Decentralized Swarms Visibility Algorithms in 3D Urban Environments

Oren Gal and Yerach Doytsher
Mapping and Geo-information Engineering
Technion - Israel Institute of Technology
Haifa, Israel
e-mail: orengal@alumni.technion.ac.il

Abstract— **In this paper, we present a unique and efficient visible trajectory planning for aerial swarm using decentralized algorithms in a 3D urban environment. By using SwarmLab environment, we are comparing two decentralized algorithms from the state of the art for the navigation of aerial swarms, Olfati-Saber's and Vasarhelyi's. The first step in our concept is to extract basic geometric shapes. We focus on three basic geometric shapes from point clouds in urban scenes that can appear: planes, cylinders and spheres, extracting these geometric shapes using efficient Random Sample Consensus (RANSAC) algorithms with a high success rate of detection. The second step is a decentralized swarm algorithm for motion planning, demonstrated on drones in urban environment. Our planner includes dynamic and kinematic platform's limitation, generating visible trajectories based on our first step mentioned earlier. We demonstrate our visibility and trajectory planning method in simulations, showing trajectory planning in 3D urban environments for drone's swarm with decentralized algorithms demonstrating performance analysis ,such as order, safety, connectivity and union.**

Keywords-Swarm; Visibility; 3D; Urban environment; Decentralized algorithms.

I. INTRODUCTION

In this paper, we study a fast and efficient visible trajectory planning for drone swarms in a 3D urban environment, based on local point clouds data. Recently, urban scene modeling has become more and more precise, using Terrestrial/ground-based LiDAR on unmanned vehicles to generate point clouds data for modeling roads, signs, lamp posts, buildings, trees and cars. Visibility analysis in complex urban scenes is commonly treated as an approximated feature due to computational complexity.

Our trajectory planning method is based on a two-step visibility analysis in 3D urban environments using predicted visibility from point clouds data. The first step in our unique concept is to extract basic geometric shapes. We focus on three basic geometric shapes from point clouds in urban scenes: planes, cylinders and spheres, extracting these geometric shapes using efficient RANSAC algorithms with a high success rate of detection. The second step include decentralized swarm algorithm for motion planning, demonstrated on drones in urban environment. Our planner includes dynamic and kinematic platform's limitation, generating visible trajectories based on our first step mentioned above. We demonstrate our visibility and trajectory planning method in simulations, showing trajectory planning in 3D urban environments for drone's swarm with decentralized algorithms including performance analysis, such as order, safety, connectivity and union.

Visibility analysis based on this approximated scene prediction is done efficiently, based on our analytic solutions for visibility boundaries. With this capability, we present a local on-line planner generating visible trajectories, exploring the most visible and safe node in the next time step, using our predicted visibility analysis.

For the first time, we propose a solution for decentralized swarm algorithm which takes visibility into account, avoiding obstacles using Velocity Obstacle (VO) search and planning method.

The rest of this paper is organized as follows: In Section II we introduce visibility analysis from point clouds data. In Section III, we introduce fast visibility analysis, and in Section VI we present decentralized swarm algorithm.

II. VISIBILITY ANALYSIS FROM POINT CLOUDS DATA

As mentioned, visibility analysis in complex urban scenes is commonly treated as an approximated feature due to its computational complexity. Recently, urban scene modeling has become more and more exact, using Terrestrial/ground-based LiDAR generating dense point clouds data for modeling roads, signs, lamp posts, buildings, trees and cars. Automatic algorithms detecting basic shapes and their extraction have been studied extensively, and are still a very active research field [2].

In this part, we present a unique concept for predicted and approximated visibility analysis in the next attainable vehicle's state at a one-time step ahead in time, based on local point clouds data which is a partial data set.

We focus on three basic geometric shapes in urban scenes: planes, cylinders and spheres, which are very common and can be used for the majority of urban entities in modeling scenarios. Based on point clouds data generated from the current vehicle's position in state k-1, we extract these geometric shapes using efficient RANSAC algorithms [3] with high success rate detection tested in real point cloud data.

After extraction of these basic geometric shapes from local point clouds data, our unified concept, and our main contribution, focus on the ability to predict and approximate urban scene modeling at the next view point V_k, i.e., at the attainable location of the vehicle in the next time step. Scene prediction is based on the geometric entities and the KF), which is commonly used in dynamic systems for tracking target systems [4][5]. We formulate the geometric shapes as states vectors in a dynamic system and predict the scene structure in the next time step, k.

