AuE 8200: Machine Perception and Intelligence

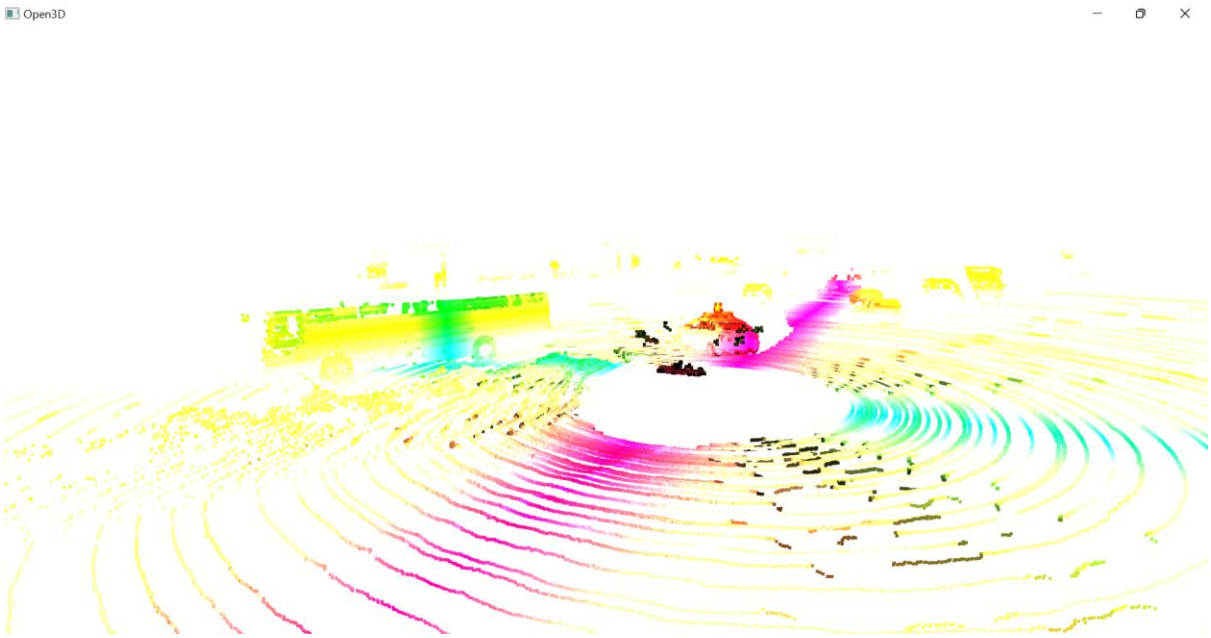
Homework 4

Submitted by: *- Priyanshu Rawat*

Question 1)

Select a frame (or a few frames) of LiDAR data file, parse the file and visualize the 3D point cloud of this frame, colored by its reflectivity value.

