EMERGING TECH CONFERENCE – Edge Intelligence

Volume 01, 2022, Page 177 – 187

Proceedings of Emerging Tech Conference:
Edge Intelligence 2022

A Decentralized Collision Avoidance System using Velocity Obstacle Principles

Ioannis Daramouskas[a,b], Niki Patrinopoulou[a], Dimitrios Meimetis[a], Vaios Lappas[c], Vassilios Kostopoulos[a]

[a]*University of Patras, Rio,Patra and 26504, Greece*
[b]*Computer Technology Institute and Press "Diophantus", Rio, Patra and 26504, Greece*
[c]*Greece National Kapodistrian University of Athens, Athens, Athens and 15772, Greece*

daramousk@ceid.upatras.gr

Abstract

Drones and unmanned aerial vehicles (UAVs) have piqued the interest of researchers during the last decade, owing to their capabilities and potential for usage in a wide range of applications and areas. One of the most critical functions that drones must execute efficiently is navigation in real-world surroundings. This often involves the capacity to plan their path and avoid collisions. It is critical that drones can conduct processes related to collision avoidance while navigating around independently and effectively. In this work, we present a decentralized system for drones aiming to navigate in unknown environments and avoid moving obstacles (other drones).

1. Introduction

Unmanned aerial vehicles (UAVs) or drones, as they are now known, may fly with varying degrees of autonomy, either autonomously by onboard computers or under remote control by human operators [1][4].

Drones may be utilized in a variety of real-world activities, including aerial photography, search and rescue missions, animal census, delivering medical supplies to remote places, forest fire monitoring, surveillance, and much more. Most UAVs are outfitted with single board processors, sensors, actuators, and cameras. As a result, they are a potent tool in the field of robotics [3].

Researchers are working on optimizing onboard operating algorithms, swarming, navigation systems, and other technologies, while industry is attempting to use drones for commercial uses such as surveillance or package delivery. Swarms of drones that communicate with one another and are routinely managed. UAVs in such systems have processing capabilities, numerous sensors, and are linked to a shared communication network. Drones create swarms in this manner, fulfilling the concepts of a distributed processing system in which drones are nodes and the system fully governs itself using internal algorithms and procedures.

However, with autonomous navigation, the capacity to plan and maintain a safe flight is a significant barrier. Indeed, detecting conflicts and making decisions about how to avoid them and route to desirable locations is seen as a demanding and complex task.

The purpose of this effort is to offer a decentralized system of drones that perform tasks safely and effectively. In this work, we present an algorithm for assisting drones in navigating in unknown environments and in avoiding obstacles. The algorithm is a conflict detection and conflict resolution method in 2D that was modified and translated from python to C++, and we validated it using a setup of ROS, Gazebo, and Iris drones in two different scenarios where the drones are forced to initially follow trajectories with conflicts.

The rest of the paper is structured as follows. Related Work presents the current state of the art in the field of conflict resolution, System Overview we present the architecture of our system. In Implementation we present the algorithm we used and in Simulation and Results the results the scenarios conducted are presented validating the efficiency of the proposed algorithm.

2. Related Work

When it comes to collision avoidance, authors in [5] provide a good illustration of the well-known potential field method. In [6] the authors describe an intriguing method based on a vision sensor and 3-D geometrical calculations. The CAS employs two monocular cameras in the paper. Lastly, the work in [7] isa good illustration of a geometric approach. In [2] they describe3 systems that they examine degree of centralization and presents their systems with some conflict resolution algorithms. In [10] the present a geometrical approach for solving conflicts name Optimal Reciprocal Collision Avoidance (ORCA), a well-known and widely used method the recent years. ORCA uses Velocity Obstacle (VO) [12] principles to apply conflict detection and resolution.

3. System Overview

3.1 ROS

The open-source Robot Operating System (ROS) [8] is a robot operating system. ROS is a structured communications layer that sits on top of the host operating systems of a heterogeneous compute cluster, rather than an operating system in the traditional sense of process management and scheduling. A ROS-based system consists of several processes that operate on many hosts and are connected in a peer-to-peer topology during runtime. ROS was designed to be language agnostic. ROS now supports C++, Python, Octave, LISP, and more programming languages.

To manage the complexity of ROS, the developers adopted a microkernel strategy, in which many small tools are utilized to build and execute the various ROS components, rather than constructing a monolithic development and runtime environment. Most of the time, code generated for one project may be reused with small changes for many other projects.

