Distributed autonomous encirclement of a wheeled robot formation with collision avoidance

Edoardo Castioni, *edoardo.castioni@studenti.unitn.it, E.N. 232250*
Pranzo Massimiliano Marco, *massimiliano.pranzo@studenti.unitn.it, E.N. 233174*

***Abstract*—The ever-increasing technologies in the field of autonomous moving robots led to the emergence of multi-agent encirclement systems against static or dynamic targets.**
This project aims to develop a distributed encirclement algorithm for a set of wheeled robots in a two-dimensional space. In particular, the agents have the task of positioning themselves on a circumference around a moving target avoiding the surrounding obstacles.
The whole algorithm is implemented in a distributed manner so each robot has to perform both individual and group operations, without a master-agent in the formation.
The reported results show that the ensemble fulfills the assigned tasks even with a heterogeneous set of vehicles.

I. Introduction

In recent years, thanks to their flexibility and computational efficiency, distributed systems have been progressively used in formation control and encirclement problems. In particular, the main applications range from surveillance to guidance or security systems.
One practical example is presented in [1] where a distributed algorithm is implemented in order to encircle chemical spills through marine vehicles.
This project presents a 2D solution to encircle a single moving target in an environment with static and dynamic obstacles. More in detail, the agents have to unfold on a circumference keeping their mutual distances equal. Each robot has to perform a self-localization operation and then, via relative measurement devices, estimate the position of its neighbours together with the target. In order to increase the precision of the estimates, the robots can exchange information within a certain range through a communication system.
Performed the estimates, the agents move accordingly to fulfil the desired task paying attention also to avoid collision, internal to the formation or with the obstacles, through a Voronoi tessellation, based on [2].

A. Paper organization

Sec. II describes how each robot has been modelled in terms of communication system, sensors used and dynamic model. Sec. III explains briefly the strategies used in the project development while Sec. IV gives a deeper explanation of the implemented algorithms. In Sec. V the most significant results are reported with detailed explanations. Finally Sec. VI states the conclusion of the project with also possible improvements.

II. Adopted models

A. Sensors and communication systems

Each robot is equipped with a GPS sensor associated with a standard deviation $\sigma_{GPS} = 3\,[\mathrm{m}]$, which provides the absolute position of the robot.
In order to perform measurements on the surrounding environment and on the other agents a stereo camera has been used with an uncertainty $\sigma_{CAM} = 0.3\,[\mathrm{m}]$. The visual measurement can take place if the object to be measured is within the sensing range of the sensor, which has been set to $R_{cam} = 12\,[\mathrm{m}]$.
Furthermore, the robots are able to exchange information with each other; this task is allowed by a communication system such as Wi-Fi or UWB. The information exchange between two robots is always bidirectional due to the same communication range $R_c = 12\,[\mathrm{m}]$.

B. Dynamics

In order to model the motion of the robot, two main dynamics have been tested: *linear* and *unicycle dynamics*.

1) Linear dynamics: The linear dynamics represents the basic model to implement the motion of a robot, indeed it allows to reach any position in the plane without any non-holonomic constraints. The equation of motion reads as:

$$\bar{x}_{R,i+1} = \bar{x}_{R,i} + \bar{u}_i + \bar{\nu}_i \tag{1}$$

where $\bar{x}_R$ is the vector containing the x and y coordinates of the robot in the absolute reference system, $\bar{u}$ is the input $(\Delta x, \Delta y)^T$ given to update the dynamics to the step *i*+1, $\bar{\nu}$ is an additive Gaussian noise that embeds both the uncertainty on the model and the odometry sensor noise, $\bar{\nu} \sim \mathcal{N}(0_2, Q)$ with Q the covariance matrix obtained with a standard deviation of $0.5\,[\mathrm{m}]$.

2) Unicycle dynamics: A more realistic dynamic such as the unicycle dynamics involves also an orientation angle θ with respect to the absolute reference frame, which leads to the following equation of motion:

$$\begin{bmatrix} \bar{x}_R \\ \theta_R \end{bmatrix}_{i+1} = \begin{bmatrix} \bar{x}_R \\ \theta_R \end{bmatrix}_{i} + \begin{bmatrix} \cos(\theta_R) & 0 \\ \sin(\theta_R) & 0 \\ 0 & 1 \end{bmatrix}_{i} \bar{u}_i + \bar{\nu}_i \tag{2}$$

where $\bar{\nu} \sim \mathcal{N}(0_3, Q)$ and $\bar{u}$ stands for $(v, \omega)^T \Delta t$ with $v = \sqrt{\dot{x}_R^2 + \dot{y}_R^2}$ and $\omega = \dot{\theta}_R$ respectively the linear and angular velocity of the robot.
This dynamics underactuated the robot since there are three

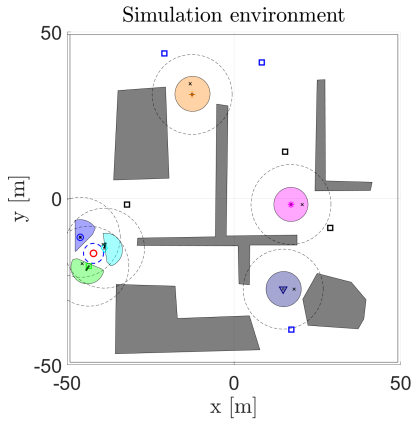


Figure 1. Example of a possible simulation environment: *robots* (small marker with Voronoi cell), *target* (red circle and formation circumference), *moving obstacles* (blue squares), *fixed obstacles* (black entities).

