Section 1: Introduction to Sensor Fusion

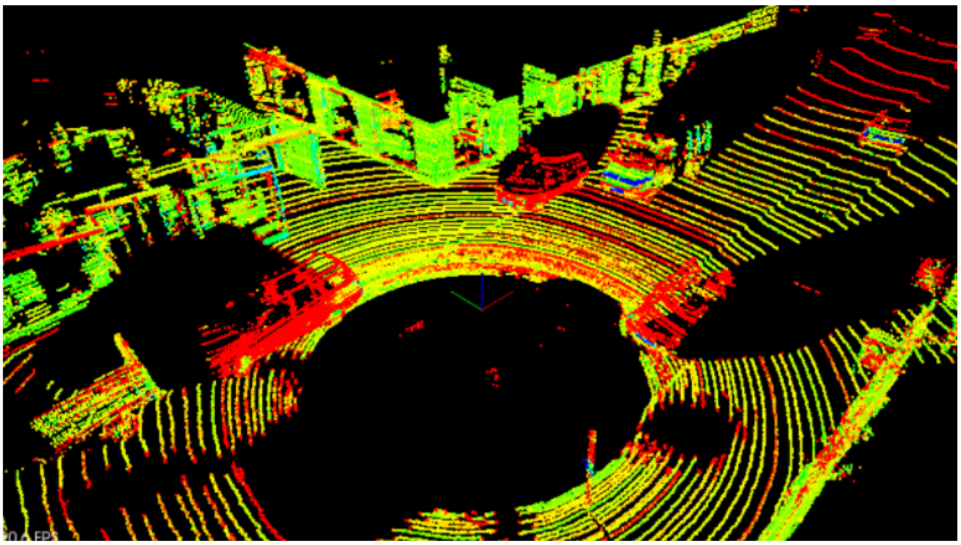
Cameras cannot measure distances and how fast things are going.
Apply signal processing to radar data to measure position based on time of wave velocity.
Measure velocity based on Doppler effect -shift of frequency based on wave bounce.
Filters rely on prediction update cycles - first predict or estimate position of different objects then update position based on sensor measurements - continues in a cycle.
Continuous or discrete measurement space - sensor environment. Converting continuous to discrete introduces uncertainty.
Kalman Filters - good for linear moving vehicles but bad for non-linear or curved paths.
Programming unscented or extended Kalman filters linearize nonlinear motions. This reduces inherent uncertainties.

Section 2: Lidar Obstacle Detection

Lesson 1: Introduction to Lidar and Point Clouds

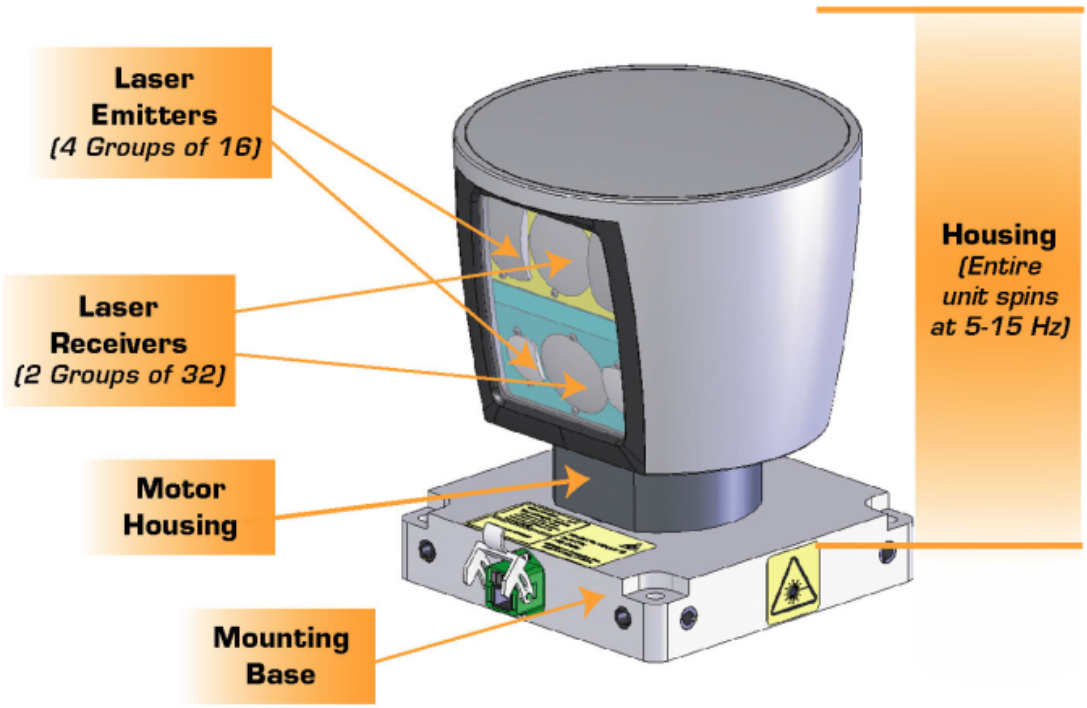
Lidar is higher resolution than radar but radar can directly measure velocity. Combining both can determine the position of an object and how fast it's moving.

Let's dive into how lidar data is stored. Lidar data is stored in a format called Point Cloud Data (PCD for short). A .pcd file is a list of (x,y,z) cartesian coordinates along with intensity values, it's a single snapshot of the environment, so after a single scan. That means with a VLP 64 lidar, a pcd file would have around 256,000 (x,y,z,i) values.



PCD of a city block with parked cars, and a passing van. Intensity values are being shown as different colors. The big black spot is where the car with the lidar sensor is located.

