Deep Reinforcement Learning for Spatial Motion Planning in 3D Urban Environments

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

Abstract—**In this paper, we present spatial motion planner in 3D environments based on Deep Reinforcement Learning (DRL) algorithms. We tackle 3D motion planning problem by using Deep Reinforcement Learning (DRL) approach, which learns agent's and environment constraints. Spatial analysis focuses on visibility analysis in 3D setting an optimal motion primitive considering agent's dynamic model based on fast and exact visibility analysis for each motion primitives. Based on optimized reward function, which consist of generated 3D visibility analysis and obstacle avoidance trajectories, we introduce DRL formulation, which learns the value function of the planner and generates an optimal spatial visibility trajectory. We demonstrate our planner in simulations for Unmanned Aerial Vehicles (UAV) in 3D urban environments. Our spatial analysis is based on a fast and exact spatial visibility analysis of the 3D visibility problem from a viewpoint in 3D urban environments. We present DRL architecture generating the most visible trajectory in a known 3D urban environment model, as time-optimal one with obstacle avoidance capability.**

Keywords - Deep Reinforcement Learning; Visibility; 3D; Spatial analysis; Motion Planning.

I. Introduction and Related Work

Spatial clustering in urban environments is a new spatial field from trajectory planning aspects [1]. The motion and trajectory planning fields have been extensively studied over the last two decades [2][4][6]. The main effort has focused on finding a collision-free path in static or dynamic environments, i.e., in moving or static obstacles, using roadmap, cell decomposition, and potential field methods [11].

The path-planning problem becomes an NP-hard one, even for simple cases such as time-optimal trajectories for a system with point-mass dynamics and bounded velocity and acceleration with polyhedral obstacles [7].

Path planning algorithms can be distinguished as local and global planners. The local planner generates one, or a few, steps at every time step, whereas the global planner uses a global search to the goal over a time-spanned tree. Examples of local (reactive) planners are [9][14]. These planners are too slow, do not guarantee safety and neglect spatial aspects.

Efficient solutions for an approximated problem were investigated by LaValle and Kuffner, addressing non-holonomic constraints by using the Rapidly Random Trees (RRT) method [15][16]. Over the years, many other semi-randomized methods were proposed, using evolutionary programming [5][18].

The randomized sampling algorithms planner, such as RRT, explores the action space stochastically. The RRT algorithm is probabilistically complete, but not asymptotically optimal [13]. The RRT* planner challenges optimality by a rewiring process each time a node is added to the tree. However, in cluttered environments, RRT* may behave poorly since it spends too much time deciding whether to rewire or not.

Overall, only a few works have focused on spatial analysis characters integrated into trajectory planning methods such as visibility analysis or spatial clustering methods [11].

Analyzing pedestrian's mobility from a spatial point of view mainly focused on route choice [3], simulation model [19] and agent-based modeling [12].

The efficient computation of visible surfaces and volumes in 3D environments is not a trivial task. The visibility problem has been extensively studied over the last twenty years, due to the importance of visibility in GIS and Geomatics, computer graphics and computer vision, and robotics. Accurate visibility computation in 3D environments is a very complicated task demanding a high computational effort, which could hardly have been done in a very short time using traditional well-known visibility methods.

The exact visibility methods are highly complex, and cannot be used for fast applications due to their long computation time. Previous research in visibility computation has been devoted to open environments using Digital Elevation Model (DEM), representing raster data in 2.5D (Polyhedral model), and do not address, or suggest solutions for dense built-up areas.

Most of these works have focused on approximate visibility computation, enabling fast results using interpolations of visibility values between points, calculating point visibility with the Line of Sight (LOS) method [7]. Lately, fast and accurate visibility analysis computation in 3D environments has been presented [10].

In this paper, we present unique spatial trajectory planning method based on DRL algorithm based on exact visibility analysis in urban environment. The generated trajectories are based on visibility motion primitives as part of the planned trajectory, which takes into account exact 3D visible volumes analysis clustering in urban environments.

The proposed planner includes obstacle avoidance capabilities, satisfying dynamics' and kinematics' agent model constraints in 3D environments, using Velocity Obstacles (VO) in 3D for Unmanned Aerial Vehicle (UAV) model.

In the following sections, we first introduce the DRL algorithm and method and our extension for a spatial analysis case, such as 3D visibility. Later on, we present the our planner, using VO method and planner model. In the last part of the paper, with planner simulation using DRL method.

II. Problem Statement

We consider the basic visibility problem in a 3D urban environment, consisting of 3D buildings modeled as 3D cubic parameterization $\sum_{i=1}^{N} C_i(x, y, z =_{h_{\min}}^{h_{\max}})$, and viewpoint $V(x_0, y_0, z_0)$.

Given:

- Parameterizations of N objects $\sum_{i=1}^{N} C_i(x, y, z =_{h_{\min}}^{h_{\max}})$ describing a 3D urban environment model

Compute:

- *Trajectory*, which consist of optimal set of all visible points, i.e., most visible points of $\sum_{i=1}^{N} C_i(x, y, z =_{h_{\min}}^{h_{\max}})$, from starting point ,$q_s$, to the goal, q_g, without collision.

This problem seems to be solved by conventional geometric methods, but as mentioned before, it demands a long computation time. We introduce a fast and efficient computation solution for a schematic structure of an urban environment that demonstrates our method based on Deep Reinforcement Learning method (DRL).

On the first part, we present the DRL algorithm, formulated to our planning problem, and the visibility analysis along with obstacles avoidance planner.

III. Deep Reinforcement Learning (DRL) Algorithm

In most Deep Reinforcement Learning (DRL) systems, the state is basically agent's observation of the environment. At any given state the agent chooses its action according to a policy. Hence, a policy is a road map for the agent, which determines the action to take at each state. Once the agent takes an action, the environment returns the new state and the immediate reward. Then, the agent uses this information, together with the discount factor to update its internal understanding of the environment, which, in our case, is accomplished by updating a value function. Most methods are using the use well-known simple and efficient greedy exploration method maximizing Q-value.