3.2 Gazebo

Gazebo [9] is building a 3D dynamic multi-robot environment that can simulate the complicated situations that the next generation of mobile robots will encounter. Gazebo's open-source status, fine-grained control, and high fidelity position it as more than simply a steppingstone between the drawing board and real hardware: data visualization, remote environment simulation, and even reverse engineering of black box systems are all possibilities.

3.3 System Architecture

The proposed module we wanted to be able to wok in a decentralized UAV system, because of the great benefits of a decentralized UAV system that is supposed to be scalable and autonomous. We have adapted the algorithm's workflow to be allocated between the agents, so the system performance be not dependent on the number of the agents. Each UAV agent in the system is a carbon clone of the others, capable of gathering and processing its own data and making its own decisions based on it. Although in our system all agents share the same responsibilities and tasks, it will be possible each agent to have a variety of different tasks and responsibilities due to the modular nature of the proposed architecture.

The system was developed in C++ and python using the Robotic Operation System (ROS) [8] in the version ROS melodic. The UAV's guidance and conflict detection and conflict resolution algorithms are written in C++ whilst the visualization scripts are written in python.

The drones are simulating that they are equipped with FLARM so they can exchange information about other drones' location, speed, heading so those data can be used by the conflict detection and conflict resolution algorithm.

The SITL simulations were conducted using the GAZEBO [9] physics engine. The selected drone is an iris px4 drone,presented in Fig.1.

Figure 1: Iris drone used in our simulations

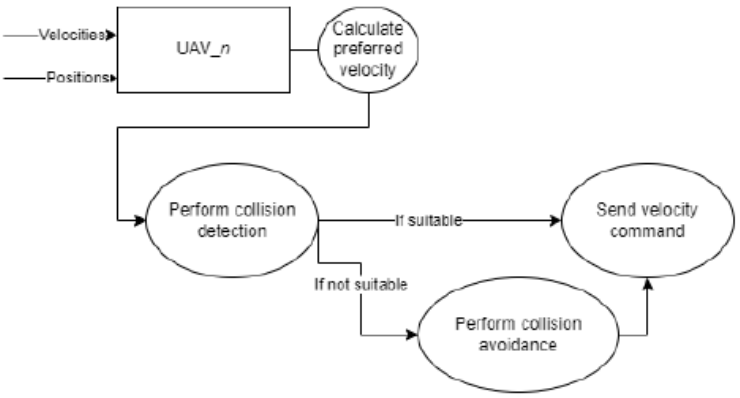

Figure 2: Flow chart for Simple RVO

In Fig.2. the flow chart of the modified algorithm, Simple RVO is given where a UAV in a given time receives information about the velocities and positions of other UAVs in the world, the collision

detection is performed using the information of the other drones and the preferred velocity, if a collision will occur in a given time, then deconfliction takes place.

