INDOOR VISUAL UNDERSTANDING WITH RGB-D IMAGES USING DEEP NEURAL NETWORKS

by

Metehan Doyran

B.S., Computer Engineering, Boğaziçi University, 2015

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2018

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. H. Levent Akın for his support, patience, and guidance. I would like to thank Prof. Lale Akarun and Assoc. Prof. Sanem Sarıel for being on my thesis committee and for their valuable comments on my thesis. I would also like to thank Assoc. Prof. Albert Ali Salah and Assist. Prof. Emre Uğur for creating opportunities for me and aiding me throughout my research.

I thank all my friends and colleagues for their valuable friendship and support, namely, Yiğit Yıldırım, Alper Kamil Bozkurt, Cihan Camgöz, Nefise Yağlıkçı, Alena Beyer, Hakan Girgin, Mehmet Burak Kurutmaz, Orkut Murat Yılmaz, Mert İmre, Alp Kındıroğlu, Doğa Siyli, Orhan Sönmez, Ersin Başaran, Okan Aşık, Binnur Görer, İbrahim Özcan, Can Kurtan, Begüm Turan, Canberk Cebeci, Dicle Gülşahin, Eda Sena Şenceylan, Enis Turhan, Haluk Açarçiçek, Kübra Yiğit, and Selçuk Ener. I would like to thank Oğulcan Özdemir separately for motivating me to work hard through countless days and sleepless nights.

I would like to thank my loving parents Nevin and Cihat Doyran for raising me as an observant and creative person. In addition to that I thank my life-long teacher Arife Koman for making me the person I am today with her wisdom and music.

Last but not least, I thank my dearest friends Özge Küçükakça, Yasemin Doyran, Erdinç Hasılcıoğulları and Sumru Kanca for being supportive and for sharing their kindness, love and wisdom with me.

ABSTRACT

INDOOR VISUAL UNDERSTANDING WITH RGB-D IMAGES USING DEEP NEURAL NETWORKS

We created a visual understanding pipeline for house robots with Red-Green-Blue-Depth (RGB-D) sensors. Our pipeline consists of three components; *(i)* an RGB-D object detection network, *(ii)* a simple tracking algorithm, and *(iii)* a 3D semantic mapping module. Our constraint is running the whole system in real-time. Most RGB-D object detection networks do not have such constraint and cannot run in real-time because they require costly preprocessing methods. Instead of the costly methods, we seek ways to make raw depth data more useful by a cheap preprocessing technique of RGB overlaying on the depth data. After adding depth branches into two state-of-the-art RGB object detection networks we compared performances of feeding raw depth input and RGB overlaid depth input on the SUN RGB-D dataset. The results of using only one type of input show that our overlaying method gets 0.9% better mean average precision (mAP) than feeding the network with raw depth data. SSD with decision level fused depth network increased the mAP around 2% compared to RGB only SSD. We collected a small household object detection dataset to test the tracking method combined with the object detector. We used median flow tracking on the object boxes detected by the object detector, which increased the mAP of the object detection network by around 5% on our dataset. Our final module consists of 3D semantic mapping which we used Robot Operating System node of the Real-time Appearance-Based Mapping library for the 3D mapping part. The semantic information of the objects are created by the object detector network combined with the tracker. Although the object detector network proposes 2D bounding boxes around the objects, labeling these pixels and seeing the object from different angles allow us to create 3D maps with object labels.

ÖZET

DERİN YAPAY SİNİR AĞLARIYLA BİNA İÇİ ÜÇ BOYUTLU GÖRSEL ANLAMA

Biz, normal fotoğrafın yanısıra derinlik bilgisi de veren (RGB-D) kameraya sahip evde çalışacak robotlar için bir görsel anlama sistemi oluşturduk. Bu sistem üç temel kısımdan oluşuyor; *(i)* RGB-D bilgisiyle nesne tanıma ağı *(ii)* basit bir takip algoritması, ve *(iii)* 3 boyutlu anlamsal haritalama modülü. Bizim en büyük kısıtımız ise sistemimizin gerçek zamanlı çalışabilmesi. Çoğu RGB-D nesne tanıma ağı derinlik bilgisine uzun süren ön işleme yöntemleri uyguladıkları için gerçek zamanlı çalışamıyor. Biz de derinlik bilgisini ham olarak kullanmayla kendi geliştirdiğimiz bir ön işleme yöntemini kullanmayı iki alanında en ileri yönteme uyarlayarak karşılaştırdık. Derinlik bilgisine normal fotoğraftaki doku bilgisini de gömerek oluşturduğumuz gerçek zamanlı çalışabilir ön işleme yöntemiyle SUN RGB-D veri kümesinde sadece ham derinlik bilgisiyle elde ettiğimizden %0.9 daha iyi bir ortalama duyarlıkların ortalaması skoru elde ettik. Bu iki girdi türüyle eğittiğimiz ağları normal fotoğraf girdisi alan orijinal ağlara karar seviyesinde entegre ettiğimizde ise sadece normal fotoğraf girdisi işleyen ağa göre %2 civarında daha iyi sonuçlar aldık. Küçük bir evde nesne tanıma veri kümesi topladık ve basit bir takip algoritması ekleyerek bu veri kümesinde aldığımız skorları %5 oranında arttırdık. 3 boyutlu anlamsal haritalama yaptığımız son modülümüzde haritalama kısmını açık kaynak kodlu RTAB-Map kütüphanesi ile oluşturduk. Anlamsal içeriği ise nesne tanıma ağlarıyla ve bir takip algoritmasıyla bu haritaya ekledik. Nesne tanıma ağları 2 boyutlu nesne kutuları önermesine rağmen biz üç boyutlu harita oluştururken robotun farklı açılardan nesneleri görmesinden faydalanarak bu bilgiyi 3 boyuta döktük.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

A	3×3 convolutional filter for detecting horizontal lines
B	3×3 convolutional filter for detecting vertical lines
g_t	Ground truth value for the t^{th} instance
h_m	m^{th} hidden node of a perceptron
L	Loss function
s	Posterior probability calculated by the sigmoid function
s_i	i^{th} posterior probability calculated by the softmax function using i^{th} output of a perceptron, y_i
T	Threshold function (such as sigmoid function)
v_m	Weight of connecting the m^{th} hidden node, h_m, to the output node, y
v_{mj}	Weight of connecting the m^{th} hidden node, h_m, to the j^{th} output node, y_j
w_i	Weight of connecting the i^{th} input node, x_i, to the output node, y
w_{ij}	Weight of connecting the i^{th} input node, x_i, to the j^{th} output node, y_j
x_i	i^{th} input node of a perceptron
y	Output node of a perceptron
y_j	j^{th} output node of a perceptron
α	Weight parameter for combining different loss functions together
λ	Learning rate

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
ANN	Artificial Neural Network
AP	Average Precision
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DPM	Deformable Parts Model
F-R-CNN	Faster Regions with Convolutional Neural Network
F-R-CNN-D	Faster Regions with Convolutional Neural Network and Depth
FC	Fully Connected
FCN	Fully Convolutional Network
FPS	Frames per second
GPU	Graphics Processing Unit
HHA	Horizontal disparity, Height above the ground, and Angle with the gravity direction
HOG	Histogram of Oriented Gradients
IoU	Intersection over Union
lr	Learning Rate
mAP	Mean Average Precision
MF	Median Flow
MSC	Multi-Scale Convolutional
NMS	Non-Maximum Suppression
OpenCV	Open Source Computer Vision Library
R-CNN	Regions with Convolutional Neural Network
RGB	Red, Green, Blue
R[GB]-D	Red, Green-Blue-Combined, Depth
RGB-D	Red, Green, Blue, Depth
RoI	Region of Interest
RPN	Region Proposal Network

RTAB-Map	Real-Time Appearance Based Mapping
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SPP	Spatial Pyramid Pooling
SSD	Single Shot MultiBox Detector
SSD-D	Single Shot MultiBox Detector with Depth
SSD-D+	Single Shot MultiBox Detector with RGB Overlaid Depth
SURF	Speeded Up Robust Features
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TSDF	Truncated Signed Distance Function
VOC	Visual Object Classes
YOLO	You Only Look Once

1. INTRODUCTION

Robots are getting better and faster at performing various kinds of tasks. Increasingly they help us in many different fields, from home assistant robots to space exploration robots. With their intricate and complex design, they fundamentally sense their environment, decide what to do and take action. Each of these three essential steps are prominent but sensing comes first. Without a proper understanding of its environment, a robot becomes useless or, even worse, dangerous.

Sensing is done via sensors and there are many commercially available ones to be used when building a robot or any other system that requires to sense its environment. Cameras, ultrasonic sensors, microphones, infrared sensors, gyroscopes, force sensors, temperature sensors are examples of different types of sensors which can be used to perceive the environment. We focus on visual perception using Kinect RGB-D camera [3] which provides both a regular camera image and a depth image which consists of the distances of the pixels from the camera. It uses infrared sensors to provide depth information.

In the past humans coded robots for every single step of how to process the incoming data from their sensors. This changed when the deep learning approaches became widespread. Now instead of telling robots how to process the data we tell them what structure to use when processing the data depending on what type of data it is. These structures are called multilayer perceptrons, or more popularly deep neural networks (DNN or networks in short). Briefly, DNNs are very complicated differentiable non-linear functions which takes inputs and gives outputs. In the case of building a DNN to classify images into say 20 categories (house, car, computer, etc.), the inputs can be images (pixel values) of these objects, and the outputs can be the numbers representing which categories the input images belong. Providing a DNN with appropriate inputs and defining a loss function, which calculates the loss between the desired output and the output the DNN gives, allow them to gradually learn about the sophisticated non-linear function between the inputs and the outputs using some mathematical methods. Robots learn directly from data using several mathematical methods such as backpropagation, stochastic gradient descent, mini-batch sampling, etc. A special-

ized type of DNNs are called convolutional neural networks (CNN) which use successive convolution operations on patches of the incoming data. They are more effective than regular DNNs in robot vision and computer vision problems because the pixels in images are almost always locally dependent to each other. We selected two CNNs to be used for object detection in our thesis because they acquired the state-of-the-art results. Detailed background knowledge about perceptrons, DNNs and CNNs can be found in Chapter 2

This thesis is a part of a larger project which aims to build a home assistant robot that can safely work with people, help them physically and socially. Our goal with this work is to create a real-time object detection based three dimensional (3D) mapping system. This system should work in real-time because the robot needs to sense its environment and interact with it smoothly. The implementation consists of three parts: *(i)* object detection, *(ii)* object tracking, and *(iii)* 3D mapping. Final outputs, 3D maps, will be used by our home assistance robot to keep the locations of the objects in memory and it will use these objects for locating smaller objects around them with another slower but more precise object detector.

Widely used state-of-the-art object detectors mostly use CNNs with two dimensional (2D) images without any depth information. Since we use a Kinect depth camera on our robot which can provide both depth information and an image, we wish to boost the network's object detection performance with the help of the depth information. We modify two of the state-of-the-art 2D object detection systems for our task. These object detectors learned to use the depth information provided by the Kinect camera on our robot after our modifications and improvements. After detecting the objects we need to track them via a fast tracking algorithms to increase our system's robustness.

We also modified a state-of-the-art 3D mapping library to include the object detection and tracking module. Our final application creates a semantic 3D map of the environment in real-time. In the larger project this map will be used by the robot for interacting with its environment in order to help the people around it. Bringing a pillow to a person, organizing some chairs around a table, bringing a book from a bookshelf, putting an empty plate on a counter, throwing rubbish into a garbage bin and many more examples can be counted as

how this mapping system will be helpful to locate these objects and interact with them. The details of our methodology and implementation are given in Chapter 4.

In Chapter 5 we explain our experiments and show results for testing CNNs for object detection in two different datasets, testing different tracking implementations. We also show how our system behaves in an environment, which is previously not seen.

2. BACKGROUND

In this chapter we give a brief background about the important topics which we refer to when we explain some methods and algorithms used in the related publications. It consists of some key concepts of the machine learning field. We begin by examining the key concepts (perceptron, artificial neural networks, convolutional neural networks) in machine learning, which has made robot vision and machine vision almost as successful as human vision recently. We later explain how convolutional neural networks can be used for transfer learning.