Based on the predicted scene in the next time step, visibility analysis is carried out from the next view point model [6], which is, of course, an approximated one. As the vehicle reaches the next viewpoint V_k, point clouds data are measured and scene modeling and states vectors are updated, which is an essential for the global swarm visible trajectory planning based on state-of-the-art decentralized algorithms.

A. *Shapes Extraction*

1) **Geometric Shapes:**

The urban scene is a very complex one in the matter of modeling applications using LiDAR, and the generated point clouds are very dense. Despite these inherent complications, feature extraction can be made very efficient by using basic geometric shapes. We define three kinds of geometric shapes: planes, cylinders and spheres, with a minimal number of parameters for efficient time computation.

Plane: center point (x,y,z) and unit direction vector from center point.

Cylinder: center point (x,y,z), radius and unit direction vector of the cylinder axis. Cylinder height dimension will be consider later on as part of the simulation.

Sphere: center point (x,y,z), radius and unit direction vector from center point.

2) **RANSAC:**

The RANSAC [7] is a well-known paradigm, extracting shapes from point clouds using a minimal set of a shape's primitives generated by random drawing in a point clouds set. A minimal set is defined as the smallest number of points required to uniquely define a given type of geometric primitive.

For each of the geometric shapes, points are tested to approximate the primitive of the shape (also known as "score of the shape"). At the end of this iterative process, extracted shapes are generated from the current point clouds data.

Based on the RANSAC concept, the geometric shapes detailed above can be extracted from a given point clouds data set. In order to improve the extraction process and reduce the number of points validating shape detection, we compute the approximated surface normal for each point and test the relevant shapes.

Given a point-clouds $P=\{p_1..p_N\}$ with associated normals $\{n_1..n_N\}$, the output of the RANSAC algorithm is a set of primitive shapes $\{\delta_1..\delta_N\}$ and a set of remaining points $R=P\setminus\{p_{\delta_1}..p_{\delta_N}\}$.

B. *Predicted Scene – Kalman Filter*

In this part, we present the global KF approach for our discrete dynamic system at the estimated state, *k*, based on the defined geometric shapes formulation defined in the previous sub-section.

Generally, the Kalman Filter can be described as a filter that consists of three major stages: Predict, Measure, and Update the state vector. The state vector contains different state parameters, and provides an optimal solution for the whole dynamic system [5]. We model our system as a linear one with discrete dynamic model, as described in (1):

$$x_k = F_{k,k-1}x_{k-1} \tag{1}$$

where x is the state vector, F is the transition matrix and *k* is the state.

The state parameters for all of the geometric shapes are defined with shape center $\vec{s}$, and unit direction vector $\vec{d}$, of the geometric shape, from the current time step and viewpoint to the predicted one.

In each of the current states *k*, geometric shape center $\vec{s_k}$, is estimated based on the previous update of shape center location $\vec{s}_{k-1}$, and the previous updated unit direction vector $\vec{d}_{k-1}$, multiplied by small arbitrary scalar factor *c*, described in (2):

$$\vec{s}_k = \vec{s}_{k-1} + c\vec{d}_{k-1} \tag{2}$$

Direction vector $\vec{d}_k$ can be efficiently estimated by extracting the rotation matrix T, between the last two states *k, k-1*. In case of an inertial system fixed on the vehicle, a rotation matrix can be simply found from the last two states of the vehicle translations in (3):

$$\vec{d}_k = T\vec{d}_{k-1} \tag{3}$$

The 3D rotation matrix T tracks the continuous extracted plans and surfaces to the next viewpoint V_k, making it possible to predict a scene model where one or more of the geometric shapes are cut from current point clouds data in state *k-1*. The discrete dynamic system can be written as formulated in (4):

$$\begin{bmatrix} \vec{s}_{x_k} \\ \vec{s}_{y_k} \\ \vec{s}_{z_k} \\ \vec{d}_{x_k} \\ \vec{d}_{y_k} \\ \vec{d}_{z_k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & c & 0 & 0 \\ 0 & 1 & 0 & 0 & c & 0 \\ 0 & 0 & 1 & 0 & 0 & c \\ 0 & 0 & 0 & T_{11} & T_{12} & T_{13} \\ 0 & 0 & 0 & T_{21} & T_{22} & T_{23} \\ 0 & 0 & 0 & T_{31} & T_{32} & T_{33} \end{bmatrix} \begin{bmatrix} \vec{s}_{x_{k-1}} \\ \vec{s}_{y_{k-1}} \\ \vec{s}_{z_{k-1}} \\ \vec{d}_{x_{k-1}} \\ \vec{d}_{y_{k-1}} \\ \vec{d}_{z_{k-1}} \end{bmatrix} \tag{4}$$