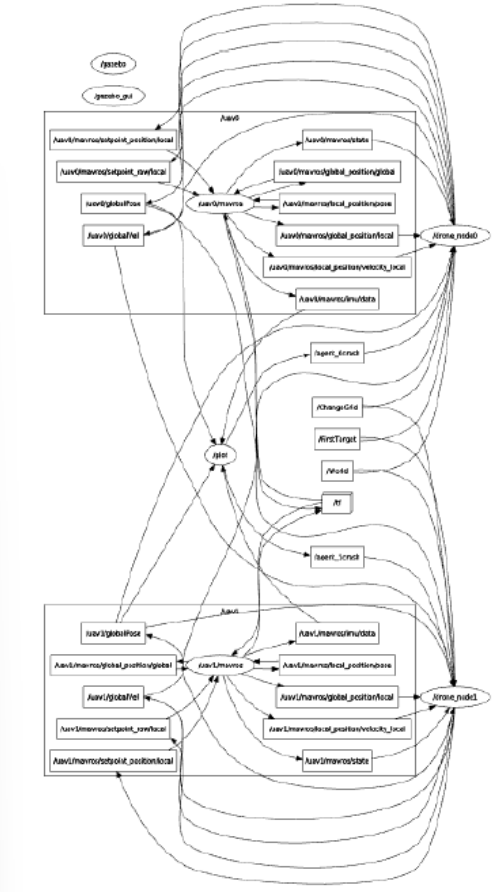

Figure 3: Ros architecture and communication of our system

In Fig. 3. we see the architecture in ROS for our drones. ROS topics are depicted with rectangles whilst ROS nodes are depicted with an ellipse. In this Fig. 3 we present 2 drones, and we can see that the ROS nodes reads information from the autopilot about heading, position in cartesian coordinates, altitude and they are sending information to the autopilot about control commands. This information is exchanged through specific topics with specific messages, and in that way each drone is establishing its communication with the autopilot. Mavros is used through MAVLink to perform communication between ROS and px4, topics like mavros/global_position/local are giving x, y, z location information to the drone and mavros/local_position/velocity_local is used to publish velocity commands from the drone to the autopilot. Agent crash topic includes a custom message that informs any drone if it has violated the separation distance.

4. Implementation

4.1 Simple RVO

The algorithm we have created for conflict resolution in the context of this paper is called Simple RVO and is version of RVO, modified to run in a decentralized architecture, in C++. RVO is a 2D technique that adheres to the Velocity obstacle concept and has been enhanced with the reciprocal feature. The code was initially created in Python [11] and was inspired by the RVO solution that was enhanced in the ORCA [10]. It is a much-reduced version of the previously

reported RVO technique, with modifications made to speed up execution and a complete conversion from Python to C++.

To counteract oscillation in the recommended velocities given by the VO [12] algorithm, the notion of reciprocal velocity barrier was developed. When the original VO is used, it is possible to establish a deadlock condition in which two UAVs oscillate between two velocities, with their viewpoint velocities switching between being in the velocity obstacle and not being in it after each timestep. Instead of selecting a new velocity outside of the velocity obstacle for each UAV, the reciprocal velocity obstacle offers a new velocity that is the average of the current velocity and a velocity outside of the other velocity obstacles.

The Reciprocal Velocity Obstacle equation can be defined as:

$$RVO\,{}^{A}_{B}(U_B, U_A) = \{U'_A | 2U'_A - U_A \in VO\,{}^{A}_{B}(U_B)\} \quad (1)$$

Once again, said code simply requires the UAV location and velocity in the x and y axes, as well as the UAV's size. It is also possible to specify a separation distance that must be followed, however unlike ORCA, there is no look ahead time and a k-d tree implementation for locating nearby UAVs during code execution. This means that on each cycle, all drones will be assessed as potential velocity impediments. Collision detection is like the 2D ORCA in that a cone is formed between the two UAVs and if the relative velocity lies within the cone, a collision will occur in the future. The optimal safe velocity for each UAV is then chosen from a set of appropriate and unsuitable speeds.

To evaluate the speedup that is offered by the C++ version we built, we employ a test bench where 25 vehicles are simulated in a 2D world along with a static obstacle and the scenario was created in such a way that all vehicles are simultaneously are part in the same conflict. We set a timestep of 0.01 seconds for a total of 200 timesteps recorded in a 2 second simulation. The algorithm is also modified to be applied in a decentralized system in opposite of its nature which is a centralized algorithm. The computational load of the algorithm is evenly distributed to the agents. The agents exchange information about their location, speed and heading and they are responsible for applying the conflict detection and conflict resolution module on their own. The modification increases the system safety and robustness because the decentralized architecture solves the single point of failure problem, latency issues and the system's safety does not deprecate by central processors limitations.

As seen in Table 1, the C++ version is over 63 times faster, completing the simulation in just 1.82 seconds compared to the 115.4 seconds of the python version. The speedup comes naturally due to C++ being compiled to native code instead of being interpreted during run time like Python. Moreover, changes were made to the C++ versions with a low memory footprint in mind along with the removal of redundant logic that was also present.

Algorithm/Metric	Execution Time (In seconds)	Iterations per second
Simple RVO (Python)	115.4	1.733
Simple RVO (C++)	1.82	109.89

Table 1: Results for execution performance of the Simple RVO algorithm

5. Simulation And Results

In this section we present the results of our simulations in two different scenarios, one with 8 drones sitting on a circle having mission to follow a straight-line trajectory to the opposite side of the circle and the other one with 5 drones executing multiple missions. The waypoints are constructed in such a way that creates conflicts with the other drones participating in that scenario. In both scenarios the separation distance is 3 meters, the conflict resolution algorithm applies changes only in two dimensions to resolve the conflict and the metrics used to evaluate the algorithm are minimum separation distance (expresses the minimum distance two drones reached before the start resolving the conflict), commanded thrust (it visualizes if the algorithm produce abrupt changes to drone's velocity), the deviation of the optimal distance and the flight time for a specific distance.

