Obstacle Detection using LIDAR

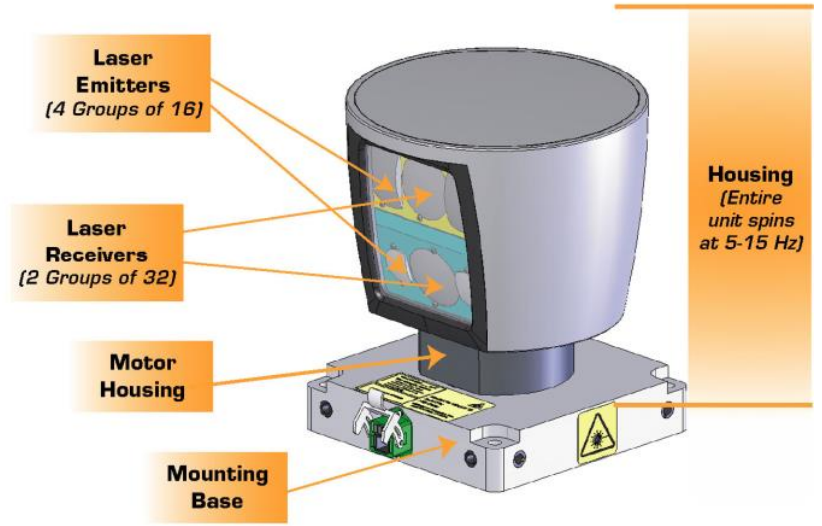
RAJNISH BHUSAL

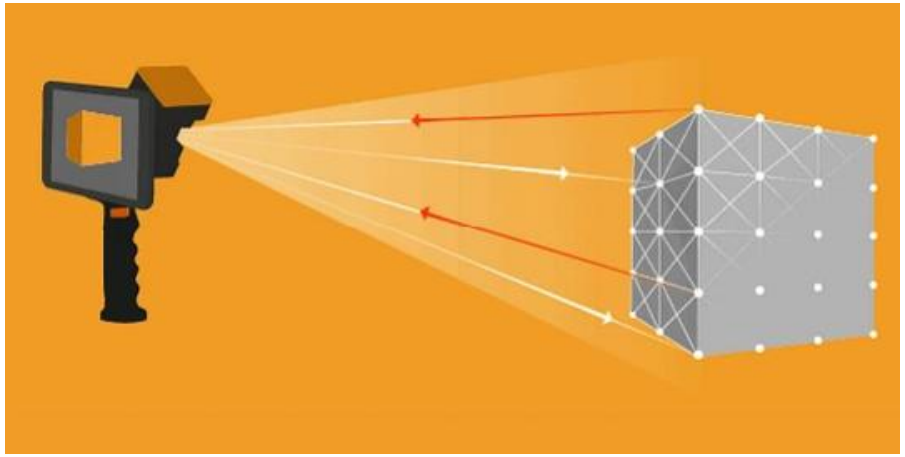
Lidar

- **LI**ght **D**etection **A**nd **R**anging
- Remote sensing (Passive sensor) method

- Emission of thousands of laser rays at different angles
- Reflected-off of obstacles
- Detected using Receivers

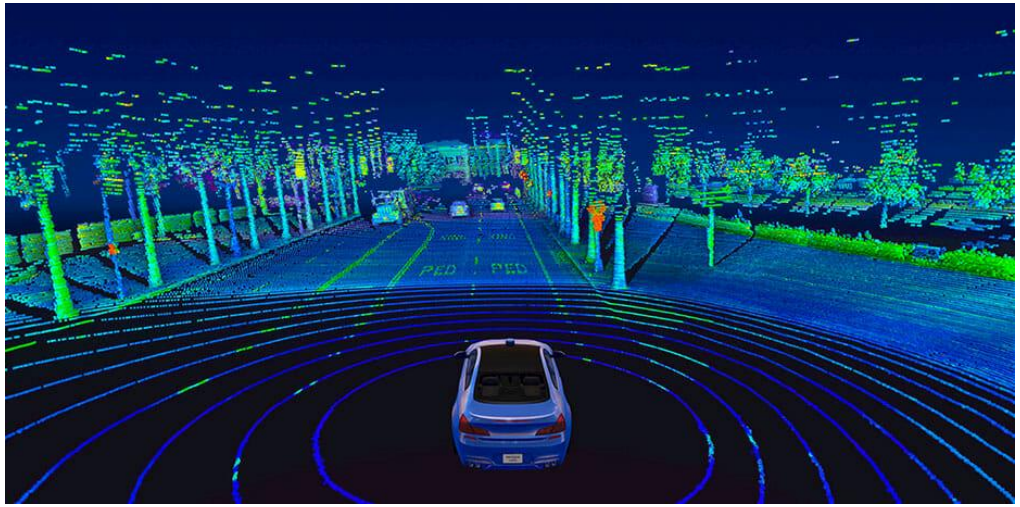
- Measures time-of-flight

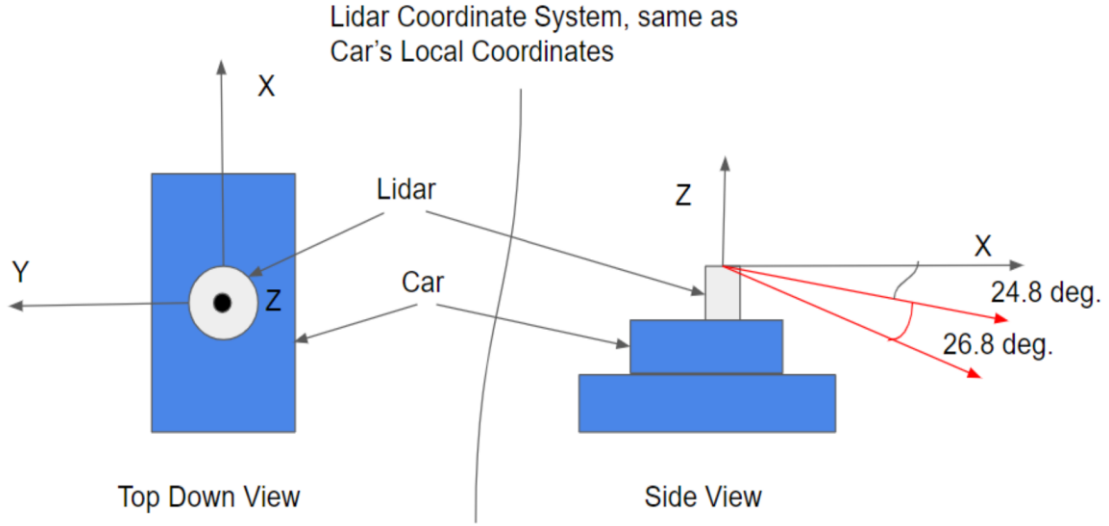




Point Cloud

❖ **Point Cloud:** set of all lidar reflections measured





Point Cloud Data (PCD)

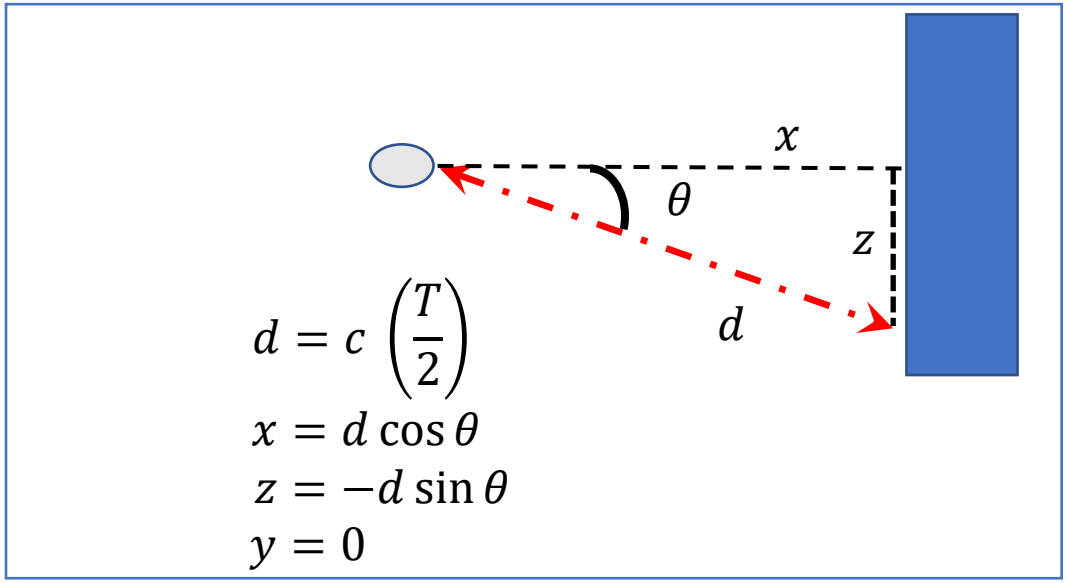
❖ Lidar data is stored in a format called PCD

❖ **(x,y,z) coordinates**

❖ **Laser intensity value**: to evaluate material properties

.pcd file

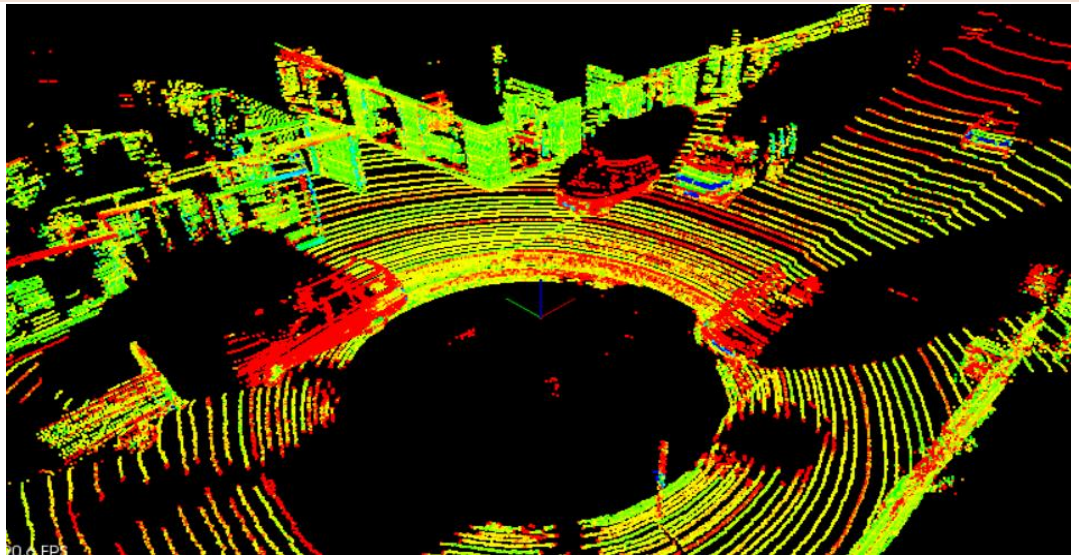
(x, y, z, I),
(x, y, z, I),
……
(x, y, z, I)



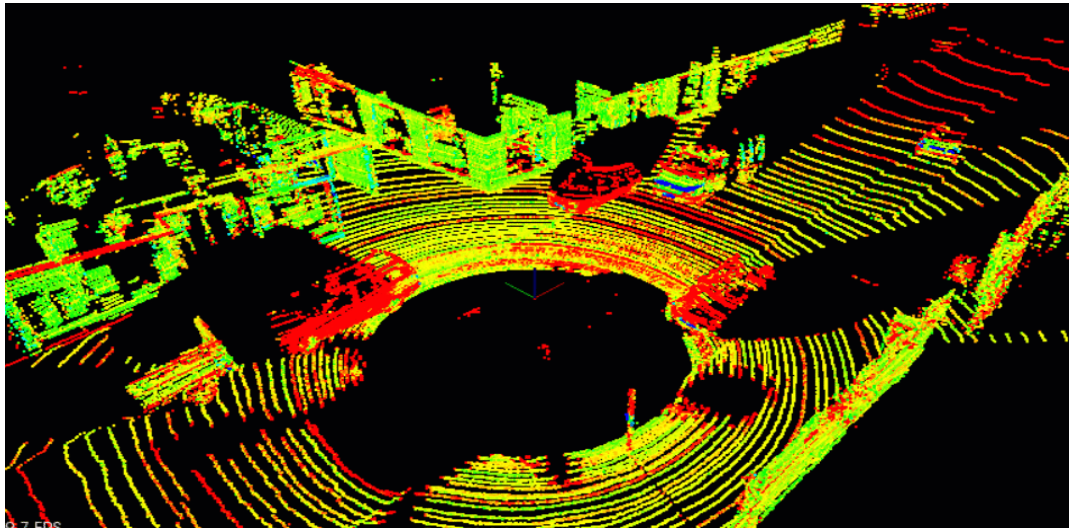
Point Cloud Processing for Obstacle Detection using Lidar

Steps for Obstacle Detection

1. **Filtering**
2. **Segmentation**
3. **Clustering**

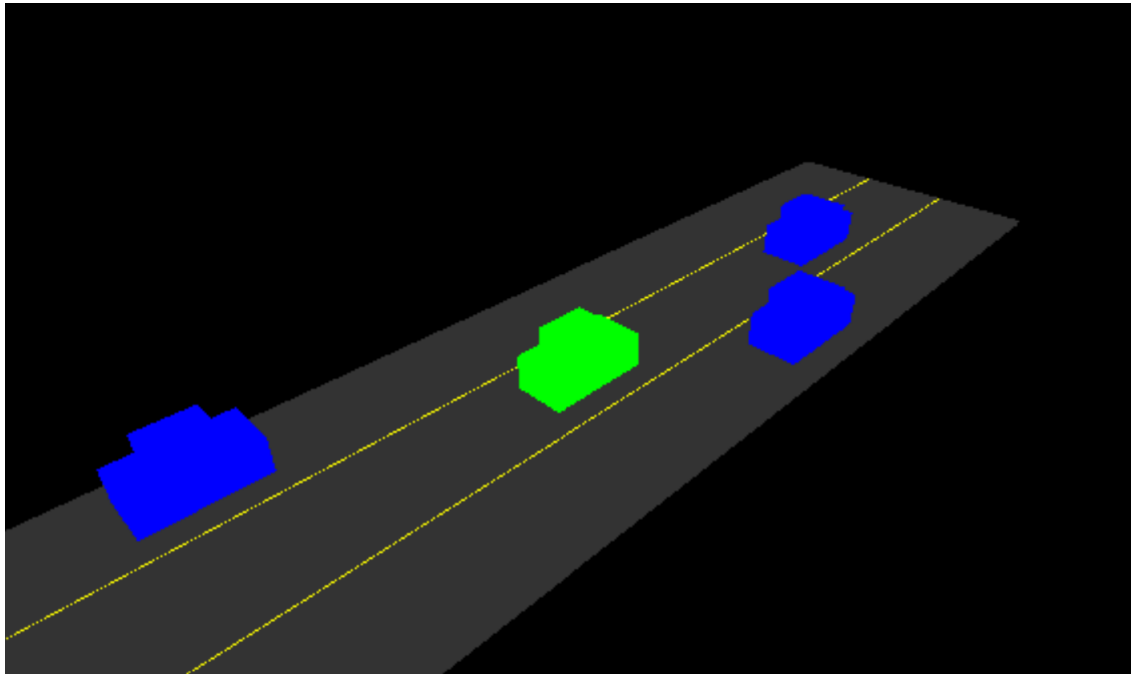


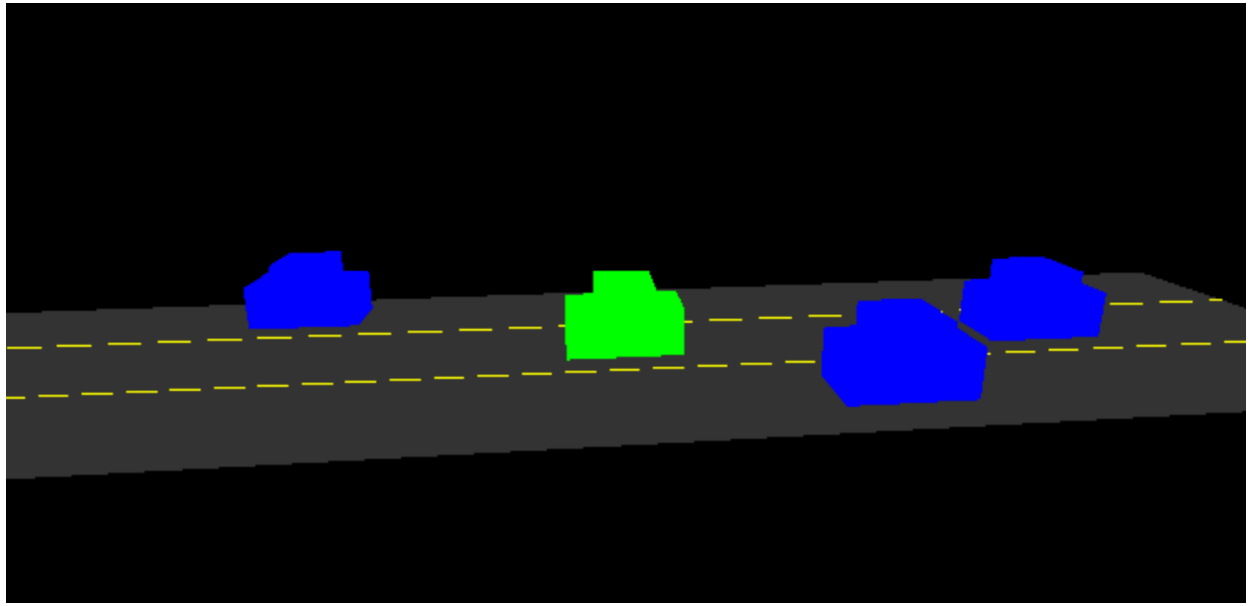
PCD of a city block



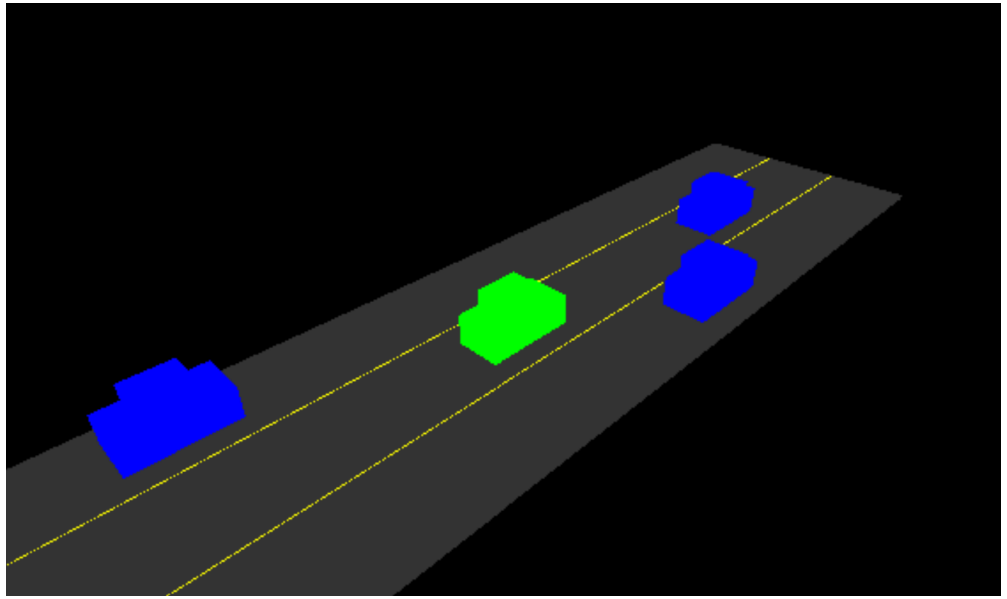
- **Black spot is where the car with the lidar sensor is located.**
- **Intensity values are being shown as different colors.**

