

# Supplementary material for the paper titled “Reward-based Autonomous Online Learning Framework for Resilient Cooperative Target Monitoring using a Swarm of Robots”

## Abstract

This document serves as supplementary material for the paper titled ‘Reward-based Autonomous Online Learning Framework for Resilient Cooperative Target Monitoring using a Swarm of Robots.’ This contains details about the control law, parametric search space for parametric studies, baselines used for comparative studies, some more MATLAB simulation results, and a description of the simulation setup for the ROS-Gazebo simulations, along with the partial code for both MATLAB and Python (ROS-Gazebo) simulations.

## 1 Control Law

*Translational Control Law:* For the  $i^{th}$  robot, the translational control law consists of two terms as given below

$$\bar{v}_{t,i} = \bar{v}_{t,i}^R + \Delta \bar{v}_{t,i} \quad (1)$$

where  $\bar{v}_{t,i}^R$  is the  $i^{th}$  robot’s reference command signal responsible for chasing the target, and  $\Delta \bar{v}_{t,i}$  is the  $i^{th}$  robot’s correction control signal responsible for avoiding collisions with other robots.

Denote  $R_{t,i} \in \mathbb{R}^{2 \times 2}$  as the  $i^{th}$  robot’s body-global rotation matrix at time  $t$ , defined as  $R_{t,i} = \begin{bmatrix} \cos \phi_{t,i} & -\sin \phi_{t,i} \\ \sin \phi_{t,i} & \cos \phi_{t,i} \end{bmatrix}$ , and  $\hat{R}_{t,i} = \begin{bmatrix} \cos \hat{\phi}_{t,i}^{P_i} & -\sin \hat{\phi}_{t,i}^{P_i} \\ \sin \hat{\phi}_{t,i}^{P_i} & \cos \hat{\phi}_{t,i}^{P_i} \end{bmatrix}$ , where  $\hat{\phi}_{t,i}^{P_i}$  is the robot’s yaw angle estimate via its proprioception.

If the  $i^{th}$  robot has detected the target ( $d_{t,i} > 0$ ) or one of its neighbors has detected the target ( $d_{t,j} > 0, j \in \Omega_{t,i}$ ), the  $i^{th}$  robot’s reference command signal  $\bar{v}_{t,i}^R$  is given as

$$\bar{v}_{t,i}^R = k_1 \hat{R}_{t,i}' \frac{\Delta \hat{x}_{t,B}^i}{\|\Delta \hat{x}_{t,B}^i\|} (\|\Delta \hat{x}_{t,B}^i\| - d_S) \quad (2)$$

where  $(\cdot)'$  represents the transpose operation,  $\|\cdot\|$  is the 2-norm or the Euclidean norm,  $k_1 > 0$  is a control parameter.  $\Delta \hat{x}_{t,B}^i := \hat{x}_{t,B}^i - \hat{x}_{t,i}^{P_i}$ , where  $\hat{x}_{t,B}^i$  is the target’s position estimate given by robot  $i$  at time  $t$ , and  $\hat{x}_{t,i}^{P_i}$  is the  $i^{th}$  robot’s position estimate given

by its proprioception at time  $t$ .  $d_S > 0$  ( $m$ ) indicates the distance each robot should maintain from the target while chasing it.

If the  $i^{th}$  robot has not detected the target ( $d_{t,i} = 0$ ) and either none of its neighbors have detected the target ( $d_{t,j} = 0, \forall j \in \Omega_{t,i}$ ) or it has no neighbors ( $n_{t,i} = 0$ ), then the  $i^{th}$  robot executes a search pattern inspired by the food foraging pattern used by *Oxyrrhis Marina*. The robot first chooses a random direction to move towards. With its longitudinal body axis aligned with that direction, it moves in that direction using its longitudinal velocity control while doing a growing sinusoidal maneuver using its lateral velocity control to cover more area as it moves. After  $T_s$  discrete-time steps, the robot randomly chooses a new direction and repeats the process.

Further, we assume that each robot is equipped with a collision avoidance system, which ensures that while chasing the target, robots do not collide. Considering eq.(1), this behavior can be modeled by the correction control signal  $\Delta \bar{v}_{t,i}$  for the  $i^{th}$  robot by using an *inter-robot collision avoidance* control law given as follows:

$$\Delta \bar{v}_{t,i} = -k_2 R'_{t,i} \frac{x_{t,p_t^i} - x_{t,i}}{\|x_{t,p_t^i} - x_{t,i}\|^2} \quad (3)$$

where  $(\cdot)'$  represents the transpose operation,  $\|\cdot\|$  is the 2-norm or the Euclidean norm,  $k_2 > 0$  is a control parameter,  $p_t^i \in [N] \setminus \{i\}$  is the index of the robot spatially nearest to  $i^{th}$  robot at time  $t$ , formally defined as  $p_t^i := \arg \min_{j \in [N] \setminus \{i\}} \|x_{t,j} - x_{t,i}\|$ . Thus,  $x_{t,p_t^i}$  is the position vector of the robot spatially nearest to the  $i^{th}$  robot at time  $t$ .

**Heading Control Law:** If the  $i^{th}$  robot has detected the target ( $d_{t,i} > 0$ ) or one of its neighbors has detected the target ( $d_{t,j} > 0, j \in \Omega_{t,i}$ ), then the robot is required to yaw in such a way that its heading direction should point towards its estimate of the target's position  $\hat{x}_{t,B}^i$ . The angle between  $\Delta \hat{x}_{t,B}^i$  and the  $i^{th}$  robot's heading direction  $h_{t,i} = [\cos \phi_{t,i} \quad \sin \phi_{t,i}]'$ , with respect to the  $\Delta \hat{x}_{t,B}^i$  direction, can be obtained as  $\Delta \phi_{t,err}^i = \text{atan2}(h_{t,i} \times \Delta \hat{x}_{t,B}^i, h_{t,i} \cdot \Delta \hat{x}_{t,B}^i)$ , where the first argument involves a cross-product and the second argument involves dot-product. As per the heading angle requirement,  $i^{th}$  robot's yaw control law can be given as

$$\bar{w}_{t,i} = k_3 \Delta \phi_{t,err}^i \quad (4)$$

where  $k_3 > 0$  is a control parameter.

If the  $i^{th}$  robot has not detected the target ( $d_{t,i} = 0$ ) and either none of its neighbors have detected the target ( $d_{t,j} = 0, \forall j \in \Omega_{t,i}$ ) or it has no neighbors ( $n_{t,i} = 0$ ), then the  $i^{th}$  robot executes a search pattern inspired by the food foraging pattern used by *Oxyrrhis Marina*, where the robot chooses a random direction to move towards, such that its heading direction aligns with that randomly chosen direction. After  $T_s$  discrete-time steps, the robot randomly chooses a new direction and repeats the process.

