VORONOI MULTI-ROBOT COVERAGE CONTROL IN NON-CONVEX ENVIRONMENTS WITH HUMAN INTERACTION IN VIRTUAL REALITY

Lucas Coelho Figueiredo*, Ítalo Lelis de Carvalho*, Luciano Cunha de Araújo Pimenta*

*Graduate Program in Electrical Engineering - Federal University of Minas Gerais
Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brasil

Emails: lucascoelho91@ufmg.br, italolelis@ufmg.br, lucpim@cpdee.ufmg.br

Abstract— This work presents the implementation of a robot coverage algorithm in non-convex environments with the interaction of the robot group with a human agent in a virtual reality (VR) environment. On the virtual reality application, the control of the group of robots is abstracted to simple commands, allowing the users to interact with the swarm by creating and removing obstacles, creating density distribution functions and altering the speed of robots. Simulations were implemented using ROS and Unity and demonstrate that robots could react to the changes of the simulation environment imposed by the human agent.

Keywords— Human swarm interaction, multi-robot coverage, multi-robot systems, virtual reality, teleoperation

Resumo— Este trabalho apresenta a implementação de um algoritmo de cobertura por múltiplos robôs em ambientes não convexos com a interação de um agente humano em ambiente de realidade virtual (VR). Na aplicação VR, o controle dos múltiplos robôs é abstraído para simples ações, permitindo com que o usuário interaja com o grupo de robôs criando e removendo obstáculos, criando funções de distribuição de densidade e alterando a velocidade dos robôs. Simulações foram implementadas usando ROS e Unity e demonstram que os robôs reagem a mudanças no ambiente de simulação impostas pelo agente humano.

Palavras-chave— Interação humano enxame, cobertura por múltiplos robôs, sistemas de múltiplos robôs, realidade virtual, teleoperação

1 INTRODUCTION

When dealing with robots, humans are very much concerned how could they interact with them and operate, in applications varying from military to entertainment, medicine to transportation and many other fields. Because of the decreasing costs of robots and introduction of simpler and more robust communication methods, one of the hottest topics in this field of robot teleoperation is human-swarm interaction (HSI) (Kolling et al. 2016). Some works in HSI, for example, allowed humans influencing the actions of groups of robots by using tablets (Diaz-Mercado et al. 2015) and webpages (Becker et al. 2014). In this work, we are interested in methods of robot coverage and control, to be used as an abstraction to allow smooth human swarm interaction using a virtual reality headset. Multi-robot coverage has also been a largely studied problem in robotics (Cortes et al. 2004, Lloyd 1982, Pimenta et al. 2008, Pierson et al. 2015, 2017, Bhattacharya et al. 2013, Breitenmoser et al. 2010), in which we are concerned in finding an optimal displacement of a group of robots which minimizes a specific criteria. Typical applications to multi-robot coverage are, for example surveillance and rescue, sensing and military. In this work, we are combining the weighted coverage present in Pimenta et al. (2008), using adaptive weightings as in (Pierson et al. 2015, 2017), but in non-convex environments, like in (Pimenta et al. 2008, Bhattacharya et al. 2013, Breitenmoser et al. 2010). When involving HSI with robot coverage, previous works are for example, concerned in enabling humans to alter the density probability functions (Diaz-Mercado et al. 2015), which can increase the presence of robots in some specific areas, using a tablet. However, in this work, we are investigating the interaction between humans and robots using a virtual reality headset.

Due to the availability of virtual reality consumer electronics nowadays, there is a strong trend in applying virtual reality in the context of robot control and teleoperation. In El-Shawa et al. (2017), authors investigated VR as a HRI for simulation in training, considering environments which robots and humans work together and collaborate. In Nigolian et al. (2017), they have implemented an interface that allowed a group of self-reconfigurable modular robots (SRMR) to be controlled by an operator. They have used Roombots modules (Spröwitz et al. 2014), which have SRMR capabilities and in their experiment, they have used a group of them to form structures.

Other recent works in VR applications involving robotics allows users, for example, to interact with manipulators using VR and a joystick (Guerin et al. 2014), groups of robots and other humans in mixed reality (Hönig et al. 2015), or teleoperating mobile robots with a headset and a joystick (Du et al. 2016). However, in this work we focus on head mounted VR, with motion tracking of the user and handheld wireless controllers, which allow a deeper level of immersiveness and control of the group of robots.