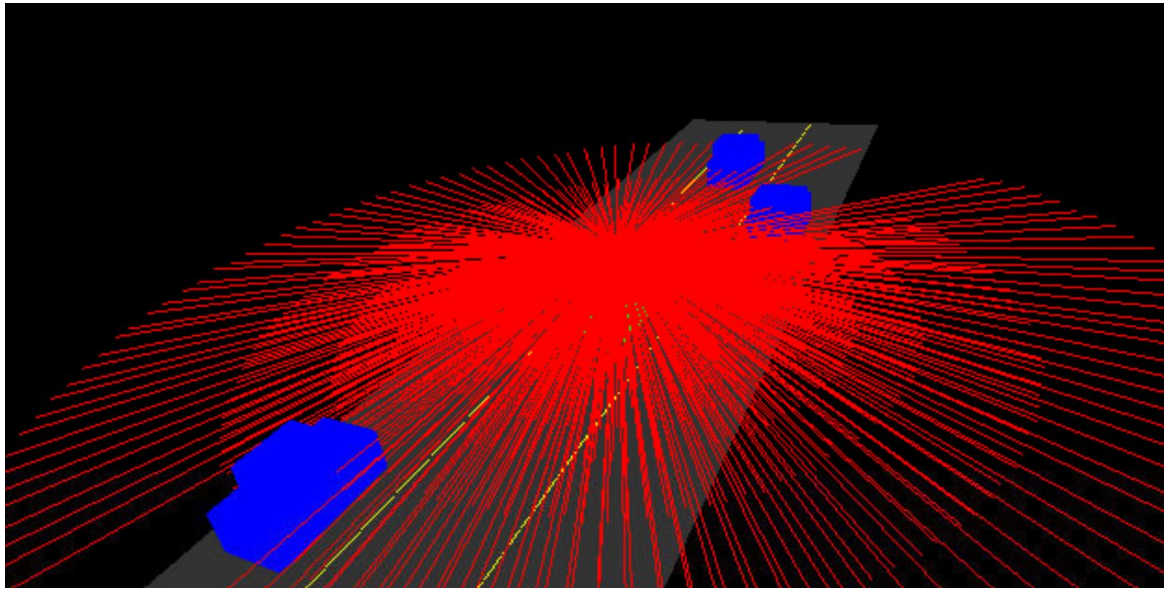
Simulation Environment: Autonomous Car in a Highway

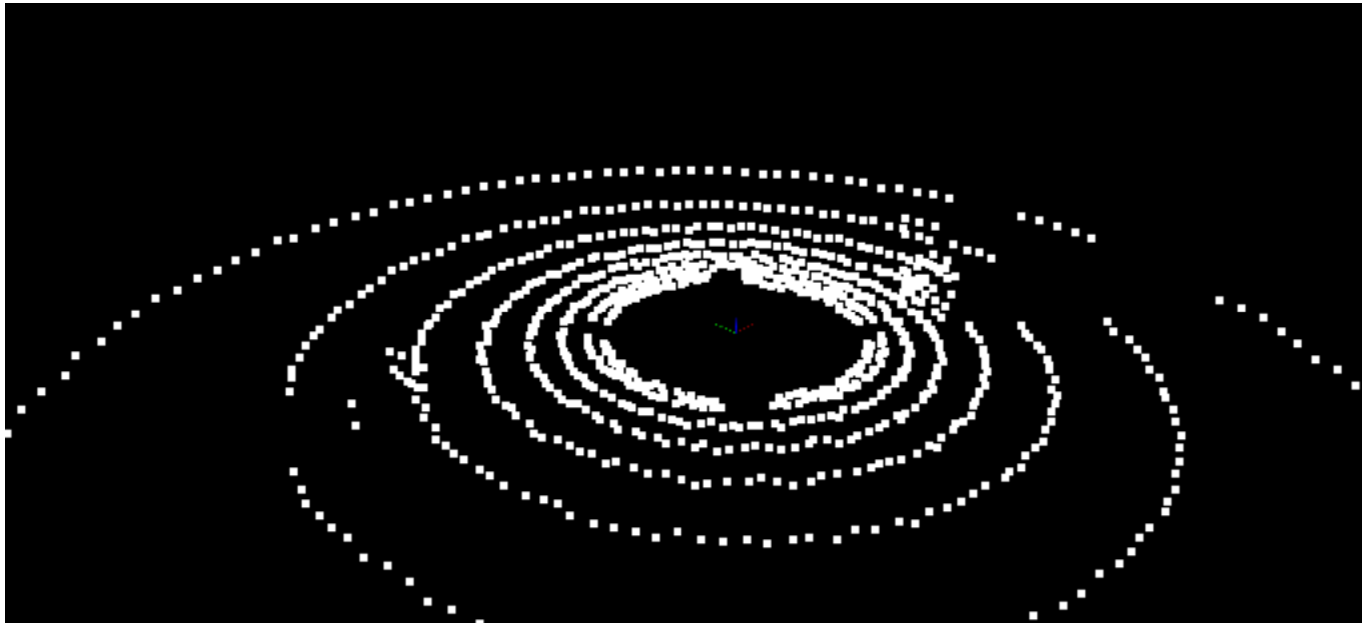




Autonomous Car with Lidar

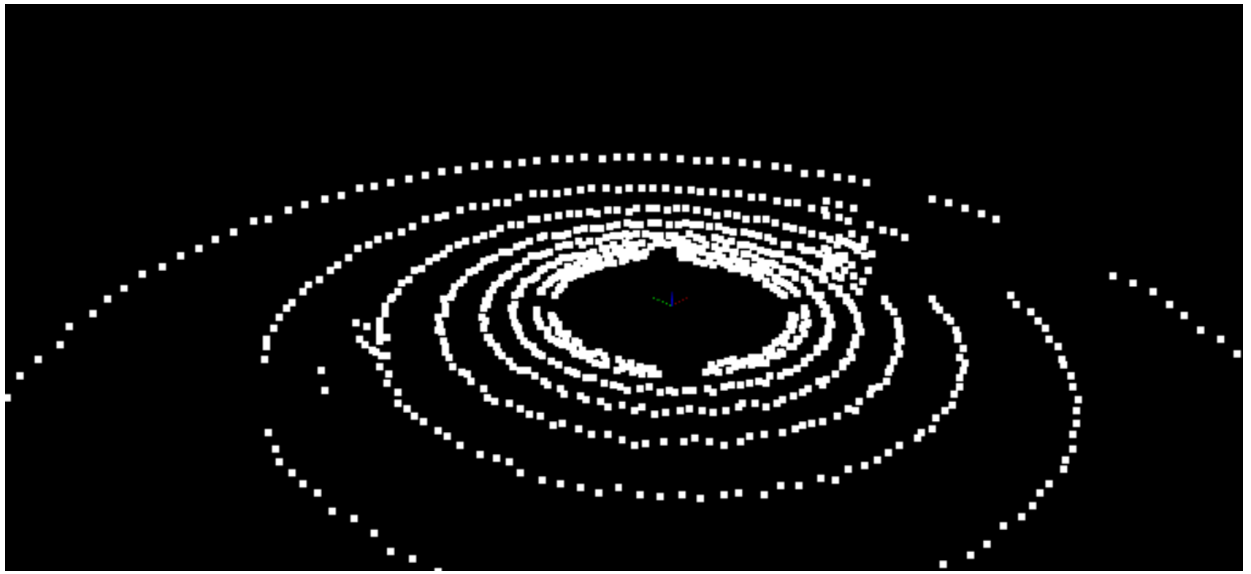


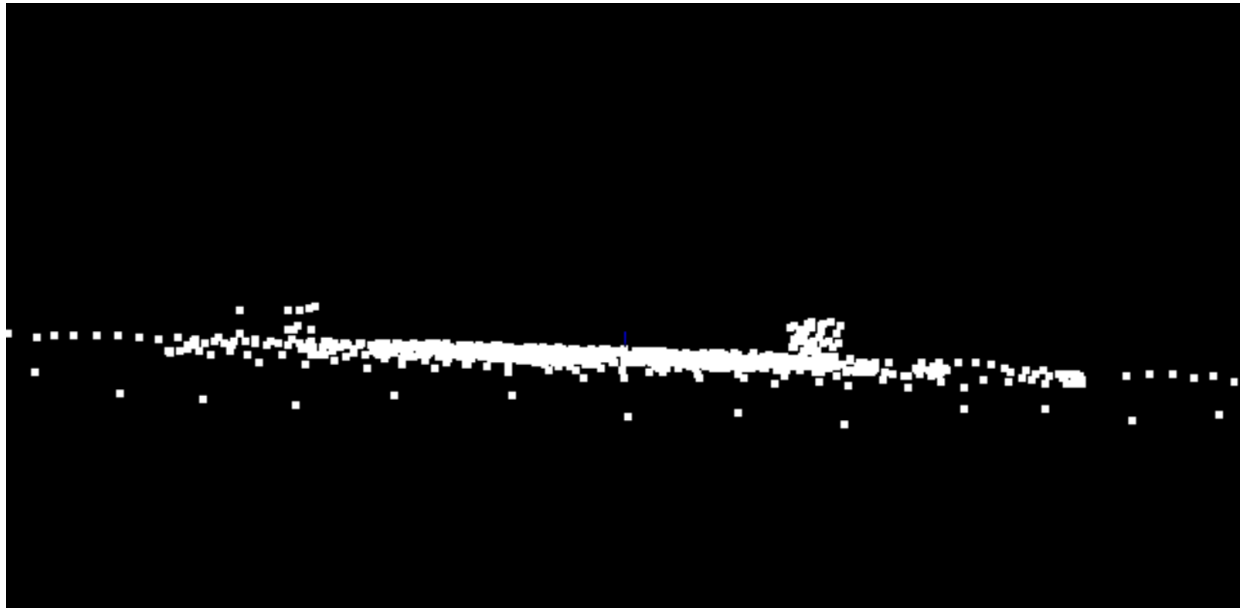




Point Cloud Segmentation

- **Our Objective**: Locate Obstacles in the scene
- Not all points in the point cloud are related to obstacles
- **What objects appear in the PCD but are not obstacles?**
- Any free space on the road is not an obstacle
- Flat road: separate road points from non-road points
- **Planar Segmentation**





Planar Segmentation using RANSAC Algorithm

- RANSAC: **RA**ndom **SA**mple **C**onsensus
- Iterative Approach

RANSAC for fitting a line in 2D

- For each iteration
  - Randomly sample two points from point cloud

$$(x_1, y_1), (x_2, y_2)$$

  - Fit a line between two points

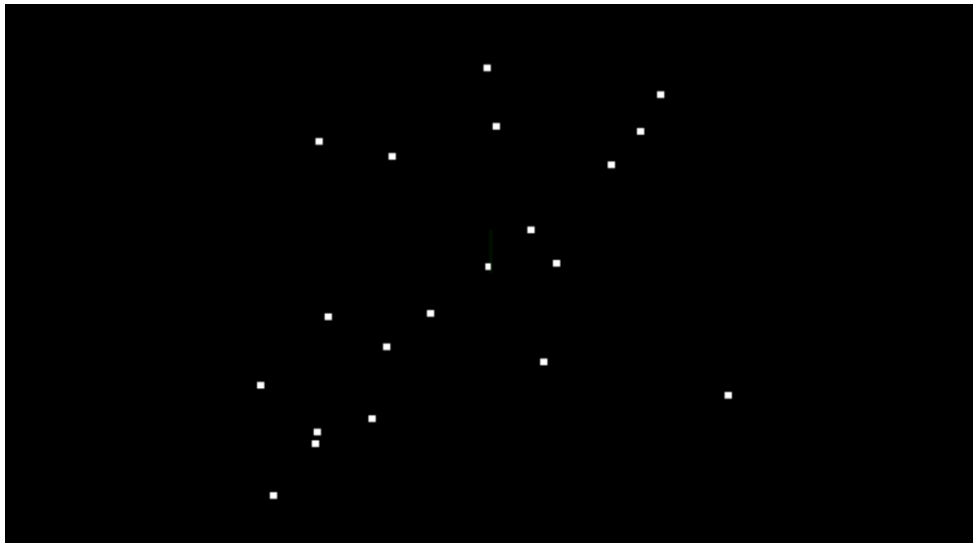
$$Ax + By + C = 0$$
$$A = y_1 - y_2$$
$$B = x_2 - x_1$$
$$C = x_1 y_2 - x_2 y_1$$

  - Calculate d from every point (x_0, y_0) to fitted line

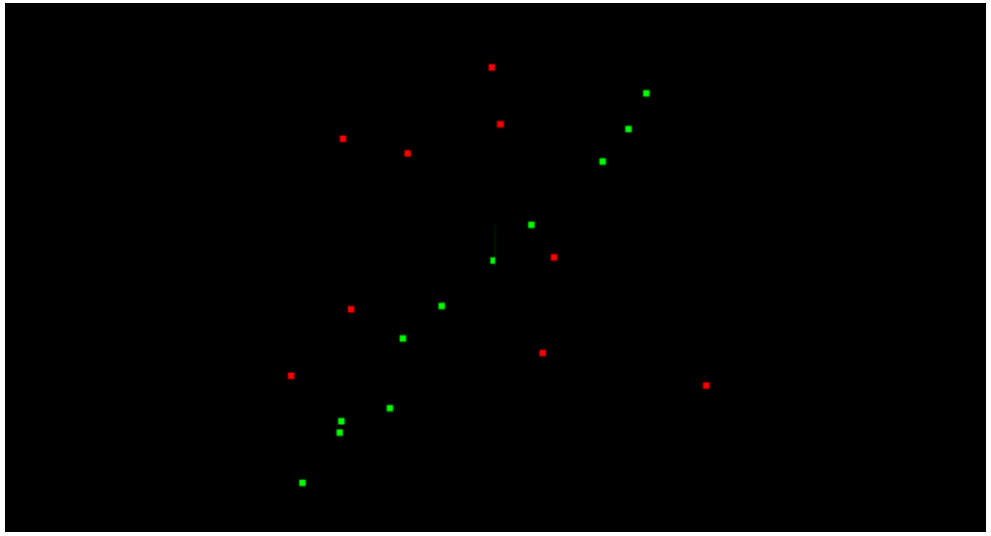
$$d = \frac{|A\,x_0 + B\,y_0 + C|}{\sqrt{A^2 + B^2}}$$

  - Inlier: if $d \leq$ tolerance

Example Problem



Segmentation using RANSAC



Planar Segmentation using RANSAC Algorithm

- RANSAC: **RA**ndom **SA**mple **C**onsensus
- Iterative Approach

RANSAC for fitting a plane in 3D

- For each iteration
  - Randomly sample **three** points from point cloud

    $(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)$

  - Fit a **plane** between three points

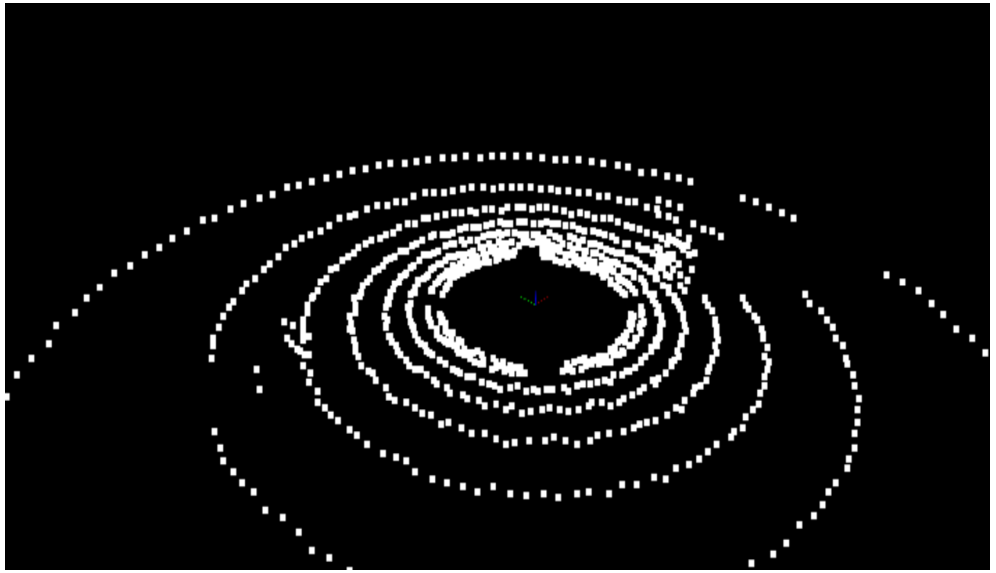
    $$Ax + By + Cz + D = 0$$

    $$A = (y_2 - y_1)(z_3 - z_1) - (z_2 - z_1)(y_3 - y_1)$$
    $$\mathrm{B} = (z_2 - z_1)(x_3 - x_1) - (x_2 - x_1)(z_3 - z_1)$$
    $$\mathrm{C} = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$$
    $$D = -(Ax_1 + By_1 + Cz_1)$$