## 2 Baselines

### 2.1 Averaging-Consensus based Fusion (ACF)

**Local estimation phase:** if the target is detected by  $i^{th}$  robot's exteroception, i.e.,  $d_{t,i} > 0$ , then

$$\hat{x}_{t,B}^{I_i} = 0.5\hat{x}_{t,B}^{S_i} + 0.5\hat{x}_{t-1,B}^i \quad (5)$$

otherwise, if the target is undetected by  $i^{th}$  robot's exteroception, i.e.,  $d_{t,i} = 0$ , then

$$\hat{x}_{t,B}^{I_i} = \hat{x}_{t-1,B}^i \quad (6)$$

**Communication phase:** The  $i^{th}$  robot broadcasts the information  $\{t, i, d_{t,i}, \hat{x}_{t,B}^{I_i}\}$  and receives the information  $\{t, j, d_{t,j}, \hat{x}_{t,B}^{I_j}\}$  from its communicating neighbors  $j \in \Omega_{t,i}$ .

**Social estimation phase:** With  $n_{t,i} = |\Omega_{t,i}|$ , we have

$$\hat{x}_{t,B}^i = \frac{1}{n_{t,i} + 1} \sum_{j \in \Omega_{t,i}} \hat{x}_{t,B}^{I_j} \quad (7)$$

### 2.2 Kalman-Consensus based Fusion (KCF)

Consider the covariance of  $\hat{x}_{t,B}^{I_i}$  and  $\hat{x}_{t-1,B}^i$  as  $C_{t,B}^{I_i}$  and  $C_{t-1,B}^i$ , respectively. Consider  $C_{t,B}^{S_i}$  as the covariance for  $\hat{x}_{t,B}^{S_i}$ .

**Local estimation phase:** if the target is detected by  $i^{th}$  robot's exteroception, i.e.,  $d_{t,i} > 0$ , then

$$(C_{t,B}^{I_i})^{-1} = \frac{1}{2} \left( (C_{t-1,B}^i)^{-1} + (C_{t,B}^{S_i})^{-1} \right) \quad (8)$$

$$\hat{x}_{t,B}^{I_i} = \left( (C_{t-1,B}^i)^{-1} + (C_{t,B}^{S_i})^{-1} \right)^{-1} \left( (C_{t,B}^{S_i})^{-1} \hat{x}_{t,B}^{S_i} + (C_{t-1,B}^i)^{-1} \hat{x}_{t-1,B}^i \right) \quad (9)$$

otherwise, if the target is undetected by  $i^{th}$  robot's exteroception, i.e.,  $d_{t,i} = 0$ , then

$$C_{t,B}^{I_i} = C_{t-1,B}^i \quad (10)$$

$$\hat{x}_{t,B}^{I_i} = \hat{x}_{t-1,B}^i \quad (11)$$

**Communication phase:** The  $i^{th}$  robot broadcasts the information  $\{t, i, d_{t,i}, \hat{x}_{t,B}^{I_i}, C_{t,B}^{I_i}\}$  and receives the information  $\{t, j, d_{t,j}, \hat{x}_{t,B}^{I_j}, C_{t,B}^{I_j}\}$  from its communicating neighbors  $j \in \Omega_{t,i}$ .

**Social estimation phase:** With  $n_{t,i} = |\Omega_{t,i}|$ , we have

$$(C_{t,B}^i)^{-1} = \frac{1}{n_{t,i} + 1} \sum_{j \in \Omega_{t,i}} (C_{t,B}^{I_j})^{-1} \quad (12)$$

$$\hat{x}_{t,B}^i = \left( \sum_{j \in \Omega_{t,i}} (C_{t,B}^{I_j})^{-1} \right)^{-1} \sum_{j \in \Omega_{t,i}} (C_{t,B}^{I_j})^{-1} \hat{x}_{t,B}^{I_j} \quad (13)$$

### 3 MATLAB simulation details and results

The proposed AOL framework is evaluated using a simulation setup involving  $N = 5, 10, 20, 30$  robots executing the cooperative target monitoring task discussed in the problem formulation. The communication range  $R_{comm.}$  and the communication link drop probability  $p_{ld}$  are set to be  $30\text{ m}$  and  $0.1$ , respectively, with the limit on the number of communication neighbors as  $n_l = 3$ . The exteroceptive sensor model's constraints are set as  $R_{FOV} = 15\text{ m}$ ,  $\theta_{FOV} = 160\text{ degrees}$ , with the target visual loss probability as  $p_{vl} = 0.1$ . The parameters for the detection confidence model are set as  $r_o = 10\text{ m}$ , and  $b_o = 0.1$ .

The simulation results are averaged over 100 simulation runs. Each run involves a time horizon of  $T = 600$  discrete time steps, with a sampling period of  $\Delta T = 0.1\text{ sec}$ . The robots follow the control law described in the problem formulation while trying to maintain a safe distance of  $d_S = 8\text{ m}$  from the target. The target randomly changes its velocity and yaw rate after every  $5\text{ seconds}$ . The robots and the target always stay inside a square region of side length  $200\text{ m}$  by overriding their control laws to get away from the region boundary. At the start of each simulation run, the robots are always spawned near the center of the square region, whereas the target is spawned randomly but sufficiently near to the robots so that at least one of the robots is likely to detect it at the start of the run. This is done since the main focus of this paper is not the target search but target detection, tracking, and monitoring.

For all three AOL variants, a simulation-based parametric study is carried out to find a suitable set of parameters that result in desirable performance. The parametric search is done over the following set of parameters:  $T_p = [5, 10, 15, 20, 25]$ ,  $\eta_w = [5, 10, 15, 20]$ ,  $\eta_\alpha = [0.01, 0.1, 1, 5, 10]$ ,  $e_{a1} = [5, 10, 20, 30]$ ,  $e_{a2} = [0.01, 0.1, 1, 5, 10]$ ,  $e_{w1} = [5, 10, 20, 30]$ ,  $e_{w2} = [0.01, 0.1, 1, 5, 10]$ ,  $p_{mag} = [0.01, 0.1, 1, 5, 10]$ . Based on the parametric study, the parameters for AOL-ver1 are set as  $D_o = 15\text{ m}$ ,  $T_p = 15$ ,  $\eta_w = 15$ , and  $\eta_\alpha = 0.01$ ; that of AOL-ver.2 are set as  $T_p = 15$ ,  $\eta_w = 15$ ,  $e_{a1} = 10$ ,  $e_{a2} = 0.1$ , and  $p_{mag} = 0.1$ ; that of AOL-ver.3 are set as  $T_p = 15$ ,  $e_{w1} = 1$ ,  $e_{w2} = 0.01$ ,  $e_{a1} = 20$ ,  $e_{a2} = 5$ , and  $p_{mag} = 0.1$ . Further, the three variants of AOL are compared with two baselines – Average-Consensus Fusion (ACF) and Kalman-Consensus Fusion (KCF).

Figures 1, 2, 3, and 4 show results for no sensor failure scenario. Whereas figures 5, 6, 7, and 8 show results for only exteroception failure (in 50% robot population) scenario. Note that these scenarios also have uncertainty in the form of communication link drop probability  $p_{ld} = 0.1$  and target visual loss probability  $p_{vl} = 0.1$ .

