On Line Visibility-Based Trajectory Planning in 3D Dynamic Environments Using Local Point Clouds Data

[1,2]Oren Gal and [2]Yerach Doytsher

[1]Department of Marine Technologies
University of Haifa
Haifa, Israel
e-mail: orengal@technion.ac.il

[2]Mapping and Geo-information Engineering
Technion - Israel Institute of Technology
Haifa, Israel
e-mail:doytsher@technion.ac.il

Abstract - **In this paper we present an efficient and fast visible trajectory planning for unmanned vehicles in a 3D urban environment based on local point clouds data. 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 is a prediction of these geometric entities in the next time step, formulated as states vectors in a dynamic system using Kalman Filter (KF). Our planner is based on the optimal time horizon concept as a leading feature for our greedy search method for making our local planner safer. We demonstrate our visibility and trajectory planning method in simulations, showing predicted trajectory planning in 3D urban environments based on real LiDAR point clouds data.**

Keywords*- *Visibility; 3D; Urban environment; Spatial analysis.

I. Introduction and Related Work

In this paper we study an efficient and fast visible trajectory planning for unmanned vehicles 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 for generating 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 is a prediction of these geometric entities in the next time step, formulated as states vectors in a dynamic system using Kalman Filter (KF).

Visibility analysis based on this approximated scene prediction is done efficiently [1], based on our analytic solutions for visibility boundaries. Based on 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, which is based on local point clouds data from the unmanned LiDAR vehicle. Our planner is based on the optimal time horizon concept as a leading feature for our greedy search method for making our local planner safer.

For the first time, we propose a solution to the basic limitation of the Velocity Obstacle (VO) search and planning method, i.e., when all the dynamic available velocities for the next time step are blocked in the velocity space and there is no legal node at the next time step of the greedy search. The computation of the minimum time horizon is formulated as a minimum time problem that generates optimal trajectories in near-time time to the goal, exploring the most visible and safest node in the next time step. We demonstrate our visibility and trajectory planning method in simulations showing predicted trajectory planning in 3D urban environments using real LiDAR data from Ford Campus Project [2].

The main challenge in motion planning is reaching the goal while searching and selecting only safe maneuvers. While reaching the goal cannot be guaranteed with an on-line planner, one can reduce the state space search to only safe states, i.e., states outside obstacles from which at least one other safe state is reachable.

Generally, we distinguish between local and global planners. The local planner generates one step, or a few steps, at every time step, whereas the global planner uses a global search toward the goal over a time-spanned tree. We can divide this work into global and local (reactive) planners. The global planners generate complete trajectories to the goal in static [3] and dynamic [4,5] environments.

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 [23]. 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 DEM models, 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 [24]. Other fast algorithms are based on the conservative Potentially Visible Set (PVS) [25]. These methods are not always completely accurate, as they may render hidden objects' parts as visible due to various simplifications and heuristics.

A vast number of algorithms have been suggested for speeding up the process and reducing computation time. Franklin [26] evaluates and approximates visibility for each cell in a DEM model based on greedy algorithms. Wang et al. [27] introduced a Grid-based DEM method using viewshed horizon, saving computation time based on relations between surfaces and the line of sight (LOS method). Later, an extended method for viewshed computation was presented, using reference planes rather than sightlines [28].

II. Visibility Analysis from Point Clouds Data

As we mentioned, visibility analysis in complex urban scenes is commonly treated as an approximated feature due to 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 extraction have been studied extensively, and are still a very active research field [34].

In this part, we present an 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 [35] 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., attainable location of the vehicle in the next time step. Scene prediction is based on the geometric entities and Kalman Filter (KF) which is commonly used in dynamic systems for tracking target systems [36,37]. We formulate the geometric shapes as states vectors in a dynamic system and predict the scene structure the 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 [38], 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 procedure for reliable KF prediction.

Our concept is based on RANSAC and KF, both real-time algorithms, which can be integrated into autonomous mapping vehicles that have become very popular. This concept can be applicable for robot trajectory planning generating visible paths, by analyzing local point clouds data and predicting the most visible viewpoint in the next time step from among several options.