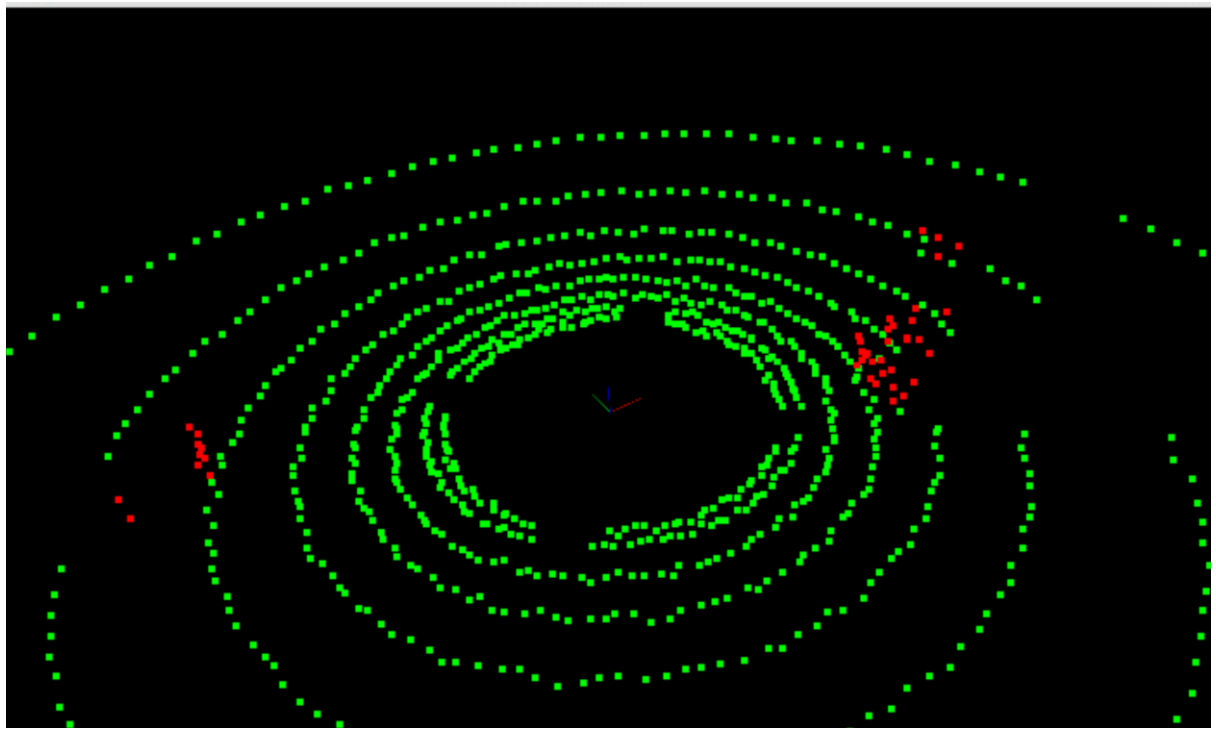
  - Calculate d from every point (x_0, y_0) to fitted plane

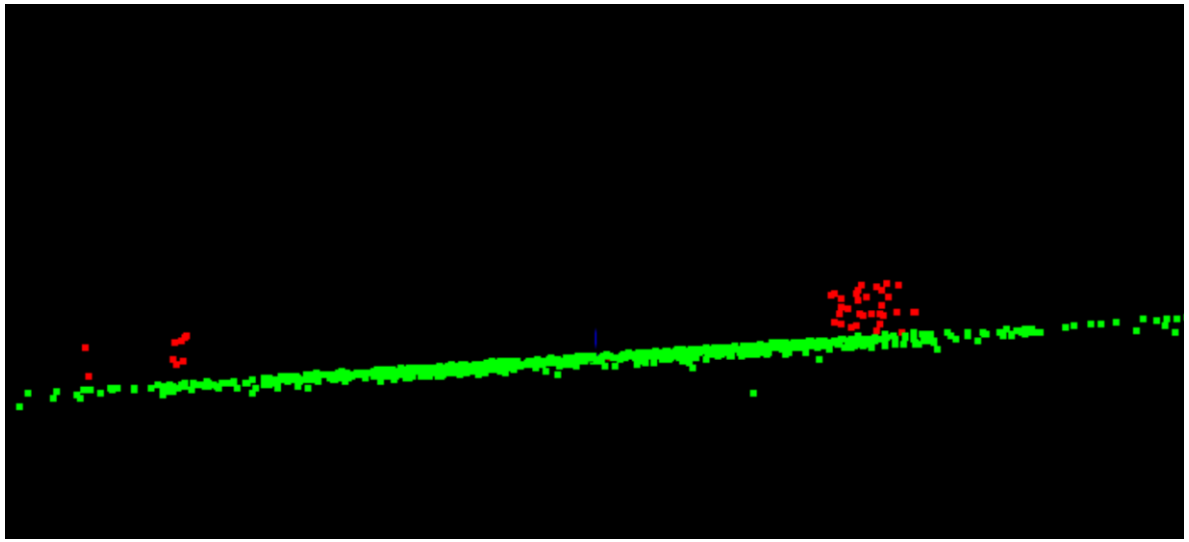
    $$d = \frac{|A\,x_0 + B\,y_0 + C\,z_0 + D|}{\sqrt{A^2 + B^2 + C^2}}$$

  - Inlier: if $d \leq$ tolerance



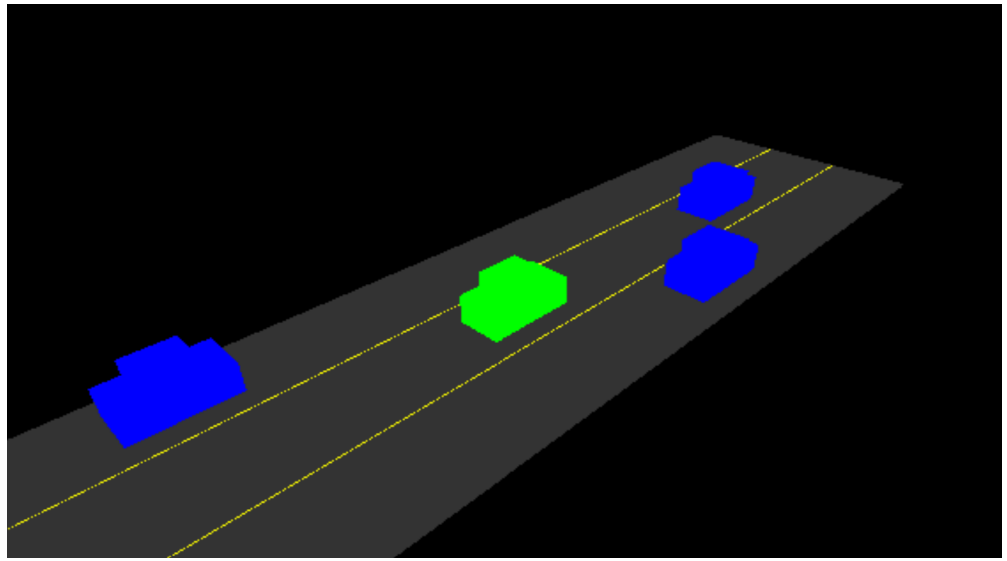
Segmentation Results



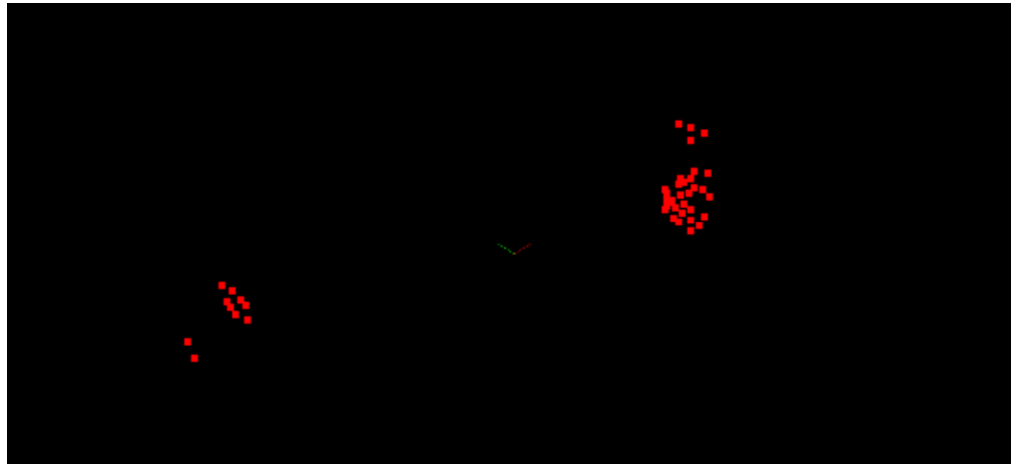


Clustering

- We now know the points which are associated with obstacles
- **Can we separate out and group those obstacle points?**
- Useful for **multiple object tracking** with cars, pedestrians, bicyclists
- Grouping and Cluster Point Cloud Data using "**Euclidean Clustering**"



Simulation Environment



Results from Segmentation

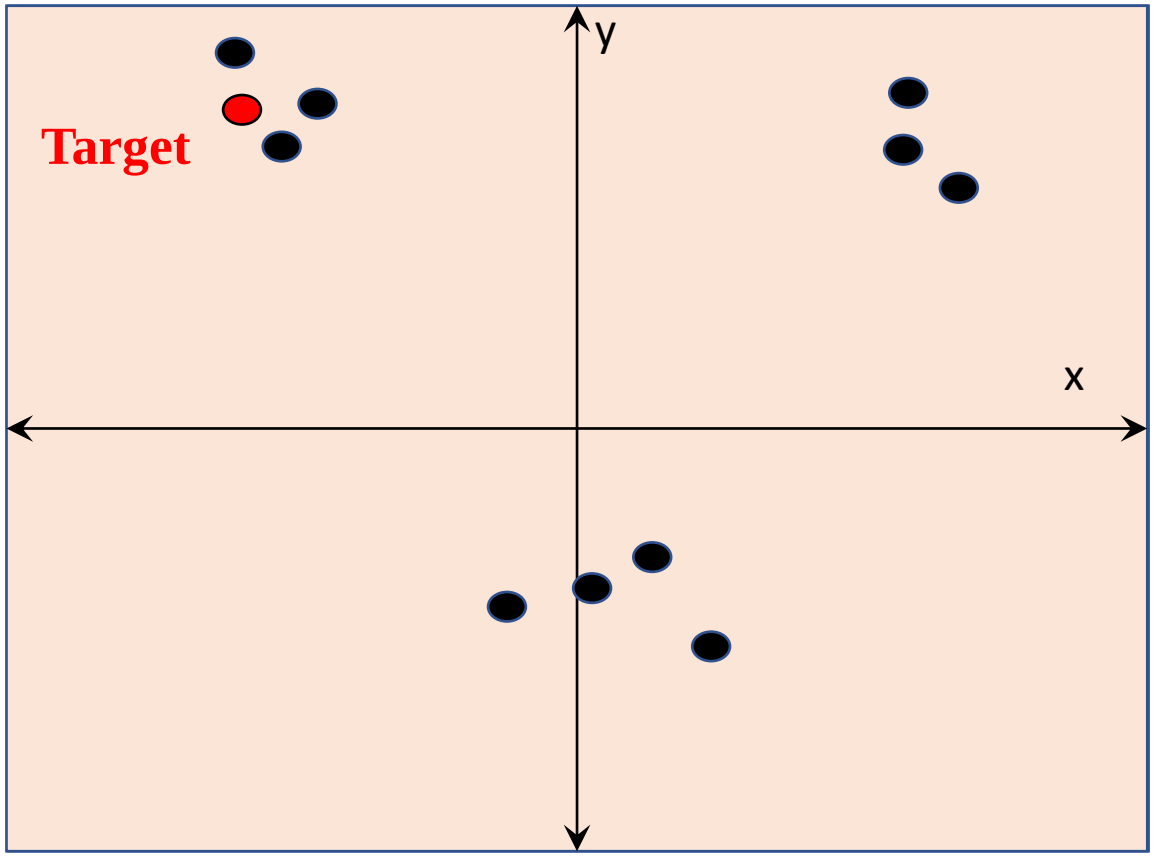
Euclidean Clustering

Idea

- Associate group of points by finding "**how close they are**" – nearest neighbors

- **Efficiency** of algorithm to do a nearest neighbor search matters !! – millions of points

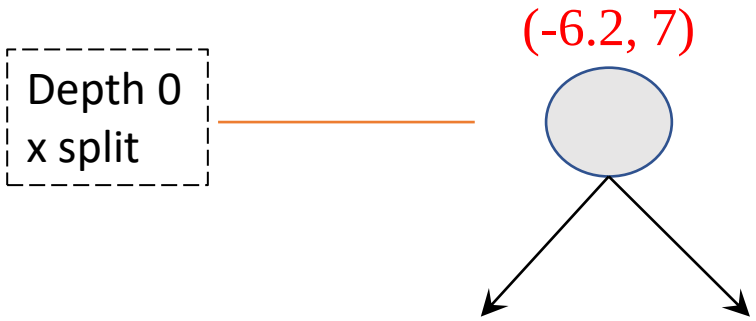
- Euclidean Clustering using KD Tree) data structure (binary search tree for points in k-dimensions