state variables and two control inputs and is characterized by the non-holonomic constraint to avoid sideway motion [3]:

$$\begin{bmatrix} -\sin(\theta_R) \\ \cos(\theta_R) \end{bmatrix}^T \dot{\bar{x}}_R = 0 \tag{3}$$

III. Solution

A. Simulation scenario

The simulation is described by the following entities:

- *Environment*: The simulations occur within a pre-defined square map. This is a conventional choice, the map size can be arbitrarily reshaped without any impact on the simulation results.
- *Obstacles*: The environment contains two different types of obstacles: punctual obstacles represented as particles, which can be either static (fixed in position) or dynamic (capable of free movement across the map); extended obstacles, which are entities with arbitrary shapes used to simulate walls, objects within the map, or other entities that robots typically have to avoid.
- *Robots*: The robots represent the main moving unit on the map. To simplify their modeling, a circumference encumbrance has been used to represent their volume. Each robot is identified by x and y coordinates, which refer to the center of this encumbrance, and eventually an orientation angle θ in the case of unicycle dynamics.
- *Target*: The target can be seen as a generic moving entity that has to be constantly tracked and encircled by the robots. The target has its dynamic (not treated) and moves according to a predefined trajectory that is unknown to the robots.

Each simulation time is defined by the length of the target trajectory. When this one reaches the final point the simulation ends.

B. Self and reciprocal localization

The state estimation of each robot is obtained by performing an Extended Kalman Filter. In particular, the robot computes a *prediction* of its own position by applying the dynamics equations on the previous state and subsequently, it performs an *update* combining this information with the GPS measurement.

The EKF is an optimal estimator since it minimizes the variance of the estimated states. This filter works under the assumption of zero mean white noise (Gaussian and uncorrelated) affecting the measurements, a condition always fulfilled by every measurement of the system of robots.

Once each vehicle has performed an EKF for itself, it measures the position of all the other agents (target included), via the stereo camera and projects it in the global reference frame. Since this operation is affected by both the covariance on the state and the uncertainty of the sensor, a global consensus is performed in order to improve the accuracy of the estimates. In particular, a distributed WLS with a Maximum Degree Weight algorithm (MDW) is performed. In this step the robots share the information contained in the vector Z and in the matrix C of Equation 4 for a certain number of iterations m[1].

$$Z = \begin{bmatrix} \bar{z}_1 \\ \vdots \\ \bar{z}_i \\ \vdots \\ \bar{z}_T \end{bmatrix} \quad C = \begin{bmatrix} C_{\bar{z}_1} & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & C_{\bar{z}_i} & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & C_{\bar{z}_T} \end{bmatrix} \tag{4}$$

Notice that if a robot i cannot measure another robot j, the corresponding estimate $\bar{z}_j$ is unchanged with respect to the previous time step, and the associated covariance $C_{\bar{z}_j}$ is increased. In this way, in the WLS, its information is less weighted with respect to the one of the other robots.

The fundamental assumption to ensure convergence of the global consensus is a doubly stochastic transition matrix Q:

$$\sum_{i=1}^{n} q_{ij} = \sum_{j=1}^{n} q_{ij} = 1 \quad \forall\, i, j \tag{5}$$

this enforces bidirectional communication within the agents which is always met due to the same communication radius of each robot. Once the convergence is reached, each robot has enhanced the accuracy of the measurements made before the consensus, and in addition, it has acquired estimates of robots or target that may have not been measured. In this way, every robot shares the same estimate Z which is used in the all following tasks of the simulation.

It is fundamental to underline that each robot *i* will not use the estimate z_i as input of the EKF in the next time instant, but it will directly use the output of the EKF before the consensus algorithm is performed. With this approach, the use of correlated inputs in the EFK is prevented.

C. Voronoi tessellation

Once the consensus algorithm is performed, it is necessary to define a safe area where the robot can freely move without the risk of colliding with other agents or surrounding obstacles.

[1]The algorithm rapidly converges in few steps, a limit of 50 iterations has been chosen.

This task has been carried out by implementing the Voronoi tessellation [2].
The main point of this algorithm is to partition the environment, assigning to each robot every point of the plane $\bar{p} \in \mathcal{P} \subseteq \mathbb{R}^2$ which is the closest to it. To implement this task in a distributed fashion, each robot has to perform it by itself using the estimates of the other agents and target coming from the consensus algorithm and the direct measures of the punctual obstacles.
Formally, for the robot *i*, it reads as:

$$\tilde{\mathcal{V}}_i = \{\bar{p} \in \mathcal{P} \,|\, \|\bar{p} - \bar{x}_{R,i}\| \leq \|\bar{p} - \bar{z}_j\|, \bar{z}_j \in \tilde{Z} \wedge i \neq j\} \quad (6)$$