In case of velocity planning space as part of spatial analysis planning, each possible action is a possible velocity in the next time step, which also represent a viewpoint. The Q-value function is based on greedy search velocity, with greedy local search method. Based on that, TD and SARSA methods for DRL can be used, generating visible trajectory in 3D urban environment.

A. Markov Decision Processes (MDP)

The standard Reinforcement Learning set-up can be described as a MDP as can be seen in Figure 1, consisting of:

- **A finite set of states** S, comprising all possible representations of the environment.
- **A finite set of actions** A, containing all possible actions available to the agent at any given time.
- **A reward function** $R = \psi(s_t, a_t, s_{t+1})$, determining the immediate reward of performing an action at from a state s_t, resulting in s_{t+1}.
- **A transition model** $T(s_t, a_t, s_{t+1}) = p(s_{t+1} | s_t, a_t)$, describing the probability of transition between states s_t and $s_{t+1}$when performing an action a_t.

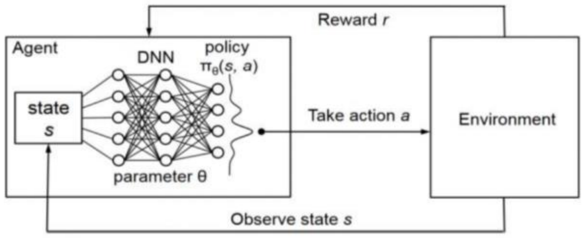


Figure 1. Standard Reinforcement Learning Methology

B. Temporal Difference Learning

Temporal-difference learning (or TD) interpolates ideas from Dynamic Programming (DP) and Monte Carlo methods. TD algorithms can learn directly from raw experiences without any model of the environment.

Whether in Monte Carlo methods, an episode needs to reach completion to update a value function, Temporal-difference learning can learn (update) the value function within each experience (or step). The price paid for being able to regularly change the value function is the need to update estimations based on other learned estimations (recalling DP ideas). Whereas in DP a model of the environment's dynamic is needed, both Monte Carlo and TD approaches are more suitable for uncertain and unpredictable tasks.

Since TD learns from every transition (state, reward, action, next state, next reward) there is no need to ignore/discount some episodes as in Monte Carlo algorithms.

C. Spatial Planning Using DRL

In this section, we present DRL approach based on the proposed spatial planning method. It considers that the value function *f* related to each point *x*. The spatial planner seeks to obtain the trajectory *T** that based on visibility motion primitives set as part of the planned trajectory, which takes into account exact 3D visible volumes analysis clustering in urban environments, based on optimizing value function *f* along *T*.

The generated trajectories are then represented by a set of discrete configuration points:

$$T = \{x_1, x_2, \cdots, x_N\} \tag{1}$$

Without loss of generality, we can assume that the value function for each point can be expressed as a linear combination of a set of sub-value functions, that will be called features $c(x) = \sum c_j f_j(x)$. The cost of path *T* is then the sum of the cost for all points in the path. Particularly, in the Velocity Obstacles as will be presented later on, the value is the sum of the sub-values of moving between pairs of states in the path:

$$c(\zeta) = \sum_{i=1}^{N-1} c(x_i, x_{i+1}) = \sum_{i=1}^{N-1} \frac{c(x_i) + c(x_{i+1})}{2} \|x_{i+1} - x_i\|$$
$$= \omega^T \sum_{i=1}^{N-1} \frac{f(x_i) + f(x_{i+1})}{2} \|x_{i+1} - x_i\| = \omega^T f(\zeta) \tag{2}$$

Based on number of demonstration trajectories D, D = $\{\zeta_1, \zeta_2, \cdots, \zeta_D\}$, by using DRL, weights ω can be set for learning from demonstrations and setting similar planning behavior. As was shown by [23,24], this similarity is achieved when the expected value of the features for the trajectories generated by the planner is the same as the expected value of the features for the given demonstrated trajectories:

$$\mathbb{E}(f(\zeta)) = \frac{1}{D} \sum_{i=1}^{D} f(\zeta_i) \tag{3}$$

Applying the Maximum Entropy Principle [25] to the DRL problem leads to the following form for the probability density for the trajectories returned by the demonstrator:

$$p(\zeta|\omega) = \frac{1}{Z(\omega)} e^{-\omega^T f(\zeta)} \tag{4}$$

Z(ω) is a normalization function that does not depend on ζ. One way to determine ω is maximizing the log-likelihood of the demonstrated trajectories under the previous model:

$$\mathrm{L}(\mathrm{D}|\omega) = -D\log(Z(\omega)) \quad + \sum_{i=1}^{D} (-w^T f(\zeta_i)) \tag{5}$$

The gradient of the previous log-likelihood with respect to *ω* is given by:

$$\nabla\mathcal{L} = \frac{\partial \mathcal{L}(\mathcal{D}|\omega)}{\partial \omega} = \mathbb{E}(f(\zeta)) - \frac{1}{D} \sum_{i=1}^{D} f(\zeta_i) \tag{6}$$

As mentioned in [23], this gradient can be intuitively explained. If the value of one of the features for the trajectories returned by the planner are higher from the value in the demonstrated trajectories, the corresponding weight should be increased to increase the value of those trajectories.

The main problem with the computation of the previous gradient is that it requires to compute the expected value of the features E(*f*(*ζ*)) for the generative distribution (4).

We suggest setting large amount of D cased, setting the relative *w* values for our planner characters.

TABLE I. DRL PLANNER PSEUDO CODE

```
DRL Planner
Setting Trajectory S Examples D, D= T*.init ($x_{init}$);
Calculate function features Weight, w
$f_D \leftarrow$ AverageFeatureCount(D);
$w \leftarrow$ random_init();
Repeat
        for each T* do
    for VelocityObstacles_repetitions do
            $\zeta_i \leftarrow$ getVOstarPath(T*,ω)
            $f(\zeta_i) \leftarrow$ calculeFeatureCounts($\zeta_i$)
        end for
        $f_{VO}(T^*) \leftarrow \sum_{i=1}^{VO\_repetitions} f(\zeta_i)$)/VO_repetitions
    end for
    $f_{VO} \leftarrow (\sum_{i=1}^{S} f_{VO})/s$
    $\nabla L \leftarrow f_{VO} - f_D$
    $w \leftarrow$ UpdatedWeigths ($\nabla L$)
Until convergence
Return w
```

IV. UAV MODEL

We introduce an Unmanned Aerial Vehicle (UAV) model, based on the well-known simple car and Dubins airplane [26]. Dubins airplane [27] model extends Dubins car model with continuous change of altitude without reverse gear, avoiding sudden altitude speed rate variation. Our UAV model includes kinematic and dynamic constraints which ignore pitch and roll rotation or winds disturbances.