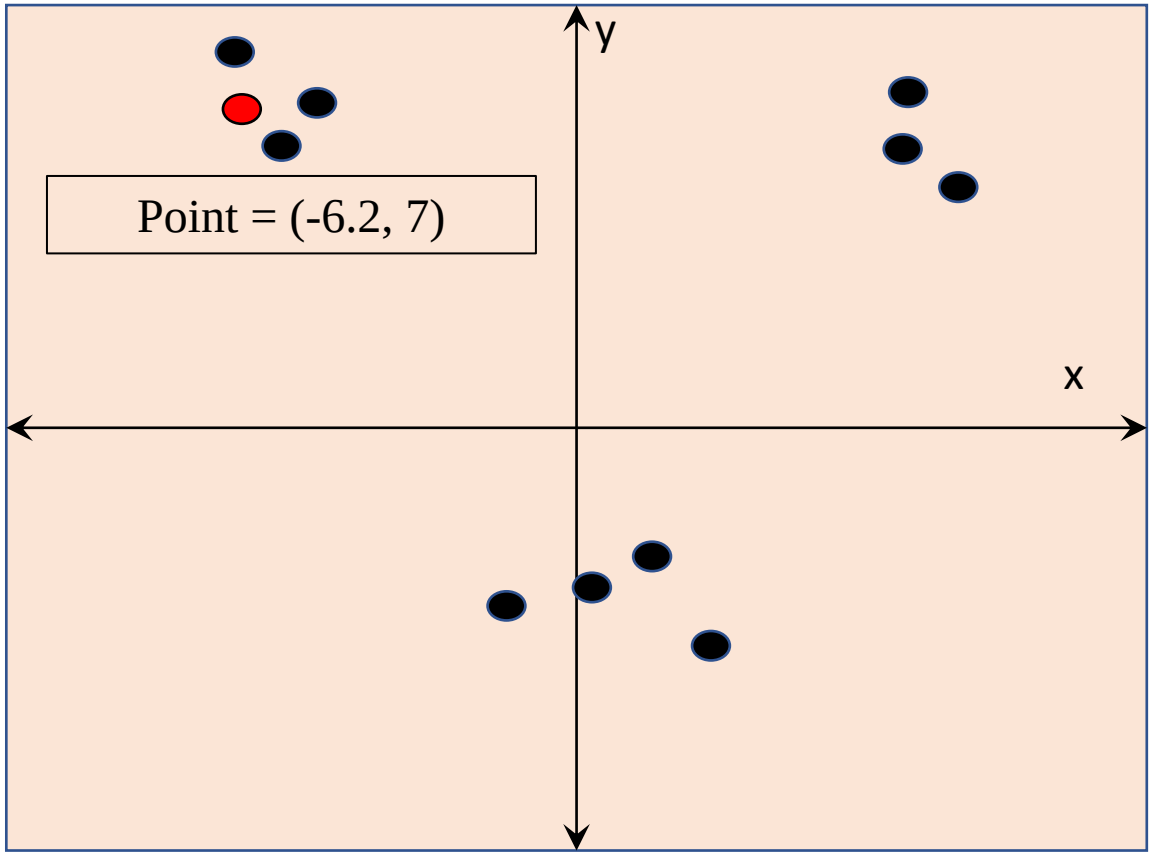


KD Tree Data Structure: Creation

KD Tree

- Binary Search Tree that splits points between alternating axes

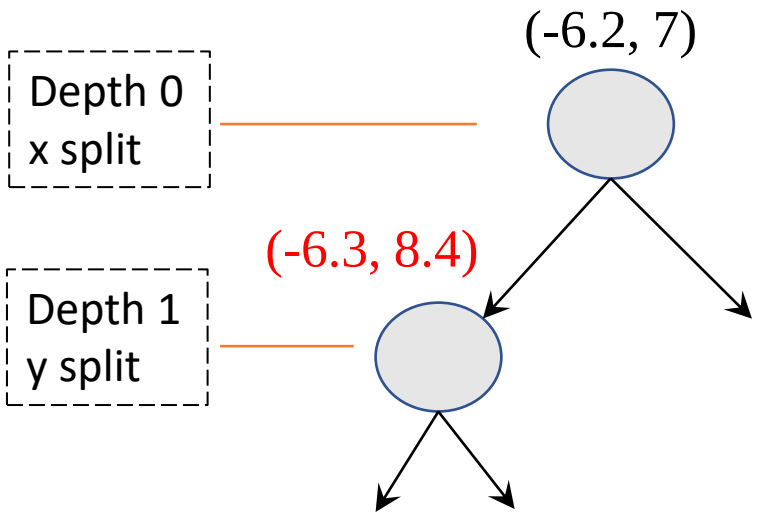


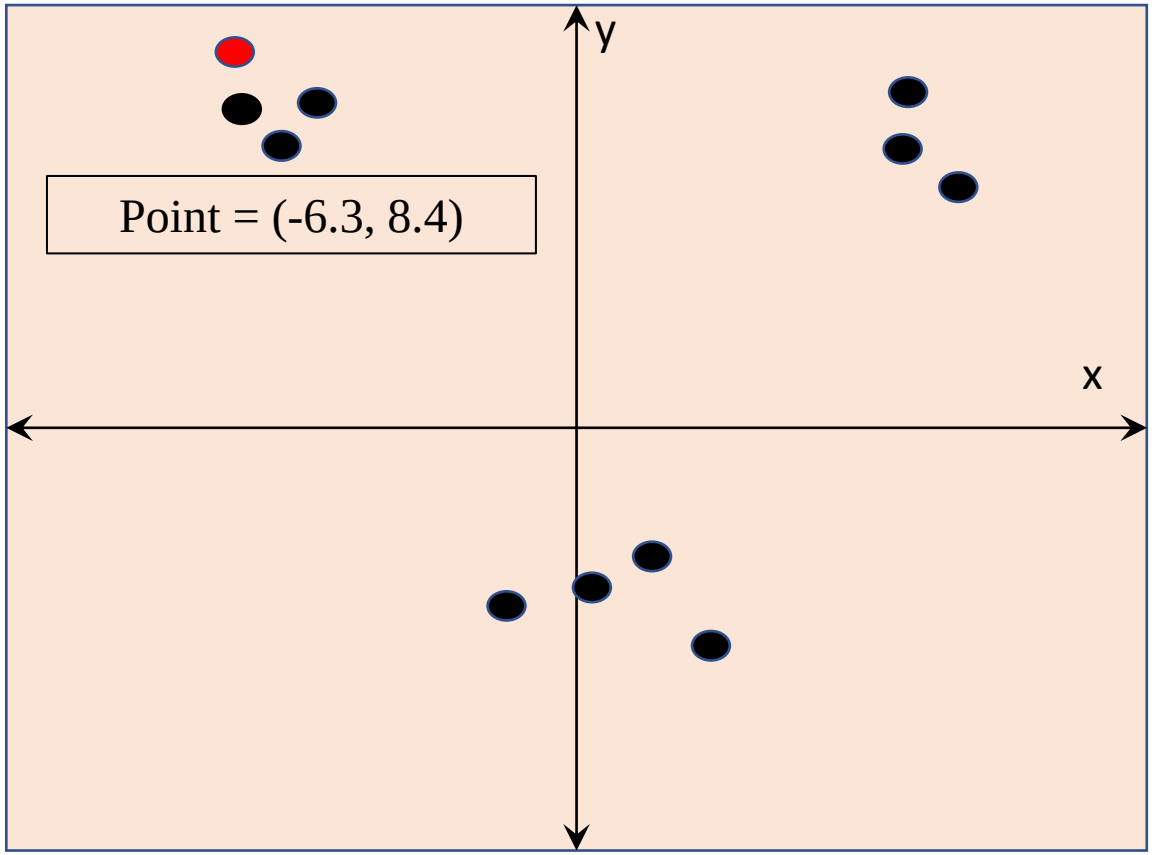


KD Tree Data Structure : Creation

KD Tree

- Binary Search Tree that splits points between alternating axes

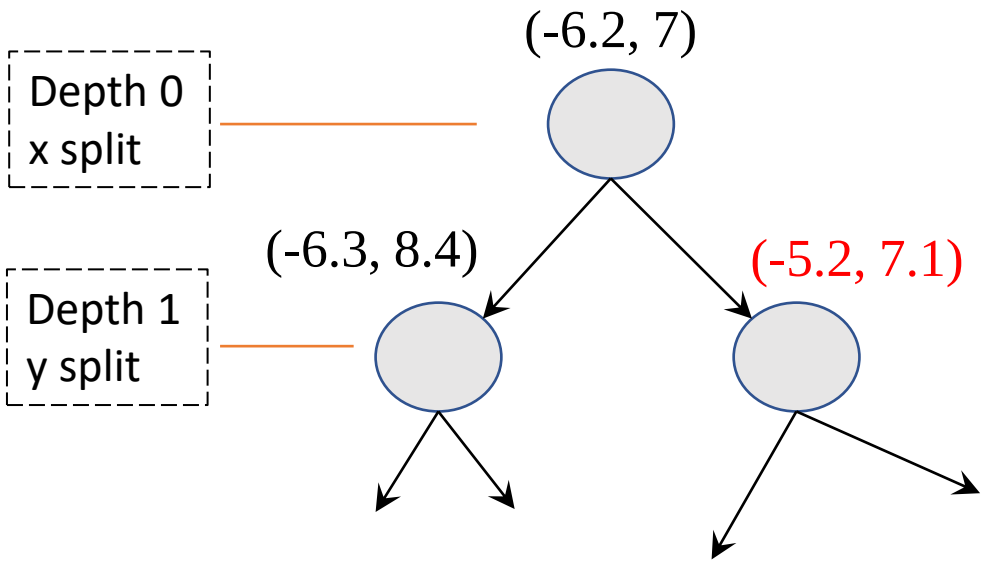


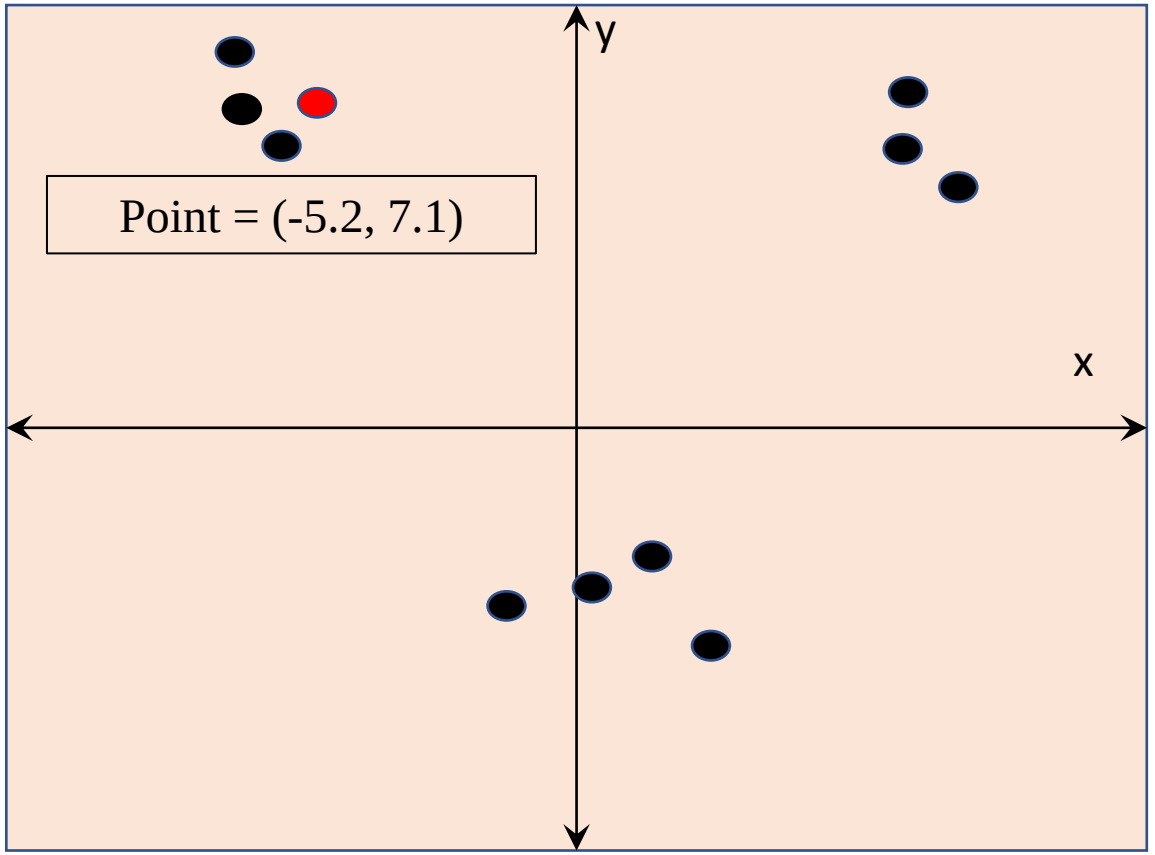


KD Tree Data Structure : Creation

KD Tree

- Binary Search Tree that splits points between alternating axes

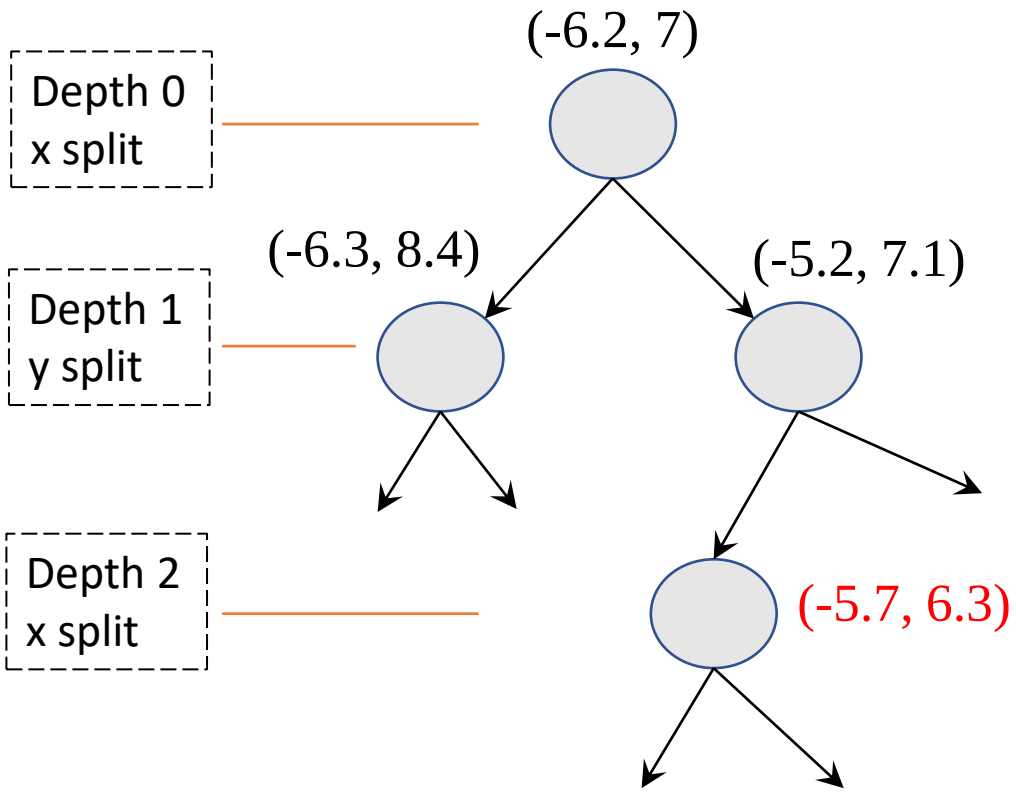


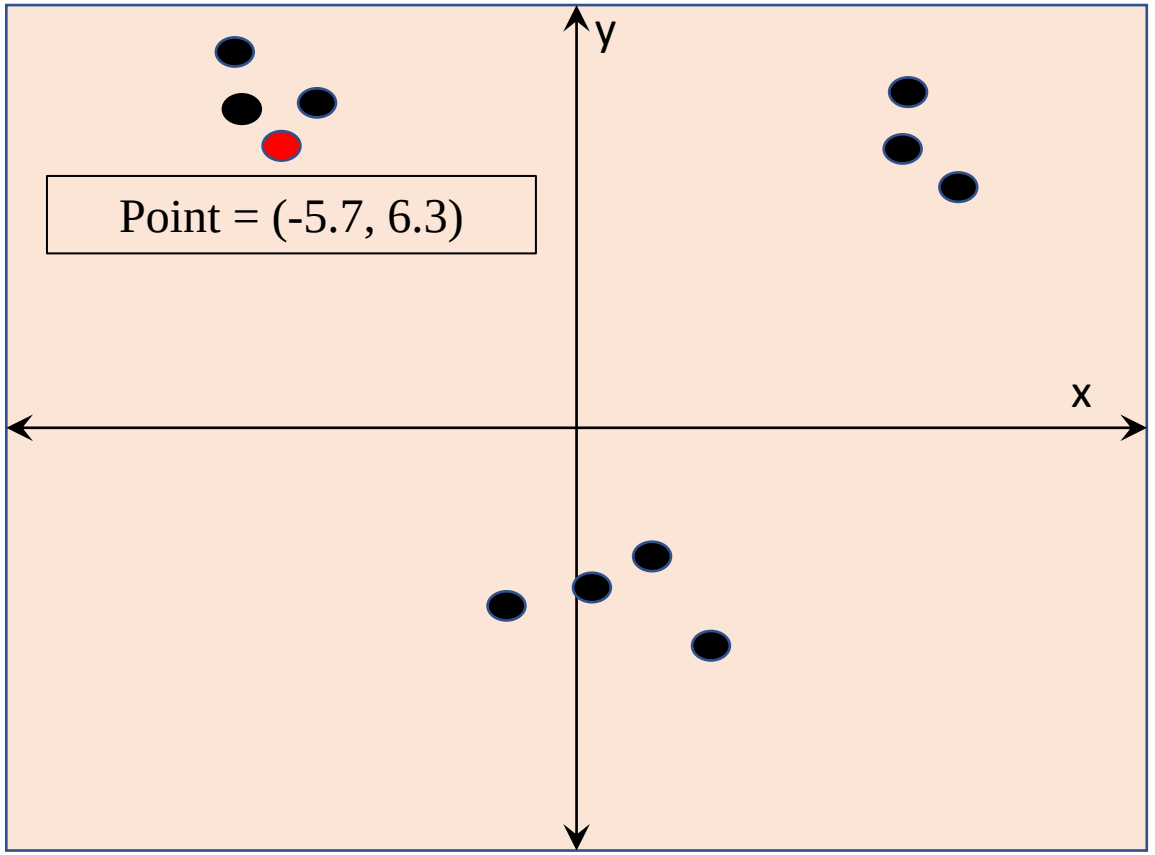


KD Tree Data Structure : Creation

KD Tree

- Binary Search Tree that splits points between alternating axes

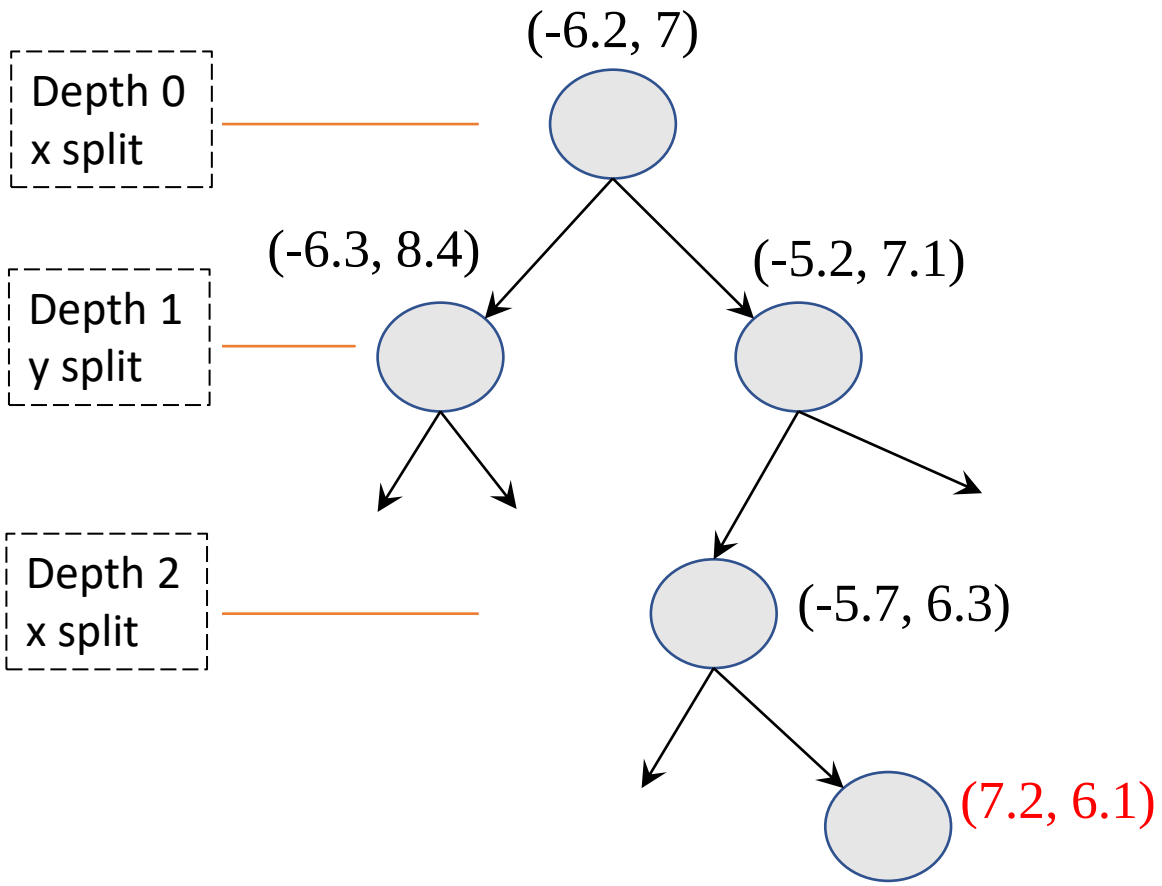


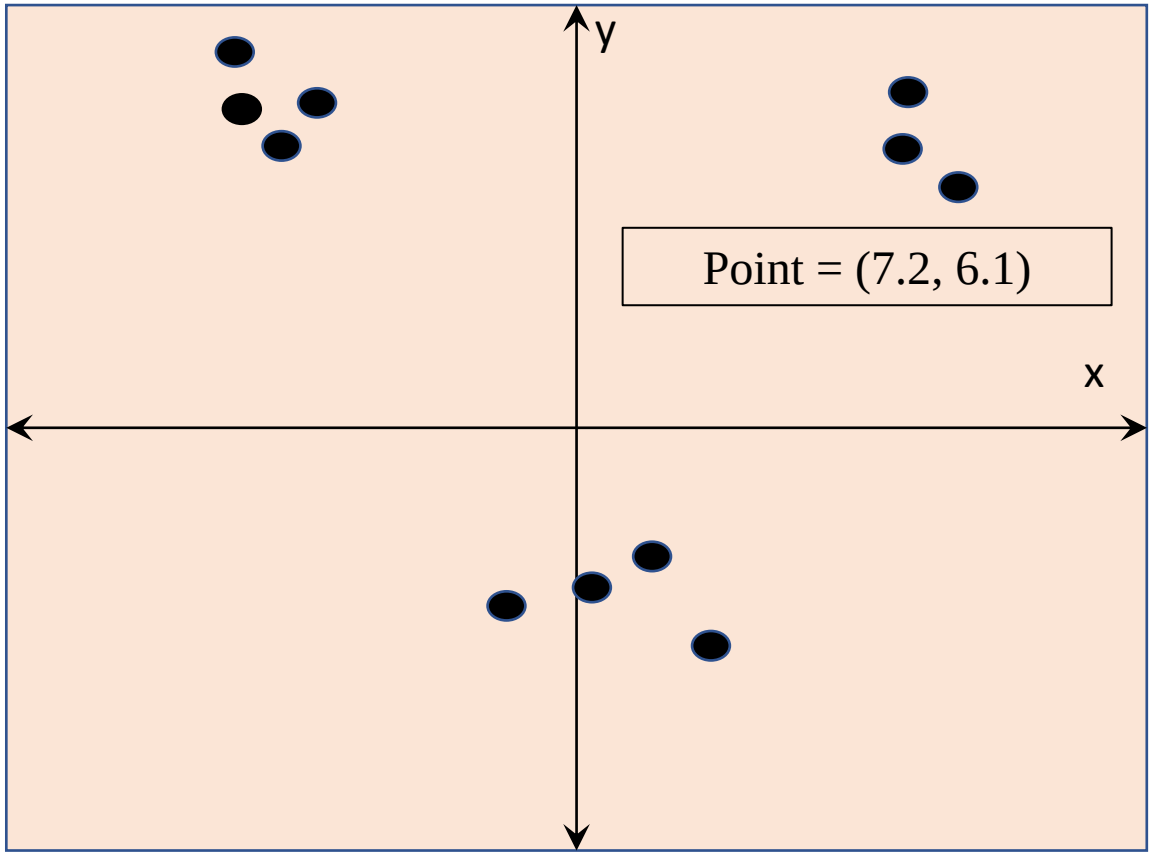