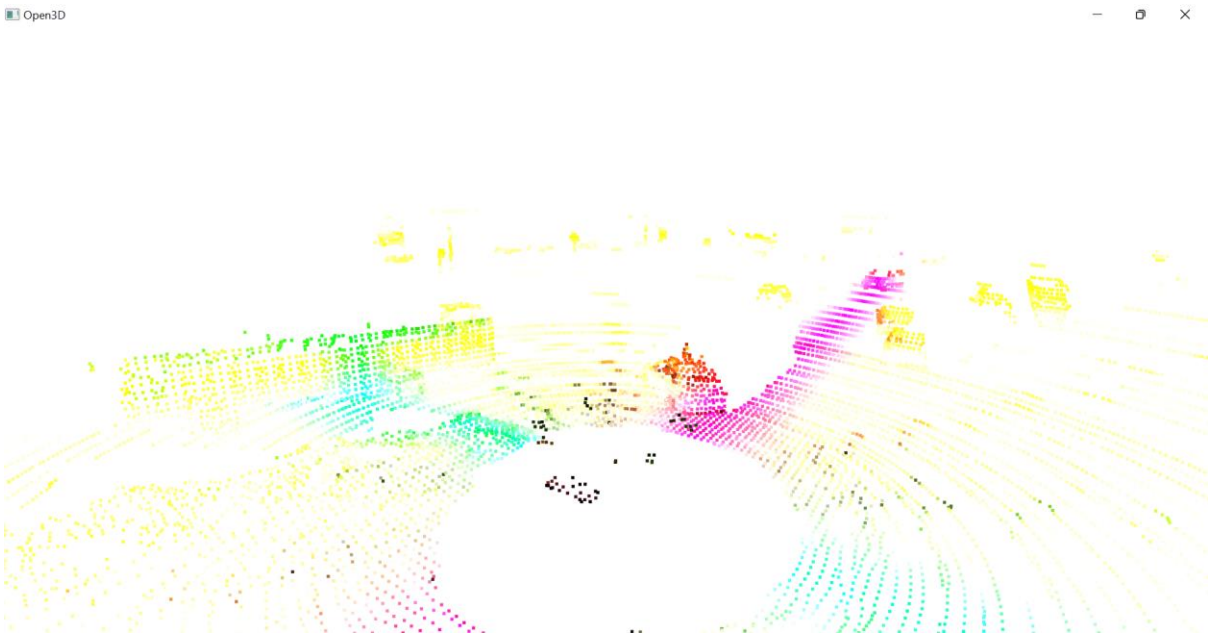
```
#### Question-1)

# Loading point cloud binary file
bin_pcd = np.fromfile("c:/clemson_university/2nd_semester/AuE_8200_Machine_Perception_and_Intelligence/hw4/LiDAR_datasets/anr

# Reshaping
bin_pcd_reshaped = bin_pcd.reshape((-1, 4))
color = np.zeros([len(bin_pcd_reshaped),3])
color[:,0] = abs(np.multiply(bin_pcd_reshaped[:,0], bin_pcd_reshaped[:,3])/255)
color[:,1] = abs(np.multiply(bin_pcd_reshaped[:,1], bin_pcd_reshaped[:,3])/255)
color[:,2] = abs(np.multiply(bin_pcd_reshaped[:,2], bin_pcd_reshaped[:,3])/255)

o3d_pcd = o3d.geometry.PointCloud(o3d.utility.Vector3dVector(bin_pcd_reshaped[:,:3]))
o3d_pcd.colors = o3d.utility.Vector3dVector(color)
o3d.visualization.draw_geometries([o3d_pcd])
```



Question 2)

Choose a 3-D resolution granularity, perform voxel filter (or box grid filter) to down-sample all the 3D point cloud points to the 3D voxel space points, and visualize the result points.

```
#### Question-2)

pcd = o3d.io.read_point_cloud("sample101.pcd") #Loading the pointcloud
voxel_pcd = pcd.voxel_down_sample(voxel_size=0.2) #Implementing voxel filter
o3d.io.write_point_cloud("sample101_voxel.pcd", voxel_pcd) #Saving the voxel pointcloud
o3d.visualization.draw_geometries([downpcd],
                                  zoom=0.3412,
                                  front=[0.4257, -0.2125, -0.8795],
                                  lookat=[2.6172, 2.0475, 1.532],
                                  up=[-0.0694, -0.9768, 0.2024])
```



Question 3)

- Apply RANSAC algorithm (or any others you prefer) to the 3D voxel space points to find a ground plane model. Print out your plane model parameter values result, visualize the plane with the points in the 3D (10 pts);

- Analyse the computational time complexity of this algorithm (5 pts).

