POLYTECHNIC OF BARI

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING

AUTOMATION ENGINEERING MASTER'S DEGREE

Course on DYNAMICAL SYSTEMS THEORY

Prof. Engr. Mariagrazia DOTOLI
Dr. Engr. Raffaele CARLI
Dr. Engr. Graziana CAVONE
Engr. Paolo SCARABAGGIO

Project work:

Consensus-based Algorithms for Controlling Swarms of Unmanned Aerial Vehicles

Students:
Antonio NIRO
Antonio VENEZIA

ACADEMIC YEAR 2019-2020

Abstract

In this work the main features of swarm intelligence strategies have been implemented, using a discrete time single integrator model. In particular aggregation, flight formation control, swarm tracking and social foraging behaviors have been analyzed. The main objective is to obtain a swarm able to achieve a predefined geometric formation, maintain it during movement, avoid collisions with obstacles and among swarm members, reach a target or follow a trajectory. In the first part, we developed a strategy for converging the members of the swarm into a single point. Then we implemented four examples of geometric formations, where a distance is predefined for each pair of UAV. The number of drones per formation has been increased, in order to test the algorithms on different complexity levels. The last part of the work is focused on moving the entire swarm with a 'leader-following' strategy. We analyzed two cases. In both of them a leader is chosen among all the members. In the first case the leader knows the position of a predefined target, and tries to minimize its distance from it. The entire swarm has to follow the leader, trying to keep the formation. In the second case, instead, the leader doesn't have a target to reach, but a trajectory to follow. The behaviour of the remaining members remains the same as in the first case.

Contents

1 Introduction

Recently there has been an increasingly interest in researching for control techniques for a considerable number of unmanned autonomous vehicles for the most disparate applications, from satellite control to military vehicles control. Among the various control techniques, distributed control stands out with the aim of achieving a common objective by using local information. In this study the performances of this type of approach have been analyzed, by contextualizing them in the field of UnManned Aerial Vehicles (UAVs). Few hypothesis have been considered:

- each UAV has limited resources in terms of communication coverage so that it can only receive information locally from a subset of the swarm;
- each UAV has a low-level controller that guarantees that the agent will move to the next point in space. The controller has not been treated.

The paper is organized as follows: in Chapter 1 the advantages and peculiarities of the distributed control applied to Swarm Intelligence are presented, in Chapter 2 the theoretical aspects used are illustrated, in Chapter 3 the results obtained from the simulations are illustrated and analyzed.

2 Preliminaries on Swarm Intelligence Problems

Swarm Intelligence algorithms are based on the imitation of existing behaviors in nature. For example, considering the behavior of animals or insects, they are generally organized such that the decisions of an individual depend on the behavior of other members: each individual acts as a data harvester in the surrounding environment so that the whole swarm can make the right decision. Sometimes it is possible that the whole swarm is dependent on the decisions of a leader. A swarm algorithm should take into account the basic behaviors of a swarm such as: Aggregation, Flight Formation, Social Foraging[1].

2.1 Aggregation

It is the swarm's basic behavior. Aggregation is the ability of the swarm to work and organize itself, moving towards a specific target, avoiding collisions among components of the same swarm and obstacles that may be present along the way. In a theoretical point of view, it is possible to reach a situation where all the members converge in a single point. Of course this is not possible in real applications, and a safety inter-agent distance must be specified to avoid collisions.

2.2 Flight Formation

It consists in the ability of the swarm to create particular geometric figures starting from a disordered configuration of the swarm itself, and it is useful for Social Foraging process. In a typical formation acquisition task, the scale of the predefined geometric pattern (i.e., the distances between the nodes in the geometric pattern indicating the desired final positions of agents relative to each other) may or may not be specified. In this work we consider the problem of formation stabilization and achievement of a predefined geometric shape with predefined inter-agent distances.

2.3 Swarm Tracking

The union of flight formation and aggregation is the swarm tracking, which is the swarm's ability to follow a mobile target while maintaining the formation. Keeping a swarm formation is important for many aspects. For example, squadrons of military planes can save fuel by flying in a V formation, and many scientists suspect that migrating birds do the same. Furthermore, in some applications, it could be important to limit the size of the formation, and this can be done by adopting geometrical formations.

2.4 Social Foraging

Social foraging has many advantages, one of which is increasing the probability of success for the individuals in the swarm. It consists in the ability to distinguish favorable and dangerous transit areas in the surrounding environment. From an engineering point of view, a favorable region can be a target to reach, and a dangerous one can represent an obstacle to avoid. There are multiple techniques the can be used to achieve this. For example the 'leader-following' approach, supposes that the entire swarm follows a particular member (leader), that knows exactly where the target or the interesting area is.

3 Definition of Distributed Control Algorithms

The approach is based on a discrete time single integrator model where consensus algorithm and distributed system have been considered. More in depth, it has been assumed that each UAV could update its own position gathering information about the position of a subset of the entire UAVs swarm. This subset, called 'neighborhood', is defined, for each node (UAV) of the graph, by all the nodes connected to it. A connection is created if the distance between UAVs is lower than the hypothesized communication range. Three typical architectures for control systems are shown in Figure 1.