This work is divided as the following: Section 2 describes the multi-robot coverage problem, Sec-

tion 3 describes the implementation of the application, Section 4 shows the simulation results.

2 Multi-robot coverage in non-convex environments

Let $\Omega \subset \mathbb{R}^2$ be a given representation of the environment, $P = \{p_1, \ldots, p_n\}$ be the configuration $p_i = [x_i, y_i]$ of n robots, where $p_i \subset \Omega$ and q a point in Ω. Like in (Pimenta et al. 2008, Pierson et al. 2015, 2017), we deploy heterogeneous robots, which have different weights $w = \{w_1, \ldots, w_n\}$. Let $\{V_1, \ldots, V_n\}$ be the Voronoi partition of Ω, with each cell defined as

$$V_i = \{q \in \Omega \mid d(q, p_i) \leq d(q, p_j), \quad \forall j \neq i\}.$$

where d is a function that measures distances between points in Ω and agents. Ω is such that it can be non-convex, like in (Pimenta et al. 2008, Bhattacharya et al. 2013). Also, for our work, we use the weighted Voronoi partition, also known as the **Power Diagram** (Pimenta et al. 2008), with each weighting w_i serving as the performance weighting for robot i, similar to (Pierson et al. 2015, Pimenta et al. 2008, Pierson et al. 2017). To couple with such considerations, we define the d function as the geodesic power distance:

$$d(q, p_i) = g(q, p_i)^2 - w_i^2. \tag{1}$$

Let $\{W_1, \ldots, W_n\}$ be the geodesic weighted Voronoi partition of Ω, with each cell defined as

$$W_i = \{q \in \Omega \mid g(q, p_i)^2 - w_i^2 \leq g(q, p_j)^2 - w_j^2, \quad \forall j \neq i\}. \tag{2}$$

As a measure of the system performance we define the *deployment functional*:

$$\mathcal{H}(P, W) = \sum_{i=1}^{n} \int_{W_i} [g(q, p_i)^2 - w_i^2] \phi(q) dq\,, \tag{3}$$

where the function $\phi : \Omega \to \mathbb{R}_+$ is a density distribution function which defines a weight for each point in Ω. Therefore, points with greater weight values should be better covered by the networked robots than points with smaller values.

With our deployment functional defined, the objective of the coverage algorithm is to find an optimal robot configuration P such that it minimizes $\mathcal{H}(P, W)$. In Cortes et al. (2004), it was proven that the critical values of the functional are such that all robots are in the centroid of their Voronoi cells. One way to drive the robots to their centroids is to use the Lloyd's algorithm (Lloyd 1982), in such way that we could reach the optimal robot configuration by: 1) constructing the Voronoi diagram, 2) computing the centroids of the Voronoi regions, 3) setting the agents position to the centroid positions, and repeat from 1).

In order to minimize the value of $\mathcal{H}(P, W)$, a control law to drive the robots needs to be chosen. Considering a holonomic robot, such that $\dot{p}_i = u_i$, and as proven in Pimenta et al. (2008), in convex environments and continuous time, a control law for a holonomic robot that can minimize $\mathcal{H}(P, W)$, is:

$$u_i = -K_i \frac{\partial \mathcal{H}}{\partial p_i}\,, \tag{4}$$

which K_i is a 2×2 positive definite gain matrix. The previous Equation can be expanded as:

$$\frac{\partial \mathcal{H}}{\partial p_i} = 2(p_i - p_i^*) \int_{W_i} \phi(q) dq\,, \tag{5}$$

where p_i^* is the centroid of W_i:

$$p_i^* = \frac{\int_{W_i} q\phi(q) dq}{\int_{W_i} \phi(q) dq}. \tag{6}$$

However, in non-convex environments, the centroid p_i^* might be outside W_i. To address for such problems, we use a control law adapted from Pimenta et al. (2008). This control law allows the agents to go towards the direction which will reduce the value of the deployment functional, but not taking into account the position of the centroid. Such control law is given by:

$$u_i = -K_i \frac{\partial \mathcal{H}}{\partial p_i} = 2K_i \int_{W_i} g(q, p_i) \phi(q) z_{p_i, q} dq, \tag{7}$$

where $z_{p_i,q}$ is a unit vector tangent at p_i to the minimal path connecting p_i and q.