where the state vector x is 6×1 vector, and the transition squared matrix is $F_{k,k-1}$. The dynamic system can be extended to additional state variables representing someof the geometric shape parameters such as radius, length etc. We define the dynamic system as the basic one for generic shapes that can be simply modeled with center and direction vector. Sphere radius and cylinder Z boundaries are defined in an additional data structure of the scene entities.

III. Fast and Approximated Visibility Analysis

In this section, we present an analytic analysis of the visibility boundaries of planes, cylinders and spheres for the predicted scene presented in the previous sub-section, which leads to an approximated visibility. For the plane surface, fast and efficient visibility analysis was already presented in [6]. In this part, we extend the previous visibility analysis concept [6] and include cylinders as continuous curves parameterization $C_{c\ln d}(x,y,z)$.

Cylinder parameterization can be described in (5):

$$C_{C\ln d}(x,y,z)=\begin{pmatrix} r\sin(\theta)\\ r\cos(\theta)\\ c\end{pmatrix}_{r=const},\quad \begin{matrix} 0\le\theta\le 2\pi\\ c=c+1\\ 0\le c\le h_{peds\_\max}\end{matrix} \tag{5}$$

We define the visibility problem in a 3D environment for more complex objects as:

$$C'(x,y)_{z_{const}}\times(C(x,y)_{z_{const}}-V(x_0,y_0,z_0))=0 \tag{6}$$

where 3D model parameterization is $C(x,y)_{z=const}$, and the viewpoint is given as $V(x_0,y_0,z_0)$. Extending the 3D cubic parameterization, we also consider the case of the cylinder. Integrating (5) to (6) yields:

$$\begin{pmatrix} r\cos\theta\\ -r\sin\theta\\ 0\end{pmatrix}\times\begin{pmatrix} r\sin\theta-V_x\\ r\cos\theta-V_y\\ c-V_z\end{pmatrix}=0 \tag{7}$$

$$\theta=\arctan\left(-\frac{-r-\frac{\left(-vy\,r+\sqrt{vx^4-vx^2\,r^2+vy^2\,vx^2}\right)vy}{vx^2+vy^2}}{vx},\ -\frac{-vy\,r+\sqrt{vx^4-vx^2\,r^2+vy^2\,vx^2}}{vx^2+vy^2}\right) \tag{8}$$

As can be noted, these equations are not related to Z axis, and the visibility boundary points are the same for each *x-y* cylinder profile, as seen in (7), (8).   .

The visibility statement leads to a complex equation, which does not appear to be a simple computational task. This equation can be efficiently solved by finding where the equation changes its sign and crosses the zero value; we used analytic solution to speed up computation time and to avoid numeric approximations. We generate two values of θ generating two silhouette points in a very short time computation. Based on an analytic solution to the cylinder case, a fast and exact analytic solution can be found for the visibility problem from a viewpoint.

We define the solution presented in (8) as x-y-z coordinates values for the cylinder case as Cylinder Boundary Points (CBP). CBP, defined in (9), are the set of visible silhouette points for a 3D cylinder, as presented in Figure 1:

$$CBP_{i=1..N_{PBP\_bound}=2}(x_0,y_0,z_0)=\begin{bmatrix} x_1,y_1,z_1\\ x_{N_{PBP\_bound}},y_{N_{PBP\_bound}},z_{N_{PBP\_bound}}\end{bmatrix} \tag{9}$$

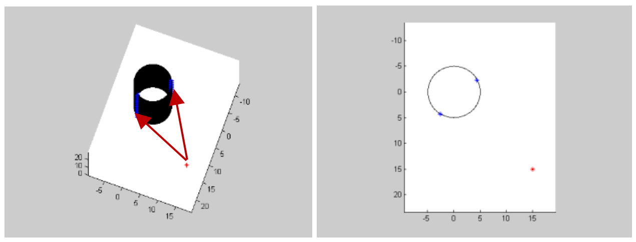

(a) (b)

Figure 1. Cylinder Boundary Points (CBP) using Analytic Solution marked as blue points, Viewpoint Marked in Red: (a) 3D View (Visible Boundaries Marked with Red Arrows); (b) Topside View.

