D4.5– UPDATED SWARM MODELING LIBRARY

Deliverable ID	**D4.5**
Deliverable Title	**Updated swarm modeling library**
Work Package	**WP4 – Models and algorithms for CPS Swarms**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**31-10-2018**
Status	**Final**
Lead Editor	**LAKE**
Main Contributors	**Micha Rappaport (LAKE), Etienne Brosse (SOFTEAM), Gianluca Prato (ISMB), Midhat Jdeed (AAU)**

Published by the CSPwarm Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731946.

Document History

Version	Date	Author(s)	Description
0.0	2018-07-09	Melanie Schranz (LAKE)	Initial TOC
0.1	2018-09-18	Melanie Schranz (LAKE)	Initial structure: state machines (different levels, different use cases, events, states), swarm algorithms (low-, high-level modeling)
0.2	2018-09-19	Melanie Schranz (LAKE)	Add content Section 4 swarm algorithms
0.3	2018-10-12	Melanie Schranz (LAKE)	Add content to Section 3, 5 and 6
0.4	2018-10-17	Melanie Schranz (LAKE)	Use input from ISMB and AAU
0.5	2018-10-17	Melanie Schranz (LAKE)	Ready for review version
0.6	2018-10-23	Micha Rappaport (LAKE)	Revise content and integrate review from FIT, TTECH, SOFT
1.0	2018-10-25	Melanie Schranz (LAKE)	Final version

Internal Review History

Review Date	Reviewer	Summary of Comments
2018-10-17	Junhong Liang (FIT)	Structural comments
2018-10-29	Andreas Eckel (TTTECH)	Structural comments
2018-10-29	Etienne Brosse (SOFT)	Structural comments

Table of Contents

1 Introduction

This deliverable D4.5 – "Updated swarm modeling library" is a public document describing the work in CPSwarm until M22. In more detail it describes the results of T4.3 – Swarm modeling. The purpose of this deliverable is to update the initial swarm modeling library presented in D4.4. The swarm models will be mainly shown on the search and rescue (SAR) use case, which was also presented during the review meeting M18 in Turin, Italy. Results for the other use cases are also part of this deliverable, but are still work in progress.
In summary, the deliverable presents the use case related behaviors that where formalized as state machines on multiple level. Furthermore, we provide a list of swarm algorithms that were selected and prepared for the SAR use case implementation during WP8. Finally, we also describe the next steps in modeling swarm algorithms as already proposed in D4.4.

1.1 Document Organization

The remainder of this deliverable is organized as follows:
Section 2 describes the state machine concept and models used for the search and rescue (SAR) use case. Section 3 gives an overview on the swarm algorithms related to the identified behaviors in the SAR use case. Section 4 describes the modeling of swarm algorithms from a high- and a low-level view. Section 5 shows the preliminary design for state machines on the other use cases (logistics and platooning). Finally, Section 6 draws conclusions and provides an overview of future activities.

1.2 Related Documents

This deliverable is strongly intertwined with WP8 Task 8.1-Swarm of drones and ground robots that works on the search and rescue (SAR) use case presented by DIGISKY. WP4 and specifically Task 4.3-Swarm Modeling is mainly focused in offering swarm algorithms and behavioral concepts for this use case.

Table 1 Related Deliverables.

ID	Title	Reference	Version	Date
[D4.1]	Initial CPS Modeling Library	D4.1	1.0	30/09/2017
[D5.2]	Initial CPSwarm Modelling Tool	D5.2	1.0	30/09/2017
[D4.4]	Initial Swarm Modeling Library	D4.4	1.0	31/10/2017
[D2.2]	Final Vision Scenarios and Use Case Definition	D2.2	1.0	30/04/2018
[D3.2]	Updated System Architecture Analysis & Design Specification	D3.2	1.0	30/06/2018
[D5.3]	Updated CPSwarm Modelling Tool	D5.3	1.0	30/06/2018
[D4.2]	Updated CPS Modeling Library	D4.2	1.0	30/09/2018
[D8.1]	Swarm of Drones and Ground Robots Demonstration	D8.1	1.0	31/12/2018
[D8.3]	Initial Swarm Logistics Demonstration	D8.3	1.0	31/12/2018
[D8.5]	Initial Automotive Demonstration	D8.5	1.0	31/12/2018

2 State Machine Approach

We propose the use of state machines to control the behaviors of the swarm members. Each swarm member executes a hierarchical set of state machines where each state machine level represents a different abstraction level. The 1st level state machine is mainly responsible for controlling the parallel execution of high-level behaviors. Each of these high-level behaviors is controlled by a 2nd level state machine. The 2nd level state machine is responsible for executing the behavior of a swarm member. The 2nd level states can either call functions from the Abstraction Library (see D5.3) or again be described with behaviors consisting of yet more sub-levels of state machines.

2.1 The Search and Rescue (SAR) Use Case

The SAR use case can be summarized in specific mission points (more details can be found in D8.1- Swarm of drones and ground robots demonstration).
At the beginning of the mission,

- the drones swarm out, patrol the selected area and search for victims and
- the rovers wait for a call for intervention.

When a drone discovers one of the victims, the drone

- communicates the position of the victim to the rovers,
- starts an arbitration process to select the most appropriate rover,
- possibly asks for further support by other drones, and
- follows the victim while communicating possible changes of its position.

The rovers then

- submit a cost value for reaching the victim (e.g. the distance).
- The selected rover navigates to the victim using the position received from the drone.

After the rover reaches the victim, it

- uses an emergency exit strategy to guide the victim to the closest exit.

From this mission description we can identify the three main behaviors shown in Figure 1, two behaviors for drones (coverage and tracking), and one behavior for the rover (find exit). We will refer to the victims as targets in the following.

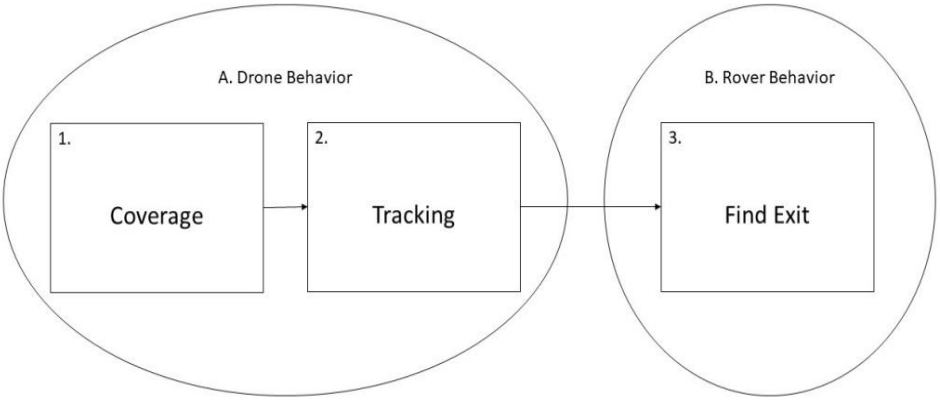

Figure 1 Three main behaviours in the SAR use case.

2.2 State Machine Design

From the mission description we derive the individual behaviors and transitions between them. We generalize this concept by designing a state machine for each use case.

2.2.1 Definition of Events

An event is something that happens during the mission and is processed by the swarm members. It can be triggered by inputs from sensors, by other local system components through the Abstraction Library, or remotely by other swarm members or workbench tools. Swarm members shall react to these events by changing their behavior. More formally, events trigger transitions in a behavior state machine. Events are defined by a unique name or identifier, timestamp, and unique identifier of the sender. Furthermore, events can have data associated with them. While the format of such data is dependent on the implementation, we use strongly typed key-value pairs which offer a flexible solution. Each event has a scope in which it is valid and propagated.

- Member scope: The event is raised locally but not propagated to other swarm members.
- Swarm scope: The event can be raised by any swarm member or workbench tool and is propagated to other swarm members. Such an event can be marked as command, which is a privileged event that can only be triggered by workbench tools and not by swarm members.
- Device scope: The event is raised by the Abstraction Library or another local system component and not propagated to other swarm members. Such an event can be imported from the Modeling Library and is specific to the CPS being used. Examples include tamper and fault events or special detection events supplied by custom software running on the CPS.

Each transition is associated with an event, and the transitions happen when such an event is triggered locally or remotely. Since events can contain data, a way to bind the data of incoming events to the inputs of the behavior is needed. In order to realize this, special inputs can be defined that can be bound to the fields in the event in order to supply constant input values to the behavior while the state is active. This binding is done on a per-transition basis, where an association between the fields of the incoming event and the special inputs of the behavior can be made. Default values for the fields can be defined in order to supply values for fields missing in the event.

The way to trigger events locally is to add a trigger condition to an outgoing event. A trigger condition can apply a mathematical formula (like a comparison operation) to any input or output of the state in order to determine if the event should be triggered or not. When the event is triggered, the fields of the outgoing event are handled similarly to how incoming events are treated. The outputs taken from the behavior are bound to the fields of the outgoing event. This binding is also done on a per-transition basis. Constant values can also be supplied for any of the fields. The events we identified for the SAR use case are listed in Table 2.

Table 2 Events for the SAR use case.

Identifier	Data	Sender	Scope
missionStart	-	Monitoring tool	swarm command
missionAbort	-	Monitoring tool	swarm command
targetFound	Target ID, target position	Swarm member	swarm
targetUpdate	Target ID, target position	Swarm member	swarm
targetLost	Target ID, target position	Swarm member	swarm
targetRescued	Target ID	Swarm member	swarm
emergencyEvent	-	Swarm member	device

2.2.2 Definition of State Types

State machines consist of states and transitions. A state defines the function signature of the behavior with inputs and outputs. The implementation of a behavior can be done manually (by writing the actual code), by using an external tool to generate it (such as the Optimization Tool FREVO), or by importing it from the Modeling Library. In the latter case, the implementation is either part of the component as found in the Modeling Library or realized by the Abstraction Library. Inputs are defined by referencing sensor outputs, and outputs are defined by referencing actuator inputs. We introduce two main types of states. First, states representing a high-level behavior that consist of a state machine. Second, states that implement a function from the abstraction library. They are shown in Figure 3 where green states represent high-level behavior and yellow states represent functions from the abstraction library.

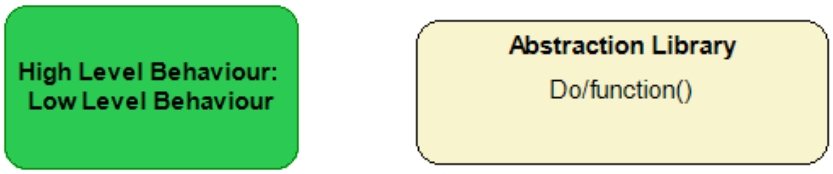

Figure 2 Different types of behavior states.