A. Kinematic Constraints

We use a simple UAV model with four dimensions, each configuration is $q=(x,y,z,\theta)$, when x,y,z are the coordinates of the origin, and θ is the orientation, in x-y plane relative to x-axis, as can be seen in Figure 2 for a simple car-like model.

The steering angle is denoted as ϕ. The distance between front and rear axles is equal to *L*. The kinematic equations of a simple UAV model can be written as:

$$\begin{aligned} \dot{x} &= u_s \cos\theta, \\ \dot{y} &= u_s \sin\theta, \\ \dot{z} &= u_{z,} \\ \dot{\theta} &= u_s \tan u_\phi \end{aligned} \tag{7}$$

Where u_s is the speed parallel to x-y plane, climb rate (speed parallel to z-axis) is u_z and the control on steering angle u_ϕ. We denote the control vector as $u=(u_s,u_z,u_\phi)$. Each of the controllers is bounded, $u_\phi \in [-\phi^{\max},\phi^{\max}]$ where $\phi^{\max} < \pi/2$, the speed $u_s \in [u_s^{\min},u_s^{\max}]$ and climb rate $u_z \in [-u_z^{\max},u_z^{\max}]$. $u_s^{\min} > 0$, so UAV cannot stop.

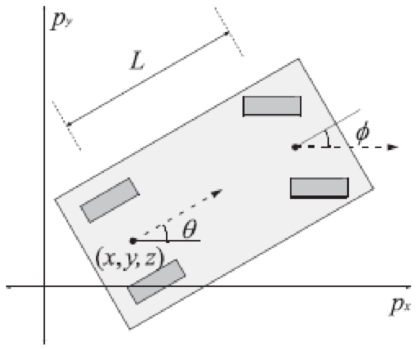


Figure 2. The Simple Car Model. The z-axis can be changed for a Simple -Airplane (Source [26])

B. Dynamic Constraints

The UAV model has to take into account the dynamic constraints, preventing instantaneous changes (increase or decrease) of the control vector $u=(u_s,u_z,u_\phi)$.

UAV model also includes dynamic constraints, $\dot{u}_s \in [-a_s,a_s]$, $\dot{u}_z \in [-a_z,a_z]$ and $\dot{u}_\phi \in [-a_\phi,a_\phi]$.

V. Analytic Visibility Computation

A. Analytic Solution for a Single Object

In this section, we first introduce the visibility solution from a single point to a single 3D object. This solution is based on an analytic expression, which significantly improves time computation by generating the visibility boundary of the object without the need to scan the entire object's points.

Our analytic solution for a 3D building model is an extension of the visibility chart in 2D introduced by Elber et al. [26] for continuous curves. For such a curve, the silhouette points, i.e., the visibility boundary of the object, can be seen in Figure 3:

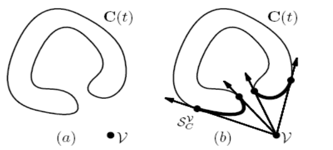


Figure 3. Visible Silhouette Points S_C^V from viewpoint *V* to curve *C(t)* (source: [26]).

The visibility chart solution was originally developed for dealing with the Art Gallery Problem for infinite viewpoint; it is limited to 2D continuous curves using multivariate solver [26], and cannot be used for on-line application in a 3D environment.

Based on this concept, we define the visibility problem in a 3D environment for more complex objects as:

$$C'(x,y)_{z_{const}} \times (C(x,y)_{z_{const}} - V(x_0,y_0,z_0)) = 0 \tag{8}$$

3D model parameterization is $C(x,y)_{z_{const}}$, and the viewpoint is given as $V(x_0,y_0,z_0)$. Solutions to equation (8) generate a visibility boundary from the viewpoint to an object, based on basic relations between viewing directions from *V* to $C(x,y)_{z_{const}}$ using cross-product characters.

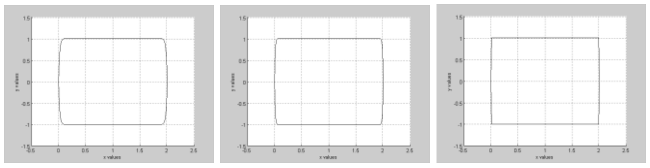
A three-dimension urban environment consists mainly of rectangular buildings, which can hardly be modeled as continuous curves. Moreover, an analytic solution for a single 3D model becomes more complicated due to the higher dimension of the problem and is not always possible. Object parameterization is therefore a critical issue, allowing us to find an analytic solution and, using that, to generate the visibility boundary very fast.

1) 3D Building Model: Most of the common 3D City Models are based on object-oriented topologies, such as 3D Formal Data Structure (3D FDS), Simplified Spatial Model (SSS) and Urban Data Model (UDM) [26]. These models are very efficient for web-oriented applications. However, the fact that a building consists of several different basic features makes it almost impossible to generate analytic representation. A three-dimension building model should be, on the one hand, simple enabling analytic solution, and on the other hand, as accurate as possible. We examined several building object parameterizations, and the preferred candidate was an extended n order sphere coordinates parameterization, even though such a model is a very complex, and will necessitate a special analytic solution. We introduce a model that can be used for analytic solution of the current problem. The basic building model can be described as:

$$x = t, y = \begin{pmatrix} x^n - 1 \\ 1 - x^n \end{pmatrix}, z = c \qquad (9)$$

$$-1 \le t \le 1, n = 350, c = c + 1$$

This mathematical model approximates building corners, not as singular points, but as continuous curves. This building model is described by equation (9), with the lower order badly approximating the building corners, as depicted in Figure 4. Corner approximation becomes more accurate using *n=350* or higher. This approximation enables us to define an analytic solution to the problem.



