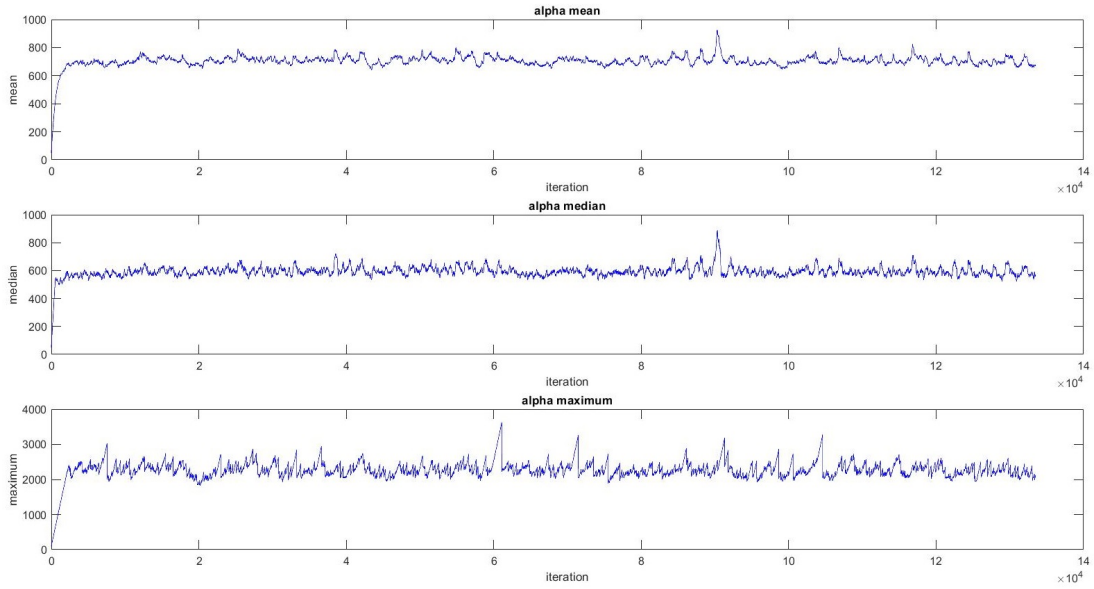


Figure 4.22: Priority with Decentralized algorithm, $T = 1500$, random initialization and uniform update

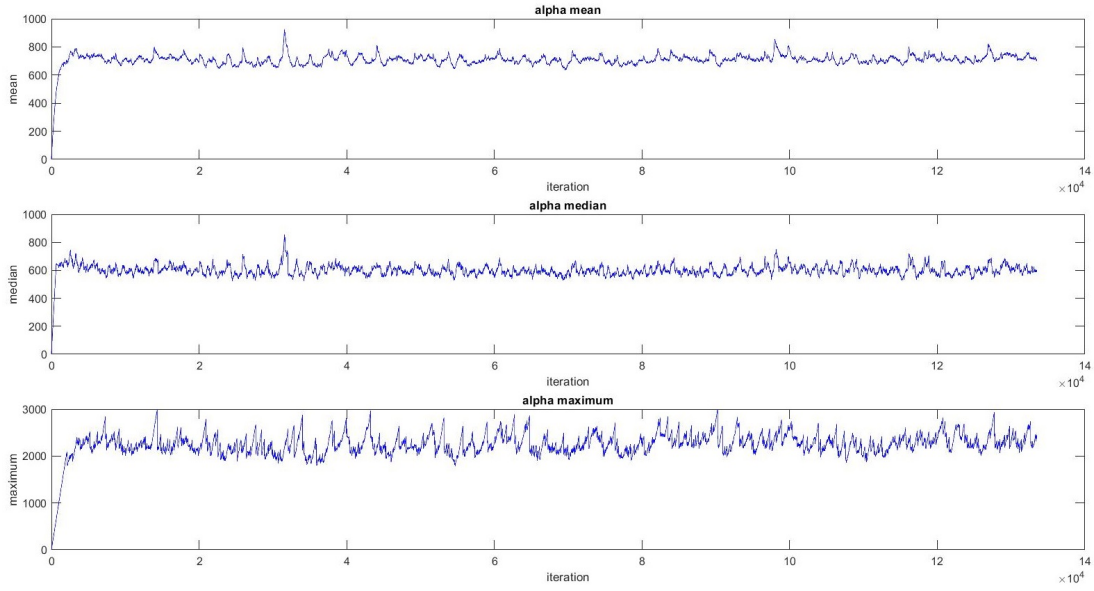


Figure 4.23: Priority with Decentralized algorithm, $T = 1500$, uniform initialization and uniform update

in this case the vehicles have more time to navigate inside the region with a minor waste of time to go toward the meeting point.

- First non-uniform update
  Observing the images about both the α and the time elapsed, Figure 4.24, Figure 4.25 and Figure 4.26, Figure 4.27, the performances are the expected ones. The longer period takes steadier trends.