The greatest inspiration sources of scientists are the final products of the evolutionary process, the living things including human beings. There are many different parts and organs which are evolved to solve separate problems over time. A kangaroo's legs move the kangaroo on the ground by creating a tension and releasing it, a fish's fins move the fish in the water by adjusting their angles, a chameleon's tongue catches insects to eat by acting like a catapult, an eagle's eye can detect its preys from great distances, and a human's brain can find out how to send rockets to other planets by learning how to solve highly complex problems. These expert parts inspire and push our technological growth. Since technology includes many fields, our scope will be limited to the robotics and vision perspective. Robotics researchers try to replicate many of these evolved animal parts to understand the idea behind them and by doing this they expand the toolbox of engineers. Some examples are given below:

- Graichen *et al.* [4] developed a kangaroo robot, inspired by the locomotion of a kangaroo and its efficiency where the energy spent for the previous jumps are stored back to be used for the future jumps when landing.
- Behbahani and Tang [5] designed a fish robot by replicating the fins of fish, which can be used for underwater search and rescue operations.
- Chameleon tongue gave Hatakeyama and Mochiyama [6] an idea to design a shooting manipulation system.

- Duan *et al.* [7] created a simulation platform based on the eagle eye for guiding unmanned flying vehicles.

In the examples above, we excluded the most puzzling and the most complex end product of evolution, the human brain. This organ is the source of inspiration for most machine learning systems get. We begin with one of the first computational models of human brain, the perceptron [8]. Later in this chapter, we move on to its evolution through multilayer perceptron to deep neural networks. Finally we will explain a specific type of deep neural networks which is called convolutional neural networks.

2.1. Artificial Neural Networks

The perceptron model includes some of the structural elements of the human brain, i.e. neurons and synapses development of which mark the beginning of the artificial neural networks (ANNs) era. In this section, different types of perceptrons are given to show how these biologically inspired artificial neural networks have begun to be used in computer science, computer vision, machine vision and many more other fields.

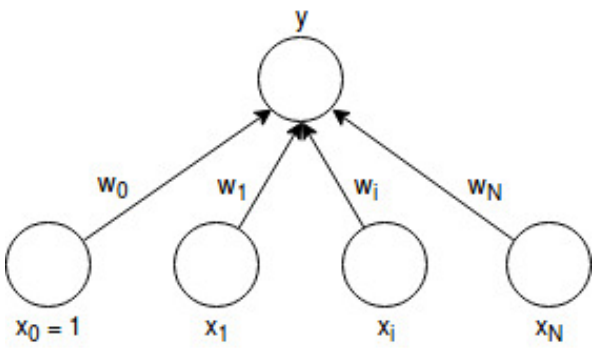

Figure 2.1. A simple perceptron with one output node.

Human brain works on a simple principle of neurons getting activated or not. The complexity and the computational power comes from the synapses that create an intricate connectivity of neurons, which gives the brain parallel processing power, and also synapses affect neurons signal level causing them to respond differently. The perceptron is a computational model using the same principle, it has neurons (inputs, outputs) and synapses (weights). Inputs are multiplied with the corresponding weights which change their signal

strength and by summing them up in the output neurons they generate the output depending on the inputs and weights [9]. Figure 2.1 shows a simple perceptron where x_0 acts as the bias unit, and N inputs (x_i) are connected to a single output y via their corresponding weights (w_i where $i = 1,\ldots,N$). The simple formulation of the perceptron is given in Equation 2.1 which defines a hyperplane in N-dimensional space:

$$y = \sum_{i=1}^{N} w_i x_i + w_0. \tag{2.1}$$

Equation 2.1 by itself is nothing new that the perceptron concept introduces, the perceptron is only a way of implementing the hyperplane defined by the formula. Using this linear equation with one input we can implement a line, or with more than one input it will be a hyperplane to be used in multivariate linear fit. These hyperplanes can also be used as linear discriminant functions separating the input space into two for 2-class classification problems. These classes' posterior probabilities can be calculated by feeding the output neurons score into a threshold function such as the sigmoid function given in Equation 2.2 (where y is calculated using Equation 2.1).

$$s = \frac{1}{1 + exp(y)} \tag{2.2}$$

A perceptron can consist of more than one output neuron, say K neurons (Figure 2.2 where y_j ($j = 1,\ldots,K$) are the output neurons, and N inputs (x_i) are connected to them via their corresponding weights (w_{ij} is the weight for the synapse that connects x_i input to y_j output)), where it behaves like K perceptrons with different weights, and these perceptrons can be used to solve K-class classification problems. If one needs the posterior probabilities of the classes, the different weighted sums (K outputs) can be fed into a threshold function such as *softmax* function given in Equation 2.3 (where $i = 1,\ldots,K$ and y_i is a single output and s_i is its output after the softmax function).

$$s_i = \frac{e^{y_j}}{\sum_{k=0}^{K} e^{y_k}} \tag{2.3}$$

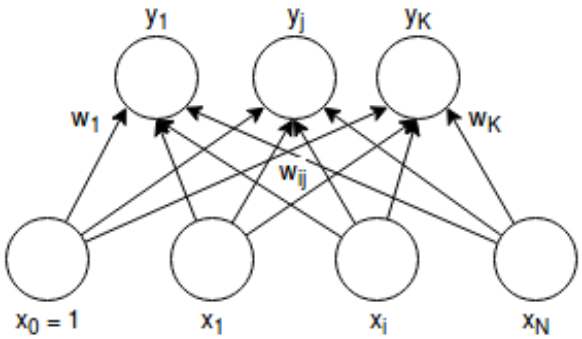

Figure 2.2. A perceptron with K output nodes.

We need to find the hyperplane which fits best to a specific problem. Therefore we need to train our network's (perceptron's) weights by minimizing an error function which calculates the fitness of a hyperplane. Even though this can be done offline (learning from all the sample data at once), most machine learning methods used in training ANNs are online (learning from each new data point gradually). The reasons for that will be clearer in Section 2.2 as the dimensionality of our networks becomes huge and the sample data needed to train the networks becomes millions of images for machine vision problems. An online learning method consists of an update rule which updates the weights of the network trying to lower the error between the output of the network for an instance from the sample data and the correct value that should be the output of the network, without forgetting the previously learned weights from other instances. The error function to be used must be selected properly considering the problem to be solved. We can use the sum of squared errors function given in Equation 2.4 for linear regression problems (where t is the index of a single instance from the sample data, g^t is the ground truth value for that single instance, weights are the weights of the network w_i: i from 0 to N, $inputs^t$ are the inputs of that single instance x_i: i from 1 to N). The update rule for updating the weights can be derived by finding the gradient of the error function (Equation 2.4) with respect to all the weights of the network, w_i. Equation 2.5 shows the update rule, where λ is the learning rate, x_i^t is the i^{th} input and w_i^t is the i^{th} weight for $i = 0, \ldots, N$.

$$Error^t(weights|inputs^t, g^t) = \left(g^t - \sum_{i=1}^{N} w_i x_i^t + w_0 \right)^2 \tag{2.4}$$

$$\Delta w_i^t = \lambda (g^t - y^t) x_i^t \tag{2.5}$$

Another error function example can be the cross entropy error function in Equation 2.6 which is suitable for classification problems (where t is the index of a single instance from the sample data, g^t is 0 if the instance belongs to first class, 1 if the instance belongs to the second class, weights are the weights of the network w_i: i from 0 to N, $inputs^t$ are the inputs of that single instance x_i: i from 1 to N, and s^t is the result of feeding y^t, the output of the network for the instance t, into the sigmoid function in Equation 2.2). The update rule for this error function is defined by the gradient of it with respect to the weights of the network, w_i. Equation 2.7 shows the update rule, where λ is the learning rate, x_i^t is the i^{th} input and w_i^t is the i^{th} weight for $i = 0, \ldots, N$.

$$Error^t(weights|inputs^t, g^t) = -g^t log(s^t) - (1 - g^t) log(1 - s^t) \tag{2.6}$$

$$\Delta w_i^t = \lambda (g^t - s^t) x_i^t \tag{2.7}$$

Both of the update rules find the gradient of the error function which can be used in a method called *stochastic gradient descent*. Even though this approach uses the same idea of normal gradient descent algorithm, following the gradient which minimizes the error function, the difference between them is that stochastic gradient descent learns from data one by one (or in mini-batch gradient descent it learns from small batches of data). This procedure speeds up the training of the network while getting a good approximation of the gradient of all the training data. This good enough approximation and stochastic jumps are also helpful because the normal gradient (gradient of all the instances) may lead the method to get stuck in one of the local minima while the stochastic gradient explores the function which is being optimized better.

A perceptron consisting of an input layer and an output layer can only solve linearly separable problems, where the input space can be separated into two with a hyperplane. Although there are many problems that are linearly separable, most of the real world problems

are not (implementing XOR function being one of the simplest and implementing object detection being one of the hardest). The real power of the perceptron comes from the non-linearity it introduces when it has one or more hidden layers (Figure 2.3 where y is the output node, h_m (m from 1 to M) are the hidden nodes with bias (h_0) which are connected to the output nodes via their corresponding weights (v_m), and N inputs (x_i) with bias (x_0) are connected to the hidden nodes via their corresponding weights (w_{im} is the weight for the synapse that connects x_i input to h_m output) and T inside the nodes represent the threshold function (e.g. sigmoid) which gives the perceptron non-linearity.). Each hidden layer applies a threshold function (activation function) such as the sigmoid function in Equation 2.2 to the weighted sum of the connected previous layer's nodes. At each hidden layer the perceptron becomes more complex.

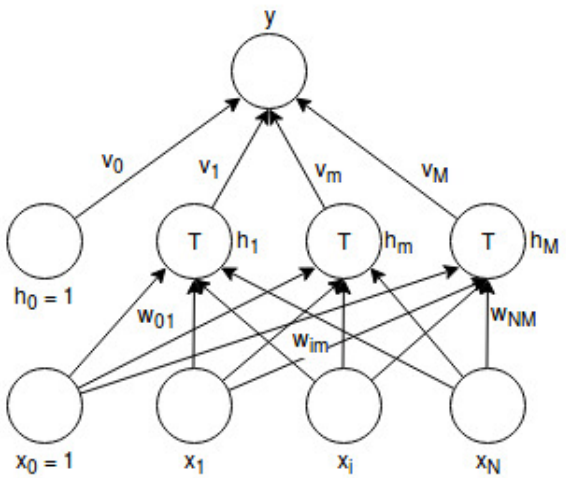

Figure 2.3. A multilayer perceptron with one hidden layer and a single output.

In multilayer perceptrons, calculating the hidden layer nodes' outputs are the same as calculating the weighted sums of the previous layer's values (input layer's in our example) and feeding them into a threshold function which in our case a sigmoid function:

$$h_m = sigmoid\left(\sum_{i=1}^{N} w_{im}x_i + x_0\right) \tag{2.8}$$

Using these h_m values for the hidden nodes' values, one can calculate the output node's value using the following equation:

$$y = \sum_{m=1}^{M} v_m h_m + v_0 \tag{2.9}$$

Assume that we want to solve a non-linear regression problem using a single output multilayer perceptron. We can use the sum of squared errors as our error function:

$$Error^t(weights|inputs^t, g^t) = (g^t - y^t)^2 \tag{2.10}$$

Then we can use its gradient for the update rule. Updating the second layer's weights are straight forward because we can calculate the output node's value with the values of the hidden nodes, and we know what the ground truth output should be from our training data:

$$\Delta v_m^t = \lambda (g^t - y^t) h_m^t \tag{2.11}$$

Update rule for the first layer's weights is trickier, because we do not know the ground truth values for the hidden nodes. The Backpropagation method [10] comes into play by using the chain rule it propagates the error from the final layer to the previous layers:

$$\frac{\partial E^t}{\partial w_{im}} = \frac{\partial E^t}{\partial y^t} \frac{\partial y^t}{\partial h_m^t} \frac{\partial h_m^t}{\partial w_{im}} \tag{2.12}$$

Therefore the first layer's weights can be found by the following equation:

$$\Delta w_{im}^t = \lambda (g^t - y^t) v_m h_m^t (1 - h_m^t) x_i^t \tag{2.13}$$

An artificial neural network can take many different shapes from having a single input node and an output node to much more complex structures like having hundreds of input nodes and many hidden layers. Figure 2.4 shows a multilayer perceptron with a single hidden

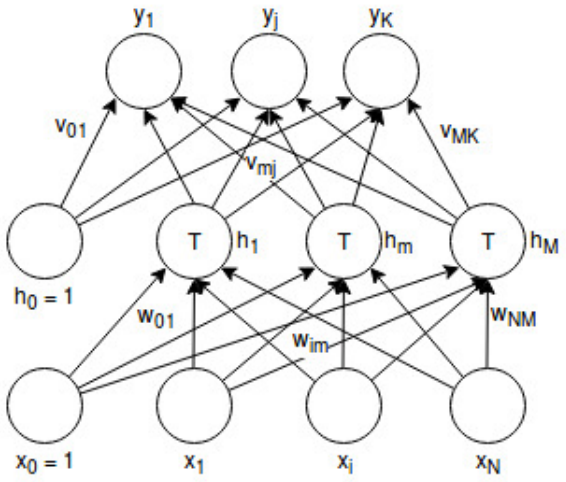

Figure 2.4. A multilayer perceptron with one hidden layer and K output nodes.

layer and K output nodes (where y_j $(j = 1,\ldots,K)$ are the output nodes, h_m $(m = 1,\ldots,M)$ are the hidden nodes with bias (h_0) which are connected to the output nodes via their corresponding weights (v_{mj} is the weight for the synapse that connects h_m hidden node to y_j output node), and N inputs (x_i) with bias (x_0) are connected to the hidden nodes via their corresponding weights (w_{im} is the weight for the synapse that connects x_i input to h_m output) and T inside the nodes represent the threshold function (e.g. sigmoid) which gives the perceptron non-linearity.)