where $\tilde{Z} = Z_{R_c} \cup P_O$, with $Z_{R_c} \subset Z$ the subset of agents seen by the robot (i.e. neighbours $\|\bar{x}_{R,i} - \bar{z}_j\| \leq R_c$) and P_O the set of punctual obstacles seen by the agent.
The Equation 6 define a region that is bounded only in the directions where an element of $\tilde{Z}$ is present. In order to make the cell a close set, it is necessary to intersect $\mathcal{V}_i$ and the closure of $R_s = R_{cam} / 2$.
The idea beyond this choice comes from the fact that, in the worst case, if two robots are exactly on the edge of their respective boundaries defined by R_{cam}, the Voronoi line that separates the two robots would be placed at exactly $R_{cam} / 2$ from each agent.
Performed that, it is necessary to take into account the extended-shape obstacles. This is done by removing the part of $\tilde{\mathcal{V}}_i$ which is covered by the obstacle. We call $\mathcal{V}_i$ the resulting cell.
Notice that the whole cell construction has to consider the uncertainties in the localization of the agents and the measures of the obstacles. Finally, the encumbrance of the robots is taken into account. This will result in a further reduction of the cell, deeply explained in Subsec. IV-D.
Once $\mathcal{V}_i$ is computed, the robot can rely on a safe starred[2] region in which it can move freely without the risk of collision. This will be used by the motion planner in order to identify the goal position for the future time step.

D. Motion control

In order to make the agents able to reach and encircle the target, it is necessary to inject an accurate input in their dynamics equations.
The design of the controller is realized in two consecutive steps:

- High-level controller: it is responsible for the trajectory planning. More specifically, it defines the $[x, y]$ Cartesian coordinates that the robot has to reach in the following time step.
- Low-level controller: it is responsible for the proper control input computation to be injected in the dynamic equation of each robot in order to reach the point planned by the high-level controller.

While the low-level controller changes according to the dynamics of the robot, the first one is valid for both linear and unicycle agents.

[2]convex w.r.t agent position

Furthermore, the high-level control has to manage all the steps of the simulation that involve: *target search*, *target reaching*, and *equidistance keeping*.

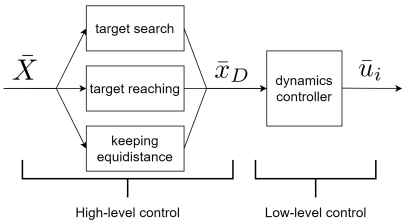


Figure 2. Block-diagram on the motion control: the input $\bar{X}$ is the estimate that a robot *i* has on itself and on the others along with the target. The high-level controller returns the position to be reached in the following time step. The low-level controller returns the input to be injected into the dynamics.

IV. Implementation details

A. Initialization

The simulation can be initialized in numerous ways. In a typical application, robots may be located at varying distances and formations relative to the target. This could involve placing some robots in proximity to the target and others in random locations across the map. It's important to note that initially, not every robot needs to have a direct line of sight to the target. Instead, it is sufficient for them to be near a robot that can see the target, and the task will still be accomplished. For those robots that can not detect either the target or the "useful" robots, they will immediately initiate the *target search*.

1) Robots properties:

- volume: the encumbrance of each agent is defined by a circle of random radius δ_i in between $[0.2, 0.4]$ [m]
- communication and sensing range: the communication radius has been set equal to the sensing range of the relative sensor, 12 [m]
- velocity: the maximum linear velocity of each agent is set in order to respect the following:

$$1 \leq \frac{v_{agent}}{v_{target}} \leq 2.5 \quad (7)$$

while the maximum angular velocity has been saturated to 1.3 $[\mathrm{rad\,s^{-1}}]$

2) Initial estimates: For each robot, the covariance matrix *P* of its state is initialized as an identity matrix.
Then, in order to simulate the initial unknowledge of the target and other robots position the associated estimates and covariances are set to high values (i.e. 10^6 $[m]$).

B. Effect of EKF on θ

A notable remark on the unicycle robots is the absence of an exteroceptive sensor for the angle estimate such as a magnetometer. This results in a non-direct update of the orientation angle θ, as opposed to what happens for the x and y coordinates thanks to the GPS.
However, during the *prediction step* of the EKF, the state covariance matrix P is computed using the Jacobian matrix J_x. This matrix is derived by differentiating the equations

of motion with respect to $\bar{x}_R$ and θ_R and allows to link the Cartesian and angular coordinates.

$$J_x = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & 1 & \cos(\theta) \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$P = J_x\, P\, J_x^T + J_Q\, Q\, J_Q^T \tag{9}$$

The main result of Equation 9 is a non-diagonal matrix P which means that the x and y coordinates are correlated to θ. Given that, in the *update-step*, once the GPS measurement occurs, it allows to improve also the estimate on the angular coordinate.

C. Consensus algorithm

The consensus algorithm can be performed by a minimum of two robots when they can communicate with each other. If a robot can not measure another agent, it will keep the most recent estimates and increment the relative covariance matrix. The covariance matrix is saturated when its norm exceeds $10^6\ [\mathrm{m}^2]$.

D. Voronoi cell creation

Referring to Equation 6 for the definition of the Voronoi cell construction, it is necessary to clarify how the encumbrance of the vehicle and the localization uncertainties are considered.
In particular, each agent *i* that performs the cell computation disposes of a vector of 2D positions $\bar{z}_j \in \tilde{Z}$ with their relative covariance matrix $C^i_{z_j}$. Notice that $\tilde{Z}$ embeds both the other agents and the punctual obstacles in the environment. Furthermore, robot *i* has its own position estimate $\bar{x}_{R,i}$ with a corresponding P_i and an encumbrance δ_i.
Calling λ_M three[3] times the maximum semiaxis of the generic covariance matrix M it is possible to define the following steps performed by robot *i* for each element of $\tilde{Z}$:

1) compensation for $C^i_{z_j}$: in order to consider the uncertainty on the position of $\bar{z}_j$, this point is moved closer to *i* by $\lambda_{C^i_{z_j}}$

$$\tilde{z}_j = \bar{z}_j + \lambda_{C^i_{z_j}} \frac{\bar{x}_{R,i} - \bar{z}_j}{\|\bar{x}_{R,i} - \bar{z}_j\|} \tag{10}$$

2) compensation for P_i: differently to the first case, the compensation for the uncertainty in the localization of agent *i* is obtained by reducing the cell by a quantity equal to λ_{P_i}[4], so two corrections are performed:

$$\tilde{z}_j = \tilde{z}_j + 2\,\lambda_{P_i} \frac{\bar{x}_{R,i} - \tilde{z}_j}{\|\bar{x}_{R,i} - \tilde{z}_j\|} \tag{11}$$

$$\tilde{R}_s = R_s - \lambda_{P_i} \tag{12}$$

3) compensation for δ_i: the reduction of the cell to accommodate the encumbrance of the agent has to be performed only if robot *i* can reach half of the distance with $\tilde{z}_j$ in a time step applying its maximum velocity (being the control input yet unknown). This implies:

$$\tilde{z}_j = \begin{cases} \tilde{z}_j & \frac{\|\bar{x}_{R,i} - \tilde{z}_j\|}{2} > v_{max_i}\,\Delta t + \delta_i \\ \tilde{z}_j + 2\,\delta_i \frac{\bar{x}_{R,i} - \tilde{z}_j}{\|\bar{x}_{R,i} - \tilde{z}_j\|} & \text{otherwise} \end{cases} \tag{13}$$

$$\tilde{R}_s = \begin{cases} \tilde{R}_s & \frac{\|\bar{x}_{R,i} - \tilde{z}_j\|}{2} > v_{max_i}\,\Delta t + \delta_i \\ \tilde{R}_s - \delta_i & \text{otherwise} \end{cases} \tag{14}$$

Notice that after these fictitious displacements, it may happen that $\tilde{z}_j$ is placed beyond the robot *i*. In this case, $\tilde{z}_j$ is forced to be on $\bar{x}_{R,i}$ (i.e. the agent *i* cannot move towards the agent *j*). Furthermore, since the encumbrance of the agent has been already compensated the safe region is referred to the set of points where the centroid of the agent can freely move.
Once these steps are performed, the Voronoi cell can be computed using the projected points $\tilde{z}_j$ and the reduced closing radius $\tilde{R}$.
Then, the cell has to be further modified according to the presence of the extended-shape obstacles. In particular the portion of $\mathcal{V}_i$ which is beyond the obstacle is removed. Notice that in this case, the compensation for the uncertainties and the encumbrance explained above is performed by an inflation of the obstacle by a factor that sums up δ_i, λ_{P_i} and $\lambda_{C_{cam}}$ (maximum semiaxis of the measurement system covariance matrix). In this last step, it is fundamental to maintain the starred property of the Voronoi cell.

E. High-level motion control

The motion planner of the algorithm is designed in order to allow the robots to properly define a goal position to be reached in the following time steps.
In particular, the high-level controller has first to define an ideal agent trajectory which is the optimal one for fulfilling the predefined tasks, and in a second moment, it has to define the actual point $\bar{x}_D$ that the robot can reach according to its Voronoi cell. The computation of $\bar{x}_D$ is done as:

$$\bar{x}_D = \frac{1}{M_{\mathcal{V}_i}} \int_{\mathcal{V}_i} \phi(\bar{q})\,\bar{q}\,dA \tag{15}$$

where $M_{\mathcal{V}_i} = \int_{\mathcal{V}_i} \phi(\bar{q})\,dA$ is the weighted area of $\mathcal{V}_i$, $\bar{q}$ is the coordinate vector of the points inside $\mathcal{V}_i$ and $\phi(\bar{q})$ is a weighting function chosen in according to the task to be fulfilled.
In particular, each agent has first to search the target until it has an estimate of its position. Performed the *target search*, the robots have to move in the target direction until they reach the encirclement circumference. This operation is called *target reaching*. Finally, they have to unfold in order to keep an angular equidistance between them: *equidistance keeping*. It has to be noticed that each controller of the formation has to identify which one of these operations a robot has to perform according to the situation that the agent is facing. Formally, the goal position to be reached is computed as:

1) Target search: This operation is needed when an agent has no information on the target. This means that the norm of

[3]This choice is done to account the 99% of probability

[4]the Voronoi boundary moves by a quantity equal to half of the displacement of the point $\tilde{z}_j$

the covariance on the target estimate, P_{target}, is larger than a given threshold (set to 10^4 $\left[\mathrm{m}^2\right]$). This may happen in two different cases:

- At the beginning of the simulation, whenever a robot is initialized far away from the target.
- During the simulation, if a robot cannot track the target due to environmental constraints (i.e. an obstacle stops the agent at a point while the target moves progressively away).

If the agent recognizes that the target is no longer visible, it moves in the direction in which the Voronoi cell has its maximum expansion. In this case $\phi(\bar{q})$ is set to be a bivariate Gaussian centered in a point of the cell which is the furthest from the robot. In this way, the efficiency of the search is optimized (i.e. if an obstacle is in front of the robot it moves in the opposite way). As a final consideration, it has to be said that if the Voronoi cell has multiple directions in which the expansion is maximum, a random one is selected and it is kept for a small amount of iteration.