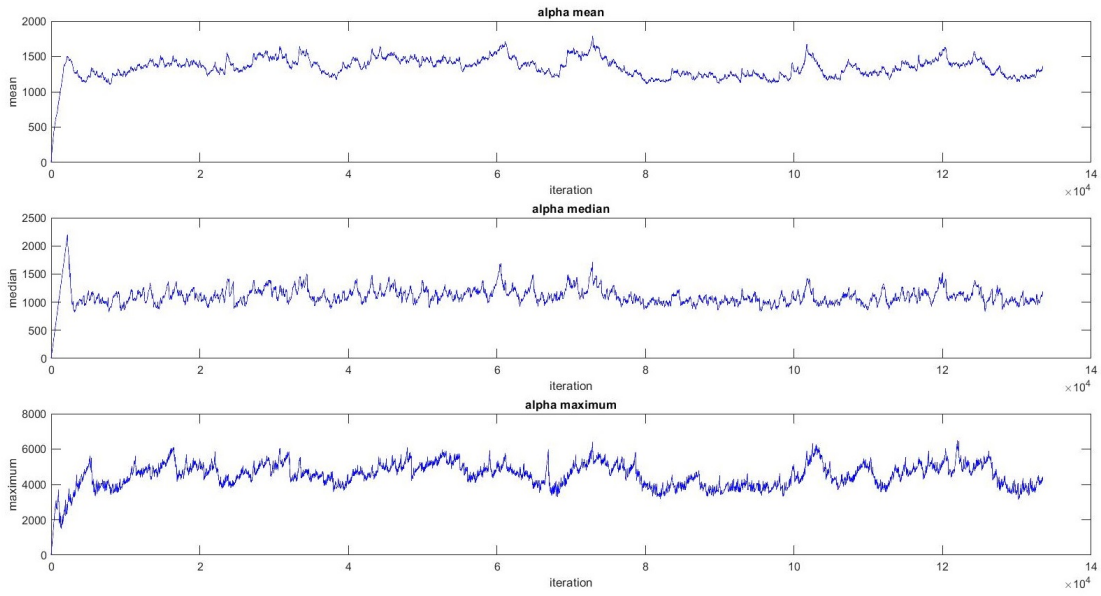


Figure 4.24: Priority with Decentralized algorithm, $T = 850$, uniform initialization and non-uniform update

- Changing non-uniform update
  Even if in this case the $T = 1500$ case becomes less regular the observations of the last point are still valid and it presents better results with respect to a shorter period.

4.3.2 Period 1500 vs Fixed Regions

The superiority of the behaviour of the longer period originates a question: would the performance be better preassigning regions to the vehicles, without performing any communication?
To answer to it the same comparisons of the previous cases in performed.

- Uniform update
  From the uniform update case the decentralized algorithm with meetings demonstrates to have a better behaviour. Even if in the non-communicant case the mean and the median of α are steadier, the maximum values presents higher peaks.

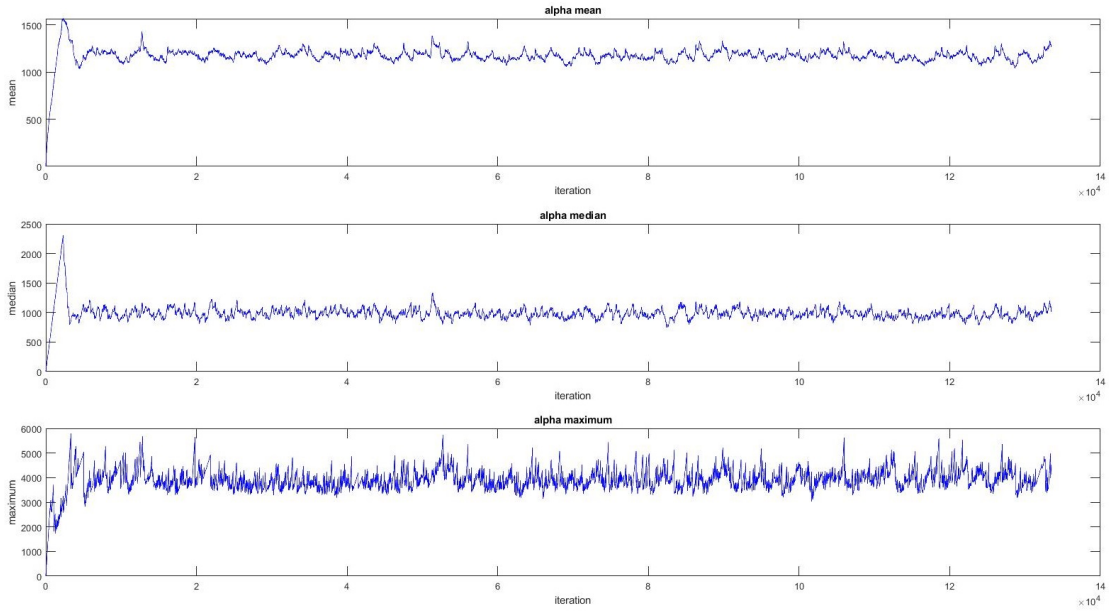


Figure 4.25: Priority with Decentralized algorithm, $T = 1500$, uniform initialization and non-uniform update

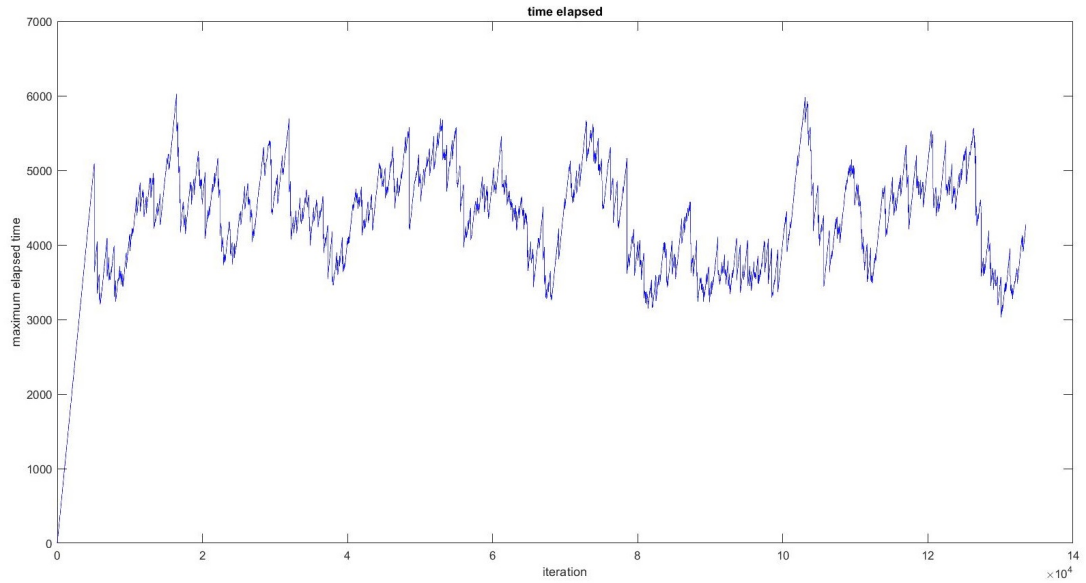


Figure 4.26: Time elapsed with Decentralized algorithm, $T = 850$, uniform initialization and non-uniform update