A. *Concept's Stages*

Our methodology can be divided into three main sub-problems:

1) *Extract basic geometric shapes from point clouds data (using RANSAC algorithms)*
2) *Predict scene modeling in the next viewpoint (using KF)*
3) *Approximated visibility analysis of a predicted scene*

Each of the following stages is done after the other, where the last stage also includes updated measurement of point clouds data validating KF for the next viewpoint analysis.

B. *Shapes Extraction*

1) Geometric Shapes:

The urban scene is a very complex one in the matter of modeling applications using ground LiDAR, and the generated point clouds is very dense. Due to these inherited 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.

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

2) RANSAC:

The RANSAC [39] paradigm is a well-known one, extracting shapes from point clouds using a minimal set of shape's primitives generated by random drawing in point clouds set. 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 and 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 extract 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 $R = P \backslash \{p_{\delta_1}..p_{\delta_N}\}$.
In this part we briefly introduce the main idea of plane, sphere and cylinder extraction from point clouds data. An extended study of RANSAC capabilities can be found in [35].

Plane: A minimal set in the case of a plane, can be found by just three points $\{p_1, p_2, p_3\}$, without considering normals in the points. Final validation of the candidate plane is computed from the deviation of the plane's normal from $\{n_1, n_2, n_3\}$. A plane is extracted only in cases where all deviations are less than the predefined angle α.
Sphere: A sphere is fully defined by two points with corresponding normal vectors. The sphere center is defined from the midpoint of the shortest line segment between the two lines given by the points and their normals.
A sphere counts as a detected shape in cases where all three points are within a distance of ε from the sphere and their normals do not deviate by more than α degrees.
Cylinder: A cylinder is set by two points and their normals, where the cylinder axis direction is the projected cross product of the normals, and a center point is calculated as the intersection of parametric lines generated from points and points' normal. A cylinder is verified by applying the thresholds ε and α to distance and to normal deviation of the samples.

C. Predicted Scene – Kalman Filter

In this part, we present the global Kalman Filter 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 [36]. We model our system as a linear one, with discrete dynamic model:

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

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*:

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

Direction vector $\vec{d}_k$ can be efficiently estimated 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:

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

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:

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

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 part of 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. The sphere radius and cylinder Z boundaries are defined in additional data structure of the scene entities.

III. Fast and Approximated Visibility Analysis

In this section, we present an analytic analysis of 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 [38].

In this part, we extend the previous visibility analysis concept [38] and include cylinders as continuous curves parameterization $C_{\mathrm{clnd}}(x,y,z)$.
Cylinder parameterization can be described as:

$$C_{Clnd}(x,y,z)=\begin{pmatrix} r\sin(\theta) \\ r\cos(\theta) \\ c \end{pmatrix}_{r=const}$$

$$0\le\theta\le 2\pi \qquad c=c+1 \qquad 0\le c\le h_{peds\_\max} \tag{5}$$

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 cylinder case. Integrating equation (5) to (6) yields:

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

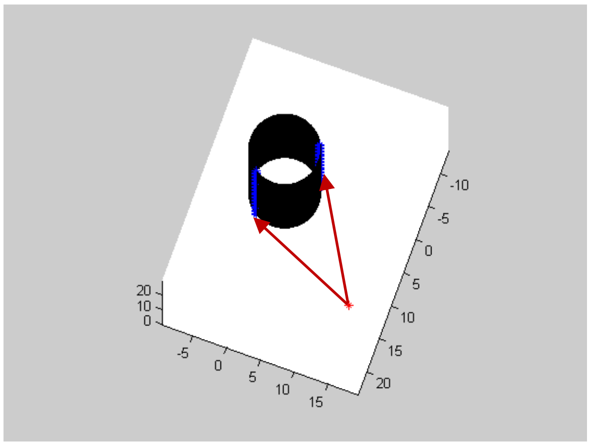
$$\theta=\arctan\left(-\frac{-r-\frac{\left(-vy\,r+\sqrt{vx^4-vx^2r^2+vy^2vx^2}\right)vy}{vx^2+vy^2}}{vx},\ \frac{-vy\,r+\sqrt{vx^4-vx^2r^2+vy^2vx^2}}{vx^2+vy^2}\right) \tag{8}$$

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.

