ISTANBUL TECHNICAL UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS

Mapping Object Detections from RGB Images to 3D Coordinates with Point Clouds

Graduation Project Final Report

Onat Şahin
150150129

Department: Computer Engineering
Division: Computer Engineering

Advisor: Prof. Dr. Hazım Kemal Ekenel

January 2020

Statement of Authenticity

I/we hereby declare that in this study

1. all the content influenced from external references are cited clearly and in detail,
2. and all the remaining sections, especially the theoretical studies and implemented software/hardware that constitute the fundamental essence of this study is originated by my/our individual authenticity.

İstanbul, Ocak 2020

Onat Şahin

Acknowledgments

Special thanks to my project advisor Prof. Dr. Hazım Kemal Ekenel for his guidance and feedback throughout the development of the project. I also would like to thank Berkin Malkoç for his constant support and the resources he provided, which made the project possible. Finally, I would like to thank my mother, my father and my friends for their motivation and support.

Mapping Object Detections from RGB Images to 3D Coordinates with Point Clouds

(SUMMARY)

Object detection is a task in computer vision that aims to detect object instances of certain classes in images [1]. 3D object detection, on the other hand, is the task of detecting object instances in 3D space with the usage of 3D data like point clouds and depth images. 3D object detection is essential for autonomous vehicles and robotics, since projects in these areas of research need to perceive the environment in 3D.

In the literature, there are four different approaches to 3D object detection [1]. The first approach uses 2D images and geometry. The models that use this approach infer 3D bounding boxes by processing RGB images and adding shape and geometric prior or occlusion patterns. The second approach makes detections directly from raw point clouds. Raw point clouds are encoded or transformed in transformation layers so that the detections will be transformation invariant. Multi-layer perceptrons are used together with these layers to solve various tasks like detection, segmentation and classification. The third approach divides point clouds to fixed sized 3D voxels with a grid to give a structure to the point cloud. Finally, the last approach projects the point clouds on a 2D plane and make detections on the projection. Bird's eye view is generally used for these projections. An additional approach is proposed by Frustum PointNets [2], which first detects object from RGB images and then infers 3D bounding boxes by using frustum point clouds of the 2D detections, which are generated with depth images.

The aim of this project is to create a system that can perform 3D object detection using a point cloud of an area, RGB images taken from the same area and camera metadata that includes camera location and pose for each RGB image. For 3D detection, objects are first detected from RGB images. Then, with the use of the point cloud and the camera's location and orientation for each image with detections, these image detections are converted to point cloud detections with 3D coordinates. The reason for using this approach is the high performance of regular object detection networks. With this system, it is possible to use any 2D object detection network to make detections in 3D as well. Also, unlike other 3D object detectors that use point clouds, this approach does not require neural network training with point clouds.

In this system, an object detection network for images is selected and trained. Then, a C++ code that uses PCL(Point Cloud Library) [3] to map the center pixels of image object detection bounding boxes to 3D coordinates using point clouds is implemented. The output of the system is predictions for center points of objects in the point cloud. This approach can be used in real life scenarios by obtaining the point cloud of an area and RGB images with camera poses for each image using an RGB-D camera. It is possible to obtain these data with an Intel Realsense D435 using its drivers, ROS [4] and RTABMap [5].

To test the system, Stanford's 2D-3D-S dataset [6] is used. This dataset collects data for 6 large-scale indoor areas and includes point clouds, meshes, RGB images, depth images, camera information and instance level annotations for both 2D and 3D data.

Faster R-CNN [7] and RetinaNet [8] 2D object detection network models are trained for the first 4 areas of the dataset and center pixels of the bounding boxes are mapped to 3D coordinates. Success of the 2D object detectors are evaluated with PASCAL VOC [9] metric, while success of the 3D mappings are measured with a custom evaluation. As a result, the approach was successful in predicting center points of objects with a simple shape like boards, sofas and tables. However, it was unsuccessful in predicting center points of objects with complex shapes like chairs. It also became apparent that the used 3D mapping method and its custom evaluation have lacking parts and shortcomings, which should be fixed in future work.

RGB İmajlardan Bulunan Objelerin Nokta Bulutları ile 3B Koordinatlara Atanması

(ÖZET)

Nesne tanıma problemi, imajlardan belirlenmiş sınıflardaki nesne örneklerini bulmayı amaçlayan bir bilgisayar görüsü problemidir [1]. Bir diğer yandan, 3 boyutlu nesne tanıma ise nokta bulutu ya da derinlik görüntüleri gibi 3 boyutlu verilerin kullanımı ile nesne örneklerini 3 boyutlu uzayda tanımayı amaçlar. 3 boyutlu nesne tanıma otonom araç geliştirme ve robotik için çok önemlidir çünkü bu alanlardaki projelerde çevrenin 3 boyutlu olarak algılanması gerekir.

Literatürde 3 boyutlu nesne tanıma problemini çözmek için dört farklı yöntem görmek mümkündür [1]. Bunlardan birincisi 2 boyutlu resimler ve geometri kullanmaktadır. Bu yöntemi kullanan modeller RGB resimleri işler ve önceden belirlenmiş şekil ile geometrik yapılar kullanarak 3 boyutlu sınırlayıcı kutular tahmin eder. İkinci yöntem tahminleri direk nokta bulutları üzerinden yapar. Bu tarz yöntemlerde yapılan tahminlerin geometrik dönüşümlerden etkilenmemesi için nokta bulutları dönüşüm katmanlarında kodlanır. Bu dönüşüm katmanları ile birlikte çok katmanlı algılayıcılar kullanarak nesne tanıma, sınıflandırma ve segmentasyon gibi bir çok problem çözülebilir. Üçüncü yöntemde nokta bulutları eşit boyutlu voksellere ayrılarak yapılandırılır. Son olarak, dördüncü yöntemde nokta bulutlarının bir düzleme izdüşümleri alınır ve bu izdüşümler üzerinde 2 boyutlu nesne tanıma yöntemleri uygulanır. İzdüşüm olarak genellikle kuş bakışı perspektifi kullanılır. Bunlara ek olarak farklı bir yöntem "Frustum PointNets [2]" isimli çalışmada gözlenebilir. Bu yöntemde nesneler önce RGB resimler üzerinde bulunur. Daha sonra, derinlik görüntüleri yardımıyla RGB resimlerde bulunan nesnelerin konik nokta bulutları oluşturulur. Bu bulutlar üzerinde önceden bahsedilen ikinci yöntem kullanılarak 3 boyutlu sınırlayıcı kutular tahmin edilir.

Bu projede amaç nokta bulutu, nokta bulutunun oluşturulduğu alanı kapsayan RGB resimler ve kamera pozları kullanarak 3 boyutlu nesne tanıma yapabilen bir sistem yaratmaktır. Nesneler ilk önce RGB resimlerden bulunur. Daha sonra nokta bulutu ve bulunmuş obje barındıran resimlerin kaydedildiği andaki kameranın pozu kullanılarak RGB resimlerdeki deteksiyonlar 3 boyutlu nokta bulutu deteksiyonuna dönüştürülür. Bu yöntemin seçilme sebebi RGB resimler üzerinde çalışan nesne tanıma nöral ağ modellerinin yüksek performansıdır. Bu sistem 2 boyut üzerinde çalışan herhangi bir nesne tanıma ağı ile 3 boyutlu nesne tanıma da yapabilmeyi mümkün kılmaktadır. Ayrıca, nokta bulutları kullanan diğer yöntemlerin aksine, bu yöntem nokta bulutları ile nöral ağ eğitimi yapmamaktadır.

Bu sistemde imajlar üzerinde nesne tanıma yapan bir nöral ağ mimarisi seçilir ve eğitilir. Daha sonra, PCL(Point Cloud Library, Nokta Bulutu Kütüphanesi) [3] kullanarak RGB imajlardaki deteksiyonların merkez piksellerini nokta bulutu deteksiyonlarına dönüştüren bir C++ kodu kullanılır. Sistemin çıktısı nokta bulutundaki objelerin orta noktaları için yapılan tahminlerdir. Bu yöntem bir RGB-D kamera kullanarak gerçek hayat problemlerinde kullanılabilir. Örnek olarak Intel Realsense D435 kamerası ile beraber ROS [4] ve RTABMap [5] kullanarak yöntem için gerekli olan nokta bulutu, RGB imajlar ve kamera bilgisi elde edilebilir.

Sistemi test etmek için Stanford Üniversitesi'nin 2D-3D-S veri seti [6] kullanıldı. Bu veri seti 6 geniş iç alandan toplanmış veri içermektedir. Bu verilere nokta bulutları, RGB resimleri, derinlik görüntüleri, resimler için kamera bilgileri ve hem resimler hem de nokta bulutları için örnek düzeyinde anotasyonlar dahildir. Bu verideki ilk 4 alandan imajlar ile Faster R-CNN [7] ve RetinaNet [8] modelleri egitildi ve test sonrasında elde edilen sınırlayıcı kutuların merkez piksellerinin 3 boyutlu koordinatları bulundu. 2 boyutlu obje deteksiyonunun başarısı PASCAL VOC [9] metriği ile ölçülürken 3. boyuta yapılan dönüşümün başarısını ölçmek için bu proje için implemente edilmiş bir kod kullanıldı. Sonuç olarak, tahta, masa gibi basit şekilli objelerin merkez noktalarının bulunmasında yüksek performans gözlenirken sandalye gibi daha karmaşık şekilli objelerin merkez noktalarını bulmada kullanılan yöntemin yetersiz kaldığı görüldü. Ayrıca, deteksiyonlarin 3D ye dönüştürülmesi ve bu dönüşümün performansının ölçülmesi için kullanılan yöntemlerde gelecekte düzeltilmesi gereken eksikler belirlendi.