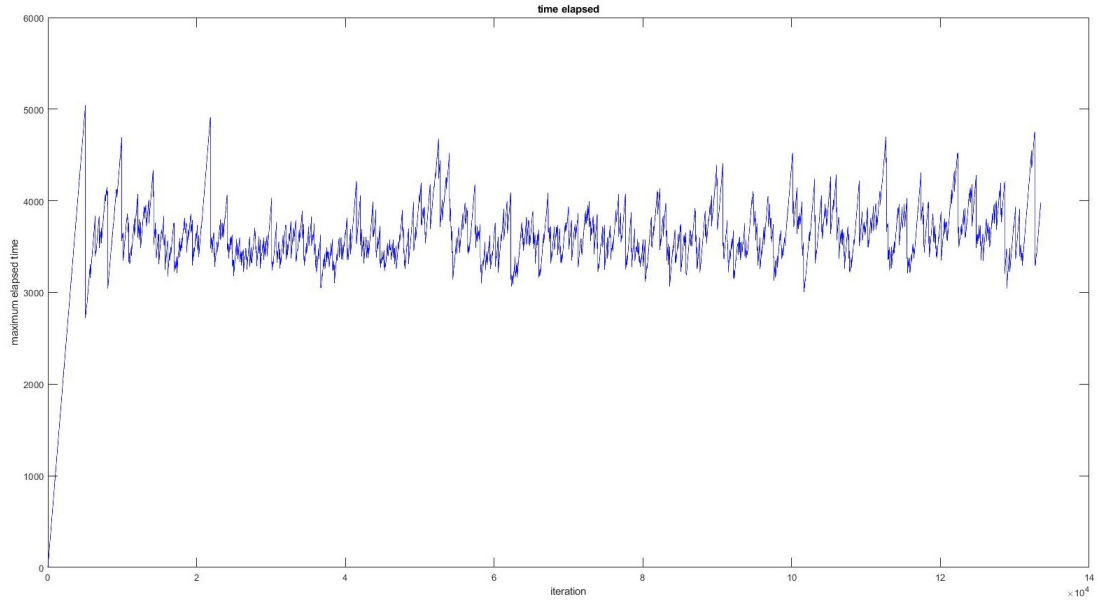


Figure 4.27: Time elapsed with Decentralized algorithm, $T = 1500$, uniform initialization and non-uniform update

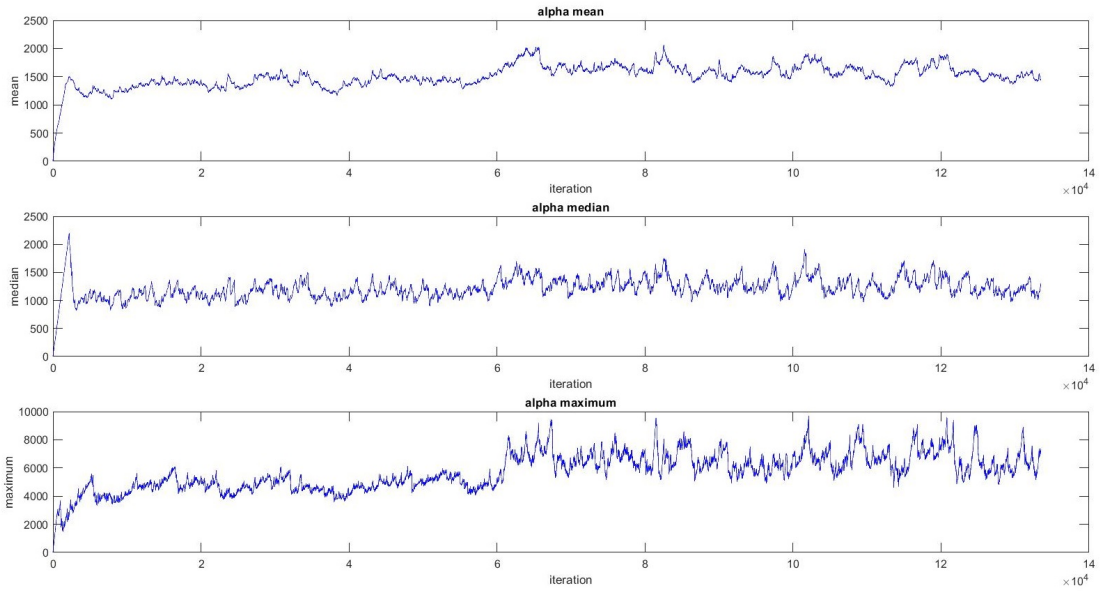


Figure 4.28: Priority with Decentralized algorithm, $T = 850$, uniform initialization and changing non-uniform update