2) Target reaching: In the case that the target is correctly estimated (P_{target} is bounded), the agent has to reach the desired formation circumference.
To accomplish this task, the destination point has to be computed using a weighting function $\phi(x, y)$ defined as follows:

$$\phi(x,y) = \exp\left(-\frac{R}{\xi}\sqrt{(x-T_x)^2+(y-T_y)^2} - R\right) \quad (16)$$

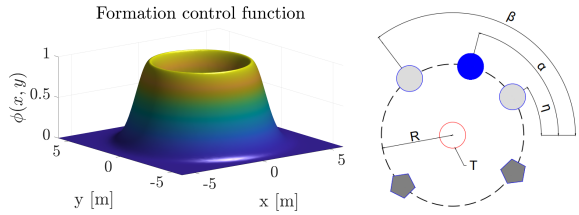
Where $\bar{T} = [T_x, T_y]$ is the target estimate, R is the radius of the formation circumference and ξ is a tuning factor that makes the exponential function thinner or larger. To better understand the mathematical discussion, Figure 3a shows a 3D plot of $\phi(x, y)$. Using this particular weighting function, each robot is attracted towards the target whenever its distance is larger than R, when instead an agent happens to be too close to the target it is pushed away towards the circumference.
In the actual implementation, a slight improvement has been applied to the computation of Equation 15, by weighting less (i.e. multiplying by 10^{-2}) the points of the cell that are beyond the robot with respect the formation circumference. Consequently, the task will be accomplished in less time.

3) Keep equidistance: The equidistance keeping is one of the tasks of the simulation. This basically means to move in order to keep the same angular distance between the robot immediately before and after with respect to a common centre, which is the target. Notice that an agent has to perform this operation only if it is sufficiently close to the formation circumference, so the following holds:

$$R - \varepsilon \leq ||\bar{x}_R - \bar{T}|| \leq R + \varepsilon \quad (17)$$

where ε is a distance tolerance generally set to be $R/2$.
In order to better understand the procedure for this part of the high-level controller, Algorithm 1 is proposed. In particular, each robot has to evaluate if there are other "neighbours" on the circle, applying the Equation 17 substituting $\bar{x}_R$ with the estimated position of the other agents. Then, in line 6 the robot *i* orders the neighbours in a clockwise sense as reported in Figure 3b, more specifically its angle is called α, the one of previous agent η and β the one of the following agent. Once the angular distances are found, a new α is computed in order to move the robot *i* towards the furthest one.
Finally, α_{new} is used to define a goal position *c* in Cartesian coordinates on the formation circumference around the target estimate. Once *c* is computed it will be used to define a bivariate Gaussian in order to find $\bar{x}_D$ via the Voronoi integral.

This algorithm allows to reach the steady state conditions quite fast. Notice that the updates of α at lines 13 and 15 ensure that between the current and the following time step, the robot *i* will be always in the middle of the two closest neighbours.



(a) (b)

Figure 3. In Figure 3a is reported the weighting function for barycentre computation in *target reaching* with $R = 3$ and $\xi = 10$.
In Figure 3b is reported a *keeping equidistance* snapshot: blue circular robot finds its neighbours in the circle to execute the Algorithm 1

Algorithm 1 Keep equidistance

1: **for** each robot R **do**
2: *Find neighbours on the circle*
3: **if** length(neighbours) < 1 **then**
4: *Maintain position on the circle*
5: **else**
6: Order the neighbours in the clockwise sense
7: $\alpha \leftarrow$ *Robot i angle on the circle*
8: $\eta \leftarrow$ *Previous robot angle on the circle*
9: $\beta \leftarrow$ *Following robot angle on the circle*
10: $\Delta^- = \alpha - \eta$
11: $\Delta^+ = \beta - \alpha$
12: **if** $\Delta^- < \Delta^+$ **then**
13: $\alpha_{new} = \alpha + \frac{1}{2}(\Delta^+ - \Delta^-)$
14: **else**
15: $\alpha_{new} = \alpha - \frac{1}{2}(\Delta^- - \Delta^+)$
16: **end if**
17: $c = [T_x, T_y] + R \cdot [cos(\alpha_{new}), sin(\alpha_{new})]$
18: **end if**
19: **end for**

F. Low-level motion control

The low-level motion control has the purpose of effectively actuating the robot, under a certain control law, once the arrival point has been computed.
The input has been designed to be a simple proportional control on the difference between the estimated position (or orientation) of the robot and the point to be reached. The proportional gain has to be set taking into account that the robot must not be able to overcome the point to be reached ($\bar{x}_D$) in a single time step.

Then, the following holds:

$$\bar{u} = k_p \left(\bar{x}_R - \bar{x}_D\right) \Delta t \leq \left(\bar{x}_R - \bar{x}_D\right) \tag{18}$$

$$k_p \leq \frac{1}{\Delta t} \ \left[\mathrm{s}^{-1}\right] \tag{19}$$

Note that the velocity input must be always compared with the maximum velocity of each robot. For the linear dynamics, it reads as:

$$\bar{x}_{R,i+1} = \bar{x}_{R,i} + \min(k_p \left(\bar{x}_{R,i} - \bar{x}_D\right), v_{MAX}) \, \Delta t \tag{20}$$