- Remove all the ground planes points in the 3D voxel space points, visualize all the off-ground points in the 3D (5 pts);

Time Complexity

RANSAC or random sample consensus is an iterative method used for estimating the parameters of a mathematical model from observed data containing outliers.

k := 0

Repeat until P{better solution exists} < η (a function of C^* and number of steps k)

k := k + 1

I. Hypothesis

(1) select randomly set $S_k \subset U$, $|S_k| = m$

(2) compute parameters pk = $f(S_k)$

II. Verification

(3) compute cost $C_k = \sum_{x \in U} \rho(p_k, x)$

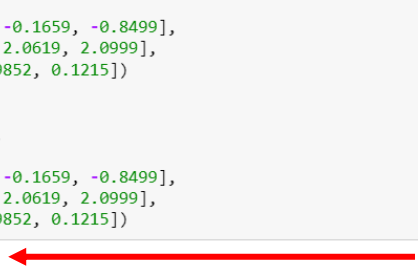
(4) if $C^* < C_k$ then $C^* := C_k$, $p* := p_k$

Time complexity for the RANSAC algorithm is given by: $J = k(t_M + N)$

The time complexity of the RANSAC algorithm depends mostly on the number of iterations even for a model with high number of outliers.

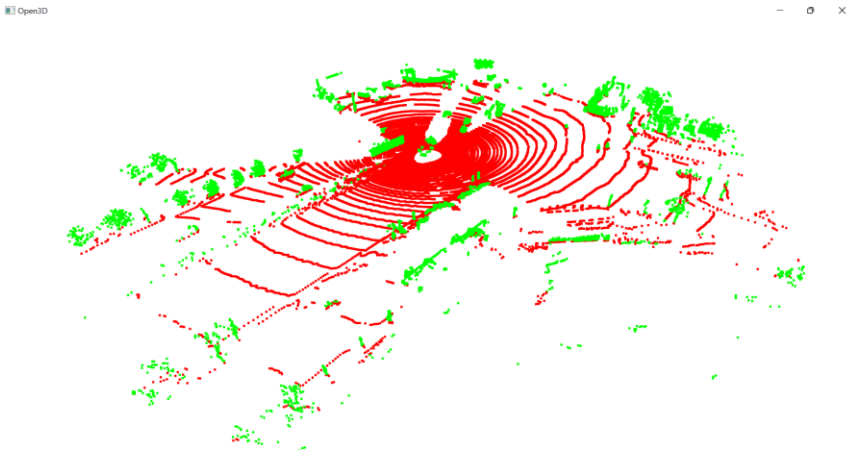
```python
#### Question-3)

pcd = o3d.io.read_point_cloud("sample101_voxel.pcd")
plane_model, inliers = pcd.segment_plane(distance_threshold=0.5,
                                         ransac_n=10,
                                         num_iterations=1000)
[a, b, c, d] = plane_model
print(f"Plane equation: {a:.2f}x + {b:.2f}y + {c:.2f}z + {d:.2f} = 0") #Printing the equation of the plane

inlier_cloud = pcd.select_by_index(inliers)
inlier_cloud.paint_uniform_color([1, 0, 0]) #Colorizing inliner points to 'red'
outlier_cloud = pcd.select_by_index(inliers, invert=True)
outlier_cloud.paint_uniform_color([0, 1, 0]) #Colorizing inliner points to 'green'

#Visualizing ground plane model in 3D
o3d.visualization.draw_geometries([inlier_cloud, outlier_cloud],
                                  zoom=0.8,
                                  front=[-0.4999, -0.1659, -0.8499],
                                  lookat=[2.1813, 2.0619, 2.0999],
                                  up=[0.1204, -0.9852, 0.1215])

#Visualizing all the off-ground points in 3D
o3d.visualization.draw_geometries([outlier_cloud],
                                  zoom=0.8,
                                  front=[-0.4999, -0.1659, -0.8499],
                                  lookat=[2.1813, 2.0619, 2.0999],
                                  up=[0.1204, -0.9852, 0.1215])
```

```
Plane equation: 0.03x + 0.01y + 1.00z + 1.43 = 0
```