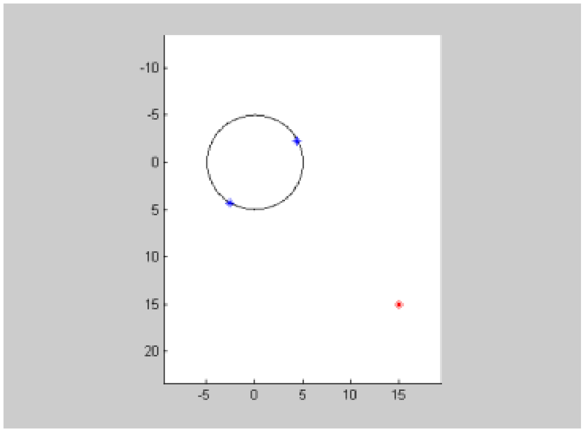
The visibility statement leads to complex equation, which does not appear to be a simple computational task. This equation can be solved efficiently by finding where the equation changes its sign and crosses zero value; we used analytic solution to speed up computation time and to avoid numeric approximations. 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 equation (8) as x-y-z coordinates values for the cylinder case as Cylinder Boundary Points (CBP). CBP 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:

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

We define the visibility problem in a 3D environment for this object as:

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

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

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

where r is set from sphere parameter, and $V(x_0,y_0,z_0)$ is changes from visibility point along Z axis. 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, that being 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. Fast Visible Trajectory Planning

Our planner is a local one, generating one step ahead at every time step reaching toward the goal, which is a depth first A* search over a tree. We extend previous planners, which take into account kinematic and dynamic constraints [16] and present a local planner for an omni-directional robot, with these constraints mounted with LiDAR in a constant Z point. As far as we know, for the first time this planner generates fast and exact visible trajectories based on an optimal analytic time horizon solution handling blocked states where all future states are inside VO, and approximates visibility based on local point clouds data for the next time step based on incomplete data. The fast and efficient visibility analysis of our method [38], extended in Section II for spheres and cylinders, allows us to generate the most visible trajectory from a starting state to the goal state in 3D urban environments, based on local decision-making capabilities, and demonstrates our capability, which can be extended to real performances in the future.

We assume incomplete data of the 3D urban environment model as mentioned in Section II, and use an extended Velocity Obstacles (VO) method with analytic optimal time horizon.

A. *Analytic Optimal Time Horizon – Escaping Mode*

The time horizon plays an important role in selecting feasible avoidance maneuvers. It allows considering only those maneuvers that would result in a collision within a specified time interval and efficiently searching for safe maneuvers in the velocity space. Setting the time horizon too high would be too prohibitive, as it would mark as dangerous maneuvers resulting in collision at a distant time; selecting a too-small time horizon would permit dangerous maneuvers that are too close and at too high speeds to avoid the obstacle.

It is essential that the proper time horizon ensures that a safe maneuver, even if temporarily pointing toward the obstacle, is selected.

The main significance of the time horizon parameter using VO was first introduced in [21]. For each obstacle, time horizon is calculated as the minimum between stopping and passing time, as approximations to the exact optimization problem. Numeric solutions of the optimal time horizon for point mass model with cubic control constraints were presented in [21], based on external trajectories generated from the boundary of the control effort. This formulation of time horizon defines approximation of VO as the boundary of ICS without analytic solution escaping VO, in a case of bounded velocity space.

B. *Analytic Optimal Time Horizon - Examples*

In this part, we focus on the efficiency of our analytic time horizon solution via classic VO demonstrated in simulations. The analytic solution extends the traditional VO planner search method and defines the strategy search in cases of blocked attainable velocity space for the next time step in velocity space.

