Distributed coverage algorithms applied to distributed target tracking - Robustness evaluation

Arnaud Klipfel, Georgia Institute of Technology
aklipfel3@gatech.edu

CONTENTS

I Related works 2
I-A Uniform densities . . . 2
I-B Time-varying densities . . . 2
I-C Target tracking . . . 3
I-D In this work . . . 3

II Distributed coverage algorithms : description 3
II-A Lloyd's algorithm . . . 3
II-B TVD Cortes . . . 4
II-C TVD-D1 . . . 4

III Distributed coverage algorithms : implementation 5
III-A Implementation method . . . 5
III-B Assumptions . . . 5
III-C Overview of the implementation . . . 6
III-D Lloyd's algorithm . . . 6
III-E TVD-Cortes . . . 6
III-F TVD-D1 . . . 6
III-G Comparison, and robustness evaluation on a tracking task . . . 7
III-G1 Protocol . . . 7
III-G2 Tracking task description . . . 7
III-G3 Analysis . . . 7

IV A fully distributed implementation : discussion 8

V Distributed estimation algorithms : discussion 8

VI Conclusion 8

Appendix A: Archive description 9

Appendix B: How to run the simulation 11

References 11

***Abstract*—Distributed pursuit-evasion games or target tracking problems can be formulated as coverage problems in time-varying environments. The time-varying density function modelling the environment can account for the uncertainty on the targets' position. Previous works have not used the network of agents to reach a consensus on the time-varying density, have considered the velocity of the target as known, and have used Lloyd's time-invariant coverage control algorithm. This work surveys the state of the art in distributed coverage control, implements the most well-established coverage control algorithms, and evaluates their robustness in simulation and hardware experiments on a single-target tracking task. Finally, the dynamic coverage algorithm TVD-D1 is found to be the best option for target tracking applications.**

***Index Terms*—Coverage control, target tracking, sensor networks, Voronoi diagrams, multi-robot systems.**

I. Related works

The coverage problem has been extensively studied in the literature. Two coverage problems were studied, the path coverage (see [1] for a survey) and the area coverage [2]–[5]. This work focuses on the area coverage in a distributed network of robots.The area coverage problem is the problem of maximally spreading out agents (robots for instance) in order to cover as much area as possible (this area is often specified), while keeping the necessary connections between agents, i.e. at least the graph of the network has to remain connected. As was discussed in [6] connections have to be kept with the Voronoi neighbors to execute a coverage control as optimally as possible. On the other hand, path coverage is a problem that can be defined with only one agent, while the area coverage problem with only one agent does not really constitute a problem anymore, it assumes that a network of agents is considered. In the path coverage problem, a sole agent or a network of agents have to sweep all the area to cover through time, as opposed to the area coverage, where agents have to see at a given time all the area to cover through their sensors.

In this section, the area coverage problem, which will be thereafter referred to as the *coverage problem* (this is how it is referred to in the distributed systems literature), will be explained from different angles and in different contexts. This section serves as a survey presenting an overview of the different sub-problems that can be encountered in the coverage problem and that were encountered in this work. Finally, the contribution of this work is explained in more depth, and the structure of the paper is outlined.

A. Uniform densities

The design of distributed controllers that could control a network of agents to cover a specified area was first studied in [4], [5], [7]. The assumption here was that the area should be covered uniformly, i.e. one location inside the coverage area is as important as any other. A distributed algorithm was proposed, *Lloyd's algorithm*, whose implementation is explained in II-A. This algorithm was initially proposed in [8], but was applied to signal processing. It is a gradient-descent or gradient-flow algorithm, which will make agents converge to a *Central Voronoi Tessellation* (CVT), where agents have minimized the coverage cost, i.e. they are optimally spread out over the coverage area. The coverage area is covered and the area to cover for each agent are minimized as much as possible. Equivalently, the quality of the coverage is maximized. Lloyd's algorithm is the basis of all distributed algorithms, which have been developed so far to tackle the coverage problem.

The area to cover is not always uniformly important to the agents. Some parts may be of more importance and would require more agents. This problem is formulated by using a locational cost or a density function. As presented in [2], [4], [5], [9], a density function specifies, which regions of the coverage area are more important. It can be seen as a probability distribution, which tells the frequency of important events over the entire area to cover. The algorithm presented in [4], [5], Lloyd's algorithm, takes into account uniform density functions. With such a control it is also possible to control agents into a specified formation through the density function, as presented in [4].

B. Time-varying densities

The coverage area may also change over time, leading to a time-varying density function (TVD). In [4] a time-varying version of Lloyd's algorithm was the first to be proposed, referred to as *TVD Cortes* in this work (after the name of one of its authors), see section II-B for the details. This algorithm makes assumptions on the density function, but not on the rate of change of the density. The inertial moment of the Voronoi region of each agent should remain constant [4]. Yet, as pointed out in [9], [10] there is no reason to assume that this holds true for every applications, and in the case, where a human user would specify these densities or the regions of interest other algorithms are then needed.

The first attempt to design a more general algorithm for the dynamic coverage problem was underwent in [11]. A controller that guarantees exponential convergence to a CVT was designed based on an energy formulation and to make agents follow the time-varying centroids of their Voronoi region. However, for the controller to work as proved the rate of change of the TVD has to be bounded. Other assumptions made on the controller were tested in simulations and experiments, but only for a Gaussian like distribution. In [12], a human-swarm-interaction system is designed allowing a user to specify at any given time a geometric pattern for the swarm (network of robots with homogeneous capabilities) to follow, by specifying interest points on a tablet-like interface, i.e. by shaping the density function. In [9], [10], [12], other distributed gradient-flow algorithms are presented to cope with the time-varying case of the coverage problem. The main idea was to design an algorithm that could from an initial CVT follow the time-varying CVT asymptotically, in contrast to [11], for details see II-C. In all the papers [9], [10], [12] only one foundational algorithm is developed, but this algorithm is not distributed and several distributed approximations of this control law are then proposed. The one-hop approximation, *TVD-D1*, i.e. when agents communicate with their direct neighbors (in this case Voronoi neighbors) is presented section

II-C.

Other more recent works have built upon the state of the art in dynamic coverage, such as [13], [14]. The main flaws in the distributed controllers designed in [4], [9], are that they either make assumptions on the density function, as in [4], or that they do not come with theoretical guarantees for convergence to a CVT, as in [9] for the distributed algorithms (only experimental explanations are given). In [13], the authors modified *TVD Cortes* [4] to provide a provable exponential convergence to a CVT. Yet, it only holds in the case, where the rate of change of the density function is bounded, which may not be the case. In [14], a constraint-based controller was built. The idea was to re-formulate the coverage problem as a constrained optimization problem, where the goal was to find the input command that minimized the coverage energy and also the input energy. The findings in [9] were used to formulate the optimization problem. The controller showed better performances than the controllers in [9]. The constraint-based controller was proven to converge to a CVT with an actuator-constraint free single integrator model, without making any assumptions on the density functions at all. Finding a closed-form controller for the dynamic coverage problem, that would give a provable convergence to a CVT with non-restrictive assumptions is still an open research problem.

