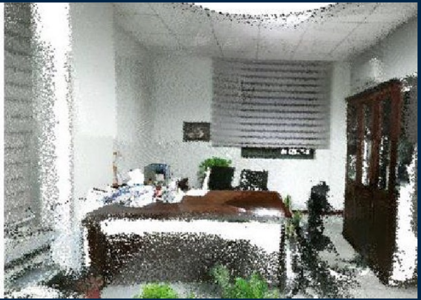
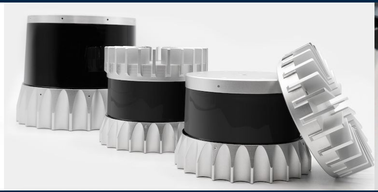
Point cloud and operations



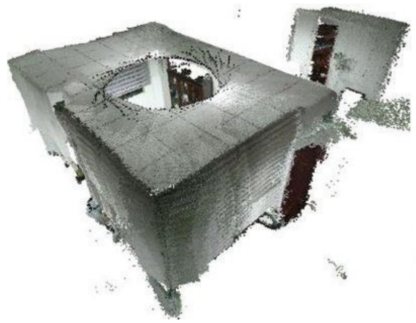
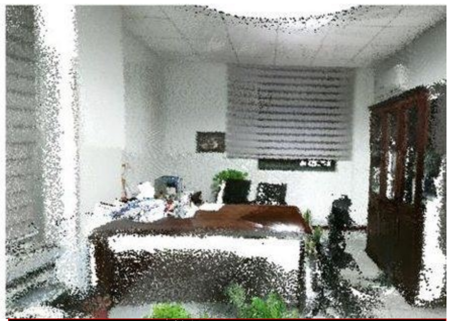
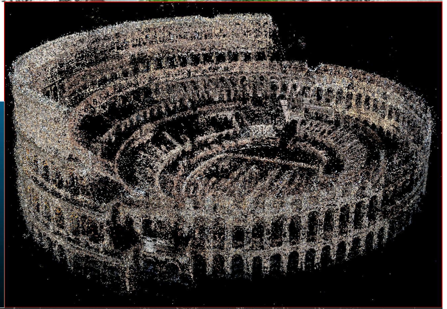
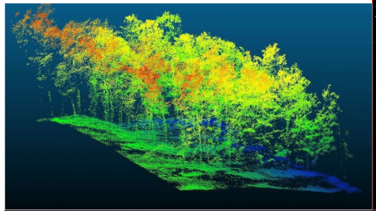
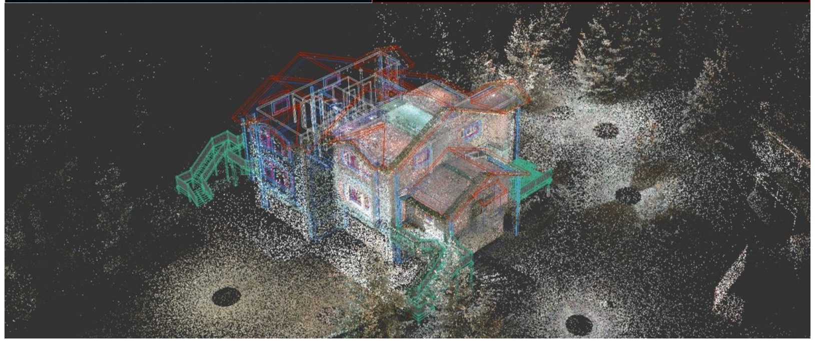


Slides adapted from Peter Ahrendt's

Point Cloud

What is a (3D) point cloud (or pointcloud)?

- A discrete set of data points in the local 3D space
- It represents the local 3D space with each point having a Cartesian coordinates [x, y, z]
- The point cloud point can also encode the color of the point (e.g. obtained by a RGB-D camera or synchronized system Lidar-RGB camera)

Point cloud for robotics

Why robot likes point cloud?