2.3 1st Level - General State Machine

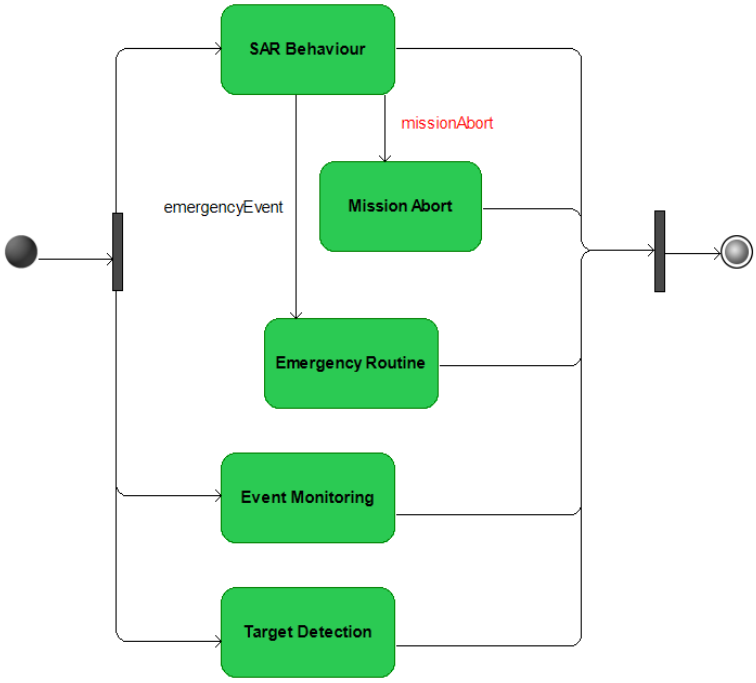

Figure 3 1st level state machine

On the first level, the state machine design for SAR shown in Figure 3 is very general.
It opens multiple threads that run in parallel. They are designed as high-level states (states that have another state machine behind), including:

- *SAR behavior*: different for drones and rovers
- *Event monitoring*: monitors the events and thus triggers state transitions
- *Target detection*: to detect a target with the on-board sensors

In the next sections we will go in more detail about the SAR behavior formalized by the 2nd level state machines.

2.4 2nd Level – Drone State Machine

The 2nd level state machine of the drones describes the SAR behavior executed by each drone. It is formalized by the state machine shown in Figure 4. After *StartUp* and being in an Idle mode, the mission is started with the *missionStart* event. First, this triggers the *TakeOff* function. This function succeeds once the drone is at the defined altitude. The SAR behavior then starts with the Coverage behavior as described in Section 3.1. If a target is found (indicated with *targetFound* event), a rover is selected for the task to rescue the target using the *SelectRover* state. The drone sends the position of the target to all available rovers. Each rover computes its distance from the target and sends this info back to the drone (together with its ID). The drone selects the closest one and answer back with the ID of the selected rover.

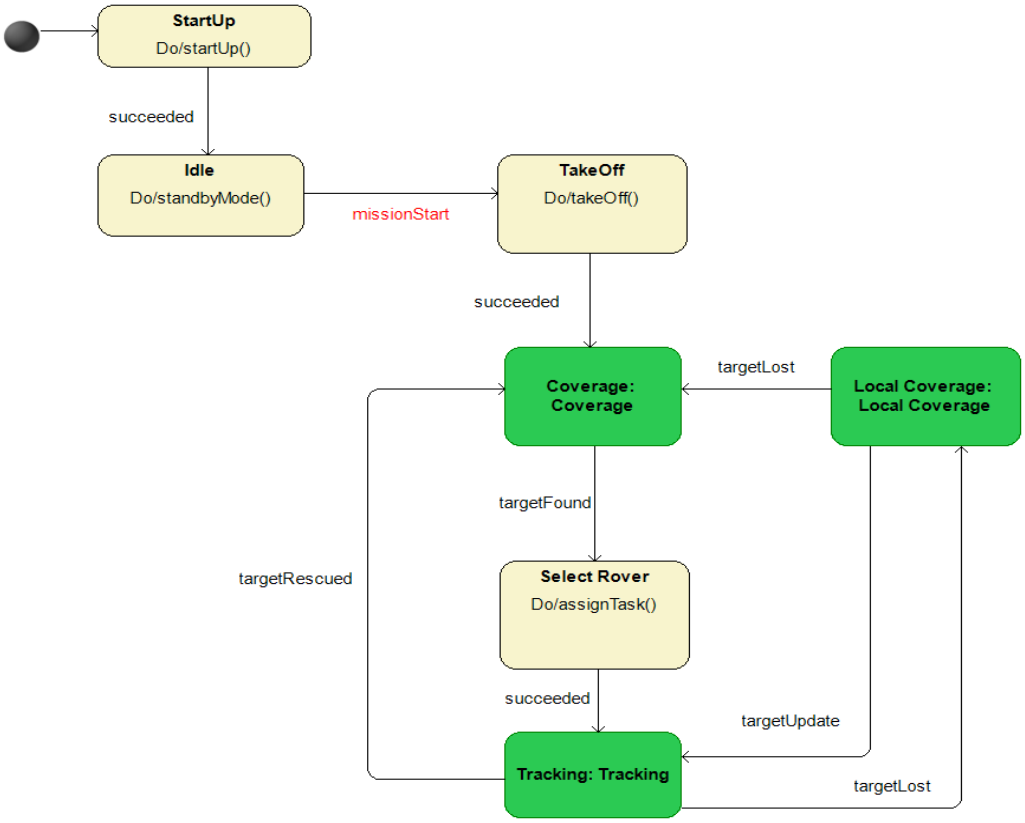

Figure 4 2nd level behavior: the state machine for the drone.

In the meantime the drone performs the Tracking behavior to not lose the target which is described in Section 3.2. If the *targetRescued* event is initiated (by a rover) the target is save and the drone restarts the Coverage behavior to find other targets. If the target is lost (indicated with the *targetLost* event) the drone starts a *LocalCoverage* behavior first. This behavior allows the drone to circle around the latest target position to find the target again. If the target is found again, the drone switches back to the Tracking behavior issuing a *targetUpdate* event. Otherwise the drone indicates that the Coverage behavior needs to be restarted with the *targetLost* event.

2.5 2nd Level – Rover State Machine

The 2nd level state machine of the rovers describes the SAR behavior executed by each rover. It is formalized by the state machine shown in Figure 5. After having executed the *StartUp* routine (in which the whole system is loaded and started) the behavior moves to the Idle state waiting for the discovery of a target. When a new target has been identified, the rover is informed through the *targetFound* event. In the following state (*SelectRover*) a negotiating task is executed in order to select the rover that will be responsible to reach the target. This operation is led by the drone that found the target. Then, if the rover has been selected, it executes

the *MoveToTarget* state in order to reach the target. Otherwise, it switches back to the Idle state. Hereafter, when the target is reached, the rover escorts it to the nearest exit executing the *ReachExit* state. Finally, the rover informs the swarm that the target has been rescued and switches again to the Idle state waiting for another target to be saved.

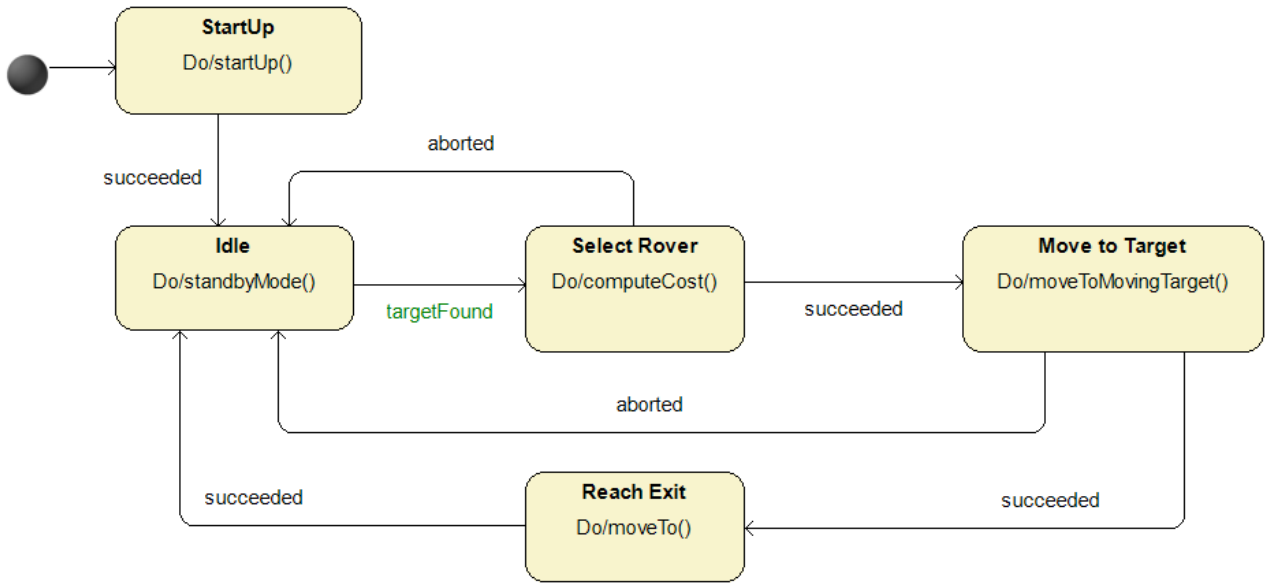

Figure 5 2nd level behavior: the state machine for the rover.

3 Swarm Intelligence Algorithms for the SAR use case

As described in the previous Section, we identified three main behaviors for the SAR use case: Coverage, Tracking and Find Exit. For each of them, we provide a theoretical elaboration covering a set of swarm algorithms that were already partially implemented as models to be used in WP8 implementation.

3.1 Coverage

Task
The drones sweep over the demo space and cover it to find the target.

Advantages of a swarm
- Parallelization of the task
- Redundancy in case drones fail, e.g., running out of battery
- Redundancy to avoid missing moving targets

Parameters
- Area of environment (polygon GPS coordinates)
- Step size: Distance that the drone travels in each iteration

Proposed algorithms
- Random walk
- Random direction
- Flocking
- Fish schooling
- Cooperative sweeping

3.1.1 Random walk

Random walk is a mathematical movement model, whereby each next step is chosen randomly. It is a stochastic process in discrete time with independent and identically distributed steps, where the position

$$X_{t+1} = X_t + v$$

changes from one step to another in a random direction v. It can be applied in 1, 2 or 3 dimensions.

A random walk could look like in Figure 6 using:
- 1000 simulation steps
- uniform distribution
- starting point S(0|0)
- area 30x30
- step size: 1

Inputs:
- Dimension (1, 2 or 3)
- Boundaries of area to cover
- Random distribution (uniform, Gaussian, Lévy, etc.)
- Prioritized directions (optional)

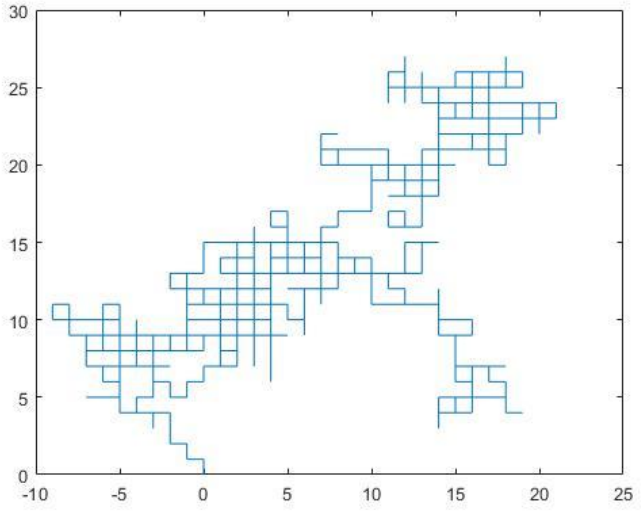

Figure 6 Random walk simulation.