Contents

1 Introduction and Project Summary **1**

2 Comparative Literature Survey **4**
2.1 3D Object Detection 4
2.2 2D Object Detection 6
2.3 2D-3D-S Dataset 7

3 Developed Approach and System Model **9**
3.1 Data Model 9
3.1.1 2D-3D-S Dataset File Structure 9
3.1.2 JSON Files Created From The Dataset 11
3.2 Structural Model 13
3.2.1 2D Object Detection 13
3.2.2 Mapping Detections to 3D coordinates 16
3.2.3 Usage With An Intel Realsense D435 RGB-D Camera 17

4 Experimentation Environment and Experiment Design **18**
4.1 2D Object Detection Test With PASCAL VOC Metric 18
4.2 Testing 3D Center Points 19

5 Comparative Evaluation and Discussion **21**

6 Conclusion and Future Work **26**

1 Introduction and Project Summary

Object detection is a computer vision task that aims to find instances of certain types of objects in images [1]. The term "object detection" generally implies finding object instances in images by enclosing them with bounding boxes. There are also segmentation tasks, which finds objects by predicting the class of each pixel, which results in obtaining more information compared to bounding box object detection, since shapes of the objects are also obtained. There are two main types of image segmentation. The first one is semantic segmentation, which only predicts a class for each pixel of the image, so it is not able to distinguish different instances. The second type of image segmentation, called instance segmentation, distinguishes different instances by predicting an object instance for each pixel instead of predicting just a class. Visualization of these computer vision tasks can be seen in Figure 1.1.

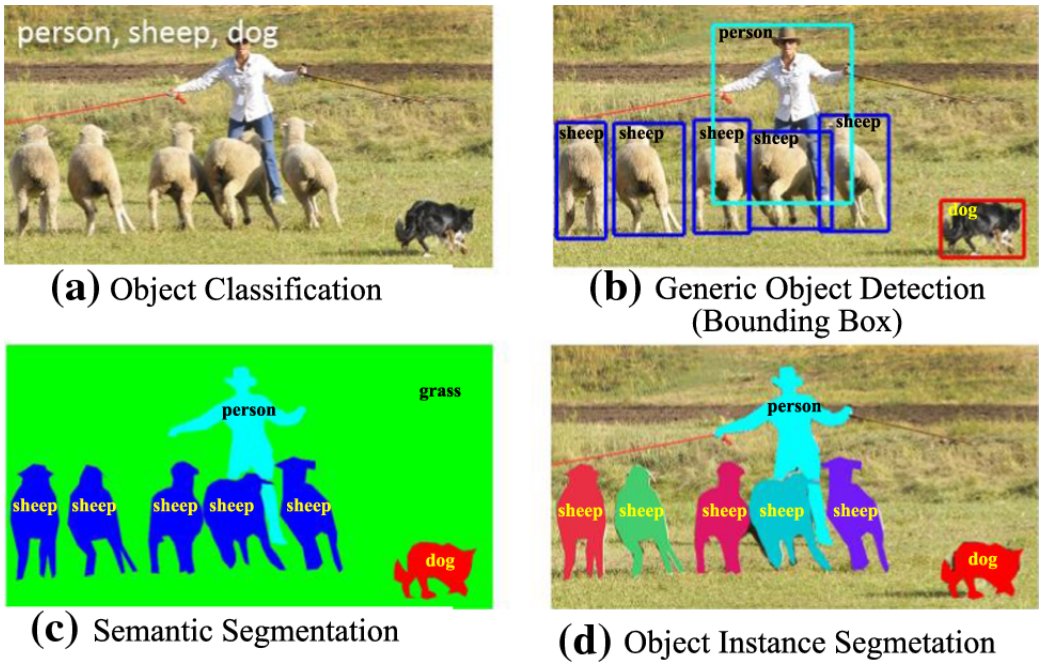
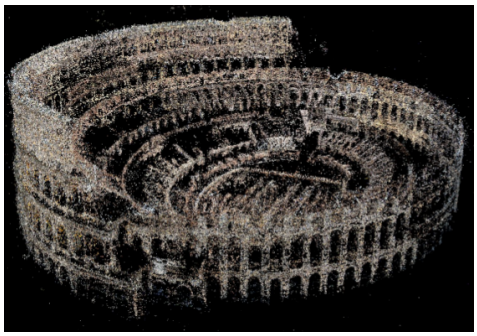


Figure 1.1: Visualization of 2D computer vision tasks related to objects [10]

In recent years, research on performing detection and segmentation tasks on 3D data has gained popularity. Named 3D object detection, this computer vision task aims to detect instances in 3D space using 3D data like RGB-D images and point clouds. Detecting object instances in 3D space makes it possible to trivially derive relations between different object instances, like distances between them. This makes 3D object detection an essential part of robotics and autonomous vehicle development, areas in which understanding the 3D space around an agent is crucial. 3D object detection methods are also used for building parsing and indoor scene understanding. Stanford University's Building Parser [11] can be given as an example to this.

There are different methods for performing 3D object detection, which can be summarized as: methods that use 2D images and geometry, methods that make detections using 3D voxel grids, methods that use bird's eye view projections of point clouds and methods that make detections from raw point clouds [1]. A point cloud is a set



```
# .PCD v0.7 - Point Cloud Data file format
VERSION 0.7
FIELDS x y z rgb normal_x normal_y normal_z curvature
SIZE 4 4 4 4 4 4 4 4
TYPE F F F F F F F F
COUNT 1 1 1 1 1 1 1 1
WIDTH 588874
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 588874
DATA binary
```

Figure 1.2: An example point cloud [12] (left) and the format of a PCD point cloud (right).

of points in 3D space defined by their coordinates. In addition to coordinates, point clouds can hold data like RGB values and normal vectors for each point. A visual representation of a point cloud along with the file format of a PCD point cloud can be seen in Figure 1.2. All these methods will be detailed in the literature review section, but the ones that relate to this project the most are the first type and the last type. The methods that use 2D images and geometry process RGB images and add shape and geometric prior to infer 3D bounding boxes [1], while the methods that use raw point clouds give the point clouds directly to a neural network architecture as a matrix and obtain detections.

This project proposes a system that can perform 3D object detection using a point cloud of an area, RGB images taken from the same area and camera data. The method for making detections can be thought as a mix of the two detection approaches described above. Like the methods that use 2D images and geometry, the proposed method processes RGB images and makes detections on them. But instead of adding geometric prior or shape, information from the provided point cloud of the area is used. The motivation for using 2D images mainly for 3D detection is that object detection on images is a solved problem with many network architectures proved to have high performance on detecting object instances. In addition to this, there is a wider variety of RGB image datasets compared to 3D datasets, which provides flexibility to the use cases of the system. Also, unlike other methods that use point clouds for object detection, this method does not perform neural network training on point clouds.

The proposed system requires a point cloud of an area, RGB images of the same area, and camera metadata that includes the location and pose of the camera at each moment an RGB image is taken. First, objects are detected from RGB images using an object detection neural network architecture trained for detecting certain classes of objects. Then, using the location and pose data, a virtual camera is positioned inside the point cloud in the exact same way as the camera that took the corresponding RGB image. With this virtual camera, a range image of the area that is visible from the RGB image is created. Using these range images, it is possible to find RGB image pixels' 3D coordinates in the point cloud. Using this method for mapping, center pixels of the bounding boxes of object instances detected from the RGB images are mapped to 3D coordinates to approximate the center locations of each object. Since all the required data for the system can be obtained with an RGB-D camera like an Intel Realsense, it is possible to scan an area with a suitable RGB-D camera and perform the described detection as long as an object detector that is trained to detect objects of interest is

available.

For training neural networks for object detection on RGB images, Detectron2 [13], which is a deep learning framework powered by PyTorch [14] is used. Creating range images from point clouds is done with a C++ library called PCL(Point Cloud Library) [3]. The dataset used to test the system is Stanford University's 2D-3D-S Dataset [6], which includes annotated point clouds, RGB images and camera metadata from buildings of Stanford University. More details about this dataset and object detection architectures used with Detectron2 are presented in the Comparative Literature Survey section.

For the evaluating the 2D object detectors, PASCAL VOC metric [9] is used and average precisions for each object class is calculated. Included object classes are: chair, sofa, table, bookcase and board. For the evaluation of predicted 3D center points, a metric that makes an evaluation based on the distance between the predicted point and the object's true center point is proposed. These metrics are detailed in the Experimentation Environment and Experiment Design section. Experiment results based on these metrics and discussions about how to improve them are included in the Comparative Evaluation and Discussion section. Finally, better approaches to improve this project and complete its lacking parts is discussed in the Conclusion and Future Work section.