This kind of dynamics implies that the robot can move in any direction in the plane. A different approach has to be taken for unicycle robots, whose movements are constrained by their orientation angles (i.e. no side motion).
In this case, it is not straightforward to determine the trajectory to reach a certain point inside the cell, since the robot can produce a curved path that can be not entirely included in the Voronoi region. To overcome this issue, at each time step, it is necessary to check that each portion of the trajectory lies completely inside the cell, if it is not the forward velocity resulting from Equation 21 is further reduced by a proper factor.
Note that this problem does not arise using linear dynamics since the robot directly moves straight in the direction of the point to be reached and, due to the starred convexity of the Voronoi region, the trajectory will always be inside the cell.
Having stated this, a slightly modified proportional input has been designed for linear and angular velocity.
The linear velocity has been weighted by the cosine of the angular mismatch between the robot's orientation angle and the angular position α_D of the destination point (we call γ_M this difference). The purpose of this modulation is to reduce the velocity even more when the mismatch tends to $90\,[^\circ]$, constraining large-radius circular paths with the advantage of reaching the destination point in less time. Furthermore, this weighted velocity allows the robot to move backward if the point to be reached differs more than $90\,[^\circ]$ with respect to the robot's orientation.
The control on the angular velocity takes into account that the rotation has to be performed in order to cover the minimum γ_M. In this way, the robot can rotate clockwise or anticlockwise, minimizing the trajectory length.
In light of what has been said, the velocity inputs to be used in Equation 2 are:

$$v = \min(k_p \, \|\bar{x}_R - \bar{x}_D\|, v_{MAX}) \, \cos\left(\gamma_m\right) \tag{21}$$

$$\omega = \min(k_p \, |\gamma_M|, \omega_{MAX}) \cdot \begin{cases} 1 & (\alpha_D \geq \theta_R \wedge \gamma_M < \pi) \\ & (\alpha_D < \theta_R \wedge \gamma_M > \pi) \\ -1 & \text{otherwise} \end{cases} \vee \tag{22}$$

It is worth emphasizing that the control on the angular input works if all the absolute angles are wrapped to 2π (i.e. $\in [0, 2\pi]$).

V. Results

In this section, the most significant results are reported. These are shown in chronological order with respect to the simulation steps.

A. Robot localization with EKF

Figure 4 displays the error on the self-localization of a unicycle robot during a generic simulation (the following considerations apply also to linear dynamics robots, the only difference is the lack of orientation angle).
The error on the cartesian coordinates, thanks to the EKF, remains bounded in between $0.6\,[\mathrm{m}]$ even if the uncertainty on the GPS is $\sigma_{GPS} = 3\,[\mathrm{m}]$. A confirmation of what is explained in Subsec. IV-B is the error on the estimate of θ which is confined even though a magnetometer is not present.

Notice that P is initialized as an identity matrix but to make clearer the scale visualization, the time axis starts an instant after the beginning of the simulation, so P is already reduced.

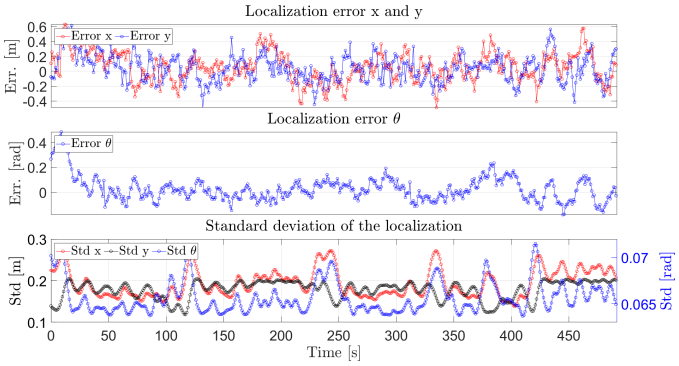


Figure 4. Example of self-localization and self-orientation.

B. Consensus algorithm

In order to inspect the efficiency of the distributed WLS, two main results are reported.

1) Effect of involved agents number: Figure 5 shows the error on the target localization as a function of the number of agents involved in the consensus. In particular, in the top image, three tests are compared, respectively with 1, 7 and 15 robots. As expected the error decreases as N increases. Notice that having $N = 1$ means that only one robot can measure the target so there is no possibility to exchange information with other neighbours. Clearly, this is the worst case, in fact, the error is larger than the other cases, and it presents some peaks around $0.35\,[\mathrm{m}]$ (the relative distance sensor has a $\sigma_{CAM} = 0.3\,[\mathrm{m}]$).
In order to better understand the amount of error decreasing, in the bottom image is reported the mean error of localization. This value spans from 0.13 to $0.03\,[\mathrm{m}]$. It is important to underline that increasing the number of involved agents in the consensus reduces significantly the error only when N reaches 7 or 8 agents. From this point the accuracy still increases, but in a more restrained manner. For this reason is not worth doing simulations with $N > 8$.

2) Covariance reduction after consensus: In Table I is reported the effect of the consensus on the localization covariance committed by robot 1. In particular the table reports σ_x and σ_y referred to itself, on other three agents and the target, before and after the consensus. As expected the values

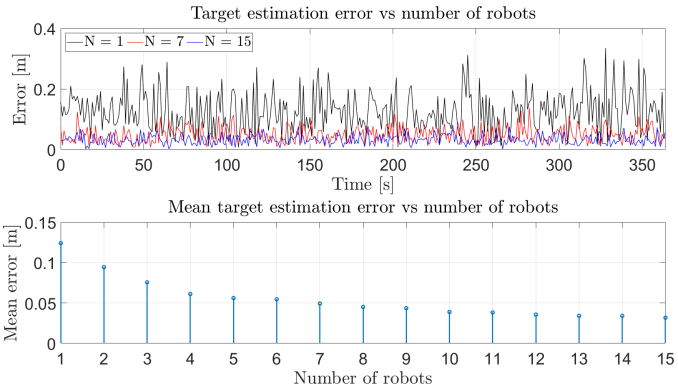


Figure 5. Target estimation error as a function of the number of robots involved in the consensus algorithm.