Here are the specs for a HDL 64 lidar. The lidar has 64 layers, where each layer is sent out at a different angle from the z axis, so different inclines. Each layer covers a 360 degree view and has an angular resolution of 0.08 degrees. On average the lidar scans ten times a second. The lidar can pick out objects up to 120M for cars and foliage, and can sense pavement up to 50M.

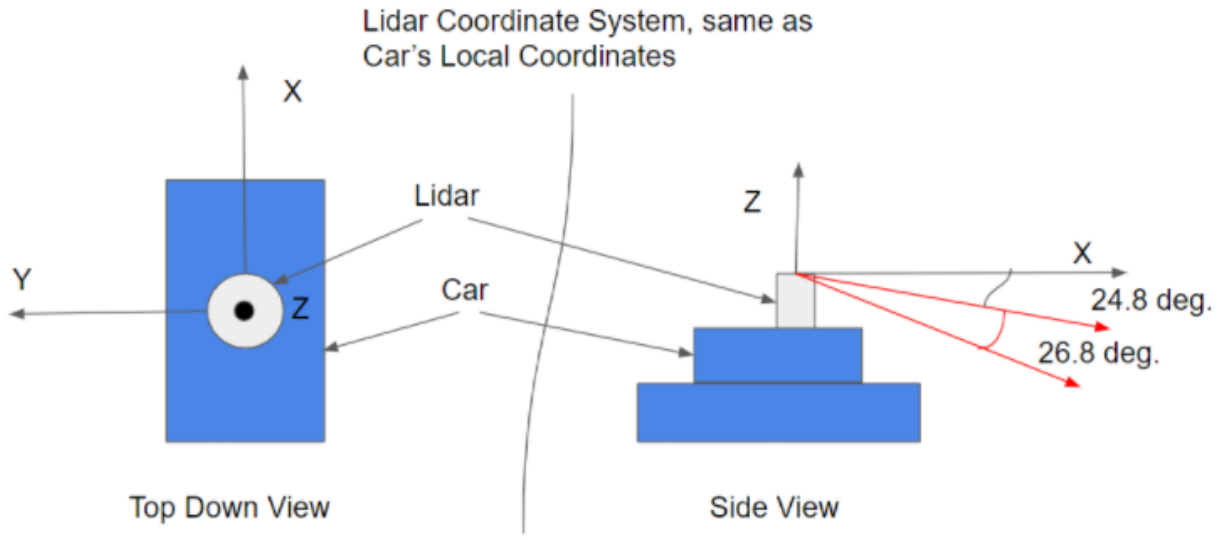


VLP 64 schematic, showing lidar emitters, receivers, and housing.

QUIZ QUESTION

Approximately how many points does an HDL 64 collect every second given these specs? Assume an average update rate of 10Hz.

With 64 layers, an angular resolution of 0.09 degrees, and average update rate of 10Hz the sensor collects (64x(360/0.08)x10) = 2,880,00 every second



PCD Coordinate System

Each of the 64 layers will be spread evenly across the 26.8 degree range.

QUIZ QUESTION

While scanning with a VLP 64, a laser signal from the top layer takes 66.7 ns to be emitted and received again. The laser is emitted at a -24.8 degree incline from the X axis and horizontally travels along the X axis. Knowing that the speed of light is 299792458 m/s, what would be the coordinates of this laser point (X,Y,Z) in meters?

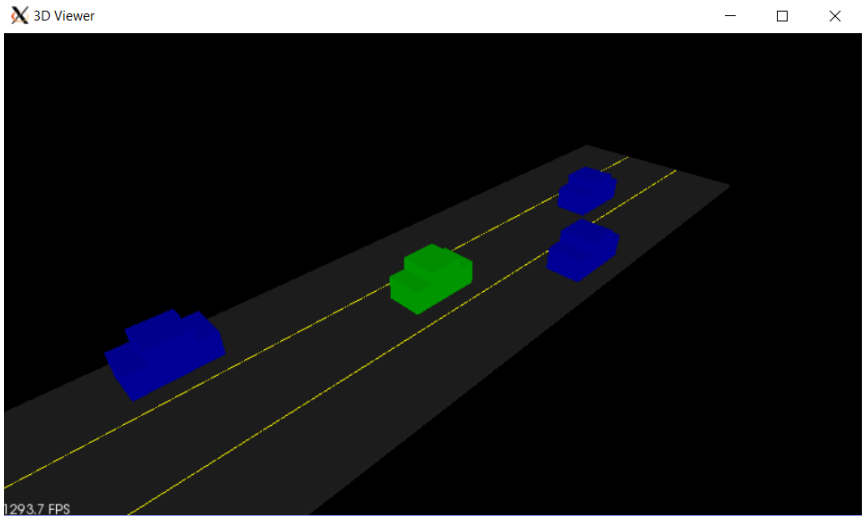
First the distance of the ray is calculated. Iit takes 66.7 ns to do a round trip, so it takes half that time to reach the object. The distance of the ray is then 299792458 *(66.7/2)* 10e-9 = 10 meters. The ray is traveling along the X-axis so the Y component is 0. The X, and Z components can be calculated by applying some Trig, X component = 10m *sin(90-24.8) = 9.08 , Z component = 10m* -cos(90-24.8) = -4.19.

8. The Point Cloud Library (PCL)

Open source library for C++. Has built in functions for filtering, segmentation, clustering with rendering. Widely used in robotics.
http://pointclouds.org/

13. The PCL Viewer

Renders points and shapes into graphics. Can add camera options and highway environment.