2 Comparative Literature Survey

2.1 3D Object Detection

The aim of this project is to create a 3D object detection system that can be used with RGB-D cameras, so the main task is performing 3D object detection, which is determining coordinates of objects in 3D space with instance segmentation or 3D bounding boxes. Agarwal et. al [1] describes four main types of 3D object detection methods: Methods that use 2D images and geometry, methods that make detections in raw point clouds, methods that make detections using a 3D voxel grid and methods that make detections in 2D after projecting the point cloud to a 2D plane. The same paper describes the first approach as methods that first process 2D images, and then add shape and geometric prior or occlusion patterns to predict 3D bounding boxes. Deep MANTA [15] is an example to this type of 3D object detection. Using classic object detection neural network elements like convolutional layers, region proposal networks and region of interest pooling layers, Deep MANTA first performs 2D object detection while also predicting other information like parts coordinates and parts visibility. Then, it uses these predictions to find a suitable 3D template among predefined templates and perform 2D/3D matching to finalize the detection. Architecture of the Deep MANTA method can be seen in Figure 2.1.

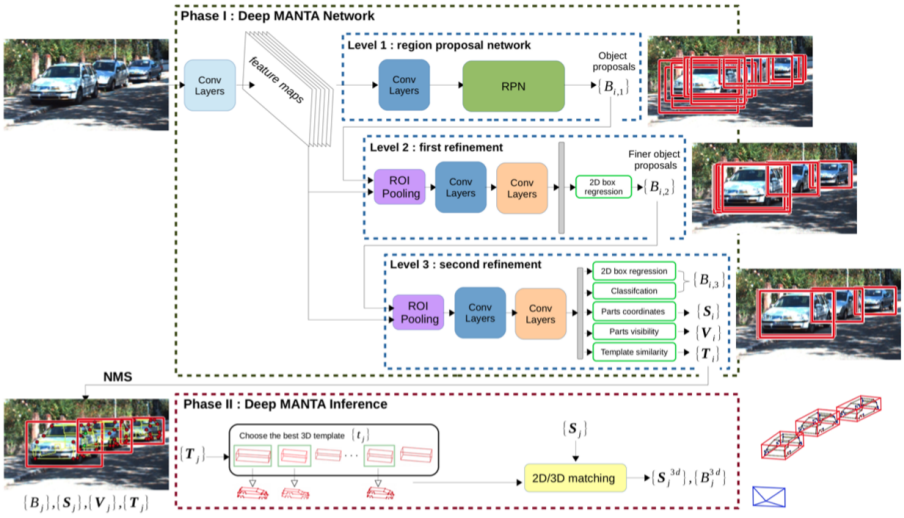


Figure 2.1: Deep MANTA architecture [15]

The second method, which is making detections from raw point clouds, is the most seen method among the papers that reach the state of the art performance. These methods generally use the architecture proposed in PointNet [16] or PointNet++ [17] as a base module and create architectures to improve performance. The architecture of PointNet is seen in Figure 2.2. PointNet architecture is mainly comprised of fully connected layers. These layers output a global feature vector from the input, which is a matrix where each row is a point in the point cloud. From this global feature vector, the whole point cloud can be classified if the task is a point cloud classification task. The network also can be used to perform segmentation by concatenating the global feature vector

to an internal feature map and predicting a class for every point in the cloud with fully connected layers. The biggest difficulty for an architecture that takes raw point clouds as input is to make the network transformation invariant. Transformation invariant means that predictions of the network should not change when an input point cloud undergoes geometric transformations. To make the network transformation invariant, two transformation layers called T-Nets are used. These networks resemble the main network in architecture and learn to predict an affine transformation matrix to align the point cloud to a canonical space. Improving on PointNet, PointNet++ uses the architecture proposed in PointNet in a hierarchical way to improve its performance. A more recent architecture, PointRCNN [18], uses PointNet++ architecture as an encoder in an encoder-decoder approach to perform better detections on the KITTI Dataset [19].

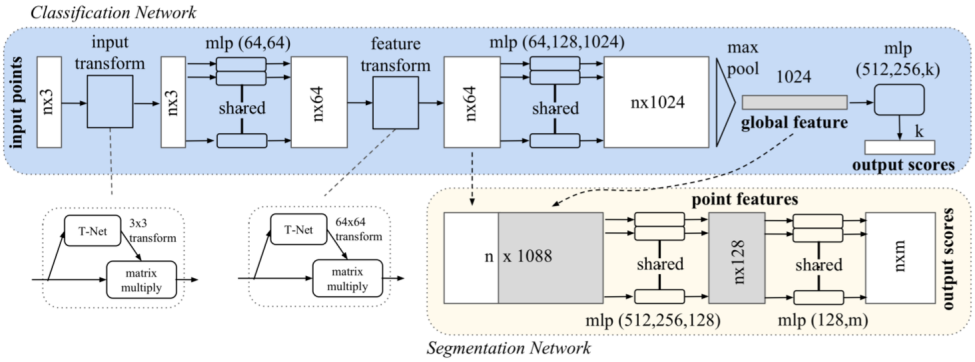


Figure 2.2: PointNet architecture [16]

Since the method proposed in this project uses 2D object detection methods with point cloud information, it can be seen as a combination of the two explained methods. The other methods mentioned by [1], which are the methods that use 3D voxel grids and the methods that project point clouds to 2D planes for 2D object detection are not really related to this project. Voxel grid methods are popularized by VoxelNet [20], while LMNet [21] is an example to the last method and makes detections by projecting the point cloud to 5 different frontal planes.

There are also methods that use both 2D object detection and point clouds together like this project. Some of the latest of these works feel similar to how this project would turn out if it was further developed. Frustum PointNets [2], for example, first detects objects from RGB images. Then, it generates a frustum point cloud of the detected object by using a depth image of the RGB image. This point cloud is given to a PointNet model for 3D bounding box inference. This architecture can be seen in Figure 2.3. Frustum ConvNet [22] takes this approach further by dividing the frustum point cloud to separate parts and giving each of them to a PointNet model. The outputs are combined as a feature vector and further processed with additional convolutional layers. These methods give state of the art results, but unlike the method proposed for this project, they require training with point clouds to obtain PointNet models with high performance.

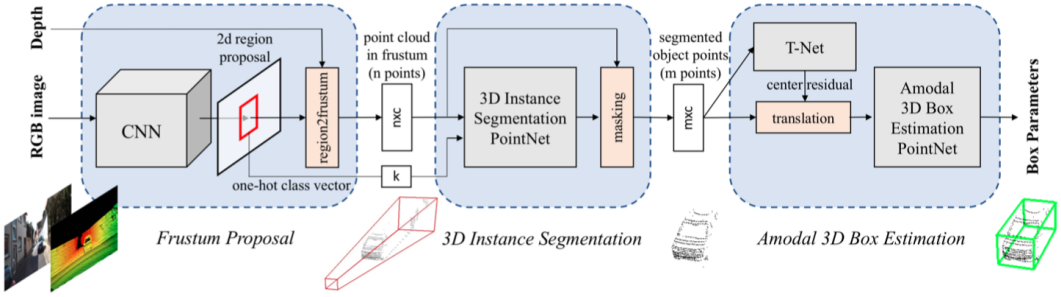


Figure 2.3: Frustum PointNets architecture [2]

2.2 2D Object Detection

Since this project detects objects from RGB images first, 2D object detection methods are used extensively. For training object detectors, Detectron2 [13] framework is used. Since Detectron2 includes Faster R-CNN and RetinaNet architectures, these are used for experiments.

Faster R-CNN [7] is a two-stage detector developed from R-CNN [23]. Before R-CNN, the naive approach for object detection was finding regions of interest from the image and using CNN's to regress and classify bounding boxes from these regions. R-CNN proposed a greedy selective search algorithm to generate 2000 region proposals and processed these regions with CNNs. Proposed after R-CNN, Fast R-CNN [24] improved the R-CNN by putting the CNN processing step before generating region proposals. This way, there is only one forward pass from the CNN using the input image instead of forwarding 2000 region proposals separately. From the output feature map, region proposals are created with selective search and these proposals are given to fully connected layers for final regression and classification. The main drawback of both these models is the selective search algorithm. Since it is a greedy algorithm, selective search is implemented on CPU and becomes a bottleneck for performance. To overcome this issue, Faster R-CNN proposed the Region Proposal Network (RPN), a network architecture that takes images of any size as input and outputs class-agnostic rectangular object proposals with objectness scores. For the proposal generation, RPN uses anchor boxes of different scales and aspect ratios with a sliding window approach. These proposals are later classified using region of interest pooling. The architectures of R-CNN, Fast R-CNN and Faster R-CNN can be seen in Figure 2.4.

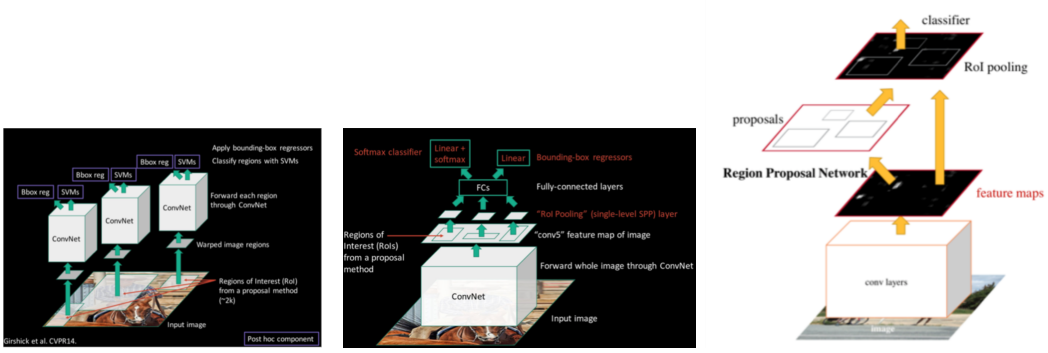


Figure 2.4: R-CNN (left), Fast R-CNN (middle), Faster R-CNN (right) architectures [25]

Unlike the previous architectures, RetinaNet [8] is a one stage detector. This means that there is not a separate module that narrows down the possible candidate region proposals. Instead, a Feature Pyramid Network (FPN) backbone is trained on top of a ResNet [26] architecture. The FPN outputs a multi-scale convolutional feature pyramid. Then, two sub-networks, one for classification and one for bounding box regression are attached to classify and regress anchor boxes. This architecture can be seen in Figure 2.5. This architecture is used with a loss function called Focal Loss [8]. The RetinaNet/Focal Loss paper states that cross-entropy loss is overwhelmed by large class imbalance in the training dataset. Focal loss is proposed to solve this issue. It is defined in a way that reshapes the cross entropy loss to down-weight easy examples and focus training on hard negatives(classes with a low amount of training data). Simplicity of the architecture, combined with focal loss results in an architecture that performs faster than Faster R-CNN.