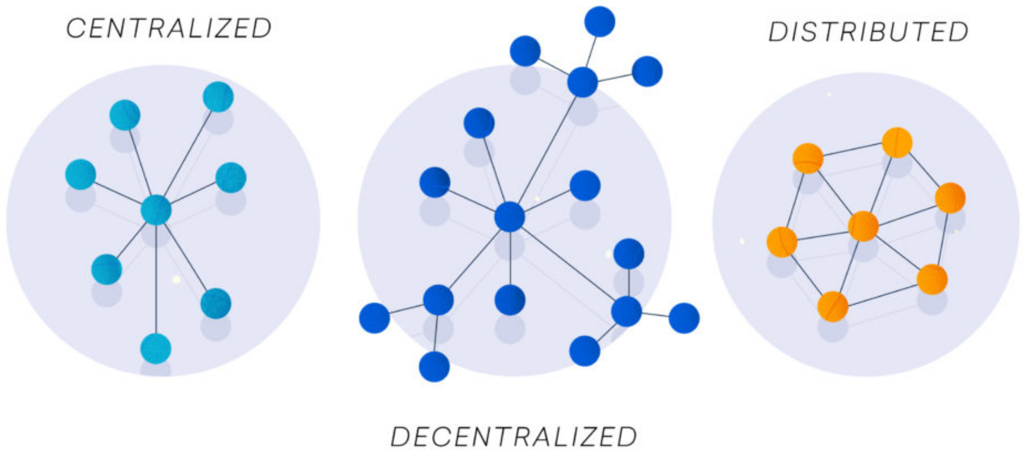

Figure 1: Comparison between centralized, decentralized and distributed architectures

During the movement of the members, the representative swarm graph may change. It's obvious that if there is a node without edges (because it's too far from anyone in the swarm), the process can't be successfully completed. The same happens if the agent positions produce two or more isolated graphs. In the best hypothesis, the swarm is represented by a complete graph. This means that each drone is close enough to any other member of the swarm, and each node of the graph is connected to all the others. However, this situation is not an interesting case of study, because consensus algorithms are not needed to reach the required goals. The situation analyzed, instead, is the one in which the swarm is represented by a connected graph. In this case there is always at least a path between every pair of vertices. The graph theory is fundamental to obtain a predefined geometrical swarm formation. The goal is to reach the formation such that the representative graph is a complete one. The initial swarm configuration has to be represented by a connected graph, because otherwise an average consensus between members can't be reached. Once the geometrical formation has been achieved, the swarm can start moving to the target. In this work it is used a 'leader-following' based algorithm, in which a member of the swarm knows exactly where the target is, and it starts moving in that direction. Alternatively it is possible to define a trajectory that will be followed by the swarm leader and, consequently, by all the other members. During the movement the entire swarm tries to maintain the formation, even if it can be modified due to any obstacles present along the way.

3.1 Modeling of the Swarm of UAVs

The agent model consists of a discrete-time single integrator model:

$$x_i(k+1) = x_i(k) + u_i(k), \qquad \forall k \in T_i \tag{1}$$

where $x_i(k), i = 1...N$ represents the state (position) of agent i at time k, $u_i(k)$ the control input and T_i the set of time instants at which agent position is updated. In order to satisfy discrete-time modelling, the following assumption is considered:

$$x_i(k+1) = x_i(k), \qquad \forall k \notin T_i, i = 2, ..., N-1 \tag{2}$$

In other words, when agent i position is not updated, it is considered stationary. Single agent is represented by a fictitious sphere where the radius ϵ_r identify the sensing range given by embedded proximity sensors in order to detect potential collisions with other agents and obstacles. A single integrator model is not able to control velocity and acceleration (features of double integrator model), that implies a non-perfect formation control in an environment with many obstacles, so we suppose this control in a safety zone, where no obstacles are present.

3.2 Control Algorithms for Aggregation

A solution to the problems described above can be achieved by using the consensus algorithm, which is well suited to distributed architectures. The distributed consensus problem deals with reaching agreement among a group of processes connected by an unreliable communications network. The average consensus is achieved if the differences between the values shared by members of the swarm are as close to zero as possible. An agent can't communicate with all the others, but only with a subset of the entire swarm. Given a swarm of N UAVs randomly placed in space, the position of $i-th$ agent $x_i \in R^3$, according to consensus algorithm[2], is defined as follow:

$$x_i(k+1) = x_i(k) - \frac{\sum_{j=1}^{dim(V_i(k))} x_i(k) - x_{v_j(k)}}{dim(V_i(k))} \tag{3}$$

where $V_i(k)$ is the set of the i-th agent's neighbors v_j, that can be varying according to the topology of the system at time k. Equation 1 represents a simple implementation of the consensus algorithm.