2.2. Convolutional Neural Networks

The function that is to be learned by the ANNs from the data gets more complex and more non-linear according to the complexity of the data and the difficulty of the learning task. In such cases, we may need many hidden layers to get to the abstraction level needed. This type of ANNs consisting of many layers are called *deep neural networks* (DNNs). In general, the layers of the DNNs are fully connected, in other words, every node in a layer is connected to every other node in that layer. In some cases using fully connected layers and feeding all the input data into every hidden node become useless as in most computer vision tasks. Convolutional layers typically better suited for computer vision tasks because there are local dependencies between closer pixels in images and videos.

Convolutional neural networks (CNNs), having convolutional layers, learn better and faster than regular DNNs, having fully connected layers, from images because the pixels are mostly locally dependent, that is to say closer pixels are more likely to be dependent than the further pixels. Using convolutional layers lower the number of parameters drastically compared to using fully connected layers, which is another great advantage of convolutional layers over fully connected layers. In CNNs the hidden nodes are only fed with a patch of data (e.g square windows in two dimensional (2D) input data, or cuboids in three dimensional (3D) input data) which allows them to learn the local information. CNNs consisting of only convolutional layers are called *Fully Convolutional Networks* (FCN).

Convolutional layers apply 2D convolution operation on their input, and output the resulting activation map. Convolution is a mathematical operation of applying an impulse response function (h) to another function (f), resulting in a modified version of the f function. In our case, the f function is a 2D image having discrete pixel values in x and y direction. Impulse response functions to be applied on the images are also 2D. They are called *filters* or *kernels*. Resulting 2D functions are called *activation maps* because they include activated regions after summing the multiplied pixels with the corresponding weights of the convolutional filters.

An interesting characteristic of convolutional layers is that they do not need to learn large number of filters for each separate area of the image. This advantage is a direct result of a simple logical assumption that if a filter works well at some spatial position, it will most likely work well at a different position. This is to say learning a filter at the bottom left corner of an image which recognizes edges will be the same filter even if it is learned separately at the top right corner of that image as long as it recognizes edges. This assumption allows us to slide a single filter over an image to create an activation map of that filter over that image. Imagine a dataset consisting of images with widths of 640 pixels and heights of 480 pixels. If we want to learn convolutional filters from each five by five squares from them that will result in $12,288$ filters, each having 25 parameters. Since there can be edges, corners, similar shapes all around the images, most of these $12,288$ filters will be duplicates of each other which will become experts of very similar types. Instead of using that many filters we can choose a much smaller number of filters and slide them through every possible five by

five areas of the images. This results in magnitudes of smaller number of parameters to be learned and used in CNNs.

The convolutional filters in the first few layers of a CNN can be thought as low level experts which can identify edges, some specific colored regions, vertical or horizontal lines, etc. As we move on towards the higher levels of CNN, convolutional filters can begin to be activated by more complex shapes, texts, faces, and even objects. Two very simple 3×3 convolutional filter examples can be seen in Figure 2.5.

$$A = \begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \qquad B = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

Figure 2.5. Two convolutional filters.

Doing convolution operation with filter A on a horizontal line results in the biggest activation so we can say that when we slide filter A over an image, it creates an activation map of horizontal lines. Filter B creates activation maps for vertical lines, vertical line of twos in the second column strengthens pixel values corresponding to their location and the other negative ones decrease the pixel values on their correspondent locations. As a result, doing convolution operation with filter B on an image of a one-pixel-wide vertical white line with black background around it results in the highest activation possible. (White color is 255, and black color is 0 as pixel values in 256-bit color format which is accepted and used most widely) An example of activation maps after doing convolution with filters A and B on an image (Figure 2.6) is shown in Figure 2.7 and in Figure 2.8 respectively.

The most significant reason why CNNs became the current norm in robot vision and machine vision is that previously learned filters of a CNN in a domain can be transferred easily to a different domain which has a similar type of input. Training CNNs is very costly (training time ranging from days to weeks, thousands or millions of labeled data, processing power which sometimes requires clusters of the most recent graphics processing units

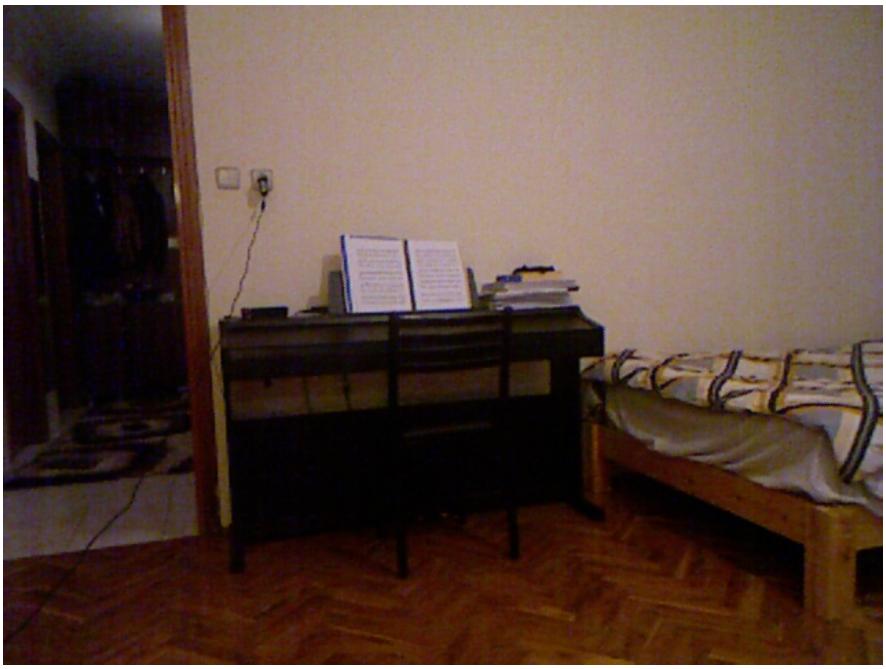

Figure 2.6. An image from the indoor mapping video we captured.

Figure 2.7. Resulting activation map of applying horizontal line filter (filter A) to the image in Figure 2.6.

Figure 2.8. Resulting activation map of applying vertical line filter (filter *B*) to the image in Figure 2.6

- GPUs) to train one. Using transfer learning from a previously trained CNN dramatically decreases all the costs mentioned above.

Transfer learning with CNNs is done in three ways most commonly; *(i)* using CNN as a feature extractor, *(ii)* fine-tuning CNN in a new domain, or *(iii)* using first layers of CNN as a feature extractor and fine-tuning the domain specific last layers. If the two domains have similar inputs such as object detection and image classification, one can train a CNN in one of them and use it without changing the parameters in the other domain as a feature extractor. The resulting output of the CNN are called deep features or convolutional features and can be used with classifiers such as softmax classifiers or Support Vector Machines (SVMs). When the performance of using the convolutional features is insufficient, one needs to fine-tune the network on the new domain. Fine-tuning is updating the pretrained model weights on a new domain without completely losing the previously learned weights from the previous domain. Another way of using transfer learning is to combine the two approaches depending on the domain differences. One can keep the first few layers of a CNN unchanged because they mostly learn filters which gets activated with very low level features like simple shapes or

colors. Then fine-tuning the higher levels can result in a much better performance than using the two approaches separately. In this work we used transfer learning by fine-tuning two of the state-of-the-art object detection CNNs which were pretrained on ImageNet dataset [11].

3. RELATED WORK

This chapter builds on the previous chapter by presenting the object detection problem and explaining how the biologically inspired machine learning structures can be implemented to solve it. The evolution of the recent state-of-the art object detection methods starting from the huge success of a CNN called AlexNet [12] is explained in detail. The next section introduces another problem, object tracking. Section three dives into semantic mapping with 3D cameras. The final section concludes the chapter by giving details about the datasets that are used in this work.

3.1. Object Detection with CNNs

Visual understanding consists of various different problems such as *image classification*, *semantic segmentation*, *scene understanding*, and *object detection*. This thesis focuses on the *object detection* problem which includes some other easier problems like *object classification* (or *image classification* if the images consist of only one object) and *object localization* (which is a very similar procedure as creating object proposals in object detection systems).

Convolutional neural networks era in computer vision begins with [12] introducing a convolutional network (AlexNet) to solve the image recognition problem that performs significantly better than its ascendants which use hand-crafted features such as HOG [13], SIFT [14], SURF [15], etc.

3.1.1. Detection with RGB

Object detection is commonly done using 2D images, having only red green and blue (RGB) channels. Summary of the most influential RGB object detection networks is shown in Figure 3.1. The boxes are shown chronologically from left to right. All the networks in it and their influences on one another are explained later on.

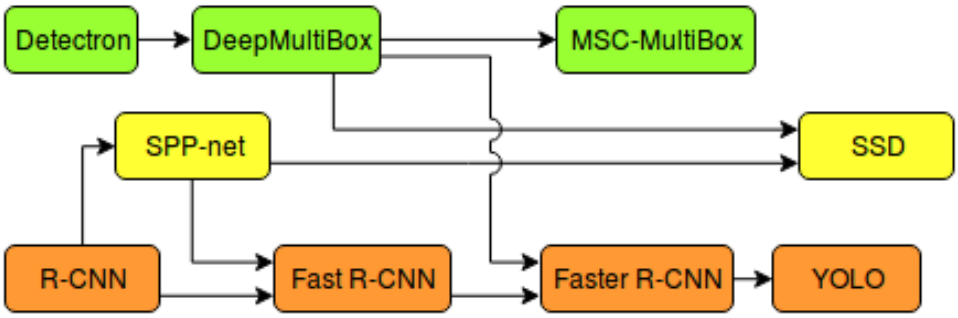

Figure 3.1. Evolution of RGB object detection networks. Arrows show the influences of the networks on the others.

AlexNet implementation [12] for the image recognition problem has inspired many researchers working on the object detection problem. There are mainly two distinct paths taken by [16] and [17]. The framework in [16] is called *R-CNN* (Regions with Convolutional Neural Network) for object detection. Figure 3.2 shows the overview of the *R-CNN* implementation explained in the original paper. The steps are the following:

(i) An object (region) proposal method proposes around 2000 rectangular regions in an image.

(ii) For each region the following steps are applied:

  (a) The proposed region in the image is preprocessed to be a proper input for the CNN that is being used. (They warp the proposed region to be used in the modified AlexNet which takes 227×227 square image)

  (b) CNN's output (convolutional features) are fed into linear SVMs which find the class of the proposed region (as one of the objects or the background)

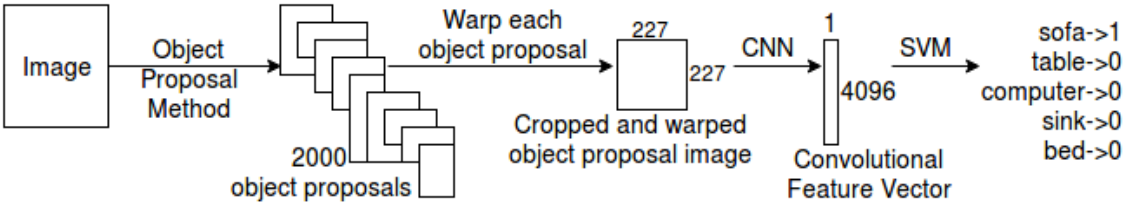

Figure 3.2. *R-CNN* framework

[16] also implement bounding-box regressors which learn for each class how to tighten the bounding boxes proposed by the object proposal method (e.g. selective search [18] in the original paper) to improve the detection results. The implementation follows the same approach of DPM (Deformable Part Models) [19] except that R-CNN trains the class-specific regressors from the convolutional features calculated by the CNN.