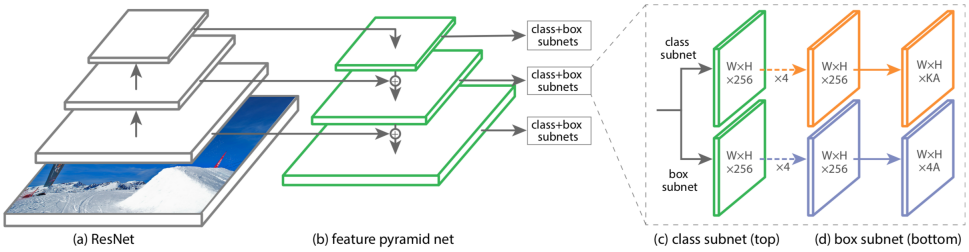


Figure 2.5: RetinaNet architecture [8]

2.3 2D-3D-S Dataset

The dataset used to run experiments on the project is Stanford University's 2D-3D-S Dataset [6]. This dataset is collected with a Matterport Camera [27] and includes data for 2D and 3D modalities. The data covers over 6000 m^2 from 6 large-scale indoor areas from three different Stanford University buildings. For each area, the dataset includes RGB images with semantic annotations, depth images and camera metadata with location and orientation information. In addition, semantically annotated point clouds that were previously available in S3DIS [11] dataset and 3D meshes are also

included. The dataset also includes data unused in the project like surface normal images, panoramic images and 3D coordinate encoded images. Example data from the dataset is shown in Figures 2.6 and 2.7.

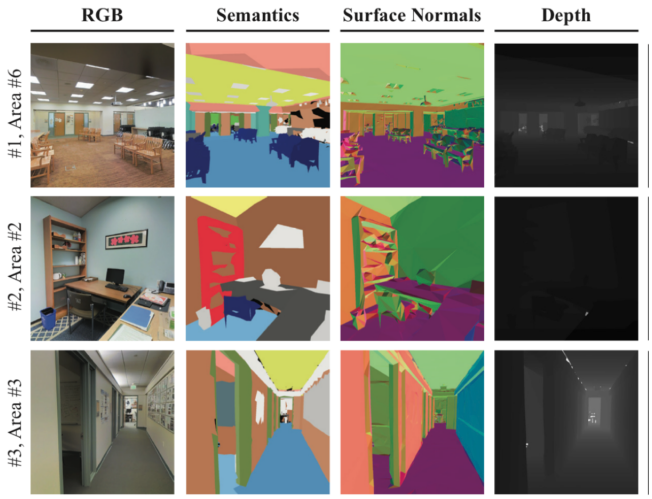


Figure 2.6: Data samples from 2D-3D-S Dataset [6]

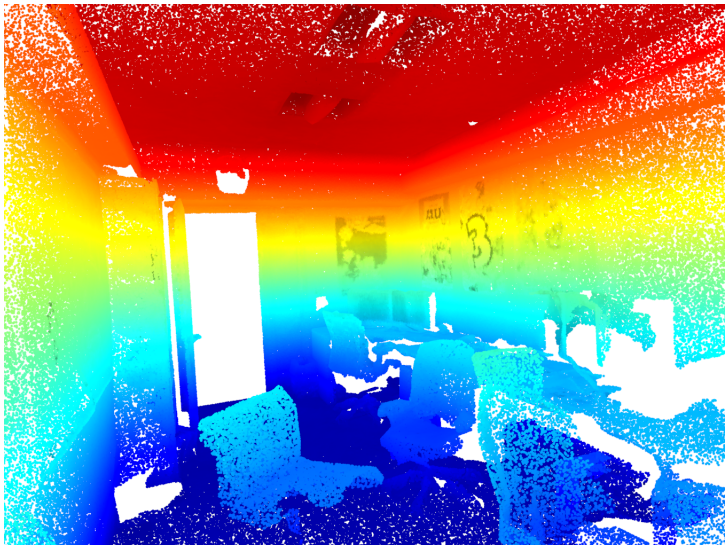


Figure 2.7: A sample point cloud of an office from S3DIS Dataset

3 Developed Approach and System Model

3.1 Data Model

Since the project is essentially about performing a machine learning task, there is a large amount of data involved in the project. Also, because the project involves both 2D and 3D data, the data formats used are very diverse. The dataset used for implementing and testing the project is the 2D-3D-S Dataset [6], as mentioned and explained in the Comparative Literature Survey section. In this section, file structure of the dataset, some JSON files which are obtained by processing the dataset and scripts that are used to obtain them will be explained.

3.1.1 2D-3D-S Dataset File Structure

Folder structure of the 2D-3D-S Dataset can be seen in Figure 3.1. From this structure, data inside "rgb", "pose", and "semantic" folders are used. The "rgb" folder contains regular RGB images, while "pose" and "semantic" folders contain camera information and instance segmentation images corresponding to each RGB image, respectively. Pose file format and example RGB and semantic images can be seen in Figures 3.2 and 3.3. In semantic images, each object instance is marked with a unique color. Each RGB color represent the index of the marked object in an object list contained in "semantic_labels.json" file. Conversions between colors and indexes can be done with functions included in "assets/utils.py" file.

```
README.md
/assets
  semantic_labels.json
  utils.py
/area_1
  /3d
    pointcloud.mat
    rgb.obj    # The raw 3d mesh with rgb textures
    rgb.mtl      # The textures for the raw 3d mesh
    semantic.obj # Semantically-tagged 3d mesh
    semantic.mtl # Tectures for semantic.obj
    /rgb_textures
      {uuid_{i}}.jpg  # Texture images for the rgb 3d mesh
  /data   # all of the generated data
    /pose
      camera_{uuid}__{room}_{i}_frame_{j}_domain__pose.json
    /rgb
      camera_{uuid}__{room}_{i}_frame_{j}_domain__rgb.png
    /depth
    /global_xyz
    /normal
    /semantic
    /semantic_pretty
  /pano    # equirectangular projections
    /pose
      camera_{uuid}__{room}_{i}_frame_equirectangular_domain__pose.json
    /rgb
    /depth
    /global_xyz
    /normal
    /semantic
    /semantic_pretty
  /raw     # Raw data from Matterport
    {uuid}_pose_{pitch_level}_{yaw_position}.txt # RT matrix for raw sensor
    {uuid}_intrinsics_{pitch_level}.txt      # Camera calibration for sensor at {pitch_level}
    {uuid}_i{pitch_level}_{yaw_position}.jpg # Raw RGB image from sensor
    {uuid}_d{pitch_level}_{yaw_position}.jpg # Raw depth image form sensor
/area_2
/area_3
/area_4
/area_5a
/area_5b
/area_6
```

Figure 3.1: 2D-3D-S Dataset's folder structure [28]

Even though point clouds are included in 2D-3D-S, they are not used because of their

```
{
  "camera_k_matrix":  # The 3x3 camera K matrix. Stored as a list-of-lists,
  "field_of_view_rads": #  The Camera's field of view, in radians,
  "camera_original_rotation": #  The camera's initial XYZ-Euler rotation in the .obj,
  "rotation_from_original_to_point":
  #  Apply this to the original rotation in order to orient the camera for the corresponding picture,
  "point_uuid": #  alias for camera_uuid,
  "camera_location": #  XYZ location of the camera,
  "frame_num": #  The frame_num in the filename,
  "camera_rt_matrix": #  The 4x3 camera RT matrix, stored as a list-of-lists,
  "final_camera_rotation": #  The camera Euler in the corresponding picture,
  "camera_uuid": #  The globally unique identifier for the camera location,
  "room": #  The room that this camera is in. Stored as roomType_roomNum_areaNum
}
```

Figure 3.2: Pose file format from 2D-3D-S Dataset [28]

Figure 3.3: An RGB image(left) and its segmentation image(right) from 2D-3D-S Dataset

file format, which is ".mat". Instead, point clouds are taken from S3DIS dataset [11], which is an earlier dataset from Stanford University that include the same point clouds in ".txt" format. In the folder structure of S3DIS dataset, each area has its own folder. In area folders, there is a folder for each room. Finally, in each room folder there is a point cloud text file that includes each point of the room as lines comprised of x, y, z, r, g, b coordinate and color values separated with spaces. As annotations, each object has a separate point cloud text file inside room folders. Using these separate object point clouds, the center point of each object is calculated and saved for evaluating 3D center point predictions by taking the average of each object's points. It was also necessary to rename the objects in S3DIS dataset because there were inconsistencies between the naming of 2D-3D-S and S3DIS datasets.