C. Target tracking

Distributed Target tracking or, for a more general designation distributed pursuit-evasion games have been widely studied in the multi-robot systems literature. Some works have used Voronoi-based control, such as [15], [16], but few have used Voronoi-based coverage control.

A pursuit-evasion game can actually be formulated as a dynamic coverage problem. [4] was the first work to advance such a formulation. The TVD was chosen as a Gaussian like distribution centered around the target's position, but the matter was not investigated further. [11] extended the formulation in [4]. An algorithm for multi-target tracking was designed. Yet, the TVD did not account for the uncertainty on each target's position, and the velocity of the targets had to remain low (for the TVD to have a bounded rate of change). Lastly, the estimation of each target's position and velocity was not discussed. Each target's state was supposed to be perfectly known. [17] implemented a single target tracking algorithm, and considered the target's state estimation. The velocity was assumed to remain constant and each agent estimates the target's position using a Kalman filter without propagating the knowledge in the network, and so without exploiting the coverage formation, which is optimal for a distributed estimation algorithm. The controller used to accomplish dynamic coverage was Lloyd's algorithm, which was designed for time invariant applications.

D. In this work

Dynamic coverage algorithms provide flexibility for pursuit-evasion games. By shaping the TVD, multiple targets can be added easily, pursuers can follow the region of exploration (where the targets may be), and adopt relevant formations in the exploration zone, such that distributed estimation algorithms can be used efficiently to narrow the search and trap the targets.

Previous works have not used the state of the art in dynamic coverage, such as *TVD-D1*, and have not tested the robustness of these algorithms to noisy estimations of the targets' states. The main question of this work is then : how robust are the state of the art algorithms in coverage control on tracking tasks in a stochastic setting, when the targets' states are uncertain?

Lastly, previous works have not discussed the integration of a distributed estimation algorithm in the stochastic setting. Different solutions will be discussed.

To accomplish these objectives, section II presents the most well-established algorithms of the state of the art in coverage control. Lloyd's algorithm, *TVD Cortes*, and *TVD-D1* have been implemented and tested both in simulation and hardware. Hardware tests have been conducted with the Robotarium platform [18]. The robustness of these algorithms is evaluated and compared. The robustness tests consist of adding noise to the target's position and velocity, and to gradually increase the velocity of the target. The integration of a distributed estimation algorithm is discussed in V.

II. Distributed coverage algorithms : description

This section presents the main algorithms constituting the state of the art in distributed coverage control. To be complete this list should also include the more recent algorithms (2019) developed in [13], [14]. Nonetheless, the algorithms presented here are sufficient for the robustness study, and already give room for interesting comparisons. These algorithms are also the basis of the more recent algorithms, and it is thus necessary to study them first. The main derivations, and results are mainly taken from [4], [9], [10]. The notations are inspired by [9].

A. Lloyd's algorithm

Lloyd's algorithm is the most common algorithm used in distributed coverage applications, and was designed for time-invariant environments. The time-varying setting is considered here, but the proofs are actually identical, and lead to the same final control law.

The coverage area is considered to be a polytope $D \in \mathbb{R}^2$, $N \in \mathbb{N}^*$ agents are located in the region D, their position is denoted as $p_i \in \mathbb{R}^2$. The idea behind Lloyd's algorithm is to minimize the coverage cost, which can be written as,

$$H(p, \mathcal{T}, t) = \sum_{i=1}^{N} \int_{q \in T_i} f(\|q - p_i\|)\phi(q,t)dq \qquad (1)$$

where $p = (p_1, \ldots, p_N)^T$, $\mathcal{T} = \{T_1, \ldots, T_N\}$ is a tessellation of the coverage area, i.e. each agent is given a sub-area or region to cover with their sensors, the region of agent i is T_i. The sensor cost is denoted as the function $f : \mathbb{R}_+ \to \mathbb{R}_+$, and is strictly increasing. In this work, as it is often considered in the literature, $f : x \mapsto x^2$. The function $\phi : D \times \mathbb{R}_+ \to \mathbb{R}_+$ is the TVD. In some cases the density denotes a probability distribution and is then normalized, $\phi : D \times \mathbb{R}_+ \to [0; 1]$ [17].

To minimize the coverage cost in (1) one needs to minimize first with respect to the tessellation $\mathcal{T}$. For one configuration of the network $p(t)$ at a given time t, the coverage cost is minimized if and only if the tessellation $\mathcal{T}$ is a Voronoi tessellation [4]. A Voronoi tessellation will thereafter be denoted as $\mathcal{V}$, and the coverage cost in (1) can be denoted as $H_{\mathcal{V}}$, now that the tessellation in not a variable to minimize. The question is then : how to find the optimal configuration p^* for the agents?

To find the configuration p^* that minimizes the coverage cost, a gradient-descent algorithm is designed. From [4], one can show that,

$$\frac{\partial H_{\mathcal{V}}}{\partial p_i}(p,t) = 2m_i(p,t)((p_i(t) - c_i(p,t))^T \tag{2}$$

where the generalized mass of the Voronoi region V_i is given as,

$$m_i(p,t) = \int_{V_i(p)} \phi(q,t)dq \tag{3}$$

And the center of mass or centroid of V_i is given as,

$$c_i(p,t) = \frac{\int_{V_i(p)} q\phi(q,t)dq}{m_i(p,t)} \tag{4}$$

Considering that the controlled system can be transformed as a single-integrator model, the control is,

$$u_i = \dot{p}_i = -k\frac{\partial H_{\mathcal{V}}}{\partial p_i}(p,t)^T \tag{5}$$

where $k > 0$ is a positive gain.

Finally, the coefficient $2m_i(p,t)$ can be removed, since in a gradient descent algorithm the only thing that matters is that the updates are made in the opposite direction of the gradient. Lloyd's algorithm is given as,

$$\boxed{u_i = -k(p_i(t) - c_i(p,t))} \tag{Lloyd}$$

Lloyd's algorithm in the time-invariant setting is proven to have a local asymptotic stability, and makes the agent converge to a CVT [5]. Local since depending on the initial positions of the agents, the CVT may be different, there is no unique CVT for a given distribution and a coverage region. For given positions of agents p, the CVT is defined as the Voronoi tessellation where all agents are located at the centroid of their respective region.

As discussed in [9], in the case of a time-varying density, the rate of change of the cost is not proven to decrease over time, i.e. condition (6) does not hold. That is what the algorithms presented in section II-B and in section II-C try to address.

B. TVD Cortes

In [4], a distributed coverage algorithm was designed in an attempt to handle TVDs. The main idea was to try to design a controller that could enforce,

$$\frac{dH_{\mathcal{V}}}{dt}(p,t) < 0, \forall p,t \tag{6}$$

By making assumptions on the density function ϕ, the controller can guarantee that. In [4], it was shown that using the parallel axis theorem, the coverage cost could be divided in two parts,

$$H_{\mathcal{V}}(p,t) = \sum_{i=1}^{N} J_{i,c_i(p,t)} + \sum_{i=1}^{N} m_i(p,t) \left\| p_i(p,t) - c_i(p,t) \right\|^2 \tag{7}$$

where $H_{\mathcal{V}_1} = \sum_{i=1}^{N} J_{i,c_i}$, and $H_{\mathcal{V}_2} = \sum_{i=1}^{N} m_i \left\| p_i - c_i \right\|^2$.