15. Creating the Lidar Object

Set up lidar with cars and a ground slope for sensing. Tells the lidar what physical object rays can bounce off of. The `Lidar` object should be created on the heap using the `new` keyword and pointer (*) type. By instatinating on the heap, we have more memory to work with than the 2MB on the stack in case the lidar object gets big from holding many data points.

```
Lidar* lidar = new Lidar(cars, groundSlope);
```

Note

The syntax of PointCloud with the **template** is similar to the syntax of vectors or other std container libraries: `ContainerName<ObjectName>`.

The Ptr type from PointCloud indicates that the object is actually a pointer - a 32 bit integer that contains the memory address of your point cloud object. Many functions in pcl use point cloud pointers as arguments, so it's convenient to return the inputCloud in this form.

The renderRays function is defined in `src/render`. It contains functions that allow us to render points and shapes to the pcl viewer. You will be using it to render your lidar rays as line segments in the viewer.

The arguments for the renderRays function is viewer, which gets passed in by reference. This means that any changes to the viewer in the body of the renderRays function directly affect the viewer outside the function scope. The lidar position also gets passed in, as well as the point cloud that your scan function generated. The type of point for the PointCloud will be `pcl::PointXYZ`. We will talk about some other different types of point clouds in a bit.

```
void renderRays(pcl::visualization::PCLVisualizer::Ptr& viewer, const Vect3& origin, const pcl::PointCloud<pcl::PointXYZ>::Ptr& cloud)
{

    for(pcl::PointXYZ point : cloud->points)
    {
        viewer->addLine(pcl::PointXYZ(origin.x, origin.y, origin.z), point,1,0,0,"ray"+std::to_string(countRays));
        countRays++;
    }
}
```

17. Templates and Different Point Cloud Data

Why Use Templates?

The lidar scan function used previously produced a pcl PointCloud object with pcl::PointXYZ points. The object uses a template because there are many different types of point clouds: some that are 3D, some that are 2D, some that include color and intensity. Here you are working with plain 3D point clouds so PointXYZ is used. However, later in the course you will have points with an intensity component as well.

Instead of defining two separate functions one with an argument for PointXYZ and the other for PointXYZI, templates can automate this process. With templates, you only have to write the function once and use the template like an argument to specify the point type.

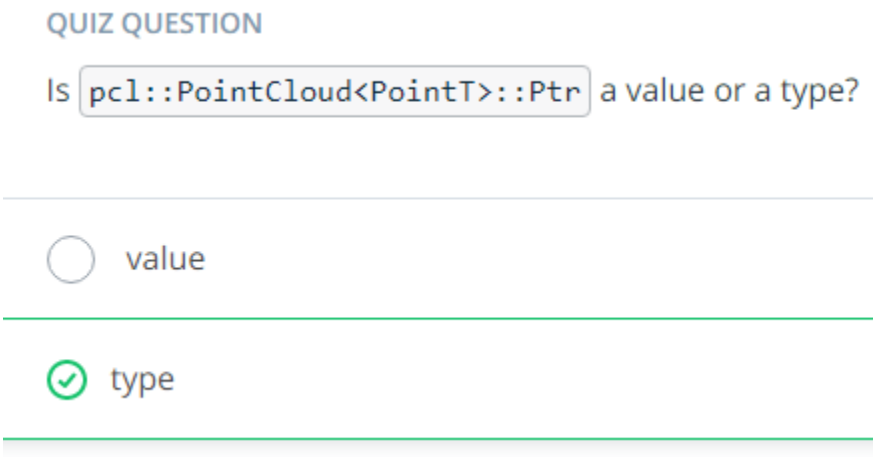
Templates and Pointers

If you haven't used templates with pointers before, you may have noticed in the code that `typename` is used whenever a pointer is used that depends on a template. For example in the function signature here:

```
typename pcl::PointCloud<PointT>::Ptr
ProcessPointClouds<PointT>::FilterCloud(typename pcl::PointCloud<PointT>::Ptr
cloud, float filterRes, Eigen::Vector4f minPoint, Eigen::Vector4f maxPoint)
```

The reason for this is the following: Given a piece of code with a type name parameter, like `pcl::PointCloud<PointT>::Ptr`, the compiler is unable to determine if the code is a value or a type without knowing the value for the type name parameter. The compiler will assume that the code represents a value. If the code actually represents a typename, you will need to specify that.