As stated in the previous paragraph, S3DIS includes separate point clouds for each room. But, for mapping 2D detections to 3D, full area point clouds are required. Therefore, for each area, room point clouds are combined and converted to PLY file format for mapping 2D detections to 3D with Point Cloud Library. Combined point cloud of Area 3 can be seen from outside and inside in Figure 3.4.

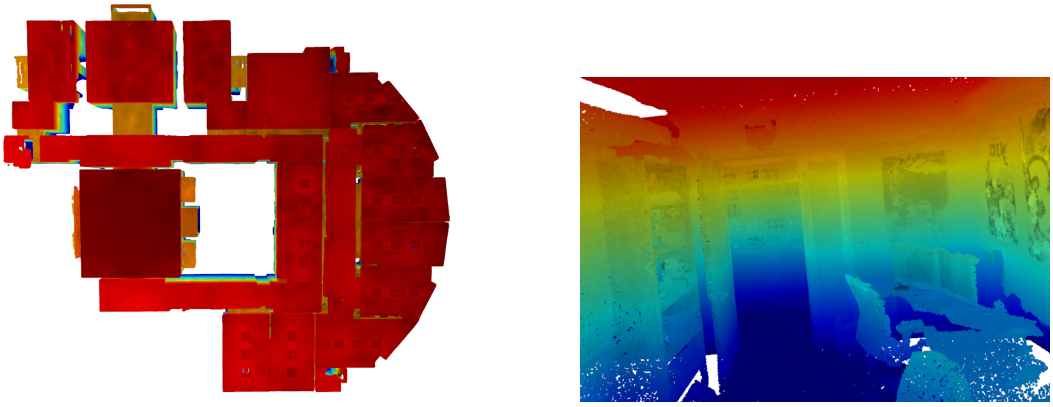


Figure 3.4: Views of the combined Area 3 point cloud from outside and inside

3.1.2 JSON Files Created From The Dataset

The first JSON file created from the data contains bounding box annotations of RGB images for the five movable object classes of 2D-3D-S dataset which are: table, bookcase, board, chair and sofa. Bounding box annotations are created from the semantic segmentation images. With these bounding box annotations, dictionaries are created for each area and saved to JSON files. It must also be mentioned that only semantic annotations larger than 250 pixels are turned into bounding boxes to only include objects that are clearly visible and to avoid including small faulty annotations, examples of which exist in the dataset. Figure 3.5 shows the format of the dictionaries contained in these JSON files. Algorithm 3.1 shows the pseudocode for creating this dictionary.

```
{
    ...,
    image name : {
        "file_name": image name
        "height": 1080 #image height
        "width": 1080 #image width
        "objects": {
            ...,
            object number: {
                "bbox": [xmin, ymin, xmax, ymax] #Two points that describe the bounding box
                "category_id": class number #Object's class' number
            },
            ...
        }
    },
    ...
}
```

Figure 3.5: Format of the json file that holds bounding box annotations

The second type of json file created is for evaluating 3D center point predictions. For each object, its index in the "semantic_labels.json" and statistics about x, y, z, r, g, b values of the points in the object's point cloud are included. These statistics include minimum, maximum and mean for each of the values mentioned. Min and max values are used to calculate the largest internal distance for each object, which is in turn used to determine if a center point prediction falls inside the object's area. This JSON file is created by saving every object in the S3DIS to a Pandas dataframe and using the *describe()* method on them which calculates the values included in the file. Then, each object's calculated values are gathered in a dictionary and dumped to a JSON file. Format of this JSON file is shown in Figure 3.6.

```
dataset_dict ← {};
anns ← Load "semantic_annots.json";

for image_path in an Area's RGB image folder do
    initialize dictionary img_dict;
    img_dict['file_name'] ← get image file name from image_path;
    sem_file_path ← get semantic image path using image_path;
    sem_img ← open sem_file_path as a NumPy array;
    img_dict['height'] ← sem_img.shape[0];
    img_dict['width'] ← sem_img.shape[1];

    obj_colors ← unique RGB values of sem_img;
    img_dict['objects'] ← {};
    obj_key ← 0;
    for color in obj_colors do
        index ← utils.get_index(color);
        if index > len(anns) then
            continue;
        end
        lbl ← parse anns[index];
        if label is in a valid class then
            mask ← pixels that are in color;
            if mask contains less then 250 pixels then
                continue;
            end
            img_dict['objects'][obj_key] ← {};
            img_dict['objects'][obj_key]['bbox'] ← [min x, min y, max x, max y values of mask pixels];
            img_dict['objects'][obj_key]['category_id'] ← class of object;
            obj_key ← obj_key + 1;

        end
    end
    if img_dict contains 0 objects then
        continue;
    end
    dataset_dict[image file name] ← img_dict;
    dump dataset_dict to output json file;
end
```

Algorithm 3.1: Pseudocode of the algorithm that creates the annotation JSON files shown in Figure 3.5.

```
{
    ...,
    object_name : { #Naming according to semantic_labels.json
        "index": index of object in semantic_labels.json
        "info_df": {
            "x": {
            ...
            "mean": mean of x coordinates of all points of the object
            "min": min value of x coordinates of all points of the object
            "max": max value of x coordinates of all points of the object
            ...
            },
            "y": {...},
            "z": {...},
            "r": {...},
            "g": {...},
            "b": {...},
        }
    },
    ...
}
```

Figure 3.6: Format of the JSON file that holds 3D object information

3.2 Structural Model

Structure of the pipeline can be divided into two main sections: Making object detections from RGB images and mapping the center points of the acquired detections to 3D coordinates. In this section, implementations of these parts will be explained. Also, directions on how to use this pipeline with a Realsense D435 RGB-D camera will be given.

3.2.1 2D Object Detection

In this first part, object detector neural networks are used to detect chair, board, sofa, bookcase and table object instances from RGB images. For this, neural network models have to be trained to perform this specific task, which is done with Detectron2 [13]. Detectron2 is Facebook AI Research's latest object detection framework, powered by PyTorch. It includes implementations for state of the art object detection and segmentation network architectures. It is used in this project to train a Faster R-CNN or RetinaNet model for each area in the 2D-3D-S dataset. Implementations for training are done according to the 2D-3D-S Dataset's structure. However, it is possible to use other datasets with the proposed pipeline with some tweaks to the codes.

For training a neural network model with Detectron2, it is necessary to register the dataset to Detectron2 in a format similar to COCO dataset [29]. This format is defined in the documentation of Detectron2 [30]. For this registration, Detectron2's *DatasetCatalog.register()* function is used, which requires a function that returns a dataset dictionary in the defined format. To convert the dictionaries inside the prepared area JSON files, which were defined in Section 3.1.2 and Figure 3.5, a function called *get_stanford_dicts* is defined. A pseudocode of this function can be seen in Algorithm 3.2.

With the *get_stanford_dicts* function, training can be started by setting some Detectron2 variables and calling some Detectron2 functions. The code for starting a Detectron2 training can be seen in Figure 3.7. The code is pretty self explanatory and some extra information is also included as comments. The only variable that is worth describing is *SOLVER.WARMUP_ITERS*. Warm-up is a technique that starts training with a very small learning rate, and linearly increases it each iteration until it reaches the base learning rate defined with the *SOLVER.BASE_LR* variable. This technique prevents early over-fitting. In Figure 3.7, warm-up is not used by setting the *SOLVER.WARMUP_ITERS* variable to zero.

Inputs: image directory path, area json file path
imgs_anns ← load area json file;
dataset_dicts ← [];
for *key, value in imgs_anns dictionary items* **do**
    record ← {};
    filename ← image directory and value[”file_name”] joined;
    record[”file_name”] ← filename;
    record[”height”] ← value[”height”];
    record[”width”] ← record[”width”];
    annos ← value[”objects”];
    objs ← [];

    for *anno_key, anno_value in annos dictionary items* **do**
        obj ← {
        ”bbox”: anno_value[”bbox”],
        ”bbox_mode”: BoxMode.XYXY_ABS, #Mode that tells detectron2 the format of bounding box
        ”category_id”: anno_value[’category_id’], Class no
        ”iscrowd”: 0
        };
        objs.append(obj);
        record[”annotations”] ← objs;
        dataset_dicts.append(record);

    end
end
return dataset_dicts;

Algorithm 3.2: Pseudocode of *get_stanford_dicts* function

```
DatasetCatalog.register("2d_3d_semantics_area4_train", lambda l =
↪  '/mnt/data/2D-3D-Semantics/area_4/data/rgb/': get_stanford_dicts(l,
↪  '/home/ubuntu/stanford-detectron/area_json_files/area_4_partitioned/area_4_train.json'))
MetadataCatalog.get("2d_3d_semantics_area4_train").set(thing_classes=['table', 'chair',
↪  'sofa', 'bookcase', 'board'])
stanford_metadata = MetadataCatalog.get("2d_3d_semantics_area4_train")

cfg = get_cfg()
cfg.merge_from_file("/home/ubuntu/detectron2_repo/configs/COCO-Detection/
↪  retinanet_R_101_FPN_3x.yaml")
cfg.DATASETS.TRAIN = (["2d_3d_semantics_area4_train"])
cfg.DATASETS.TEST = ()   # no metrics implemented for this dataset
cfg.DATALOADER.NUM_WORKERS = 2

#Loading model weights. The first line loads a previous model saved while training. The
↪  second
#commented out line loads weights pretrained on ImageNet available from Detectron2's
↪  repository
cfg.MODEL.WEIGHTS =
↪  "/home/ubuntu/stanford-detectron/out_retinanet_R_101_FPN_3x_4train_lr001/
↪  model_0099999_weights.pth"
#cfg.MODEL.WEIGHTS =
↪  "detectron2://COCO-Detection/retinanet_R_101_FPN_3x/138363263/model_final_59f53c.pkl"

cfg.SOLVER.IMS_PER_BATCH = 8
cfg.SOLVER.BASE_LR = 0.0001
cfg.SOLVER.WARMUP_ITERS = 0
cfg.SOLVER.MAX_ITER = 1000000000    # Number of iterations to train
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 5  # Classes: chair, sofa, table, board, bookcase
cfg.OUTPUT_DIR = "./out_retinanet_R_101_FPN_3x_4train_lr0001"

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
print(trainer.train())
```