3.3 Control Algorithms for Flight Formation

Flight formation' principle is based on the idea of defining polynomial potential function for each agent such that targets and obstacles constitute the zeros of these functions. More in detail, an iterative algorithm which generates agent's positions, is used to achieve predefined distances between UAVs based on the negative gradient of the target function according to the formation geometries, including collision avoidance based on the positive gradient of the obstacles function. Flight formation control can be synthesized into three main steps. For the $i-th$ agent:

- defining the desired position related to the neighbors positions, according to the formation's geometries;
- defining the target and obstacle polynomial potential function, according to the target/obstacles positions;
- updating agent position by using Newton's iterative method.

By considering:

$$d_{ij} = ||x_i - x_j||$$

the inter-agent desired distance between the $i-th$ and $j-th$ UAV, the set of target t_j for the $i-th$ agent is defined as follow:

$$H_T^i(k) = \bigcup_{j=1,\dots N, j\neq i} H_T^{ij}(k) \tag{4}$$

where

$$H_T^{ij}(k) = \{t_j(k) = x_j(k) + \frac{d_{ij}}{||x_i(k) - x_j(k)||}, j \neq i\} \tag{5}$$

Collision avoidance is guaranteed if agents are considered as obstacles by the $i-th$ UAV. The set of obstacles o_j for the $i-th$ agent is defined as follow:

$$H_O^{ij}(k) = \{o_j(k) = x_j(k), j \neq i\} \tag{6}$$

Each set defined above for the $i-th$ agent includes only one target and one obstacle, respectively. Given the set H_T, an attraction potential function $F_T : R^n -> R^n$ is defined as[3]:

$$F_T(y) = \begin{cases} f_1(y) \\ f_2(y) \\ . \\ . \\ f_n(y) \end{cases} \tag{7}$$

such that, for each $t_i \in H_T(y)$, $F_T(t_i) = 0$. For simplicity a two-dimensional coordinates frame is considered such that the position of the $i-th$ agent is defined as:

$$x_i = [x_{1i} \;\; y_{2i}]^T \tag{8}$$

and the position of the $j-th$ target as:

$$t_j = [t_{1j} \;\; t_{2j}]^T \tag{9}$$

The potential attraction function is:

$$F_T^{ij}(x_i(k)) = \begin{cases} x_{1i}(k)x_{2i}(k) - x_{1i}(k)t_{2j}(k) + x_{1i}(k) - x_{2j}(k) \\ x_{2i}(k) - t_{2j}(k) \end{cases} \tag{10}$$

As it can be seen, $F_T = 0$ if $x_i(k) = t_j(k)$. The zero of this function can be achieved with the Newton's method:

$$x_i(k+1) = x_i(k) + \lambda \Delta x_a(k) \tag{11}$$

where λ is a step size defining the attraction coefficient, and

$$\Delta x_{ai}(k) = \sum_{j=1, j\neq i}^{N} \frac{A_{ij}(k)}{||A_{ij}(k)||} \tag{12}$$

where

$$A_{ij}(k) = -[\nabla F_T(x_i(k))]^{-1} F_T^{ij}(x_i(k)) \tag{13}$$

Collision avoidance can be taken in account with similar considerations. Repulsion function $F_O : R^n -> R^n$ is defined as[3]:

$$F_O(y) = \begin{cases} f_1(y) \\ f_2(y) \\ . \\ . \\ f_n(y) \end{cases} \tag{14}$$

By defining the $j-th$ obstacle position as:

$$o_j = [o_{1j}(k) \;\; o_{2j}(k)]^T \tag{15}$$

the repulsion function for the $i-th$ agent and $j-th$ obstacle is:

$$F_O^{ij}(x_i(k)) = \begin{cases} x_{1i}(k)x_{2i}(k) - x_{1i}(k)o_{2j}(k) + x_{1i}(k) - o_{1j}(k) \\ x_{2i}(k) - o_{2j}(k) \end{cases} \tag{16}$$

The position of the $i-th$ agent can be updated as follow:

$$x_i(k+1) = x_i(k) + \lambda(\Delta x_a(k) + \Delta x_r(k)) \tag{17}$$

where:

$$\Delta x_r(k) = \sum_{j=1, j\neq i}^{N} \bar{R}_{ij}(k) \tag{18}$$

with

$$\bar{R}_{ij}(k) = \begin{cases} \frac{1}{(1+(\frac{||R_{ij}(k)||}{C_r})^{\mu})||R_{ij}(k)||^3} R_{ij}(k) - \frac{\epsilon_r}{(1+(\frac{\epsilon_r}{C_r})^{\mu})\epsilon_r^3}, & ||x_i(k) - x_j(k)|| < \epsilon_r \\ 0, & ||x_i(k) - x_j(k)|| > \epsilon_r \end{cases} \tag{19}$$

and

$$R_{ij}(k) = +[\nabla F_O(x_i(k))]^{-1} F_O(x_i(k)) \tag{20}$$

C_r and μ define the coefficient of the repulsion component, according to the safety inter-agent distance ϵ_r.