J_{i,c_i} is the polar moment of inertia of the Voronoi region of agent i. It quantifies the distribution of the mass in V_i with respect to the axis (z, c_i), i.e. axis z passing through the centroid of V_i. $H_{\mathcal{V}_1}$ only accounts for the contribution of the TVD, and not of the agents as in $H_{\mathcal{V}_2}$.

It was then supposed that the TVD could verify the property,

$$\frac{dH_{\mathcal{V}_1}}{dt} = 0 \tag{8}$$

In that case, by choosing,

$$\dot{p}_i = c_{i,t}(p,t) - \left(k + \frac{m_{i,t}(p,t)}{m_i(p,t)}\right)(p_i(p,t) - c_i(p,t)) \tag{9}$$

where $f_t(x) = \dfrac{\partial f}{\partial t}(x)$ as a notation, it is then possible to have,

$$\frac{dH_{\mathcal{V}}}{dt} = -kH_{\mathcal{V}_2} \tag{10}$$

Supposing that the system can be controlled as a single-integrator, the control law is given by,

$$\boxed{u_i = c_{i,t} - \left(k + \frac{m_{i,t}}{m_i}\right)(p_i - c_i)} \tag{TVD-Cortes}$$

where,

$$m_{i,t}(p,t) = \int_{V_i(p)} \phi_t(q,t)dq \tag{11}$$

$$c_{i,t}(p,t) = \frac{1}{m_i(p,t)}\left(\int_{V_i(p)} q\phi_t(q,t)dq - m_{i,t}(p,t)c_i(p,t)\right) \tag{12}$$

TVD Cortes is a controller that ensures local asymptotic convergence to a CVT in the case where the density verifies condition (8). Yet, as pointed out in [9] it is rarely the case in time-varying environments, and it is even unclear how to design such TVD [13].

C. TVD-D1

In [9], [10], an algorithm that did not make any assumptions on the TVD was designed. As described in the original work [9], the main idea was to track the CVT. Once the agents have reached the CVT, i.e. $p(t_0) = c(t_0, p_0)$, where $c = (c_1, \ldots, c_N)^T$, the controller was designed such that $\dot{p}(t) = \dot{c}(t,p)$. It is then possible to write,

$$\dot{p} = \left(I_{2N} - \frac{\partial c}{\partial p}\right)^{-1}\frac{\partial c}{\partial t} \tag{13}$$

where I_k is the identity matrix of size $k \times k$. In order to ensure that the agents converge to the CVT in the first place a proportional term was added, as,

$$\dot{p} = \left(I_{2N} - \frac{\partial c}{\partial p}\right)^{-1} \left(-k(p-c) + \frac{\partial c}{\partial t}\right) \tag{14}$$

Later in [2], another proof relying on an energy formulation was used. The same control law was derived. If the agents can be controlled as single integrators the control law is given by,

$$u = \left(I_{2N} - \frac{\partial c}{\partial p}\right)^{-1} \left(-k(p-c) + \frac{\partial c}{\partial t}\right) \tag{15}$$

where $u = (u_1, \dots, u_N)^T$.

As pointed out in [2], [9] the main issue is that the control law presented in (15) is centralized. The inversion of $I_{2N} - \frac{\partial c}{\partial p}$ does not make a distributed control possible. One way to decentralize the control is by considering Neumann series. For a matrix $A \in \mathbb{R}^{m\times m}, m \in \mathbb{N}^*$,

$$(I_m - A)^{-1} = \sum_{k=0}^{+\infty} A^k \tag{16}$$

as long as $|\lambda_{max}| < 1$, i.e. the maximum eigenvalue of A has to be strictly bounded by 1 in absolute values, otherwise the Neumann series in not defined, since it diverges.

It is then possible to truncate the series at a chosen order. To choose the order, one other observation on the structure of $\frac{\partial c}{\partial p}$ was made in [9]. $\frac{\partial c_i}{\partial p_j}$ accounts for the variation of the center of mass of V_i due to agent j, located at p_j. V_i is only defined by its Voronoi neighbors $\mathcal{V}_i$, and thus the centroid is only defined by $\mathcal{V}_i$. As a results if $j \notin \mathcal{V}_i$,

$$\frac{\partial c_i}{\partial p_j} = 0_{2\times 2} \tag{17}$$

The matrix $\frac{\partial c}{\partial p}$ has actually the same block sparsity as the adjacency matrix of the Delaunay triangulation corresponding to the Voronoi tessellation $\mathcal{V}$. The distributed one-hop controller, i.e. using only the direct Voronoi neighbors of each agent, is then given by a truncation at the first order, as,

$$\boxed{u = \left(I_{2N} + \frac{\partial c}{\partial p}\right) \left(-k(p-c) + \frac{\partial c}{\partial t}\right)} \tag{TVD-D1}$$

The controller TVD-D1 is an approximation of the controller presented in (15). The controller in (15) ensures local exponential convergence to a CVT [2], but not its one-hop approximation. Its one-hop approximation will be as close to the centralized control as $|\lambda_{max}| \ll 1$. TVD-D1 is a distributed algorithm, and does not make any assumptions on the TVD. However, it does not not come with theoretical guarantees at all. As showed experimentally in [9] and in this work, see section III-G, the controller shows close performance to the centralized controller. One obstacle in the implementation of the controller TVD-D1 is the computation of the term $\frac{\partial c}{\partial p}$, which is explained in section III-F.

III. Distributed coverage algorithms : implementation

This section presents the implementation of the algorithms in section II. The reader should refer to the extensive in-code documentation that is provided for more details, and programming specificities.

A. Implementation method

The implementation of the algorithms described in II has been done in *Matlab*. Elements given along with this work (videos, code, ...) are presented in appendix A, and instructions to launch a simulation are provided in appendix B. The simulations coming along with this work use the package[1] implemented for the support of the *Robotarium*, which is a remotely accessible swarm platform that allows, when using the Robotarium simulator, to test algorithms in simulation and then deploy on the hardware platforms of the Robotarium [18], [19]. The simulations are then more close to the real hardware than pure mathematical simulations, as were done in [4], [5] for instance. Thus, it is important to take the dynamics of the robots into account for the simulation.

The control laws have all been normalized to avoid any actuator saturation, using the function,

$$\begin{aligned} g :&\mathbb{R}^2 \to \mathbb{R}^2 \\ &u \mapsto \frac{u}{1 + \|u\|} \end{aligned} \tag{18}$$

as was proposed in [4], s.t. $\|g(u)\| \in [0; 1]$.

The agents were controlled as single-integrators. The dynamics of the robot is the one of a unicycle [4], [14], [19]. A Near-Identity-Diffeomorphism provided in the Robotarium simulator is used to map the computed control law from the single-integrator space to the unicycle space. Control laws in the unicycle space have also been investigated, see [10].