We use a planner similar to the one presented by [21] with the same cost function, and the Omni-directional robot model mentioned above. The search is guided by a cost function planner applying the safest maneuver at every time step. Unsafe states ahead in time are recognized before the robot enters into unsafe states, also called ICS. For one obstacle, our planner can ensure safety, but the planner is not a complete one. By using an analytic search, the planner computes near-time optimal and safe trajectory to the goal.

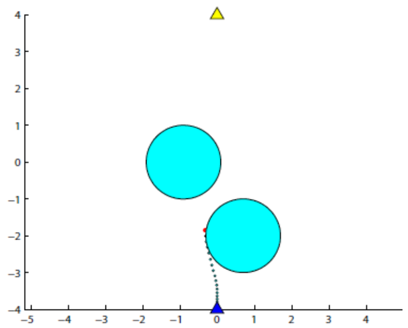


Figure 2. Avoiding Two Obstacles Using Analytic Time Horizon.

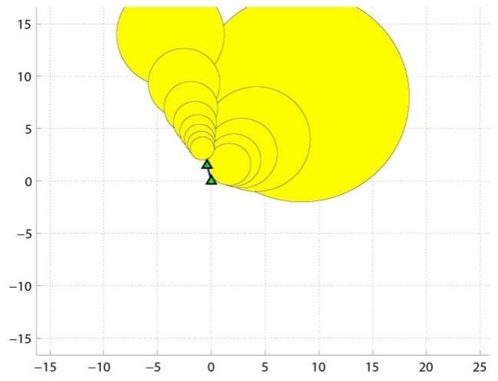


Figure 3. Blocked Velocity Space Avoiding Two Obstacles.

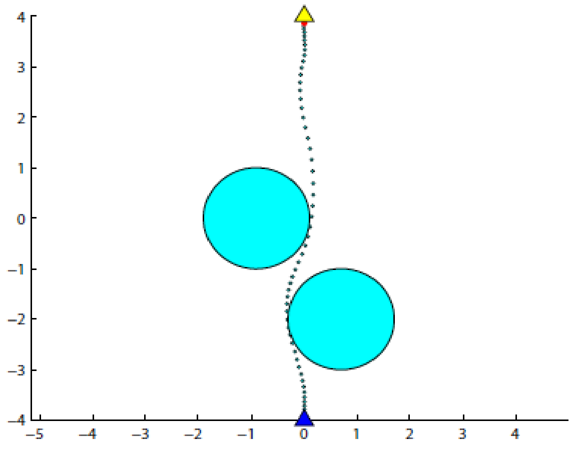


Figure 4. Final Trajectory Avoiding Two Obstacles Using Analytic Time Horizon.

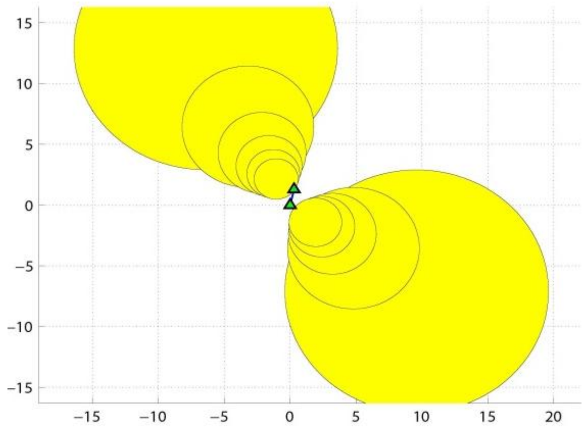


Figure 5. Escaping Blocked Velocity Space Using Analytic Time Horizon.

The main contribution of this section is to demonstrate cases of blocked nodes in the velocity space in the search tree for the next time step. In cases of blocked nodes, i.e., all of the nodes located inside the VO, the planner choose the node that leads outside VO as soon as possible, avoiding collision and formulated as analytic time horizon based search. Without using analytic time horizon formulation, there is no safe and legitimate option for the next node to be explored. As a result, conservative trajectories are computed, and in some cases safe trajectory to the goal cannot be found and collision eventually occurs.