In the same way, sphere parameterization can be described as formulated in (10):

$$C_{Sphere}(x,y,z)=\begin{pmatrix} r\sin\phi\cos\theta\\ r\sin\phi\sin\theta\\ r\cos\phi\end{pmatrix}_{r=const}\qquad \begin{matrix} 0\le\phi<\pi\\ 0\le\theta<2\pi\end{matrix} \tag{10}$$

We define the visibility problem in a 3D environment for this object in (11):

$$C'(x,y,z)\ \times(C(x,y,z)\ -V(x_0,y_0,z_0))=0 \tag{11}$$

where the 3D model parameterization is $C(x,y,z)$, and the viewpoint is given as $V(x_0,y_0,z_0)$. Integrating (10) to (11) yields:

$$\theta=\arctan\left(\frac{r\sin(\phi)}{v\_y}-\frac{1}{v\_y\left(v\_y^2+v\_x^2\right)}\left(v\_x\left(r\sin(\phi)\,v\_x-\sqrt{-v\_y^2\,r^2\sin(\phi)^2+v\_y^4+v\_x^2\,v\_y^2}\right)\right),\ \frac{r\sin(\phi)\,v\_x-\sqrt{-v\_y^2\,r^2\sin(\phi)^2+v\_y^4+v\_x^2\,v\_y^2}}{v\_y^2+v\_x^2}\right) \tag{12}$$

Where *r* is defined from sphere parameter, and $V(x_0, y_0, z_0)$ are changes from visibility point along Z axis, as described in (12). The visibility boundary points for a sphere, together with the analytic solutions for planes and cylinders, allow us to compute fast and efficient visibility in a predicted scene from local point cloud data, which are updated in the next state.

This extended visibility analysis concept, integrated with a well-known predicted filter and extraction method, can be implemented in real time applications with point clouds data.

IV. DECENTRALIZED SWARMS TRAJECTORY PLANNING

In this part, we focus on decentralized swarm algorithm with visibility analysis in urban environment as cost function for our trajectory.

For our simulation, we used SwarmLab [10], drone swarm simulator that was implemented and adapted two representative algorithms belonging to the category of decentralized swarming. Decentralized approach can make the system easily scalable and robust to the failures of a single individual. SwarmLab includes algorithm developed by Olfati-Saber [12], who proposes a formal theoretical framework for the design and analysis of swarm algorithms based on potential fields and graph theory.

The second algorithm that was implemented is an adaptation of the recent Vasarhelyi's algorithm [13], defined by the following rules: repulsion to avoid inter-agent collisions, velocity alignment to steer the agents to an average direction, and self-propulsion to match a preferred speed value. In addition, the algorithm includes friction forces that reduce oscillations and ease the implementation on real robots.

In decentralized approaches, one agent's movement is only influenced by local information coming from its neighbors. Neighbors' selection can be operated according to different metrics.

In our work, we adopted these algorithms with visibility analysis as part of swarm's trajectory by leading the swarm to the most visible areas in the scene by the swarm, as presented in the previous section.

Unlike the original SwarmLab simulation where obstacle avoidance is based on simulating the obstacles as virtual agents, we used the Velocity Obstacles [8] local obstacles avoidance method.

This obstacle avoidance method allows us to deal better with swarm behavior and can be more precise and gentler, avoiding obstacles in dense environments.

A. *The Planner*

As mentioned above, our planner is based on an iterative local planning method, where the swarm is moving to the most visible area. By using RANSAC algorithm, point clouds data are extracted each time step into three possible objects: plane, cylinder and sphere. The scene is formulated as a dynamic system using KF analysis for objects' prediction. The objects are approximated for the next time step, and each safe attainable state that can be explored is set as candidate viewpoint. The cost for each agent is set as total visible surfaces, based on the analytic visibility boundary, where the optimal and safe node is explored for the next time step.

At each time step, the planner computes the next Attainable Velocities (AV). The safe nodes not colliding with objects such as cubes, cylinders and spheres, i.e., nodes outside VO, are explored. Where all nodes are inside VO, a unified analytic solution for time horizon is presented, generating an escape option for these radical cases without affecting visibility analysis. The planner computes the cost for these safe nodes based on predicted visibility and chooses the node with the optimal cost for the next time step. We repeat this procedure while generating the most visible trajectory.