- Non-uniform update

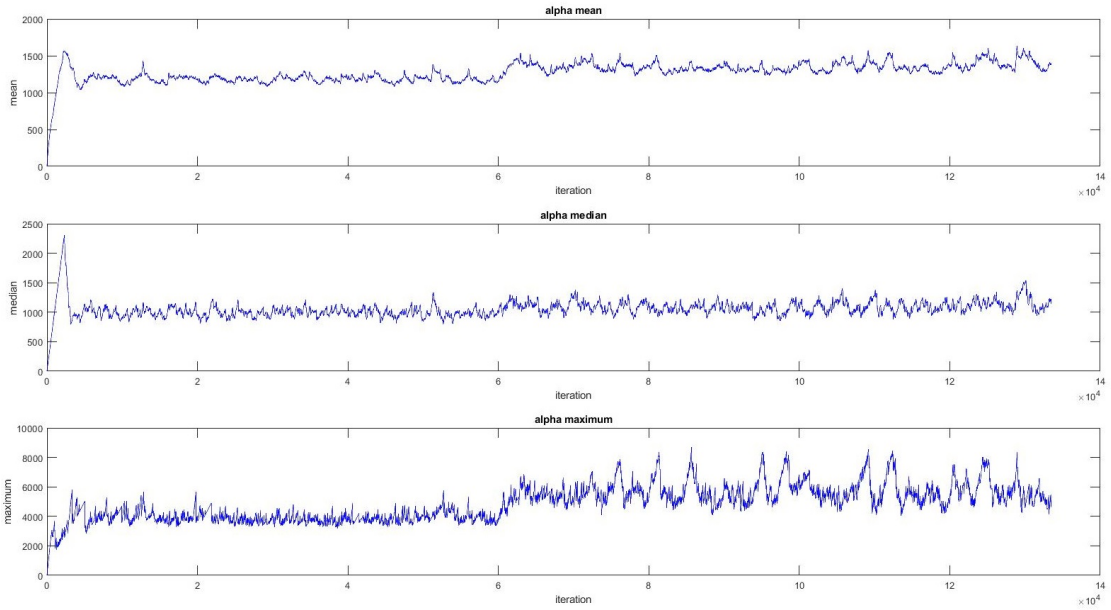


Figure 4.29: Priority with Decentralized algorithm, $T = 1500$, unform initialization and changing non-uniform update

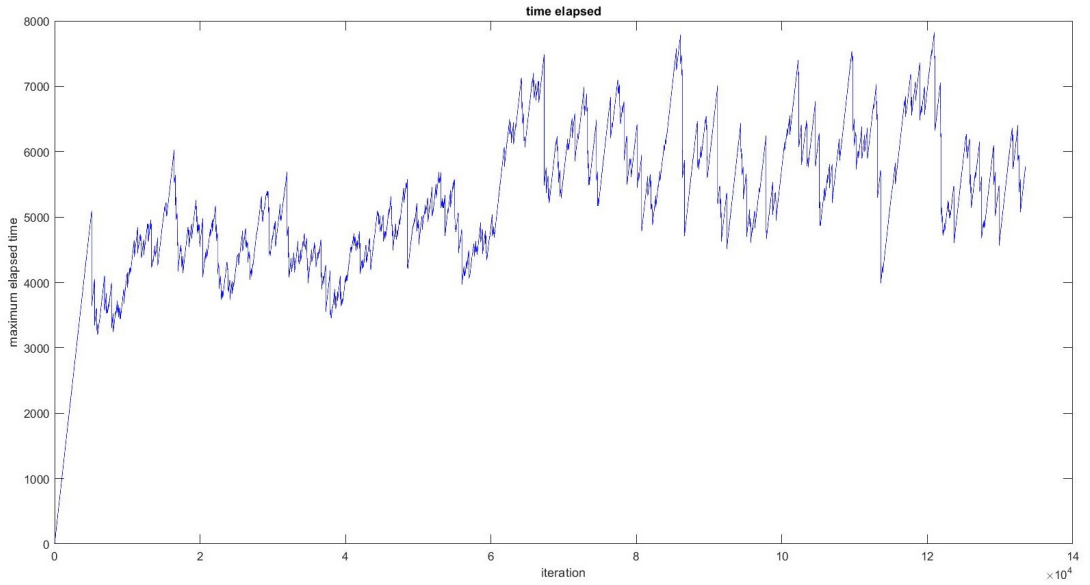


Figure 4.30: Time elapsed with Decentralized algorithm, $T = 850$, uniform initialization and changing non-uniform update

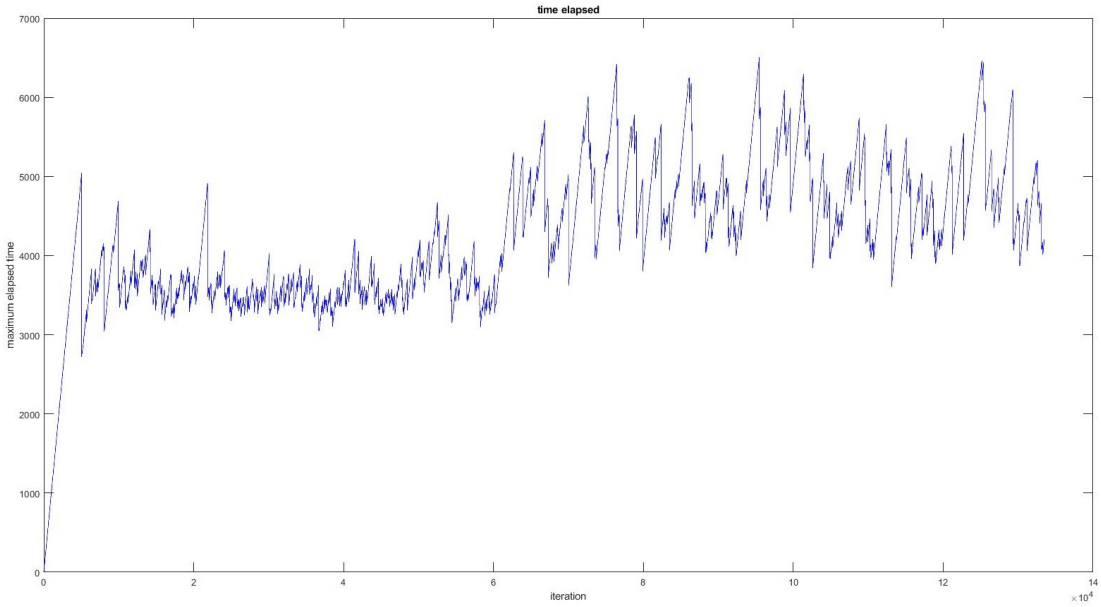


Figure 4.31: Time elapsed with Decentralized algorithm, $T = 1500$, uniform initialization and changing non-uniform update