An alternative method called Detectron [17] uses many CNNs each of which localizes objects of a single class. Instead of the softmax layer, they use a regression layer. Rest of the network follows the AlexNet implementation. They train 5 CNNs for each object class, one for the whole object mask, the rest for the top, bottom, left and right object masks. They all do regression towards the ground truth bounding boxes, and at the end combined together to have a stronger detector.

In [20], the Detectron [17] method is criticized for having 5 networks for each object class, therefore not being applicable to real world applications. Their alternative method, DeepMultiBox [20] uses only two networks, an object proposal network and a classification network. They also follow the AlexNet implementation for the classification network. However, the object proposal network is a novel deep learning approach which minimizes two loss terms together, *objectness confidence score* and *bounding box location*. They maximize the objectness confidence score (converging to 1) if the predicted bounding box matches with a ground truth box of any object classes, otherwise they minimize the objectness confidence score (converging to 0). They say that the convergence of the randomly initialized prediction boxes is problematic, therefore they propose a method to propose prior boxes using k-means clustering of the ground truth box locations. They get a 200 times faster test time than the *R-CNN* approach, while having slightly worse detection performance on the VOC 2007 object detection dataset[1] . [21] improves the *R-CNN* approach by merging it with the Spatial Pyramid Pooling (SPP) approach, and call their method SPP-net. Feeding 2000 regions with lots of overlaps into CNN is the biggest reason why R-CNN runs very slow even in testing. [21] also use selective search like the *R-CNN* method and gets 2000 region proposals for each image, but instead of cropping and warping the image, they feed the whole image into their

[1]Most of the object detection methods publish their results on the Pascal Visual Object Classes (VOC) Challenge [1] dataset.

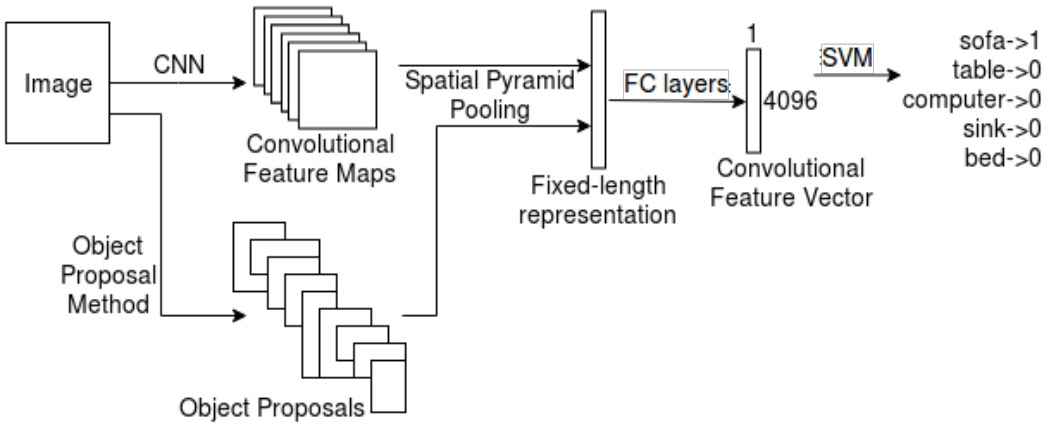

Figure 3.3. SPP-net framework

SPP-net, calculate convolutional feature maps, then from the proposed regions on the maps they apply spatial pyramid pooling. That way they collect fixed sized representations for each proposal region which are comparable to each other and can be used to train SVMs (Figure 3.3). They also follow the linear SVM training procedure same as the R-CNN approach. The methodological difference in feeding the whole image results in 38 times speed up in detection time while preserving almost similar detection performance of *R-CNN* on the Pascal VOC 2007 object detection dataset.

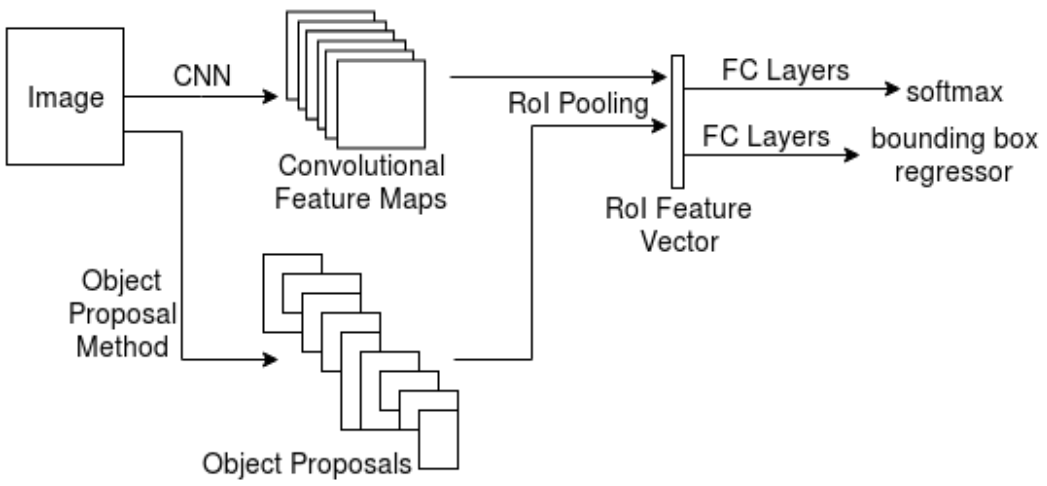

Figure 3.4. Fast R-CNN framework

The next step in object detection comes with *Fast R-CNN* [22] which inherits the idea of *feeding whole images into CNNs* from *SPP-net* to *R-CNN* and enhances it by proposing a single stage method which eliminates the obligation for training SVMs. *Fast R-CNN* uses

SPP-net's idea of pooling from convolutional feature maps (Figure 3.4), but only uses single scale pooling which pools directly the corresponding size of the proposed region on the feature maps. This is called the Region of Interest (RoI) pooling layer. After this layer calculates a RoI feature vector, one set of fully connected (FC) layers output softmax scores, and another branch of FC layers output bounding box regressor. Similar to [20], Fast R-CNN utilizes a multi-task loss, but instead of using the objectness scores it uses class scores. Different from the bounding box regressors of both R-CNN, SPP-net, and DeepMultiBox using L_2 loss in their loss function, Fast R-CNN method uses smooth L_1 loss (Equation 3.1), which is a differentiable version of L_1 loss. It is known that smooth L_1 loss is more robust to outliers than L_2 loss because L_2 loss penalizes large errors drastically.

$$|x|_{smooth_{L_1}} = \begin{cases} 0.5x^2 & \text{if } |x| \leq 1 \\ |x| - 0.5 & \text{if } |x| > 1 \end{cases} \tag{3.1}$$

Fast R-CNN also utilizes the truncated singular value decomposition (SVD) approach in its fully connected layers. Truncated SVD approximates the fully connected layer's weight matrix, learned in training stages, considerably lowers the computation need spent for multiplication operations in test time while preserving the performance of the original network. In test time it is 213 times faster than R-CNN, and in training time it is 8.8 times faster, while achieving significantly better results on the Pascal VOC 2012 object detection dataset [23].

Faster R-CNN (F-R-CNN) [24] merges Fast R-CNN and DeepMultiBox's [20] region proposal network into a single network that can be trained end to end (Figure 3.5). Considering the convergence speed, they adopt a 4-step alternating training approach:

(i) training the region proposal network (RPN),
(ii) using the regions it proposes training the Fast R-CNN,
(iii) fine-tuning only the RPN layers again, and
(iv) fine-tuning the unique layers of Fast R-CNN.

Unlike DeepMultiBox's region proposal network using k-means clustering to generate priors, RPN uses predefined prior boxes with different scales and aspect ratios (they use three

scales and three aspect ratios in their papers) in a sliding window over the convolutional feature maps. They call these prior boxes anchors. These anchors are translation invariant since the same anchors are calculated at every sliding window location. DeepMultiBox's prior boxes are critized for not being translation invariant.

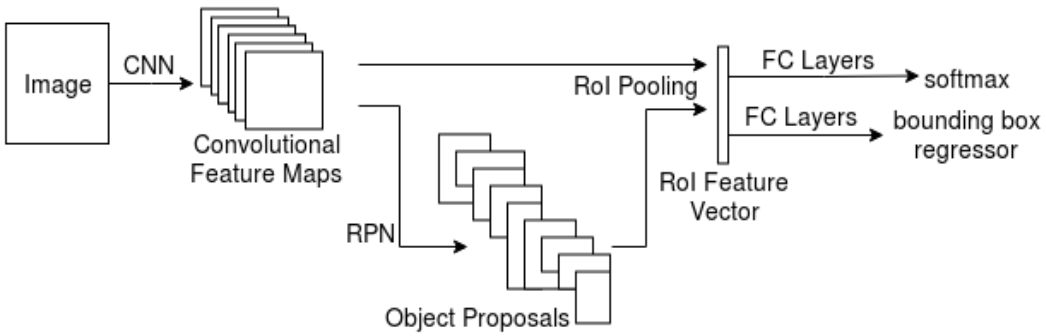

Figure 3.5. Faster R-CNN framework

Multi-scale convolutional MultiBox (MSC-MultiBox) improves the prior box generation stage of DeepMultiBox. Instead of using k-means clustering, MSC-MultiBox uses convolutional priors where they maximize the image coverage while preserving some minimum overlap threshold (0.5 in their paper) with any ground truth boxes. They also use an improved Inception-style [25] CNN where different scales of detection boxes are predicted as the network's outputs.

Before You Only Look Once (YOLO) [26], object detection could not be done in real-time with high accuracy. On the VOC 2007 dataset, the best performing method was Faster R-CNN with a mean average precision (mAP) score of 73.2 with a 7 frames per second (FPS) processing rate. YOLO with 45 FPS got an mAP score of 63.4 which is almost 7 times faster than Faster R-CNN with tripling the mAP score of the best performing real-time object detection system before YOLO (30Hz DPM [27], 26.1% mAP). YOLO system implements a similar idea to MSC-MultiBox's prior generation with maximum image coverage. YOLO divides the image into equal sized grids (7×7 in their paper), and for each grid they predict some constant number of bounding boxes (2 in their paper). That way they force their system to predict object bounding boxes from every part of the image.

Single Shot MultiBox Detector (SSD) [28] uses smooth L_1 loss just like the Fast R-CNN method. It also uses predefined prior boxes like the anchor boxes in the RPN of Faster R-CNN. Faster R-CNN uses these anchor boxes to predict object bounding boxes. Later, they use these boxes to pool convolutional features and use them to train and calculate classifiers. This approach forces Faster R-CNN to train their network in an alternating training scheme. Unlike Faster R-CNN's 4-step alternating training, SSD trains all the layers of the network together. This is achieved by SSD to generate object class scores in object bounding boxes. SSD inherits and extends DeepMultiBox's MultiBox loss function (Equation 3.2) which simultaneously predicts object bounding boxes and evaluates their object class scores. This loss function (L) is a weighted sum of softmax loss of confidence scores (L_{conf}) and smooth L_1 loss of the difference between the ground truth boxes and predicted object bounding boxes(L_{loc}). In their paper they say they found α to be equal to 1 by cross validation.

$$L = L_{conf} + \alpha L_{loc} \tag{3.2}$$

3.1.2. Detection with RGB-D

Following the commercialization of depth cameras 3D images became more available and some researchers searched different ways to improve object detection using 3D images, having a depth channel in addition to a regular RGB image (RGB-D). Most of the work in the literature suffers from the lack of a large RGB-D dataset.

Different from RGB datasets containing millions of labeled images, RGB-D datasets only have tens of thousands of labeled instances. Researchers explore different ways to overcome this problem. One way some researchers choose is using expensive preprocessing techniques for the depth data. These preprocessing techniques find some high level abstraction over the depth image, and they take some of the load off of machine learning systems.

Two of the most commonly used expensive preprocessing techniques on depth data are HHA (horizontal disparity, height above the ground, and angle with the gravity direction) encoding [29,30], and TSDF (Truncated Signed Distance Function) [31–33]. These encoding methods are costly and therefore not suitable to be used when there is a real-time processing constraint.