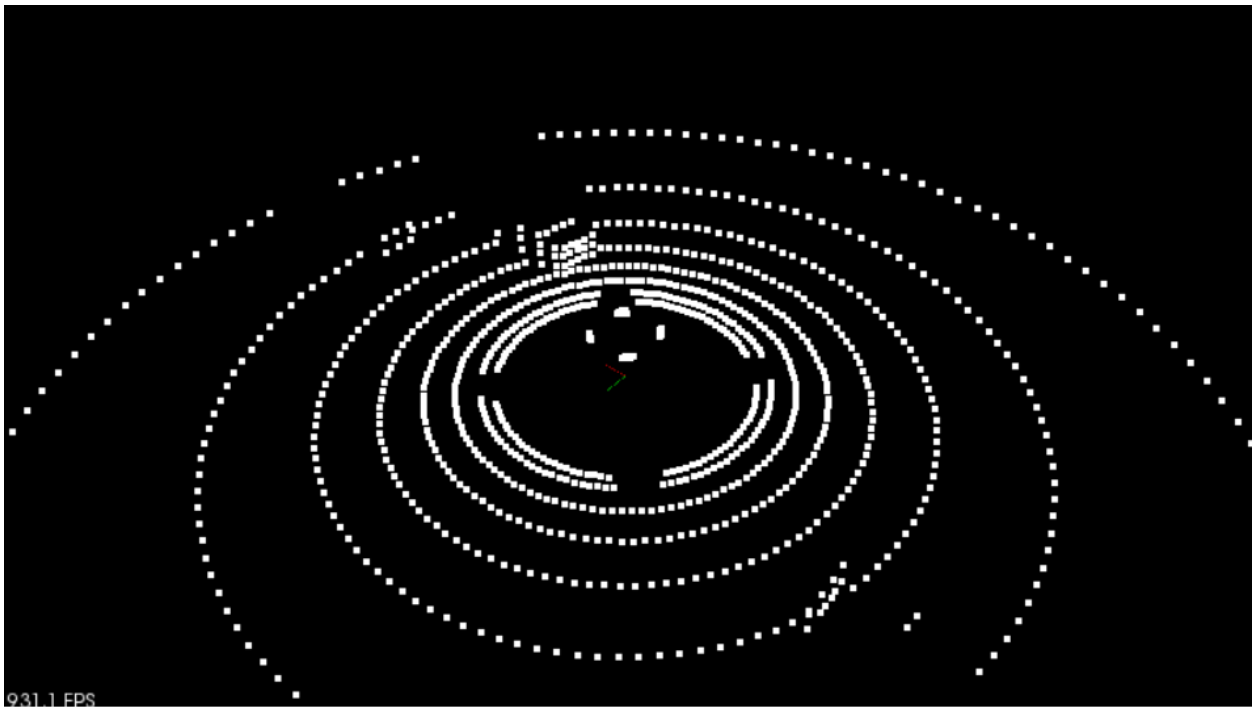
Test your own intuition with the quiz below. You can use this documentation for help.



18. Adjusting Lidar Parameters

numLayers: changes the number of horizontal inclined layers from laser
horizontalLayerIncrement: number of rotational layer groups
minDistance: remove lasers that are hitting objects at this distance
Noise: adds some noisy data so the circular rings are more scrambled

19. Examining the Point Cloud



Clusters can be objects like cars or noise.

Lesson 2: Point Cloud Segmentation

To locate objects in point clouds, we need to distinguish obstacles. For a straight road, it is fairly simple to pick out road points from non-road points. To do this we will use a method called Planar Segmentation which uses the RANSAC (random sample consensus) algorithm.

Point processor objects have methods including filtering, segmentation, clustering, and load/save pcd. It has templates for different PC types.

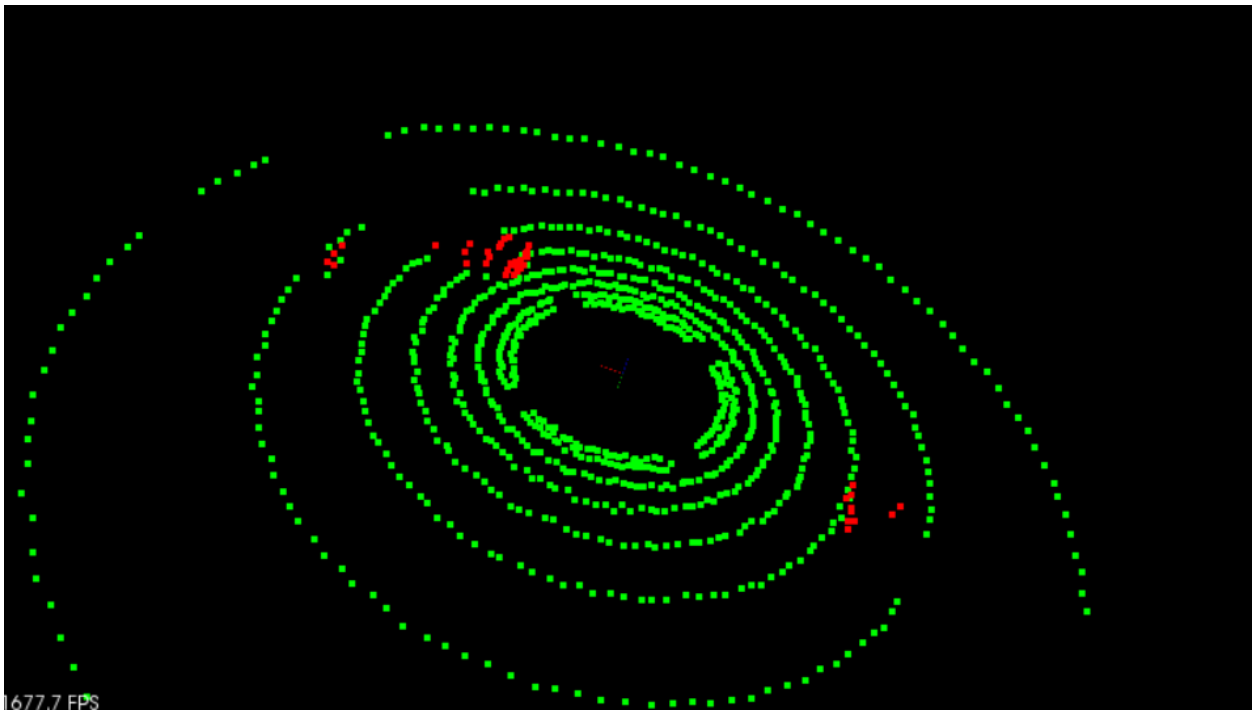
4. Segmenting the Plane with PCL

```
std::pair<typename pcl::PointCloud<PointT>::Ptr, typename
pcl::PointCloud<PointT>::Ptr> SegmentPlane(typename pcl::PointCloud<PointT>::Ptr
cloud, int maxIterations, float distanceThreshold);
```

The function accepts a point cloud, max iterations, and distance tolerance as arguments. Segmentation uses an iterative process. More iterations have a chance of returning better results but take longer. The segmentation algorithm fits a plane to the points and uses the distance tolerance to decide which points belong to that plane. A larger tolerance includes more points in the plane.