1) *Visibility Analysis as Swarm Cost Function*

Our swarm direction and movement is guided by minimum invisible parts from viewpoint V to the approximated 3D urban environment model in the next time step, $t + \Delta t$, set by KF after extracting objects from point clouds data using the RANSAC algorithm. The cost function next state is a combination of IRV and ISV, with different weights as functions of the required task.

The cost function presented in (13) is computed for each agent from his current state, considering the agent's future location at the next time step $(x_1(t+\Delta t), x_2(t+\Delta t))$ as viewpoint:

$$w\big(x(t+\Delta t)\big) = \alpha \cdot ISV(x(t+\Delta t)) + \beta \cdot IRV(x(t+\Delta t)) \quad (13)$$

where $\propto, \beta$ are coefficients affecting the trajectory's character, as shown in (14). The cost function $w(x(t+\Delta t)$ produces the total sum of invisible parts from the viewpoint to the 3D urban environment.

We divide point invisibility value into Invisible Surfaces Value (ISV) and Invisible Roofs Value (IRV). This classification allows us to plan delicate and accurate trajectories upon demand. We define ISV and IRS as the total sum of the invisible roofs and surfaces (respectively). Invisible Surfaces Value (ISV) of a viewpoint is defined as the total sum of the invisible surfaces of all the objects in a 3D environment, as described in (14):

$$ISV(x_0, y_0, z_0) = \sum_{i=1}^{N_{obj}} IS_{VP_i^{j=1..N_{bound}-1}}^{VP_i^{j=1..N_{bound}-1}} \quad (14)$$

In the same way, we define Invisible Roofs Value (IRV) as the total sum of all the invisible roofs' surfaces, as described in (15):

$$IRV(x_0, y_0, z_0) = \sum_{i=1}^{N_{obj}} IS_{VP_i^{j=N_{bound}}}^{VP_i^{j=N_{bound}}} \quad (15)$$

Extended analysis of the analytic solution for visibility analysis for known 3D urban environments can be found in [6].

V. SIMULATIONS

We implemented the presented algorithm and tested some urban environments on a Intel(R) Core (TM) i5-10210U CPU 2.11 GHz with Matlab. We computed the visible trajectories using our planner, simulating cloud points using Matlab functions.

On the first part, we tested our visibility analysis integrated into decentralized drones swarm algorithms as described above. The workflow of a swarm simulation is summarized in Figure 2, were typical scenario of cylinder objects in our environment can be seen in Figure 3.

In the first case, we tested our algorithm with a relatively large number of agents. As can be seen in Figure 4, thirty agents in the swarm moving forward in straight line, presenting swarm trajectory, distance between the agents during mission, speed and accelerations during movement. The swarm navigates based on modified Olfati-Saber's algorithm where obstacle avoidance implemented by Velocity Obstacles, where agents are simulated by point mass model. Swarm cost function is based on visibility analysis computed each time step as mentioned in the previous section.

In the second case, we tested our algorithm with ten agents in the swarm, so each agent simulated with quadrotor dynamic model. As can be seen in Figure 5, ten agents in the swarm moving forward in straight line with C. Vasarhelyi's etc. algorithm, but visibility analysis and dynamic constraints swift the swarm to the right side. presenting swarm trajectory, distance between the agents. Figure 5 also includes speed and accelerations during movement, performances analysis and total distance to the obstacles during mission.

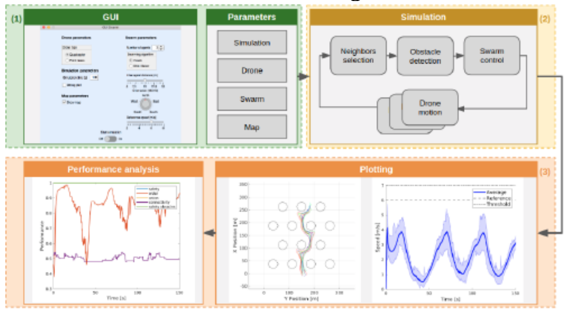


Figure 2. SwarmLab simulation workflow. From the top left, in clockwise order: (1) in the GUI, the user sets the parameters related to the simulation, drone typology, swarm algorithm and environment. (2) the main simulation loop computes control commands for the drones, based on the information of the map and neighboring drones; (3) both real-time and post-simulation (Source [10]).