The second way of overcoming the difficulty caused by the lack of large depth datasets is proposed in [34]. They apply weakly supervised learning which starts from pretraining on a very large dataset such as ImageNet and transferring that knowledge to another problem such as object detection with depth data. They use a Gaussian Process Classification (GPC) to output objectness scores. Labeling few of these outputs is enough to train GPC to learn and as it learns gradually it labels the unlabeled data in an unsupervised manner. The problem of this approach is even it uses few labeled data, it heavily depends on collecting data in a controlled environment such as each set of images (or a video) should contain only one type of object.

3.1.3. Non-Maximum Suppression

Non-Maximum Suppression (NMS) is a technique that is widely used by object detection systems. Since these systems may detect different parts of the same object multiple times, they apply NMS over these detection boxes depending on the intersection over union (IoU) threshold to remove the redundant overlapping boxes in order to leave the box with the highest confidence score. The algorithm starts from the detection box that has the highest confidence score. Then it finds other detection boxes that have IoU rates above the threshold (0.3 for Faster R-CNN, and 0.45 for SSD). The algorithm removes these extra detection boxes because it assumes that they are the same object as the box with the highest confidence score. This procedure continues until there are no largely overlapping detection boxes, which have IoU higher than the NMS threshold, left. NMS is applied over the detection boxes of the same object class.

An example overlapping detection boxes for the chair class are in Figure 3.6. They have confidence scores of 0.89 and 0.56. NMS algorithm starts from checking all the over-

lapping boxes with the box which has the highest confidence score (0.89 in this example). Then it checks the IoU rate between this box and the other boxes. Since the IoU rate between these two boxes is higher than 0.45 (which is the default NMS threshold used for SSD), the detection box with the confidence score of 0.56 will be removed.

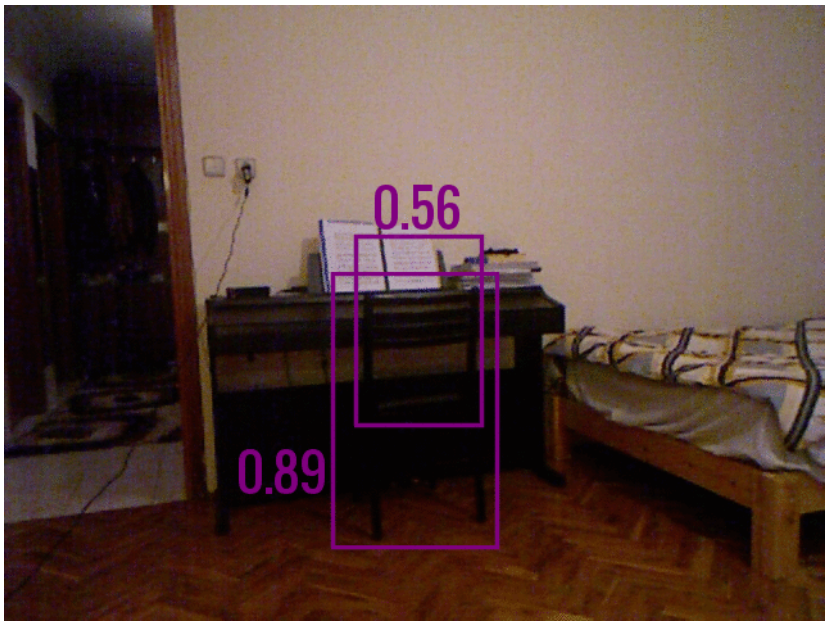

Figure 3.6. Chair detection boxes with different confidence scores.

3.1.4. Mean Average Precision

Mean Average Precision (mAP) is the most frequently used evaluation metric for the problems where the ranking matters. In an object detection system, ranking the detection boxes from the one with the highest confidence score to the one with the lowest, it is better that the higher confidence scored boxes are true positives (correctly labelled and matches a ground truth). That is why ranking matters for object detection systems and therefore mAP is a well-suited evaluation metric for the object detection problem.

There are object boxes as both ground-truth boxes and predicted detection boxes for training object detection systems. Matching these detection boxes to label them as true or false needs an IoU threshold. This threshold is 0.5 for the SUN RGB-D dataset.

mAP is the mean of the Average Precision (AP) values of all the object classes. Calculating AP for a class requires following steps (explained on an example):

(i) Ranking the detection boxes from the highest confidence score to the lowest (Table 3.1).

(ii) Matching these detection boxes with ground-truth boxes starting from the highest ranked one. This step requires a minimum IoU threshold to match the boxes. If there are more than one matching ground-truth box, the one with the highest IoU is selected. If there are more than one matching object detection box with the same ground-truth box, the object detection box with the highest rank is counted as true positive and the rest will be counted as false positives.

(iii) Calculating precision and recall values starting from the highest ranked detection (Table 3.1).

(iv) Calculating the area under the precision-recall curve (Figure 3.7) by approximating it with maximum precision-recall curve (Figure 3.8). Maximum precision values are calculated for different recall values by choosing the maximum precision level for all the recall values above the current recall value (Table 3.2).

(v) The resulting value of the area is AP.

3.2. RGB-D Mapping

Real-Time Appearance Based Mapping (RTAB-Map) is an open source library which uses a graph-based Simultaneous Localization and Mapping (SLAM) [35] and an online loop closure detection technique [36]. In Figure 4.6 if the orange box shows the pipeline of RTAB-Map. It starts taking an RGB and a depth image from the camera. Then it calculates the robot pose by the visual odometry cues such as optical flow and feature matching techniques. If the images are selected as the key-frame (enough new and important info to be applied into the map), then this key-frame is added to the global graph, which is a graphical representation of the map. The system checks if there is any loop-closures, any matching key-frames in the graph to optimize the graph accordingly. After the graph optimization an odometry correction is calculated and a 3D map is generated using the graph nodes.

Table 3.1. Example ranking, precision and recall scores of 10 predicted object detection boxes when there are 4 ground-truth boxes

Confidence	Rank	Positive?	Precision	Recall
0.99	1	True	1.00	0.25
0.95	2	True	1.00	0.50
0.93	3	False	0.66	0.50
0.81	4	False	0.50	0.50
0.79	5	True	0.60	0.75
0.76	6	False	0.50	0.75
0.74	7	False	0.43	0.75
0.68	8	True	0.50	1.00
0.66	9	False	0.44	1.00
0.61	10	False	0.40	1.00

Table 3.2. Example maximum precision values for different recall values for the Table 3.1.

Recall	Max Precision
0.0	1.00
0.1	1.00
0.2	1.00
0.3	1.00
0.4	1.00
0.5	1.00
0.6	0.60
0.7	0.60
0.8	0.50
0.9	0.50
1.0	0.50

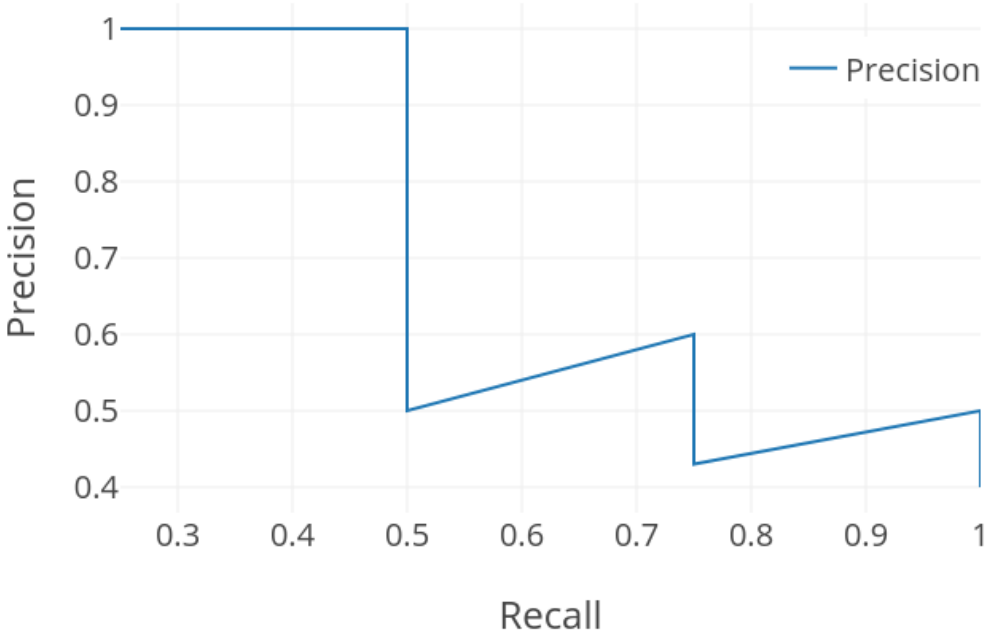

Figure 3.7. Precision-recall curve.

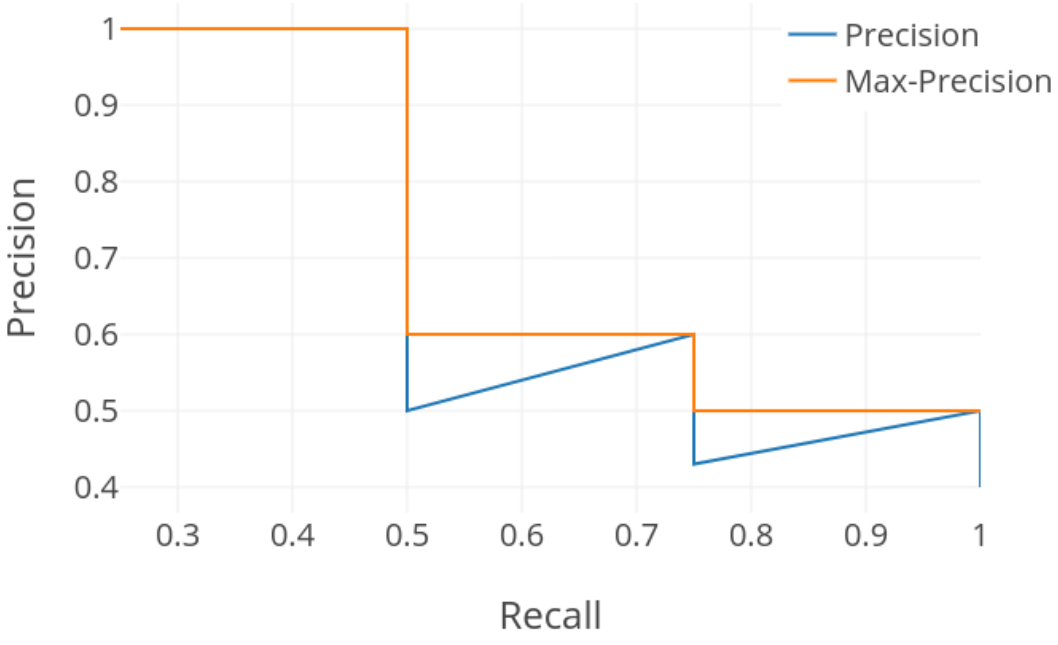

Figure 3.8. Max precision-recall curve over precision-recall curve.

RGB-D semantic mapping researches like [37], [38], [39] uses pixel-wise semantic segmentation systems which depend heavily on pixel-wise precisely labeled data to train them effectively. We aim to overcome this problem by mapping much simpler labels such as 2D object boxes to pixels in RGB-D maps.

3.3. Datasets

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset [11] provides object recognition and detection datasets which have millions of labeled images. Since the dimensionality of the very deep object detection networks (F-R-CNN, SSD, etc.) is huge, it takes millions of images to train them. After the networks gather the abstract knowledge about the objects they can be fine-tuned for different and much smaller datasets, which makes us of transfer learning with CNNs (as mentioned in Section 2.2).

3.3.1. PASCAL VOC Datasets

State-of-the-art object detection methods use PASCAL VOC datasets for benchmarking and comparing results with other object detection methods. Each year new challenges are announced on PASCAL VOC datasets. More labeled instances and more object classes are published with each new challenge. PASCAL VOC 2007 and PASCAL VOC 2012 datasets are the most common used ones. Researchers still publish their object detection method's performance on these two datasets. They both have 20 object classes for object detection which are person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, tv. They include both indoor and outdoor images.

PASCAL VOC 2007 dataset has $9,963$ images, with $24,640$ labeled objects. PASCAL VOC 2012 dataset increased the total number of images to $11,540$, with $27,450$ labeled objects. You can see some example images from the PASCAL VOC object detection datasets in Figure 3.9.