(a) (b) (c)

Figure 4. Topside view of the building model using equation (2) - (a) n=50; (b) *n=200*; (c) *n=350*.

We introduce the basic building structure that can be rotated and extracted using simple matrix operators (Figure 4). Using a rotation matrix does not affect our visibility algorithm, and for simple demonstration of our method we present samples of parallel buildings.

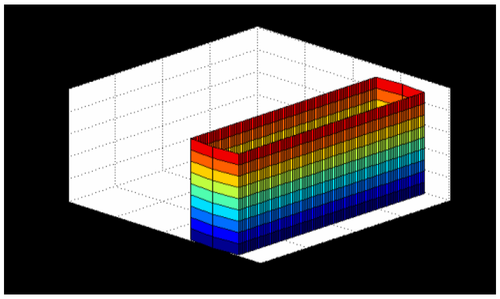


Figure 5. A Three-dimension Analytic Building Model with Equation (8), where $z_{h_{\min}=0}^{h_{\max}=9}$

2) Analytic Solution for a Single Building: In this part we demonstrate the analytic solution for a single 3D building model. As mentioned above, we should integrate building model parameterization to the visibility statement. After integrating eqs. (8) and (9):

$$\begin{aligned} & C'(x,y)_{z_{const}} \times (C(x,y)_{z_{const}} - V(x_0,y_0,z_0)) = 0 \rightarrow \\ & x^n - V_{y_0} - n \cdot x^{n-1}(x - V_{x_0}) - 1 = 0 \\ & x^n + V_{y_0} - n \cdot x^{n-1}(x - V_{x_0}) - 1 = 0 \\ & n = 350, -1 \le x \le 1 \end{aligned} \qquad (10)$$

where the visibility boundary is the solution for these coupled equations. As can be noticed, these equations are not related to Z axis, and the visibility boundary points are the same ones for each x-y surface due to the model's characteristics. Later on, we treat the relations between a building's roof and visibility height in our visibility algorithm, as part of the visibility computation.

The visibility statement leads to two polynomial *N* order equations, which appear to be a complex computational task. The real roots of these polynomial equations are the solution to the visibility boundary. These equations can be solved efficiently by finding where the polynomial equation changes its sign and cross zero value; generating the real roots in a very short time computation (these functions are available in Matlab, Maple and other mathematical programs languages). Based on the polynomial cross zero solution, we can compute a fast and exact analytic solution for the visibility problem from a viewpoint to a 3D building model. This solution allows us to easily define the Visible Boundary Points.

Visible Boundary Points (VBP) - we define VBP of the object *i* as a set of boundary points *j=1..N_{bound}* of the visible surfaces of the object, from viewpoint *V(x₀, y_{0,} z₀)*.

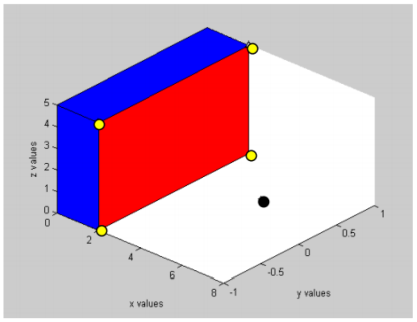
$$VBP_{i=1}^{j=1..N_{bound}}(x_0,y_0,z_0) = \begin{bmatrix} x_1, y_1, z_1 \\ x_2, y_2, z_2 \\ .. \\ x_{N_{bound}}, y_{N_{bound}}, z_{N_{bound}} \end{bmatrix} \qquad (11)$$

Roof Visibility – The analytic solution in equation (10) does not treat the roof visibility of a building. We simply check if viewpoint height *V(z₀)* is lower or higher than the building height $h_{\max_{C_i}}$ and use this to decide if the roof is visible or not:

$$V_{z_0} \ge Z = h_{\max_{C_i}} \qquad (12)$$

If the roof is visible, roof surface boundary points are added to VBP. Roof visibility is an integral part of VBP computation for each building.

Two simple cases using the analytic solution from a visibility point to a building can be seen in Figure 6. The visibility point is marked in black, the visible parts colored in red, and the invisible parts colored in blue. The visible volumes are computed immediately with very low computation effort, without scanning all the model's points, as is necessary in LOS-based methods for such a case.


(a)

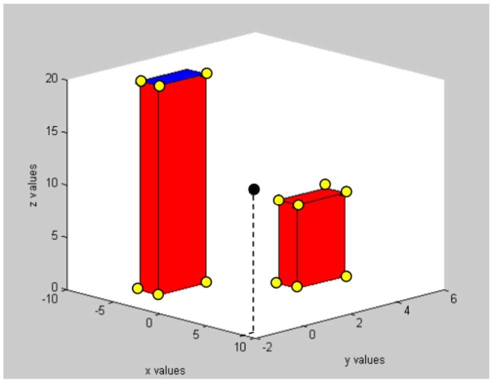

(b)

Figure 6. Visibility Volume computed with the Analytic Solution. Viewpoint is marked in black, visible parts colored in red, and invisible parts colored in blue. VBP marked with yellow circles - (a) single building; (b) two non-overlapping buildings.

B. *Visibility Computation in Urban Environments*

In the previous sections, we treated a single building case, without considering hidden surfaces between buildings, i.e., building surface occluded by other buildings, which directly affect the visibility volumes solution. In this section, we introduce our concept for dealing with these spatial relations between buildings, based on our ability to rapidly compute visibility volume for a single building generating VBP set.

Hidden surfaces between buildings are simply computed based on intersections of the visible volumes for each object. The visible volumes are defined easily using VBP, and are defined, in our case, as Visible Pyramids. The invisible components of the far building are computed by intersecting the projection of the closer buildings' VP base to the far building's VP base.

1) The Visible Pyramid (VP): we define $VP_i^{j=1..Nsurf}(x_0, y_0, z_0)$ of the object i as a 3D pyramid generated by connecting VBP of specific surface j to a viewpoint $V(x_0, y_0, z_0)$. Maximum number of N_{surf} for a single object is three. VP boundary, colored with green arrows, can be seen in Figure 6. The intersection of VPs allows us to efficiently compute the hidden surfaces in urban environments, as can be seen in the next sub-section.