5.1 Scenario 1

In this scenario we have 8 drones located on a circle with radius of 20 meters and their mission is to follow a straight-line trajectory to the opposite side of the circle. In Table 2 we can see the distance travelled of each drone and their flight time. We can see that there is a very small deviation of distance travelled to optimal distance which proves that the drones are making small changes in their paths to resolve conflicts.

Vehicle/Metric	Distance Travelled	Optimal Distance (in Meters)	Flight Time (in Seconds)
UAV 0	48.05	40	68.29
UAV 1	41.98	40	63.66
UAV 2	42.97	40	65.34
UAV 3	40.96	40	70.18
UAV 4	44.94	40	65.82
UAV 5	41.55	40	63.26
UAV 6	41.47	40	65.43
UAV 7	51.89	40	67.05

Table 2: Results for 8-way collision avoidance

In Fig.4. we can see the trajectory followed by the drones in this scenario, we can see that the Simple RVO algorithm made all the drones to avoid the scheduled collision in point (0,0) with the maneuvers that are shown for each drone.

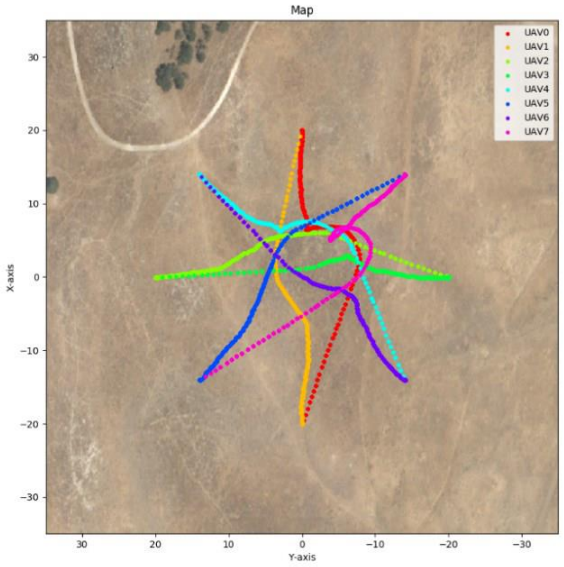

Figure 4: The deconfliction path of 8 UAVs

In Fig.5. we can see a histogram of thrust for all the drones participated in this scenario and we can observe that most of the observations the algorithm commands thrust in such a way that smooth trajectories can be achieved because the algorithm does not produce abrupt changes to drone's velocity.

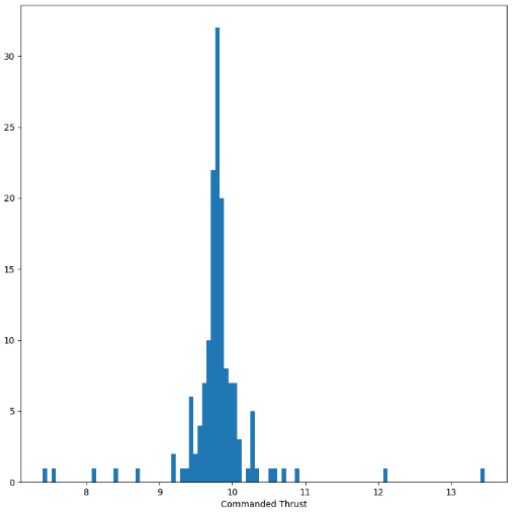

Figure 5: Commanded Thrust Histogram for 8 UAVs

In Fig.6. we can see a plot depicting the separation distance in red, and the minimum distance over time that the drones reached during their conflicts. We can observe that in all cases the minimum distance between two drones was above the desired threshold of 3 meters.