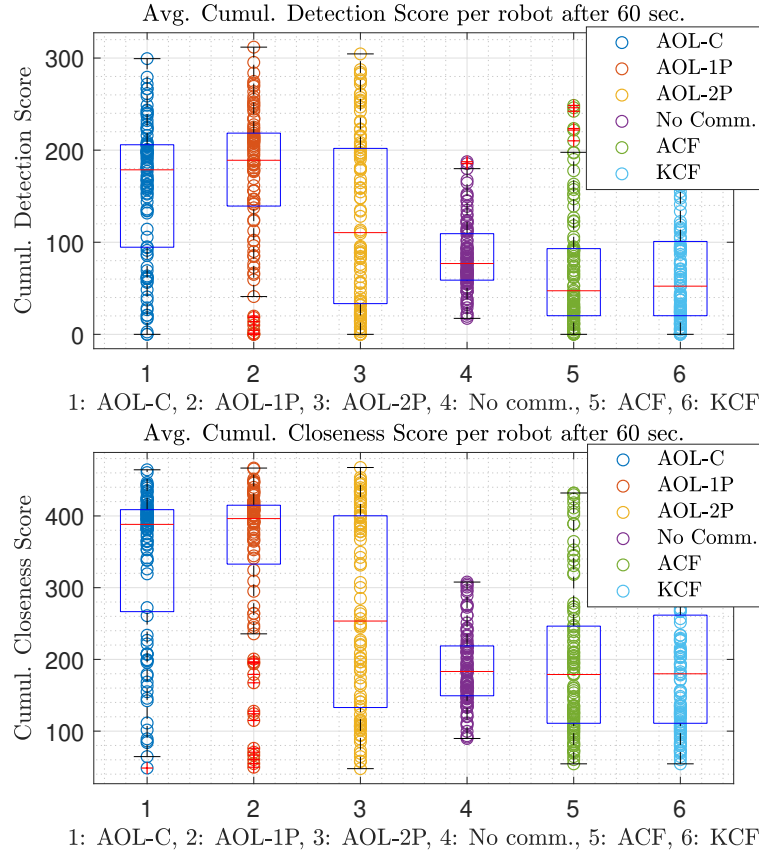


Figure 1: Comparison results – no sensor failures; 100 sim. runs for each method,  $N = 5$

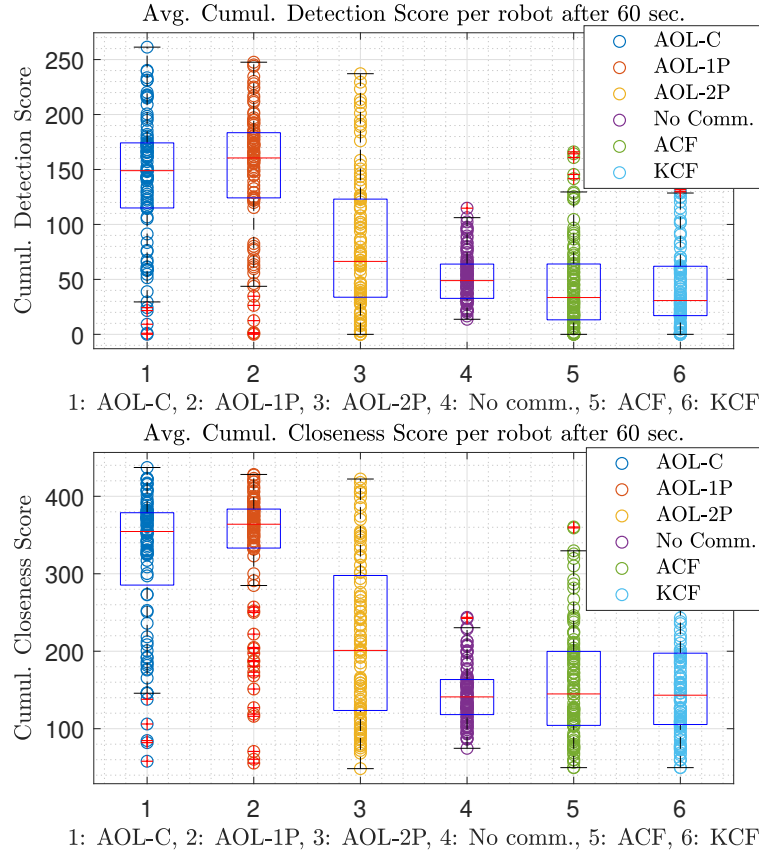


Figure 2: Comparison results – no sensor failures; 100 sim. runs for each method,  $N = 10$

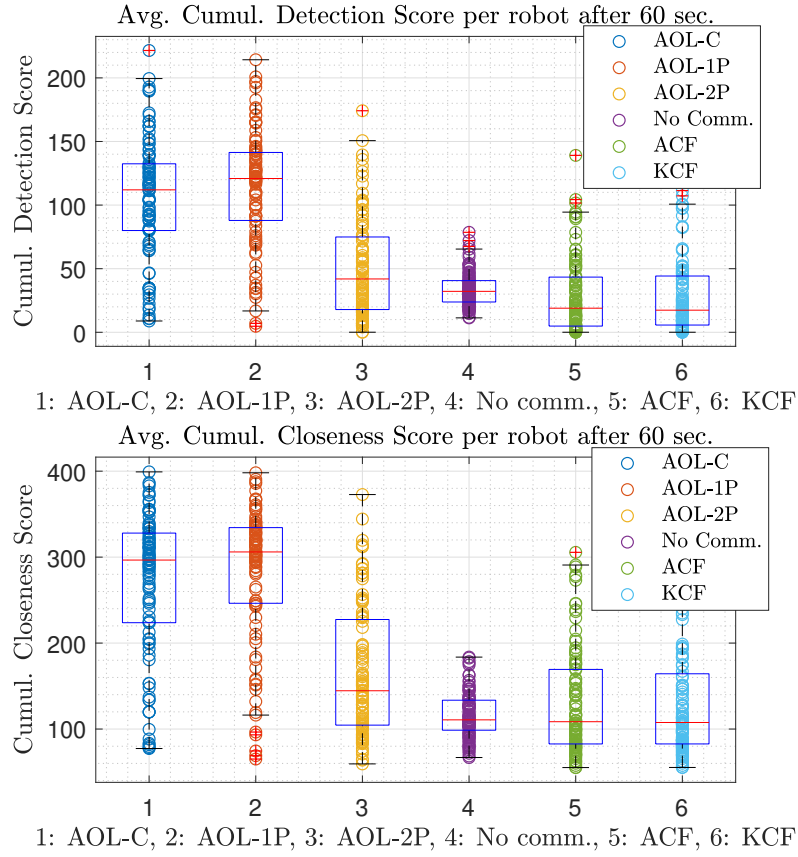


Figure 3: Comparison results – no sensor failures; 100 sim. runs for each method,  $N = 20$