2) Hidden Surfaces between Buildings: As we mentioned earlier, invisible parts of the far buildings are computed by intersecting the projection of the closer buildings' VP to the far buildings' VP base.

For simplicity, we demonstrate the method with two buildings from a viewpoint $V(x_0, y_0, z_0)$ one (denoted as the first one) of which hides, fully or partially, the other (the second one).

As can be seen in Figure 7, in this case, we first compute VBP for each building separately, $VBP_1^{1..4}$, $VBP_2^{1..4}$, based on these VBPs, we generate VPs for each building, VP_1^1, VP_2^1. After that, we project VP_1^1 base to VP_2^1 base plane, as seen in Figure 8, if existing. At this point, we intersect the projected surface in VP_2^1 base plane and update $VBP_2^{1..4}$ and VP_2^1 (decreasing the intersected part).

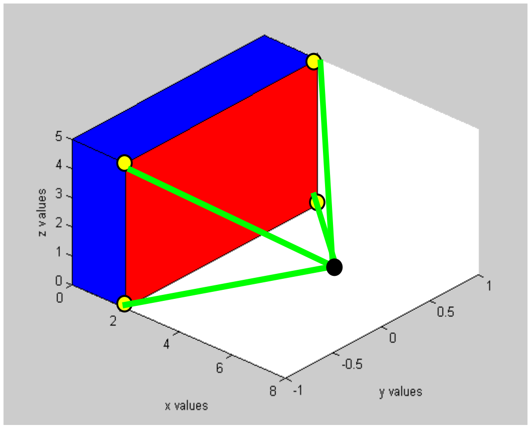
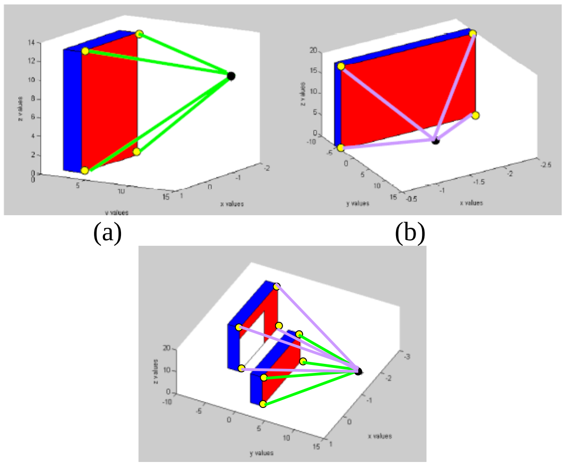


Figure 7. A Visible Pyramid from a viewpoint (marked as a black point) to VBP of a specific surface


(a) (b)

(c)

Figure 8. Generating VP - (a) VP_1^1 boundary colored in green arrows; (b) VP_2^1 boundary colored in purple lines; (c) the two buildings - VP_1^1 in green and VP_2^1 in purple, from the viewpoint.

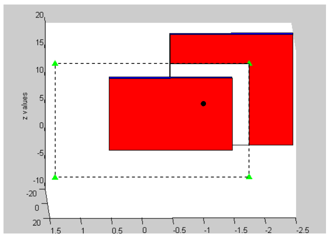


Figure 9. Projection of VP_1^1 to VP_2^1 base plane marked with dotted lines.

The intersected part is the invisible part of the second building from viewpoint $V(x_0, y_0, z_0)$ hidden by the first building, which is marked in white in Figure 9. In the case of a third building, in addition to the buildings introduced in Figure 9, the projected VP will only be the visible ones, and the VBP and VP of the second building will be updated accordingly.

We demonstrated a simple case of an occluded building. A general algorithm for more a complex scenario, which contains the same actions between all the combinations of VP between the objects, is detailed in the next sub-section. Projection and intersection of 3D pyramids can be done with simple computational geometry elements, which demand a very low computation effort.

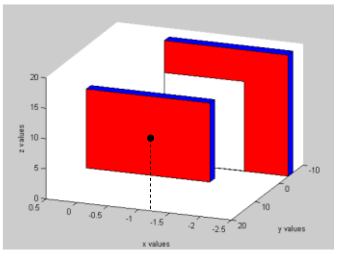


Figure 10. Computing Hidden Surfaces between Buildings by using the Visible Pyramid Colored in White on VP_2^1 Base Plane.

C. Viewpoint Invisibility Value

Planning UAVs visible trajectory is based on the ability to accumulate the visibility value of each viewpoint explored as part of the planner algorithm. We calculate the exact invisible value of a specific viewpoint, i.e., the total sum of the invisible surfaces and roofs from viewpoint.

We divide point invisibility value into Invisible Surfaces Value (ISV) and Invisible Roofs Value (IRV). This classification allows us to plan delicate and accurate trajectory 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 equation (13):

$$ISV(x_0, y_0, z_0) = \sum_{i=1}^{N_{obj}} IS_{VP_i^{j=1..N_{bound}-1}}^{VP_i^{j=1..N_{bound}-1}} \tag{13}$$

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

$$IRV(x_0, y_0, z_0) = \sum_{i=1}^{N_{obj}} IS_{VP_i^{j=N_{bound}}}^{VP_i^{j=N_{bound}}} \tag{14}$$

VI. Deep Reinforcement Learning (DRL) Planner

Our planner, as described in Table 1, based on DRL method, generate visible sequence of optimal-visible waypoints as a candidate trajectory. We extend previous planners which take into account kinematic and dynamic constraints [26][27] and present a local planner for UAV with these constraints, which for the first time generates fast and exact visible trajectories based on analytic solution. The fast and efficient visibility analysis of our method presented above, allows us to generate the most visible trajectory from a start state to the goal state in 3D urban environments, and demonstrates our capability, which can be extended to real performances in the future. We assume knowledge of the 3D urban environment model and use the well-known Velocity Obstacles (VO) method to avoid collision with buildings presented as static obstacles.

For obstacle avoidance capability, at each time step, the planner computes the next eighth Attainable Velocities (AV). The safe nodes not colliding with buildings, i.e., nodes outside Velocity Obstacles [25], are explored. The planner computes the cost for these safe nodes and chooses the node with the lowest cost. Trajectory can be characterized by the most visible roofs only, surfaces only, or another combination of these kinds of visibility types. We repeat this procedure while generating the most visible trajectory.