In order to improve flight formation's convergence, an adaptive step size λ_i for the $i-th$ agent can be defined. By assuming:

$$\epsilon(k) = \frac{\sum_{j=1, j\neq i}^{dim(V_i(k))} ||d_{ij} - (x_i(k) - x_j(k))||}{dim(V_i)} \tag{21}$$

as the error relative to the desired inter-agent distance formation at time k, the adaptive step size shape is:

$$\lambda_i(k+1) = pe^c \tag{22}$$

where p is a tuning coefficient and

$$c = \epsilon(k) - \epsilon(k-1) \tag{23}$$

Figure 2 shows the trend of the adaptive step size in relation to the error and the tuning coefficient p,

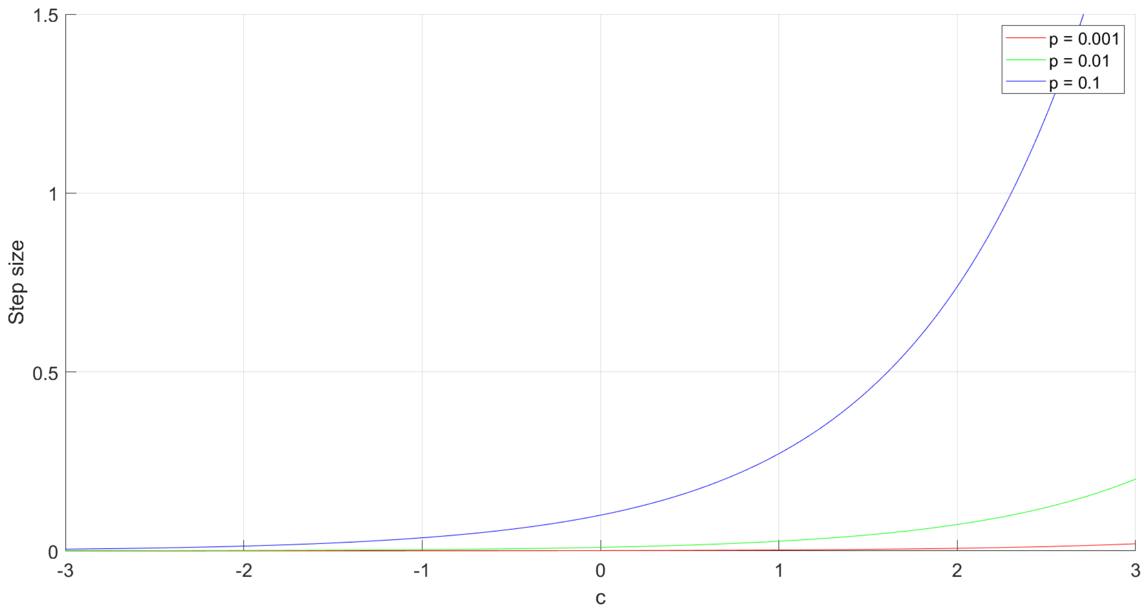

Figure 2: Adaptive step size

3.4 Control Algorithms for Swarm Tracking and Social Foraging

Swarm tracking and social foraging can be modelled by considering the same concepts shown in par. 3.1 and 3.2. More in depth, swarm tracking is based on selecting a leader whose attraction potential function depends on a set of targets corresponding to the points of the trajectory. Meanwhile, the other agents maintain the flight formation according to par. 3.2. Social foraging can be modelled by considering either a subset of regions of the flight space as obstacles (and the corresponding repulsion function) and flight formation algorithm.

4 Simulations Results and Discussion

The main aims of this work can be summarized in three key points:

1 Achievement of a predefined formation.

2 Start of the movement towards the chosen target.

3 Collision avoidance.

Since we use a discrete time model, note that after each iteration during swarm movement, collision avoidance function is called up. This means that, after the target has been chosen, the leader starts moving on the right directions avoiding obstacles along the way. Simultaneously, all the other members have to follow it keeping the formation as much as possible, and have to avoid obstacles too.

4.1 Aggregation

According to the considerations in section 3.1, aggregation has been implemented. As can be seen in Figure 3, the swarm takes 200 iterations to complete aggregation with a step size λ=0.05 common for all the agents.