In regards to the gain K_i used, it is a matrix with a gain $k_{p_i} > 0$. Note that every robot can have a different gain. The K_i matrix is represented as:

$$K_i = \begin{bmatrix} k_{p_i} & 0 \\ 0 & k_{p_i} \end{bmatrix} \tag{8}$$

Also, as in Pierson et al. (2015), the weights of our agents are performance-based. This means that the value of $w = \{w_1, \ldots, w_n\}$ is not fixed, but rather adjusted depending on the value of K_i present in Equation (7) and Equation (8).

A controller that can adjust the weights of the robots based on their K_i gains could be:

$$\dot{w}_i = \frac{-k_w}{M_{W_i}} \sum_{j \in \mathcal{N}_i} ((w_i - f(K_i)) - (w_j - f(K_j))) \tag{9}$$

where k_w is a positive proportional gain constant, $f(K_i)$ is some function of the properties of K_i, $\mathcal{N}_i$ is a list of Voronoi cell neighbors of robot i

and M_{W_i} is the mass of the Voronoi partition W_i, which can be calculated as:

$$M_{W_i} = \int_{W_i} \phi(q) dq. \tag{10}$$

Proofs in Pierson et al. (2015) show that the weightings using the controller in Equation (9) are bounded and that weightings converge when $(w_i - f(K_i)) - (w_j - f(K_j))$ is equal for all i and j.

3 Implementation

3.1 Voronoi coverage control algorithm design

In order to implement the control algorithm, we need a way to calculate the Voronoi tessellations. In this work, we use a discrete graph search based in Bhattacharya et al. (2013). The environment Ω is discretized in uniform tiling, creating a graph G with vertexes $\mathcal{V}(G)$ and edges $\mathcal{E}(G)$ and a discretized density distribution $\overline{\phi}$. The idea behind the algorithm present in Bhattacharya et al. (2013) is to use a modified version of Dijkstra's algorithm (Dijkstra 1959), in which each robot would be a different starting Voronoi sites of the algorithm. The places where the wavefronts from different robots collide form the boundaries of the Voronoi tessellation. For this work, we modified the algorithm from Bhattacharya et al. (2013) in order to obtain the Voronoi partitions that are neighbors of one another, which is necessary for adjusting the weights as in Equation (9). The algorithm was implemented using Python and ROS. The source code is also available at GitHub. [1].

The complete pseudo-code for computing tessellations and control commands is the **Adapted_Tessellation_and_Control** algorithm, where η is a list of neighbors of a specific vertex and $P(u)$ is a function that returns the position of the center of the node that the two-dimensional array u belongs to. Lines 33, 34 and 35 were added to the original algorithm described in Bhattacharya et al. (2013) to add the determination of the Voronoi cell neighbors $\mathcal{N}_i$.

[1]Source code available at `https://github.com/lucascoelhof/voronoi_hsi`

$\{\tau, \{p_i'\}, \mathcal{N}_i, M_{W_i}\}$ = **Adapted_Tessellation_and_Control** $(G, \{p_i\}, \{w_i\}, \overline{\phi})$

Modified from Bhattacharya et al. (2013)

Inputs: a. Graph G
b. Agent locations $p_i \in \Omega,\ i = 1, 2, \cdots, N$
c. Agent weight $w_i \in \mathbb{R},\ i = 1, 2, \cdots, N$
d. Discretized density distribution function $\overline{\phi}$