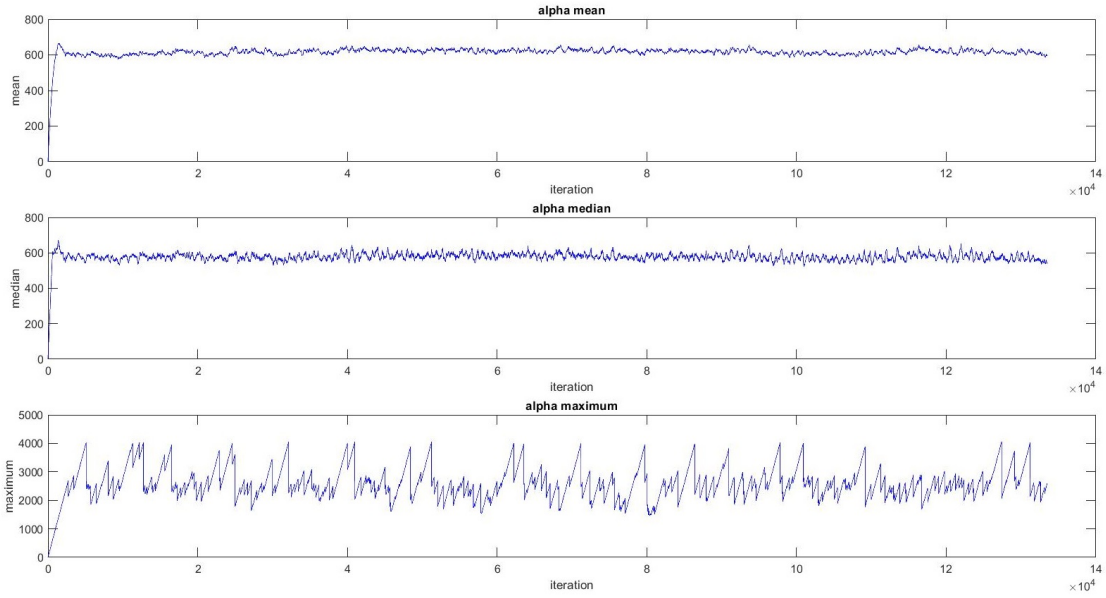


Figure 4.32: Priority with fixed regions, uniform initialization and uniform update

With a non-uniform update the performance of the two navigation methods get closer, with a still more regular trend in case of fixed region. In this case

it is valid also for $\max(\alpha)$, but even though the variance of $T = 1500$ case is greater, it oscillates around a smaller value with respect to the fixed regions case.
The time behaviour shows similar characteristics, with the changing region case that has greater variance and values that fluctuates around a smaller time, that presents less oscillations than the fixed regions case.

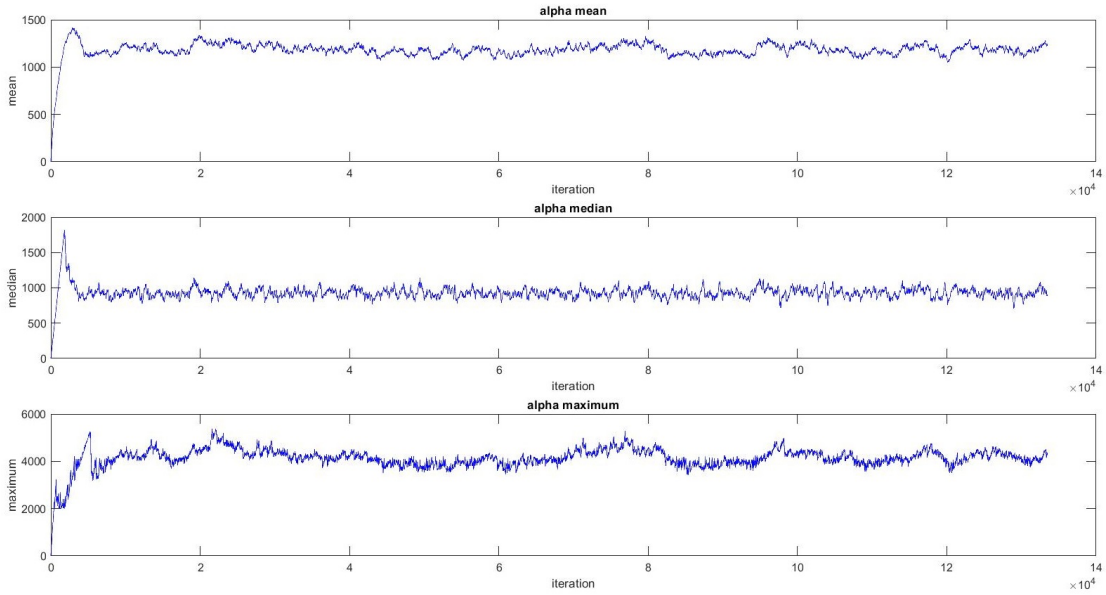


Figure 4.33: Priority with fixed regions, uniform initialization and non-uniform update

- Changing non-uniform update
  Increasing the non-homogeneity in the middle of the simulation gives information about the ability to adapt in the middle of the navigation. While the behaviour of the median is very similar, both maximum and mean have different reactions.
  After the update modification in the fixed region case $\overline{\alpha}$ presents a clear step, increasing the values assumed, in the case of allowed communication it does not happen and the consequence of the modification is the growth of the variance and the slight increase of the mean value.
  About the maximum of α in case of fixed regions the trend is oscillating between 8000 and 6000, while in case of $T = 1500$ the maximum oscillates around 6000 with some peaks that reaches 8000 in correspondence of the meetings.