Obstacle avoidance, i.e. avoidance of other agents and of boundaries (the agents cannot leave the coverage area D), are done using barrier certificates already implemented in the Robotarium simulator [19]. The barrier certificates impact the control laws. In some cases a CVT was not reachable because of the barrier certificates (agents are too close). Yet, the barrier certificates are necessary.

Since the algorithms are designed to be deployed on hardware a special attention has been given to their spacial and temporal complexity.

B. Assumptions

Although the control laws implemented in this work are distributed in nature, they require information to be computed. The implementation presented here computes this information in a centralized way, i.e. Voronoi cells are computed in a centralized way, and in the code, the different agents a part of one class the controller class (`LloydController`, `TVDCortes`, or `TVDD1`). As explained in [4] since the control laws involve integrals, accuracy is essential. A fully distributed implementation would be less accurate than a centralized

[1] https://github.com/robotarium/robotarium-matlab-simulator

implementation for the information necessary to the control laws. A centralized computation for the necessary information has been chosen since it does not impact the comparison of the different algorithms with respect to their convergence to a CVT, thus it is sufficient for the robustness evaluation proposed in this work. In addition, no parallel implementation is actually possible in the Robotarium simulator. On real robots, algorithms would be launched in parallel on the different agents. As a consequence, having a distributed implementation for the voronoi cells would take more time, than it would take on real agents. Nonetheless, ideas are given in IV to design a fully-distributed implementation.

The implementation presented here tackles a single-target tracking scenario, and does not contain a distributed estimation mechanism. The integration of a distributed estimation algorithm is discussed in V.

Agents are also supposed to be able to localize themselves in a global reference frame $\mathcal{R}_0$. All the quantities, which need a frame, have been projected to $\mathcal{R}_0$. For instance, each agent knows its Voronoi cell in the frame $\mathcal{R}_0$.

The connection between agents is supposed to be represented by a complete graph. It follows that agents are supposed to be able to communicate with all other agents, and as a result with their Voronoi neighbors. Connection issues and dealing with Voronoi neighbors, which are out of sensor range, are not in the scope of this implementation.

C. Overview of the implementation

The code comes with five classes :

- `VoronoiRegion.m` : contains some information relating to the each Voronoi region and contains integration schemes for the computation of the center of mass of a Voronoi region.
- `Target.m` : class of the target that is tracked by the agents. The target is not physical, i.e. only represented through a density function that the agents try to track.
- `LloydController`: implements the Lloyd controller.
- `TVDCortes`: implements the TVD-Cortes controller.
- `TVDD1`: implements the TVD-D1 controller.

D. Lloyd's algorithm

The expression of the Lloyd controller requires each agent to be able to compute the centroid c_i of its own Voronoi region V_i.

In order to do so in a centralized way, the function Matlab `voronoin` was used. The only issue is that the Voronoi regions need to be bounded, and the basic Matlab implementation does not bound the regions. To obtain bounded Voronoi regions in D, an *artificial problem* was defined, i.e. artificial agents were added for each agent i in order to add to the boundaries of the Voronoi region V_i the boundaries of the polytope D. The implementation is detailed in the method `control` of the controller classes.

Once the vertices of each Voronoi regions were known in $\mathcal{R}_0$. The Voronoi regions had to be integrated to compute the centroids. A bounding box of each Voronoi cell is integrated instead of the entire coverage region D, and Riemann integrals are used to compute the centroids [20]. Riemann integrals have been used for all the integral quantities in this implementation. A simple Riemann integral can be written as a summation over the domain, as

$$\int_V f(q)dq \approx \sum_{k=1}^{n} f(q_k)\Delta q \tag{19}$$

where $n \in \mathbb{N}/\{0\}$ is the number of samples in the region V. Experimentally, $\Delta q = 0.1\ m$ has been found as the optimal trade-off between the control accuracy and the computation speed. The integration can be found in the `VoronoiRegion` class, as the `centroid` method.

E. TVD-Cortes

The controller TVD-Cortes is quite similar to the Lloyd controller. The only difference is that more integrals are needed in order to come-up with the final control law, and these quantities take into account differential quantities with respect to time. The integration scheme can be found in the method `region_integration_TVD_cortes` of the `VoronoiRegion` class.

F. TVD-D1

In TVD-D1 the most computationally cumbersome term to numerically approximate was $\frac{\partial c}{\partial p}$. From [10] the expressions of the coefficients $\frac{\partial c_i}{\partial p_j} \in \mathbb{R}^{2\times 2}$, can be derived in algebraic form, if $i = j$,

$$\begin{aligned}\frac{\partial c_i}{\partial p_i} &= \textstyle\sum_{j\in\mathcal{V}_i} \left(\int_{\partial V_{i,j}} \phi(q,t) q \frac{(q-p_i)^T}{\|p_j-p_i\|} dq\right)/m_i \\ &- \left(\int_{V_i(P)} \phi(q,t) q dq\right)\left(\int_{\partial V_{i,j}} \phi(q,t)\frac{(q-p_i)^T}{\|p_j-p_i\|} dq\right)/m_i^2\end{aligned} \tag{20}$$

and if $i \neq j$,

$$\begin{aligned}\frac{\partial c_i}{\partial p_j} &= \left(\int_{\partial V_{i,j}} \phi(q,t) q \frac{(p_j-q)^T}{\|p_j-p_i\|} dq\right)/m_i \\ &- \left(\int_{V_i(P)} \phi(q,t) q dq\right)\left(\int_{\partial V_{i,j}} \phi(q,t)\frac{(p_j-q)^T}{\|p_j-p_i\|} dq\right)/m_i^2\end{aligned} \tag{21}$$

Algebraic forms were used for faster computations.

In order to compute a Riemann integral of $\frac{\partial c}{\partial p}$ a book-keeping mechanism was needed to store the Voronoi neighbors for each agent. The Voronoi neighbors were detected by checking if two common vertices existed between two Voronoi regions. The code is presented in the method `VoronoiNeighbors` of the `TVDD1` class.

Once the Voronoi neighbors were known for each agent, the line integrals at the borders $\partial V_{i,j}$ between V_i and V_j had to be computed. To do so the diffeomorphism,

$$\begin{aligned} h :&[0; \|v_i - v_j\|] \to \mathbb{R}^2 \\ &s \mapsto v_j + s\frac{v_i - v_j}{\|v_i - v_j\|}\end{aligned} \tag{22}$$

was used. By doing so the segment was parameterized with a real s and the line integrals were computed, changing in (20, 21): $dq = ds$ and $q = h(s)$. The code can be found in the `compute_dcdp` in the `TVDD1` class. Experimentally, an optimal integration step was chosen to be : $ds = 0.05\ m$.

G. Comparison, and robustness evaluation on a tracking task

This section presents the results of the robustness evaluation.

1) Protocol: The robustness evaluation on the tracking task described in III-G2 was evaluated in different ways. The robustness of the different algorithms presented in II was tested for an increasing target's velocity (see Figure 1 for selected results), a wrong estimate of the target's velocity (see Figure 2 for selected results), a gradually increasing error on the target's position simulating the absence of a filtering mechanism (see Figure 3 for a selected experiment), and finally the algorithms were compared on hardware (see Figure 4 for a selected experiment).