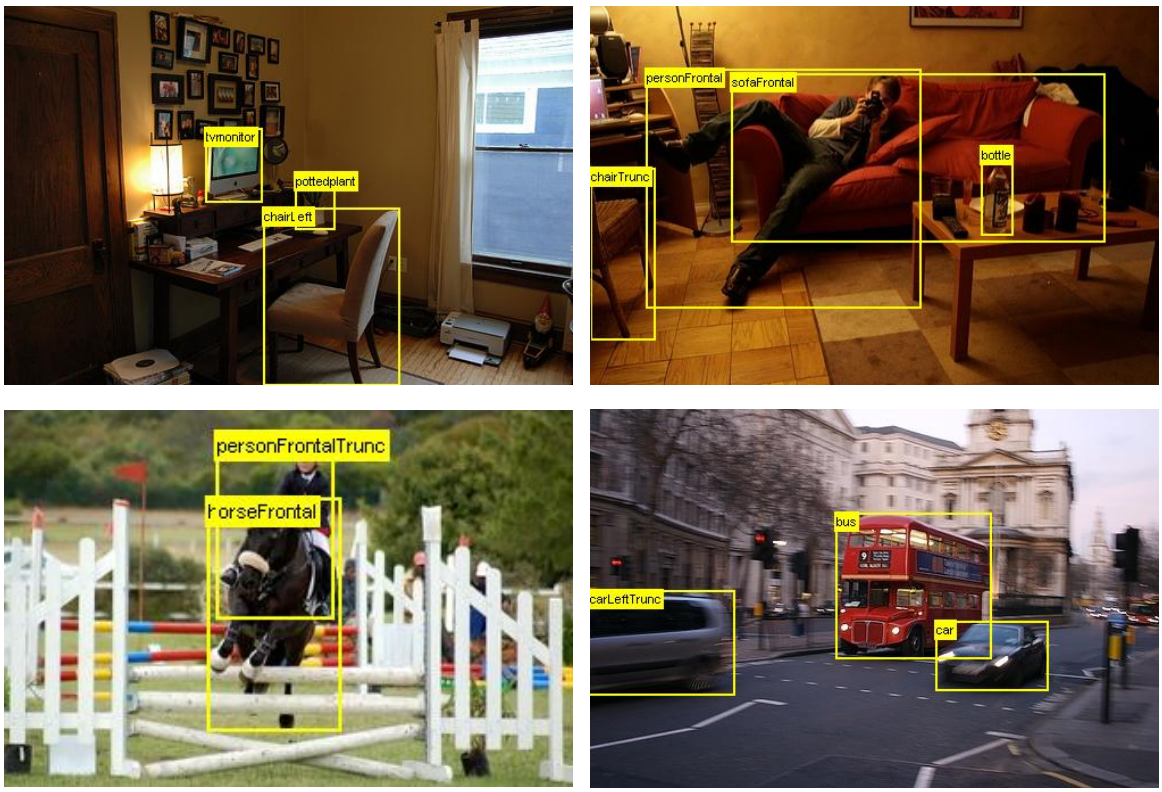

Figure 3.9. Example images from PASCAL VOC object detection datasets [1].

3.3.2. SUN RGB-D Dataset

SUN RGB-D dataset [2] contains RGB-D images from SUN3D [40], Berkeley B3DO [41] and NYU depth v2 [42]. It is one of the largest indoor object detection RGB-D dataset with its 10,335 images. It has more than 50 object classes but only 19 of them has enough instances to do proper learning. In the dataset's paper, Song *et al.* refers these 19 classes as "19 popular object categories" and the baseline scores are published using only these 19 classes. These 19 popular object categories are bathtub, bed, bookshelf, box, chair, counter, desk, door, dresser, garbage bin, lamp, monitor, night stand, pillow, sink, sofa, table, tv, and toilet. Some example images from this dataset are shown in Figure 3.10.

The dataset includes RGB-D images captured with 4 different depth cameras (Microsoft Kinect v1 [3], Microsoft Kinect v2 [43], ASUS Xtion and Intel RealSense [44]) in different places, that is why it is considered a challenging dataset.

The most recent state-of-the-art mAP score for 2D object detection on the SUN RGB-D dataset is 35.20% [29].

Figure 3.10. Example images from the SUN RGB-D dataset [2].

4. METHODOLOGY AND IMPLEMENTATION

4.1. Object Detection

Our aim was to extend a state-of-the-art RGB object detector into a better performing RGB-D object detector by the help of depth data. We did not directly used a previously trained RGB-D object detector because the most successful ones cannot run in real-time which is our most important constraint in our project. We chose two of the best performing RGB object detection networks F-R-CNN and SSD. The differences of the network implementations forced us to try different approaches when modifying the networks to process the extra depth info. We will explain the implementation details in two subsections (Subsection 4.1.1 and Subsection 4.1.2). After successfully modifying the two networks we also wanted to explore a new way of preprocessing type which we came up with. The details of this preprocessing technique is explained thoroughly in Subsection 4.1.3.

4.1.1. Faster R-CNN and Depth

As shown in Figure 4.1 in the F-R-CNN method images are fed into a DNN which consists of convolutional layers at the beginning which can be thought of the CNN part of the whole network. After these convolutional layers produce convolutional feature maps from an image, they are first fed into the Region Proposal Network (RPN) to propose object bounding boxes. Then these proposed object bounding boxes and the convolutional feature maps are passed through three FC layers followed by a softmax layer and a bounding-box regressor. The output is the 2D bounding boxes with object class confidence scores. To modify this network to process an extra input depth image and learn from it, we followed two different approaches:

(i) input fusion, and
(ii) middle fusion.

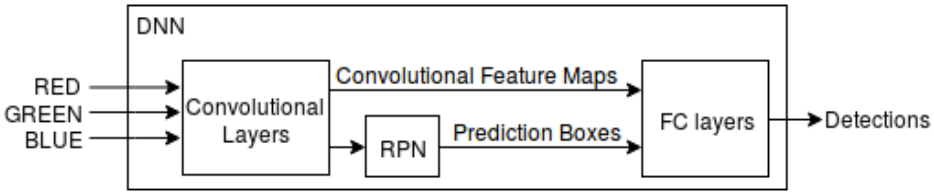

Figure 4.1. Faster R-CNN overall structure

4.1.1.1. Input Fusion. In this first method, we fuse the depth data with RGB before they are fed into the network. Since the network was built and pretrained to get 3-channel inputs we combine green and blue channels into one channel by taking their pixel-wise average. We put the single channel depth input into the remaining channel. In the end our fused input has one red channel, one green-blue combined channel and one depth channel (R[GB]-D). We leave the network untouched other than modifying the inputs. Figure 4.2 shows the visualization of the network taking this modified input.

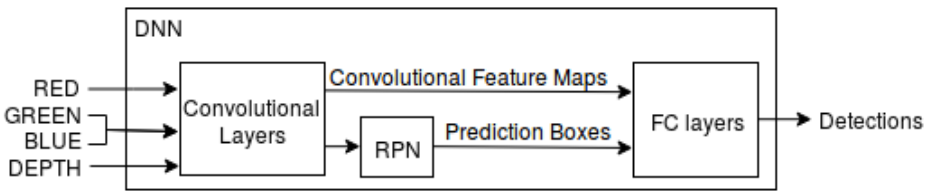

Figure 4.2. Faster R-CNN with input fusion

4.1.1.2. Middle Fusion. We copy the Faster R-CNN's convolutional layers until the RPN and create a twin network with convolutional layers which takes depth input. After the convolutional layers separately calculate convolutional feature maps, we send them to the RPN together. That way we included the convolutional feature maps learned from depth images into the Faster R-CNN network. We call this new network with middle fusion, Faster R-CNN and Depth (F-R-CNN-D). Figure 4.3 shows the visualization of this modified network.

4.1.2. SSD and Depth

After some attempts to improve Faster R-CNN network with depth input, we decided to experiment on a more recent network, SSD, which is faster than Faster R-CNN and it got state-of-the-art results on VOC datasets. SSD is also publicly available, and it was straight-

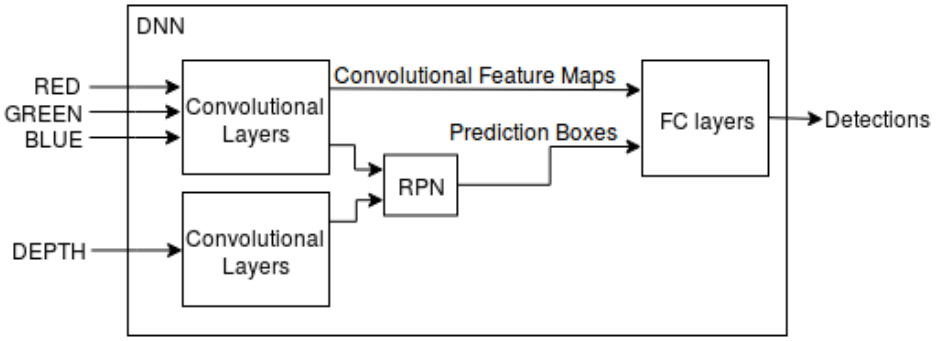

Figure 4.3. Faster R-CNN with middle fusion

forward setting up the libraries in a few hours. We began by fine-tuning the pretrained SSD network on the SUN RGB-D dataset using only the RGB images.

For creating the SSD with Depth (SSD-D) network we followed a similar implementation strategy with the middle fusion technique we used for F-R-CNN-D. We keep the pretrained SSD weights when fine-tuning the network with jet-colored depth images. Even though the original network was pretrained with the RGB images, using jet-coloring allows us to transfer the knowledge into our new network which takes depth input. Jet-coloring is done by mapping single channel depth image which includes only the distances to the points, into three channel depth image. Jet-colored depth image has colors depending on the distances (blue color being the closest and red color being the farthest).

Figure 4.4 shows the jet-color mapping scale and Figure 4.5 bottom two images show a single channel depth image and its jet-colored version. The darkest blue regions are caused from errors in the depth image. Since depth cameras use infrared sensors, some materials or distances are harder or sometimes even impossible to be sensed.

Figure 4.4. Jet-coloring from closest to farthest (blue to red)

Our experiments with F-R-CNN with middle fusion and input fusion show that we need to find other ways of combining depth info into an RGB network. That is why we try two other approaches for combining SSD-D with the original network:

(i) decision level fusion, and

(ii) detection level fusion.

4.1.2.1. Decision Level Fusion. The first method we implemented was combining the detection boxes of SSD-D and the original SSD network (SSD-RGB) by taking weighted average of the detection box offsets that were regressed from the same default box, and their confidence scores. For each default box SSD network outputs location offsets from the default box and the confidence scores for each object classes.

4.1.2.2. Detection Level Fusion. The second method was combining the detection boxes at the NMS level. This method is safer because it will be choosing the high confident detections from the two fused networks.