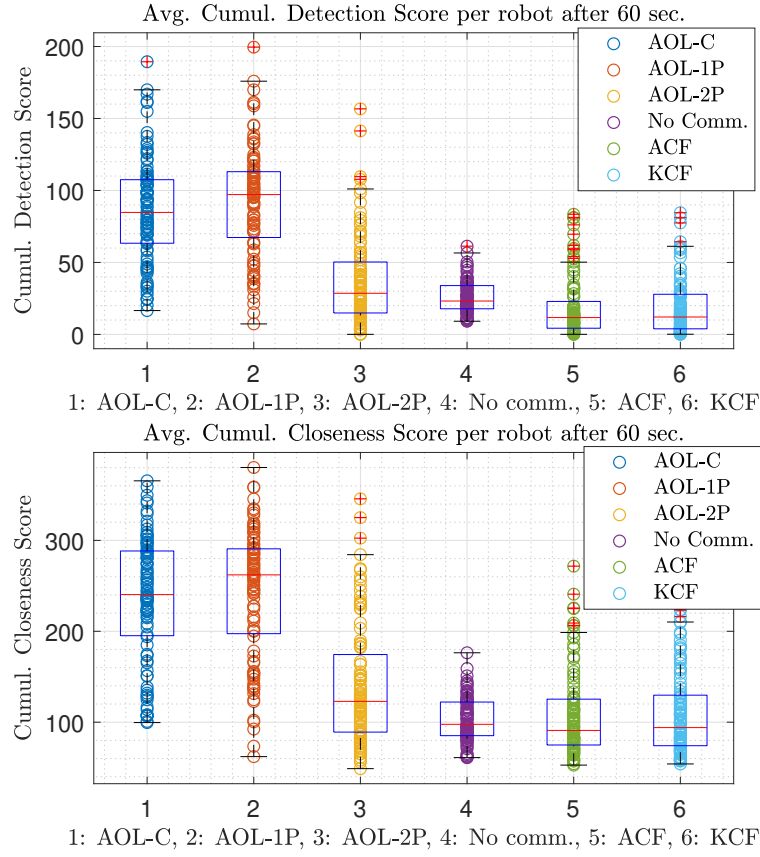


Figure 4: Comparison results – no sensor failures; 100 sim. runs for each method,  $N = 30$



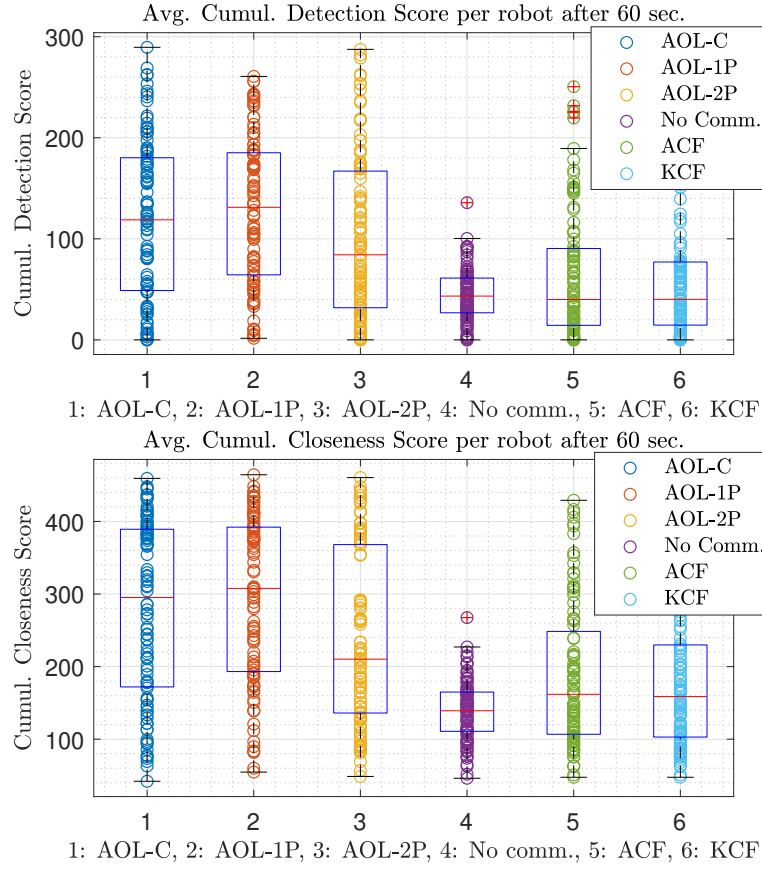


Figure 5: Comparison results – exteroceptive sensor failures in 50% of the total no. of robots; 100 sim. runs for each method,  $N = 5$

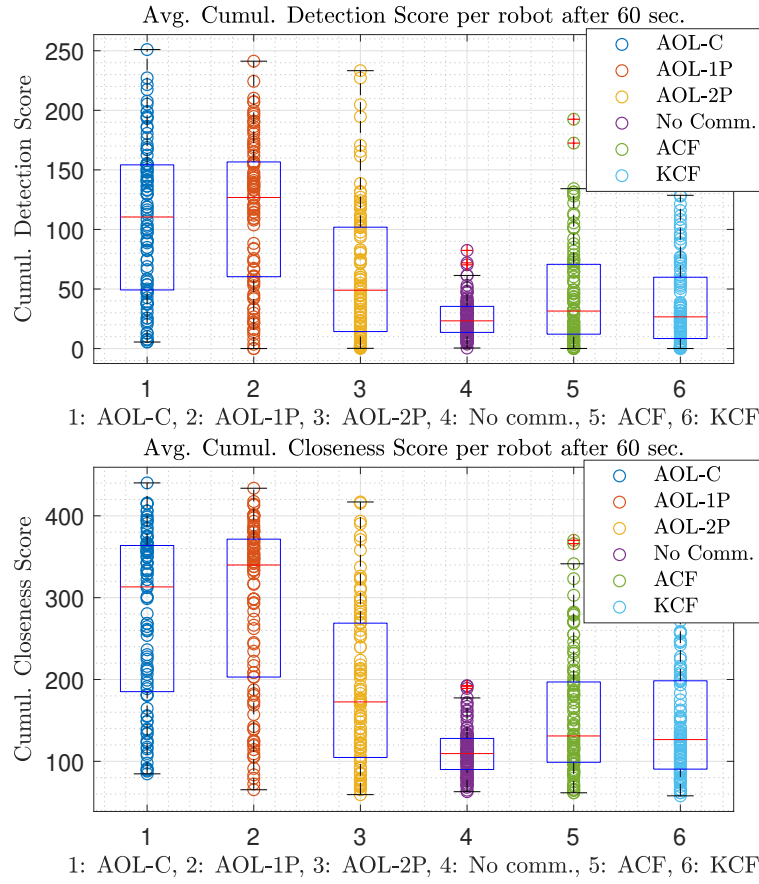


Figure 6: Comparison results – exteroceptive sensor failures in 50% of the total no. of robots; 100 sim. runs for each method,  $N = 10$