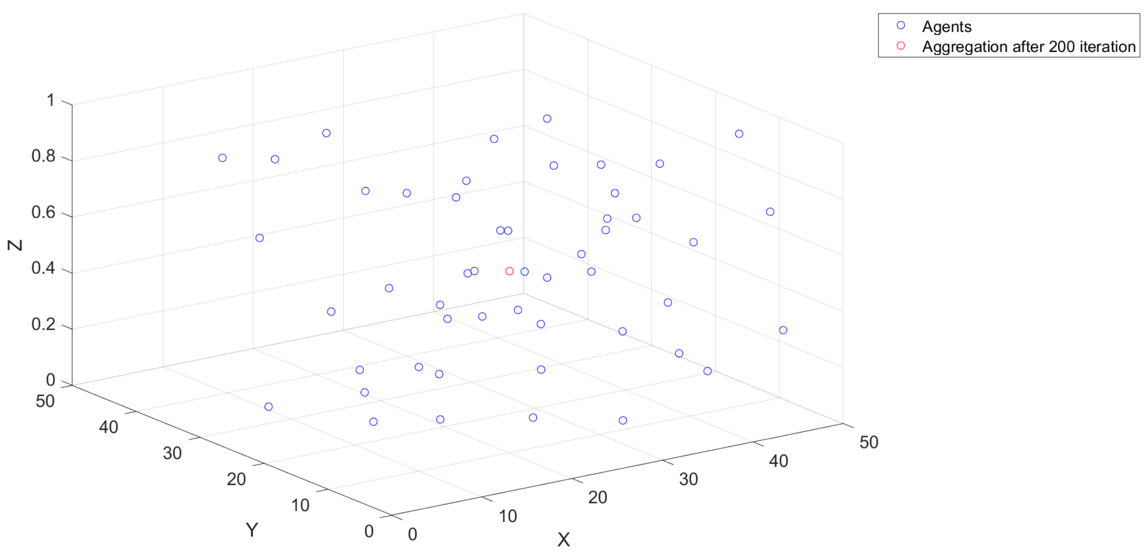

Figure 3: Aggregation

For our purposes, this algorithm is used to obtain the same flight altitude for all drones, in order to have an easier flight formation control process.

4.2 Implemented formations

To achieve a geometric formation, it is necessary to specify the desired distance for each pair of drones. The initial positions of the swarm members are randomly generated. Given all the positions, a graph representing the current situation has been obtained. Each UAV knows its distance from all the UAVs in its neighborhood, and compares it with the desired distance. A visual representation of the average error trend is shown in Figure 4.

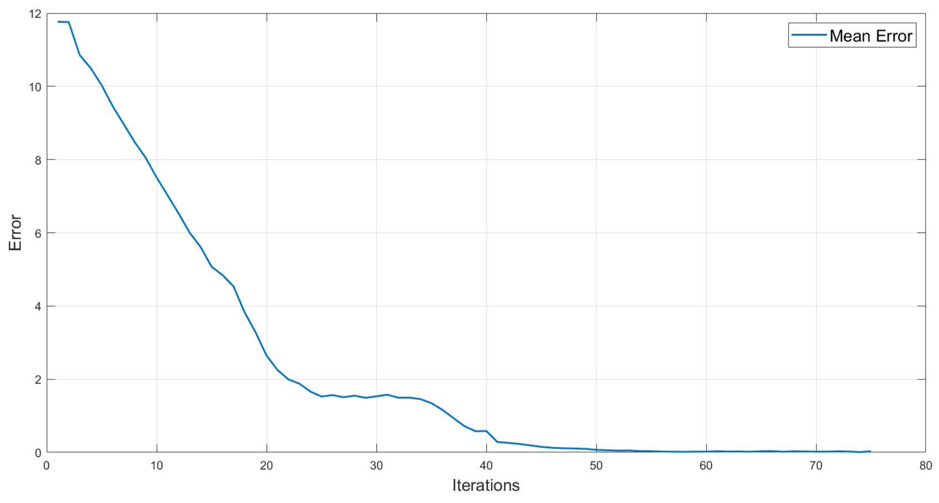

Figure 4: Average error

After each iteration, an average formation error is computed, in order to understand when each swarm member is in the correct position. Each UAV contributes to the error computation only with the information obtained from its neighbors. All the formations obtained are shown below.