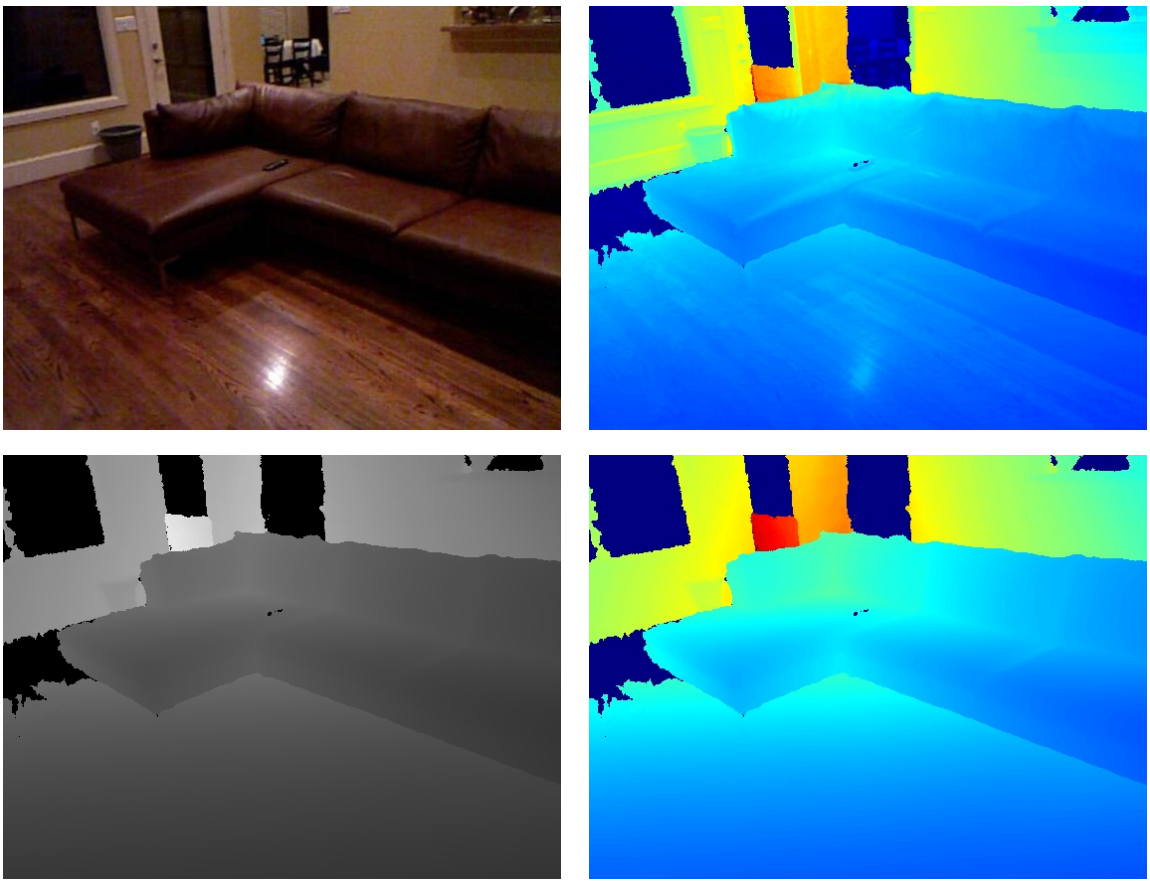

Figure 4.5. RGB image at the top left, jet-colored RGB overlaid depth image at the top right, single channel depth image at the bottom left, and jet-colored depth image at the bottom right

4.1.3. Overlaying RGB over Depth

Depth images taken with most of the commercially available cameras can have lots of undefined regions where the cameras cannot perceive. We wanted to create a fast pipeline where these corrupted regions can be healed by the help of the RGB images. For each RGB and depth image pairs, we first turn the RGB image into gray scale, normalize and shift the values into $[0.6, 1.4]$ interval. Before we multiply each depth pixel value with the corresponding pixel value of this normalized and shifted gray scale image, we set the undefined regions' value to the mean value of the depth image. After multiplying and overlaying the RGB image over the depth image we subtract the mean value from the undefined regions. By doing this we embed the RGB values into the undefined depth regions. Other than embedding some information in those regions, we create a depth image with the texture of the objects and surroundings, like the braille alphabet. These RGB overlaid images also include reflectivity and brightness of materials in the image as opposed to depth image not including anything except the distances from the camera to the pixels. An example image and its depth image's RGB overlaid jet-colored image is shown in Figure 4.5. Top right image is the image that is created by using this overlaying technique we came up with.

4.2. Object Tracking

Our environment consists of mostly the same objects but transformed slightly in consecutive frames. Therefore applying tracking over detected boxes can be used either to boost our detection scores or to let us run object detection network less (once in two frames or so). We focused on boosting the detection scores by using tracking as a correction mechanism.

Our method starts with a simple tracking algorithm, median flow (MF), which is available in the Open Source Computer Vision Library (OpenCV) [45]. After an object detection network predicts object bounding boxes inside a frame, we apply median flow tracking to the next frame using the previous bounding boxes. For this next frame we also run the object detection network and gather new predictions. We combine the tracked boxes from the previous frame with the newly detected boxes by doing a matching between them using intersection over union (IoU). We boost the confidence scores of the matched detection boxes

and lower the scores of the others by half. In the next frame we track both the previous frame's detection boxes and the previously tracked boxes. We consider the detection box with the same object class label of a tracked box with the maximum IoU (if above 0.7) the same objects. We increase the confidence score of the detection box by half of the tracked detection from the last frame's confidence score, and remove the tracked box since we longer need it because the object detection network detected the same object again and the matched tracked box boosted its confidence score. That way we gradually increase the confidence scores of successively tracked and detected prediction boxes. We also halve the confidence scores of successively tracked but rarely detected prediction boxes.

Adding MF tracking on top of F-R-CNN detection boxes with the strategy explained above only increases or decreases the confidence scores depending on the last frame's detection boxes and the current frame's detection boxes. We increase the memory and the robustness of this method by applying moving average over more than 2 frames. We test the performance of our system using the MF tracking and moving average method with 7 different window size (Table 5.7).

4.3. RGB-D Semantic Mapping with Objects

The main goal of our study is to create semantic RGB-D maps of rooms or whole houses for our robots. Instead of focusing on creating a real-time 3D mapping system, we used a high performing open source RTAB-Map library [35]. We modified it and embedded our system (Figure 4.6) in it so that we can create useful RGB-D semantic maps. We aim to use these RGB-D semantic maps when our robot interacts with users.

The biggest problem we had to overcome was converting 2D object bounding boxes, detected by our object detector network SSD-D, into pixel-wise labeling of object classes. SSD-D outputs detection boxes and their confidence scores between 0 and 1. For example a box can have a confidence score of 0.96 belonging to bed class. Although SSD-D does not provide any other scores for any other classes for that box, it can find another box intersecting with that one which has a different object class label and confidence score. We claim that we

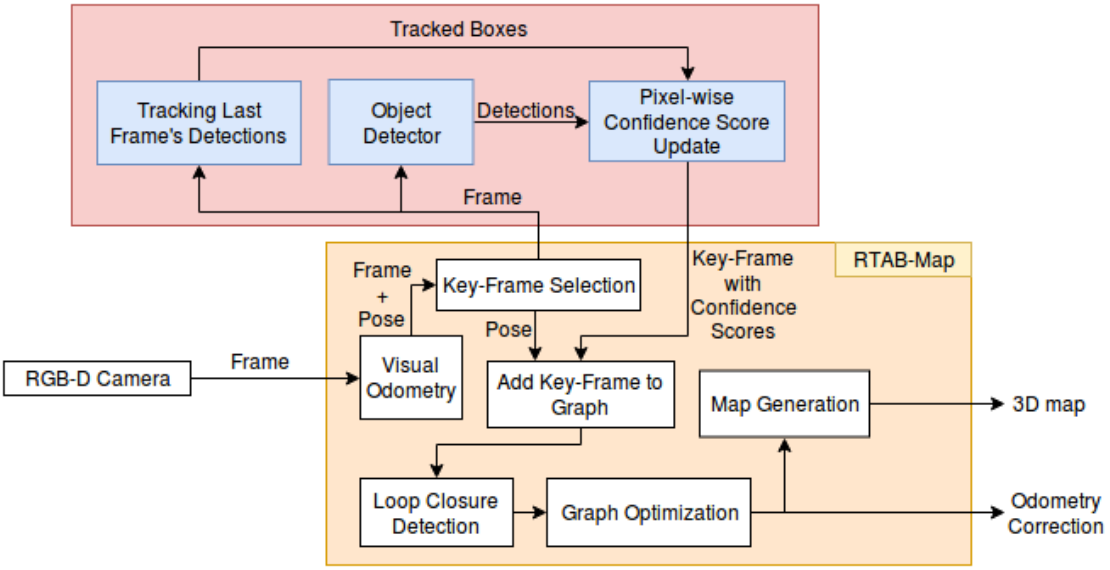

Figure 4.6. Flow chart of our pipeline with RTAB-Map.

can create an update rule to convert these 2D object bounding boxes and confidence scores into pixel-wise object class probabilities.

Object detection and object tracking parts of the system are explained in the previous sections in detail. After they produce tracked and predicted object boxes our pixel-wise confidence update algorithm maps these boxes into depth pixels (for each pixel we added a confidence array for the object classes which starts from the uniform distribution). We multiply the confidence scores of the object boxes with the pixel-wise confidence scores. Since each box has only one class confidence score, the remaining part is distributed among the other classes. After the confidence array of a pixel is updated, it is then normalized to keep the sum of the confidence scores equal to 1. The object detector does not provide any object boxes for the background class, and this is an important drawback of this updating method. If we keep updating all the other classes then eventually the confidence score of a pixel being background will converge to 0. We prevented that by adding an update rule for the non-detected pixels in the image which increases for these pixels the confidence score of the background class ten percent of their current background confidence scores.

4.4. Object Detection Video Dataset

There are no publicly available indoor video datasets with 2D object bounding boxes. Because of this we collected and labeled a small one consisting of a video which is around 2000 frames. We applied our detection box tracking approach with the F-R-CNN-D object detection network.

There are 11 different object classes in the dataset: bed, box, chair, desk, door, lamp, monitor, night stand, pillow, sofa and table. We labeled the objects' locations by enclosing them in the tightest 2D bounding boxes in only one orientation as in PASCAL VOC object detection datasets. Each bounding box rectangles have minimum (x, y) coordinates for the upper left corner, and maximum (x, y) coordinates for the lower right corner of the rectangle.

5. EXPERIMENTS AND RESULTS

5.1. SUN RGB-D Object Detection

5.1.1. F-R-CNN-D

We compare different settings of our system's performance on the SUN RGB-D dataset [2], with the network (RGB-D R-CNN [46]) the dataset publishers evaluated the baseline performance with. Figure 5.1 shows the results from the experiments we ran trying different learning rates (lr) for the original Faster R-CNN (F-R-CNN) network taking RGB input and our middle fusion version of F-R-CNN network processing RGB-D input. We also included the results from our input level fusion version of F-R-CNN network processing R[GB]-D input. We selected the 19 popular object classes used in the original paper of the dataset. In Table 5.1 a detailed analysis of the different networks are shown, including class-based performances and the overall mean average precision (mAP) in the last line.

Overall we see that fine-tuning F-R-CNN network without any modifications and with only RGB images on the SUN RGB-D dataset with the learning rate of 0.001 performs significantly better than the baseline approach R-CNN-D [29], which uses both RGB and depth images. We can see that object class-based performances also differ a lot in favor of F-R-CNN except two object classes: table and bed. F-R-CNN performs 0.11% better for the table class which is not a significant enough difference. Interestingly the R-CNN-D network has 3% better average precision (AP)for the bed class, while F-R-CNN has 2.5% to 40% better APs for all the remaining object classes.

Comparing the early fusion F-R-CNN R[GB]-D with the original F-R-CNN RGB network, it is clear that this method lost all the pretrained weights when we change the green channel into average of green-blue and blue channel into depth channel. Since we only fine-tune the network we heavily depend on the pretrained weights. Therefore this network may perform better that the original if it can be trained from scratch on a much larger RGB-D

dataset. It may still perform worse though because combining by averaging green and blue channels might lose a lot of valuable information in those channels stored separately.

Fine-tuning he original network with a higher learning rate, 0.002, we see a little decrease in overall performance compared to the default learning rate of 0.001, which we guess it is caused by small overfitting to the training set.

F-R-CNN RGB-D with the middle fusion and with a learning rate of 0.001 we see slight improvement in the total mAP compared to the F-R-CNN RGB network. This improvement is not as big as we expected. Adding the depth images should in theory increase the information learned about the images and therefore should result in a significant improvement. Increasing the learning rate to 0.002 with F-R-CNN RGB-D to inspect if we can train a better performing network with RGB-D input by changing the pretrained weights more. Again the network performs slightly (0.09%) better than the previous setting. Trying larger or smaller learning rates does not change the performance any more significantly. We believe that the pretrained weights of the RPN on only RGB images' convolutional feature maps cannot be fine-tuned to handle convolutional feature maps from both RGB and depth images with such a small dataset.

5.1.2. SSD and Depth

Our experiments with SSD network divides into two categories. The first part consists of training separate networks with different input types (RGB, depth, or RGB overlaid depth). The second part includes our experiments on fusing each pair of these separate networks into one. SSD-D network takes only raw depth data and SSD-D+ takes only our RGB overlaid depth data.