- Obstacle representation
- Geometrical information
- Easy to obtain and understand

Example: Robot Navigation

A typical robot navigation using point cloud

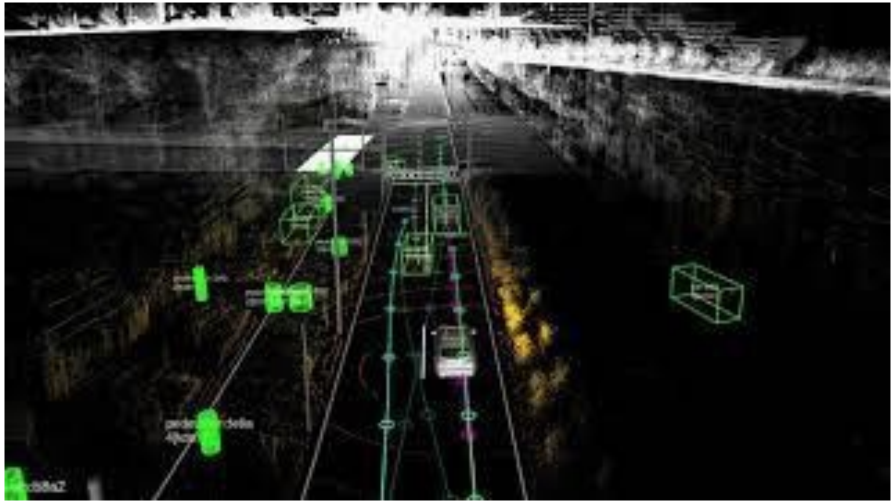
- The robot obtains its pose thanks to a pointcloud-based localization (with combined information also from other sensors)

- The robot uses a voxel map representation building from the point cloud

Example: Object Detection

Object detection and classification using point cloud

- The robot detects an object and its semantic information from its point cloud sensor



Obtaining 3D point cloud

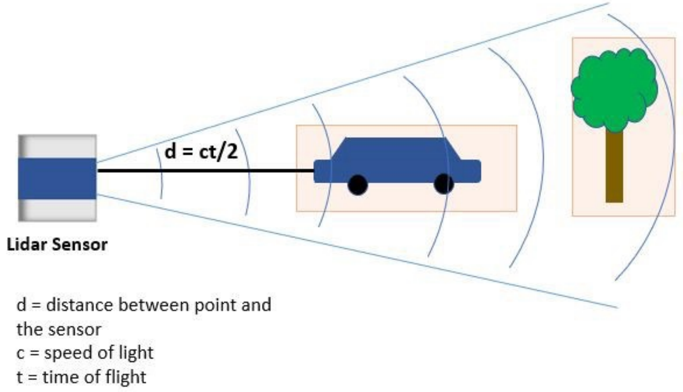
How can 3D point cloud be obtained?

- Using 3D laser scanner and time-of-flight sensors
- Using stereo camera
- Using depth camera

3D Laser Scanners

3D laser scanners can:

- Provide dense 3D point cloud using scanning a time-of-flight sensor
- Often obtained via multiple pulsed light beams at different angles. These pulses bounce off surrounding objects and return to the sensor. The sensor uses the time it took for each pulse to return to the sensor to calculate the distance it traveled.
- Different technologies have been proposed to make this process efficiently