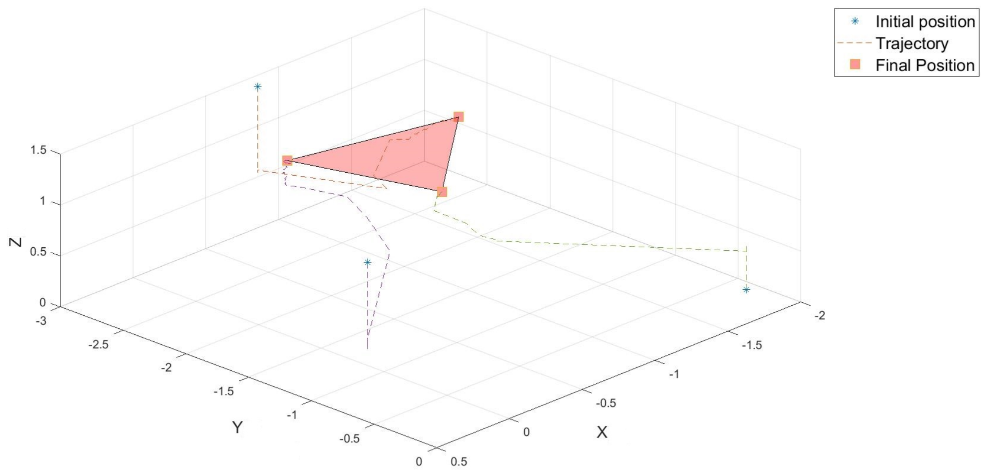

Figure 5: Triangle formation

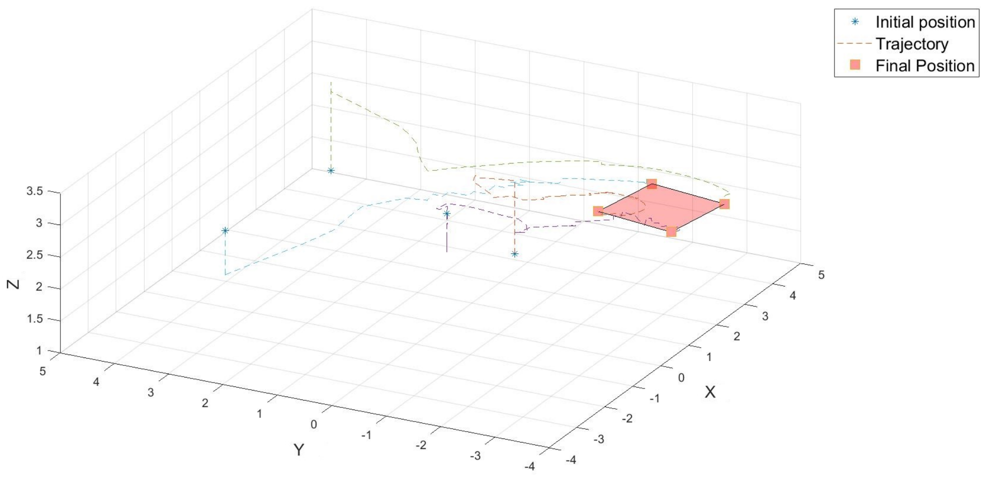

Figure 6: Square formation

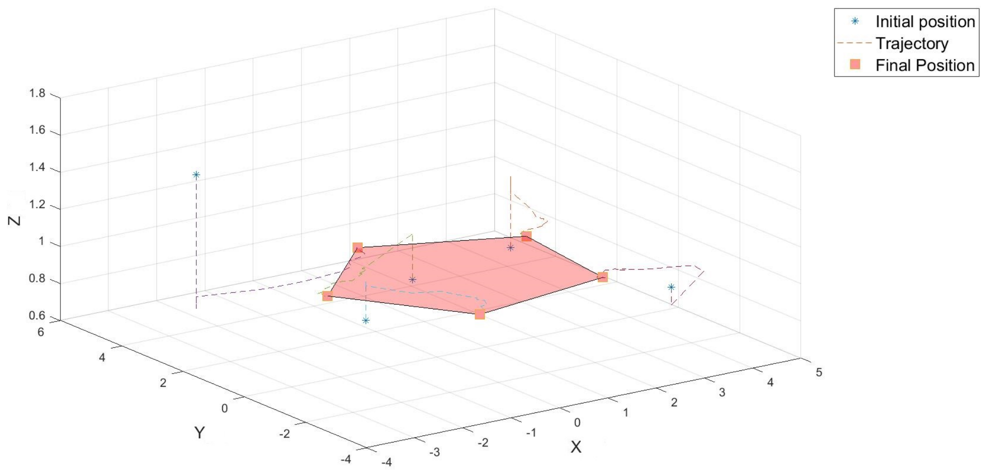

Figure 7: Pentagon formation

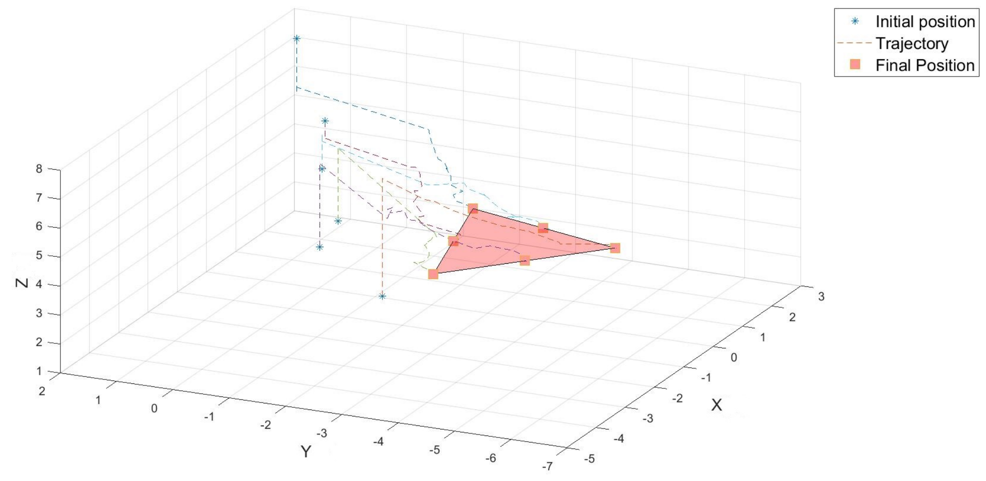

Figure 8: 'Six drones triangle' formation

4.3 Swarm tracking

It's possible to assign a predefined trajectory to the leader such that all the members of the swarm will follow the same trajectory remaining in formation as much as possible. This is useful in cases where the environment is perfectly known, and we can create a well defined trajectory in order to avoid obstacles and reach a target as soon as possible. In Figure 9 it's shown an example where the leader follows a sinusoidal trajectory.