Outputs: a. The tessellation map $\tau \in \mathbb{I}^2$
b. The next position of each robot, $p_i' \in \Omega,\ i = 1, 2, \cdots, N$
c. The list of neighbors of each robot, $\mathcal{N}_i\ i = 1, 2, \cdots, N$
d. Mass of Voronoi partitions $M_{W_i},\ i = 1, 2, \cdots, N$

```
1  Initiate g: Set g(v) := ∞,
                for all v ∈ V(G)  // Geodesic distances
2  Initiate ρ: Set ρ(v) := ∞, ∀v ∈ V(G)  // Power dists.
3  Initiate τ: Set τ(v) := −1, ∀v ∈ V(G)  // Tessellation
4           // Pointer to robot neighbor. η : V(G) → V(G)
5  Initiate η: Set η(v) := ∅,  ∀v ∈ V(G)
6  for each (i ∈ {1, 2, ··· , N})
7      Set g(p_i) = 0
8      Set ρ(p_i) = −w_i^2
9      Set τ(p_i) = i
10     Set I_i := 0    // The control integral. I_i, 0 ∈ C
11     for each (q ∈ η_G(p_i))  // For each neighbor of p_i
12         Set η(q) = q
13 Set Q := V(G)    // Set of un-expanded vertices
14 while (Q ≠ ∅)
15     q := arg min_{q'∈Q}  ρ(q')  // Maintained by a heap
                                   data-structure.
16     if (g(q) == ∞)
17         break
18     Set Q = Q − q    // Remove q from Q
19     Set j := τ(q)
20     Set s := η(q)
21     if (s != ∅)    // Equivalently, q ∉ {p_i}
22         Set I_j += φ̄(q) × g(q) × (P(s) − P(p_j))
23         Set M_{W_i} += φ̄(q)
24     for each (w ∈ η_G(q))    // For each neighbor of q
25         Set g' := g(q) + g(q, w)
26         Set ρ' := d(g', w_j)
27         if (ρ' < ρ(w))
28             Set g(w) = g'
29             Set ρ(w) = ρ'
30             Set τ(w) = j
31             if (s != ∅)    // Equivalently, q ∉ {p_i}
32                 Set η(w) = s
33         else
34             m = τ(w)    // Gets robot id
               // Adds robot m to the list of neighbors of j
35             N_j += m
36 for each (i ∈ {1, 2, ··· , N})
37     Set p_i' := arg max_{u∈η_G(p_i)} (P(u) − P(p_i)) · I_i
           // Choose action best aligned along I_i.
38 return {τ, {p_i'}, N_i, M_{W_i}}
```

3.1.1 Graph construction

The graph G is constructed using the occupancy grid data provided from a ROS topic. The occupancy grid is a $M \times N$ matrix, in which each element represents a small discretized area from the map along a probability of that area being an obstacle, with values varying from 0% to 100%.

This matrix is used to build a graph that connects the vertexes with their neighboring vertexes based on the probability of obstacle. If the neighbor has over 20% obstacle probability value, it is considered an obstacle and thus it is not connected to the graph. The algorithm can also receive at any time new occupancy grid information from the VR application, and when it does, a new graph needs to be created.

3.1.2 Motion Algorithm

To couple with the possibility of dynamically changing the environment Ω, some small changes to the Adapted Lloyd's algorithm present in Bhattacharya et al. (2013) were implemented. But the overall algorithm still consists of iterating over "**Adapted_Tessellation_and_Control**" and updating the positions of the robots at each iteration. The pseudo algorithm follows:

$\{\tau^f, \{p_i\}^f, \{w_i\}^f\}$ = **Motion_Algorithm** $(G, \{p_i\}, \{W_i\}, \overline{\phi})$

Inputs: a. Occupancy Grid O_G
b. Initial agent locations $p_i \in \Omega$, $_{i=1,2,\cdots,N}$
c. Discretized density distribution function $\overline{\phi}$
d. Agent weight $w_i \in \mathbb{R}$, $_{i=1,2,\cdots,N}$

Outputs: a. Final tessellation map $\tau \in \mathbb{I}^2$
b. Robot final position, $p_i' \in \mathcal{V}(G)$, $_{i=1,2,\cdots,N}$
b. Robot final weightings, $w_i' \in \mathbb{R}$

while $(t < n)$ // not converged
  $G =$ **build_graph**(O_G)
  Set $\{\tau, \{p_i'\}, M_{W_i}\} :=$ **Adapted_Tessellation_and_Control** $(G, \{p_i\}, \{w_i\}, \overline{\phi})$
  for each $(i \in \{1, 2, \cdots, N\})$
    Move i^{th} robot from $P(p_i)$ to $P(p_i')$
    Set $p_i = p_i'$ // Update robot positions
    Adapt_Weightings(w_i, M_{W_i})
return $\{\tau, \{p_i\}\}$ // Latest tessellation & positions

where **Adapt_Weightings** is an implementation of the Equation (9) and **build_graph** is a function that builds the graph as described in subsection 3.1.1 when there are changes on occupancy grid O_G. t is the current elapsed time and n is a time set for convergence.

3.2 Virtual reality application

Virtual reality can bring some benefits to human swarm interaction. Some of the main advantages of VR robot teleoperation are:

3.2.1 Physical Isolation

On virtual reality, the user does not need to be close to the agents to control them, as long the agents still have communication with the human who is controlling the swarm. This greatly expands the operational flexibility because it removes the distance constraints, allowing the human to control the swarm even when the robots are in dangerous or inhospitable locations.