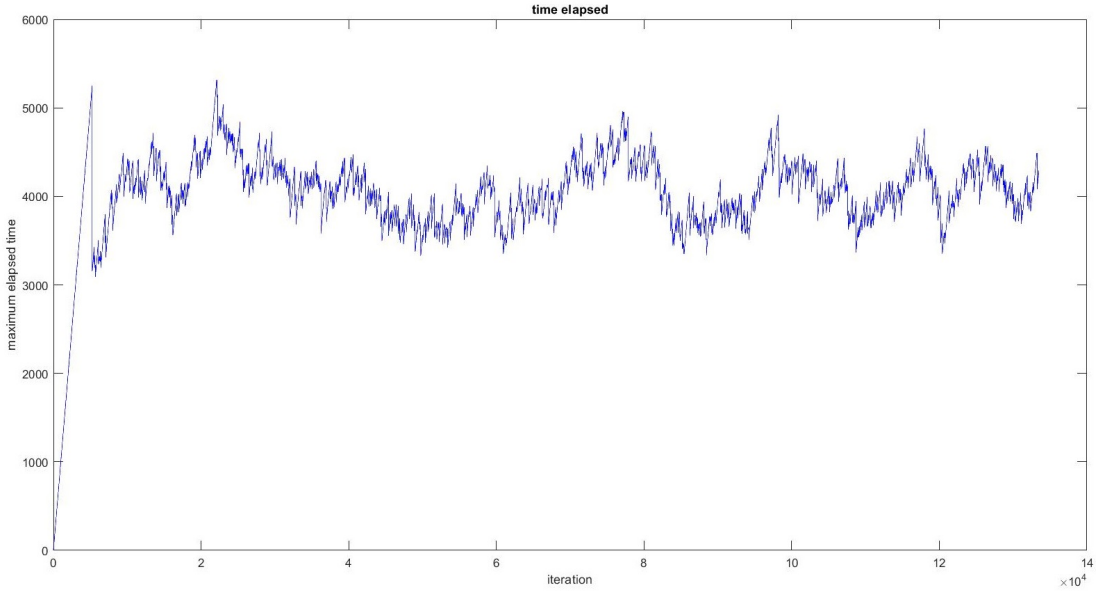


Figure 4.34: Time elapsed with fixed regions, uniform initialization and non-uniform update

Considering then the maximum value of time spent from the previous visit, the variance of the fixed regions case is greater than the changing regions one. In fact in the first case the time between two successive visits can vary from 5000 and 9000 while in the second case it spans from 4000 to 6500.

The comparison between the simulations results shows clearly as also in this last case the adaptability allows to reach a better result. In particular the changing update case demonstrates the fundamental importance of the ability to modify the navigation behaviour as the non-uniformity inside the region increases. It is true not only for the α update, but also for the time spent between two visits that, being inferior without sacrificing the α performance, guarantees a better monitoring of the region.
It is interesting to observe that from the time point of view also the case with $T = 850$ produces a nicer result.

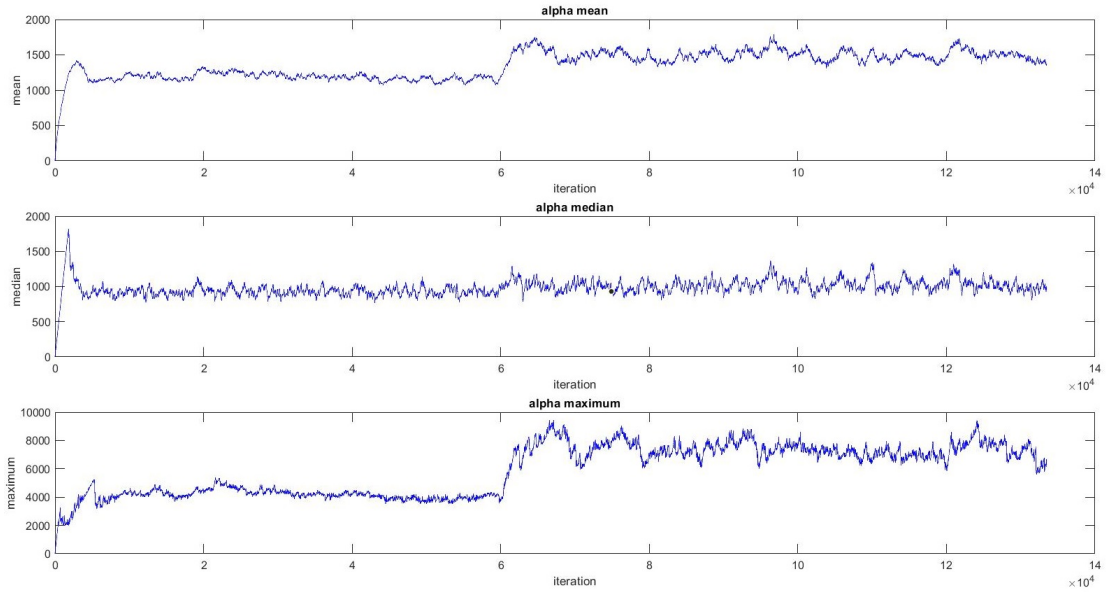


Figure 4.35: Priority with fixed regions, uniform initialization and changing non-uniform update

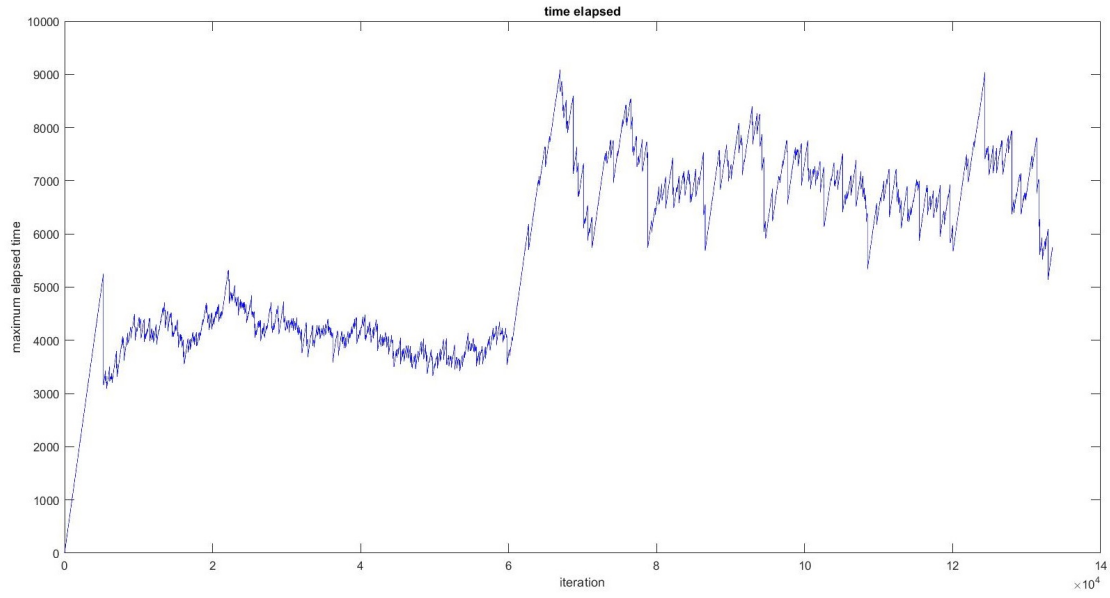


Figure 4.36: Time elapsed with fixed regions, uniform initialization and changing non-uniform update