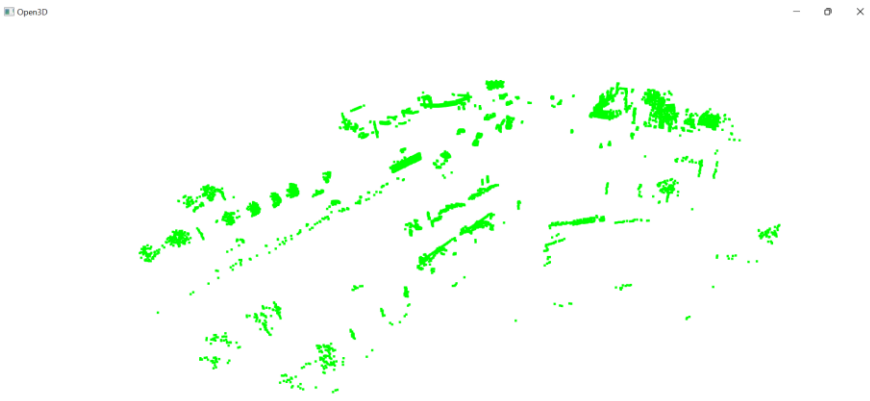


Ground plane model in 3D –

- Green points represent the outliers
- Red points represent the inliers



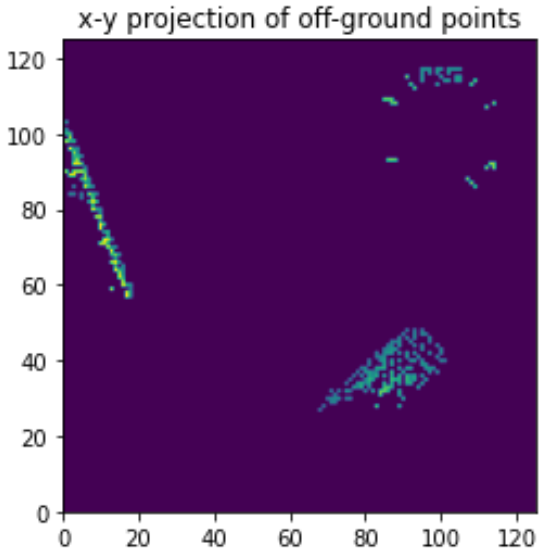
Off-ground points in 3D –



Question 4)

Perform a x-y projection to the off-ground points, and get a 2D matrix (you decide what is the element value), and visualize the 2D matrix as an image.

```
#### Question-4)

#Converting outlier point cloud data to -->> numpy array
outlier_points = np.asarray(outlier_cloud.points)

x_points = outlier_points[:, 0]
y_points = outlier_points[:, 1]
z_points = outlier_points[:, 2]

fwd_range = (-10., 10.)
side_range=(-10., 10.)
res=0.1
height_range=(-2., 2.)

# Filter to return only indices of points within desired cube
f_filt = np.logical_and((x_points > fwd_range[0]), (x_points < fwd_range[1]))
s_filt = np.logical_and((y_points > -side_range[1]), (y_points < -side_range[0]))
filter = np.logical_and(f_filt, s_filt)
indices = np.argwhere(filter).flatten()

x_points = x_points[indices]
y_points = y_points[indices]
z_points = z_points[indices]

# Coverting to pixel position values on the basis of resolution
x_img = (-y_points / res).astype(np.int32)  #x-axis is -y in LIDAR
y_img = (-x_points / res).astype(np.int32)  #y-axis is -x in LIDAR

x_img -= int(np.floor(side_range[0] / res))
y_img += int(np.ceil(fwd_range[1] / res))

pixel_values = np.clip(a=z_points,
                       a_min=height_range[0],
                       a_max=height_range[1])

pixel_values = (((pixel_values - height_range[0]) / float(height_range[1] - height_range[0])) * 255).astype(dtype="float32")

x_max = 1 + int((side_range[1] - side_range[0]) / res)
y_max = 1 + int((fwd_range[1] - fwd_range[0]) / res)
im = np.zeros([y_max, x_max], dtype=np.uint8)

# Populating image array with pixel values
im[y_img, x_img] = pixel_values

#Visualizing the result
plt.imshow(im)
plt.title("x-y projection to the off-ground points")
plt.xlim(0, 125)
plt.ylim(0, 125)
plt.show()
```