Pseudo Code:

```
1   rand_min=0;
2   rand_max=5;
3   X(1)=0;
4   Y(1)=0;
5   for t:t_max
6      rand_value=round(random_number(rand_min,rand_max));
7      switch rand_value
8          case 1: X(i)=X(i-1)+1; Y(i)=Y(i-1); //move right
9          case 2: X(i)=X(i-1)-1; Y(i)=Y(i-1); //move left
10         case 3: Y(i)=Y(i-1)+1; X(i)=X(i-1); //move up
11         case 4: Y(i)=Y(i-1)-1; X(i)=X(i-1); //move down
13         case 0,5: X(i)=X(i-1); Y(i)=Y(i-1);
12  end for
```

Hardware requirements:

- Down-facing camera

Software requirements:

- Target detection
- Collision avoidance

Advantages:

- Uncontrolled movement (like a drunk), and therefore it is not expected to predict the drone's next position

Disadvantages:

- A lot of space will not be covered in a reasonable time → convergence rather low

3.1.2 Random direction

Random direction is a mathematical movement model, whereby an agent moves straight forward until it reaches a point of interest. There, it changes its direction randomly (see sketch in Figure 7).

A random direction algorithm could look like in Figure 8 using:

- 1000 simulation steps
- starting point S(0|0)
- area 30x30
- angles: uniform distribution as soon as area boundaries are hit

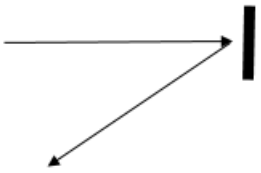

Figure 7 The principle of random direction.

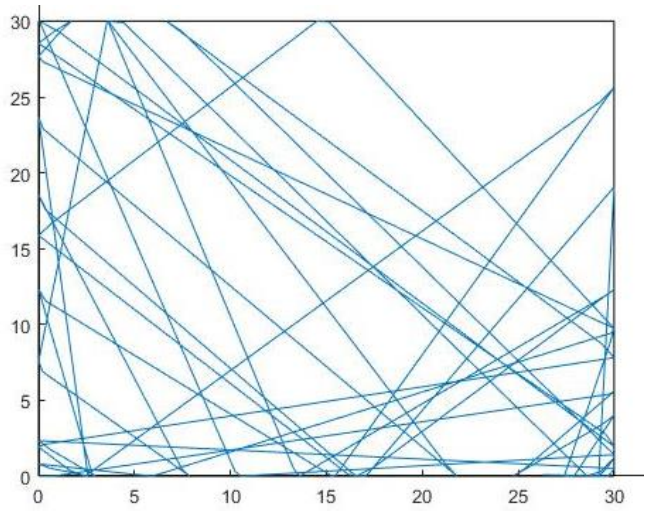

Figure 8 Random direction simulation.

Inputs:

- Dimension (1, 2 or 3)
- Define distribution (uniform, Gaussian, Lévy, etc.)
- Angle: Direction in which the drone travels, measured counter clockwise from the positive x-axis.
- Assign priorities to angles (optional)

Pseudo Code:

```
x_max=30;
y_max=30;
x_min=0;
y_min=0;
rand_min=0;
rand_sin=pi;
rand_cos=pi/2;
rand_max=x_max;

rand_angle_sin=(rand_sin-rand_min)*rand+rand_min;
rand_angle_cos=(rand_cos-rand_min)*rand+rand_min;

for t:t_max
      if (X(t-1)>=x_max)
        rand_angle_sin=(2*pi-rand_min)*rand+rand_min;
        rand_angle_cos=(3*pi/2-pi/2)*rand+pi/2;
      elseif (X(t-1)<=x_min)
        rand_angle_sin=(2*pi-rand_min)*rand+rand_min;
        rand_angle_cos=(pi/2-rand_min)*rand+rand_min;
      elseif (Y(t-1)>=y_max)
        rand_angle_sin=(2*pi-pi)*rand+pi;
        rand_angle_cos=(2*pi-rand_min)*rand+rand_min;
      elseif (Y(t-1)<=y_min)
        rand_angle_sin=(pi-rand_min)*rand+rand_min;
        rand_angle_cos=(2*pi-rand_min)*rand+rand_min;
      end

      X(t)=X(t-1)+cos(rand_angle_cos);
      Y(t)=Y(t-1)+sin(rand_angle_sin);
end
```

Advantages:

- Covers a lot of area in short time

Disadvantages:

- Target evades easily as the drone movement pattern is simple and therefore the next position easy to predict

3.1.3 Flocking

Flocking is the behavior observed in flocks of birds. The flocking behavior is a self-organized collective motion of individuals. It follows three simple rules (proposed by Reynolds [1]):

- Separation - avoid crowding neighbors (short range repulsion)
- Alignment - steer towards average heading of neighbors
- Cohesion - steer towards average position of neighbors (long range attraction)

These rules are adapted for flocking in drone swarms. Drones need short-range repulsion from other units, middle-range velocity alignment with neighbors, and a global positional constraint to remain with the flock [2].

1. Short-range repulsion: pair potentials

A repulsive distance-based potential acts between all close units to avoid collisions:

$$\vec{a}_{pot}^{i} = \begin{cases} -D \sum_{j \neq i} min\left(r_1,\ r_0 - \left|\vec{x}^{ij}\right|\right) \frac{\vec{x}^{ij}}{\left|\vec{x}^{ij}\right|}, \\ \text{if } \left|\vec{x}^{ij}\right| < r_0, \\ 0 \text{ otherwise}, \end{cases}$$

D: spring constant of a repulsive half-spring
$x^{ij}=x^j-x^i$: difference of positions of unit i and j
r_0: equilibrium distance (distance above which there is no repulsion between units)
r_1: used to define an upper threshold for the repulsion to avoid over-excitation of units

2. Middle-range Velocity Alignment: Viscous Friction

Units close to each other damp their velocity difference to reduce oscillations and to synchronize collective motion with a viscous friction-like term.

$$\vec{a}_{slip}^{i} = C_{frict} \sum_{j \neq i} \frac{\vec{v}^{ij}}{\left(max\left(\left|\vec{x}^{ij}\right| - (r_0 - r_2),\ r_1\right)\right)^2},$$

C_{frict}: viscous friction coefficient
$v^{ij}=v^j-v^i$: velocity difference between units i and j
r_2: constant slope around the equilibrium distance r_0
r_1: threshold to avoid division by close-to-zero distances (e.g. due to measured GPS position error)

NOTE: both equations depend on r_0 to allow for the dynamic tuning of flock density. This is essential if we apply higher velocity ranges and hence need larger breaking distances.

3a. Global Positional Constraint I: Flocking

Units move around within the walls of an area

$$\vec{v}_{spp}^{i} = v_{flock} \frac{\vec{v}^{i}}{\left|\vec{v}^{i}\right|},$$

v_{flock}: constant flocking speed units try to maintain

Bounding walls are defined globally, and appear as local attractive shill agents. They try to pull units back towards the center of the flight area through virtual velocity alignment:

$$\vec{a}^i_{wall} = C_{shill} f\left(\left|\vec{x}_{trg} - \vec{x}^i\right|, R, d\right)\left(v_{flock}\frac{\vec{x}_{trg}-\vec{x}^i}{\left|\vec{x}_{trg}-\vec{x}^i\right|} - \vec{v}^i\right)$$

C_{shill}: viscous friction coefficient of the wall
x_{trg}: center of the flight area (position of a global reference/target point)
f(x,R,d): smooth transfer function:

$$f(x,R,d) = \begin{cases} 0 & if\ x \in [0,R] \\ \frac{\sin\left(\frac{\pi}{d}(x-R)-\frac{\pi}{2}\right)+1}{2} & \text{if } x \in [R, R+d] \\ 1 & if\ x \in [R+d, \infty] \end{cases}$$

R: distance between the reference point and the wall
d: width of the decay (softness of the wall)

3b. Global Positional Constraint II: Formation Flight

$$\vec{v}^i_{track} = \begin{cases} v_0 \frac{\vec{v}^i_{shp}+\vec{v}^i_{trg}}{\left|\vec{v}^i_{shp}+\vec{v}^i_{trg}\right|} & if\ \left|\vec{v}^i_{shp} + \vec{v}^i_{trg}\right| > v_0 \\ \vec{v}^i_{shp} + \vec{v}^i_{trg} & otherwise \end{cases},$$

$$\vec{v}^i_{shp} = \beta v_0 f\left(\left|\vec{x}^i_{shp} - \vec{x}^i\right|, R, d\right)\frac{\vec{x}^i_{shp}-\vec{x}^i}{\left|\vec{x}^i_{shp}-\vec{x}^i\right|},$$

$$\vec{v}^i_{trg} = \alpha v_0 f\left(\left|\vec{x}_{trg} - \vec{x}^i_{COM}\right|, R, d\right)\frac{\vec{x}_{trg}-\vec{x}^i_{COM}}{\left|\vec{x}_{trg}-\vec{x}^i_{COM}\right|}.$$

v_{shp}: defines how units arrange themselves into shapes relative to the individual calculated center of mass
v_{trg}: defines how the center of mass follows the target with smoothly adjusted, variable speed
alpha, beta: range [0,1] - define the strength of components relative to v_0 (maximal tracking velocity)
x_{com}: locally calculated center of mass
x_{shp}: desired position in the formation around x_{com}
f(x,R,d): smooth transfer function to avoid abrupt changes in the dynamics

<u>For line formation:</u>

- define the signed neighborhood metric as the projected distance on the line from x_{com}
- set x_{shp} as the average of the two neighbors on the line
- Units with only one or zero neighbors must approach the closest end of the line to ensure that units settle with the equilibrium distance r_0 between them
- R is set to a small value or zero
- The angle of the line can be defined in a self-organized way from a linear fit of actual positions

Desired velocity of the drone

$$\vec{v}^i(t+\Delta t) = \vec{v}^i(t) + \frac{1}{\tau}\left(\vec{v}^i_{spp} + \vec{v}^i_{track} - \vec{v}^i(t)\right)\Delta t + \left(\vec{a}^i_{pot} + \vec{a}^i_{slip} + \vec{a}^i_{wall}\right)\Delta t,$$

tau: characteristic time needed to reach the velocity $v_{spp} + v_{track}$
(t+delta t): we set the system for a time instance far ahead to prepare it for the future and the system reacts in time (delta t ~ 1sec)

Hardware Requirements

- Down-facing camera
- GPS
- accelerometer
- gyroscope
- magnetometer
- pressure sensor
- XBee (2.4GHz to/from other units)

Software Requirements

- Target detection
- Collision avoidance

Advantages

- More effective when solving foraging or navigational tasks
- Robust (flock holds its actual goal due to redundancy and parallel processing structure)
- Alert regarding environmental threats and more defensive against attacks
- Already implemented decentralized [1]

3.1.4 Fish schooling

Fish schooling is inspired by the emergent behavior of fish schools searching food. The algorithm is performed in a bounded search space called aquarium and each fish has a visual distance to others. The basic concept is composed of feeding and movement operators (swimming). Feeding process is used to update the weights of the fish, the weights are updated iteratively depending on the food amount. The movement operators have three steps [3] [4] [5]:

1. **The individual movement:** it is performed individually by each fish, then each fish assesses whether the food density is better or not.
2. **The collective – instinctive movement:** it is performed based on a weighted average of the previous step of all fish.
3. **The collective – volitive movement:** this is be based on the overall performance of the fish school. If the fish school has been successful in finding food, the radius of the school contracts, otherwise expands. Expanding or contracting is based on the school barycentre which is obtained by considering all fish positions and weights.