A. Velocity Obstacles

The Velocity Obstacles (VO) [25] is a well-known method for obstacle avoidance in static and dynamic environments, used in our planner to prevent collision between UAV and the buildings (as static obstacles), as part of the trajectory planning method.

The VO represents the set of all colliding velocities of the UAV with each of the neighboring obstacles, in our case static obstacles - buildings. Each building is bounded by cylinder instead of circle in 2D case [25] and mapped as static obstacle into the UAV's velocity space.

We introduce the velocity obstacles of a planar circular obstacle, B, which is moving at a constant velocity v_b, as a cone in the velocity space of UAV ,*A*, reduced to a point by correspondingly enlarging obstacle B.

Each point in VO represents a velocity vector that originates at *A*. Any velocity of *A* that penetrates VO is a colliding velocity that would result in a collision between *A* and *B* at some future time. Figure 10 shows two velocities of *A*: one that penetrates VO, v_{a1}, and is hence a colliding velocity, and one that does not, v_{a2}.

All velocities of A that are outside of VO are safe as long as B stays on its current course or in our case a static one. The velocity obstacles thus allows us to determine if a given UAV velocity will cause a collision.

B. *Attainable Velocities*

Based on the dynamic and kinematic constraints, UAVs velocities at the next time step are limited. At each time step during the trajectory planning, we map the Attainable Velocities (AV), the velocities set at the next time step $t+\tau$, which generate the optimal trajectory, as is well-known from Dubins theory [27].

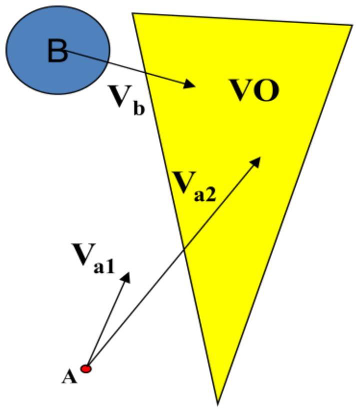


Figure 11. Linear Velocity Obstacles

We denote the allowable controls as $u=(u_s,u_z,u_\phi)$ as U, where $V\in U$.

We denote the set of dynamic constraints bounding control's rate of change as $\dot{u}=(\dot{u}_s,\dot{u}_z,\dot{u}_\phi)\in U'$.

Considering the extremals controllers as part of the motion primitives of the trajectory cannot ensure time-optimal trajectory for Dubin's airplane model [27], but is still a suitable heuristic based on time-optimal trajectories of Dubin - car and point mass models.

We calculate the next time step's feasible velocities $\tilde{U}(t+\tau)$, between $(t,t+\tau)$:

$$\tilde{U}(t+\tau)=U\cap\{u\mid u=u(t)\oplus\tau\cdot U'\} \quad (15)$$

Integrating $\tilde{U}(t+\tau)$ with UAV model yields the next eight possible nodes for the following combinations:

$$\tilde{U}(t+\tau)=\begin{pmatrix}\tilde{U}_s(t+\tau)\\ \tilde{U}_z(t+\tau)\\ \tilde{U}_\phi(t+\tau)\end{pmatrix}=\begin{pmatrix}u_s^{\min}{}_{,}u_s(t)+a_s\tau\\ -u_s^{\max}\tan\phi^{\max},u_s(t)\tan u_\phi(t)+u_s^{\max}\tan a_\phi\\ u_z^{\max}u_z(t)-a_z\tau\end{pmatrix} \quad (16)$$

At each time step, we explore the next eight AV at the next time step as part of our tree search. Each node $(q,\dot{q})$, where $q=(x,y,z,\theta)$, consist of the current UAVs position and velocity at the current time step. At each state, the planner computes the set of Attainable Velocities (AV), $\tilde{U}(t+\tau)$, from the current UAV velocity, $U(t)$, as shown in Figure 12. We ensure the safety of nodes by computing a set of Velocity Obstacles (VO).

In Figure 12, nodes inside VO, marked in red, are inadmissible. Nodes out of VO are further evaluated; safe nodes are colored in blue. The safe node with the lowest cost, which is the next most visible node, is explored in the next time step. This is repeated while generating the most visible trajectory.

Attainable velocities profile is similar to a trunked cake slice, as seen in Figure 12, due to the Dubins airplane model with one time step integration ahead. Simple models attainable velocities, such as point mass, create rectangular profile [25].

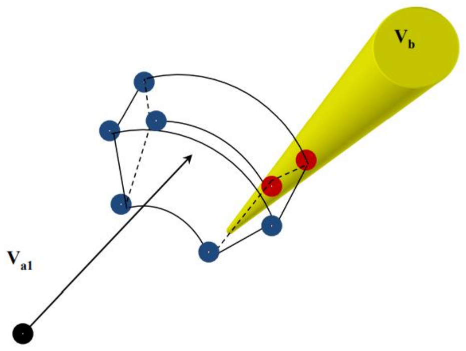


Figure 12. Tree Search Method. Attainable Velocities marked in Blue and Red Circles; Nodes inside VO (marked Red) are Inattainable; Nodes outside VO, Colored in Blue with Lowest Cost, are Explored

C. *Cost Function*

Our search is guided by minimum invisible parts from viewpoint V to the 3D urban environment model. The cost function for each node is a combination of IRV and ISV, with different weights as functions of the required task.

The cost function is computed for each safe node $(q,\dot{q})\notin VO$, i.e., node outside *VO*, considering UAV position at the next time step $(x(t+\tau),y(t+\tau),z(t+\tau))$ as viewpoint:

$$w(q(t+\tau))=\alpha\cdot ISV(q(t+\tau))+\beta\cdot IRV(q(t+\tau)) \quad (17)$$

Where α,β are coefficients, effecting the trajectory character. The cost function $w(q(t+\tau))$ produces the total sum of invisible parts from the viewpoint to the 3D urban environment, meaning that the velocity at the next time step with the minimum cost function value is the most visible node in our local search.