In a two-obstacles case shown in Figure 2, the robot, represented by a point, starts near point (0,-4) at zero speed, attempting to reach the goal at point (0,4) (marked by a yellow triangle) at zero speed, while avoiding two static obstacles. The trajectory is dotted with a red dot representing the current position of the robot. The bounded velocity space, representing velocity obstacles as yellow cycles and velocity vector (with green triangles), can be seen in Figure 3, relating to the state space position as shown in Figure 2.

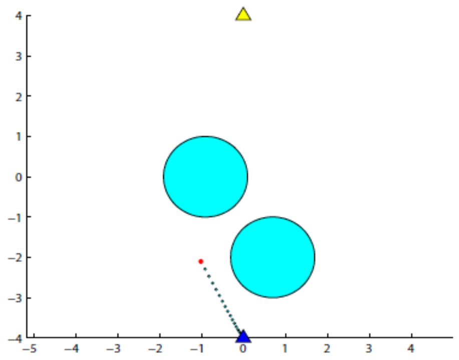


Figure 6. Conservative Solution of Avoiding Two Obstacles Using Constant Time Horizon: Blocked Velocity Space Caused to Conservative Trajectory Turning Left vs. Sliding on their Edges and Passing Between them.

Clearly, there is no gap to enter between VO's in Figure 3 and the velocity vector is bounded in the velocity space. The trivial VO, with a conservative and constant time horizon, cannot find the ultimate solution in such a case, and as a result, a conservative trajectory will be computed. The robot avoids the obstacles to the left with high time horizon values, as shown in Figure 6. Moreover, in some other cases of dense and bounded velocity space, no solution will be available at all. By using an analytic time horizon, the robot escapes velocity obstacles and searches for a safe maneuver in state space, as shown in Figure 4, and velocity space, respectively, as shown in Figure 5.

C. *The Planner*

By using RANSAC algorithm, at each time step point clouds data are extracted 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 node is set as the 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 Velocity Obstacles 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 considering 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) *Attainable Velocities*

The set of maneuvers that are dynamically feasible over a time step is represented by AV. At each time step during the trajectory planning, we map the attainable velocities that the robot can choose under the effort control envelope.

Attainable Velocities, $AV(t+\Delta t)$, are integrated from the current state (x_1, x_2) by applying all admissible controls $u(t) \in U$. The geometric shape of AV depends on system dynamics. In our case,as described in (13):

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= u \end{aligned} \tag{13}$$

where $x_1, u \in R^2$.

$$AV(t+\Delta t) = \{v \mid v = v(t) + \Delta t u, u \in U\}$$

The attainable velocities at time $t+\Delta t$ apply to the position $x(t+\Delta t)$. Thus, the attainable velocities, when intersected with VO that correspond to the same position, would indicate those velocities that are safe if selected at time $t+\Delta t$.

2) *Cost Function*

Our search 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 for each node is a combination of IRV and ISV, with different weights as functions of the required task.

The cost function presented in (14) is computed for each safe node, i.e., node outside VO, considering the robot's future location at the next time step $(x_1(t+\Delta t), x_2(t+\Delta t))$ as viewpoint:

$$w\big(x(t+\Delta t)\big) = \alpha \cdot ISV(x(t+\Delta t)) + \beta \cdot IRV(x(t+\Delta t)) \tag{14}$$

where $\propto, \beta$ are coefficients, affecting the trajectory's character. The cost function $w(x(t+\Delta t)$ 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, based on our approximation.

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 (15):

$$ISV(x_0, y_0, z_0) = \sum_{i=1}^{N_{obj}} IS_{VP_i^{j=1..N_{bound}-1}}^{VP_i^{j=1..N_{bound}-1}} \tag{15}$$

In the same way, we define Invisible Roofs Value (IRV) as the total sum of all the invisible roofs' surfaces, as described in (16):

$$IRV(x_0, y_0, z_0) = \sum_{i=1}^{N_{obj}} IS_{VP_i^{j=N_{bound}}}^{VP_i^{j=N_{bound}}} \tag{16}$$

Extended analysis of the analytic solution for visibility analysis for known 3D urban environments can be found in [37].