Chapter 5

Conclusion and Future Work

The present work describes the process of developing a series of algorithms for the autonomous navigation of a fleet of UAVs inside a known region with non-uniform visiting urgency. At first a set of requirements to take into account is written down and then the different algorithms are tested in order to both check the respectfulness of the needs and compare their performance.
Initially to proof the usefulness of the adaptability of the algorithm to a non-homogeneous priority setting the Absolute Maximum Search algorithm performances are compared with the Limit algorithm ones.
Once this test is passed some considerations about the generated trajectory takes to make some modifications on the choice of the next position. In fact trying simply to catch the maximum creates a zig-zag path that worsens the efficiency of the monitoring, since with this navigation method a vehicle that sees two points, one, called A, of priority 80 at distance 5 m and one, called B, of priority 90 at distance 40 m, will cover the points following the sequence $B \rightarrow A$ and not, as it is more logical, $A \rightarrow B$. To avoid this phenomenum the List Maxima algorithm is developed, according to it the next destination to reach is chosen as the closest of the k maximum priority spots inside the region.
Even if in this way the behaviour is improved the tendency to left unsees spots persists, so a method to choose the single next step, and not the next destination, is searched. Due to the necessity to have a compact coverage that would be at the same time adaptable at the changes and would take into account the future steps a Model Predictive Control based algorithm is developed. It shows to behave properly in a small region but, due to the complexity of the objective function, its simulation time in a more complex environment was too long. To try to overcome this problem a simplified version of the MPC algorithm is developed, but in this

way the performance degrades since in the covering process some spots are left non-visited.
Thinking about a real-life implementation a decentralized motion planning control has been developed. It is based on the subdivision of the region with a Lloyd algorithm each period T of time, each sub-region is associated to a vehicle that will visit it in the time between two meetings. During the meetings, the vehicles update their information and, based on it, they will reorganize the navigation as explained above. The influence of the value of T is shown and also the importance that the meetings have in getting a better performance. In the present work the motion inside the sub-regions is performed with a LM algorithm but any other method can be used.

Starting from the navigation methods presented, there are some possible improvements, in particular about the MPC algorithm that is the least effective but the one with the greater potential. Since all the simulations are realized in Matlab it is possible to adapt the proposed optimization problem to an external solver in such a way to create better performances from a computational point of view. Moreover at the moment, the next positions of the vehicles is found applying the MPC to the v vehicles singularly and this creates a deterioration of the performance as the prediction horizon increases, so it would be an interesting development to compute with the same optimization problem all the v next positions. Wanting to remain in the Matlab environment it is possible to try to tune the terms of the objective function in the simplified algorithm in such a way to have variable importance of the local and global terms according to the value of the maximum priority in the region. A similar approach has been tried, imposing the complete elimination of the term regarding the local values of α in case the maximum increases more than an established threshold but this has not taken relevant improvements.
Also the decentralized control method can be improved, in particular about the choice of T, a crucial variable in its functioning. In fact at the moment T is chosen in a heuristic way but to find a mathematical way to find the optimal one or to make it variable and adaptable at the state of the region at the meeting instant would create for sure a better functioning of the algorithm. A similar improvement may be reached also through the allowance of the communication among vehicles also in the interval of time between two meetings and the recomputation of the respective sub-regions of interest.
Moreover, some improvements can be performed from the setting point of view. In the present case the specific constraints required for the motion of a winged vehicle, such as the limit on the minimum curvature radius, are not applied, but to create a control motion directly suitable on a fleet of UAVs are necessary.
A prospect on the algorithms and on the possible future improvements can be found in 5.1.

This work represents a starting point for the navigation inside a non-homogeneous priority region, respecting the deadlines included explicitly or implicitly in the algorithms.