D. Planner Neural Network

In our DRL model, we are using fully-connected layers, consisting of:

- the state space of 37 dimensions
- Two hidden layers (64 nodes each)
- An output of four actions

Our network structure can be seen in Figure 13.

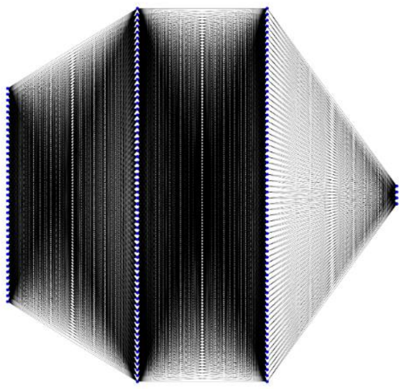


Figure 13. DRL planner network model based on fully-connected layers

E. Simulation Results

We have implemented the presented algorithm and tested some urban environments. We computed the visible trajectories using our DRL planner, as described above. We used the proposed UAV model with several types of trajectories consisting of roof and surfaces visibility, based on the introduced visibility computation method. Obstacle avoidance capability tested by VO method.

The initial parameters values are: $u_s(t=0)=10$ [m/s], u_z $\theta(t=0)=5[\text{deg}]$. UAV dynamic and kinematic constraints are $\phi^{\max}=\pi/4$, $u_z^{\max}=0.3[m/s]$. $u_s^{\min}=1$ [m/s], $u_s^{\max}=15$ [m/s].

In the following figures the start and goal points are marked, in number of scenarios with various start's and goal's points location.

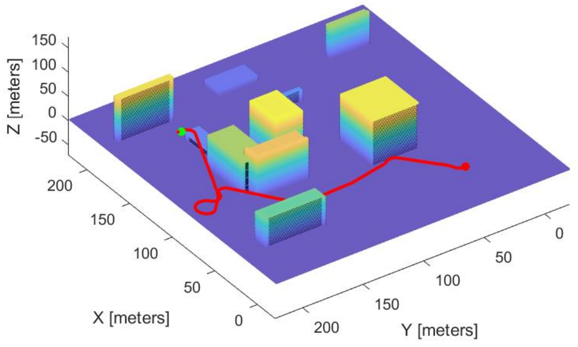


Figure 14. Trajectory Planning in Urban Environment Using DRL. Start and Goal Points with Scenario Demonstration.

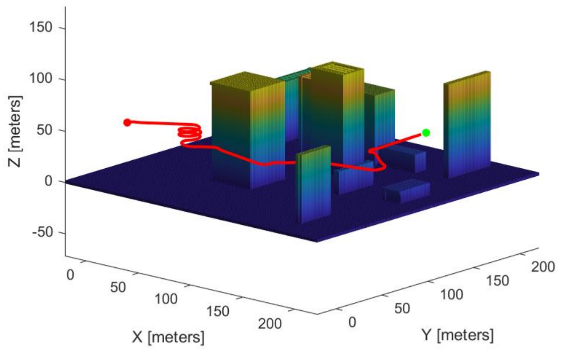


Figure 15. Trajectory Planning in Urban Environment Using DRL. Setting other Start and Goal Points with Scenario Demonstration.

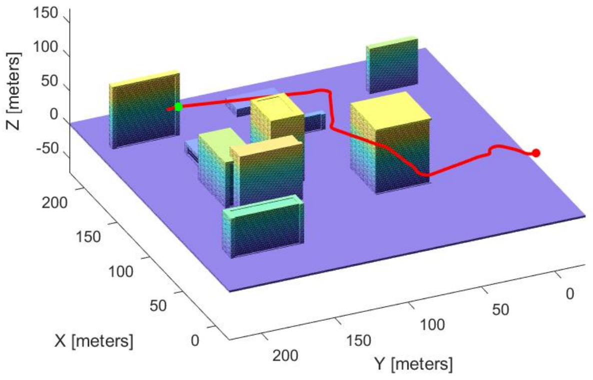


Figure 16. Trajectory Planning in Urban Environment Using DRL. Setting other Start and Goal Points with Scenario Demonstration.

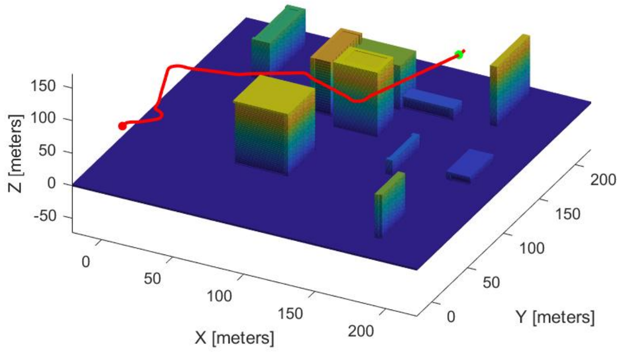


Figure 17. Trajectory Planning in Urban Environment Using DRL. Setting other Start and Goal Points with Scenario Demonstration.

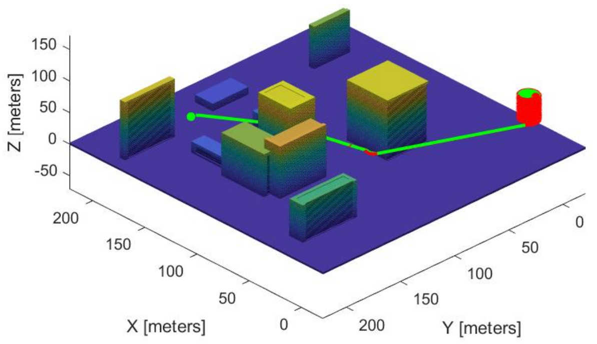


Figure 18. Trajectory Planning in Urban Environment Using DRL. Setting other Start and Goal Points with Scenario Demonstration.

VII. CONCLUSIONS

In this paper, we present spatial motion planner in 3D environments based on Deep Reinforcement Learning (DRL) algorithms. We tackled 3D motion planning problem by using Deep Reinforcement Learning (DRL) approach, which learns agent's and environment constraints.