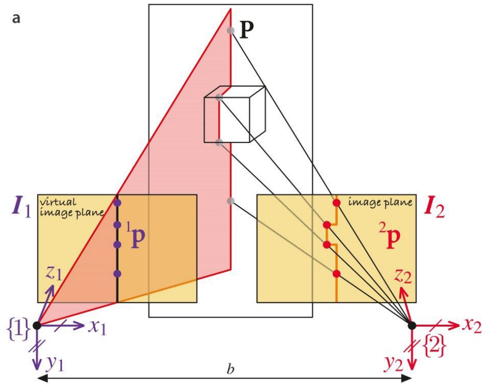


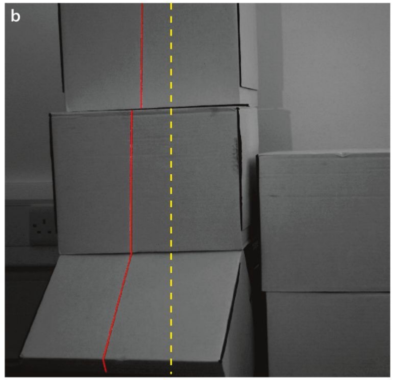
3D Laser Scanners

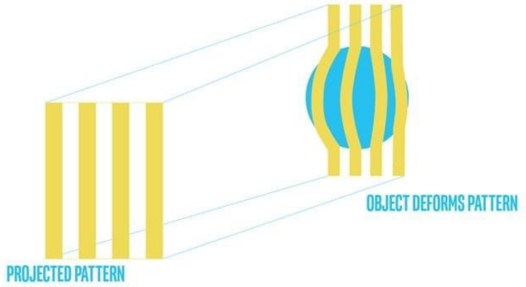
- Effective
- Dense point cloud, with rich information
- Bulky

Structured Light Sensors

- Projected light is patterned (visually or temporally) that will create an image
- The disparity between an expected image and actual image will be used to infer depth information
- Good for indoor, but poor in outdoor