3.2.2 Robot and environment additions

Since the world is a computer graphics world, we are free to modify it and add any features that are impossible in real world. For example, for this work we implemented a flyover and teleporting feature, which is totally feasible in virtual world, but not yet fully replicable in real world for regular humans.

3.2.3 Intuitivity and immersion

A virtual reality headset with motion tracking and handheld wireless controllers feels very intuitive to humans. It gives them the freedom to look and walk around and interact with the world with common gestures, like grabbing, pinching and pressing. Also, because of the high field of view and resolution, the users feel totally immersed on the virtual world.

On the implementation of the virtual reality application, we used *Oculus Rift*. Since Oculus is only officially supported in Windows and ROS in Ubuntu, we used ROS in an Ubuntu virtual machine hosted by a Windows computer, which is running Unity to create the virtual reality application, with the help of VRTK. VRTK provides tools and libraries that help on the development of VR applications in Unity. To communicate between Unity and ROS, we used ROS# (Bischoff (2018)), which is a set of libraries and tools in C# for communication between ROS and C# applications. We used *Stage* simulator [2] for robot simulation, *map_server*[3] for reading maps from image files, and *rosbridge* [4] to communicate with ROS#. Figure 1 shows the setup used for the simulations. The Linux VM was used only for simplification, as we could have two separate computers using exactly the same implemented software.

Oculus Rift is a virtual reality headset developed by Oculus VR. It has a 1080x1200 resolution OLED display per eye, with a 110° field of view. Oculus has also a set of two controllers, called Oculus Touch, and a positional tracking system, *Constellation*. It uses two infrared cameras to get the position of the headset and the two controllers in three dimensions. Figure 2 shows an user wearing the *Oculus Rift*. For this work, a VR application was developed using Unity, ROS# and Oculus Rift, and this application is available as open source[5].

[2] Stage simulator `http://wiki.ros.org/stage`

[3] ROS map_server package. `http://wiki.ros.org/map_server`

[4] Rosbridge package.`http://wiki.ros.org/rosbridge_suite`

[5] Source code available at `https://github.com/lucascoelhof/VoronoiUnityTeleoperation`

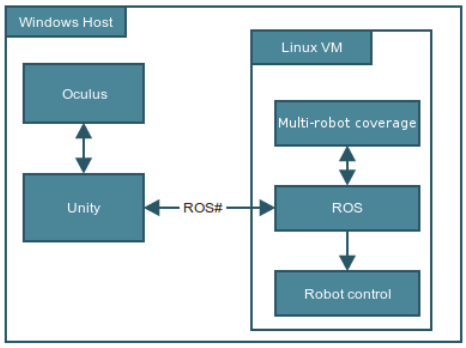

Figure 1: Setup for interfacing ROS and Oculus Rift.

Figure 2: User wearing the virtual reality headset.

On the VR application, users can see the robots, obstacles, the Voronoi tessellation and the density distribution function. In order to interact with the robots, users can teleport, fly over, create and remove obstacles, set robot weights and create density distribution functions. These are the implemented interactions between the users and the group of robots:

3.2.4 Obstacle creation and removal

When users press the A button, an obstacle is created in the location of their right arm. When creating this obstacle, ROS# publishes on ROS the new occupancy grid, and the Voronoi application reads it and updates the graph and the tessellation. Likewise, they can also remove obstacles using B button. Since the graphs are updated and the tessellation algorithm runs regularly, the robots can react to the new obstacles.

3.2.5 Density function interaction

The Voronoi algorithm implemented allows the use of density distribution functions, which can attract the robots. For simplicity, we modeled these density distributions as a 2 dimensional Gaussian function. When the user presses the right trigger button, a Gaussian is created. The function is modeled using the following equation,

$$f(x,y) = B + Ae^{-\left(\left(\frac{x-x_c}{\sigma}\right)^2+\left(\frac{y-y_c}{\sigma}\right)^2\right)} \qquad (11)$$

where A is the height where the arm of the user is, B is a constant, x_c and y_c are the position of the right arm in relation to the tessellation plane in each respective coordinate and σ can be controlled by moving the left arm in relation to the right arm. Approaching the arms reduces the value of σ while moving them away increases σ. When the user sets the new density function, ROS# sends the Gaussian parameters to the Voronoi application, which regenerates the density distribution and updates the Voronoi diagram.