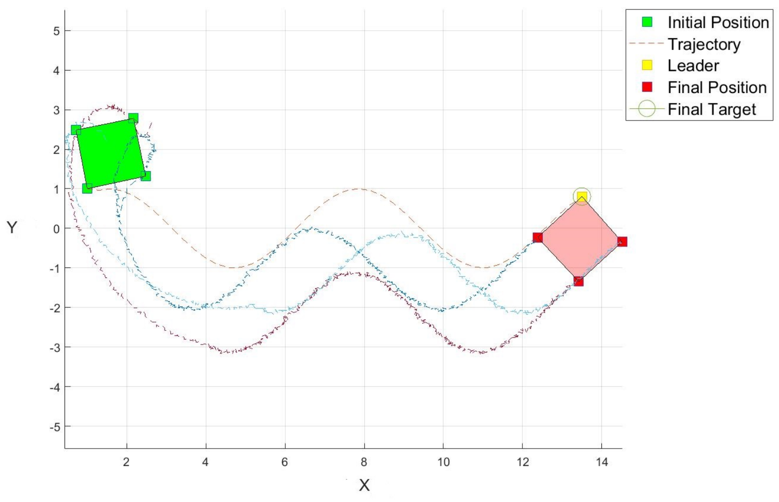

Figure 9: Sinusoidal trajectory

4.4 Social foraging

Once the geometric formation has been achieved, the swarm can start moving. Everything is ruled by the swarm leader, that knows exactly where the target is, but doesn't have a predefined trajectory. The leader tries to minimize its distance from the target, regardless of the position of the other members of the swarm. Sometimes it is not possible to perfectly maintain the formation, because of obstacles. In these cases, each member must first of all avoid the obstacles and the other members of the swarm. Once the leader has reached the target, the formation is formed again, keeping leader's position fixed. This situation is shown in Figure 10

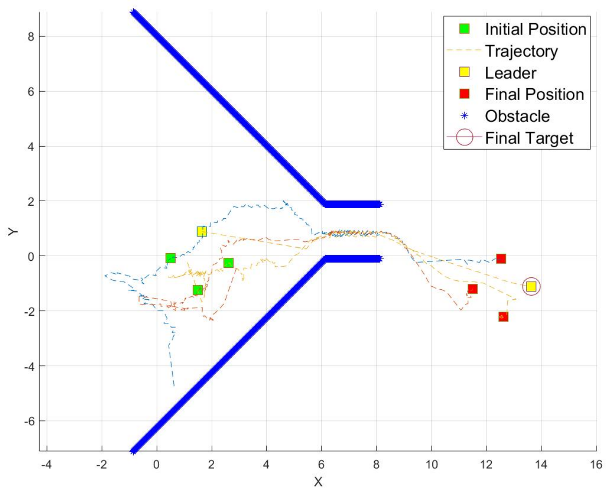

Figure 10: Target achievement

5 Conclusions and Future Works

Studying swarms in nature has allowed researchers to establish that collaborative swarming behavior that we observe in these groups provides survival advantages. Lots of these strategies are pretty adaptable to engineering applications and, nowadays, they can be implemented at a lower cost than in the past. Approaches that focus on enabling collectives of devices to operate semi-autonomously are also increasing in relevance, because of the developing of new networks infrastructure (such as 5G), that allows us to exchange data in a faster and more efficient way. At the same time, computational power of autonomous vehicles has grown too, allowing the use of increasingly complicated algorithms. With this work we analyzed techniques that could be physically implemented on real autonomous vehicles, using some simple aspects coming from the study of dynamical systems theory. More in depth, it has been shown that simple swarm behavior such as aggregation using consensus approach, is useful to accomplish complex tasks like formation control and social foraging. Collision avoidance implementation is mandatory for a real application in order to prevent inter-agent and obstacle collisions.

Further implementations should analyze a double integrator model in order to include control in acceleration and velocity, according to the dynamics of the agents.

References

[1] Elmer P. Dadios Argel Bandala Laurence A. Gan Lim. “Swarming Algorithm for Unmanned Aerial Vehicle (UAV) Quadrotors: Swarm Behavior for Aggregation, Foraging, Formation, and Tracking”. In: *Journal of Advanced Computational Intelligence and Intelligent Informatics* 18.5 (2014), pp. 745–751.

[2] Randal W. Beard Wei Ren. *Distributed Consensus in Multi-vehicle Cooperative Control - Theory and Applications*. Springer, 2008.

[3] Kevin M. Passino Veysel Gazi. *Swarm Stability and Optimization*. Springer, 2011.

A Appendix: Matlab Code

The following sections show main the implementations of the proposed algorithms in the MATLAB environment.