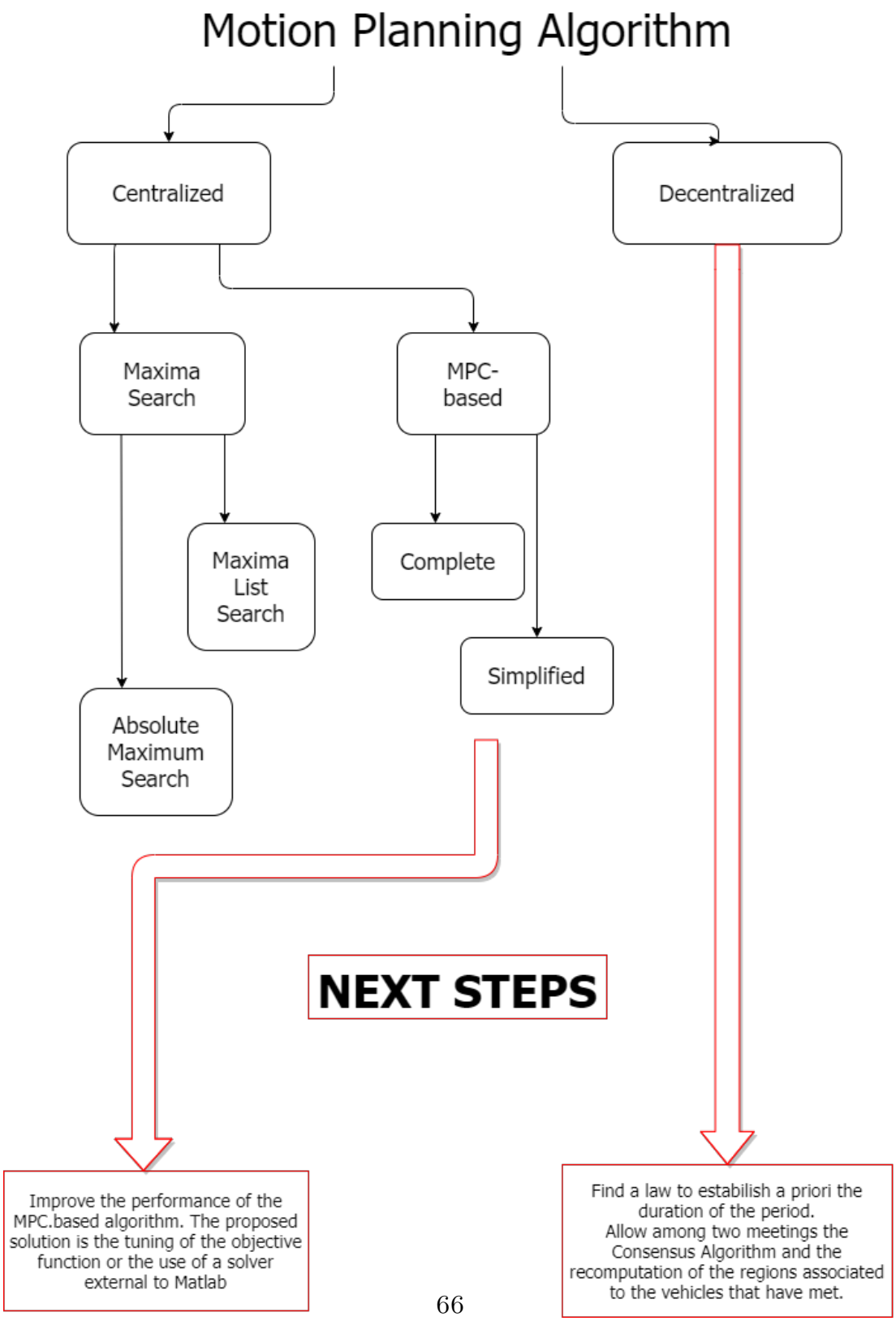


Figure 5.1: Developed algorithms and possible improvements

Bibliography

[1] Jennifer K. Balch, Bethany A. Bradley, John T. Abatzoglou, R Chelsea Nagy, Emily J. Fusco, and Adam L. Mahood. «Human-started wildfires expand the fire niche across the United States». In: *Proceedings of the National Academy of Sciences of the United States of America* 114 (2017), pp. 2946–2951 (cit. on p. 1).

[2] Enric Galceran and Marc Carreras. «A survey on coverage path planning for robotics». In: *Robotics and Autonomous Systems* 61 (Dec. 2013), pp. 1258–1276. DOI: 10.1016/j.robot.2013.09.004 (cit. on p. 5).

[3] Howie Choset. «Coverage of Known Spaces: The Boustrophedon Cellular Decomposition». In: *Auton. Robots* 9 (Dec. 2000), pp. 247–253. DOI: 10.1023/A:1008958800904 (cit. on p. 5).

[4] Chuan Zhu, Chunlin Zheng, Lei Shu, and Guangjie Han. «A survey on coverage and connectivity issues in wireless sensor networks». In: *J. Network and Computer Applications* 35 (Mar. 2012), pp. 619–632 (cit. on p. 6).

[5] H. Zhang and J.C. Hou. «Maintaining Sensing Coverage and Connectivity in Large Sensor Networks». In: *Ad Hoc Sensor Wirel. Netw.* 1 (Nov. 2004) (cit. on p. 6).

[6] Teddy M. Cheng, Andrey V. Savkin, and Faizan Javed. «Decentralized Control of a Group of Mobile Robots for Deployment in Sweep Coverage». In: *Robot. Auton. Syst.* 59.7–8 (July 2011), pp. 497–507. ISSN: 0921-8890 (cit. on p. 7).

[7] G. A. Hollinger and S. Singh. «Multirobot Coordination With Periodic Connectivity: Theory and Experiments». In: *IEEE Transactions on Robotics* 28.4 (2012), pp. 967–973 (cit. on p. 7).

[8] Garret Van Ryzin Dimitris J. Bertsimas. «A Stochastic and Dynamic Vehicle Routing Problem in the Euclidean Plane». In: *Oper. Res.* 39 (1991), pp. 601–615 (cit. on p. 7).

[9] Stephen Smith, Marco Pavone, Francesco Bullo, and Emilio Frazzoli. «Dynamic Vehicle Routing with Priority Classes of Stochastic Demands». In: *SIAM J. Control and Optimization* 48 (Aug. 2010), pp. 3224–3245 (cit. on p. 7).

[10] M. Pavone, N. Bisnik, E. Frazzoli, and V. Isler. «A Stochastic and Dynamic Vehicle Routing Problem with Time Windows and Customer Impatience». In: *Mob. Netw. Appl.* 14.3 (June 2009), pp. 350–364. ISSN: 1383-469X. DOI: 10.1007/s11036-008-0101-1. URL: https://doi.org/10.1007/s11036-008-0101-1 (cit. on p. 7).

[11] Dimitris J. Bertsimas and Garrett Van Ryzin. «Stochastic and dynamic vehicle routing with general demand and interarrival time distributions». In: *Advances in Applied Probability* 25.4 (1993), pp. 947–978 (cit. on p. 7).

[12] Harilaos Psaraftis, Min Wen, and Christos Kontovas. «Dynamic Vehicle Routing Problems: Three Decades and Counting». In: *Networks* 67 (Aug. 2015). DOI: 10.1002/net.21628 (cit. on p. 7).

[13] K. Savla, E. Frazzoli, and F. Bullo. «Traveling Salesperson Problems for the Dubins Vehicle». In: *IEEE Transactions on Automatic Control* 53.6 (2008), pp. 1378–1391 (cit. on p. 7).

[14] Citation Enright, Ketan Savla, and Emilio Frazzoli. «Stochastic and Dynamic Routing Problems for Multiple Uninhabited Aerial Vehicles». In: *Journal of Guidance, Control, and Dynamics* 32 (July 2009), pp. 1152–1166. DOI: 10.2514/1.41616 (cit. on p. 7).

[15] Stuart P. Lloyd. «Least squares quantization in PCM». In: *IEEE Trans. Inf. Theory* 28 (1982), pp. 129–136 (cit. on p. 9).

[16] J.A. Bondy and U.S.R Murty. *Graph Theory*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 1846289696 (cit. on p. 12).