3.2.6 Robot gains and weight

To allow humans to manually set the speed gain of the robots, they can select a speed gain k_{p_i} and then hold the right trigger button to create a radius that is used to select a group of robots and increase their k_{p_i} gains, as in Figure 3. As shown on Equation (2) and (9), this will have impact on the weight values of the robots and thus, their Voronoi partition W_i. When a value is set, we use ROS# to publish to our Voronoi application the new values of k_{p_i}.

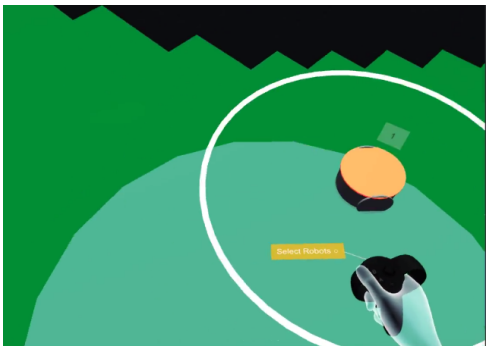

Figure 3: User selecting a robot.

3.2.7 Teleporting

Although using the *Oculus Constellation* the users can walk freely on the virtual world, it can be tiring or limiting to walk around always. To solve this problem, we implemented a teleporting feature using *VRTK*. When users press the right thumb button in *Oculus Touch*, a laser pointer shows up, and when the button is released, the user is teleported to the pointer location.

3.2.8 Aerial view

To better see the world and the robots and get a panoramic view of the simulation, we imple-

mented a feature that allows users to have an aerial view of the simulated area. When users press the left right thumb button, they are teleported to a place above, with a better view. Also, using the aerial view they can produce density distributions with values much higher than standing on ground.

4 Simulations

To demonstrate the VR application and the Voronoi controller, we conducted simulations [6]. For a better understanding of the simulation and results, we suggest to watch the video available in the footnote. Using a rectangular environment with three obstacles in the middle, the results demonstrate the interaction between the user and eight robots and the convergence of the algorithm even after modifications made by the human operator. This simulation could also work with a bigger number of robots, but we have chosen eight for arbitrary reasons.

All agents start with $k_{p_i} = 1$. The white radius around the robots show their current weight. On simulation, we have used differential drive robots, so we have used a point-offset controller (Michael & Kumar 2009) to address the nonholonomic constraints. To show the convergence of the algorithm, the human agent didn't interact with the group of robots until around $t = 150$. Then, the human agent selects robot 1 (green) to increase its k_{p_i} value to 3.2. Because of the adaptive weightings controller on Equation (9), increasing the k_{p_i} value increases the weight of robot 1, while decreasing the weightings of other robots, as shown on Figure 10. Since we use a geodesic power distance as in Equation (2), increasing the weight of the robot will increase its Voronoi partition V_1. At $t = 200$, the human agent created a Gaussian shaped density distribution function centered on the intersection of the Voronoi partitions of robots 7 (orange), 2 (blue) and 5 (purple). This changed the convergence and attracted more robots to that area. Figure 7 shows the convergence of robots and the increased number of robots on that area, differently from Figure 6, which the robots are more spread around. Then, the human agent creates a flat density function, which removes the importance of the previously set area. Around $t = 300$, the human agent changed the occupancy grid of the area, connecting some of the obstacles in the area. On Figure 8, we can see that because of the dynamic graph generation present on the **Motion_Algorithm**, the robots could react to the changes of the obstacles and reached another optimal distribution. Figure 9 show the deployment functional over time. Spikes present around $t = 200$ and $t = 250$ are because of changes on the density distribution function.

Figure 4: Distribution of robots and tesselation at $t = 0$

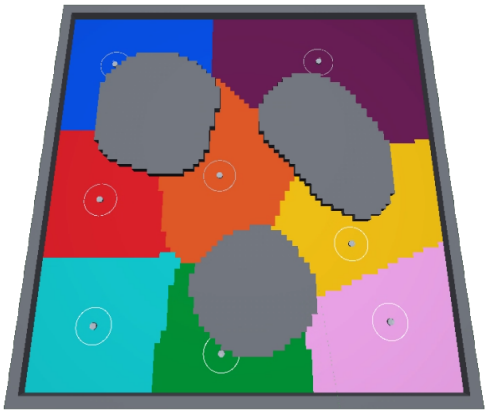

Figure 5: Convergence of robots to an optimal position at $t = 150$

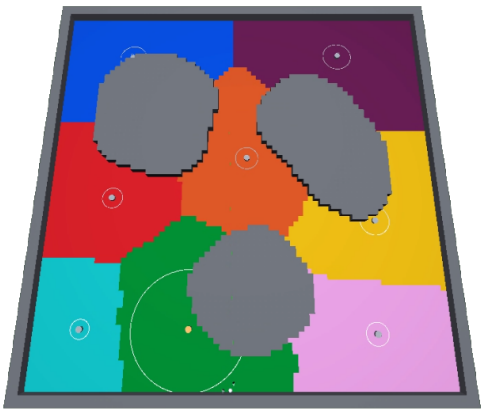

Figure 6: Robot 1 with increased weight, with a larger Voronoi partition than on Figure 5