Order metric captures the correlation of the agents movements and gives an indication about how ordered the flock. Safety metrics, measure the risk of collisions among the swarm agents or between agents and obstacles.

Union metric counts the number of independent subgroups that originates during the simulation.

Connectivity metric defined from the algebraic connectivity of the sensing graph that underlines the considered swarm configuration. Detailed mathematical definitions of these performances' parameters can be found in [11].

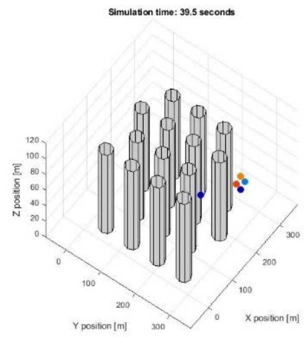
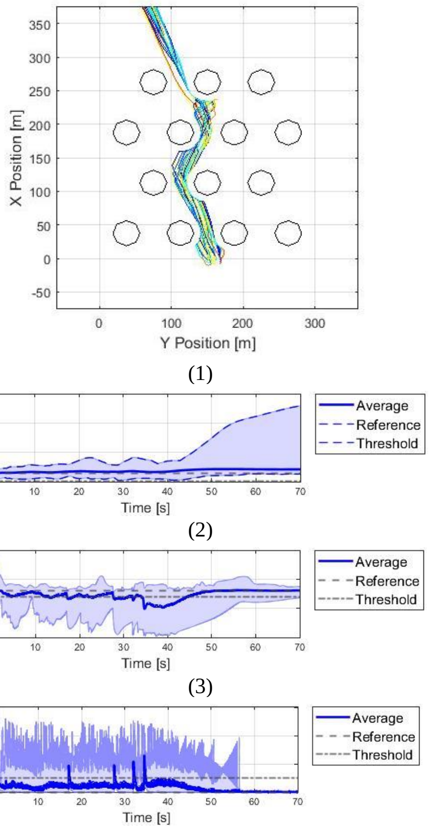
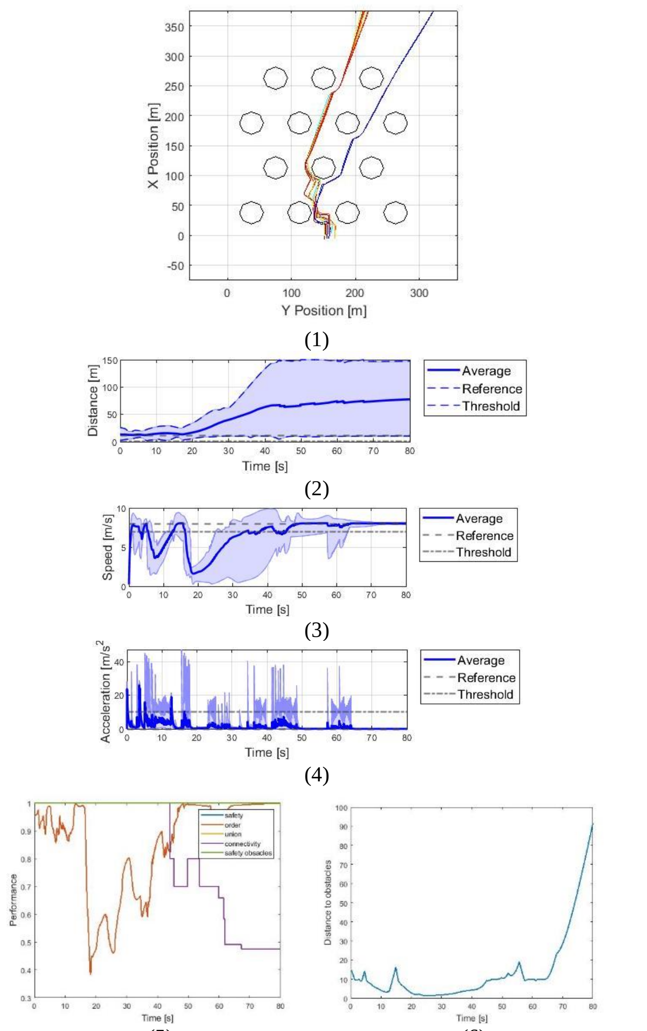


Figure 3. Typical Scenario of Environmmet Obstacles Simulation



(4)

Figure 4. Thirty agents swarm moving forward in straight line using Olfati-Saber's algorithm with visibility analsysis; (1) presenting swarm trajectory; (2) distance between the agents during mission; (3) speed and (4) accelerations during movement.