KD Tree Data Structure : Creation

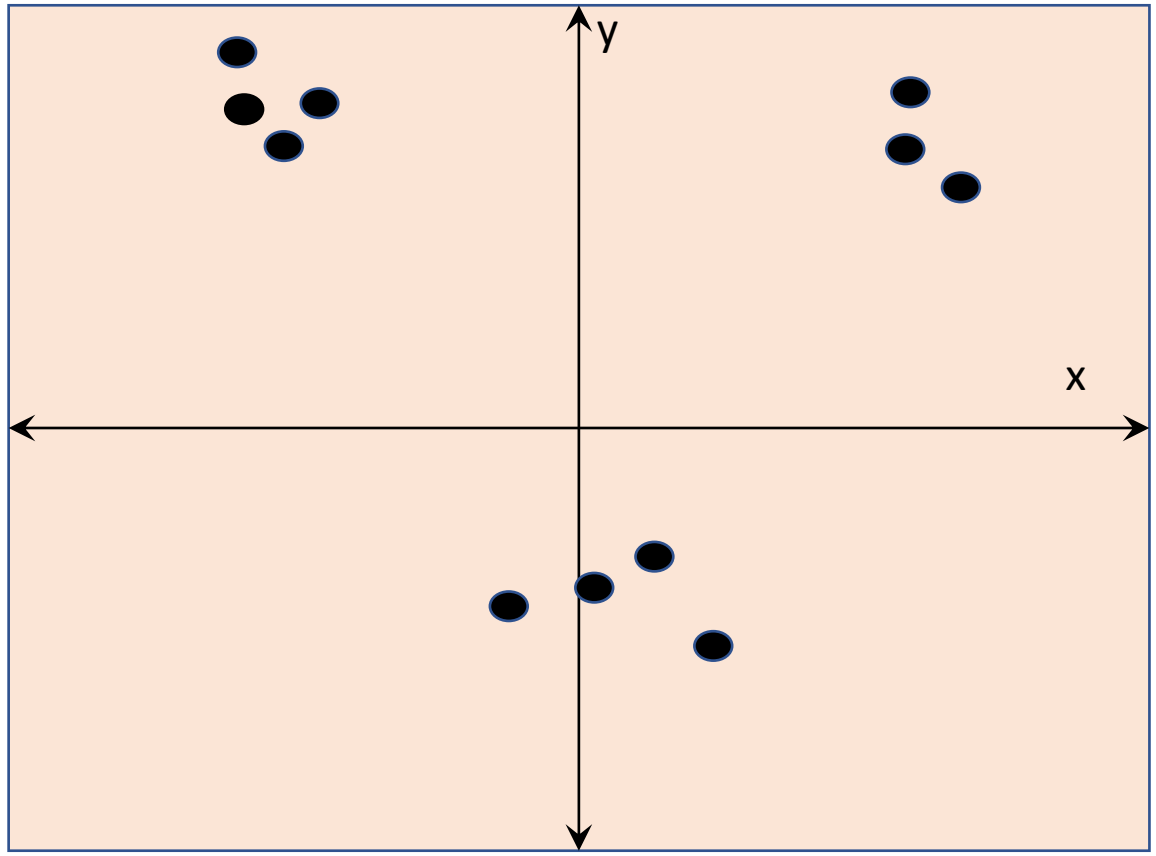
KD Tree

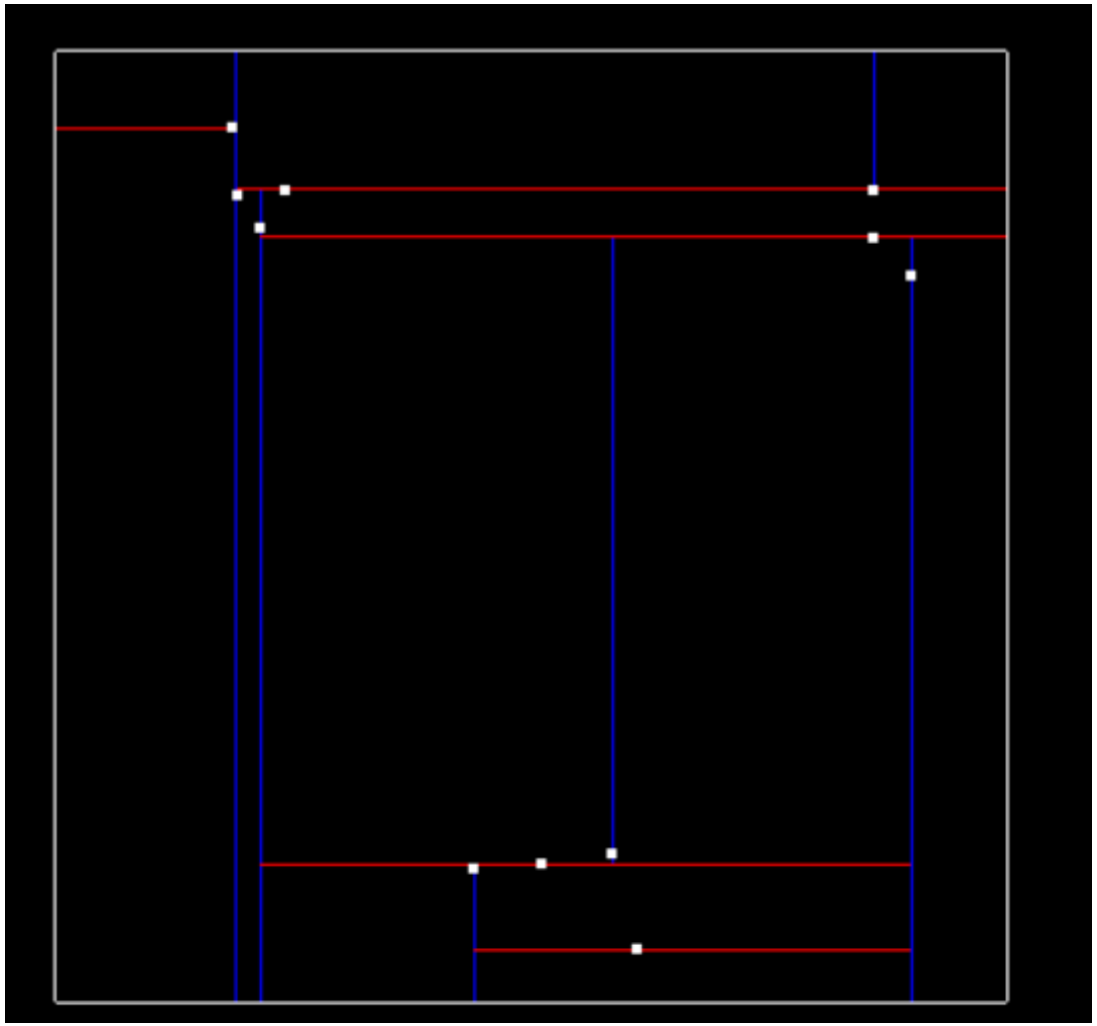
- Binary Search Tree that splits points between alternating axes





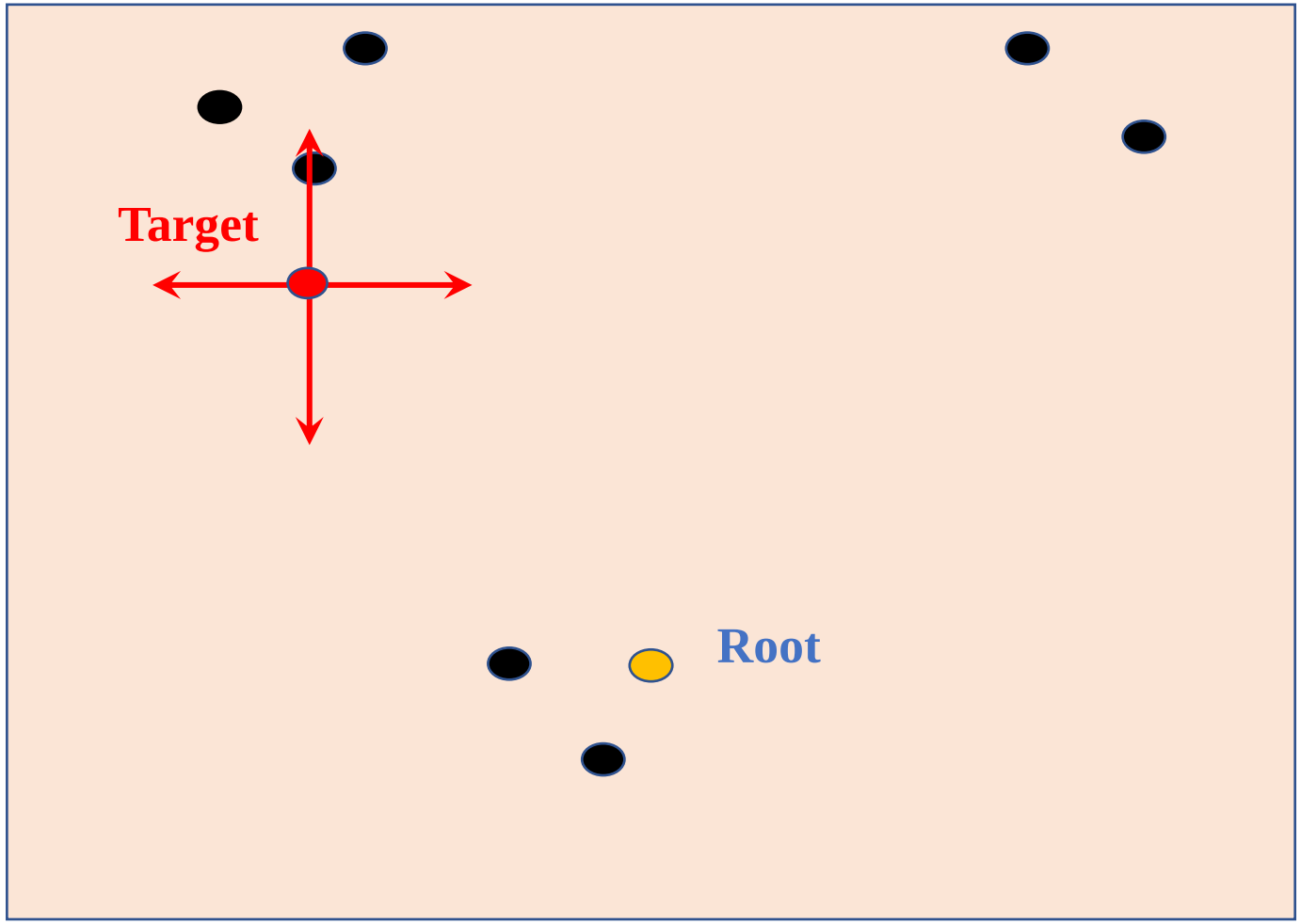
KD Tree Data Structure: Implementation





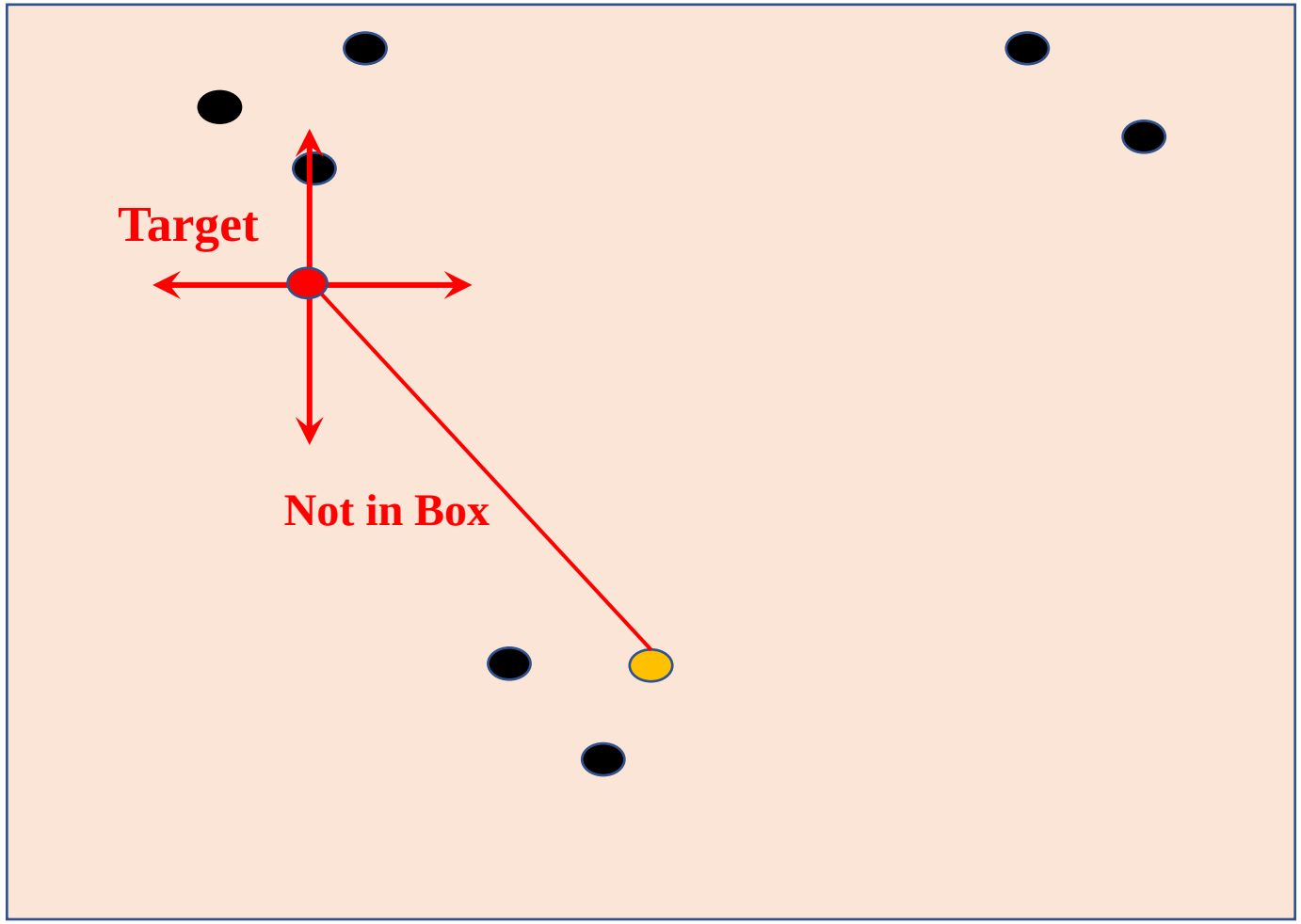
Nearest Neighbors Search using KD Tree Data Structure

- Box calculation
  - Box Size = 2 ×tolerance
- If distance from target to point is inside the box
  - Check if distance ≤ tolerance
    - If yes, assign to the **Cluster**



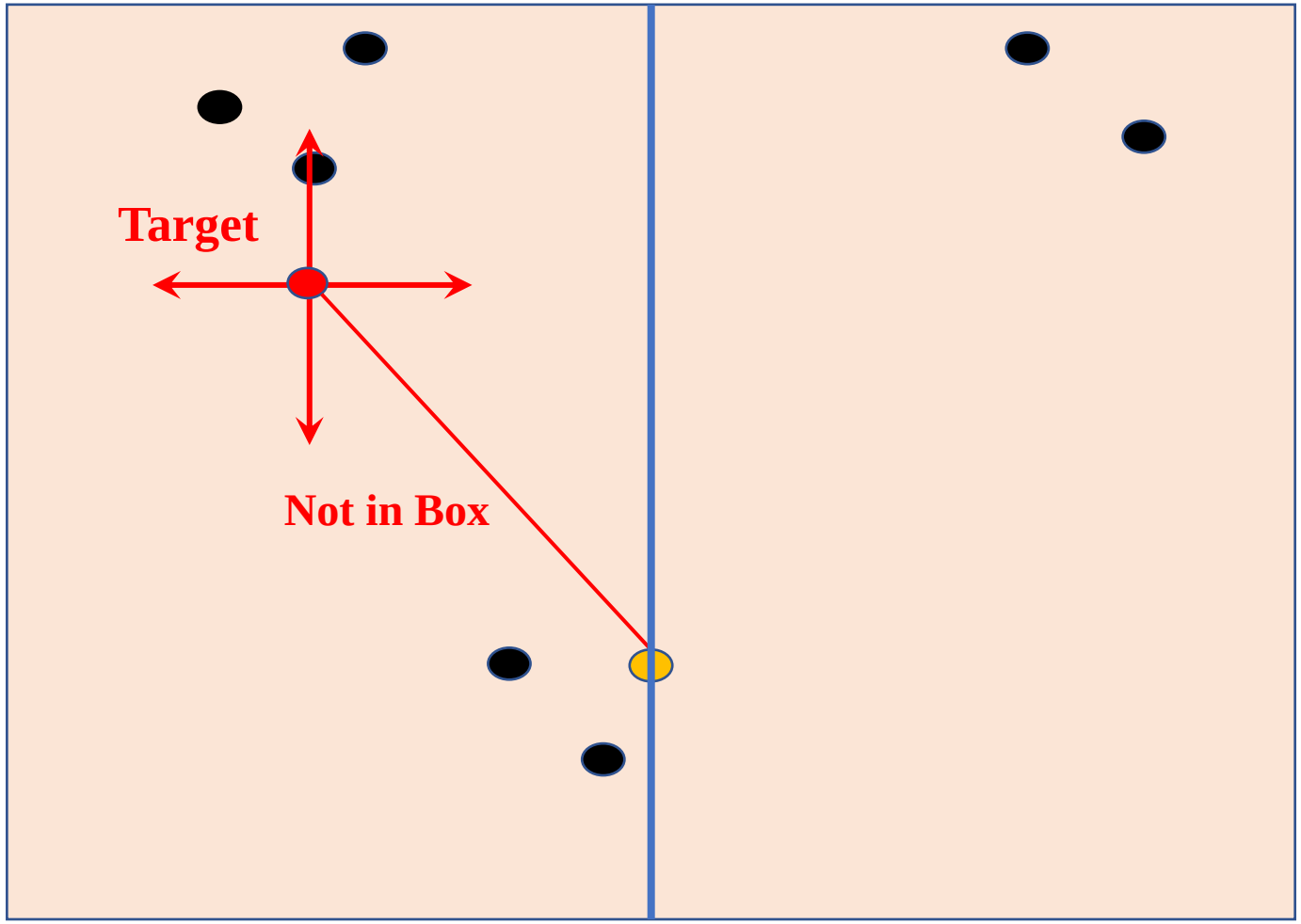
Nearest Neighbors Search using KD Tree Data Structure

- Box calculation
  - Box Size = 2 ×tolerance
- If distance from target to point is inside the box
  - Check if distance ≤ tolerance
    - If yes, assign to the **Cluster**