Figure 3.7: The main detectron2 code for training object detectors

The next step after training an object detector to obtain a model with good performance is performing detections on test RGB images and saving the detections for 2D object detection evaluation and 3D mapping. To perform inference on test images with a trained network, *get_stanford_dicts* is used first for the same reasons as the training code. Then, also like the training code, some Detectron2 variables are set. These variables can be seen in Figure 3.8. Batch size for this operation is set to 1 because each image is processed and its detections are saved to files individually.

```
cfg = get_cfg()
cfg.merge_from_file("/home/ubuntu/detectron2_repo/configs/COCO-Detection/
↪  retinanet_R_101_FPN_3x.yaml")
cfg.DATALOADER.NUM_WORKERS = 2
cfg.SOLVER.IMS_PER_BATCH = 1
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 5  # only has one class (ballon)
cfg.MODEL.WEIGHTS = args.model_path

cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7   # set the testing threshold for this model
```

Figure 3.8: Detectron2 variables set for performing object detection on test images.

After the variables are set, each dataset dictionary that includes an image's path and object ground truths obtained from *get_stanford_dicts* is traversed to perform object detection and write the detections with their classes and bounding box coordinates to a file exclusive to that image. Ground truths are also saved to files for evaluating the performance of the trained object detection model. Pseudocode of the code that performs these operations is given in Algorithm 3.3. These detection files are later used for mapping the detections to 3D coordinates.

Inputs: image directory path, area json file path
predictor ← DefaultPredictor(cfg created in Figure 3.7);
dataset_dicts ← get_stanford_dicts(image directory, area json);
for *d in dataset_dicts* **do**
    det_file_path ← Create a detection text file path from d["file_name"];
    gt_file_path ← Create a ground truth text file path from d["file_name"];

    for *ann in d["annotations"]* **do**
        line ← Create a string that includes class and bounding box of the ann annotation;
        Write the line to gt_file_path;
    end
    im ← Load image from d["file_name"] with OpenCV;
    output ← predictor(im);
    number of detections, detection scores, bounding boxes, classes of detections ← obtain from the output;
    Write each detection to det_file_path as lines;
end

Algorithm 3.3: Pseudocode of the code that performs object detection on RGB images and saves the results to files.

3.2.2 Mapping Detections to 3D coordinates

Mapping detections from RGB images to 3D coordinates is done with Point Cloud Library [3], which implements many point cloud processing operations. The full PCL library is only available in C++, so a module that maps the center pixels of detections from an image to 3D coordinates and prints the results is implemented in C++. This module is used in a Python code that reads detections from previously created detection files and camera information from pose files. With the required values obtained, the Python code runs the C++ module for each image using Python's subprocess library, and reads the resulting 3D coordinates from stdout. From these results, along with 2D bounding boxes and detection confidences, a CSV file is created. A sample from a dataframe created from this CSV file is given in Figure 3.9.

	2d_Detection_File	Class	Confidence	x_min	y_min	x_max	y_max	x	y	z
0	camera_4d491624b8dd4db9999935affb0c4ada_hallwa...	bookcase	0.999934	359.872162	623.580627	536.943970	960.730408	22.45890	1.024760	0.764248
1	camera_4d491624b8dd4db9999935affb0c4ada_hallwa...	bookcase	0.999806	366.362671	283.822784	537.079041	569.667786	22.57530	1.231510	2.054130
2	camera_4d491624b8dd4db9999935affb0c4ada_hallwa...	chair	0.999786	454.236725	849.288635	538.963318	1076.020874	22.24710	0.283692	0.380640
3	camera_b4ea4a673c654438b4e0218c74467c49_lounge...	chair	0.999911	500.347137	811.448792	674.723816	932.269043	7.31175	-7.508770	0.680815
4	camera_b4ea4a673c654438b4e0218c74467c49_lounge...	table	0.999901	23.123035	887.927979	745.532166	1079.347900	8.21978	-6.851880	0.736935
...	...	...	...	...	...	...	...	...	...	...

Figure 3.9: Result CSV file. x_min, y_min, x_max, and y_max define the object's bounding box in the RGB image, while x, y and z are the mapped 3D coordinates of the center of the bounding box.

The C++ module maps the detected bounding boxes' center points to 3D by creating a planar range image that corresponds to the RGB image. Planar range image is a data structure implemented in PCL and created by placing a virtual camera inside of a point cloud. The planar range image created stores the image seen from the virtual camera along with each visible point's 3D coordinates in the point cloud. In Figure 3.10, an RGB image from 2D-3D-S dataset along with a visual representation of the range image created with the camera location, orientation and metadata from the RGB image's pose file is shown. Once a range image that corresponds to an RGB image is created, it becomes possible to find 3D coordinates of pixel locations in the RGB image. This is how center pixels of bounding boxes are mapped to 3D. To create a range image, the module requires 3D coordinates, orientation(roll, pitch, yaw) and focal length of the virtual camera, range image width and range image height. These inputs are provided by the Python code that runs the module. Pseudocode for the C++ module is given in Algorithm 3.4.

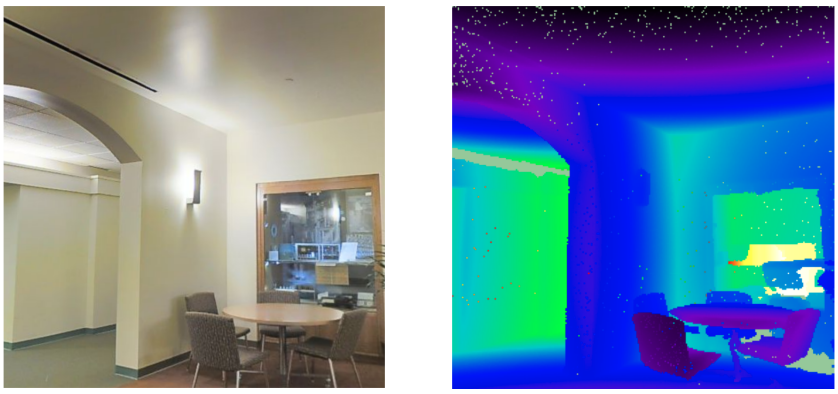


Figure 3.10: A sample RGB image(left) and a range image created from the information in the RGB image's pose file(right)

Inputs: point cloud, camera location, camera orientation, camera focal length, range image width-height, pixel locations for conversion

pointCloud ← loadPLYFile(point cloud);
sensorpose ← getTransformation(camera location, camera orientation);
coordinate_frame ← pcl::RangeImagePlanar::LASER_FRAME Reference frame used for the sensor pose;

Initialize pcl rangeImage;
rangeImage.createFromPointCloudWithFixedSize(pointCloud, sensorPose, range image width-height, camera focal length, coordinate_frame);
rangeImage_half_res ← rangeImage.getHalfImage();
rangeImage_quarter_res ← rangeImage_half_res.getHalfImage();

for *2D point in pixel locations for conversion* **do**
    xyz_coordinates ← rangeImage.getPoint(2D point);
    print xyz_coordinates;
end

Algorithm 3.4: Pseudocode of the C++ 3D mapping module. As it can be seen, resolution of the range image is reduced to the quarter of the original resolution of the RGB image. The reason for this is to eliminate pixels in the RGB image that do not represent a point in the point cloud. These pixels exist because of the space between point cloud points.

3.2.3 Usage With An Intel Realsense D435 RGB-D Camera

To use this approach with an Intel Realsense D435 RGB-D camera, it is necessary to be able to use the camera to scan an area for a point cloud while also taking RGB pictures and tracking the location and orientation of the camera. For this, ROS (Robot Operating System) [4] is used, which is a meta-operating system used in robotics applications for connecting software and sensors. An Intel Realsense D435 camera can be used with ROS by installing Intel's Realsense drivers and Intel Realsense ROS Wrapper [31]. The final requirement is RTABMap [5], which is a ROS software that performs simultaneous localization and mapping (SLAM) with stereo and RGB-D cameras. This software can communicate with the camera through ROS to perform scanning and obtain the necessary data. The acquired point cloud, camera information and RGB images can then be used with the proposed system.

4 Experimentation Environment and Experiment Design

The experimentation in this project aims to measure the success of the proposed 3D object detection approach. Since 2D object detection is performed first and then results are mapped to 3D in the proposed pipeline, performance of the 2D detection and accuracy of the 3D center points which are the outputs of 3D mapping is evaluated separately. For evaluating 2D object detection, PASCAL VOC [9] metric is used. For evaluating 3D mappings of objects' center points, a custom evaluation that depends on the distance between predicted 3D center points and ground truth 3D center points is implemented.