5 CONCLUSION

In this paper, authors have implemented an application for coverage control of non-convex environ-

[6] Simulation video available at `https://www.youtube.com/watch?v=cpniwb6UrF8`

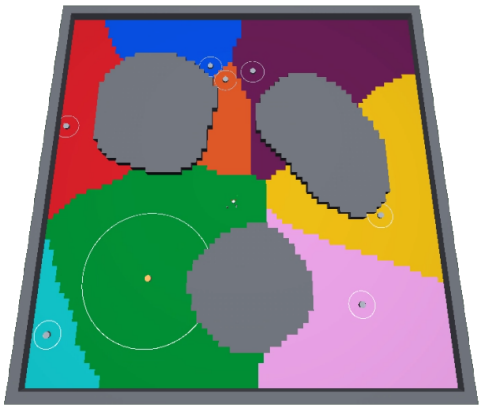

Figure 7: Convergence showing robots attracted by a Gaussian density function.

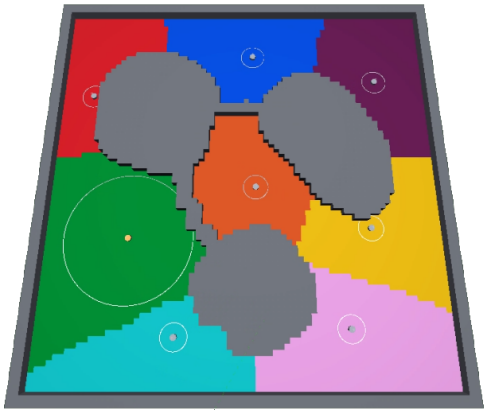

Figure 8: Changes on the obstacles were introduced but robots were able to reach a new convergence.

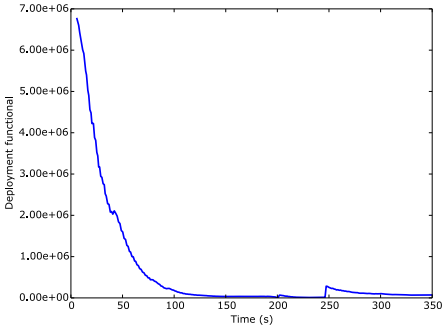

Figure 9: Deployment functional over the simulation.

ments with the interaction of a human agent using a virtual reality headset. Users could then interact with the swarm by creating obstacles, changing the density distribution function (and thus altering the concentration of robots on the area), selecting robots to increase their speed and their power on the area and move freely on the area, either by walking around, teleporting or flying. We could show that teleoperating a group of robots in VR can be beneficial because it gives a very deep

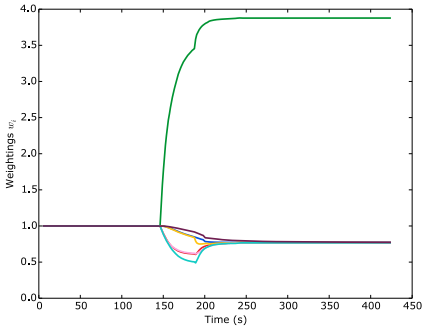

Figure 10: Weightings of the robots. Around $t = 150$, the human changed the gain k_{p_i} of robot 1 (green), which changed the weightings of all robots.

level of immersion, but at the same time keeping the human physically isolated, which could be applied for example for teleoperating swarm of robots in hazardous environments or simply too far away from the human operator. Also, it's worth mentioning that since we've used common libraries and abstraction layers, like Unity and VRTK, one could easily adapt the current software to work with any other VR headset other than *Oculus Rift*. Also, other contributions of this work are the open software *Oculus Rift* application for multi-robot interaction and the coverage control algorithm, both available at *GitHub*.