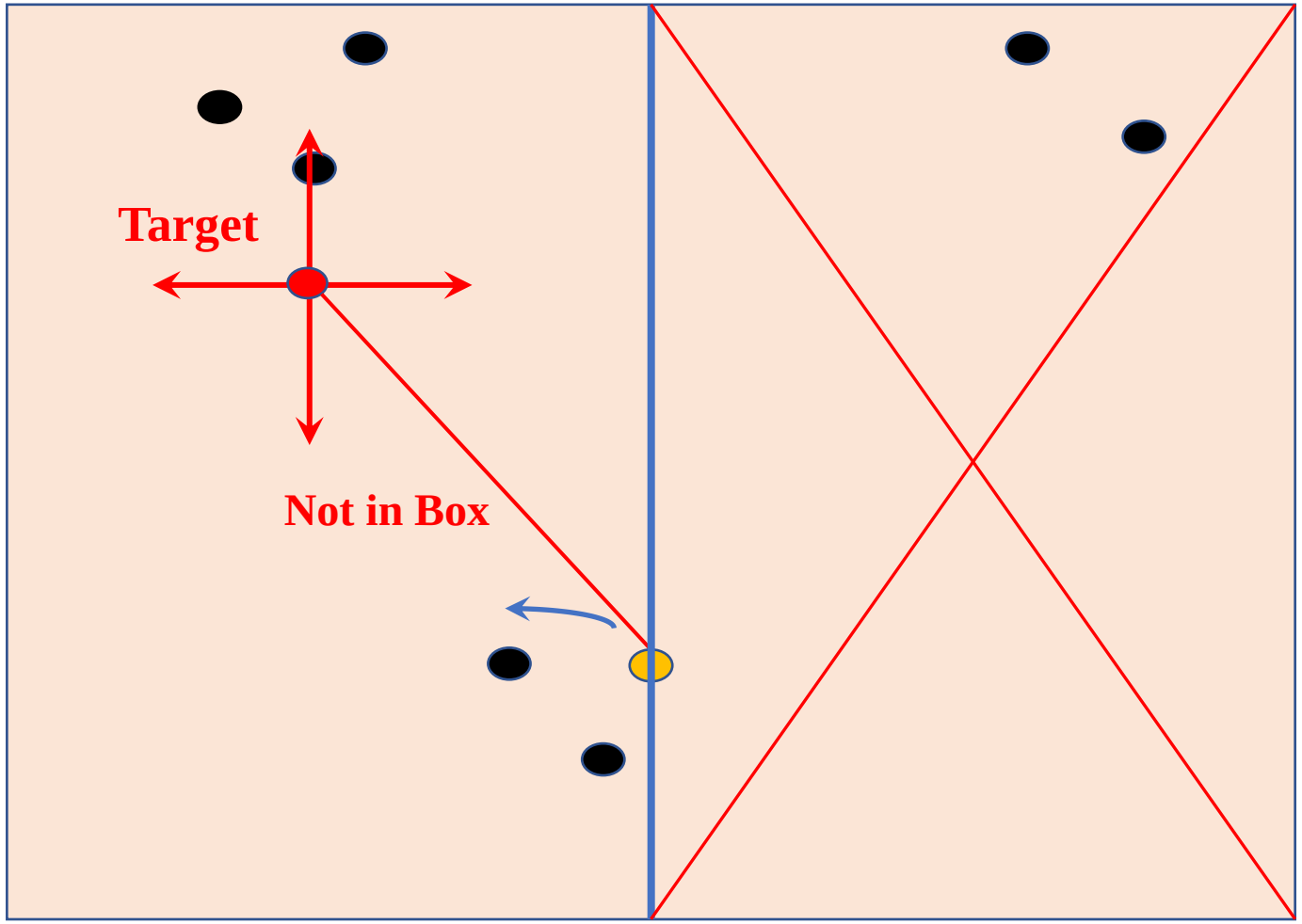
Nearest Neighbors Search using KD Tree Data Structure

- Box calculation
  - Box Size = 2 ×tolerance
- If distance from target to point is inside the box
  - Check if distance ≤ tolerance
    - If yes, assign to the **Cluster**



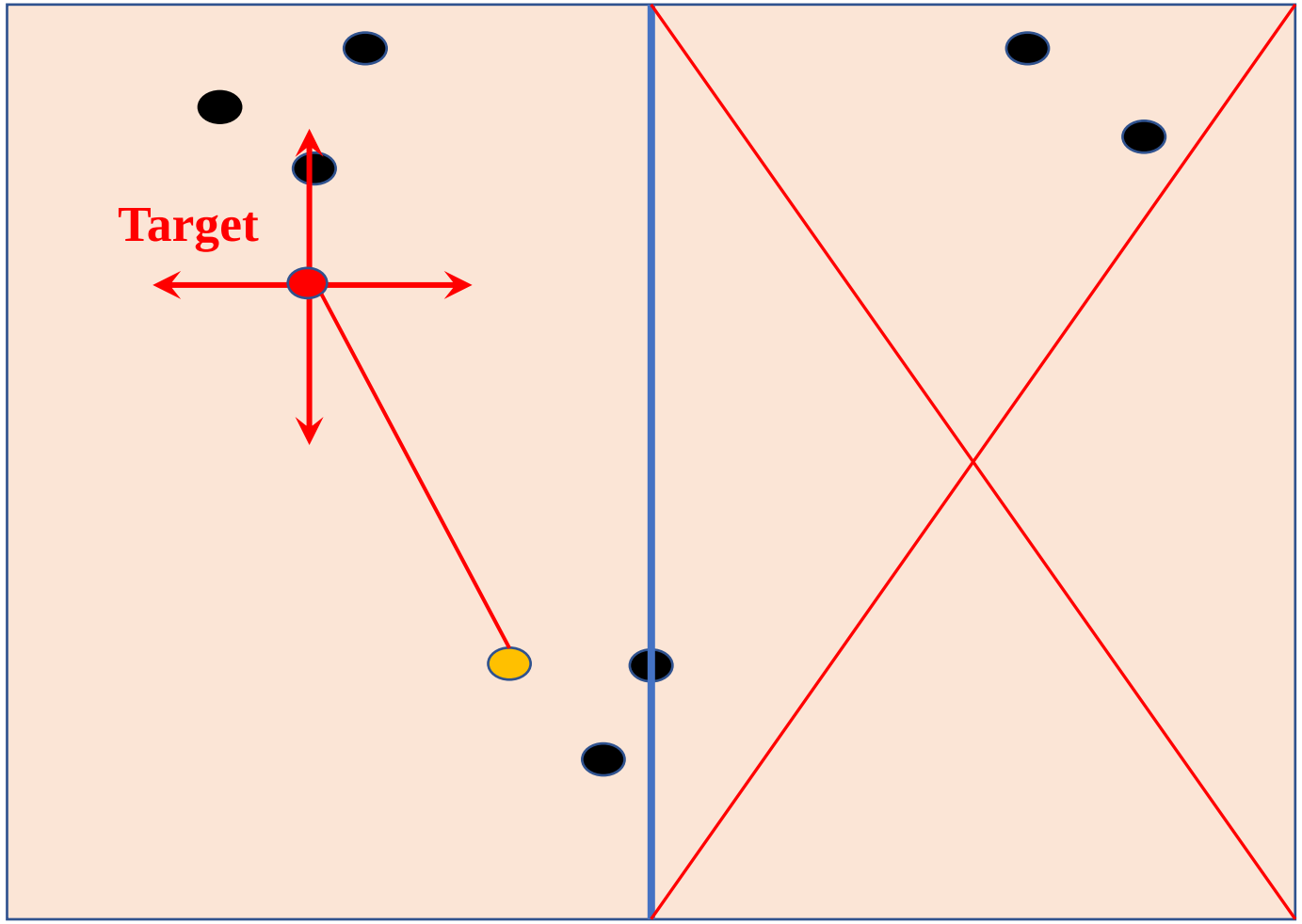
Nearest Neighbors Search using KD Tree Data Structure

- Box calculation
  - Box Size = 2 ×tolerance
- If distance from target to point is inside the box
  - Check if distance ≤ tolerance
    - If yes, assign to the **Cluster**



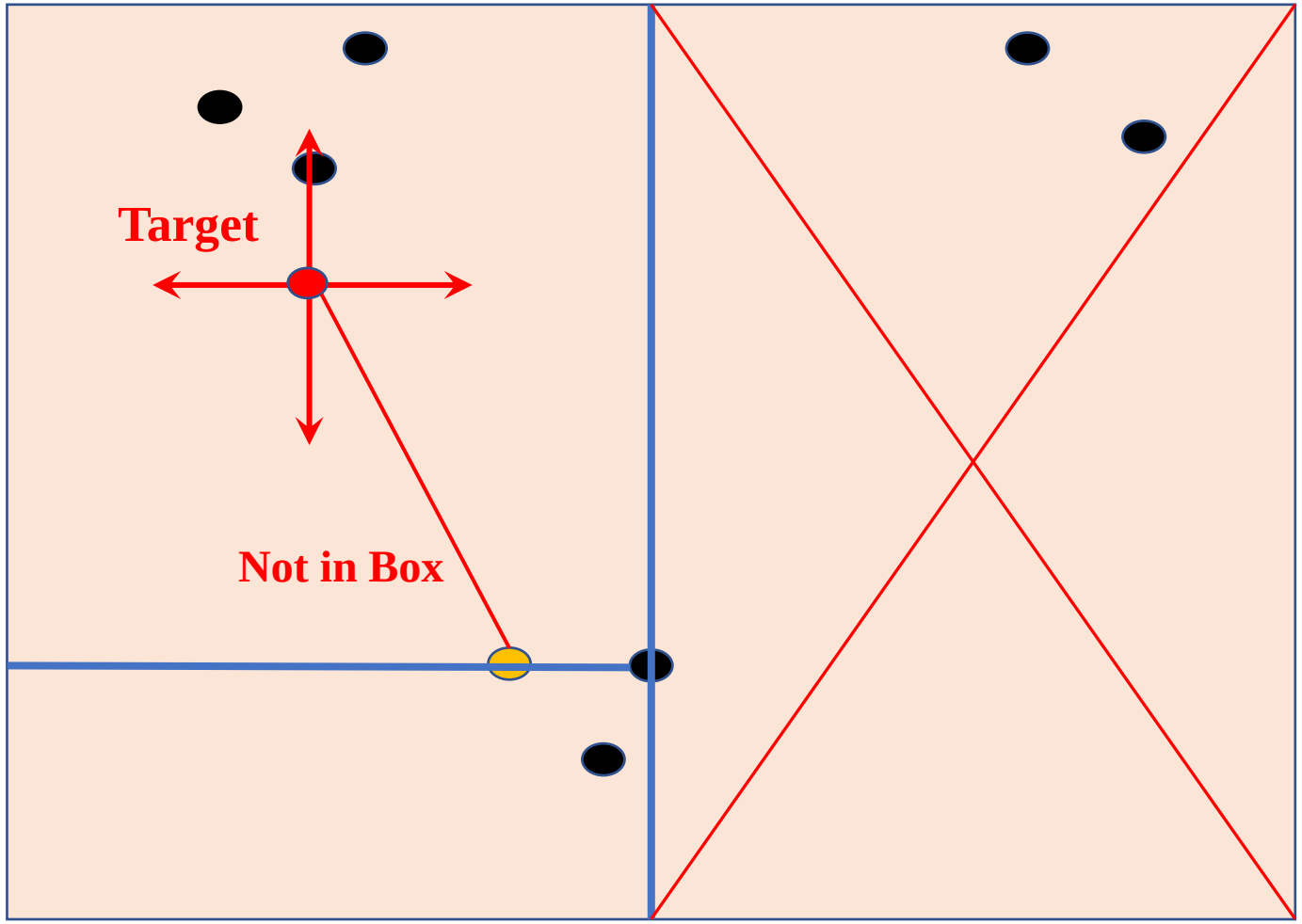
Nearest Neighbors Search using KD Tree Data Structure

- Box calculation
  - Box Size = 2 ×tolerance
- If distance from target to point is inside the box
  - Check if distance ≤ tolerance
    - If yes, assign to the **Cluster**



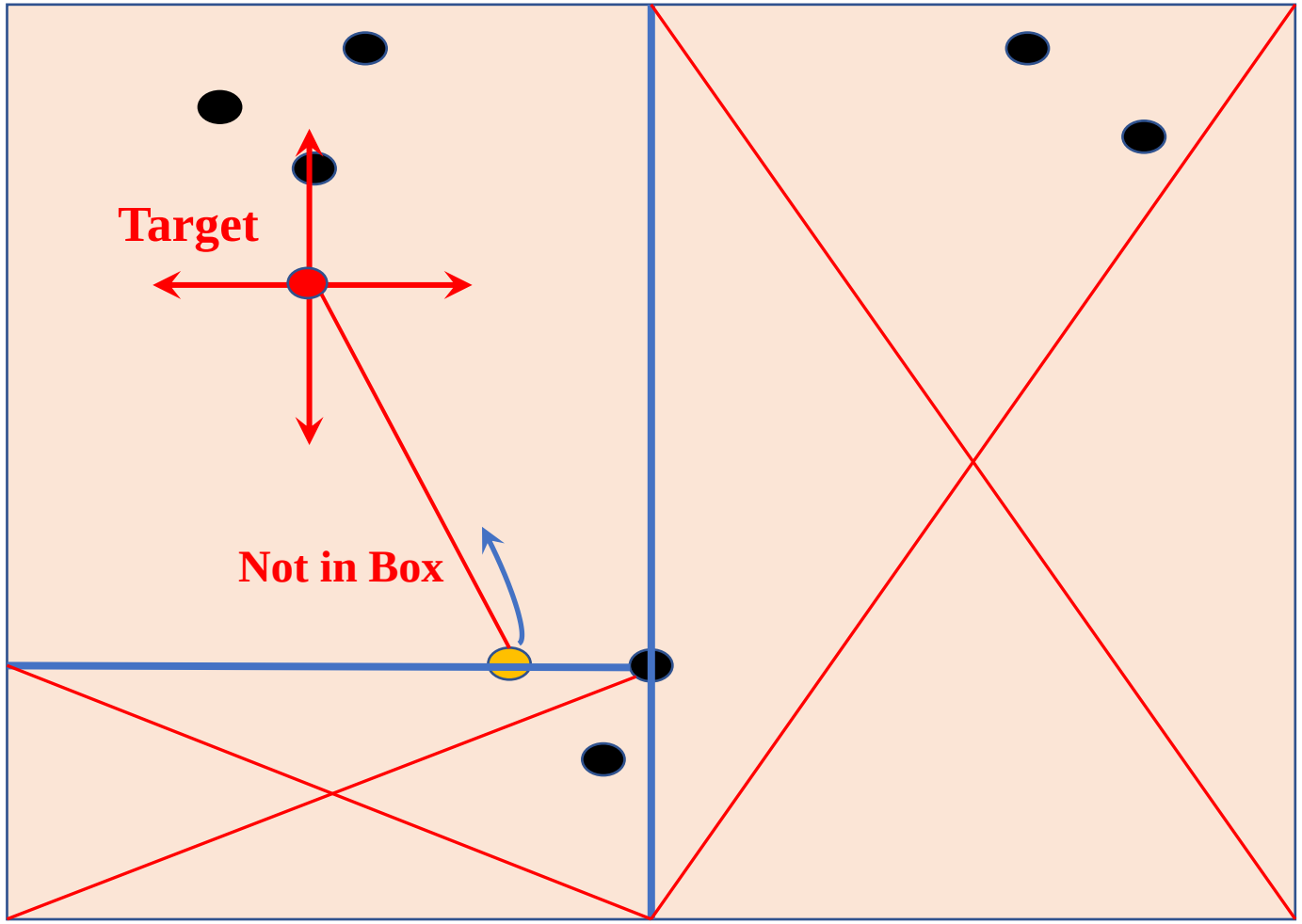
Nearest Neighbors Search using KD Tree Data Structure

- Box calculation
  - Box Size = 2 ×tolerance
- If distance from target to point is inside the box
  - Check if distance ≤ tolerance
    - If yes, assign to the **Cluster**



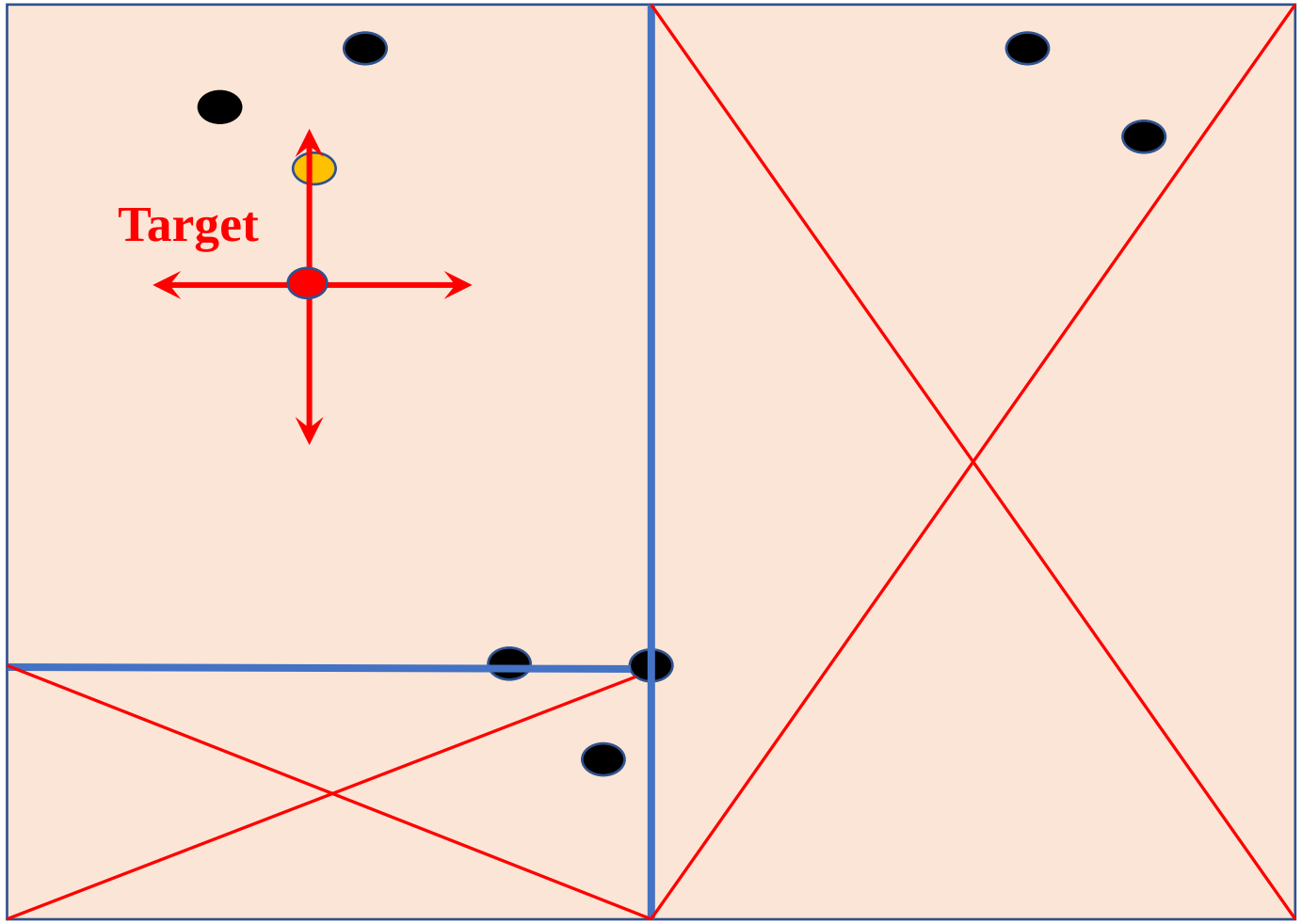
Nearest Neighbors Search using KD Tree Data Structure

- Box calculation
  - Box Size = 2 ×tolerance
- If distance from target to point is inside the box
  - Check if distance ≤ tolerance
    - If yes, assign to the **Cluster**