4.1 2D Object Detection Test With PASCAL VOC Metric

In object detection, precision is defined as $\frac{true\ positives}{true\ positives+false\ positives}$ and measures the success rate of the bounding box predictions for the objects. Recall, on the other hand, is defined as $\frac{true\ positives}{true\ positives+false\ negatives}$ and measures how well the predictions cover the ground truth. Whether a prediction is considered true positive or false positive is determined by the intersection over union(IoU) value. Visualization of intersection over union calculation is given in Figure 4.1. The default IoU threshold value for considering a prediction positive is 0.5.

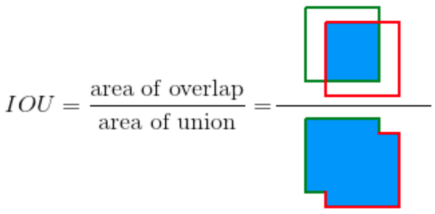


Figure 4.1: Visual representation of intersection over union value [32]

PASCAL VOC challenge uses precision-recall curves and average precision for evaluating object detectors. Precision-recall curve is obtained by plotting precision in varying recall values. A good object detector's precision stays high as its recall increases, which means precision and recall is affected little by changes in the confidence threshold. PASCAL VOC represents the precision-recall curve numerically by calculating Average Precision (AP) for each object class, which means averaging precision values for recall values between 0 and 1 [32].

To evaluate the object detectors trained for the project with the PASCAL VOC metric, R. Padilla's implementation on GitHub [32] is used. This Object Detection Metrics repository includes a Python code that makes the evaluation. The code requires ground truth and detection folders, with each folder including a text file for each image. Ground truth files include two points for the ground truth bounding box, and object class for

each object seen in the image as lines. Detection files, on the other hand, includes two points for the predicted bounding box, the predicted class and prediction confidence for each object prediction as lines. The code for performing 2D object detection in this project, previously defined in Algorithm 3.3, creates these files so that R. Padilla's implementation can be used.

4.2 Testing 3D Center Points

3D mapping was done by the C++ code described in Algorithm 3.4, resulting in a CSV file shown in Figure 3.9. An evaluation code is implemented to evaluate the correctness of these center point predictions, and the main metric used here is the distance between predicted center points and the predicted object's true center point. If this distance is smaller than half of the object's largest internal distance, the prediction is assumed to be correct. The internal distance information of objects can be obtained from the previously created JSON file described in Figure 3.6. Using this distance metric, true positives, false positives and false negatives are counted. There are two types of false positives: The first type is mapping the center pixel of a bounding box to a different type of object in the point cloud. The second type is when mapping is done to the correct object type, but the distance between the prediction and ground truth point is above the previously defined threshold. The reasons for these types of false positives will be discussed further in the Comparative Evaluation and Discussion section. The steps of this evaluation is described below.

- For each test set, a json file is created that includes a boolean list which shows whether an object in the "semantic_labels.json" file is found in the test set. The first step is loading the json file for the test set used and acquiring indexes of objects from "semantic_labels.json" that are available and not available in the test set.
- A list that is the same length as the lists in "semantic_labels.json" and the index JSON file is created and filled with the value "9999". This list will hold the calculated distances for each detected object.
- The detection CSV file described in Figure 3.9 is loaded to a Pandas dataframe for each center point prediction to be evaluated.
- For each center point prediction, the semantically annotated image of the RGB image that the center point is mapped from is loaded.
- From the semantically annotated image, five pixels from inside the 2D bounding box is selected: center pixel and the middle pixel between the center pixel and each corner. Colors of these pixels are converted to "semantic_labels.json" indexes. The index that is obtained the most from these five pixels is accepted to be the object that the bounding box encloses.
- Since the object is found, the distance between the predicted 3D center point and the object's true center point is calculated and saved to the list created in the second step. It is possible that there are multiple detections for the same object.

In this case, the smallest distance to the ground truth center point is saved to the list. Normally, there needs to be an algorithm that eliminates multiple detections for an object, but that could not be finished and classified as future work.

- As a result of the previous steps the list that has the same length as the list in "semantic_labels.json" is changed. The elements in indexes that are not detected have "9999" as value. For the detected objects, their indexes in the list store the distance between prediction and ground truth points. Also, indexes of objects that exist in the test data were acquired in the first step.

- The resulting list is traversed. If an object is in the test data and the distance value is 9999, it means that this object could not be detected and this is classified as a false negative. Else if the object is in the test data and the distance value is smaller than half of the largest internal distance of the object's individual point cloud, it is classified as a true positive. If the object is in test data but it does not fit neither of the previous two descriptions, it means that an object in the correct class is detected but the distance is larger than it should be. This is classified as a false positive and it most likely means that while the detected class is correct, the instance is not. Finally, if the object is not in the test data but a detection is made anyway, it is classified as a false positive.

5 Comparative Evaluation and Discussion

Using the metrics described in the previous section, experiments are done on the first four areas of the Stanford 2D-3D-S Dataset. It should be noted that these metrics are not suitable for comparisons with the goals set in the interim report. In the interim report, the goals were set as accuracies. However, after research on the topic, the evaluation metrics presented here are decided to be more informative and suitable to the task.

Even though each area has the same types of objects, styles of them vary significantly from area to area. Because of this reason, attempts at training a more general 2D object detection model by mixing images from different areas has resulted poorly. Therefore, separate object detectors are trained for each area with images from that area, which makes sure that 2D object detection has good performance. With well performing 2D object detectors, results of 3D mapping can be evaluated more clearly, since its performance depends on 2D object detection. For Area 3, a Faster R-CNN R101-FPN model is trained. After some research, it turns out that RetinaNet architecture models with similar success can be trained in a shorter time. Therefore, for areas 1, 2 and 4, RetinaNet R101-FPN models are trained.

For each area, the images of the area is split to training and test sets. A 2D object detector is trained on the training set. With the acquired model, object detection is ran on the test set images and center pixels of the bounding boxes are mapped to 3D coordinates. These 3D center predictions are then evaluated. In Table 5.1, train and test splits of images of the first four area is shown. In Table 5.2, room information about these areas is given. In tables from 5.3 to 5.10, test results and evaluations of 2D object detection networks and 3D object center points are presented.

Table 5.1: Area Train-Test Image Splits

Area No	Train	Test
1	8000	748
2	12442	1000
3	2433	312
4	9111	1000

Table 5.2: Rooms of the Areas

Area No	Rooms
1	1 WC, 2 conference rooms, 1 copy room, 8 hallways, 31 offices, 1 pantry
2	2 WCs, 2 auditoriums, 1 conference room, 12 hallways, 14 offices, 9 storage rooms
3	2 WCs, 1 conference room, 6 hallways, 2 lounges, 10 offices, 2 storages
4	4 WCs, 3 conference rooms, 14 hallways, 2 lobbies, 22 offices, 4 storage rooms

Table 5.3: Area-1 2D Object Detection Results According to PASCAL VOC

Class	Average Precision (AP)
board	96.67%
bookcase	96.39%
chair	95.35%
sofa	97.19%
table	94.97%
Mean Average Precision (mAP): 96.12%	

Table 5.4: Evaluation of Area-1 3D Center Point Predictions

Object Count: 340			
TP	FN	FP (False Object)	FP (Large Distance)
board: 26 bookcase: 79 chair: 117 sofa: 6 table: 58	bookcase: 5 chair: 25 table: 8	beam: 5 ceiling: 1 clutter: 142 column: 29 door: 40 floor: 26 wall: 106 window: 6	bookcase: 2 chair: 11 table: 3
Total: 286	Total: 38	Total: 355	Total: 16

Table 5.5: Area-2 2D Object Detection Results According to PASCAL VOC

Class	Average Precision (AP)
board	94.98%
bookcase	93.97%
chair	82.49%
sofa	94.10%
table	95.03%
Mean Average Precision (mAP): 92.11%	

Table 5.6: Evaluation of Area-2 3D Center Point Predictions

Object Count: 658			
TP	FN	FP (False Object)	FP (Large Distance)
board: 17 bookcase: 46 chair: 127 sofa: 6 table: 44	bookcase: 1 chair: 365 table: 1	beam: 2 ceiling: 6 clutter: 82 column: 12 door: 54 floor: 28 wall: 113 window: 3	chair: 51
Total: 240	Total: 367	Total: 300	Total: 51

Table 5.7: Area-3 2D Object Detection Results According to PASCAL VOC

Class	Average Precision (AP)
board	95.28%
bookcase	93.65%
chair	92.69%
sofa	95.94%
table	92.07%
Mean Average Precision (mAP): 93.93%	

Table 5.8: Evaluation of Area-3 3D Center Point Predictions

Object Count:			
TP	FN	FP (False Object)	FP (Large Distance)
board: 12 bookcase: 39 chair: 54 sofa: 9 table: 28	chair: 12 table: 2	clutter: 22 column: 1 door: 6 floor: 10 wall: 19	bookcase: 1 chair: 1
Total: 142	Total: 14	Total: 58	Total: 2

Table 5.9: Area-4 2D Object Detection Results According to PASCAL VOC

Class	Average Precision (AP)
board	95.94%
bookcase	96.20%
chair	93.78%
sofa	96.45%
table	95.70%
Mean Average Precision (mAP): 95.61%	

Table 5.10: Evaluation of Area-4 3D Center Point Predictions

Object Count:			
TP	FN	FP (False Object)	FP (Large Distance)
board: 10 bookcase: 97 chair: 101 sofa: 14 table: 76	bookcase: 1 chair: 33 table: 1	ceiling: 1 clutter: 71 column: 11 door: 44 floor: 23 wall: 90 window: 4	chair: 25
Total: 298	Total: 35	Total: 244	Total: 25

Generally, 2D object detectors work well with over 90% average precisions for classes. It is noticeable that average precision of the chair class in each area is on the lower side.