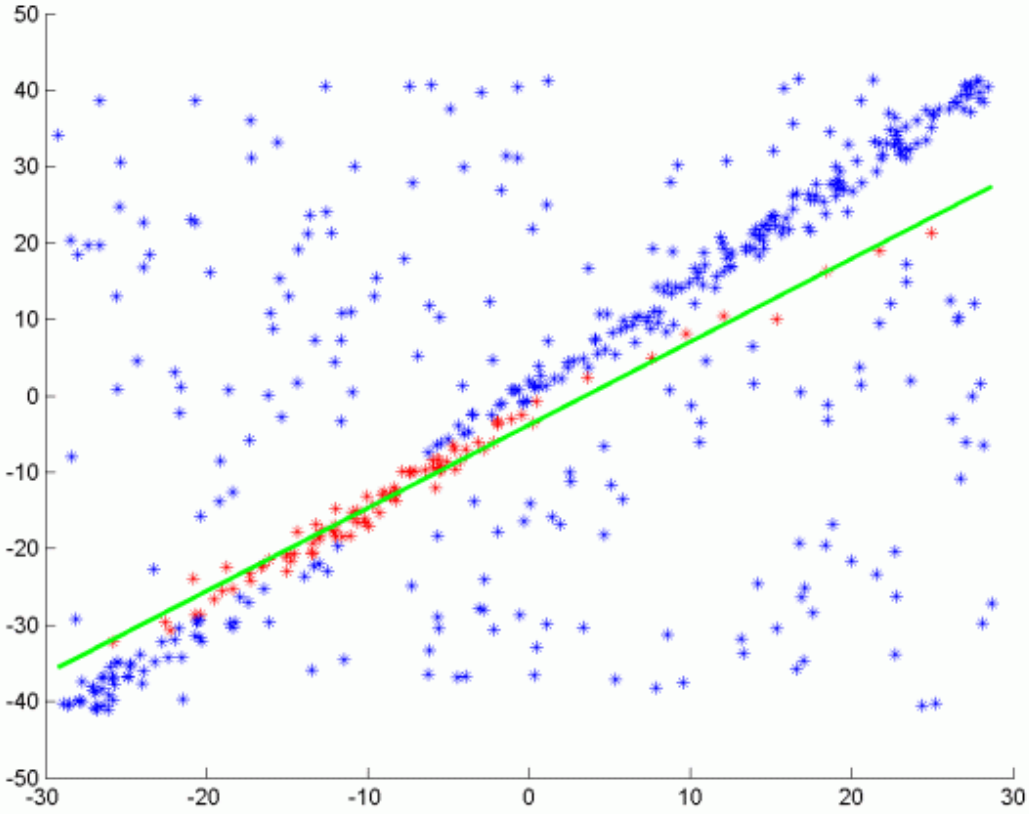
Have a look at the return type in the code above. `SegmentPlane` will return a `std::pair` holding point cloud pointer types. If you are not familiar with pairs check out the documentation here. You will use the pair object to hold your segmented results for the obstacle point cloud and the road point cloud. This way, you can later visualize both point clouds in the pcl viewer and analyze the results.

Inliers are a vector of integers that will separate the obstacles we want (cars, trees) from a plane (road). Obstacles are highlighted in red while the road is segmented out of the overall point cloud data.



6. RANSAC

RANSAC = Random Sample Consensus - used to detect outliers in data.
An iterative method which randomly picks a subset of points until it finds the best line to fit the most inliers.



One type of RANSAC version selects the smallest possible subset of points to fit. For a line, that would be two points, and for a plane three points. Then the number of inliers are counted, by iterating through every remaining point and calculating its distance to the model. The points that are within a certain distance to the model are counted as inliers. The iteration that has the highest number of inliers is then the best model.

Other methods of RANSAC could sample some percentage of the model points, for example 20% of the total points, and then fit a line to that. Then the error of that line is calculated, and the iteration with the lowest error is the best model. This method might have some advantages since not every point at each iteration needs to be considered. It’s good to experiment with different approaches and time results to see what works best.

Lesson 4: Working with Real PCD

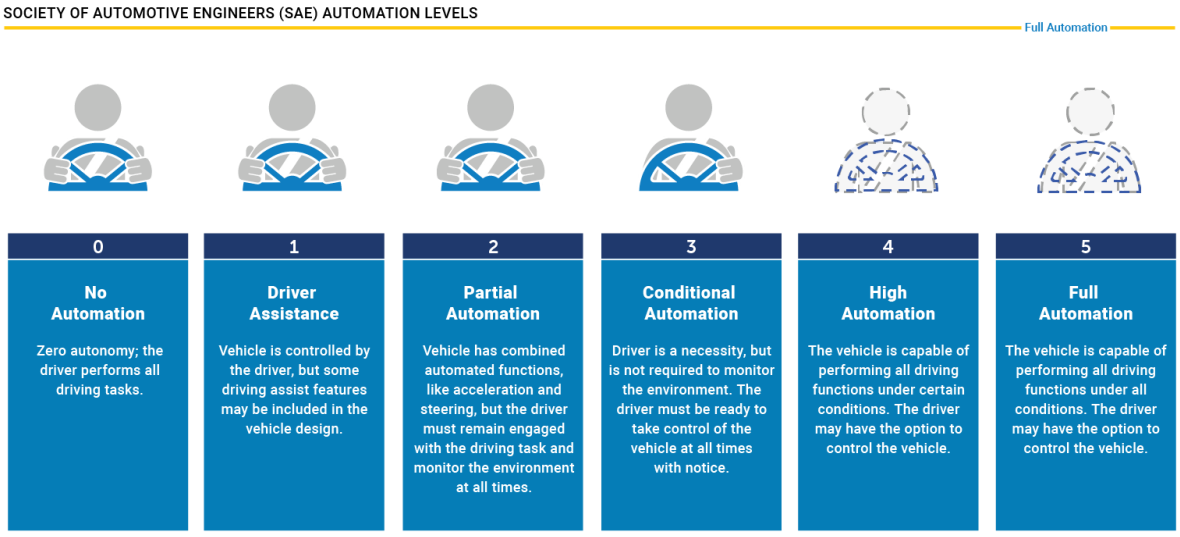
5. Filtering with PCL (Point Cloud Library)