Possible fish schooling approach from the Drones Scenario perspective:
Generally, the fish schooling behaviors recommended for applications like group escape, where all members of a swarm should stick together, which is not the case in the search and rescue scenario.
On the other hand, to apply such an algorithm for SAR scenario, each drone should have a minimum and maximum distance to keep with its neighbors. Once a drone found a target, a contracting process is executed. Then, the drones execute another round of fish schooling while ignoring the target that has been found.

Inputs

- The boundary of the desired area
- Number of targets

Pseudo Code

```
1:Initialize randomly all drones' positions
2:Initialize randomly all drones' weights
3:while stop criterion is not met do
4:              for each drone do
5:                   Find neighbor position;
6:                   Evaluate fitness according to and perform greedy search;
7:                   update the drone weight;
8:              end for
9:Evaluate the drift ( influenced only by a drone that successfully performed
individual movements);
10:             for each drone do
11:                  Execute instinctive movement;
12:             end for
13:Calculate barycenter;
14:             for each drone do
15:                   Execute volitive movement;
16:              end for
17: Update the volitive step and the individual step.
18: end while
```

Stop criterion, e.g. Time limit, maximum school radius

Messages

- finding a target

Hardware Requirements

- Sensors for estimating the distance between drones and maintaining a constant range
- Kind of processing to detect the target (image processing)

3.1.5 Cooperative sweeping

The algorithm is a decentralized, cooperative sweep coverage algorithm. It uses a market-based auctioning strategy for task allocation among the drones. The general idea is that the area to be covered is divided into sections. Each section is covered by one drone using a sweeping pattern. The algorithm consists of two parts: 1. task allocation which assigns a section to each drone; 2. sweeping where every drone covers the assigned section. Once a drone discovers a target, it stops sweeping and switches to the tracking behavior [6] [7] [8].

Inputs

- Coordinates of area to be covered

Pseudo Code

Task Allocation

```
detect number of drones
compute sections of area proportional to number of drones
while no section assigned:
  if auction is received:
    reply with bid for that section
    wait for result
  else:
    select unassigned section of area
    start auction including bid for that section
    wait until timeout
```

```
    notify winner
  if won auction:
    assign section
```

Sweeping

```
compute sweep pattern
while sweeping:
  if detection:
    broadcast location
    broadcast state change
    start tracking
    break
  else if help call:
    reply with bid
    wait for result
    if assigned:
      broadcast state change
      go to help
      break
  else if received state change:
    reassign area that is now uncovered
  sweep the assigned section
```

Messages

- Auction for area section
- Bid for area section
- Auction winner notification
- Location of target
- State change of swarm member
- Bid for help call

Hardware Requirements

- Localization
- Communication between drones
- Down facing camera

Software Requirements

- Target detection
- Path planning
- Navigation
- Collision avoidance
- Communication protocol

3.2 Tracking

Task

If a drone finds a target, its task is not to lose it. It might call other drones for help, and transmits the GPS coordinates of the target to the rover.

Advantages of a swarm

- Parallelization of the task
- Tracking of multiple targets
- Redundancy in case drones fail, e.g., running out of battery
- Redundancy in order to better track moving targets
- Limited perception of single drones du to imperfect sensors

Parameters

- The target to track

Proposed algorithms

- Piranha
- Ants foraging
- Slime mold movement
- Wolf packs
- Cooperative tracking

3.2.1 Piranha

The literature defines no official Piranha swarm algorithm, but there is Piranha swarm behavior in nature to protect the single fish in the swarm. Furthermore, there is a Piranha inspired pattern matching algorithm for intrusion detection. It is based on the observation that if the rarest substring of a pattern does not appear, then the whole pattern will definitely not match [9].

Possible Piranha approach from the Drones Scenario perspective:
One drone calls others, once a possible target has been found. This behavior relates to the clichés provided in folklore and movies (real attacks on humans are quite infrequent and not deadly [10]), but nevertheless the concept can be helpful. Communication among Piranhas is done via different sounds emitted via the fish's swim bladder and teeth [11].

Outputs

- Movement
- Call
- Action on object

The concept of the Piranha algorithm is given in Figure 9.

Messages

- "Come over here!" message

Software Requirements

- Detection: Possibility of individual swarm members to detect prey or predators.
- Communication: Possibility to call for other swarm members.

Hardware Requirements

- Sensors to detect and assess the distance to other swarm members
- Locomotion unit
- Sensors to detect object of interest
- (Local) broadcast communication feature
- Ability to detect direction of a broadcast call

Advantages

- Simple

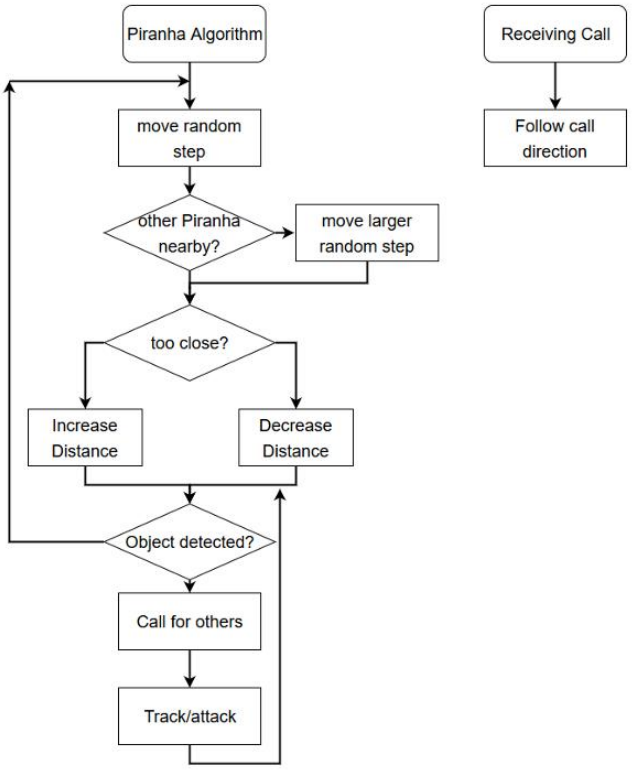

Figure 9 The Piranha algorithm.

Disadvantages

- Require global call
- Requires tracking after being called
- Benefit of schooling is not clear in technical scenario

3.2.2 Ants foraging (different sources for food)

It is a decentralized algorithm, based on pheromones to mark trails connecting the home (nest of ants) and targets (food resources). Basically, there are two types of pheromones, one for finding a target and one leading back home. The used algorithm is inspired by ants foraging and assumes that both types can co-exist at the same location. An indirect communication is used between the ants via pheromone, known as stigmergy. Each pheromone has a weight to determine the shortest path between home and a target. When the ants are looking for a path toward (food or nest), they are moving to locations with maximum pheromones or leaving their own pheromones to make a new path [12] [13] .

Ants foraging from the Drones Scenario perspective:

- The target of the drones should not be changing continuously.
- The drone must be able to mimic the chemical pheromone of ants to make their path traceable from other drones.
- The recommended number of agents for applying such an algorithm is from 100 to thousands.

3.2.3 Slime mold movement (accumulation of drones equal size)

Slime mold is a generic term for many different species that use spores to reproduce. Slime mold can be used as a biological inspiration to solve networking and path finding problems, e.g. shortest path through a maze or optimal network construction. It can solve NP-hard network problems efficiently, e.g. traveling salesman problem [14].

Biological computation method for network construction

1. Place food at POIs
2. Introduce slime mold at a central / major POI
3. Observe as it produces a network.

Basic network optimization algorithm

1. Start with random network connections
2. Pump virtual protoplasm through it
    a. Connections with "fast flow" grow wider and flow faster
    b. Connections with little flow disappear

Potential application areas

- Transport network creation: Using rovers to guide targets to safety
- Minimal exposure network: Threat avoidance for drones / rovers (emergency exit strategy) [15]

3.2.4 Wolf packs

The Gray Wolf Optimizer (GWO) [16] mimics the hierarchy and hunting mechanisms of gray wolves. Grey wolves (canis lupus; apex predators) prefer to live in a pack. On average the group size is 5-12. In the pack, discipline is much more import than its strength.

They use 4 types of wolves:

1. alpha
   - a male and a female
   - responsible for making decisions about switching between 4 steps (described below) which are dictated to the pack
   - not necessarily the strongest member, but the best in terms of managing the pack
2. beta
   - male or female
   - helps the alpha in decision-making or other pack activities
   - best candidate to substitute the alpha if necessary
   - respects the decisions made by alpha, but commands the other lower-level wolves and gives feedback to the alpha (adviser to the alpha, discipliner for the pack)
3. delta
   - if not alpha, beta, or omega
   - submit to alphas, betas, but dominate the omega
   - possible roles:
     - scouts (responsible for watching the boundaries of the territory and warning the pack),
     - sentinels (protect and guarantee safety of the pack),
     - elders (experienced wolves, used to be alphas or betas),
     - hunters (help alphas and betas),
     - caretakers (care for weak, ill, wounded wolves)
4. omega
   - lowest ranking gray wolf
   - scapegoat: always submits to higher-ranked wolves, last wolf that is allowed to eat
   - important role: violence and frustration is put on the omega which assists in satisfying the entire pack and maintaining the dominance structure
   - sometimes also the babysitter

The GWO has 4 steps:

1. **Encircling prey:**

$$\vec{D} = |\vec{C}.\overrightarrow{X_p}(t) - \vec{X}(t)|$$

$$\vec{X}(t+1) = \overrightarrow{X_p}(t) - \vec{A}.\vec{D}$$

where t indicates the current iteration, A and C are coefficient vectors , $X_p(t)$ is the position vector of the prey, and X indicates the position vector of a gray wolf.

The vectors A and C are calculated as follows:

$$\vec{A} = 2\vec{a}.\overrightarrow{r_1} - \vec{a}$$

$$\vec{C} = 2.\overrightarrow{r_2}$$

Where components of vector a are linearly decreased from 2 to 0 over the course of iterations and r_1, r_2 are random vectors in [0,1].