V. SIMULATIONS

We have implemented the presented algorithm and tested some urban environments on a 1.8GHz Intel Core CPU with Matlab. We computed the visible trajectories using our planner, with real raw data records from LiDAR as part of the Ford Campus Project.

Point clouds data are generated by Velodyne HDL-64E LiDAR [39]. Velodyne HDL-64E LiDAR has two blocks of lasers, each consisting of 32 laser diodes aligned vertically, resulting in an effective 26:8 Vertical Field Of View (FOV). The entire unit can spin about its vertical axis at speeds up to 900 rpm (15 Hz) to provide a full 360 degree azimuthal field of view. The maximum range of the sensor is 120 m and it captures about 1 million range points per second. We captured our data set with the laser spinning at 10 Hz.

Due to these huge amounts of data, we planned a limited trajectory in this urban environment for a limited distance. In Figure 7, point clouds data from the start point can be seen, also marked as start point "S" in Figure 10. Planes extracted by RANSAC can be recognized. As part of the Ford Project, these point clouds are also projected to the panoramic camera's systems, making it easier to understand the scene, as seen in Figure 8.(34)

As described earlier, at each time step the planner predicts the objects in the scene using KF. In Figure 9(a), objects in the scene are presented from a point clouds data set. These point clouds predicted using KF, and predicted to the next time step in Figure 9(b).

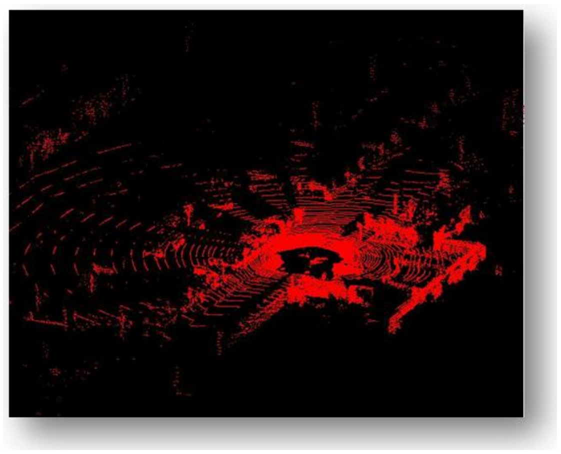


Figure 7. Point Clouds Data set at Start Point.

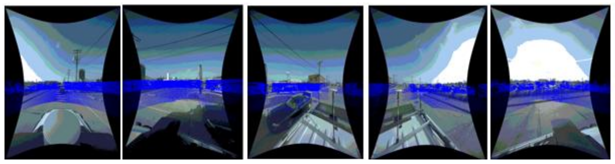
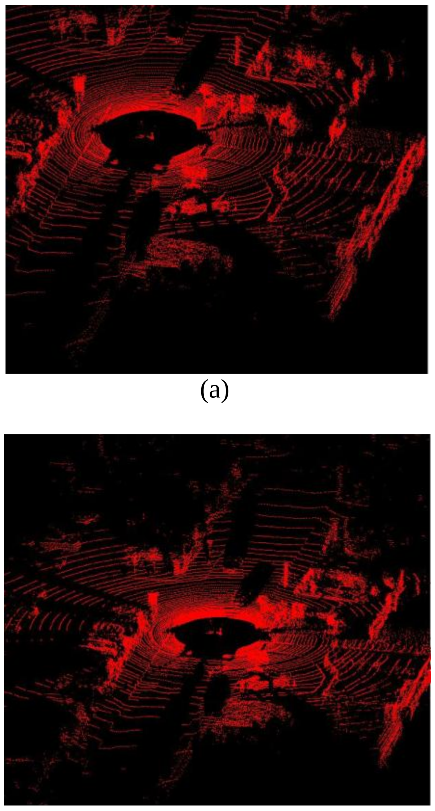


Figure 8. Point Clouds Data Projected to Panoramic Camera Set at Start Point.