(5) (6)

Figure 5. Ten agents swarm moving forward in straight line using Vasarhelyi's algorithm with visibility analsysis, with quadrotor synamic model for agent; (1) presenting swarm trajectory; (2) distance between the agents during mission; (3) speed and (4) accelerations during movement; (5) performances analysis; (6) total distance to the obstacles during mission.

VI. CONCLUSION AND FUTURE WORK

In this research, we have presented an efficient swarm trajectory planning algorithm for visible trajectories in a 3D urban environment.

We extend our analytic visibility analysis method to cylinders and spheres, which allows us to efficiently set the visibility boundary of predicted objects in the next time step. Based on these fast computation capabilities, the on-line planner can approximate the most visible state as part of a decentralized swarm algorithm.

By using SwarmLab environment, we compare two decentralized algorithms from the state of the art for the navigation of aerial swarms, Olfati-Saber's and Vasarhelyi's.

Our planner includes dynamic and kinematic platform's limitation, generating visible trajectories based on our first step mentioned earlier.

We demonstrate our visibility and trajectory planning method in simulations, showing trajectory planning in 3D urban environments for drone's swarm with decentralized algorithms with performance analysis, such as order, safety, connectivity and union.

Further research will focus on advanced geometric shapes, which will allow precise urban environment modeling, facing real-time implementation with on-line data processing from sensors.

REFERENCES

[1] G.Pandey, J.R. McBride, R.M. Eustice, "Ford campus vision and lidar data set." International Journal of Robotics Research, 30(13), pp. 1543-1552, November 2011.

[2] G. Vosselman, B. Gorte, G. Sithole, T. Rabbani. "Recognizing structure in laser scanner point clouds.", The International Archives of the Photogrammetry Remote Sensing and Spatial Information Sciences (IAPRS), 2004, vol. 36, pp. 33–38.

[3] R. Schnabel, R. Wahl, R. Klein, "Efficient RANSAC for Point-Cloud Shape Detection," Computer Graphics Forum, 2007, vol. 26, no.2, pp. 214-226.

[4] R. Kalman. "A new approach to linear filtering and prediction problems.", Transactions of the ASME-Journal of Basic Engineering, 1960, vol. 82, no. 1, pp:35–45.

[5] J. Lee, M. Kim, I. Kweon. "A kalman filter based visual tracking algorithm for an object moving," In IEEE/RSJ Intelligent Robots and Systems, 1995, pp. 342–347.

[6] O. Gal, and Y. Doytsher, "Fast Visibility Analysis in 3D Procedural Modeling Environments," in Proc. of the, 3rd International Conference on Computing for Geospatial Research and Applications, Washington DC, USA, 2012.

[7] H. Boulaassal, T. Landes, P. Grussenmeyer, F. Tarsha- Kurdi. "Automatic segmentation of building facades using terrestrial laser data", The International Archives of the Photogrammetry Remote Sensing and Spatial Information Sciences (IAPRS), 2007, vol. 36, no. 3.

[8] O. Gal, Z. Shiller, E. Rimon, "Efficient and safe on-line motion planning in dynamic environment," in Proceedings of the IEEE International Conference on Robotics and Automation, 2009, pp. 88–93.

[9] Velodyne 2007: Velodyne HDL-64E: A high definition LIDAR sensor for 3D applications. Available at: http://www.velodyne.com/lidar/products/white paper. [Accessed 1/23/2017].

[10] E. Soria, F. Schiano and D. Floreano, "SwarmLab: a Matlab Drone Swarm Simulator," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 8005-8011, doi: 10.1109/IROS45743.2020.9340854.

[11] E. Soria, F. Schiano, and D. Floreano, "The influence of limited visual sensing on the Reynolds flocking algorithm," IEEE Third International Conference on Robotic Computing (IRC), 2019.

[12] R. Olfati-Saber, "Flocking for Multi-Agent Dynamic Systems: Algo-rithms and Theory," IEEE Transactions on Automatic Control, vol. 51, no. 3, pp. 401–420, 2006.

[13] G. V´as´arhelyi, C. Vir´agh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Optimized flocking of autonomous drones in confined environments," Science Robotics, vol. 3, no. 20, 2018.