With the above equations, a gray wolf at the position (X,Y) can update its position according to the position of the prey (X*,Y*). Different places around the best agent can be reached with respect to the current position by adjusting the value of A and C vectors. For instance, (X*-X,Y*) can be reached by setting A=(1,0) and C=(1,1) Note that the random vectors r_1 and r_2 allow wolves to reach any position between the two particular points.

2. **Hunting:**

$$\vec{D_\alpha} = |\vec{C_1}.\vec{X_\alpha} - \vec{X}|$$

$$\vec{D_\beta} = |\vec{C_2}.\vec{X_\beta} - \vec{X}|$$

$$\vec{D_\delta} = |\vec{C_3}.\vec{X_\delta} - \vec{X}|$$

$$\vec{X_1} = \vec{X_\alpha} - \vec{A_1}.(\vec{D_\alpha})$$

$$\vec{X_2} = \vec{X_\beta} - \vec{A_2}.(\vec{D_\beta})$$

$$\vec{X_3} = \vec{X_\delta} - \vec{A_3}.(\vec{D_\delta})$$

$$\vec{X}(t+1) = \frac{\vec{X_1} + \vec{X_2} + \vec{X_3}}{3}$$

In an abstract search space we have no idea about the location of the optimum (prey) and the best candidate solution (alpha, beta, or delta). Therefore, we save the first three best solutions obtained so far and oblige the other search agents (including the omegas) to update their positions according to the position of the best search agent.

3. **Attacking prey (exploitation):**

The gray wolves finish the hunt by attacking the prey when it stops moving. To mathematically model approaching the prey we decrease the value of a. Note that the fluctuation range of A is also decreased by vector a. In other words A is a random value in the interval [-2a,2a] where a is decreased from 2 to 0 over the course of iterations. When random values of A are in [-1,1], the next position of a search agent can be in any position between its current position and the position of the prey.

4. **Search for prey (exploration):**

Over the course of iterations, alpha, beta, and delta wolves estimate the probable position of the prey. Each candidate solution updates its distance from the prey. The parameter a is decreased from 2 to 0 in order to emphasize exploration and exploitation, respectively. Candidate solutions tend to diverge from the prey when

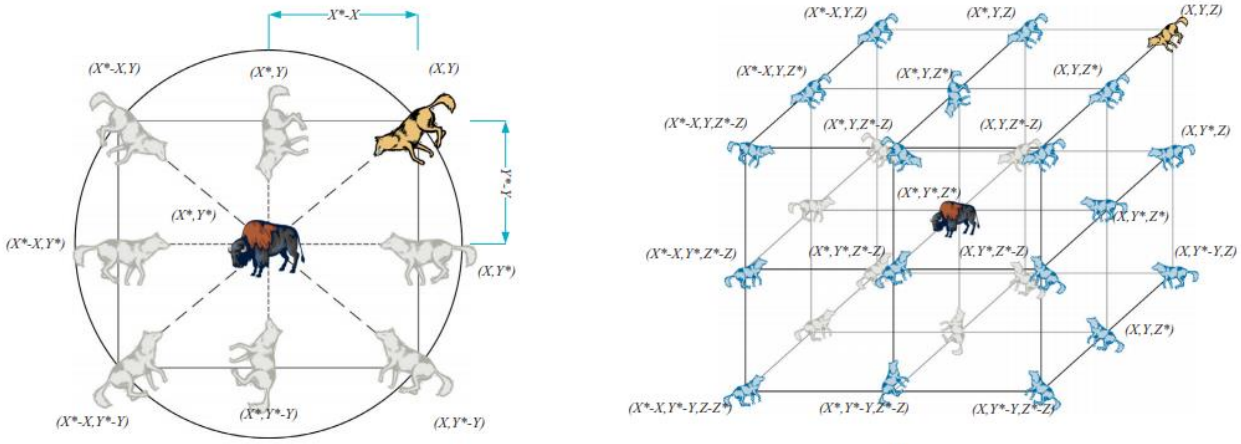

Figure 10 2D and 3D position vectors and their possible next locations.

|A| > 1 and converge towards the prey when |A|<1. Finally, the GWO algorithm is terminated by the satisfaction of an end criterion.

|A|>1 forces the gray wolves to diverge from the prey to hopefully find a fitter prey. Another component of GWO that favors exploration is C, which contains random values in [0, 2]. This component provides random weights for prey in order to stochastically emphasize (C>1) or de-emphasize (C<1) the effect of prey in defining the distance. This assists GWO to show a more random behavior throughout optimization, favoring exploration and local optima avoidance. It is worth mentioning here that C is not linearly decreased in contrast to A. We deliberately require C to provide random values at all times in order to emphasize exploration not only during initial iterations but also final iterations. This component is very helpful in case of local optima stagnation, especially in the final iterations.

Inputs

- Boundaries of area to cover
- Initialize a, A, and C

Pseudo Code

```
1  Initialize the gray wolf population Xi (i = 1, 2, ..., n)
2  Initialize a, A, and C
3  Calculate the fitness of each search agent
   X_alpha=the best search agent
   X_beta=the second best search agent
   X_delta=the third best search agent
4  while (t < Max number of iterations)
5    for each search agent
6      Update the position of the current search agent by above equations
7    end for
8    Update a, A, and C
9    Calculate the fitness of all search agents
10   Update X_alpha, X_beta, and X_delta
11   t=t+1
12   end while
13 return X_alpha
```

Messages

- Broadcast local vector X_i

Hardware Requirements

- Down-facing camera
- Broadcast module

Software Requirements

- Target detection
- Collision avoidance

Advantages

- no chance to evade

Disadvantages

- A drone cannot prevent the target to pass the way
- The GWO algorithm is prone to stagnation in local solutions with these operators. It is true that the encircling mechanism proposed shows exploration to some extent, but GWO needs more operators to emphasize exploration.

3.2.5 Cooperative tracking

Each drone estimates the position of a moving target using off-board image processing. The drones exchange messages of their estimation to improve the overall estimation. The estimation uses a linear state estimation based on the Kalman filter. The time is discretized by a sampling rate. Continuous dark areas ("blobs") in the image are assumed to be objects of interest [17].

Outputs

- Position of target

Pseudo Code

```
1  set C = Ø; (The set of objects in the current frame)
2  set P = Ø; (The set of persistent objects)
3  while true do:
4    for p in P do:
5      p.f = false; (Mark p as "not found")
6    end for;
7    F = getCurrentFrame(); (Get the current image frame)
8    G = preprocessFrame(F ); (Downsample image and convert to binary image)
9    C = findObjectsInFrame(G); (Find objects in the current frame)
10   for c in C do:
11     f = false;
11     for p in P do:
12       if (|c.x - p.x| < εx ) and (|c.A - p.A| < εA ) do:
         (Determine whether c is similar to p in terms of location in the
         image x and area a)
13         p.f = true; (Mark p as "found")
14         p.n = p.n + 1; (Add 1 to the running frame count of p)
15         f = true;
16         break; (Since p is determined, there is no need to continue
           examining the other elements of P )
17     end for; (Ends for p in P )
18     if f == false do:
19       P.push(c); (If c is not found anywhere in P , append c to P )
20     end if;
21   end for; (Ends for c in C)
22   for p in P do:
23     if p.f == false do: (If p was not found in this frame...)
24       p.n = p.n - 1; (...subtract 1 from the running frame count of p)
25     end if;
26     if p.n == 0 do: (If the frame count of p is zero...)
27       P.pop(p); (...remove p from P )
28     end if;
29   end for;
30 end while;
```

Messages

- Location of drone
- Estimated direction of target relative to drone position

Hardware Requirements

- Localization
- Communication between drones
- Camera

Software Requirements

- Target detection (image processing)

- Path planning
- Navigation
- Collision avoidance
- Communication protocol

Advantages

- Multiple drones provide redundancy, allowing for continued tracking even when individual vehicles experience failures.
- Using multiple drones with different lines of sight increases the probability that the target will remain observable to the group of drones even when individual vehicles' lines of sight are blocked.
- Because more observations are available at a given time, multiple drones working together can estimate the target's state more accurately than a single drone could.
- Since the entire process runs in linear time with respect to the number of drones, this method is very computationally efficient.
- This method requires only six numbers to be transmitted by each drone, making it well suited for environments where communication bandwidth is limited.

Disadvantages

- Image processing might be too computational complex

3.3 Find Exit

Task

Find target via GPS coordinates and guide it to closest emergency exit

Proposed algorithms

- Online Aerial Terrain Mapping for Ground Robot Navigation
- Evolved algorithm

3.3.1 Online Aerial Terrain Mapping for Ground Robot Navigation [18]

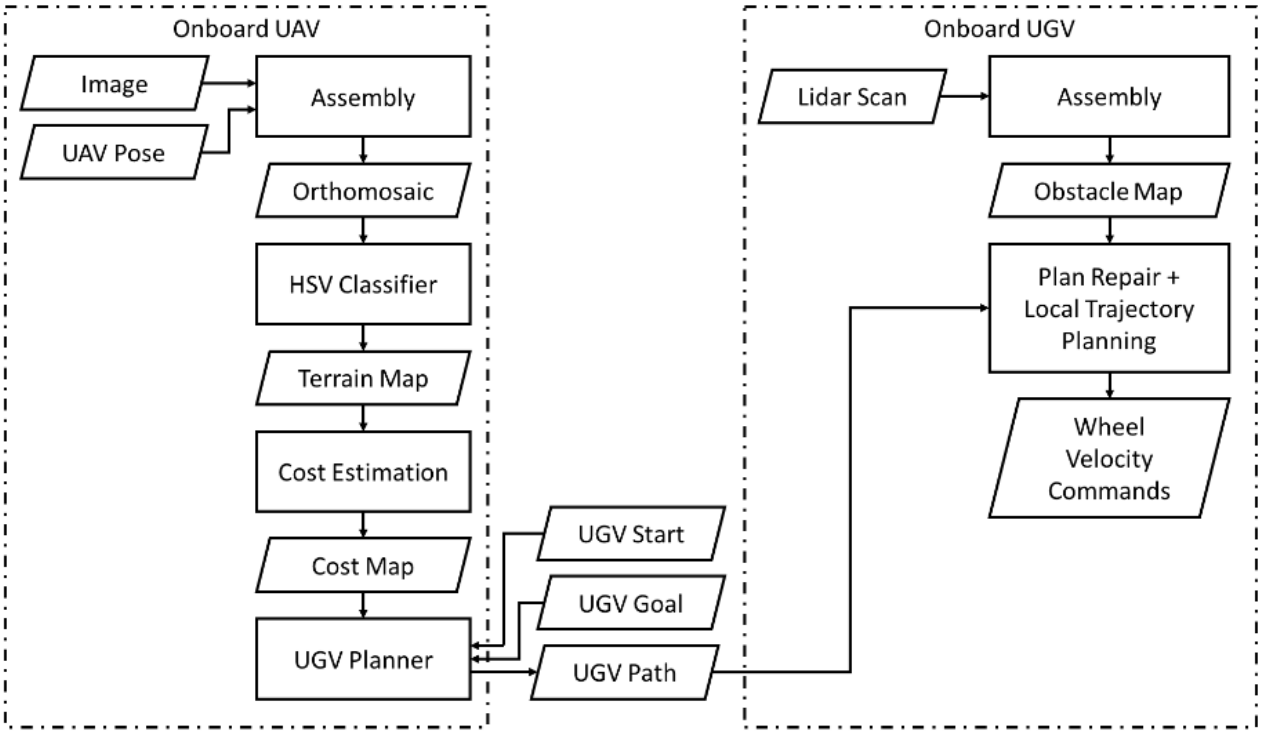

Figure 11 Flowchart of the cooperative drone rover path finding algorithm.