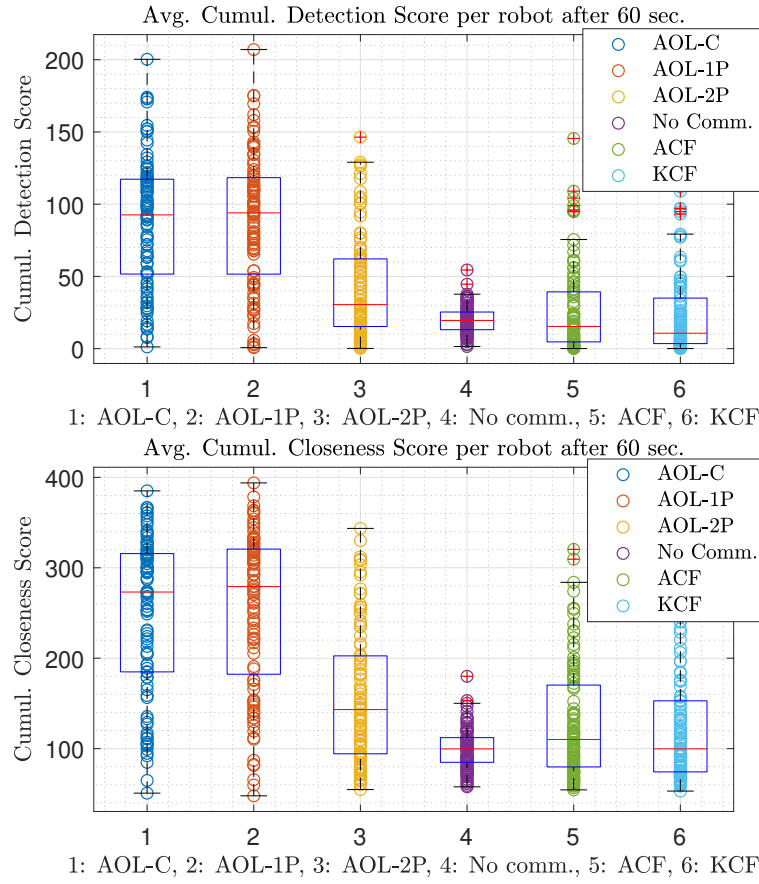


Figure 7: Comparison results – exteroceptive sensor failures in 50% of the total no. of robots; 100 sim. runs for each method,  $N = 20$

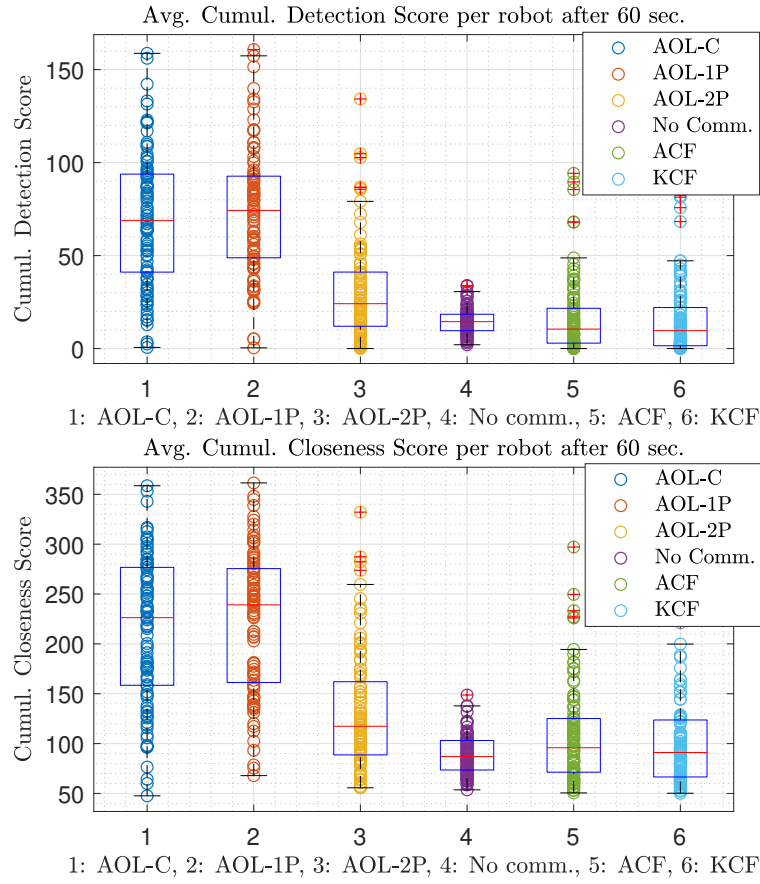


Figure 8: Comparison results – exteroceptive sensor failures in 50% of the total no. of robots; 100 sim. runs for each method,  $N = 30$

## 4 ROS-Gazebo setup

In order to test the AOL framework on robots, we used a multi-robot setup in the Gazebo simulator as seen in Fig. 9. The simulation setup contains 1 Botsync’s Copernicus (Target) and 6 Botsync’s Voltas (Swarm Robots). Each Volta has a 2D lidar and Intel RealSense camera onboard. Copernicus consists of just 2D Lidar and is teleoperated in the simulation. Each Volta is running custom-trained YOLOv5 for detecting the Copernicus. Using a custom message the robots communicate the required parameters between themselves. The confidence score of YOLOv5 is passed to the AOL fusion layer and to further test the robustness of the AOL framework an additional bias dependent upon the confidence score is added to the detected target location.

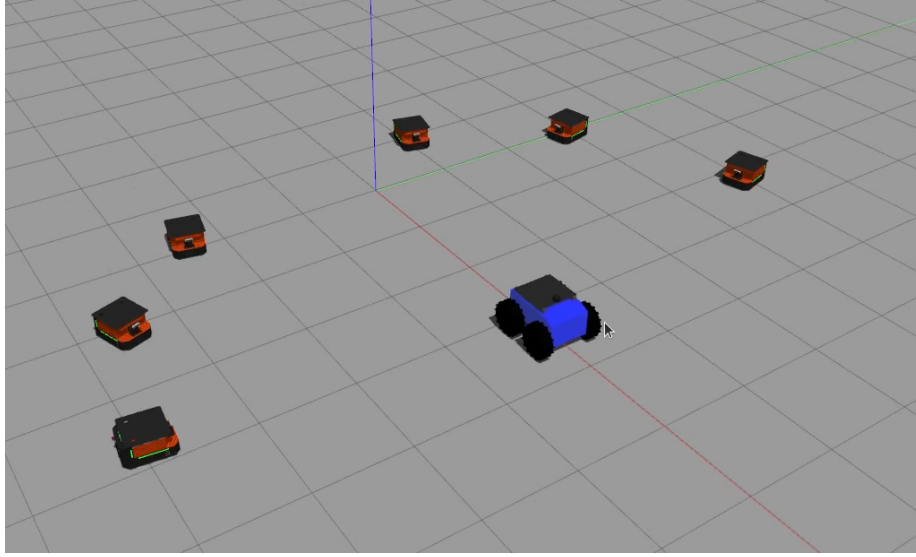


Figure 9: Simulation Setup

Overall architecture can be seen in the Fig. 10. The *multi\_volta.launch* node spawns a swarm of Voltas and Copernicus in the Gazebo with the relevant sensor suite. The *robot\_controller.py* runs for individual Voltas and is responsible for controlling the linear and angular velocity of the robot depending upon the obstacles and the target location to be reached. The *aol\_framework.py* determines the target location using its own prediction and the neighbor’s predicted target location.