The reason for this is that the visible shape of a chair differs greatly between different angles, which makes it harder for object detection networks to learn a chair's shape. Figure 5.1 shows three different photos from Area 1 of the dataset, which illustrates this difference between visible shapes.

Figure 5.1: Three sample images from Area 1 of 2D-3D-S Dataset that includes chairs with different shapes

Even if the aforementioned issue is considered, Area 2 is still an anomaly with an 82.49% AP for chairs. This is because in Area 2, there are two auditoriums, which are large conference halls with lots of seats in rows. As a result, it becomes really difficult for a network to learn distinguishing different instances. Also, the annotations of these seats in auditoriums are pretty inconsistent with noticeable errors, which also contribute to the low average precision. This also affects the results of 3D mapping greatly, as the amount of false positive and false negative chair instances is very large in Area 2 compared to other areas. Some example images from Area 2's auditoriums can be seen in Figure 5.2.

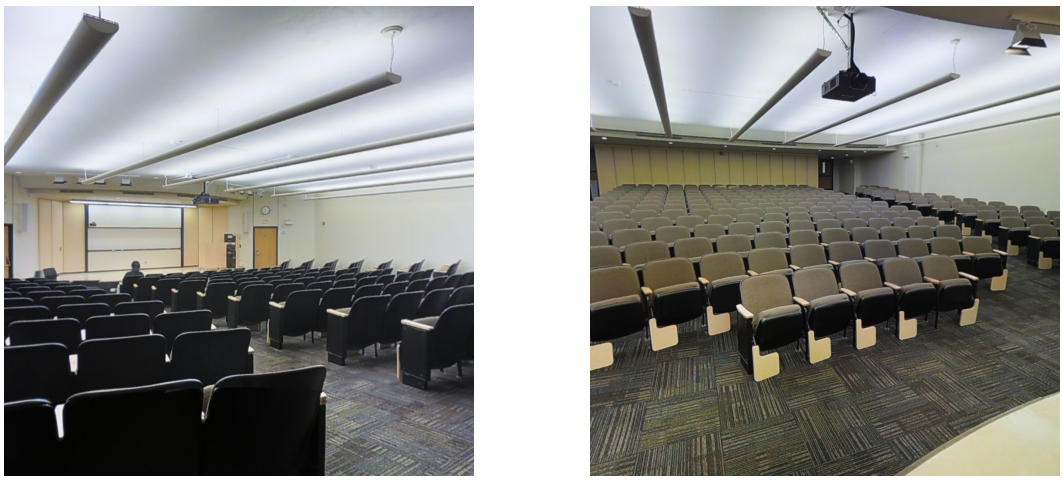


Figure 5.2: Images from auditoriums of Area 2

When it comes to 3D results, false negatives and false positives will be discussed. False negatives are when an object could not be detected. It is noticeable that almost all the false negatives across four areas are chair instances. The reason for this is once again the shape of a chair. From certain angles, a chair may not cover most of its ground truth

bounding box, which may result in the bounding box's center pixel to not fall on the object itself, mapping it to another object or wall behind. It was mentioned previously that during evaluation, five pixels are selected inside the bounding box and the ground truth object is assumed to be the object that is obtained the most from these pixels using the semantically annotated image. However, if most of these pixels are mapped to a different object like a wall behind, which is very possible for a chair that is seen from the side, it results in a false positive, since the detection and evaluation is done for a different object. This results in a large false positive count. This is why FP (False object) section is filled with falsely detected objects like walls, doors, and windows that are not included in classes to be detected. The amount of false object FPs are much larger than false negatives because each object is seen from multiple images that are taken from different angles, which significantly increases the probability of detecting objects correctly at least once. However, each failure adds to the false positives. Also there is another case where the five pixels successfully determine the ground truth object but the center pixel do not fall on that object. This is the case of FP (Large Distance), which actually means that a different object instance near the true object is detected. This category is also filled with chair instances.

From the discussion above, it is clear that mapping centers of bounding boxes to 3D and evaluating them with the 5 pixel approach do not work well with objects that have complex shapes, holes and blank areas like a chair. The other object classes have simpler shapes without blank areas and 3D center point predictions done for their instances are pretty successful.

6 Conclusion and Future Work

In this project, a pipeline is proposed that performs object detection on RGB images and maps the center pixels of bounding boxes to 3D using point clouds and camera information to find the locations of objects in 3D space. This pipeline is tested with 2D-3D-S Dataset. From these tests, it became clear that while the approach works for objects with simple shapes, for small and complex shaped objects like chairs, only mapping the center pixel of the bounding box for finding an object's 3D location and evaluating it with the proposed approach that uses five pixels inside the bounding box performs poorly.

As future work, the first thing that should be done is to develop better 3D mapping and evaluation algorithms that uses multiple points. This way, performance on objects with complex shapes can be improved. A good idea is 3D mapping several pixels from the bounding box and using clustering to differentiate object points and background points. Furthermore, with multiple points belonging to the object successfully found, it is possible to create a 3D bounding box for the object. 3D bounding box detection gives way more information than center point detection. It would also be possible to evaluate the approach with state of the art 3D object detection benchmarks with a 3D bounding box detection.

Another thing that the project lacks is the elimination of multiple 3D detections of the same object. As it is, it is not possible to scan an area with an RGB-D camera and have each object of interest detected clearly, since there will be multiple detections for each object. Therefore, a way to reduce detections for each object to one should be found as future work. Clustering algorithms or deep learning approaches could be possible solutions to this problem.

Another idea is to perform segmentation on RGB images in the deep learning part of the pipeline instead of object detection. After a successful instance segmentation, every pixel can be mapped to 3D to achieve 3D instance segmentation.

Finally, after the present problems of the approach is solved, a software that works with an RGB-D camera like Intel Realsense D435 and implements all the operations for performing 3D detection on any area can be developed.

References

[1] S. Agarwal, J. O. D. Terrail, and F. Jurie, "Recent advances in object detection in the age of deep convolutional neural networks," *CoRR*, vol. abs/1809.03193, 2018. [Online]. Available: http://arxiv.org/abs/1809.03193

[2] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[3] "Pcl - point cloud library," http://pointclouds.org, accessed: 2020-01-15.

[4] "Ros - robot operating system," https://www.ros.org, accessed: 2020-01-15.

[5] "Rtab-map. real-time appearance-based mapping," http://introlab.github.io/rtabmap/, accessed: 2020-01-15.

[6] I. Armeni, S. Sax, A. R. Zamir, and S. Savarese, "Joint 2d-3d-semantic data for indoor scene understanding," *CoRR*, vol. abs/1702.01105, 2017. [Online]. Available: http://arxiv.org/abs/1702.01105

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 91–99. [Online]. Available: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

[8] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[9] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[10] L. Liu, W. Ouyang, X. Wang, P. W. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *CoRR*, vol. abs/1809.02165, 2018. [Online]. Available: http://arxiv.org/abs/1809.02165

[11] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "3d semantic parsing of large-scale indoor spaces," in *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[12] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski, "Building rome in a day," *Commun. ACM*, vol. 54, no. 10, pp. 105–112, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/2001269.2001293

[13] "Detectron2," https://github.com/facebookresearch/detectron2, accessed: 2020-01-15.

[14] "Pytorch," https://pytorch.org, accessed: 2020-01-15.

[15] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teuliere, and T. Chateau, "Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[16] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[17] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5099–5108. [Online]. Available: http://papers.nips.cc/paper/7095-pointnet-deep-hierarchical-feature-learning-on-point-sets-in-a-metric-space.pdf

[18] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 770–779.

[19] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[20] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[21] K. Minemura, H. Liau, A. Monrroy, and S. Kato, "Lmnet: Real-time multiclass object detection on cpu using 3d lidar," in *2018 3rd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, July 2018, pp. 28–34.

[22] Z. Wang and K. Jia, "Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection," *CoRR*, vol. abs/1903.01864, 2019. [Online]. Available: http://arxiv.org/abs/1903.01864

[23] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[24] R. Girshick, "Fast r-cnn," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[25] F.-F. Li, A. Karpathy, and J. Johnson, "Stanford university cs231n: Convolutional neural networks for visual recognition, lecture 8: Spatial localization and detection lecture slide," http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf, accessed: 2020-01-16.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[27] "Matterport: 3d models of interior spaces," https://matterport.com, accessed: 2020-01-15.

[28] S. Sax, "The data skeleton from joint 2d-3d-semantic data for indoor scene understanding," https://github.com/alexsax/2D-3D-Semantics, accessed: 2020-01-15.

[29] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, 2014, pp. 740–755. [Online]. Available: https://doi.org/10.1007/978-3-319-10602-1_48

[30] "Detectron2's documentation," https://detectron2.readthedocs.io, accessed: 2020-01-15.

[31] "Intel realsense ros wrapper," https://github.com/IntelRealSense/realsense-ros, accessed: 2020-01-15.

[32] R. Padilla, "Object detection metrics," https://github.com/rafaelpadilla/Object-Detection-Metrics, accessed: 2020-01-15.