A drone is used to map the environment and to compute the path for a rover. The rover follows the pre-planned path using its own sensors for obstacle avoidance. The environment can also be classified by the cost for the rover to traverse it. This can be represented as a cost map.
A flowchart of the algorithm is given in Figure 11.

Inputs

- Rover start location, i.e. location of target
- Rover goal, i.e. exit

Messages

- Rover path

Requirements Drone
Hardware

- Localization (common coordinate system)
- Camera (monocular)
- Decentralized communication

Software

- Mapping (2d image mosaicking, obstacle detection by color)
- Communication protocol

Requirements Rover
Hardware

- Localization (common coordinate system)
- Decentralized communication

Software
- Navigation
- Collision avoidance
- Communication protocol

Advantages
- Rover takes (near) optimal path

Disadvantages
- Map of environment required

3.3.2 Evolved algorithm

In the EmergencyExit example, multiple swarm members move in a 2D, discrete environment and try to find one of two emergency exits. In each discrete time step, a swarm member senses the neighboring cells and moves to a free cell. When a swarm member reaches an emergency exit, it is removed from the environment. The goal is that all swarm members exit the environment (see D4.1 for further explanation).

Inputs to simulate such a behavior are:
- Simulation steps
- Width, height, and exits positions of an area
- Initial position

In such simple behavior:
- The time step is discrete.
- A member senses the neighboring cells to avoid obstacles, if any, and moves randomly to a free cell.
- A member is removed from the environment after reaching one of the existing emergency exits.
- When all members are removed, the goal is achieved.

There is no pseudo code available, as it is evolved with the optimization tool FREVO and follows a neural network representation.

Advantages
- The algorithm can be customized to the given environment via evolution.

Disadvantages
- If the environment changes, the process of evolving the algorithm needs to be restarted.

4 Modeling of Swarm Algorithms

The initial concept of modeling swarm algorithms was already described in D4.1 – Initial Swarm Modelling Library. We introduced a common modeling standard for swarm behavior. The main idea is to have a formulation and definition of models visually representing in SysML. We divide this representation into two views. First, the high-level view that models the whole swarm intelligence algorithm as a single block together with its inputs and outputs connected to the hardware model. They are comparable to the high-level behaviors of the state machines described above. Second, the low-level view that models the individual atomic behaviors as a state machine. This allows reusing the behaviors for creating new swarm algorithms in the Modeling Tool. In both cases, the goal is to deposit code to those models and model pieces to speed up the development and foster reusability.

4.1 High-level view

This modeling approach represents swarm intelligence algorithms from a high-level view. In the final modeling library, they can be found ready to use, including

- a description of their functionality,
- defined inputs and outputs,
- defined local states (if necessary), and
- deposited code (e.g., Java or C++ code).

This modeling is shown on the example of *RandomDirection* (Figure 12) and *FishSchooling* (Figure 13). These two algorithms were described in the previous section and also implemented in the demo as Coverage behavior for the SAR use case.

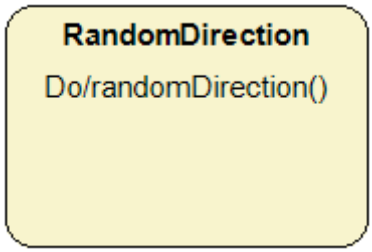

Figure 12 High-level view of the RandomDirection algorithm.

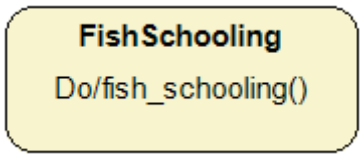

Figure 13 High-level view of the FishSchooling algorithm.

4.2 Low-level view

Typically, real-world applications come with needs that cannot be directly fulfilled by existing swarm intelligence algorithms. Therefore, it is useful to have a process that allows the construction of customized swarm intelligence algorithms. This is enabled by a library that provides single behaviors extracted out of given swarm intelligence algorithms in form of states. From those behaviors it is possible to construct a state machine. Each state has deposited code.

Furthermore, each behavioral element has a number of inputs and outputs that allow connecting with other elements. This state machine can be integrated into a high-level model which can be used as model for the behavior of the swarm member.

The low-level swarm model for the *RandomDirection* algorithm is shown in Figure 14, for the *FishSchooling* algorithm in Figure 15. The description for both algorithms can be found in Section 3.1.

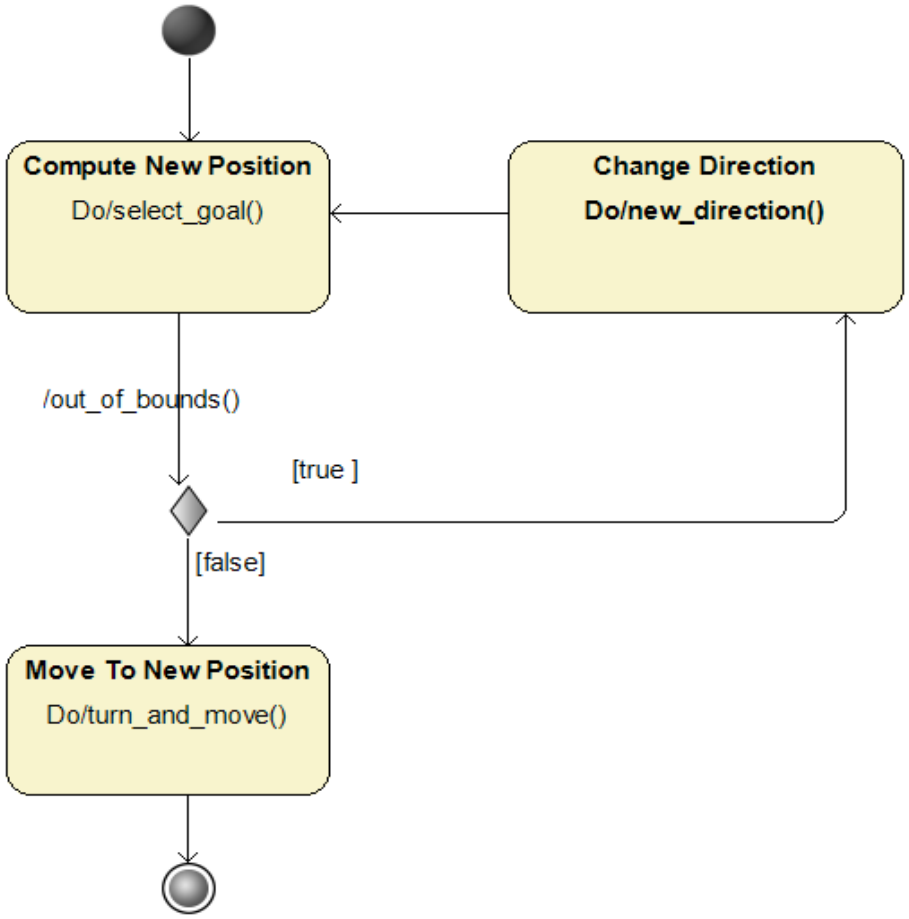

Figure 14 Low-level view of the RandomDirection algorithm.

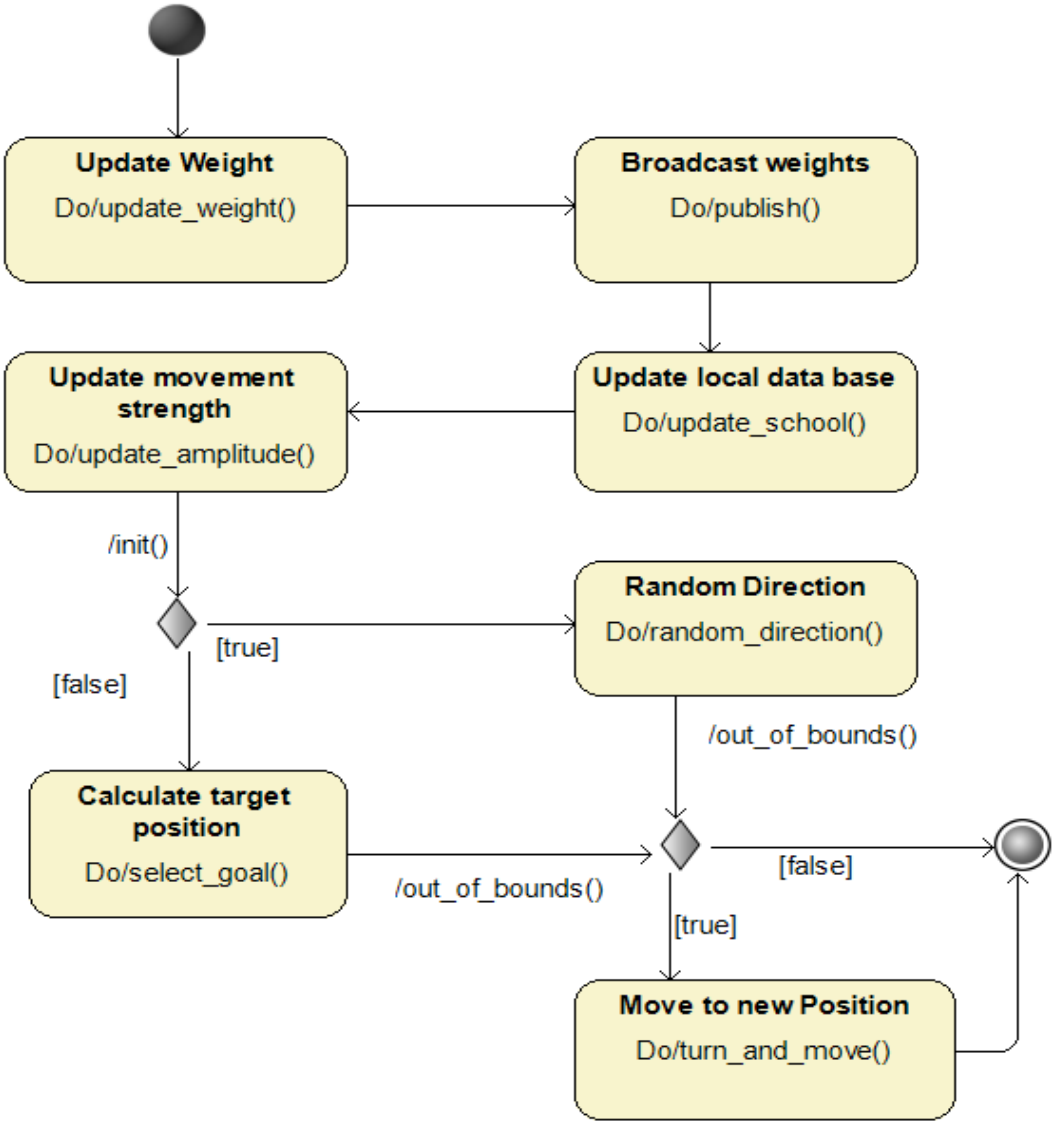

Figure 15 Low-level view of the FishSchooling algorithm.