The protocol was to first test for different initial conditions and configurations (number of agents, initial poses, target's state, noise, and control gains), they were generated randomly to maximize the variety of the data collected and stayed identical for the different algorithms to compare. Representative results of all the tests were then selected, and the same initial configurations and conditions (number of agents, initial poses, and control gains) were kept across all the tests in order to maximize the comparability of the results. In the results presented here $k = 0.9$ and $N = 4$, the initial poses are provided along with this work in the `initialPoses.mat` file. In the Figure 1, 2, 3, and 4 the time t is the simulation time and represents real $0.033\ s$.

In order to quantify the robustness of the different algorithms, the instantaneous coverage cost and the total simulation coverage cost were used as metrics [10]. the instantaneous coverage cost is given by, if only the time dependence is made explicit since $p = p(t)$,

$$H_{\mathcal{V}}(t) = \sum_{i=1}^{N} \int_{q \in V_i(t)} \|q - p_i(t)\|^2 \phi(q,t) dq \tag{23}$$

and the total coverage cost is given in continuous form as,

$$H_{tot}(T) = \int_{t=0}^{T} H_{\mathcal{V}}(t) dt \tag{24}$$

For more details on the different testing conditions, and protocol, the testing framework is given in the files : `coverage_simulation_main.m` for a manual test of one algorithm or `coverage_simulation_main_auto.m`, `auto_test.m` for an automated test of the different algorithms.

2) Tracking task description: The tracking task involved a single target. The target was not physical, and was represented by a Gaussian like density function. The density function of the coverage problem is then given as,

$$\phi(q,t) = a e^{-(\gamma(x-x_T)^2 + \gamma(y-y_T)^2)} + b \tag{25}$$

where, $q = (x,y)^T$, x_T, y_T are the mean of the target's position, $\gamma = 1/2\sigma^2$ and σ is the variance on the target's position (identical along both axis). $a > 0$ is a coefficient that can be set to increase the attraction of the tracking task, and $b > 0$ to increase the attraction of the uniform coverage [11]. In the results presented here $a = 1$ and $b = 10^{-5}$.

In the results presented in this work, $\gamma = 10$ if it was not changed as in Figure 3.

Since no estimation algorithm was implemented, for the tests x_T, y_T were taken as the actual target's position, and the uncertainty on the position was added through the variance only. If an estimation algorithm is added the TVD could be chosen as the distribution function on the target's position. b should still be > 0.

The target evolves following a trajectory (position and velocity) given by the Bernoulli Lemniscate 2D curve, expressed as,

$$\begin{pmatrix} x_T \\ y_T \end{pmatrix} = \begin{pmatrix} \frac{\cos(t)}{1+\sin(t)^2} \\ \sin(t) x_T \end{pmatrix} \tag{26}$$

For more details please refer to the `Target.m` class. This trajectory was ideal since it covered most of the coverage area, it describes an ∞.

3) Analysis: For the increasing target's velocity test presented in Figure 1, one can observe that in case 1a (static target), Lloyd and TVD Cortes performances are indistinguishable, they are performing slightly better than TVD-D1 for the first 800 iterations and TVD-D1 performs better after 800 iterations. It is consistent with the theory presented in II, since with a uniform density function Lloyd and TVD Cortes should exhibit local asymptotic stability. TVD-D1 has not this theoretical guarantee, but the results shown here are consistent with the one in [10], TVD-D1 is actually close to its centralized version, since it has an asymptotic stability. In all the remaining cases, where the velocity of the target is gradually increased from case 1b to 1d, the observations are the same. H_{tot} increases when the target velocity increases, with a significant increase in case 1d, which shows that the faster the target goes the harder for the robots the tracking of the exploration zone or the density function is. Increasing the velocity of the target further would produce misleading results, since the actuators would saturate.TVD-D1 performs better than Lloyd and TVD Cortes in all the cases, and TVD Cortes performs better than Lloyd in all the cases, when looking at H_{tot}. This can by explained be the fact that Lloyd was designed for time-invariant applications. TVD Cortes does not exhibit an asymptotic convergence, which is consistent with the fact that the time-varying tracking density violates the assumptions at the core of TVD Cortes. The variations in $H(t)$ of TVD Cortes and Lloyd illustrate the violated theoretical guarantees. In all cases, TVD-D1 manages to converge almost asymptotically. In case 1d $H(t)$ stabilizes slightly after 1000 iterations. Overall, TVD-D1 was far more robust to increase in the target's velocity. Although there is no theoretical guarantees for asymptotic convergence for TVD-D1, TVD-D1 exhibits an almost asymptotic behavior.

When the algorithms use the same wrong estimate of the target's velocity, as presented Figure 2, it can be noticed that Lloyd is not affected, which is logical since Lloyd does not use the estimate of the target's velocity in its control law. However, TVD Cortes and TVD-D1 are affected, since they use that estimate in their control. This can be observed in cases 2a and 2b. Although both TVD-D1 and TVD Cortes exhibit oscillations, TVD-D1 oscillates less than TVD Cortes, and even when the wrong estimate of the velocity is used the

total coverage cost is the lowest for TVD-D1. This observation can be explained by the difference in the control laws of TVD Cortes and TVD-D1. TVD Cortes only uses local information, i.e. of its own Voronoi cell, in contrast to TVD-D1, which uses information of its direct Voronoi neighbors. When $\partial c/\partial t$ is not accurate, $\partial c/\partial p$ gives to each agent the knowledge of how much the density is located in their Voronoi neighbors' regions, which reduces the error induced by the wrong $\partial c/\partial t$. Overall, TVD-D1 is more robust to wrong estimations and noise of the target's velocity than TVD Cortes, and still performs better than Lloyd.

In the case 3, where the zone of exploration expands, since the uncertainty of the target's location increases, TVD-D1 tracks the exploration zone better than Lloyd and TVD Cortes as both $H(t)$ and H_{tot} corroborate it. TVD Cortes is worse than Lloyd, which is due to the fact that the density function violates the assumptions behind TVD Cortes. Overall, TVD-D1 is more robust to noise on the target's position.

The hardware experiment in Figure 4 is consistent with the corresponding simulation results. TVD-D1 performs better than TVD Cortes and Lloyd, and TVD Cortes is slightly better than Lloyd. The difference observed in the data between simulation and hardware, when comparing 1d and 4 may be due to the noise in the actuators, and the difference in initial poses of the agents. Even though the same initial pose was given for simulation and hardware experiments, hardware experiments have to take into account actuator and sensor noise. It can be noticed that all algorithms performed better in this hardware case. TVD-D1 shows an almost asymptotic behavior.

All in all, the different robustness evaluations are sufficient evidence to prove that experimentally, TVD-D1 is far more robust for tracking tasks than Lloyd and TVD Cortes. It can adapt to the velocity of the target, track the exploration zone with more accuracy when the target's position is more uncertain, and compensate for noise and wrong estimates of the target's velocity. Nonetheless, it should be reminded that Lloyd was designed for time-invariant tasks, and that the tracking density violates the assumptions behind TVD Cortes. The results found are actually quite logical. Overall, what can be surprising is that Lloyd performs better than TVD Cortes on tracking tasks, when considering all the robustness tests conducted here. Among the state of the art in coverage control presented here, TVD-D1 was shown experimentally to be the most robust algorithm for target tracking.