Table I
X AND Y STANDARD DEVIATION OF ROBOT 1 IN LOCALIZING THE SET OF AGENTS AND THE TARGET, BEFORE AND AFTER THE CONSENSUS.

	Before		After	
	σ_x	σ_y	σ_x	σ_y
R_1	0.21	0.22	0.15	0.15
R_2	0.21	0.22	0.12	0.11
R_3	0.28	0.27	0.13	0.12
R_4	0.26	0.25	0.12	0.12
T	0.22	0.23	0.13	0.12

decrease both in x and y.
Notice that the mean error of localization on the other agents decreases by around $0.08\,[\mathrm{m}]$ which confirms Figure 5. In fact with N = 1 the error is $0.13\,[\mathrm{m}]$ and with N = 4 it falls up to $0.06\,[\mathrm{m}]$, so the drop is of $0.07\,[\mathrm{m}]$.

C. Voronoi cell construction

Figure 6 represents the steps of the Voronoi construction. Each robot can measure the two obstacles and the other agent. Notice that the moving obstacle is treated as an additional agent so the Voronoi area is limited in the middle of the two points. The steps of the cell construction are identified as follows:

- *continuous line*: this is the default cell, so only the rough measures are considered. Notice that (in this case) for both robots this area intersects the fixed obstacle.
- *dashed line*: this cell is obtained after a reduction of the default area by a quantity equal to the volume of the robot.
- *colored area*: it is the final cell where also the uncertainty on the localization is considered. In order to perform this passage a coverage factor of 3 is chosen.

It is worth underlining that after all the area reduction, the cell has to remain a starred set with respect to the robot's position. As a final remark, it has to be noticed that the orange robot might appear outside its final region. This is not a problem because, by construction, the cell represents all the points in which the center of the robot can stay since the encumbrance has already been compensated.

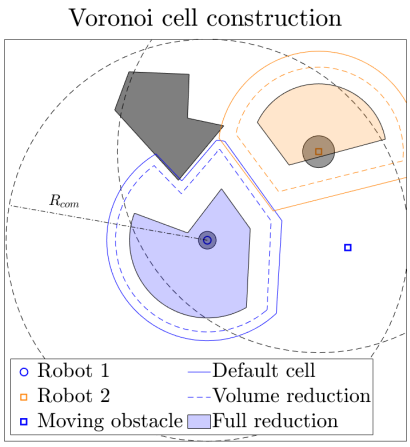
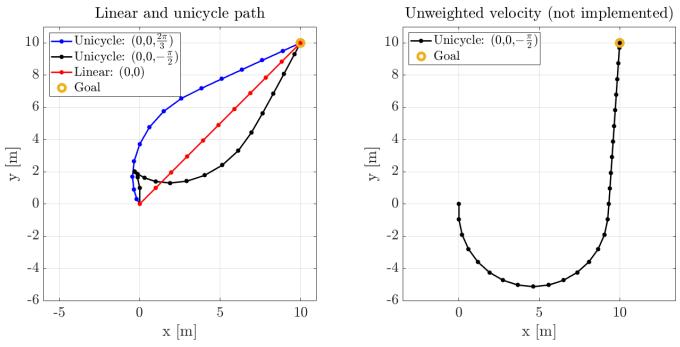


Figure 6. Steps of the Voronoi cell construction considering the encumbrance of the agents and reciprocal localization uncertainty.

D. Trajectory planning

In this section are reported the graphical results of the low-level control explained in Subsec. IV-F.
Figure 7a shows a comparison of three different trajectories obtained in a small number of time steps in reaching a goal position at $[10; 10]$ from the origin: the red one for a linear robot and the others two for a unicycle-like agent. As expected the linear dynamics is the fastest in reaching the goal position, since the robot travels along the shortest path.
The blue one corresponds to a non-linear robot with an initial angle equal to $120\,[^\circ]$. Since the relative angle between the goal position and the robot is less than $90\,[^\circ]$, the controller injects a positive forward velocity and a negative angular velocity. Conversely, when the angle between the robot and the target point is obtuse, the agent performs a "straightening" maneuver which allows it to reach the goal with the front of the robot. Furthermore in the blue and black path is possible to see the effect of the reduction of the forward velocity by the cosine of the relative angle between the agent and the goal (Equation 21). Indeed, in the initial time steps the robot advances of a small quantity. When instead it is almost aligned with the goal position, it proceeds with the maximum velocity. Figure 7b shows the resulting trajectory of a robot oriented as the black one of Figure 7a but without the effect of the cosine factor so the agent has no possibility of going backward. The path is clearly larger, even if the robot uses always the maximum velocity. It is important to notice that considering the purpose of the project it is not mandatory to reach the target with the front of the robot (i.e. with positive linear velocity). Indeed it would be better, in some cases, to proceed backward from the initial to the final point. Applying this strategy would be the same trajectory planning as the implemented one (in Subsec. IV-F) with the only difference of considering the back of the robot as the front.
However having a robot that reaches the target with the front is more realistic, so the exposed controller has been chosen.