5 State Machine Design for other use cases

In addition to the state machine for the SAR use case, we also present the state machines for the Logistics and Platooning use case. This work is still in progress and the state machines listed in the following subsections are drafts. The corresponding use case descriptions can be found in the vision scenarios in D2.2 – Final Vision Scenarios, and in the related WP8 deliverables due to M24 (D8.3, D8.5).

5.1 Use Case: Logistic

For this use case, the 1st level state machine is envisioned as shown in Figure 16. The Logistic behavior is a state machine running in parallel with Event Monitoring. The state machine is terminated in case of a *missionAbort*

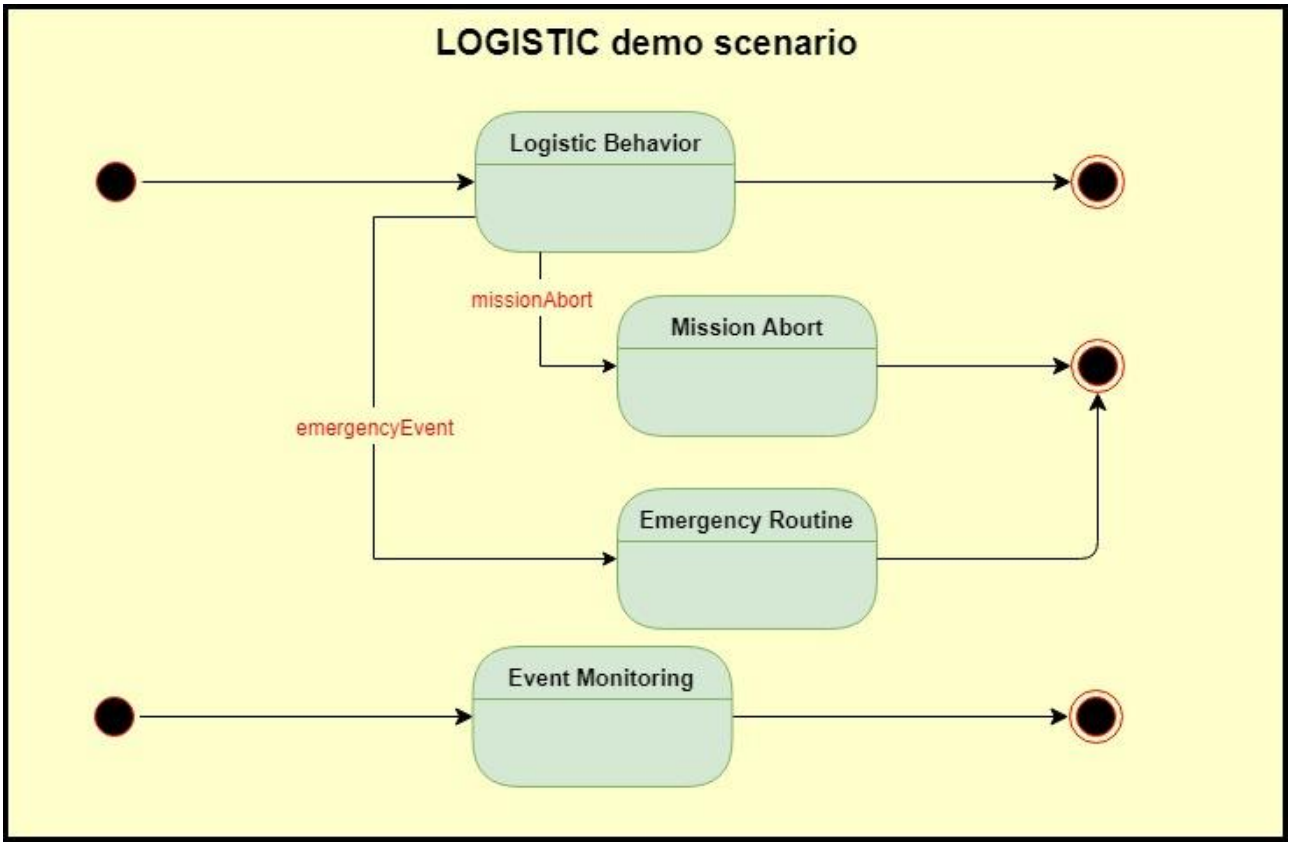

Figure 16 1st level state machine: logistic use case.

or *emergencyEvent* event using the corresponding states/behaviors.
The 2nd level of the state machine was designed in a very reduced way so far (see Figure 17). When a rover powers up in the *StartUp* state it is first in the Idle mode before the mission is started triggered by the *missionStart* event. The behavior in the Logistic Algorithm is not defined yet, and an open topic for the next period in WP4.

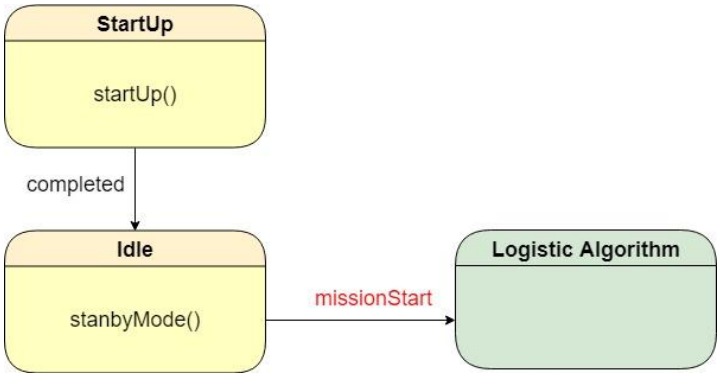

Figure 17 2nd level state machine for the rovers in the logistic use case.

5.2 Use Case: Platooning

For this use case, the 1st level state machine is modeled as shown in Figure 18. The Platooning behavior is a state machine running in parallel with Event Monitoring, Obstacle Detection and Status Message behaviors. The state machine is terminated in case of a *missionAbort* or *obstacleDetected* event using the corresponding states/behaviors.

The 2nd level behavior of the state machine is the following is given in Figure 19. Each truck is powered on by *StartUp* and goes into Idle mode. In case of a *shutDown* event, it triggers a power off. In case of the *missionStart* event it triggers a Shortest Path Algorithm behavior to calculate the shortest path to its destination. It starts the route with a Free Driving behavior following the lane. As soon as it meets another vehicle (using the *meetVehicle* event) the Select Role behavior is initiated to decide on being a leading or a following vehicle. If it is a leading vehicle, it turns again into the *FreeDriving* behavior until it a) meets again another vehicle or b) the mission is over (*missionOver* event). If it is a following vehicle, it follows the lead (instead of the lane) until a) the mission is over (*missionOver* event), or the way to destination changes, so the vehicle is again in the Free Driving behavior and follows the lane instead of the leading vehicle.

In addition, we designed also a preliminary state machine for the Emergency Routine. If an obstacle is detected (*obstacleDetected* event), the vehicle performs a Change Lane or Emergency Brake behavior, depend on lane availability (see Figure 20).

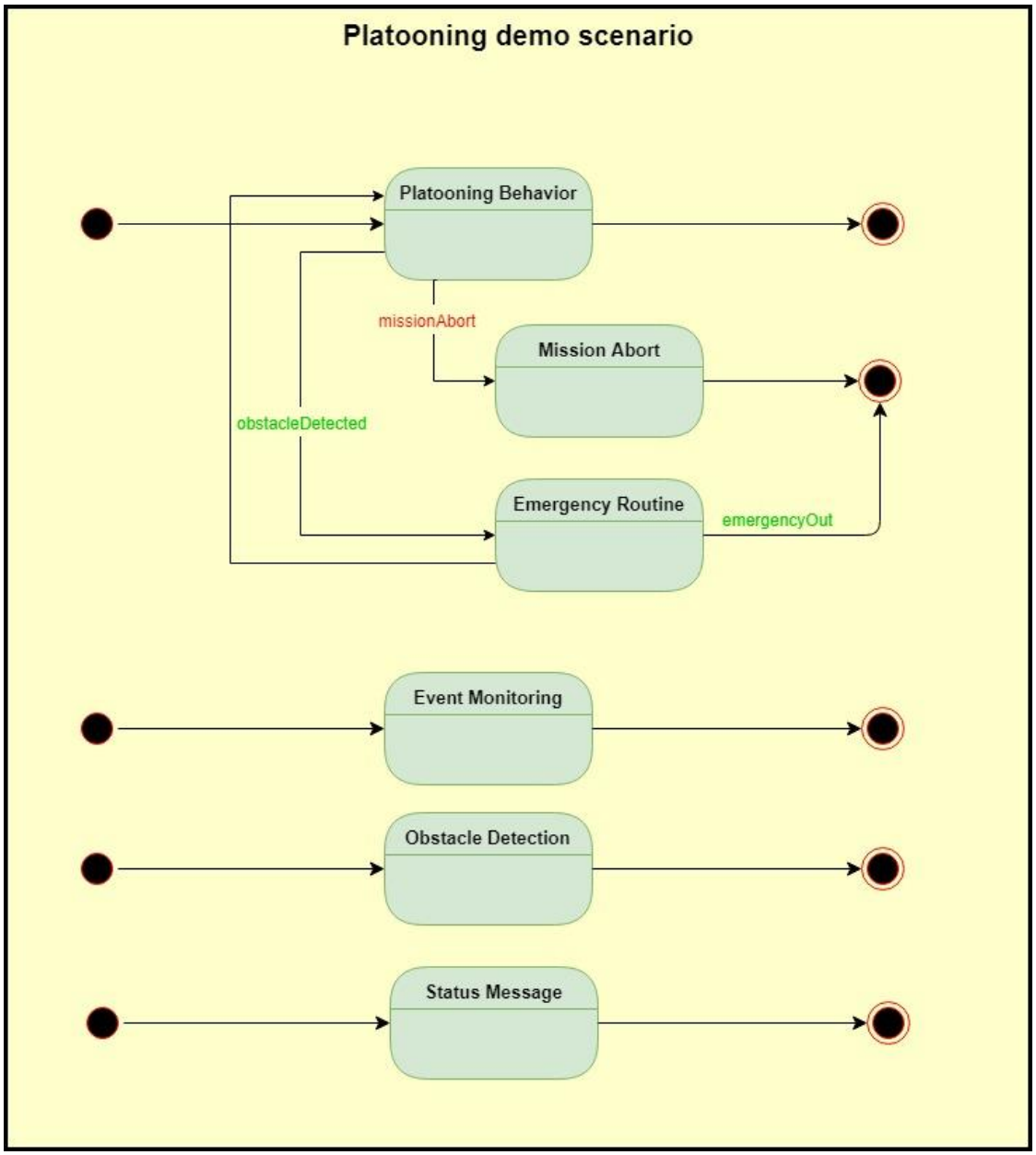

Figure 18 1st level state machine: platooning use case.