The received depth image from the RealSense camera can be seen in Fig. 11. In order to find the location of the target (i.e. Copernicus Robot) YOLOv5 is used along with LiDAR. A custom YOLOv5 model was trained on Copernicus and Volta robot images. Subsequently, the trained YOLOv5 model’s score and bounding box were integrated with data coming from LiDAR and the estimate was provided to the AOL framework. The bias in the estimate is proportional to  $e^{1-detection\_score} - 1$ . The Output from YOLOv5 can be seen in the Fig. 13. For the LiDAR data, K-Means clustering can be used to detect the object and fusing this data with the camera can be used to

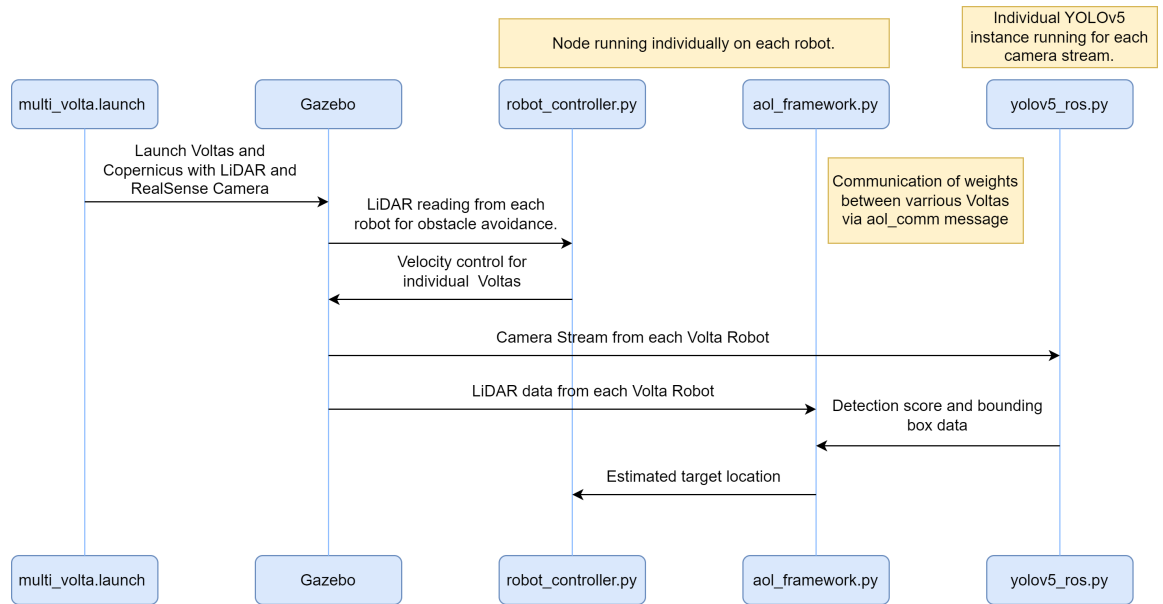


Figure 10: Architecture diagram of the simulation

detect the location of the known target. Within AOL framework, the model's detection outputs, including bounding box coordinates and confidence scores, are processed to provide the best estimate of the target robot.

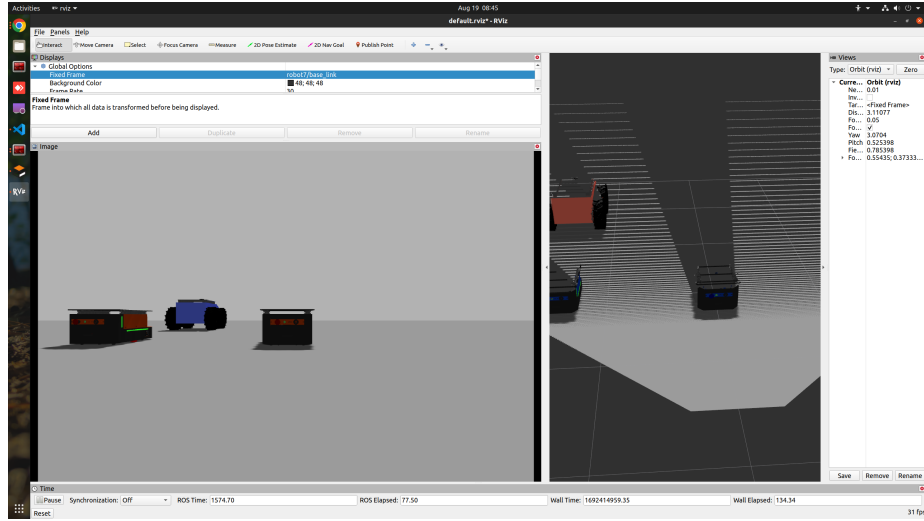


Figure 11: Visualization of the environment using the camera-based exteroception present on Volta

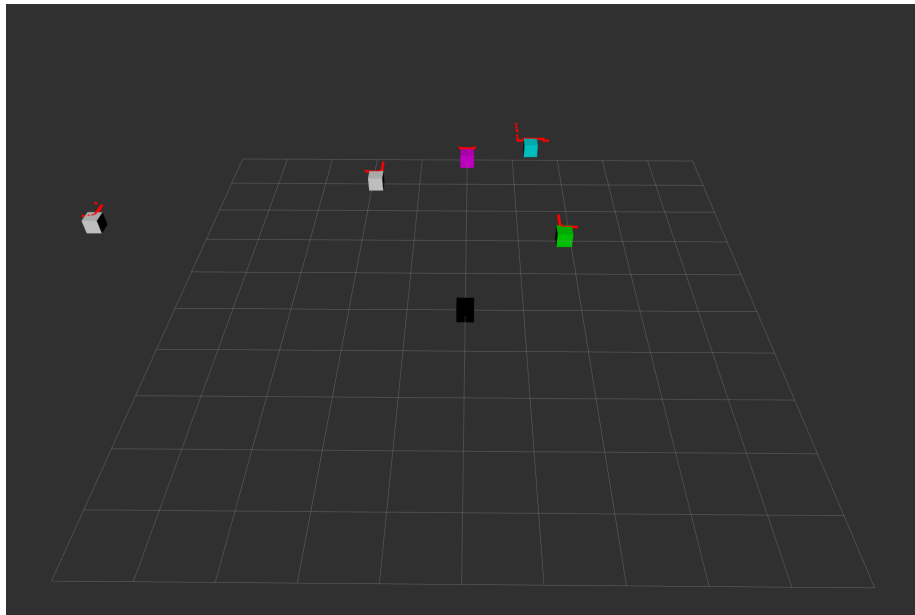


Figure 12: Demonstration of K-Means clustering for finding objects using LiDAR data