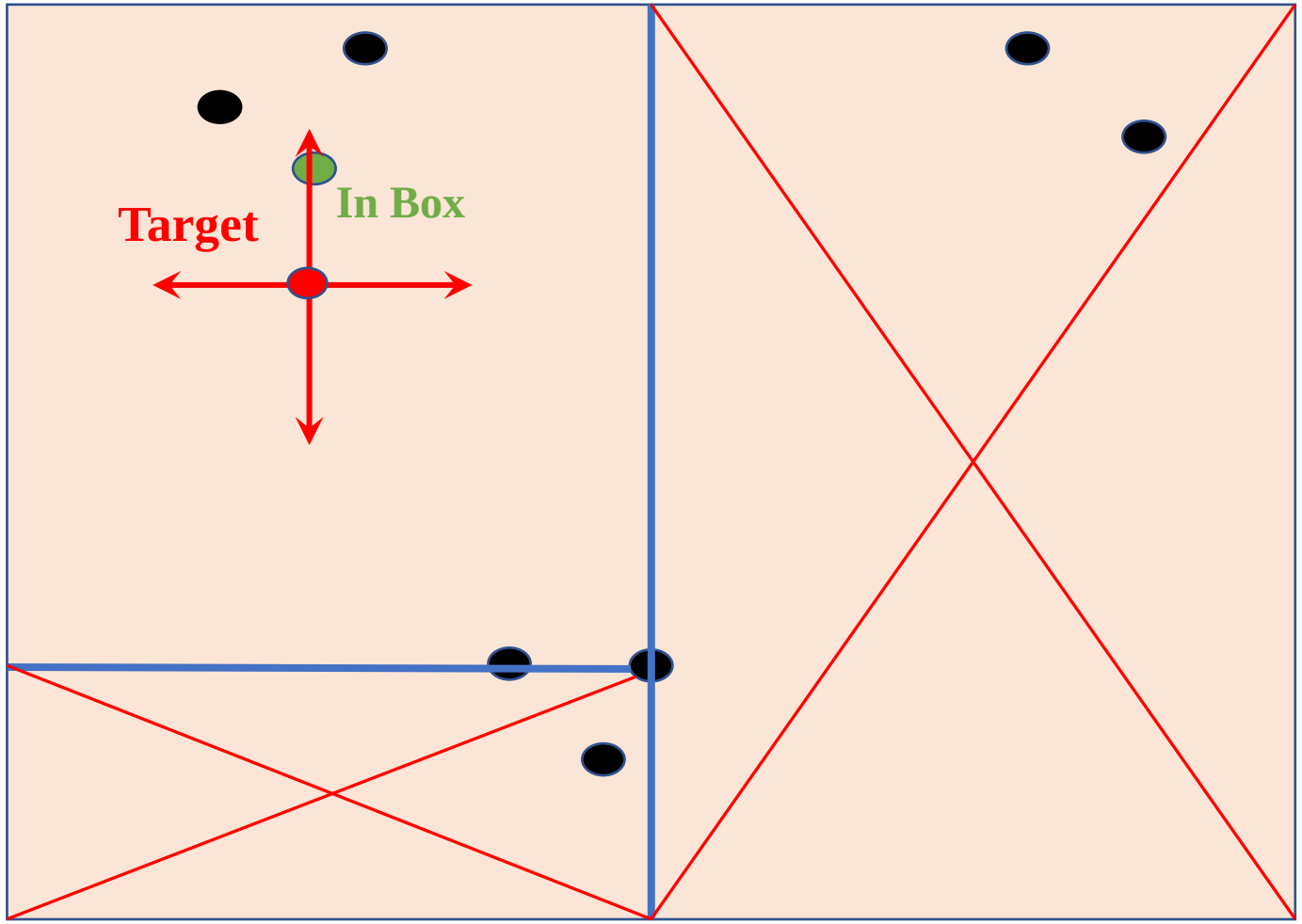
Nearest Neighbors Search using KD Tree Data Structure

- Box calculation
  - Box Size = 2 ×tolerance
- If distance from target to point is inside the box
  - Check if distance ≤ tolerance
    - If yes, assign to the **Cluster**



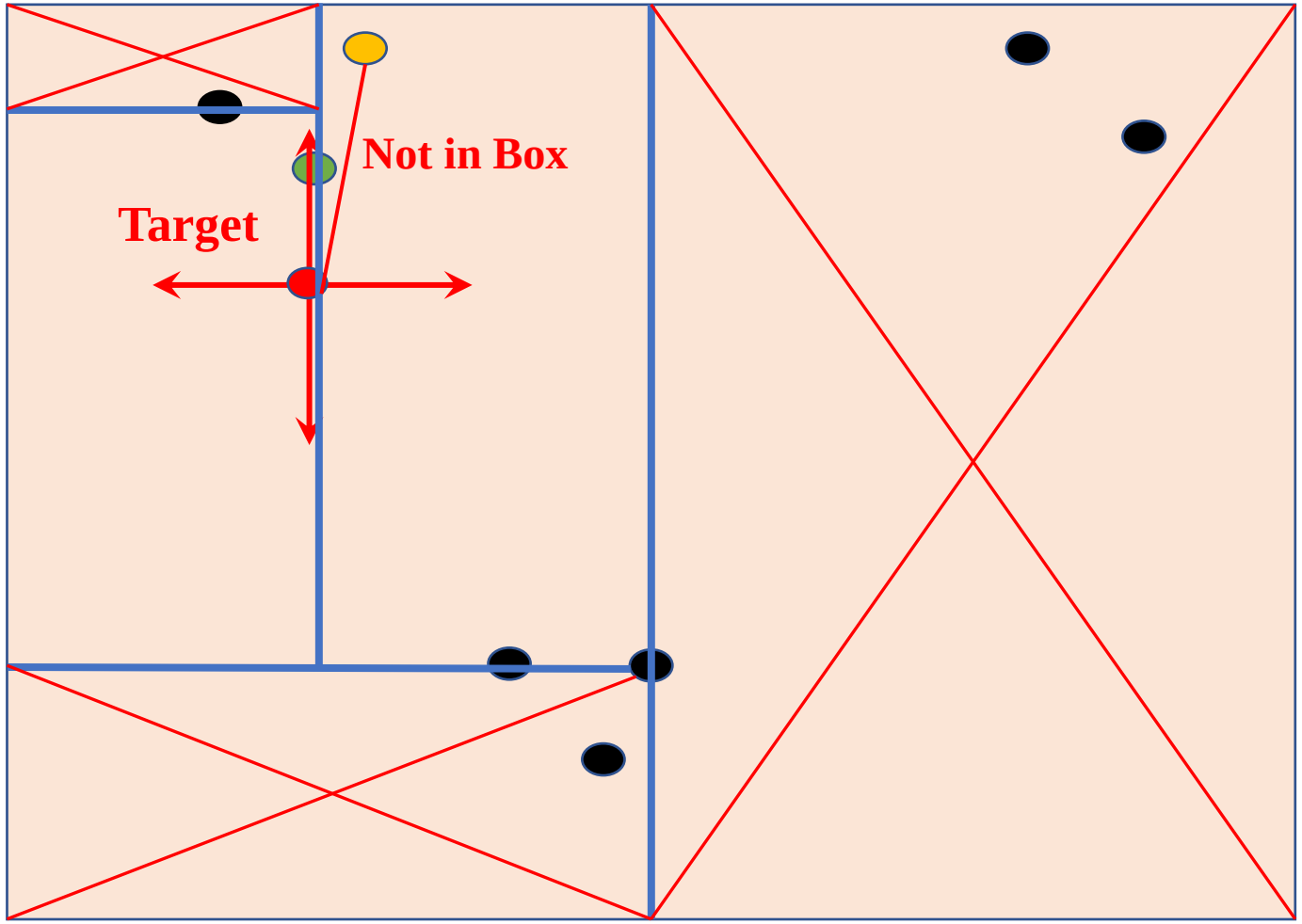
Nearest Neighbors Search using KD Tree Data Structure

- Box calculation
  - Box Size = 2 ×tolerance
- If distance from target to point is inside the box
  - Check if **distance ≤ tolerance**
    - If yes, **assign to the Cluster**

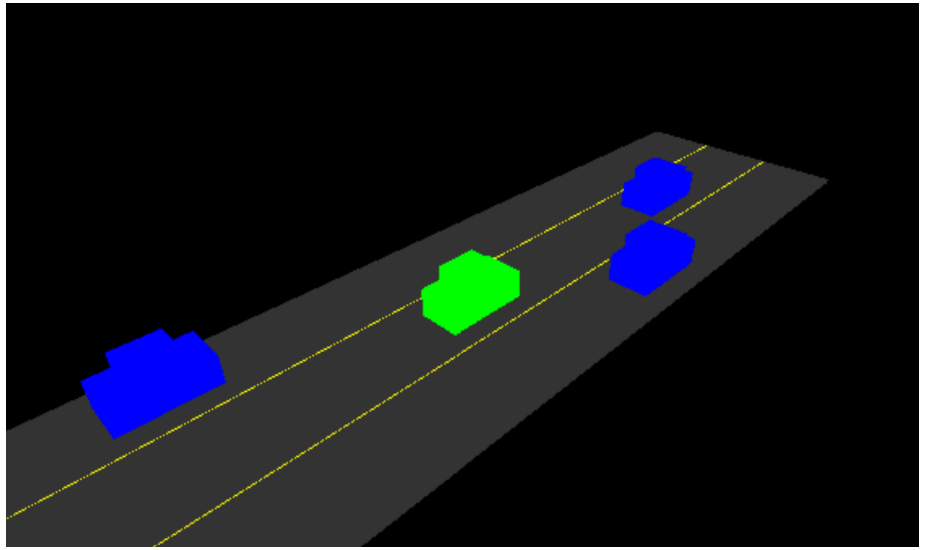


Nearest Neighbors Search using KD Tree Data Structure

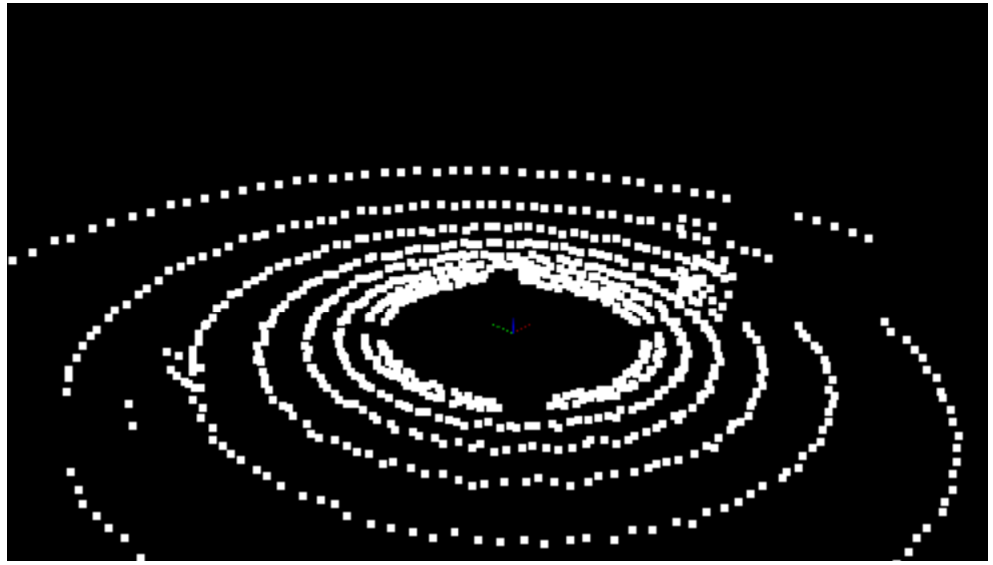
- Only one point is within the distance tolerance
- Cluster of two points
- Number of cheap box searches = 5
- Number of distance calculations = 1