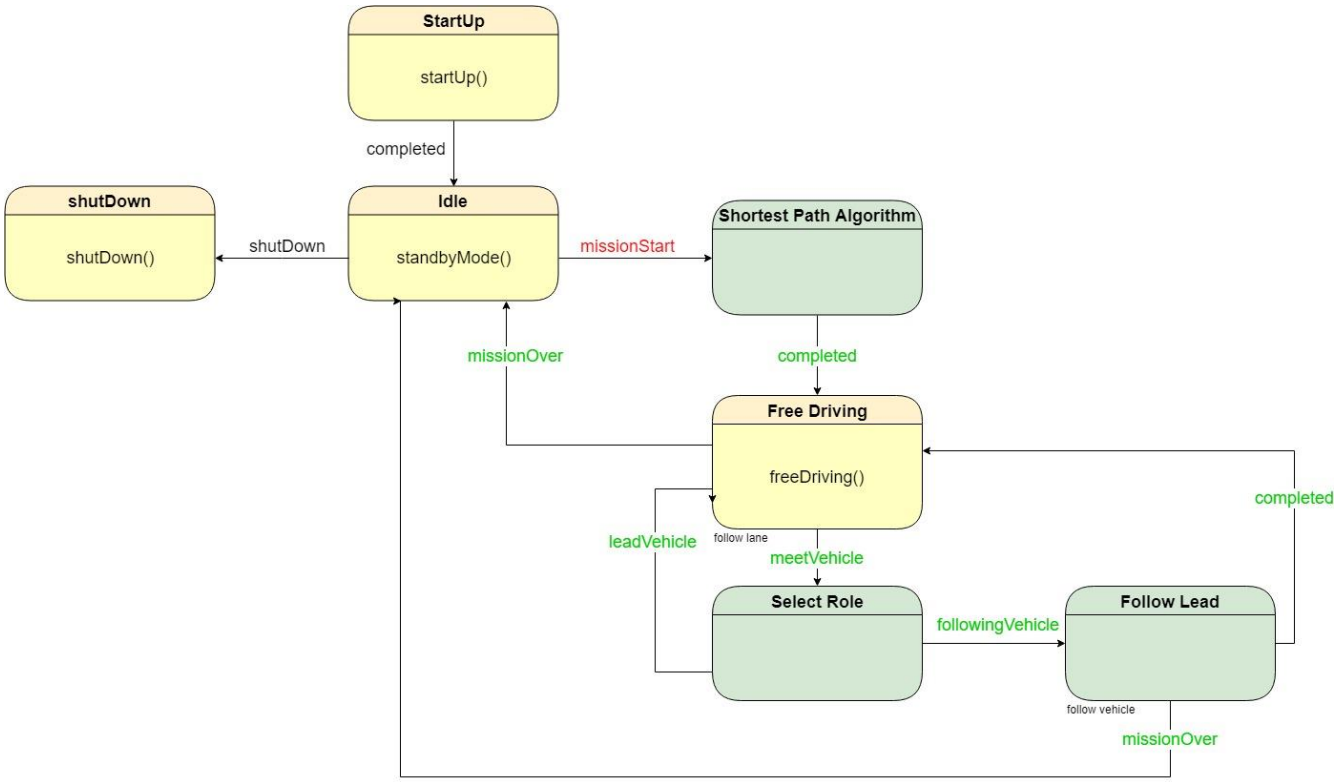

Figure 19 2nd level state machine for the vehicles behavior in the platooning use case.

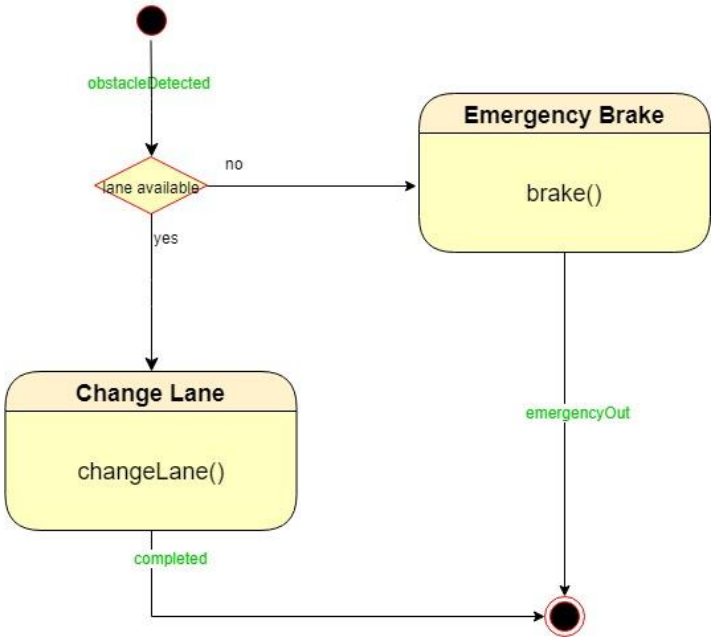

Figure 20 2nd level state machine for the vehicles emergency routine in the platooning use case.

6 Conclusions

In this deliverable we provide an updated swarm modeling library. We use the concepts of state machines to describe the behavior of the use cases, with a focus on the search and rescue use case. Therefore, we proposed several levels of the behaviors and focused on respective the swarm algorithms. Some of the ideas documented in this deliverable where already presented during the M18 review meeting.

In future work, the swarm intelligence algorithms will be elaborated for the other use cases: logistics and platooning. Furthermore, the proposed swarm algorithms will be split up into a compact library of atomic behaviors to enable the creation of new swarm algorithms.

7 References

[1] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques,* pp. 25-34, 1987.

[2] G. Vásárhelyi, C. Viragh, G. Somorjai, N. Tarcai, T. Szörenyi, T. Nepusz and T. Vicsek, "Outdoor flocking and formation flight with autonomous aerial robots," *IEEE/RSJ International Conference on Intelligent Robots and Systems ,* pp. 3866-3873, 2014.

[3] C. J. A. Bastos-Filho and D. O. Nascimento, "An enhanced fish school search algorithm," *BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence,* pp. 152-157, 2013.

[4] C. J. A. Bastos-Filho, F. B. de Lima Neto, A. J. C. C. Lins, A. Nascimento and M. P. Lima, "A novel search algorithm based on fish school behavior," *IEEE International Conference on Systems, Man and Cybernetics,* pp. 2646-2651, 2008.

[5] H. Min and Z. Wang, "Group escape behavior of multiple mobile robot system by mimicking fish schools," *IEEE International Conference on Robotics and Biomimetics,* pp. 320-326, 2010.

[6] T. W. Min and H. K. Yin, "A decentralized approach for cooperative sweeping by multiple mobile robots," *IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications,* pp. 380-385, 1998.

[7] D. Kurabayashi, J. Ota, T. Arai and E. Yoshida, "Cooperative sweeping by multiple mobile robots," *IEEE International Conference on Robotics and Automation,* pp. 1744-1749.

[8] M. Schwager, B. J. Julian and D. Rus, "Optimal coverage for multiple hovering robots with downward facing cameras," *IEEE International Conference on Robotics and Automation,* pp. 3515-3522, 2009.

[9] S. Antonatos, M. Polychronakis, P. Akritidis, K. G. Anagnostakis and E. P. Markatos, "Piranha: Fast and Memory-Efficient Pattern Matching for Intrusion Detection," in *Security and Privacy in the Age of Ubiquitous Computing*, Chiba, Japan, 2005.

[10] J. H. Mol, "Attacks on humans by the piranha Serrasalmus rhombeus in Suriname," *Studies on Neotropical Fauna and Environment,* vol. 41, no. 3, 2006.

[11] H. Queiroz and A. E. Magurran, "Safety in numbers? Shoaling behavior of the Amazonian red-bellied piranha," *Biology Letters,* vol. 1, no. 2, 2005.

[12] L. Panait and S. Luke, "Ant foraging revisited," in *Proc. International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, 2004.

[13] M. Beekman, D. J. T. Sumpter and F. L. W. Ratnieks, "Phase transition between disordered and ordered foraging in Pharaoh's ants," *Proceedings of the National Academy of Sciences,* vol. 98, no. 17, 2001.

[14] A. Nakajima, S. Ishihara, D. Imoto and S. Sawai, "Rectified directional sensing in long-range cell migration," *Nature Communications,* vol. 5, 2014.

[15] L. Liu, Y. Song, H. Ma and X. Zhang, "Physarum optimization: A biology-inspired algorithm for minimal exposure path problem in wireless sensor networks," in *Proc. IEEE INFOCOM*, 2012.

[16] S. Mirjalili, S. M. Mirjalili and A. Lewis, "Grey Wolf Optimizer," *Advances in Engineering Software,* vol. 69, 2014.

[17] B. Bethke, M. Valenti and J. How, "Cooperative Vision Based Estimation and Tracking Using Multiple drones," *Advances in Cooperative Control and Optimization, Lecture Notes in Control and Information Sciences,* vol. 369, 2007.

[18] J. Peterson, H. Chaudhry, K. Abdelatty, J. Bird and K. Kochersberger, "Online Aerial Terrain Mapping for Ground Robot Navigation," *Sensors,* vol. 18, no. 2, 2018.

[19] X.-S. Yang, Nature-Inspired Metaheuristic Algorithms, Luniver Press, 2008.

[20] V. M. a. A. C. M. Dorigo, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* vol. 26, no. 1, pp. 29-41, 1996.

[21] D. Chrysostomou, A. Gasteratos, L. Nalpantidis and G. C. Sirakoulis, "Multi-view 3D scene reconstruction using ant colony optimization techniques," vol. 11, no. 23, 2012.

[22] H. H. Thomas Schmickl, "BEECLUST: A Swarm Algorithm Derived from Honeybees. Derivation of the Algorithm, Analysis by Mathematical Models and Implementation on a Robot Swarm," in *Bio-inspired Computing and Networking*, CRC Press, 2011, pp. 95-137.

[23] H. Ghayour, M. Abdellahi and M. Bahmanpour, "Artificial intelligence and ceramic tools: Experimental study, modeling and optimizing," *Ceramics International,* no. 40, pp. 13470-13479, 2015.

[24] G. Di Caro, F. Ducatelle and L. M. Gambardella, "AntHocNet: An Ant-Based Hybrid Routing Algorithm for Mobile Ad Hoc Networks," *Parallel Problem Solving from Nature,* no. 3242, pp. 461-470, September 2004.

8 Acronyms

Acronym	Explanation
CPS	Cyber-Physical Systems
GWO	Grey Wolf Optimizer
SAR	Search and Rescue
SysML	System Modeling Language

9 List of Figures