IV. A fully distributed implementation : discussion

The content presented here was not implemented, and is given as possible horizons for future works.

As explained in III-B, the Voronoi cells have been computed in a centralized way. The concepts and implementation in this work could be adapted to a decentralized setting, where the communication topology of the network would not be a complete graph, as for instance a proximity graph. Yet, it may not be as optimal as the algorithms presented in [21] to compute the Voronoi cells. Once we have a distributed algorithm to compute the Voronoi cells, another question remains : how does each agent know that its Voronoi cell is complete, i.e. that it sees all of its Voronoi neighbors? As discussed in [6], loosing Voronoi neighbors can have huge impacts on the coverage performance. It is then crucial in a distributed setting to make sure that the Voronoi neighbors are visible to each agent.

In [5], agents adapted their sensor range to see all their Voronoi neighbors. It was showed that it was possible to know whether or not a Voronoi cell was complete in a distributed way provided that the agents could modulate their sensor range, which is not always the case. Other techniques have been developed in [21]. The reader is referred to [21] for a more extensive survey.

In this work another option was considered. Agents could run a consensus algorithm in order to have a complete graph as connection topology. This would require a centralized clock, but once this first phase is validated, agents could switch to a distributed coverage algorithm, and try to conserve the connections with their Voronoi neighbors only. During a coverage control, Voronoi neighbors may vary, but they usually stabilize at some point depending on the density function. This idea would need to be investigated further.

V. Distributed estimation algorithms : discussion

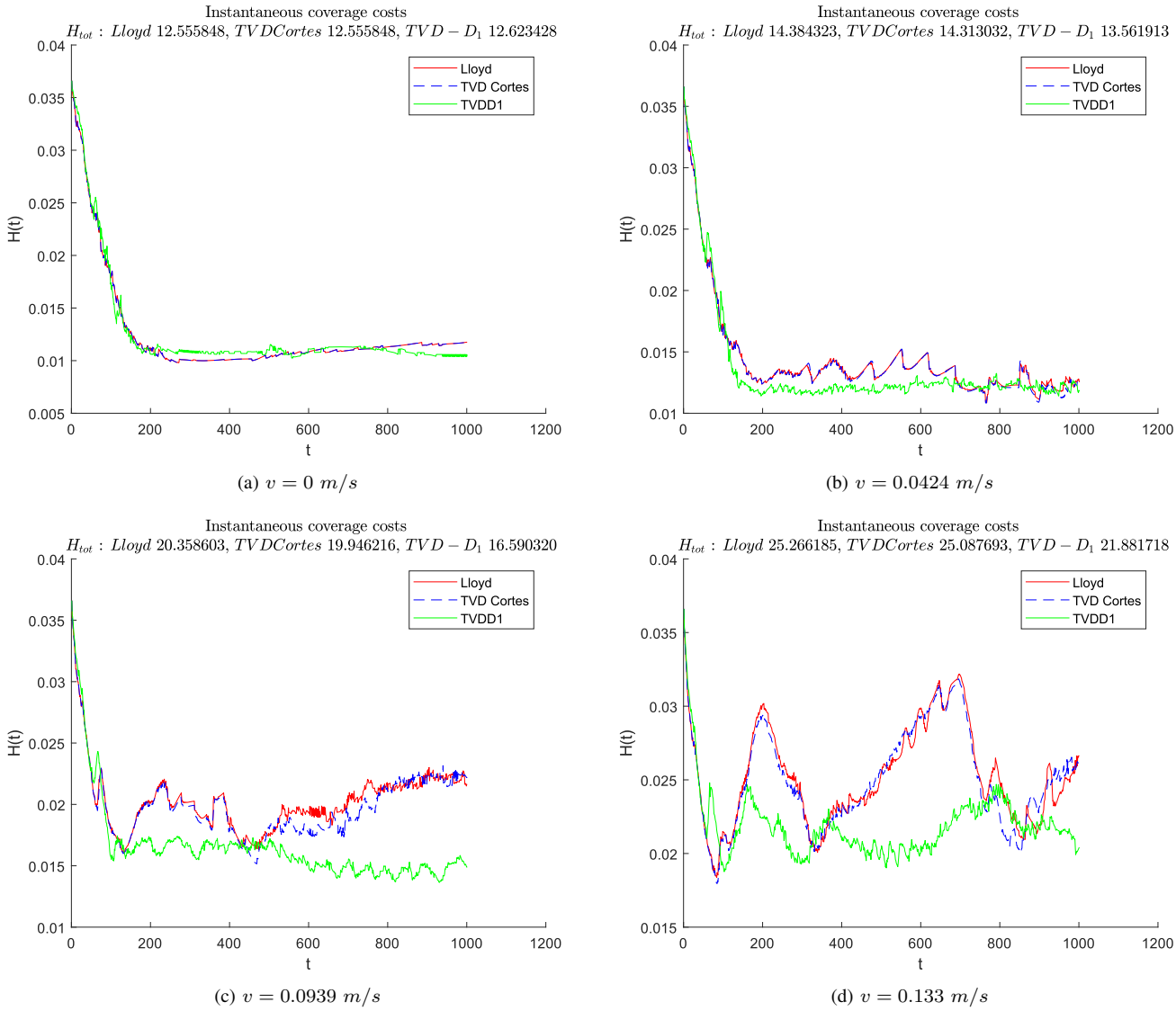
The content presented here was not implemented, and is given as possible horizons for future works.

When the target's state is unknown it is necessary to have an estimation algorithm. Since the implementation has to be distributed the estimation algorithm has to be distributed. The estimation will then benefit from the coverage formation. One option that was considered was a Kalman Consensus Filter. Other options are presented in [22]. For estimation algorithms applied directly to the learning of unknown density functions, [23], [24] are interesting works.

VI. Conclusion

In this work, existing coverage control algorithms have been surveyed, Lloyd's algorithm, TVD Cortes and TVD-D1 have been implemented, and their robustness has been evaluated on a single-target tracking task. Robustness was tested in different ways. The velocity of the target was gradually increased, wrong estimates of the target's velocity were given to the agents, and noise was added to the target's position. No theoretical guarantees are given on the performances of the algorithms in the target tracking setting considered here. Lloyd's algorithm is a time-invariant algorithm, the TVD violates the assumptions of TVD Cortes, and TVD-D1 is an approximation of a centralized algorithm. Yet, it was shown experimentally that, overall, the time-invariant Lloyd's algorithm performed better than the time-variant TVD Cortes algorithm. Lastly, the most robust option for target tracking was experimentally shown to be TVD-D1.

Now that a distributed coverage algorithm was chosen, to design a fully distributed target tracking system, a distributed algorithm would need to be integrated, and the Voronoi cells would need to be computed in a distributed way.