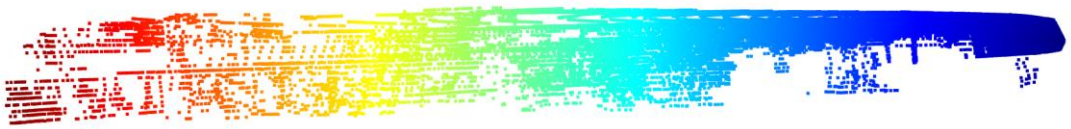


Question 5)

- **Based on the raw point cloud data (Questions 1), which is in Cartesian Coordinate, represent and visualize all the point cloud in Polar Coordinate (with horizontal and vertical angels and distance to the original) (5 pts).**

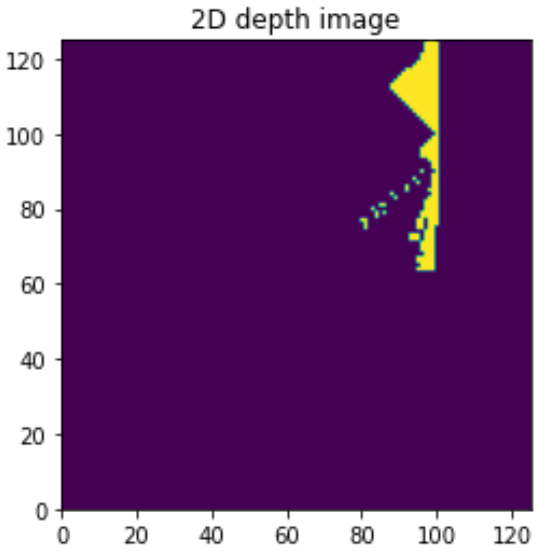
```
#### Question-5) a)

theta = []
phi = []
r = []

#Calculating r, theta and phi values
for k in range(0, len(bin_pcd_reshaped)):
    r.append(np.sqrt(bin_pcd_reshaped[k,0]**2 + bin_pcd_reshaped[k,2]**2 + bin_pcd_reshaped[k,2]**2))
    theta.append(np.arctan(bin_pcd_reshaped[k,1]/bin_pcd_reshaped[k,0]))
    phi.append(math.asin(bin_pcd_reshaped[k,2]/r[k]))

X = r * np.sin(phi) * np.cos(phi)
Y = r * np.sin(phi) * np.sin(phi)
Z = r * np.cos(phi)

a = np.zeros(shape=(len(X),3))

a[:,0] = X
a[:,1] = Y
a[:,2] = Z

o3d_pcd = o3d.geometry.PointCloud(o3d.utility.Vector3dVector(a[:,:3]))
o3d.io.write_point_cloud("sample101.pcd", o3d_pcd) #Saving the pointcloud
o3d.visualization.draw_geometries([o3d_pcd])
```

Open3D



- **Finally, generate the projected 2D depth image w.r.t horizontal and vertical angels, with intensity value using the distance. Visualize the 2D depth image (5 pts).**

```
#### Question-5) b)

x_points = a[:, 0]
y_points = a[:, 1]
z_points = a[:, 2]

fwd_range = (-10., 10.)
side_range=(-10., 10.)
res=0.1
height_range=(-2., 2.)

# Filter to return only indices of points within desired cube
f_filt = np.logical_and((x_points > fwd_range[0]), (x_points < fwd_range[1]))
s_filt = np.logical_and((y_points > -side_range[1]), (y_points < -side_range[0]))
filter = np.logical_and(f_filt, s_filt)
indices = np.argwhere(filter).flatten()

x_points = x_points[indices]
y_points = y_points[indices]
z_points = z_points[indices]

# Coverting to pixel position values on the basis of resolution
x_img = (-y_points / res).astype(np.int32)  #x-axis is -y in LIDAR
y_img = (-x_points / res).astype(np.int32)  #y-axis is -x in LIDAR

x_img -= int(np.floor(side_range[0] / res))
y_img += int(np.ceil(fwd_range[1] / res))

pixel_values = np.clip(a=bin_pcd_reshaped[:,3], #Distance as intentisy values from the pointcloud data
                       a_min=height_range[0],
                       a_max=height_range[1])

pixel_values = (((pixel_values - height_range[0]) / float(height_range[1] - height_range[0])) * 255).astype(dtype="float32")

x_max = 1 + int((side_range[1] - side_range[0]) / res)
y_max = 1 + int((fwd_range[1] - fwd_range[0]) / res)
im = np.zeros([y_max, x_max], dtype=np.uint8)

# Populating image array with pixel values
im[y_img, x_img] = pixel_values

#Visualizing the result
plt.imshow(im)
plt.title("x-y projection of off-ground points")
plt.xlim(0, 125)
plt.ylim(0, 125)
plt.show()
```