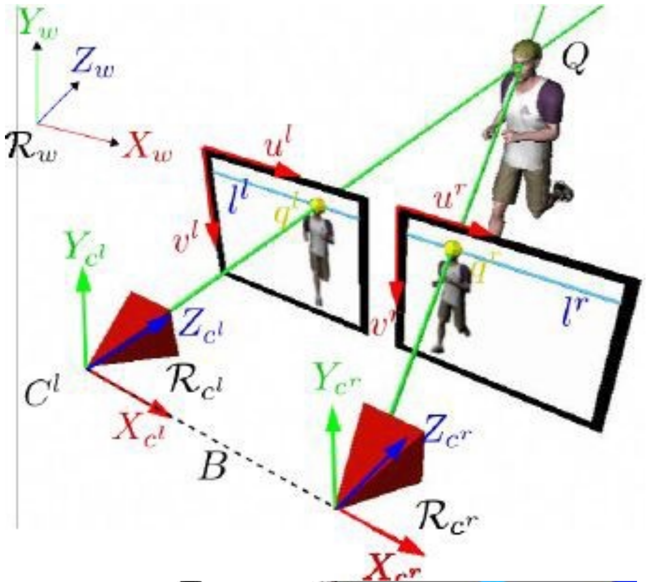


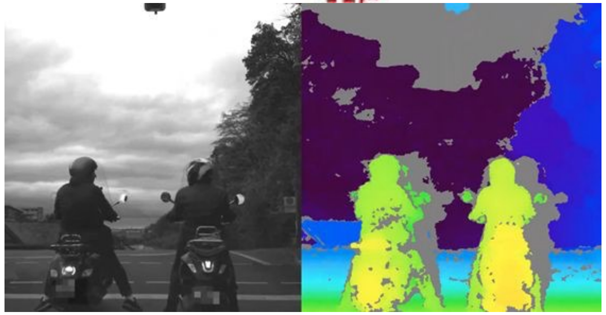




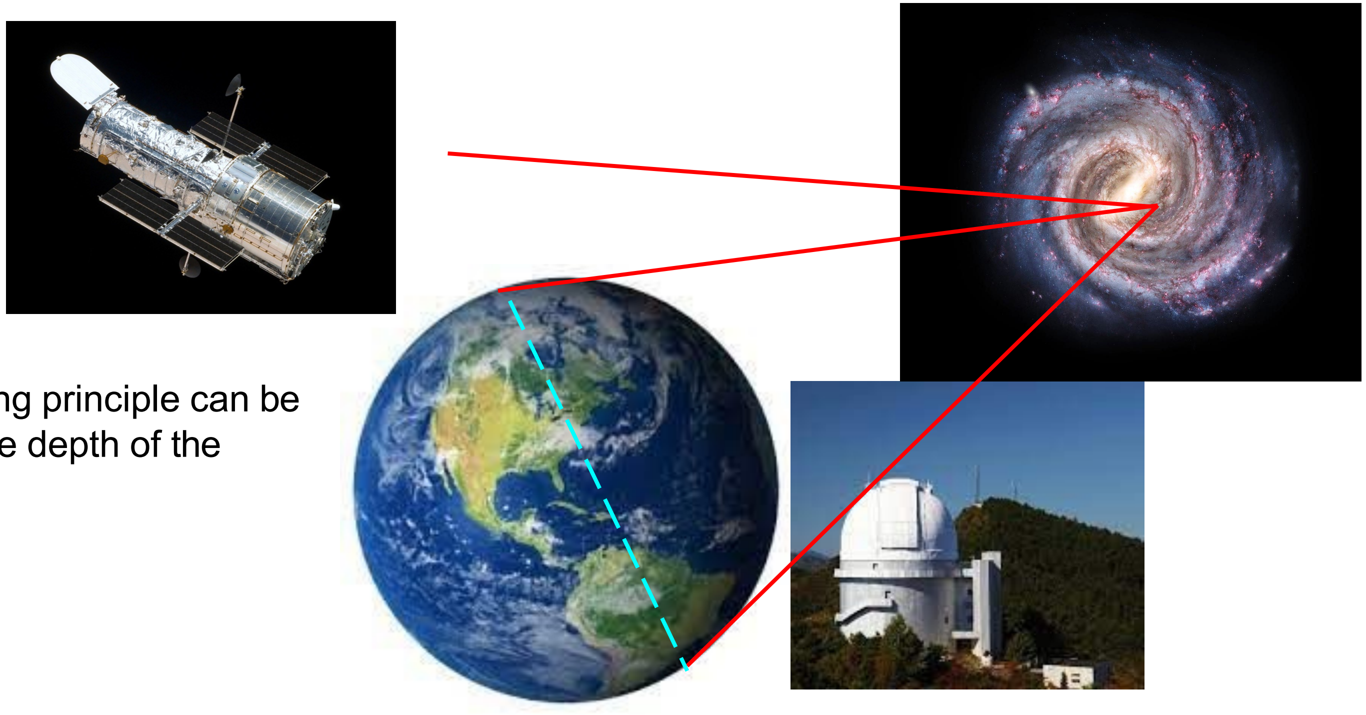
Stereo Vision

- A (passive) solution to find the depth information of a scene through triangulations.
- Does not have any restriction and difficulties in deployments (e.g. can work outdoor)
- Once calibrated, it can work with arbitrary ranges
- The output is a bit noisy
- Computationally demanding as triangulation calculation and association can be expensive