(a) $v = 0\ m/s$ (b) $v = 0.0424\ m/s$

(c) $v = 0.0939\ m/s$ (d) $v = 0.133\ m/s$

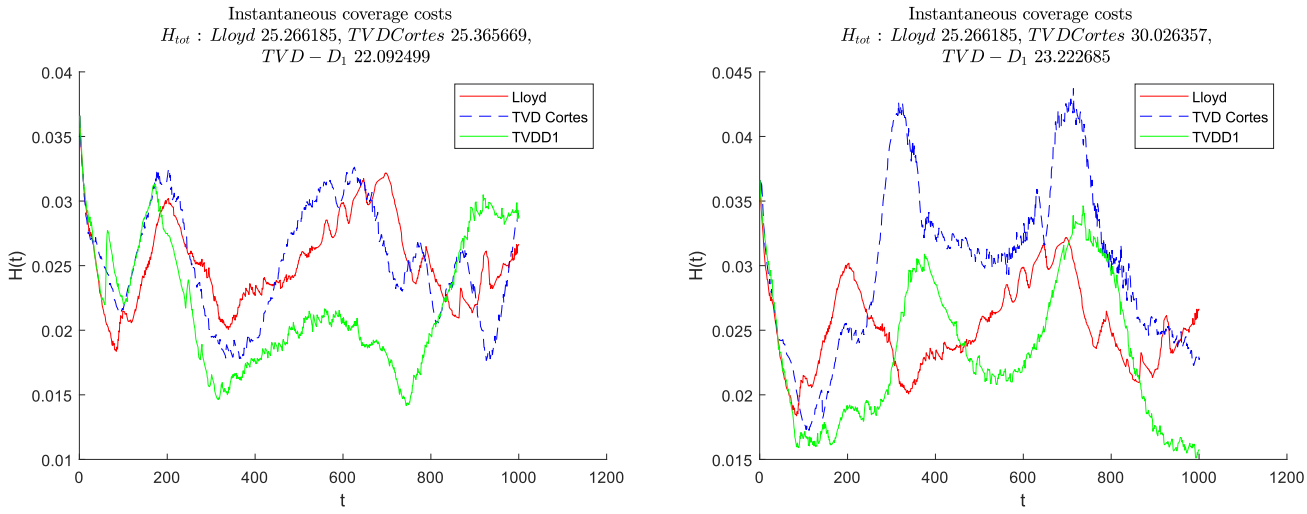
Fig. 1: Robustness comparison of the different coverage algorithms for an increasing velocity v of the target and identical initial poses (orientation, and position). In the different experiments the agents always started at the same initial poses (they were scattered in the coverage area). The controller gains were all set to $k = 0.9$ (found to be working optimally in experiment), $N = 4$ agents. The velocity of the target v was gradually increased and was kept under the maximal linear velocity of the robots, $V_{max} = 0.2\ m/s$.

APPENDIX A
ARCHIVE DESCRIPTION

The archive given along with this work contains:

- A folder `code`: this folder contains the files necessary to run the Matlab simulation. The Matlab files are :
  - `auto_test.m` : a file that runs three successive simulations, one for each controller and plots the results.
  - `coverage_simulation_main.m` : a file used to run the simulation of one chosen controller.
  - `coverage_simulation_main_auto.m` : a file that is used by the file `auto_test.m` to run a full test autonomously.
  - `data_processing.m` : a program that displays the results.
  - `LloydController.m`, `TVDCortes.m`, `TVDD1.m`: the classes implementing the controllers.
  - `VoronoiRegion.m`, `Target.m` : other utility classes.

  There is also a sub-folder `Results`. This folder contains the simulation results after running `auto_test.m`. It was run before submitting this work. The `.mat` files are Matlab data objects used to save or load information in the programs.
- A folder `videos` containing the Robotarium videos. It contains the videos in `target-tracking-comparison` of the experiments, which have produced the results presented in Figure 4, along with 2 other videos showing a uniform



(a) $\tilde{v} = v + \alpha\mathbb{1}$ (b) $\tilde{v} = v - \alpha\mathbb{1}$

Fig. 2: Robustness comparison of the different algorithms to noise or wrong estimations of the target's velocity. In the experiment presented in this Figure, the velocity of the target was $0.133\ m/s$, $N = 4$ agents, $k = 0.9$, and the agents started at the same initial poses and were scattered in the coverage area. The simulation where the agents had access to the right estimate of the target's velocity is Figure 1d. To be able to compare the results the same outlier values were added to the target's velocity for the different algorithms. Independent simulations were needed for all three controllers. The experiment shown here gives a totally wrong estimation $\tilde{v}$ for the target's velocity v, $\alpha = 1.5\ m/s$.

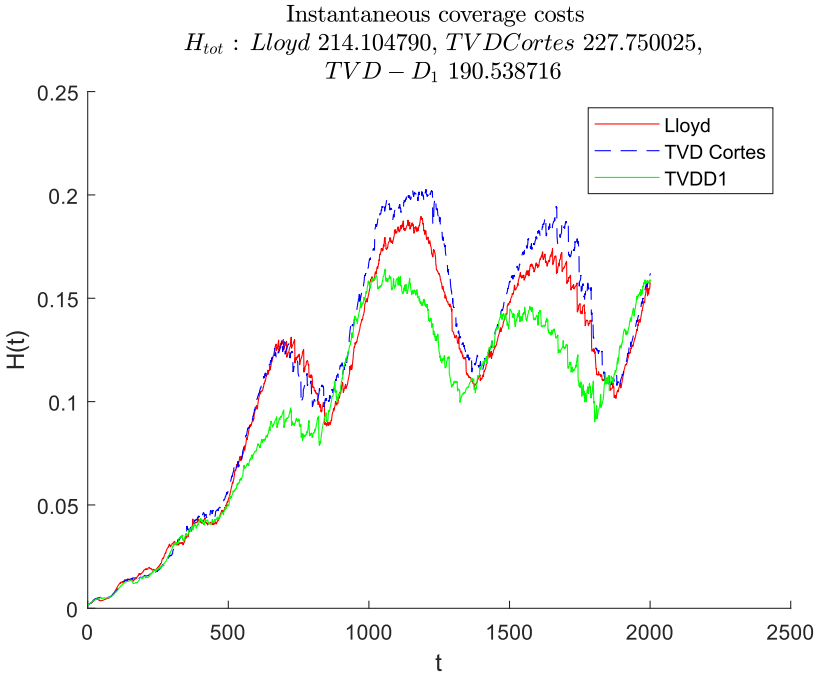


Fig. 3: Robustness comparison of the different algorithms to noise on the target's position. In this experiment the velocity of the target was $0.133\ m/s$, $N = 4$, $k = 0.9$, and the agents started at the same initial poses and were scattered in the coverage area. The simulation where the agents had access to non degrading estimates of the target's position, i.e. $\gamma = 10$ and the variance was kept constant, is Figure 1d. This experiment models the absence of a filtering mechanism on the target's position. The variance σ was gradually grown and then stabilized after 1000 iterations to see how the different algorithms would behave. Equivalently, the value of γ was gradually decreased, $\gamma \in [1; 1000]$.