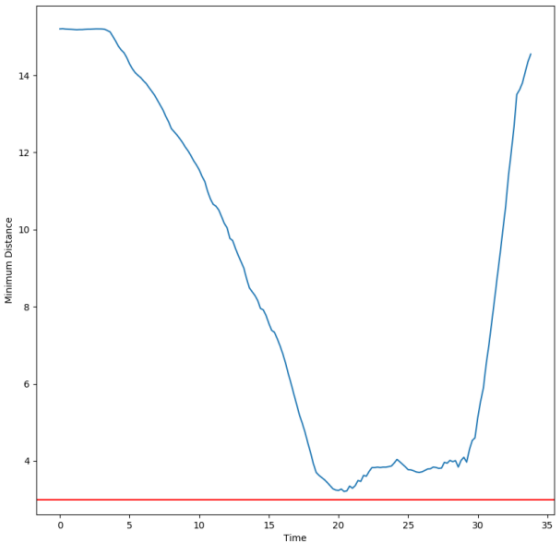

Figure 6: Minimum Distance over time for 8 UAVs

5.2 Scenario 2

In this scenario we have 5 drones executing multiple missions. The waypoints are constructed in such a way that creates conflicts with the other drones participating in that scenario. In Table 3 we can see the distance travelled of each drone and their flight time. We can see that there is a very small deviation of distance travelled to optimal distance which proves that the drones are making small changes in their paths to resolve conflicts.

Vehicle/Metric	Distance Travelled	Optimal Distance (in Meters)	Flight Time (in Seconds)
UAV 0	395.93	390.63	423
UAV 1	340.12	335.53	353
UAV 2	308	304.23	317
UAV 3	308.62	304.18	328.2
UAV 4	341.β5	335.43	362.2

Table 3: Results for 5-way continues collision avoidance

In Fig.7. we can see the trajectory followed by the drones in this scenario, we can see that the Simple RVO algorithm made all the drones to avoid the scheduled collision in point (0,0) with the maneuvers that are shown for each drone. This figure presents just one of the mission waypoints of the drone, and more specifically the first set of waypoints. We have chosen to present just one set to avoid congestion in the plot and which would make the plot unreadable.

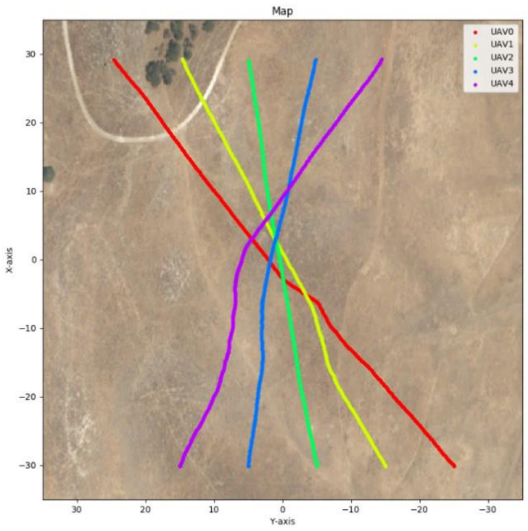

Figure 7: The deconfliction path of 5 UAVs

In Fig.8. we can see a histogram of thrust for all the drones participated in this scenario and we can observe that most of the observations the algorithm commands thrust in such a way that smooth trajectories can be achieved because the algorithm does not produce abrupt changes to drone's velocity.

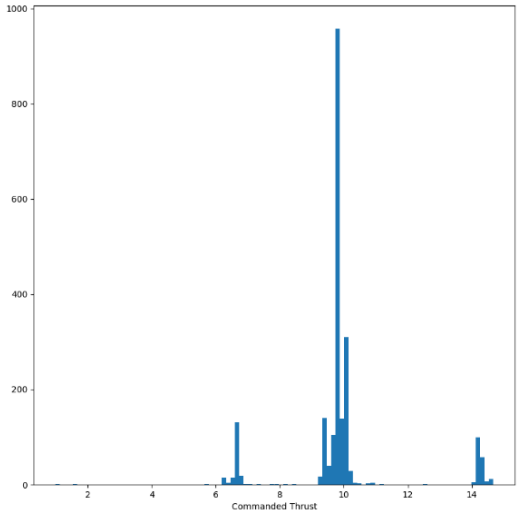

Figure 8: Commanded Thrust Histogram for 5 UAVs.

In Fig.9. we can see a plot depiction the separation distance in red, and the minimum distance over time that the drones reached during their conflicts. We can observe that in all cases the minimum distance between two drones was above the desired threshold of 3 meters.