Because there are many points in a PCD, it's best to filter the cloud down so the processor pipeline can digest point clouds efficiently. There are two methods to do this:

1) Voxel Grid
    a) Voxel grid filtering will create a cubic grid and will filter the cloud by only leaving a single point per voxel cube, so the larger the cube length the lower the resolution of the point cloud.
2) Region of Interest
    a) A boxed region is defined and any points outside that box are removed.
    b) To apply these methods you will fill in the point process function `FilterCloud`. The arguments to this function will be your input cloud, voxel grid size, and min/max points representing your region of interest. The function will return the downsampled cloud with only points that were inside the region specified. To get started check out the documentation from PCL for voxel grid filtering and region of interest.

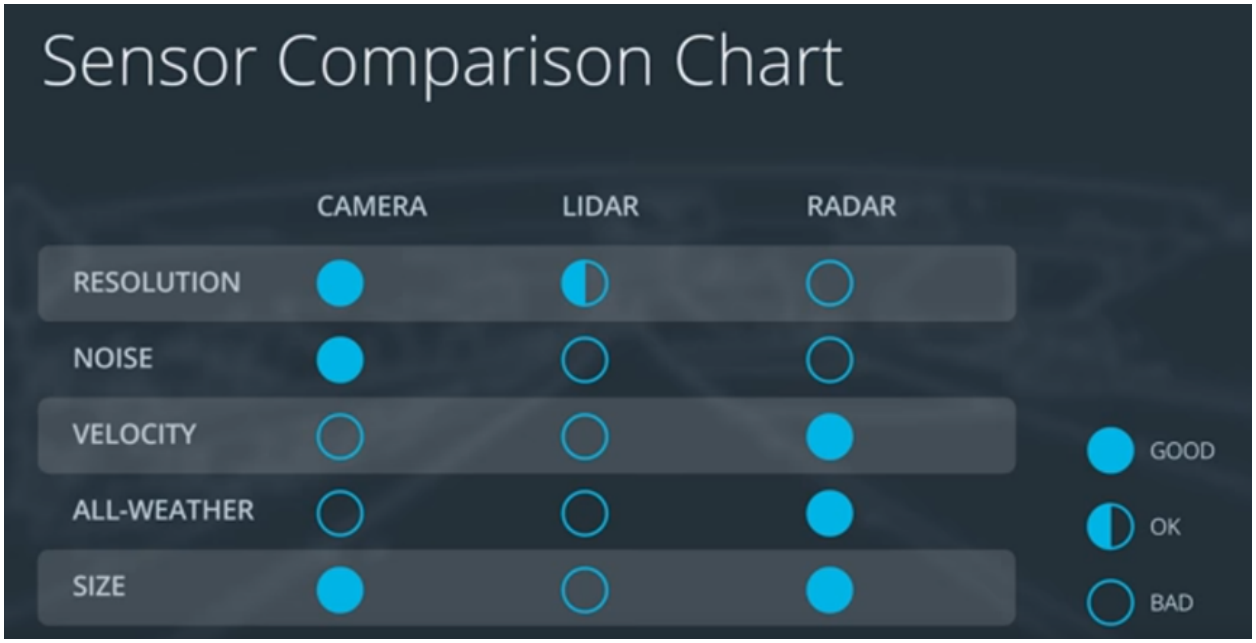
Section 3: Cameras

2.1 Levels of Autonomous Vehicle Driving



LEVELS OF AUTOMATION	WHO DOES WHAT, WHEN
Level 0	The human driver does all the driving.
Level 1	An advanced driver assistance system (ADAS) on the vehicle can sometimes assist the human driver with either steering or braking/accelerating, but not both simultaneously.
Level 2	An advanced driver assistance system (ADAS) on the vehicle can itself actually control both steering and braking/accelerating simultaneously under some circumstances. The human driver must continue to pay full attention ("monitor the driving environment") at all times and perform the rest of the driving task.
Level 3	An automated driving system (ADS) on the vehicle can itself perform all aspects of the driving task under some circumstances. In those circumstances, the human driver must be ready to take back control at any time when the ADS requests the human driver to do so. In all other circumstances, the human driver performs the driving task.
Level 4	An automated driving system (ADS) on the vehicle can itself perform all driving tasks and monitor the driving environment – essentially, do all the driving – in certain circumstances. The human need not pay attention in those circumstances.
Level 5	An automated driving system (ADS) on the vehicle can do all the driving in all circumstances. The human occupants are just passengers and need never be involved in driving.