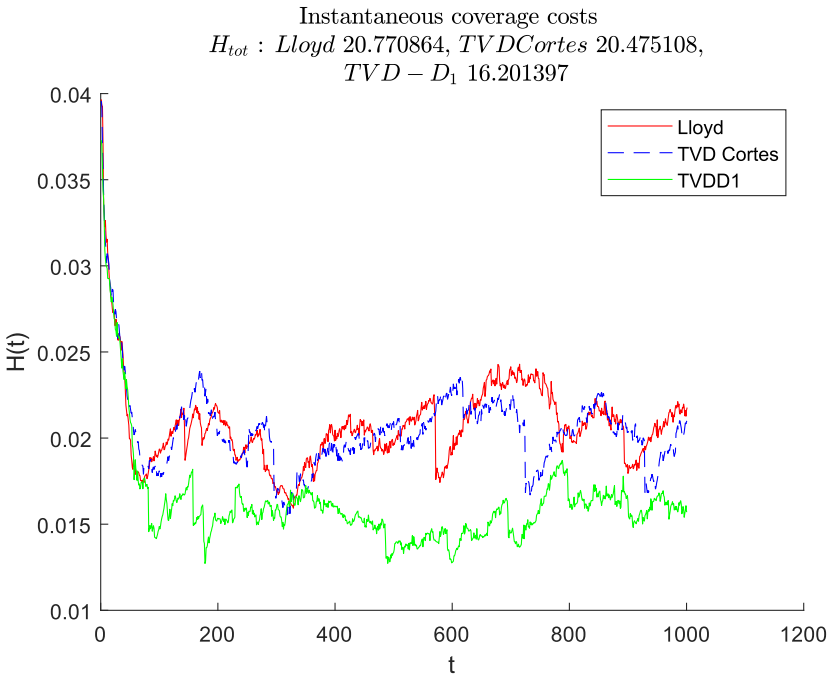


Fig. 4: Instantaneous coverage costs obtained in a hardware experiment. The simulation with the same initial conditions and parameters is the Figure 1d. Videos for this test and for each controller are given along with this work.

coverage, and one with a density function describing a line formation.

APPENDIX B
HOW TO RUN THE SIMULATION

To run this code you should download the Robotarium simulator. Run `init.m` provided with the package to add the Robotarium utilities to your current path, and then you can run the code `auto_test.m`. It will run 3 successive simulations. One for each controller. At the end the results will be plotted on the screen and the results will be stored in the `Results` folder. Should you want to add more agents, and change other parameters of the simulation, the parameters are explained and can be changed in the first section of `auto_test.m`.

REFERENCES

[1] H. Choset, "Coverage for robotics – a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 113 – 126, October 2001.

[2] J. CORTES and M. EGERSTEDT, "Coordinated control of multi-robot systems: A survey," *SICE Journal of Control, Measurement, and System Integration*, vol. 10, no. 6, pp. 495–503, 2017.

[3] M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*, stu - student edition ed. Princeton University Press, 2010. [Online]. Available: http://www.jstor.org/stable/j.ctt1287k9b

[4] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks: variations on a theme," *IEEE Transactions on Robotics and Automation*, 2002.

[5] ——, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[6] Cortés, Jorge, Martínez, Sonia, and Bullo, Francesco, "Spatially-distributed coverage optimization and control with limited-range interactions," *ESAIM: COCV*, vol. 11, no. 4, pp. 691–719, 2005. [Online]. Available: https://doi.org/10.1051/cocv:2005024

[7] J. Cortes, "Coverage optimization and spatial load balancing by robotic sensor networks," *IEEE Transactions on Automatic Control*, vol. 55, no. 3, pp. 749–754, 2010.

[8] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[9] S. G. Lee, Y. Diaz-Mercado, and M. Egerstedt, "Multirobot control using time-varying density functions," *IEEE Transactions on Robotics*, vol. 31, no. 2, pp. 489–493, 2015.

[10] S. G. Lee and M. Egerstedt, "Controlled coverage using time-varying density functions*," *IFAC Proceedings Volumes*, vol. 46, no. 27, pp. 220 – 226, 2013, 4th IFAC Workshop on Distributed Estimation and Control in Networked Systems (2013). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667015402319

[11] L. Pimenta, M. Schwager, Q. Lindsey, V. Kumar, D. Rus, R. Mesquita, and G. Pereira, "Simultaneous coverage and tracking (scat) of moving targets with robot networks," vol. 57, 01 2008, pp. 85–99.

[12] Y. Diaz-Mercado, S. G. Lee, and M. Egerstedt, "Distributed dynamic density coverage for human-swarm interactions," in *2015 American Control Conference (ACC)*, 2015, pp. 353–358.

[13] J. Kennedy, A. Chapman, and P. M. Dower, "Generalized coverage control for time-varying density functions," in *2019 18th European Control Conference (ECC)*, 2019, pp. 71–76.

[14] M. Santos, S. Mayya, G. Notomista, and M. Egerstedt, "Decentralized minimum-energy coverage control for time-varying density functions," in *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 2019, pp. 155–161.

[15] A. Pierson, Z. Wang, and M. Schwager, "Intercepting rogue robots: An algorithm for capturing multiple evaders with multiple pursuers," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 530–537, 2017.

[16] H. Huang, W. Zhang, J. Ding, D. M. Stipanović, and C. J. Tomlin, "Guaranteed decentralized pursuit-evasion in the plane with multiple pursuers," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, 2011, pp. 4835–4840.

[17] A. Pierson and D. Rus, "Distributed target tracking in cluttered environments with guaranteed collision avoidance," in *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 2017, pp. 83–89.

[18] D. Pickem, L. Wang, P. Glotfelter, Y. Diaz-Mercado, M. Mote, A. D. Ames, E. Feron, and M. Egerstedt, "Safe, remote-access swarm robotics research on the robotarium," *CoRR*, vol. abs/1604.00640, 2016. [Online]. Available: http://arxiv.org/abs/1604.00640

[19] S. Wilson, P. Glotfelter, L. Wang, S. Mayya, G. Notomista, M. Mote, and M. Egerstedt, "The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of

multirobot systems," *IEEE Control Systems Magazine*, vol. 40, no. 1, pp. 26–44, 2020.

[20] J. Lebl, *Basic Analysis: Introduction to Real Analysis II*, 2020, chapter 10. [Online]. Available: https://www.jirka.org/ra/realanal2.pdf

[21] M. L. Elwin, R. A. Freeman, and K. M. Lynch, "Distributed voronoi neighbor identification from inter-robot distances," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1320–1327, 2017.

[22] A. T. Kamal, J. A. Farrell, and A. K. Roy-Chowdhury, "Information consensus for distributed multi-target tracking," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2403–2410.

[23] A. Benevento, M. Santos, G. Notarstefano, K. Paynabar, M. Bloch, and M. Egerstedt, "Multi-robot coordination for estimation and coverage of unknown spatial fields," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 7740–7746.

[24] M. Schwager, J. Slotine, and D. Rus, "Consensus learning for distributed coverage control," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 1042–1048.