(a)

(b)

Figure 9. (a) Objects in point clouds data set. (b) Predicted objects using KF in the next time step.

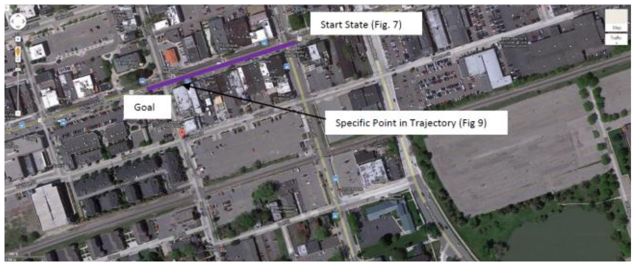


Figure 10. Vehicle Planned Trajectory Colored in Purple.

The planned trajectory presented in Figure 10 with a purple line. The starting point, marked as "S", presented in Figure 10, where the cloud points in this state are presented in Figure 8. An arbitrary state during the planned trajectory, which is marked with an arrow, is also presented in Figure 10, where point clouds prediction using KF in this state are presented in Figure 9. For this trajectory, $\propto = 1, \beta = 1$, robot velocity is set to $v_a = 10 \left[\frac{km}{hr}\right]$. In this case, the robot avoided two other cars, without handling cases of analytic optimal time solution for deadlocks with bounded velocity space.

VI. Conclusion and Future Work

In this research, we have presented an efficient trajectory planning algorithm for visible trajectories in a 3D urban environment for an Omni-directional model, based on an incomplete data set from LiDAR, predicting the scene at the next time step and approximating visibility.

Our planner is based on two steps visibility analysis in 3D urban environments using predicted visibility from point clouds data. The first step is to extract the basic geometric shapes: planes, cylinders and spheres, using RANSAC algorithms. The second step is a prediction of these geometric entities in the next time step, formulated as states vectors in a dynamic system using the Kalman Filter (KF).

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, generated by KF and RANSAC methods. Based on these fast computation capabilities, the on-line planner can approximate the most visible state as part of a greedy search method.

As part of our planner, we extended the classical VO method, where the velocity space is bounded and the robot velocity cannot escape from the velocity obstacles in the current state. We presented an escape mode based on an analytic time-optimal minimization problem which, for the first time, defines time horizon for these cases.

The visible trajectory is an approximated one, allowing us to configure the type of visible objects, i.e., roof or surfaces visibility of the trajectory, and can be used for different kinds of applications.

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 LiDAR.

VII. REFERENCES

[1] O.Gal, Y.Doytsher, Fast Visible Trajectory Spatial Analysis in 3D Urban Environments Based on Local Point Clouds Data, GeoProcessing 2017.

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

[3] J.-C. Latombe, Robot Motion Planning. Kluwer Academic Publishers, 1990.

[4] M. Erdman and T. Lozano-Perez, On multiple moving objects, Algorithmica, vol. 2, pp. 447–521, 1987.

[5] K. Fugimura and H. Samet, A hierarchical strategy for path planning among moving obstacles, IEEE Transactions on Robotics and Automation, vol. 5, pp. 61–69, 1989.

[6] L. Ulrich and J. Borenstien, Vfh+: Reliable obstacle avoidance for fast mobile robots, in Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1572–1577, 1998.

[7] N. Ko and R. Simmons, The lane-curvature method for local obstacle avoidance, in International Conference on Intelligence Robots and Systems, pp. 1615–1621, 1998.

[8] J. Minguez and L. Montano, Nearest diagram navigation. a new real-time collision avoidance approach, in International Conference on Intelligence Robots and Systems, pp. 2094–2100, 2000.

[9] T. Fraichard, Planning in dynamic workspace: a state-time space approach, Advanced Robotics, vol. 13, pp. 75–94, 1999.