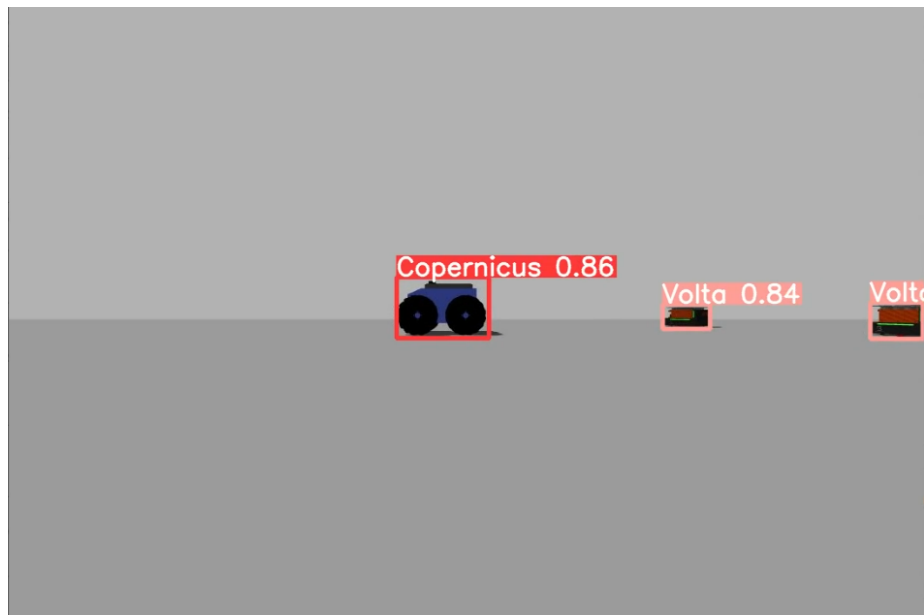


Figure 13: Detection output of YOLOv5



## 5 AOL Code

Partial code for both MATLAB and Python (ROS-Gazebo) simulations is presented in this section. The full code will be released after the paper's acceptance.

### 5.1 MATLAB Code

---

```
function [xhatI_Tgt_i, alpha_i] =
    AOL_FusionLayer1(xhatS_Tgt_i, xhat_Tgt_i_prev,
        alpha_hat_i, alphapr_hat_i, det_Tgt_i, det_Tgt_neigh)

det_i_lg = double(det_Tgt_i > 0);

if det_Tgt_i > 0 % tgt detected
    alpha_i = (alpha_hat_i)/(alpha_hat_i + alphapr_hat_i);
else % tgt undetected
    alpha_i = 0;
end

xhatI_Tgt_i = alpha_i.*xhatS_Tgt_i +
    (1-alpha_i).*xhat_Tgt_i_prev;

end
```

---

```
function [xhat_Tgt_i, wii] = AOL_FusionLayer2(xhatI_Tgt_i,
    xhatI_Tgt_j, what_i, what_j)

wii = what_i./(what_i + sum(what_j));
wij = what_j./(what_i + sum(what_j));

xhat_Tgt_i = wii.*xhatI_Tgt_i + sum(wij.*xhatI_Tgt_j,2);

end
```

---

```
function [alpha_hat_i_nxt, alphapr_hat_i_nxt, what_i_nxt] =
    AOL_LearningPhaseC(t, Tp, eta_alp, eta_w, det_Tgt_i,
    xhatI_Tgt_j, xhatI_Tgt_i, xhatS_Tgt_i, xhat_Tgt_i_prev,
    xhat_Tgt_j_prev, what_j, what_i, alpha_hat_i,
    alphapr_hat_i)

whatij = [what_j what_i];
xhatI_Tgt_ij = [xhatI_Tgt_j xhatI_Tgt_i];
xhat_Tgt_ij = [xhat_Tgt_j_prev xhat_Tgt_i_prev];
[wmax, IDstar] = max(whatij);

xhat_star = xhatI_Tgt_ij(:, IDstar);
```

```

if ceil(t/Tp) == floor(t/Tp)
    what_i = 1;
    alpha_hat_i = 1;
    alphapr_hat_i = 1;
end

alpha_hat_i_nxt =
    alpha_hat_i*exp(-eta_alp*min(norm(xhatS_Tgt_i -
    xhat_star)/15,1));
alphapr_hat_i_nxt =
    alphapr_hat_i*exp(-eta_alp*min(norm(xhat_Tgt_i_prev -
    xhat_star)/15,1));
det_Tgt_i_lgcl = det_Tgt_i;
what_i_nxt = what_i*exp(-eta_w*(1-det_Tgt_i_lgcl));
end



---


function [alpha_hat_i_nxt, alphapr_hat_i_nxt, what_i_nxt] =
    AOL_LearningPhase1P(t,Tp, eta_w, det_Tgt_i, what_i,
    alpha_hat_i, alphapr_hat_i, alpha_i, del_alpha_i,
    del_detTgt_i, p_mag, e_a1, e_a2)

if t == 1
    e_pa1 = randi([0 1])*p_mag;
    e_pa2 = p_mag - e_pa1;
else
    e_pa1 = 0;
    e_pa2 = 0;
end

if ceil(t/Tp) == floor(t/Tp)
    what_i = 1;
    alpha_hat_i = 1;
    alphapr_hat_i = 1;
    e_pa1 = randi([0 1])*p_mag;
    e_pa2 = p_mag - e_pa1;
end

det_Tgt_i_lgcl = det_Tgt_i;
what_i_nxt = what_i*exp(-eta_w*(1-det_Tgt_i_lgcl));

alpha_hat_i_nxt =
    alpha_hat_i*exp(e_a1*del_alpha_i*del_detTgt_i +
    e_pa1*(1-det_Tgt_i) + e_a2*alpha_i*det_Tgt_i);
alphapr_hat_i_nxt =
    alphapr_hat_i*exp(-e_a1*del_alpha_i*del_detTgt_i +
    e_pa2*(1-det_Tgt_i) + e_a2*(1-alpha_i)*det_Tgt_i);

end

```

---

---

```

function [alpha_hat_i_nxt, alphapr_hat_i_nxt, what_i_nxt] =
    AOL_LearningPhase2P(t, Tp, det_Tgt_i, what_i,
        alpha_hat_i, alphapr_hat_i, alpha_i, w_ii, del_alpha_i,
        del_w_ii, del_detTgt_i, p_mag, e_a1, e_a2, e_w1, e_w2)

if t == 1
    e_pa1 = randi([0 1])*p_mag;
    e_pa2 = p_mag - e_pa1;
    e_pw = rand*p_mag;
else
    e_pa1 = 0;
    e_pa2 = 0;
    e_pw = 0;
end

if ceil(t/Tp) == floor(t/Tp)
    what_i = 1;
    alpha_hat_i = 1;
    alphapr_hat_i = 1;
    e_pa1 = randi([0 1])*p_mag;
    e_pa2 = p_mag - e_pa1;
    e_pw = rand*p_mag;
end

alpha_hat_i_nxt =
    alpha_hat_i*exp(e_a1*del_alpha_i*del_detTgt_i +
        e_pa1*(1-det_Tgt_i) + e_a2*alpha_i*det_Tgt_i);
alphapr_hat_i_nxt =
    alphapr_hat_i*exp(-e_a1*del_alpha_i*del_detTgt_i +
        e_pa2*(1-det_Tgt_i) + e_a2*(1-alpha_i)*det_Tgt_i);
what_i_nxt = what_i*exp(e_w1*del_w_ii*del_detTgt_i +
    e_pw*(1-det_Tgt_i) + e_w2*w_ii*det_Tgt_i);

end