Future work may implement a mixed reality concept, in which the view of the human operator could come from another robot, like for example a quadcopter flying over the swarm of robots. Other possibilities may also include ways of treating the creation of obstacles in such way that when an obstacle is created, it would create a repulsion on the robots to avoid robots colliding with the newly created obstacles.

ACKNOWLEDGMENT

We gratefully acknowledge the support from the Brazilian institutions FAPEMIG, CNPq and CAPES for this work.

References

Becker, A., Ertel, C. & McLurkin, J. (2014), 'Crowdsourcing swarm manipulation experiments: A massive online user study with large swarms of simple robots', *2014 IEEE International Conference on Robotics and Automation (ICRA)* pp. 2825–2830.

Bhattacharya, S., Ghrist, R. & Kumar, V. (2013), 'Multi-robot coverage and exploration on Riemannian manifolds with boundaries', *Int. J. Rob. Res.* **33**(1), 113–137.

Bischoff, M. (2018), 'ROS#'.
URL: *https://github.com/siemens/ros-sharp*

Breitenmoser, A., Schwager, M., Metzger, J. C., Siegwart, R. & Rus, D. (2010), 'Voronoi coverage of non-convex environments with a group of networked robots', *2010 IEEE International Conference on Robotics and Automation* pp. 4982–4989.

Cortes, J., Martinez, S., Karatas, T. & Bullo, F. (2004), 'Coverage control for mobile sensing networks', *IEEE Transactions on Robotics and Automation* **20**(2), 243–255.

Diaz-Mercado, Y., Lee, S. G. & Egerstedt, M. (2015), 'Distributed dynamic density coverage for human-swarm interactions', *2015 American Control Conference (ACC)* pp. 353–358.

Dijkstra, E. W. (1959), 'A note on two problems in connexion with graphs', *Numer. Math.* **1**(1), 269–271.

Du, J., Sheng, W. & Liu, M. (2016), 'Human-guided robot 3D mapping using virtual reality technology', *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* pp. 4624–4629.

El-Shawa, S., Kraemer, N., Sheikholeslami, S., Mead, R. & Croft, E. A. (2017), '"Is this the real life? Is this just fantasy?": Human proxemic preferences for recognizing robot gestures in physical reality and virtual reality', *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* pp. 341–348.

Guerin, K. R., Riedel, S. D., Bohren, J. & Hager, G. D. (2014), 'Adjutant: A framework for flexible human-machine collaborative systems', *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* pp. 1392–1399.

Hönig, W., Milanes, C., Scaria, L., Phan, T., Bolas, M. & Ayanian, N. (2015), 'Mixed reality for robotics', *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* pp. 5382–5387.

Kolling, A., Walker, P., Chakraborty, N., Sycara, K. & Lewis, M. (2016), 'Human Interaction With Robot Swarms: A Survey', *IEEE Trans. Hum.-Mach. Syst.* **46**(1), 9–26.

Lloyd, S. (1982), 'Least squares quantization in PCM', *IEEE Trans. Inf. Theory* **28**(2), 129–137.

Michael, N. & Kumar, V. (2009), 'Planning and Control of Ensembles of Robots with Non-holonomic Constraints', *Int. J. Rob. Res.* **28**(8), 962–975.

Nigolian, V., Mutlu, M., Hauser, S., Bernardino, A. & Ijspeert, A. (2017), 'Self-reconfigurable modular robot interface using virtual reality: Arrangement of furniture made out of roombots modules', *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)* pp. 772–778.

Pierson, A., Figueiredo, L. C., Pimenta, L. C. A. & Schwager, M. (2015), 'Adapting to performance variations in multi-robot coverage', *2015 IEEE International Conference on Robotics and Automation (ICRA)* pp. 415–420.

Pierson, A., Figueiredo, L. C., Pimenta, L. C. & Schwager, M. (2017), 'Adapting to sensing and actuation variations in multi-robot coverage', *Int. J. Rob. Res.* **36**(3), 337–354.

Pimenta, L. C. A., Kumar, V., Mesquita, R. C. & Pereira, G. A. S. (2008), 'Sensing and coverage for a network of heterogeneous robots', *2008 47th IEEE Conference on Decision and Control* pp. 3947–3952.

Spröwitz, A., Moeckel, R., Vespignani, M., Bonardi, S. & Ijspeert, A. J. (2014), 'Roombots: A hardware perspective on 3D self-reconfiguration and locomotion with a homogeneous modular robot', *Rob. Auton. Syst.* **62**(7), 1016–1033.