Question 6: Survey

3D Reconstruction of an Accident Scene

Importance

Reconstruction of an accident scene is helpful in determining the cause of the accident by applying laws of physics and analysing the evidences. Typically, reconstruction of accidents is performed by investigators, forensic experts, private consultants or law enforcement agencies to provide valuable information for purposes such as – fault determination, litigation, vehicle design and road safety. [1]

There are many tools, methods and processes to accurately accomplish this goal of reconstructing an accident involving a vehicle. Lately technologies like 3D LiDAR scanning have become immensely popular in mapping an accident scene. The upcoming sections of this survey discusses the accuracy and highlights the features of different techniques used to reconstruct an accident. Furthermore, it alludes the detailed benefits and shortcomings of most of these popular methods.

Challenges

In case of 3D laser scanners, the ambient light from the environment may blend with the laser interfering with the scan's accuracy. Moreover, depending on the severeness of this interference, the scan might be noisy and almost unusable. Hence, the lighting in the environment is an essential factor affecting the quality of the 3D scan.

Adding on, depending the optical nature of the 3D scanning tool, it might be impossible to record surfaces which are out of range of the scanner's line of sight. This implies, that multiple 3D scanning tool should be placed in accordance with their range to accurately map the 3D accident. Also, to capture the internal geometries of vehicles or environment scans are to be taken from different angles to gather sufficient data about the accident. [2]

Lastly, another challenge in reconstruction of accidents is the involvement of autonomous vehicles. It is hard to predict the decision made by an autonomous vehicle just before the occurrence of accident. The main challenges for such vehicles for instance are misjudgements, incorrect evasive manoeuvres, or mistakes made by other automated cars.

Existing Approaches

To map an accident scene in the form of x, y, and z coordinates – the data in the form of 'point cloud' is used. Existing approaches for 3D accident reconstruction involve any of the following [3]:

1. Digital photogrammetry
2. LiDAR 3D scanning
3. IR or structured light scanning

LiDAR scanning is the most accurate and expensive method out of the three to map an accident scene in 3D. The laser scanner works by sending millions of light pulses, these pulses after bouncing from the surfaces of surrounding objects return to the sensor. The sensor then measures the time it took for each pulse to travel back to find the geometry of the scene.

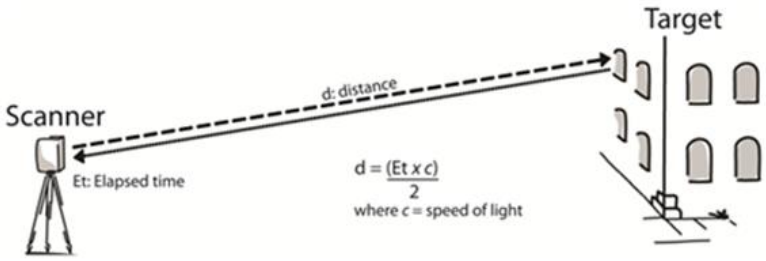


Figure-1:

LiDAR Scanning [4]

In the elementary form, photogrammetry has been around longer than scanning. By knowing the size of a reference object in an image and the location a picture was clicked, the position and sizes of object around it can be estimated using triangulation. The algorithm of triangulation greatly increases the accuracy, speed, and reliability of photogrammetry in 3D accident reconstruction.