```

---

## 5.2 Python Code

---

```

def AOLfusionLayer1(xhatS_Tgt_i, xhat_Tgt_i_prev,
    alpha_hat_i, alphapr_hat_i, det_Tgt_i):
    if det_Tgt_i > 0.0: # tgt detected
        alpha_i = (alpha_hat_i)/(alpha_hat_i + alphapr_hat_i)
    else: # tgt undetected
        alpha_i = 0.0

    xhatI_Tgt_i = alpha_i*numpy.array(xhatS_Tgt_i) +
        (1-alpha_i)*numpy.array(xhat_Tgt_i_prev)

```

---

---

```

    return xhatI_Tgt_i, alpha_i

```

---

```

def AOLfusionLayer2(xhatI_Tgt_i, xhatI_Tgt_ij, what_ii,
    what_ij):
    wii = what_ii/(what_ii + sum(what_ij))
    j = 0
    wij = numpy.zeros(shape=(len(what_ij),1))
    for whatij in what_ij:
        wij[j][:] = whatij/(what_ii + sum(what_ij))
        j = j + 1
    xhat_Tgt_i = wii*xhatI_Tgt_i +
        sum(numpy.multiply(numpy.array(wij),numpy.array(xhatI_Tgt_ij)))
    return xhat_Tgt_i.tolist(), wii

```

---

```

def LearningPhaseAOL_C(t_count, det_Tgt_i, xhatI_Tgt_ij,
    xhatI_Tgt_i, xhatS_Tgt_i, xhat_Tgt_i_prev,
    xhat_Tgt_j_prev, what_ij, what_i, alpha_hat_i,
    alphapr_hat_i):
    Tp = 15.0
    eta_a = 0.01
    eta_w = 15.0

    whatij = numpy.append(what_ij, what_i)
    xhat_Tgt_i_prev = numpy.array([xhat_Tgt_i_prev])
    xhat_Tgt_ij_p = numpy.concatenate((xhat_Tgt_j_prev,
        xhat_Tgt_i_prev), axis=0)
    # print("Check me",xhat_Tgt_ij_p)
    # wmax = max(whatij)
    ID_star = numpy.array(whatij).argmax()
    xhat_star = xhat_Tgt_ij_p[ID_star][:]
    # print(xhat_star, ID_star)

    if math.ceil(t_count/Tp) == math.floor(t_count/Tp):
        what_i = 1.0
        alpha_hat_i = 1.0
        alphapr_hat_i = 1.0
        print("RESET DONE")
    lss_alpha = min(vec_dist(xhatS_Tgt_i,xhat_star)/15.0,1.0)
    lss_alphapr =
        min(vec_dist(xhat_Tgt_i_prev[0],xhat_star)/15.0,1.0)

    alpha_hat_i_nxt = alpha_hat_i*math.exp(-eta_a*lss_alpha)
    alphapr_hat_i_nxt =
        alphapr_hat_i*math.exp(-eta_a*lss_alphapr)
    what_i_nxt = what_i*math.exp(-eta_w*(1-det_Tgt_i))

    return alpha_hat_i_nxt, alphapr_hat_i_nxt, what_i_nxt

```

---

---

```

def LearningPhaseAOL_1P(t_count, det_Tgt_i, what_i,
    alpha_hat_i, alphapr_hat_i, del_alpha_i, del_detTgt_i,
    alpha_i):
    Tp = 15.0

    eta_w = 15.0
    e_a1 = 10.0
    e_a2 = 5.0
    p_mag = 0.1

    if math.ceil(t_count/Tp) == math.floor(t_count/Tp):
        what_i = 1.0
        alpha_hat_i = 1.0
        alphapr_hat_i = 1.0
        print("RESET DONE")

    if t_count == 0:
        e_pa1 = random.choice([0, 1])*p_mag
        e_pa2 = p_mag - e_pa1
    else:
        e_pa1 = 0
        e_pa2 = 0

    alpha_hat_i_nxt =
        alpha_hat_i*math.exp(e_a1*del_alpha_i*del_detTgt_i +
            e_pa1*(1-det_Tgt_i) + e_a2*alpha_i*det_Tgt_i)
    alphapr_hat_i_nxt =
        alphapr_hat_i*math.exp(-e_a1*del_alpha_i*del_detTgt_i
            + e_pa2*(1-det_Tgt_i) + e_a2*(1-alpha_i)*det_Tgt_i)
    what_i_nxt = what_i*math.exp(-eta_w*(1-det_Tgt_i))

    return alpha_hat_i_nxt, alphapr_hat_i_nxt, what_i_nxt

```

---

```

def LearningPhaseAOL_2P(t_count, det_Tgt_i, what_i,
    alpha_hat_i, alphapr_hat_i, del_alpha_i, del_detTgt_i,
    alpha_i, del_w_ii, w_ii):
    Tp = 15.0

    e_w1 = 8
    e_w2 = 0

    e_a1 = 10.0
    e_a2 = 5.0

    p_mag = 0.1

    if math.ceil(t_count/Tp) == math.floor(t_count/Tp):

```

```

        what_i = 1.0
        alpha_hat_i = 1.0
        alphapr_hat_i = 1.0
        e_pw = random.uniform(0,1)*p_mag

    if t_count == 0:
        e_pa1 = random.randint([0, 1])*p_mag
        e_pa2 = p_mag - e_pa1
        e_pw = random.uniform(0,1)*p_mag
    else:
        e_pa1 = 0.0
        e_pa2 = 0.0
        e_pw = 0.0

    alpha_hat_i_nxt =
        alpha_hat_i*math.exp(e_a1*del_alpha_i*del_detTgt_i +
            e_pa1*(1-det_Tgt_i) + e_a2*alpha_i*det_Tgt_i)
    alphapr_hat_i_nxt =
        alphapr_hat_i*math.exp(-e_a1*del_alpha_i*del_detTgt_i
            + e_pa2*(1-det_Tgt_i) + e_a2*(1-alpha_i)*det_Tgt_i)
    what_i_nxt = what_i*math.exp(e_w1*del_w_ii*del_detTgt_i
        + e_pw*(1-det_Tgt_i) + e_w2*w_ii*det_Tgt_i)

    return alpha_hat_i_nxt, alphapr_hat_i_nxt, what_i_nxt

```

---

```

def vec_dist(vec1,vec2):
    dist =
        math.sqrt(pow(vec1[0]-vec2[0],2)+pow(vec1[1]-vec2[1],2))
    return dist

```

---