2.2 Autonomous Vehicle Sensor Sets



1. **Cameras**: The roof cameras are able to focus both close and far field, watch for braking vehicles, crossing pedestrians, traffic lights, and signage. The cameras feed their material to a central on-board computer, which also receives the signals of the other

sensors to create a precise image of the vehicle's surroundings. Much like human eyes, the performance of camera systems at night is strongly reduced, which makes them less reliable to locate objects with the required detection rates and positional accuracy.

2. **Radar**: Radar systems emit radio waves which are reflected off of (many but not all) objects. The returning waves can be analyzed with regard to their runtime (which gives distance) as well as their shifted frequency (which gives relative speed). The latter property clearly distinguishes the radar from the other two sensor types as it is the only one who is able to directly measure the speed of objects. Also, radar is very robust against adverse weather conditions like heavy snow and thick fog. Used by cruise control systems for many years, radar works best when identifying larger objects with good reflective properties. When it comes to detecting smaller or "soft" objects (humans, animals) with reduced reflective properties, the radar detection performance drops.

3. **Lidar**: Lidar works in a similar way to radar, but instead of emitting radio waves it uses infrared light. The roof-mouted sensor rotates at a high velocity and builds a detailed 3D image of its surroundings. In case of the Velodyne VLS-128, a total of 128 laser beams is used to detect obstacles up to a distance of 300 meters. During a single spin through 360 degrees, a total of up to 4 million datapoints per second is generated. Similar to the camera, Lidar is an optical sensor. It has the significant advantage however of "bringing its own light source", whereas cameras are dependent on ambient light and the vehicle headlights. It has to be noted however, that Lidar performance is also reduced in adverse environmental conditions such as snow, heavy rain or fog. Coupled with low reflective properties of certain materials, a Lidar might thus fail at generating a sufficiently dense point cloud for some objects in traffic, leaving only a few 3D points with which to work. It is thus a good idea to combine Lidar with other sensors to ensure that detection performance is sufficiently high for autonomous navigation through traffic.

On the importance of forward-looking cameras

It has to be noted however that in all sensor setups, be it Uber, Tesla, Waymo or traditional manufacturers such as Mercedes or Audi, cameras are always used. Even though there is an ongoing debate on whether radar or Lidar or a combination of the two would be best, cameras are never in question.

Sensor Selection Criteria

1. **Range** : Lidar and radar systems can detect objects at distances ranging from a few meters to more than 200m. Many Lidar systems have difficulties detecting objects at very close distances, whereas radar can detect objects from less than a meter, depending on the system type (either long, mid or short range) . Mono

cameras are not able to reliably measure metric distance to object - this is only possible by making some assumptions about the nature of the world (e.g. planar road surface). Stereo cameras on the other hand can measure distance, but only up to a distance of approx. 80m with accuracy deteriorating significantly from there.

2. **Spatial resolution** : Lidar scans have a spatial resolution in the order of 0.1° due to the short wavelength of the emitted IR laser light . This allows for high-resolution 3D scans and thus characterization of objects in a scene. Radar on the other hand can not resolve small features very well, especially as distances increase. The spatial resolution of camera systems is defined by the optics, by the pixel size on the image and by its signal-to-noise ratio. Details on small object are lost as soon as the light rays emanating from them are spread to several pixels on the image sensor (blurring). Also, when little ambient light exists to illuminate objects, spatial resolution decreases as objects details are superimposed by increasing noise levels of the imager.
3. **Robustness in darkness** : Both radar and Lidar have an excellent robustness in darkness, as they are both active sensors. While daytime performance of Lidar systems is very good, they have an even better performance at night because there is no ambient sunlight that might interfere with the detection of IR laser reflections. Cameras on the other hand have a very reduced detection capability at night, as they are passive sensors that rely on ambient light. Even though there have been advances in night time performance of image sensors, they have the lowest performance among the three sensor types.
4. **Robustness in rain, snow, fog** : One of the biggest benefits of radar sensors is their performance under adverse weather conditions. They are not significantly affected by snow, heavy rain or any other obstruction in the air such as fog or sand particles. As an optical system, Lidar and camera are susceptible to

adverse weather and its performance usually degrades significantly with increasing levels of adversity.

5. **Classification of objects** : Cameras excel at classifying objects such as vehicles, pedestrians, speed signs and many others. This is one of the prime advantages of camera systems and recent advances in AI emphasize this even stronger. Lidar scans with their high-density 3D point clouds also allow for a certain level of classification, albeit with less object diversity than cameras. Radar systems do not allow for much object classification.
6. **Perceiving 2D structures** : Camera systems are the only sensor able to interpret two-dimensional information such as speed signs, lane markings or traffic lights, as they are able to measure both color and light intensity. This is the primary advantage of cameras over the other sensor types.
7. **Measure speed** : Radar can directly measure the velocity of objects by exploiting the Doppler frequency shift. This is one of the primary advantages of radar sensors. Lidar can only approximate speed by using successive distance measurements, which makes it less accurate in this regard. Cameras, even though they are not able to measure distance, can measure time to collision by observing the displacement of objects on the image plane. This property will be used later in this course.
8. **System cost** : Radar systems have been widely used in the automotive industry in recent years with current systems being highly compact and affordable. The same holds for mono cameras, which have a price well below US$100 in most cases. Stereo cameras are more expensive due to the increased hardware cost and the significantly lower number of units in the market. Lidar has gained popularity over the last years, especially in the automotive industry. Due to technological advances, its cost has dropped from more than US$75,000 to below US$5,000. Many experts predict that the cost of a Lidar module might drop to less than US$500 over the next few years.