Spatial analysis focuses on visibility analysis in 3D setting an optimal motion primitive considering agent's dynamic model based on fast and exact visibility analysis for each motion primitives. Based on optimized reward function, which consist of generated 3D visibility analysis and obstacle avoidance trajectories, we introduced DRL formulation, which learns the value function of the planner and generates an optimal spatial visibility trajectory.

We demonstrated our planner in simulations for Unmanned Aerial Vehicles (UAV) in 3D urban environments.

Our spatial analysis is based on a fast and exact spatial visibility analysis of the 3D visibility problem from a viewpoint in 3D urban environments.

We presented DRL architecture generating the most visible trajectory in a known 3D urban environment model, as time-optimal one with obstacle avoidance capability.

VIII. REFERENCES

[1] O. Gal and Y. Doytsher, "Spatial Visibility Clustering Analysis In Urban Environments Based on Pedestrians' Mobility Datasets," The Sixth International Conference on Advanced Geographic Information Systems, Applications, and Services, pp. 38-44, 2014.

[2] J. Bellingham, A. Richards, and J. How, "Receding Horizon Control of Autonomous Aerial Vehicles," in Proceedings ofthe IEEE American Control Conference, Anchorage, AK, pp. 3741–3746, 2002.

[3] A. Borgers and H. Timmermans, "A model of pedestrian route choice and demand for retail facilities within inner-city shopping areas," Geographical Analysis, vol. 18, No. 2, pp. 115-128, 1996.

[4] S. A. Bortoff, "Path planning for UAVs," In Proc. of the American Control Conference, Chicago, IL, pp. 364–368, 2000.

[5] B. J. Capozzi and J. Vagners, "Navigating Annoying Environments Through Evolution," Proceedings of the 40th IEEE Conference on Decision and Control, University of Washington, Orlando, FL, 2001.

[6] H. Chitsaz and S. M. LaValle, "Time-optimal paths for a Dubins airplane," in Proc. IEEE Conf. Decision. and Control., USA, pp. 2379–2384, 2007.

[7] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic Motion Planning," Journal of the Association for Computing Machinery, pp. 1048–1066, 1993.

[8] Y. Doytsher and B. Shmutter, "Digital Elevation Model of Dead Ground," Symposium on Mapping and Geographic Information Systems (Commission IV of the International Society for Photogrammetry and Remote Sensing), Athens, Georgia, USA, 1994.

[9] W. Fox, D. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," IEEE Robotics and Automation Magazine, vol. 4, pp. 23–33, 1997.

[10] O. Gal and Y. Doytsher, "Fast and Accurate Visibility Computation in a 3D Urban Environment," in Proc. of the Fourth International Conference on Advanced Geographic Information Systems, Applications, and Services, Valencia, Spain, pp. 105-110, 2012 [accessed February 2014].

[11] O. Gal and Y. Doytsher, "Fast and Efficient Visible Trajectories Planning for Dubins UAV model in 3D Built-up Environments," Robotica, FirstView, Article pp. 1-21 Cambridge University Press 2013 DOI: http://dx.doi.org/10.1017/S0263574713000787, [accessed February 2014].

[12] M. Haklay, D. O'Sullivan, and M.T. Goodwin, "So go down town: simulating pedestrian movement in town centres," Environment and Planning B: Planning & Design, vol. 28, no. 3, pp. 343-359, 2001.

[13] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," Int. J. Robot. Res., vol. 30, no. 7, pp. 846–894, 2011.

[14] N.Y. Ko and R. Simmons, "The lane-curvature method for local obstacle avoidance," In International Conference on Intelligence Robots and Systems, 1998.

[15] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," TR 98-11, Computer Science Dept., Iowa State University, 1998.

[16] S. M. LaValle and J. Kuffner. "Randomized kinodynamic planning," In Proc. IEEE Int. Conf. on Robotics and Automation, Detroit, MI, pp. 473–479, 1999.

[17] L.R. Lewis, "Rapid Motion Planning and Autonomous Obstacle Avoidance for Unmanned Vehicles," Master's Thesis, Naval Postgraduate School, Monterey, CA, December 2006.

[18] C. W. Lum, R. T. Rysdyk, and A. Pongpunwattana, “Occupancy Based Map Searching Using Heterogeneous Teams of Autonomous Vehicles,” Proceedings of the 2006 Guidance, Navigation, and Control Conference, Autonomous Flight Systems Laboratory, Keystone, CO, August 2006.

[19] S. Okazaki and S. Matsushita, "A study of simulation model for pedestrian movement with evacuation and queuing," Proceedings of the International Conference on Engineering for Crowd Safety, London, UK, pp. 17-18, March 1993.

[20] P. Abbeel and P. Ng, "Apprenticeship learning via inverse reinforcement learning," in Proceedings of the twenty-first international conference on Machine learning, ICML ’04, ACM, New York, NY, USA, http://doi.acm.org/10.1145/1015330.1015430, 2004.

[21] M. Kuderer, S. Gulati, and W. Burgard, "Learning driving styles for autonomous vehicles from demonstration," in Proceedings of the IEEE International Conference on Robotics & Automation (ICRA), Seattle, USA. vol. 134, 2015

[22] B. Ziebart, A. Maas, J. Bagnell, and A. Dey, "Maximum entropy inverse reinforcement learning," in Proc. of the National Conference on Artificial Intelligence (AAAI), 2008.

[23] S. M. LaValle, "Planning Algorithms,” Cambridge,U.K.:Cambridge Univ. Pr., 2006.

[24] H. Chitsaz and S. M. LaValle, Time-optimal paths for a Dubins airplane, in Proc. IEEE Conf. Decision. and Control., USA, pp. 2379–2384, 2007.

[25] P. Fiorini and Z. Shiller, “Motion planning in dynamic environments using velocity obstacles,” Int. J. Robot. Res.17, pp. 760–772, 1998.

[26] G. Elber, R. Sayegh, G. Barequet and R. Martin. “Two-Dimensional Visibility Charts for Continuous Curves,” in Proc. Shape Modeling, MIT, Boston, USA, pp. 206-215, 2005.

[27] S. A. Bortoff, “Path planning for UAVs,” In Proc. of the American Control Conference, Chicago, IL, pp. 364–368 ,2000.