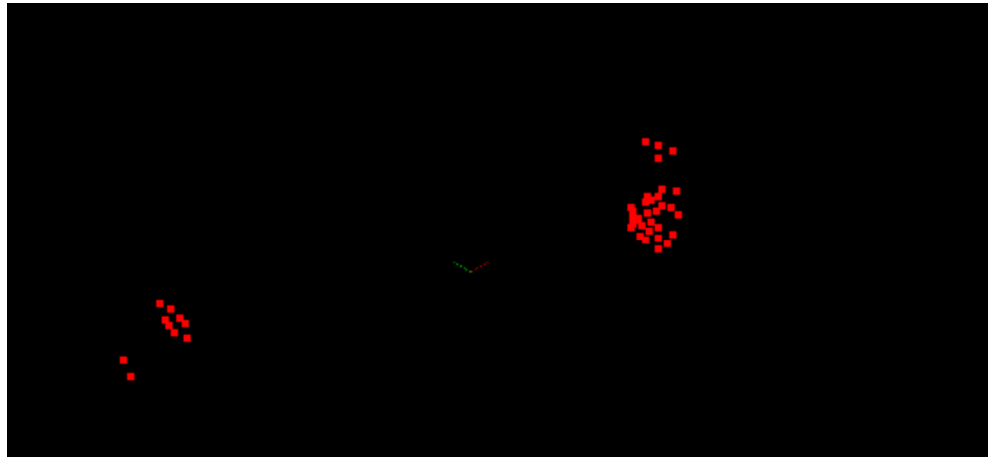
Euclidean Clustering Results



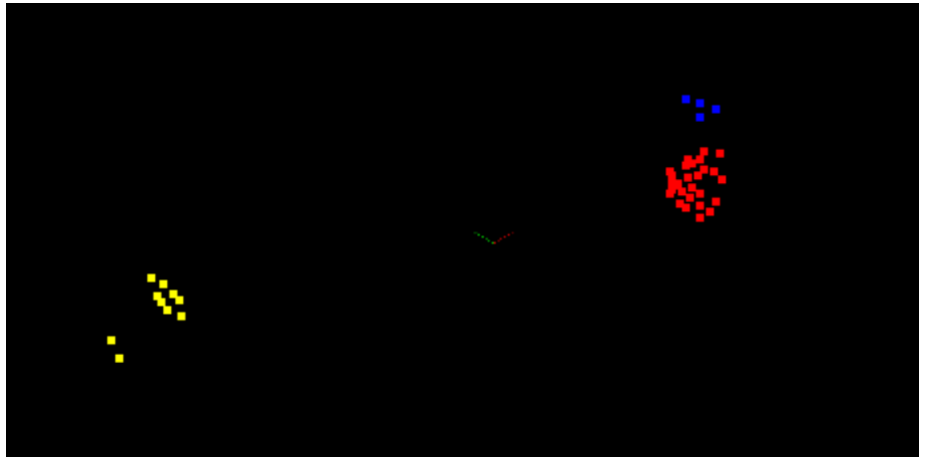
Point Cloud
from Lidar



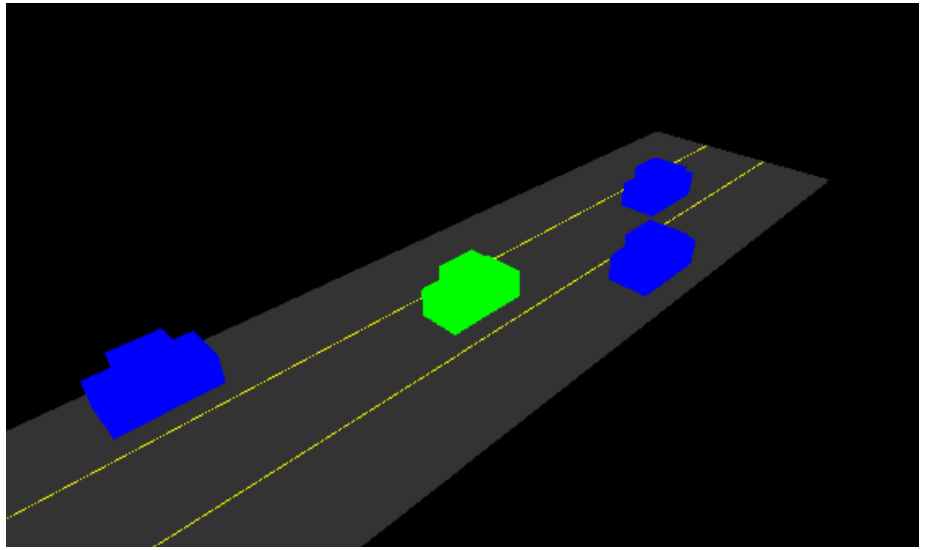
Planar
Segementation

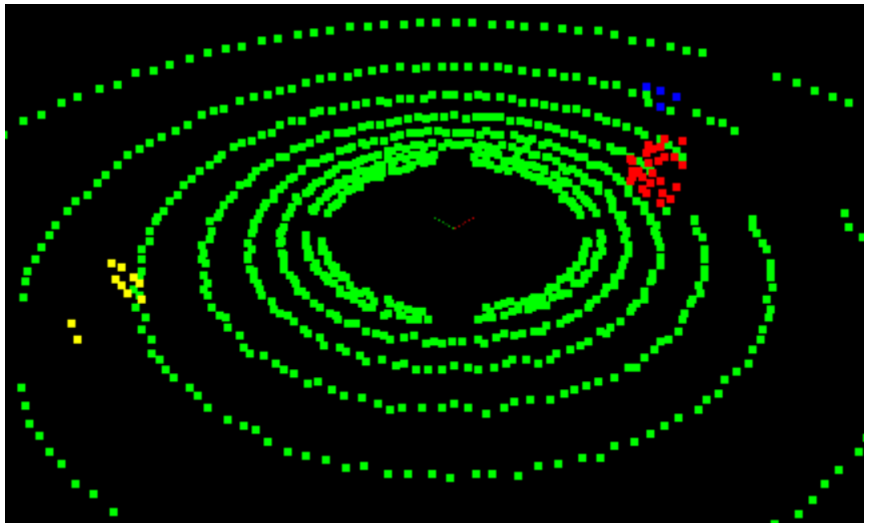


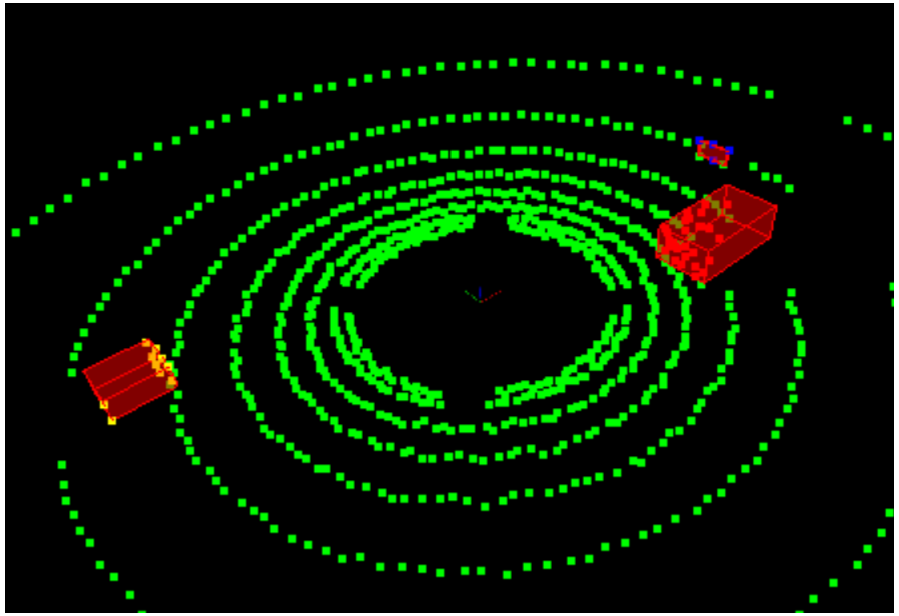
Euclidean
Clustering



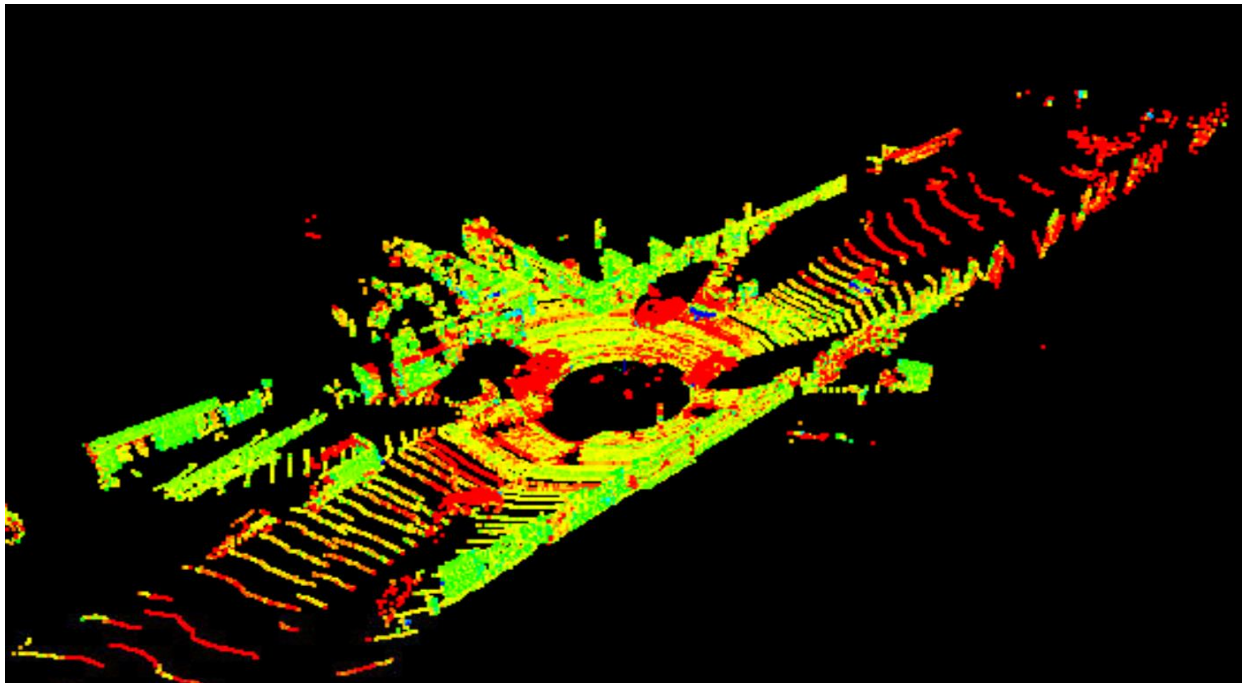
Bounding Box for Detected Obstacles

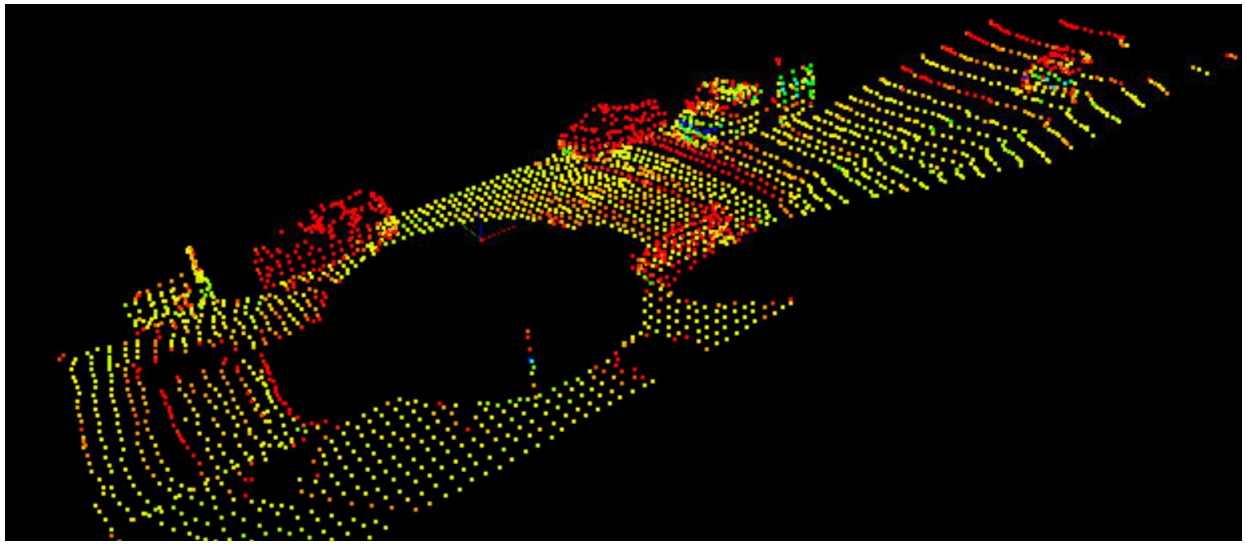






Working with Real PCD





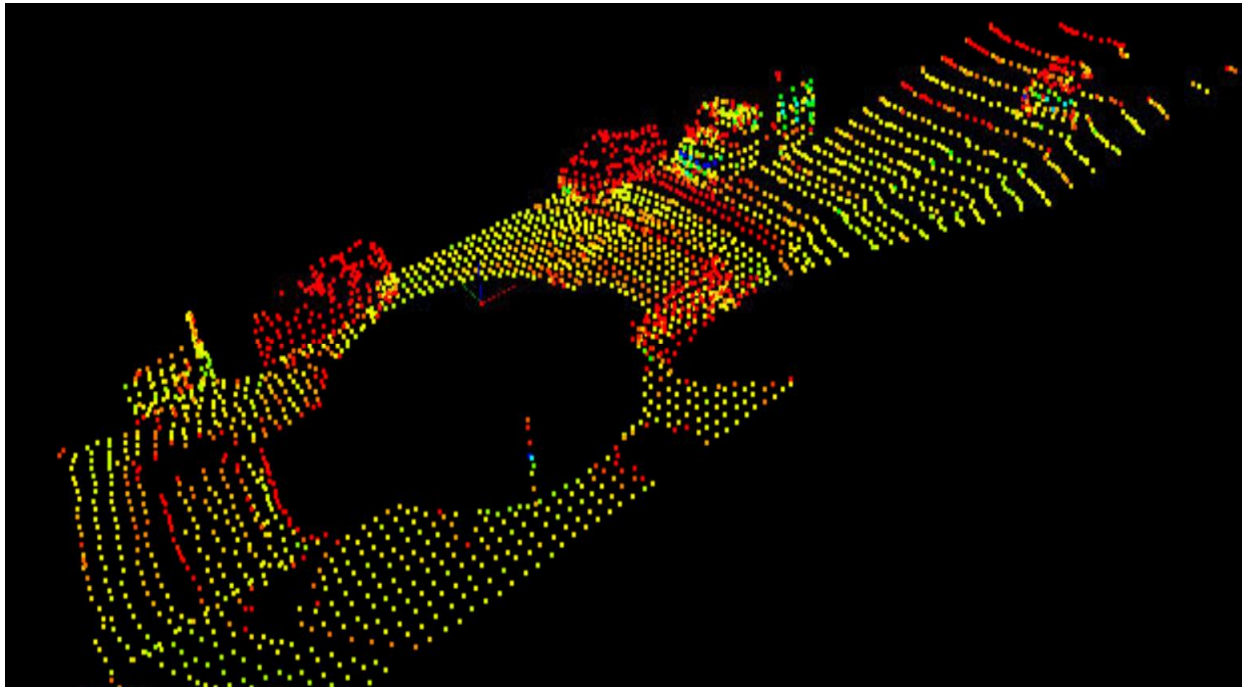
Number of points in Point Cloud = 119,978

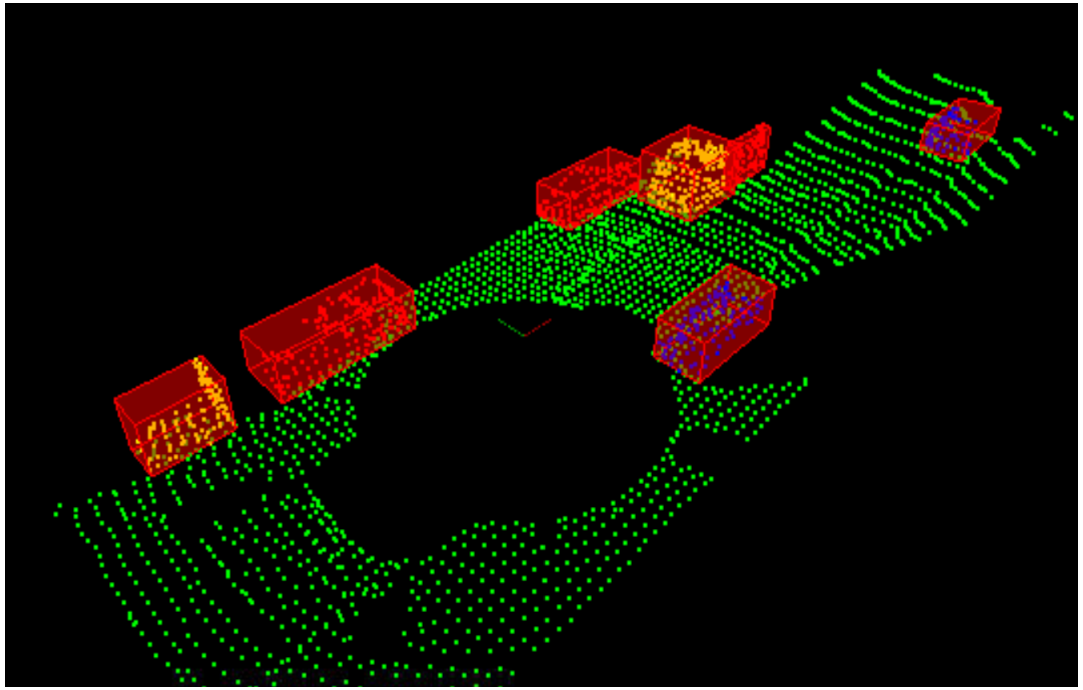
- Quite high resolution
- Spans far distance
- Some of the obstacles need not be processed
- Increases processing time of segmentation and clustering

Filtering → Downsampling

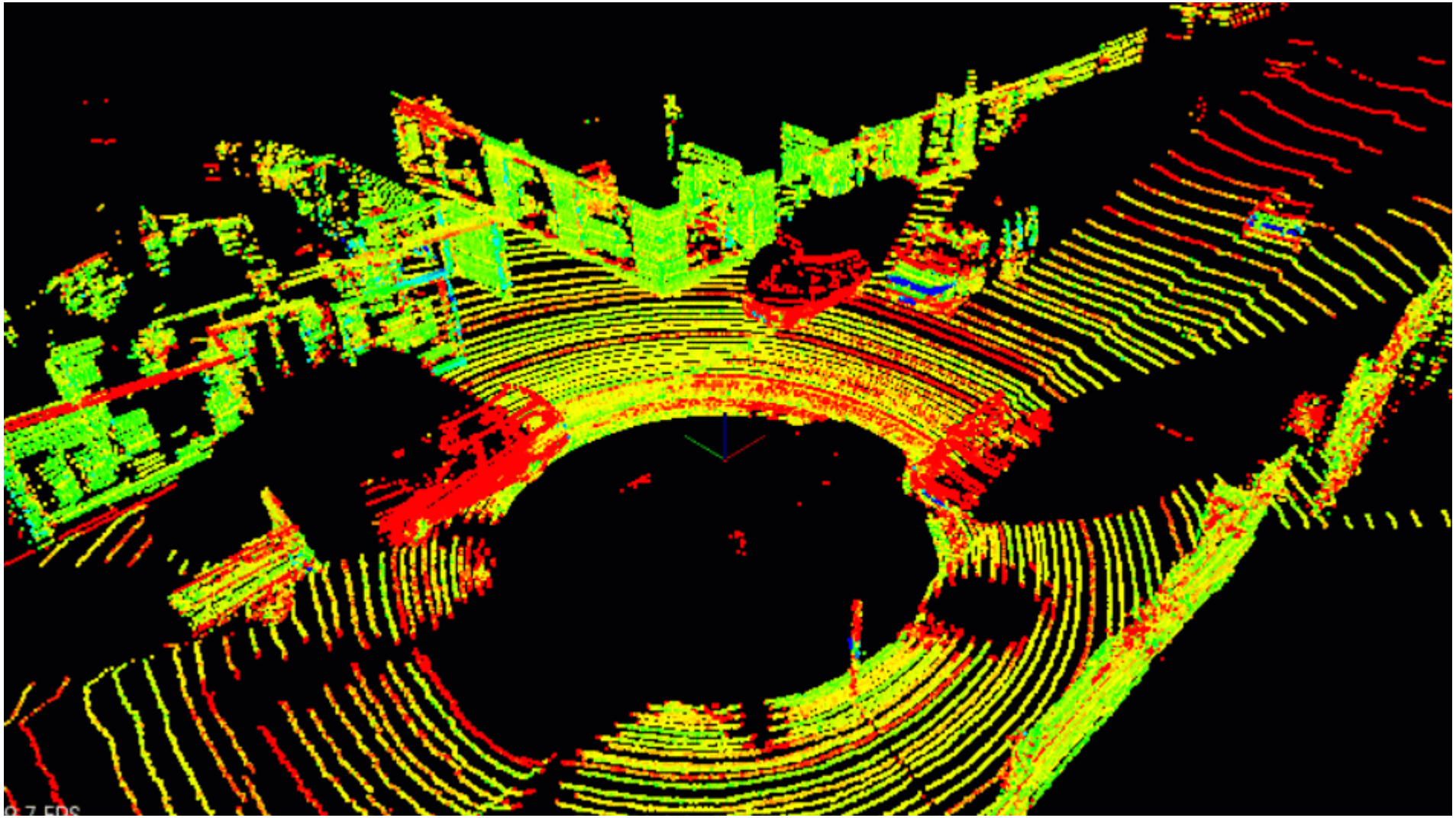
- **Voxel Grid Filtering**
  - Creates a cubic grid
  - Filters the cloud by only leaving a single point per voxel cube
  - Larger the cube length, lower the resolution
- **Region of Interest-based Filtering**
  - Any points outside a boxed region is removed

Working with Real PCD: Results

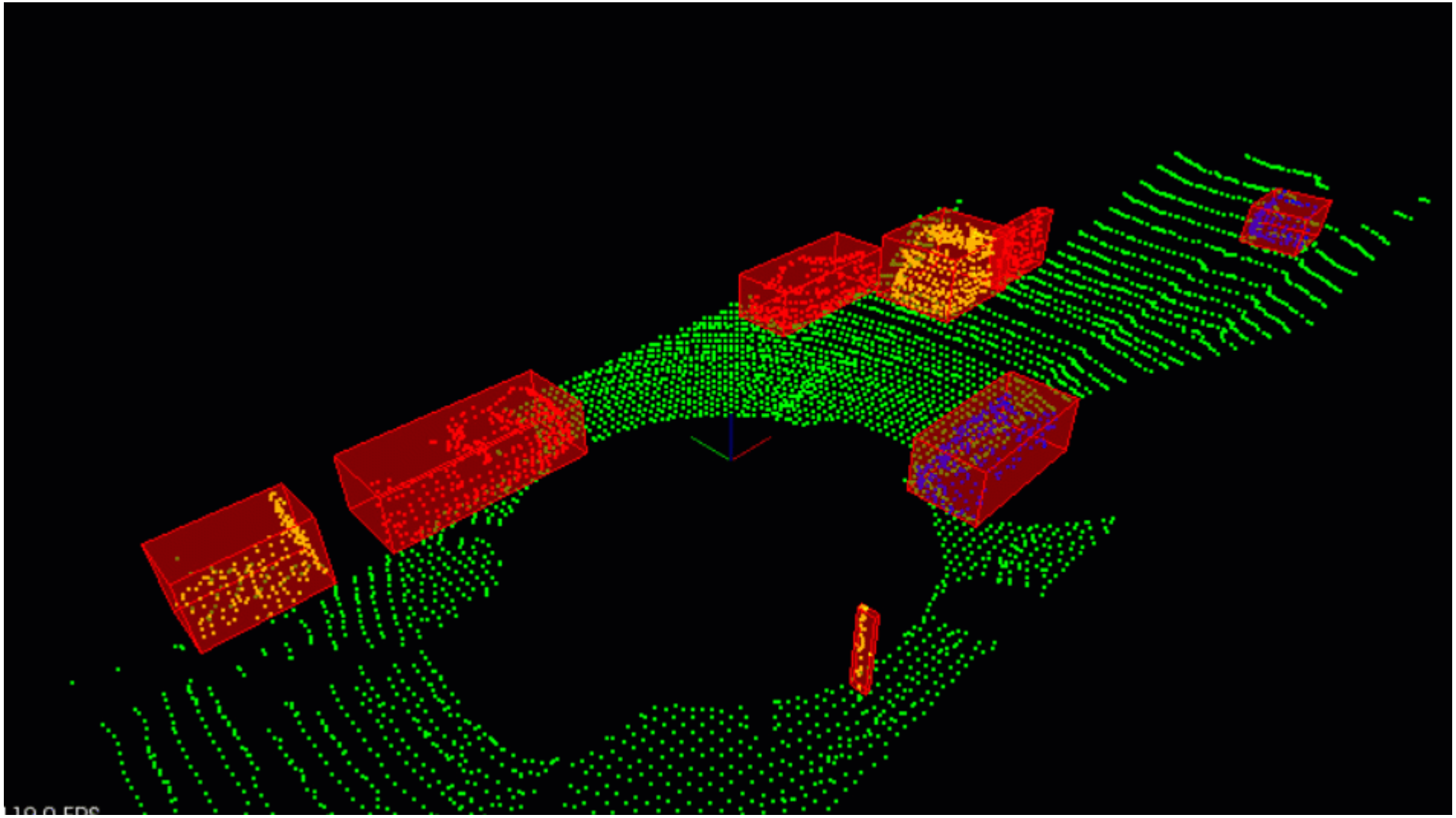




Working with Real PCD: Streaming Multiple PCD Files



Working with Real PCD: Streaming Multiple PCD Files



Thank You