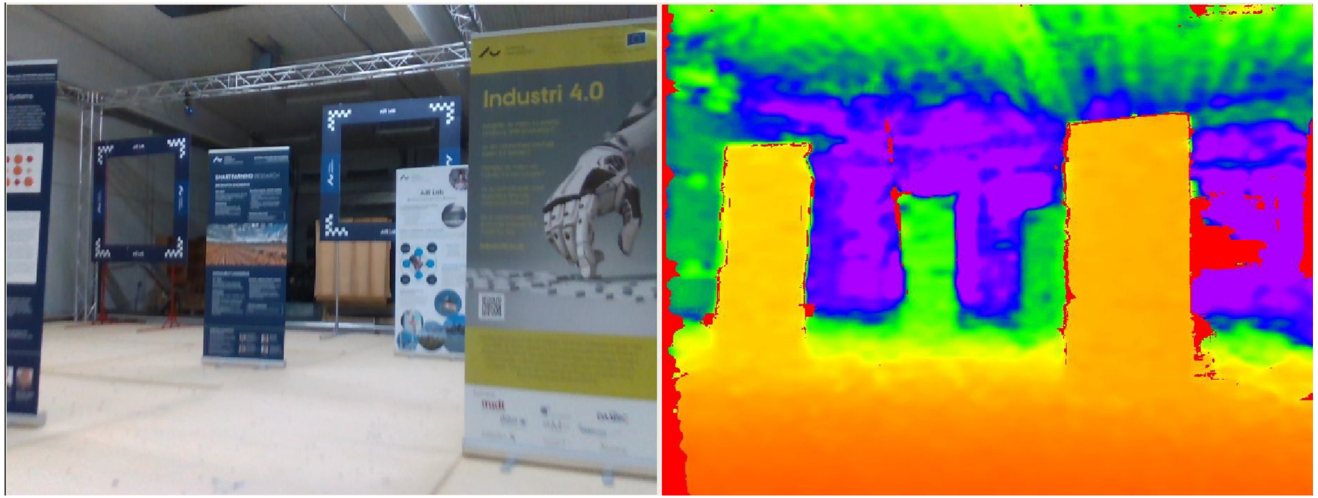
Stereo Vision for Depth of the Universe



Stereo vision working principle can be used to measure the depth of the universe!

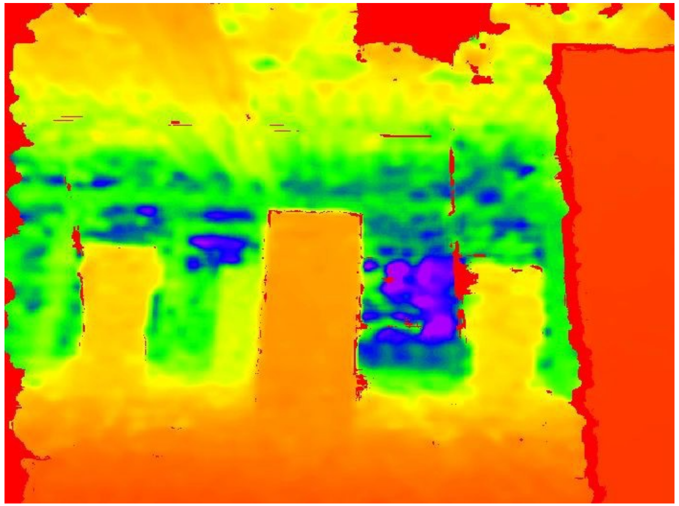
Depth camera

- Modern depth camera uses stereo depth by inferring the depth from stereo vision while actively projecting infrared lights onto the scene to improve the accuracy of the data



Depth map to point cloud

- Unlike normal visual image, for each pixel (X, Y), a depth map also stores additional information (Z): that is the distance from the camera to the scene object
- It is often encoded as a 32-bit float value, but can be rescaled to [0, 255] to display as an "image"
- This depth map can be readily converted into point cloud

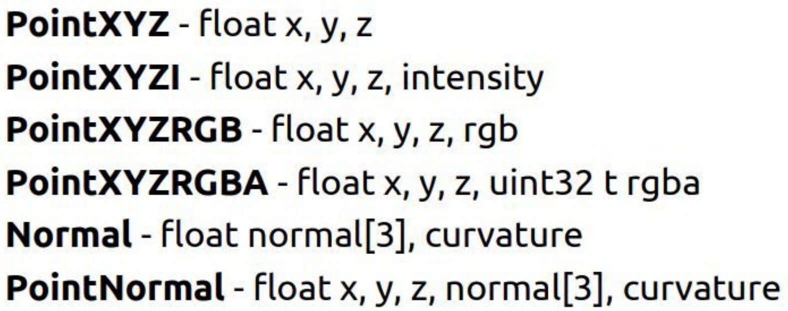


Point Cloud Data Structures

- Organizing large pointcloud for robots in real environments normally requires powerful libraries, such as PCL
- There are two types: ordered or unordered
- They can contain more information: intensity, colors, etc…