[10] H. R. K. J.-C. Latombe and S. Rock, Randomized kinodynamic motion planning with moving obstacles, Algorithmics and Computational Robotics, vol. 4, pp. 247–264, 2000.

[11] O. Brock and O. Khatib, Real time replanning in high-dimensional configuration spaces using sets of homotopic paths, in Proceedings of the IEEE International Conference on Robotics and Automation, pp. 550–555, 2000.

[12] N. S. J. Minguez L. Montano and R. Alami, Global nearest diagram navigation, in Proceedings of the IEEE International Conference on Robotics and Automation, pp. 33–39, 2001.

[13] M.D. Feron and E. Frazzoli, Real time motion planning for agile autonomous vehicles, AIAA Journal of Guidance Control and Dynamics, vol. 25, pp. 116–129, 2002.

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

[15] T. Wikman and W. N. M.S. Branicky, Reflexive collision avoidance: a generalized approach, in Proceedings of the IEEE International Conference on Robotics and Automation, pp. 31–36, 1993.

[16] S. Lavalle. J. Kuffner, Randomized kinodynamic planning, International Journal of Robotics Research, vol. 20, pp. 378–400, 2001.

[17] T. Fraichard, A short paper about safety, in Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1140–1145, 2007

[18] S. P. T. Fraichard, Safe motion planning in dynamic environment, in International Conference on Intelligence Robots and Systems, pp. 885–897, 2005.

[19] T. Fraichard and H. Asama, Inevitable collision state-a step towards safer robots? Advanced Robotics, vol. 18, pp. 1001–1024, 2004.

[20] N. Chan and M. Z. J. Kuffner, Improved motion planning speed and safety using region of in- evitable collision, in ROMANSY, pp. 103–114, July 2008.

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

[22] Z. Shiller. F. Large and S. Sekhavat, Motion planning in dynamic environments: Obstacle moving along arbitrary trajectories, in Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3716–3721, 2001.

[23] H. Plantinga, and R. Dyer, Visibility, Occlusion, and Aspect Graph, The International Journal of Computer Vision, vol. 5, pp. 137-160, 1990.

[24] 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.

[25] F. Durand, 3D Visibility: Analytical Study and Applications, PhD thesis, Universite Joseph Fourier, Grenoble, France, 1999.

[26] W.R. Franklin, Siting Observers on Terrain, in Proc. of 10th International Symposium on Spatial Data Handling. Springer-Verlag, pp. 109–120, 2002.

[27] J. Wang, G.J. Robinson, and K. White, A Fast Solution to Local Viewshed Computation Using Grid-based Digital Elevation Models, Photogrammetric Engineering & Remote Sensing, vol. 62, pp. 1157-1164, 1996.

[28] J. Wang, G.J. Robinson, and K. White, Generating Viewsheds without Using Sightlines, Photogrammetric Engineering & Remote Sensing, vol. 66, pp. 87-90, 2000.

[29] C. Ratti, The Lineage of Line: Space Syntax Parameters from the Analysis of Urban DEMs', Environment and Planning B: Planning and Design, vol. 32, pp. 547-566, 2005.

[30] L. De Floriani, and P. Magillo, Visibility Algorithms on Triangulated Terrain Models, International Journal of Geographic Information Systems, vol. 8, pp.13-41, 1994.

[31] B. Nadler, G. Fibich, S. Lev-Yehudi, and D. Cohen-Or, A Qualitative and Quantitative Visibility Analysis in Urban Scenes, Computers & Graphics, vol. 5, pp. 655-666, 1999.

[32] B. Mederos, N. Amenta, L. Velho, L.H. Figueiredo, Surface reconstruction from noisy point clouds. In: Euro- graphics Symposium on Geometry Processing, pp. 53-62, 2005.

[33] J.P. Grossman, Point sample rendering. In: Rendering Techniques, pp. 181-192, 1998.

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

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

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

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

[38] 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.

[39] H. Boulaassal, T. Landes, P. Grussenmeyer, and 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), vol. 36, no. 3, 2007.

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