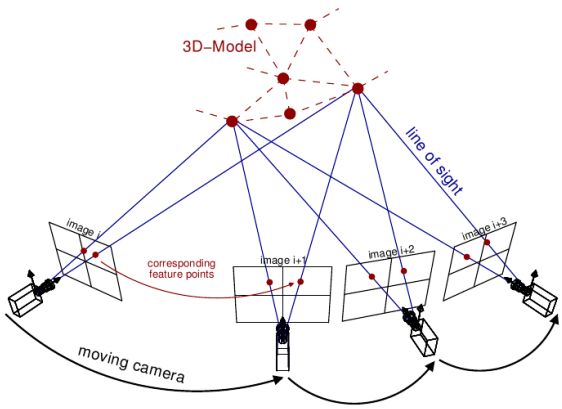


Figure-2: Photogrammetry Scanning [5]

Laser Scanning	Photogrammetry
Accuracy within ¼th of an inch	Accuracy is lower compared to laser scanners
Error rate is fixed depending on the capabilities of equipment	Errors can be high when dealing with reflective and transparent surfaces
Automated process, once laser is positioned and started	Less scope for automation giving rise to operator induced errors
Accurate for large spaces	Accuracy in large spaces depends on lens
Fuzzy point cloud on reflective or high textured surfaces	Visually better representation of textures in objects

Besides, laser scanning and photogrammetry, IR (Infrared) scanning can also used for 3D reconstruction of an accident. Using projected light and a camera system to shoot light onto the objects, creating a 'line of light', distortions in the line of light are recorded to recreate the geometrical surfaces' of objects. At any given moment, there are many stripes of light, producing distortions depending on the object's geometry. Also, since the stripes of light are parallel to each other, the 3D coordinates of object geometry can be easily estimated.

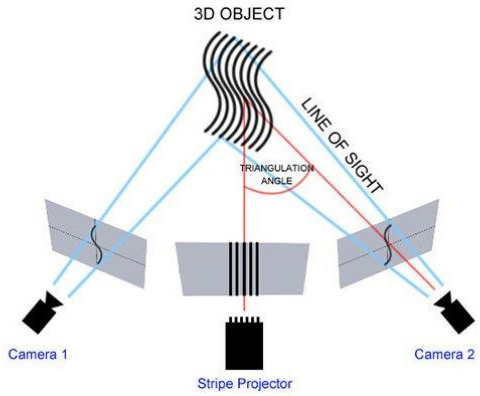


Figure-3: Structured Light Scanning [6]

Existing problems of these existing approaches

Most of these existing approaches for 3D accident reconstruction are time consuming and include bulky equipment. Moreover, the accuracy of these existing methods depends highly on the environmental factors like – weather, sunlight, fog etc. Besides, they also depend on the geometry and surfaces of the object or vehicles in the scene. For instance, a reflective or transparent surface can cause inaccuracy in measuring the geometry of surfaces.

To counter some of these drawbacks, professionals are now using SLAM-based handheld LiDAR scanners. These solutions are becoming immensely popular because of their accuracy, speed, low cost and ease of use. [7]

References

1. https://www.evuonline.org/attachments/article/3071/EVU2020-02-21%20Paula%20Benefits.pdf
2. https://3space.com/advantages-disadvantages-of-3d-laser-scanning/
3. https://www.engineering.com/story/3d-scanning-understanding-the-differences-in-lidar-photogrammetry-and-infrared-techniques
4. https://www.forensicdjs.com/blog/lidar-just-what-is-it/
5. https://www.researchgate.net/figure/Structure-from-Motion-SfM-photogrammetric-principle-Source-Theia-sfmorg-2016_fig3_303824023
6. https://www.3dnatives.com/en/laser-3d-scanner-vs-structured-light-3d-scanner-080820194/
7. https://lidarmag.com/2019/11/20/the-past-and-future-of-handheld-3d-scanning/