See also: [ROS PointCloud2 message]()

pointcloudlibrary



PointXYZ	- float x, y, z
PointXYZI	- float x, y, z, intensity
PointXYZRGB	- float x, y, z, rgb
PointXYZRGBA	- float x, y, z, uint32 t rgba
Normal	- float normal[3], curvature
PointNormal	- float x, y, z, normal[3], curvature

Fitting a plane

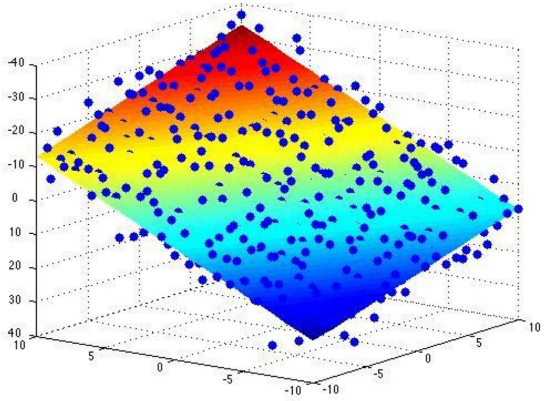
- The radii of the ellipsoid are the eigenvalues of the inertial matrix of the points J, with the eigenvector corresponding to the smallest eigenvalue is the direction of the minimum radius which is the normal to the plane.
- The inertial matrix can be calculated as follows (**P** denotes the coordinate of each point):

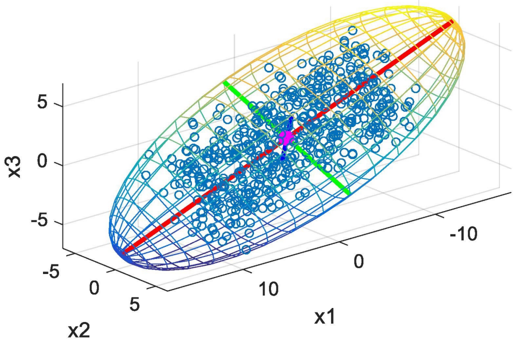
$$J = \sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T$$

$$\boldsymbol{x} = \boldsymbol{P}_i - \overline{\boldsymbol{P}}$$

$$\overline{\boldsymbol{P}} = {}^1\!/_N \sum_{i=1}^{N} \boldsymbol{P}_i$$

Centroid of the points





Point cloud registration

- This is the problem to find a way for matching two sets of points
- This can be used to determine the rigid body motion between two data frames: the current model M and new observed, noisy data D
- It can be solved by matching points between the two sets (finding correspondence)
- An effective solution can be found by an approach called iterated closest point (ICP) (See Rusinkiewicz et al, 2001)

$$ {}^{D}\xi_{M}^{*} = \arg\min_{\xi} \sum_{i,j} \left\| \boldsymbol{D}_j - \xi \bullet \boldsymbol{M}_i \right\| $$

Algorithm 1. Iterative Closest Point Algorithm Iteration

```
  Data: Model set: $X$ with $n_X$ points and dimension $d_X$.
  Data set: $P_k$ with $n_P$ points and dimension $d_P$.
  Result: Transformed data set $P_{k+1}$
1 Compute the closest points $Y_k = C(P_k, X)$;
2 Compute centers of mass $\mu_P = col.means(P), \mu_X = col.means(Y_k)$;
3 Compute the cross-covariance matrix $\Sigma_{PX}$;
4 Compute rotation $R = VU^T$, where $UWV^T = SVD(\Sigma_{PX})$;
5 Compute translation $q_T = \mu_X - R\mu_P$;
  // Applying transformation
6 for $i = 1..n_P$ do
7     $P_{k+1,i} = RP_{k,i} + q_T$;
```

Pointcloud and Robotics

See more:

- LOAM: Lidar Odometry and Mapping in Real-time

SOLIDUM PETIT IN PROFUNDIS · UNIVERSITAS ARHUSIENSIS ·