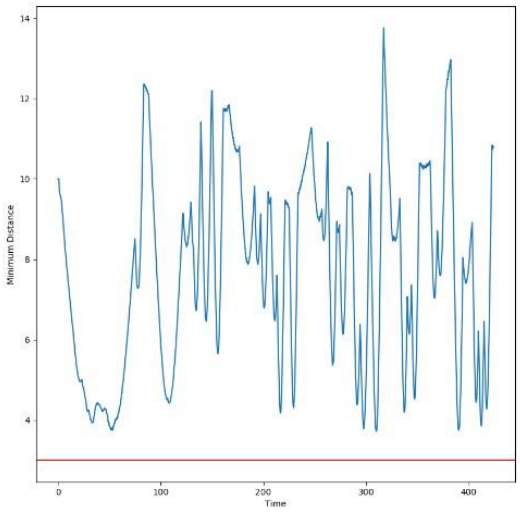

Figure 9: Minimum Distance over time for 5 UAVs.

6. Conclusions

Drones and unmanned aerial vehicles (UAVs) have aroused academics' interest over the last decade due to their capabilities and potential for use in a broad variety of applications and domains. To allow UAVs to achieve their tasks in that plethora of applications, their safety must be ensured. This frequently entails the ability to plan their journey and avoid accidents. It is vital that drones can perform collision avoidance techniques while flying autonomously and successfully.

In this paper, we presented a decentralized system of drones executing tasks in an area and a decentralized algorithm for ensuring safety during the flights by applying velocity obstacle principles. The proposed algorithm's results indicate that it is suitable for real time applications, and it can ensure safety and efficiency during multi drones deconfliction.

References

[1] Daramouskas, I. Perikos, I. Hatzilygeroudis, V. J. Lappas and V.Kostopoulos, "A Methodology For Drones to Learn How to Navigate And Avoid Obstacles Using Decision Trees," 2020 11th InternationalConference on Information, Intelligence, Systems and Applications (IISA, 2020, pp. 1-4, doi: 10.1109/IISA50023.2020.9284337.

[2] N. Patrinopoulou et al., "Metropolis II: Investigating the Future Shape of Air Traffic Control in Highly Dense Urban Airspace," 2022 30th Mediterranean Conference on Control and Automation (MED), 2022, pp. 649-655, doi: 10.1109/MED54222.2022.9837201.

[3] I. Daramouskas, N. Patrinopoulou, D. Meimetis, V. Lappas and V.Kostopoulos, "A design and simulation of a target detection, tracking and localisation system for UAVs," 2022 30th Mediterranean Conference on Control and Automation (MED), 2022, pp. 382-388, doi: 10.1109/MED54222.2022.9837230.

[4] I. Daramouskas, I. Perikos, & I. Hatzilygeroudis, A Method for Performing Efficient Real-Time Object Tracing for Drones. Advances in Smart Systems Research, 2017, 6(2), 55.

[5] Miura, A., Morikawa, H., Mizumachi, M.: Aircraft collision avoidance withpotential gradient—ground-based avoidance for horizontal maneuvers.Electronics and Communications in Japan Part Iii-fundamental Electronic Science 78, 104–114 (1995)

[6] Saha, S., Natraj, A., Waharte, S.: A real-time monocular vision-based frontal obstacle detection and avoidance for low cost uavs in gps denied environment. In: 2014 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology, pp. 189–195 (2014).

[7] Park, J.-W., Oh, H.-D., Tahk, M.-J.: Uav collision avoidance based on geometric approach. In: 2008 SICE Annual Conference, pp. 2122–2126 (2008).

[8] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an Open-source Robot Operating System. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2004)

[9] Koenig, N., Howard, A.: Design and Use Paradigms for Gazebo, an Open-source Multi-robot Simulator. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2004)

[10] van den Berg, J., Guy, S., Lin, M., Manocha, D.: Reciprocal n-Body Collision Avoidance, vol. 70, pp. 3–19 (2011)

[11] M. Guo and M. M. Zavlanos, "Multirobot Data Gathering Under Buffer Constraints and Intermittent Communication," in IEEE Transactions on Robotics, vol. 34, no. 4, pp. 1082-1097, Aug. 2018, doi: 10.1109/TRO.2018.2830370.

[12] P. Fiorini, Z. Shiller. Motion planning in dynamic environments using Velocity Obstacles. Int. Journal of Robotics Research 17(7), pp. 760-772, 1998