9. **Package size** : Both radar and mono cameras can be integrated very well into vehicles. Stereo cameras are in some cases bulky, which makes it harder to integrate them behind the windshield as they sometimes may restrict the driver's field of vision. Lidar systems exist in various sizes. The 360° scanning Lidar is typically mounted on top of the roof and is thus very well visible. The industry shift towards much smaller solid-state Lidar systems will dramatically shrink the system size of Lidar sensors in the very near future.
10. **Computational requirements** : Lidar and radar require little back-end processing. While cameras are a cost-efficient and easily available sensor, they require significant processing to extract useful information from the images, which adds to the overall system cost.

	Range measurement	Robustness in darkness	Robustness in rain, snow, or fog	Classification of objects	Perceiving 2D Structures	Measure speed / TTC	Package size
Camera	-	-	-	++	++	+	+
Radar	++	++	++	-	-	++	+
Lidar	+	++	+	+	-	+	-

QUESTION 1 OF 4

Based on the criteria discussed above, please complete the rating of *spatial resolution* for each sensor by adding +, ++ or -.

Submit to check your answer choices!

SENSOR	SPATIAL RESOLUTION
Camera	+
Radar	-
Lidar	++

QUESTION 2 OF 4

Based on the criteria discussed above, please complete the rating of *robustness in daylight* for each sensor by adding + or ++.

Submit to check your answer choices!

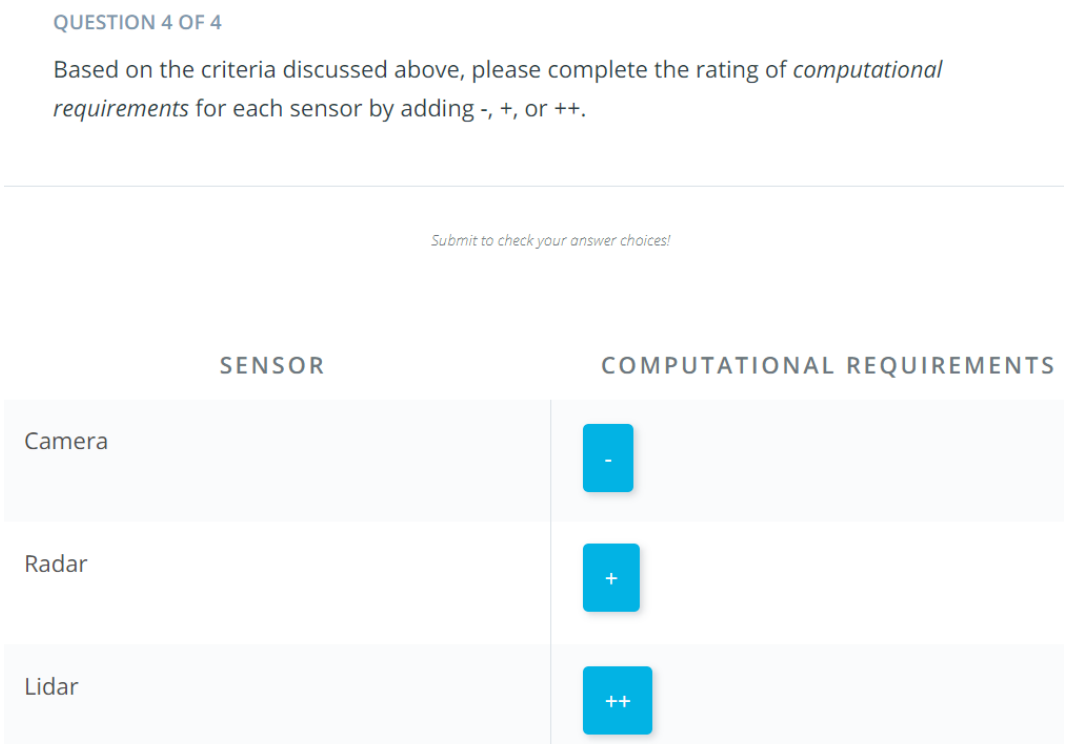
SENSOR	ROBUSTNESS IN DAYLIGHT
Camera	+
Radar	++
Lidar	+

QUESTION 3 OF 4

Based on the criteria discussed above, please complete the rating of *system cost* for each sensor by adding + or -. Here, a higher cost should receive a lower rating.

Submit to check your answer choices!

SENSOR	SYSTEM COST
Camera	+
Radar	+
Lidar	-



2.5 The OpenCV Computer Vision Library

Lesson 4: Tracking Image Features

4.1 Overview

1. Points of interest: describe suitable keypoints by using intensity gradients and image filtering basics. Intensity can be used to describe noteworthy points mathematically.
2. Learn about the Harris detector - a famous computer vision algorithm.
3. Overview of contemporary keypoint detectors and their properties, such as robustness under rotation of the image content and other transformations that occur in practice.
4. Track keypoints from one image to the next to be used in reconstructing vehicle motion. Need to employ “descriptors” which describe the surrounding area near a keypoint in a unique manner. Keypoints can be located again in the next image provided by the camera. There are many types of descriptors available.
5. Need suitable measures to describe the similarities between keypoints.

4.12 Descriptor Matching

https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/

https://stackoverflow.com/questions/16996800/what-does-the-distance-attribute-in-dmatches-mean

https://stackoverflow.com/questions/29870726/brief-implementation-with-opencv-2-4-10

In general, you will notice that the ORB and BRISK generally perform better than other binary descriptors. Whereas in the gradient-based descriptors, SIFT is much better than SURF. Though the results are dependent on the underlying dataset, and the environment you are working in.

https://docs.opencv.org/4.x/d9/d97/tutorial_table_of_content_features2d.html