(a) *(b)*

Figure 7. Example of trajectories produced by linear and unicycle dynamics (a). Example of trajectories of a unicycle robot with weighted and unweighted (not implemented) linear velocity. Note that the weighted velocity trajectory is significantly more constrained than the unweighted one (b).

E. Simulation evaluation

In order to evaluate the performance of the encirclement algorithm, at the end of each simulation two metrics are computed:

- $\mu_{distance}$: is the mean distance error between a robot and the closest point on the formation circumference. This metric validates mostly the performance on the *target reaching* part of the algorithm. Notice that since a robot cannot always localize the target due to initialization reasons, the $\mu_{distance}$ error is computed only in the timesteps in which the target is inside the communication radius of the robot.
- μ_{angle}: is the mean angular difference between a robot and its neighbors. Referring to Figure 3b;

$$\mu_{angle} = (\beta - \alpha) - (\alpha - \eta) \tag{23}$$

This metric allows to validate the efficiency of the *equidistance keeping* algorithm. μ_{angle} is useful to understand if Algorithm 1 can effectively be used to encircle the target while it is moving and not only in static simulations. Notice that this metric is computed only in those timesteps in which Equation 17 holds.

In Table II and Table III are reported the evaluated metrics with the relative standard deviations, for two different simulations. In the first one only linear robots are involved, in the second one the ensemble of robots is composed of three linear robots and two unicycles (R_4 and R_5). Both tables refer to a simulation in which the formation circumference radius is set to 3 [m] and the movement of the robots is not affected by the presence of obstacles in order to fairly test the efficiency of the algorithms.

As expected, the performances of a full linear set of agents are better. In fact in Table II the largest $\mu_{distance}$ is in the order of 0.1 [m] while in the second simulation, the maximum distance error is significantly larger. This is due to the presence of the non-linear agents. In fact, while the target is moving the trajectory planned by the unicycles may interfere with the one of the other vehicles so even if R_3 is a linear agent, in some cases, it is forced to deviate from the shortest path to reach the goal position. In general this is not a problem but, in order to test the worst-case scenario, the maximum forward velocity of R_3 has been set almost equal to the one of the target. So it is reasonable that once it is forced to move a little bit away from the formation circumference, it takes some time instants to recover the correct position.

The same considerations hold for μ_{angle}, with the only difference that in this case, the gap of performances is smaller. This is due to the fact that this metric is computed only when the robot is on the formation circumference, so if the agent is forced to move a little bit further from the target, it stops temporarily to compute the error.

Table II
COMPUTED METRICS FOR A SIMULATION WITH ONLY LINEAR ROBOTS. THE RESULTS ARE GIVEN IN [m] AND [rad]

	R_1	R_2	R_3	R_4	R_5
$\mu_{distance}$	-0.017	0.108	0.057	0.026	0.037
$\sigma_{distance}$	0.092	0.163	0.286	0.066	0.136
μ_{angle}	0.003	0.028	-0.028	-0.014	0.010
σ_{angle}	0.135	0.234	0.138	0.208	0.159

Table III
COMPUTED METRICS FOR A SIMULATION WITH 3 LINEAR AGENTS AND 2 UNICYCLES (R_4 AND R_5). THE RESULTS ARE GIVEN IN [m] AND [rad]

	R_1	R_2	R_3	R_4	R_5
$\mu_{distance}$	-0.008	0.041	1.769	0.714	0.554
$\sigma_{distance}$	0.134	0.163	1.397	1.493	1.146
μ_{angle}	0.008	0.026	-0.076	0.025	-0.025
σ_{angle}	0.481	0.454	0.759	0.466	0.635

VI. CONCLUSIONS

In this project, the main steps to develop a distributed encirclement algorithm have been explained. This goal required the study of robot localization (via absolute and relative sensors), the exchange of information to reach a consensus on the information and motion control.

The exposed results have shown that the task has been fulfilled correctly with different types of environments and agents. In particular, the simulations responded positively to all steps from localization to robot control without any collision.

Even if simplifying choices were made, this project may be used as a fast track for further investigations. More specifically, the accuracy on the robot localization would increase if the relative distance sensors were used to map the surrounding environment in a distributed manner (distributed SLAM).

Furthermore optimal control strategies (such as MPC) could be used to enhance the performance of the trajectory planning on the unicycle-like robots. As a final addiction, environment-adaptive formation control functions may improve the encirclement performances in the presence of obstacles.

The whole code with corresponding results, developed in Matlab [4], is avaible here: https://github.com/edocas01/DISTRIBUTE_ESTIMATION_PROJECT.

REFERENCES

[1] J. M., C. R., and Pascoal, "A. chemical spill encircling using a quadrotor and autonomous surface vehicles: A distributed cooperative approach." 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8952236/

[2] M. Boldrer, L. Palopoli, and D. Fontanelli, "A unified lloyd-based framework for multi-agent collective behaviours," *Robotics and Autonomous Systems*, vol. 156, p. 104207, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092188902200118X

[3] A. İşleyen, N. van de Wouw, and Ömür Arslan, "Feedback motion prediction for safe unicycle robot navigation," 2023. [Online]. Available: https://arxiv.org/pdf/2209.12648v3.pdf

[4] T. M. Inc., "Matlab version: 9.13.0 (r2022b)," Natick, Massachusetts, United States, 2022. [Online]. Available: https://www.mathworks.com