Aggregation

```
clear all
clc
close all
syms x1 x2 x3
N = 50;
d=1.5;
lambda = 0.05;
Cr =1;
mu = 0.1;
sec_dist = 0.5;
for i = 1:N
     pos(1,i) = 50*rand();
     pos(2,i) = 50*rand();
     pos(3,i) = abs(rand());
end
k=1;
flag = true;
G=create_graph(pos,sqrt(2)*d);
flag=true;
p=0;
G=create_graph(pos,sqrt(2)*d);
flag=true;
dXr =0;
while flag == true && p<200
     p = p+1;
     for i=1:N
          vicini=neighbors(G,i);
          varx=0;
          vary=0;
          varz=0;
          for j=1:length(vicini)
               varx=varx + pos(1,i)-pos(1,vicini(j));
               vary=vary + pos(2,i)-pos(2,vicini(j));
               varz=varz + pos(3,i)-pos(3,vicini(j));

          end
          var = [varx vary varz]';
          pos(:,i) = pos(:,i) - lambda*(var/length(vicini)) ;
     end
     [gradientx,gradienty,gradientz]=gradient(pos,G);
     if (abs(gradientx) <=10^(-5) && abs(gradienty) <=10^(-5) && abs(
        gradientz) <=10^(-5))
          flag = false;
     end
```

```
end
```

Potential Function Implementation

```
function pos = adjustement(pos,obst,dest,x1,x2,dist_matrix,lambda,
    sec_dist,mu,Cr,G,c)
for i=c:size(pos,2)
    c1 = 0;
    c2 = 0;
    dXr = 0;
    dXro = 0;
    for j=1:size(pos,2)
        if ismember(j,neighbors(G,i))
            dist = dist_matrix(i,j);
            target = pos(1:2,j)+dist/(norm(pos(1:2,i)-pos(1:2,j)))*(pos
                (1:2,i)-pos(1:2,j));
            P1a = x1*target(2)-(x1-target(1));
            P2a = target(2);
            Ft = [x1*x2-P1a;x2-P2a];
            Ja=jacobian(Ft);
            A = -inv(Ja)*Ft;
            N = norm(A);
            dXa(j,:)=A/N;
            if norm([pos(1:2,i)-pos(1:2,j)])<=sec_dist
                P1o = x1*pos(2,j)-(x1-pos(1,j));
                P2o = pos(2,j);
                Fo = [x1*x2-P1o;x2-P2o];
                Jo = jacobian(Fo);
                R = inv(Jo)*Fo;
                dXr=dXr+R/(1+(norm(R)/Cr)^mu*norm(R)^3)-sec_dist/(
                    sec_dist^3*(1+(sec_dist/Cr)^mu));
                disp("PERICOLO");
            end
        end
    end
    for s = 1:size(obst,2)
        if(norm(pos(:,i)-obst(:,s))<=sec_dist)
            if pos(2,i)<=obst(2,s)
                c1 = c1+1;
            else
                c2 = c2+1;
            end
            P1o = x1*obst(2,s)-(x1-obst(1,s));
            P2o = obst(2,s);
            Fo = [x1*x2-P1o;x2-P2o];
            Jo = jacobian(Fo);
            R = inv(Jo)*Fo;
            dXr=dXr+0.005*((R/(1+(norm(R)/Cr)^mu*norm(R)^3))-sec_dist/(
                sec_dist^3*(1+(sec_dist/Cr)^mu)));
            disp("PERICOLO Ostacolo ");
            end
        end
        if i==1 && not(isempty(dest))
            target = dest(1:2);
            vers = (target-pos(1:2,i))/norm(pos(1:2,i)-target);
            Dxr = double(sum(subs(dXr,[x1,x2],[pos(1,i),pos(2,i)])));
            if c2>c1
                Dxr = Dxr*sign(Dxr);
            end
            if Dxr ~= 0
                pos(1:2,i) = pos(1:2,i)+lambda*(Dxr)';
            else
                pos(1:2,i) = pos(1:2,i)+lambda*(vers);
            end
        else
            Dxa = double(sum(subs(dXa,[x1,x2],[pos(1,i),pos(2,i)])));
            Dxr = double(sum(subs(dXr,[x1,x2],[pos(1,i),pos(2,i)])));
            if c2>c1
                Dxr = Dxr*sign(Dxr);
            end
            Dx = (Dxa+Dxr)';
            pos(1:2,i) = pos(1:2,i)+lambda*(Dx);
        end
    end
end
```

Target Achievement

```
while norm(pos(:,1)-target(:,1)) > 0.1

    pos(:,1) = adjustement(pos(:,1),obst,target,x1,x2,0,8*lambda,0.8,mu
        ,Cr,G,1);
    pos = adjustement(pos,obst,[],x1,x2,dist_matrix,5*lambda,1,mu,Cr,G
        ,2);
    G = create_graph(pos,sqrt(2)*d);
    check = checkDist(pos,dist_matrix,G);

    while check > 0.5
        pos = adjustement(pos,obst,[],x1,x2,dist_matrix,3*lambda,
            sec_dist,mu,Cr,G,1);
        G = create_graph(pos,sqrt(2)*d);
        check = checkDist(pos,dist_matrix,G);
    end

end
```