In Table 5.2, SSD-RGB is the original network taking only RGB input. Its mAP score was 52.81% on the SUN RGB-D dataset with 19 popular object categories. To check the effect of the randomness in the mini-batch stochastic gradient descent algorithm we fine-tuned another network taking only RGB input, SSD-RGB'. Since its mAP score was 52.73%, the randomness did not have a significant effect as expected, because the SUN RGB-D dataset

Table 5.1. F-R-CNN object detection results on 19 classes of SUN RGB-D

Object Classes	R-CNN-D [46]	F-R-CNN R[GB]-D	F-R-CNN RGB		F-R-CNN RGB-D	
			lr=0.001	lr=0.002	lr=0.001	lr=0.002
bathtub	49.56	8.06	51.99	48.41	52.56*	47.16
bed	75.97*	60.96	71.73	72.90	72.60	72.22
bookshelf	34.99	28.48	44.32	44.21	45.11	45.62*
box	5.78	11.13	14.38	17.30	17.59	17.91*
chair	41.22	47.23	55.27	55.64	55.74	55.93*
counter	8.08	40.23	49.37	50.25*	48.52	49.65
desk	16.55	14.78	26.67*	24.95	25.43	26.47
door	4.17	35.63	54.66*	52.41	53.08	52.13
dresser	31.38	27.70	37.93*	37.04	36.82	36.73
garbage bin	46.83	35.57	53.84	52.44	54.16*	51.95
lamp	21.98	43.63	53.91*	52.39	53.48	53.83
monitor	10.77	18.54	43.31	45.28	45.44*	44.43
night stand	37.17	29.20	52.91	51.81	53.01	54.67*
pillow	16.5	31.19	46.27*	46.19	44.80	44.61
sink	41.92	41.65	60.62	58.89	59.85	63.47*
sofa	41.2	33.66	46.32	48.38	48.31	51.49*
table	43.02	33.86	43.13	44.20	43.45	44.30*
tv	32.92	13.60	38.08*	36.07	35.22	33.45
toilet	69.84	71.62	79.65	79.71	83.52	84.81*
mAP	35.20	32.99	48.65	48.34	48.88	48.97*

has challenging and wide variety of object class instances. This decreases the effect of the randomness because each mini-batch (32 images) include very different types of object instances and therefore randomness cannot include any significant bias into the system. SSD-D is the new network that we trained with only jet-colored raw depth images. Using only RGB overlaid jet-colored depth images (SSD-D+) yielded around 1% better performance than SSD-D. These results show that transferring pretrained weights of the original network with different input type can still perform fairly well but if we can train this network on only depth input with more than $5,000$ images or maybe millions of images than we might expect a better performance for both SSD-D and SSD-D+ network.

Table 5.2. SSD object detection results using single networks with different input types on 19 classes of SUN RGB-D

Network	mAP
SSD-RGB	52.81*
SSD-RGB'	52.73
SSD-D+	48.05
SSD-D	47.15

After training with only one type of input we fused these networks using two different approaches. The first approach was learning a class-specific relation between the two networks, decision level fusion. Unfortunately this technique was not suitable for the convolution-only structure of the SSD network. In the end, each of these convolutional layers' output was a vector consisting of the offsets from their related default box's coordinates and object class confidence scores. The problem of combining these outputs of two different networks in a weighted average manner was the offsets of the same default boxes could be very different than each other in size and/or direction. We left this strategy after combining SSD-RGB and SSD-D+ got 52.04% mAP. We fine-tuned the combined network long enough but the network always performed worse than the SSD-RGB network. The second and the straight-forward fusion was detection level fusion. After the two networks separately finds detection boxes, we applied non-maximum suppression (NMS) to all of the boxes. This way we increased the SSD-RGB network's performance by almost 2% with fusing SSD-RGB

and SSD-D networks. SSD-RGB and SSD-D+ fusion had around 1.5% gain compared to SSD-RGB network. The cause of 0.5% lower mAP between SSD-RGB with SSD-D fusion and SSD-RGB with SSD-D+ fusion can be the similarity of the knowledge learned by SSD-RGB and SSD-D+ because they both include RGB images, although SSD-D+ includes it in the input fusion level. When these two networks were fused together the overlapping knowledge decreased the overall performance. On the other hand using raw depth images (SSD-D) yields less overlapping information with SSD-RGB, therefore the fusion of these two had 0.5% extra boost.

We wanted to test the effect of increasing the number of detection boxes, because when we fuse two networks with detection level fusion we increase the number of detection boxes before non-maximum suppression. We fused two separately trained SSD networks taking RGB inputs. The resulting mAP score was 53.86% which is 1% better than using only one SSD-RGB network. This shows that increasing the number of object detection boxes increases performance. SSD-D and SSD-RGB fusion still outperforms two SSD-RGB fusion by 1%. The pure effect of fusing SSD-D with SSD-RGB is around 1% without the increase caused by the number of detection boxes.

The final fusion we tested was between SSD-D and SSD-D+ networks. The results were promising with 49.96% mAP. This fusion performed almost 2% better than each of the fused networks separately. This shows that our encoding technique included significant information from RGB images on top of raw depth images.

Table 5.3. SSD object detection results on 19 classes of SUN RGB-D with network fusions.

Fusion Level	Networks	mAP
Decision	SSD-RGB and SSD-D	52.04
Detection	SSD-D and SSD-D+	49.96
Detection	SSD-RGB and SSD-RGB'	53.86
Detection	SSD-RGB and SSD-D+	54.34
Detection	SSD-RGB and SSD-D	54.80*

Finally in Table 5.4 we compare the performance of the different approaches we implemented on two different networks (F-R-CNN and SSD) with the only published 2D object detection results (R-CNN-D [29]) on the SUN RGB-D dataset. Since most of the publications using this dataset focuses on 3D object detection, there is only one published results for the 2D object detection. For this reason we compared our results with the original F-R-CNN and the original SSD-RGB networks we trained on this dataset. Although F-R-CNN and SSD-RGB use only RGB images, they both dramatically outperformed R-CNN-D [46], which was built on the first R-CNN model [16]. The SSD-D+ network, using only our RGB overlaid depth images, reached to a similar performance as the F-R-CNN network. The middle fused F-R-CNN-D network couldn't make a significantly different performance than the F-R-CNN network. Although the original SSD-RGB network got nearly 5% better mAP than the F-R-CNN network, our implementation of detection level fused SSD-RGB and SSD-D network got approximately 7% higher mAP.

Table 5.4. SSD object detection results on 19 classes of SUN RGB-D

Networks	mAP
R-CNN-D [46]	35.20
SSD-D+	48.05
F-R-CNN	48.34
F-R-CNN-D	48.97
SSD-RGB	52.81
SSD-RGB and SSD-D	54.80*

On 2 NVIDIA GTX 1080Ti graphics cards, the training time of any single SSD network took around 1.5 days, and testing runs in 42 FPS.

5.2. Indoor Object Detection Video Dataset

5.2.1. Faster R-CNN and Depth

Our experiments began by comparing different threshold levels for applying non-maximum suppression (NMS) on our small dataset consisting of 11 classes (Table 5.5). The default NMS threshold used in the Faster R-CNN paper is 0.3 for VOC datasets. Lowering this threshold results in less number of final detection boxes. Since mAP for NMS thresholds 0.3, 0.1, and 0.05 are very similar we can see that 0.3 threshold is also suitable for our dataset. We should underline the fact that we did not use any images from this dataset to fine-tune our network. We fine-tuned our F-R-CNN-D network on the training set of SUN RGB-D. Our dataset includes new instances for the 11 classes. F-R-CNN-D alone reached 37.48% mAP and our aim was improving this by implementing a simple tracking algorithm which we will further use in our final system.

Table 5.5. F-R-CNN-D object detection results on 11 classes of our dataset with different NMS thresholds

NMS Threshold	0.3	0.1	0.05
mAP	37.48*	37.1	36.81

5.2.2. F-R-CNN-D with Tracking

We created a tracking method of the detection boxes coming from the F-R-CNN-D network by implementing median flow (MF) tracking and an update rule of the detected and tracked boxes. After we implemented tracking the resulting object boxes increased, that is why we tested applying another NMS after tracking is applied between each consecutive frames. Table 5.6 shows that without any NMS after tracking we got better results than applying NMS with 0.3 threshold. This shows that our update rule takes care of the wrongly tracked boxes in few frames and therefore not applying NMS allows some of the well tracked boxes to survive. Overall implementing MF tracking and our update rules increased the mAP by 2% on our dataset.

Table 5.6. F-R-CNN-D object detection results on our 11-class dataset using MF tracking on predicted object boxes with different before and after NMS thresholds

NMS Threshold		mAP
After tracking	Before tracking	
-	0.3	39.32
	0.1	39.39
	0.05	39.43*
0.3	0.3	38.99
	0.1	38.86
	0.05	38.77

Our experiments continue without applying NMS after tracking because the empirical results show that our update procedure after tracking interferes with NMS. Table 5.7 shows the results of applying a smoothing algorithm (moving average) over the confidence scores of the tracked detection boxes. We tested applying moving average over different number of frames (W). Choosing a number between 10 and 25 seems to perform well enough and similar to each other. The best performing result was applying moving overage over each 15-frame windows, which increased the total mAP around 2.5% more than applying only MF tracking.

Table 5.7. Applying moving average over W frames using the scores of predicted bounding boxes and median flow tracking (No NMS after tracking)

NMS Threshold \W	5	10	15	20	25	50	100
0.05	41.07	41.47	41.75	41.52	41.47	41.14	39.83
0.3	41.07	41.49	41.89*	41.64	41.24	40.86	38.35

Overall class-based AP analysis of using only F-R-CNN-D object detection, adding tracking with our update rules, and applying moving average over tracked results are shown in Table 5.8. We see that although applying only tracking method increases most of the

scores, the real gain comes from smoothing the confidence scores of the tracked boxes which implements long-term memory into the system.

Table 5.8. F-R-CNN-D object detection results on 11 classes of our dataset with Median Flow tracking and averaging over confidence scores (0.3 NMS threshold before tracking and 15-frames moving average are used)

Object Classes	No tracking	MF Tracking	MF + Moving Average
bed	47.12	48.25	57.57*
box	0.02	0.11*	0.08
chair	36.16	38.71	40.34*
desk	20.93	23.58	26.48*
door	10.57	13.58	14.42*
lamp	66.24	67.83	68.88*
monitor	67.95	67.59	71.68*
night stand	26.32	32.48	38.23*
pillow	73.97	74.45	74.96*
sofa	60.84	63.22	64.86*
table	2.21	2.67	3.32*
mAP	37.48	39.32	41.89*

5.2.3. 3D Semantic Mapping

In this part of our experiments we finished our pipeline by integrating the previous two modules we created and tested on different datasets into our final module, 3D mapping. For the first module we chose to use the SSD-RGB fused with SSD-D network which we fine-tuned on the SUN RGB-D dataset as our object detector because it had better performance on the SUN RGB-D dataset than any other object detection networks to the best of our knowledge.

We captured videos from 2 different room. Both of the videos were captured under poor lighting conditions because our robot will be working mostly under such conditions. In the first room there are 1 bed, 1 sofa, 1 night stand, 2 chairs (1 office chair and 1 regular chair), 1 monitor, 5 pillows (4 on the sofa, 1 on the floor), 1 desk, and 1 small table which are the objects that we want to detect with our detector, SSD. There is also a digital piano in the room which looks like a desk or a table to our object detector, since it is not trained with pianos. In Figure 5.1 the 3D map of the first room is shown. This map was created by only using RTAB-Map without our modifications. Figure 5.2 shows the 3D semantic map of the same room which was the output of our object detection, tracking and pixel-wise confidence score updating pipeline in addition to RTAB-Map. This color map was created with a maximum distance threshold of restraining this system from trying to apply pixel-wise confidence mapping to further objects. As can be seen from this color map with maximum threshold:

(i) The bed is only partially detected since we apply a distance thresholding.
(ii) The piano is detected as table and also chair. There is also a chair in front of the piano but it is very hard to detect exactly because of its dark color mixing with the background. That is why it was detected poorly and the piano was marked as chair.
(iii) The sofa and the 2 pillows are detected very well.
(iv) The small table is also detected well.
(v) The office chair has some artifacts behind it which can be cleared by ground detection and extraction.
(vi) The desk is detected both as a desk and a table.
(vii) The monitor on the desk is correctly detected but most of it is not colored since the depth image had some missing pixels of its reflecting screen.
(viii) The missing detections are the 2 pillows on the sofa, 1 pillow on the ground, and the night stand.

Another color map from the same room without any thresholding is shown in Figure 5.3 which has the same color mapping as Figure 5.2. In the creation of this map when labeling the sofa and the office chair the system labeled lots of background pixels.

Overall in the first room, the detection system performed well enough to let this 3D map to be used to locate the large objects which can be used as landmarks. The missing detection of the night stand is the biggest failure but still the robot can recover from it by communicating with the user. User can tell the robot that night stand is near the bed and the robot will approach near the bed. As it approaches it will recognize more areas of the bed and explore its surroundings. Eventually our system should locate the night stand.

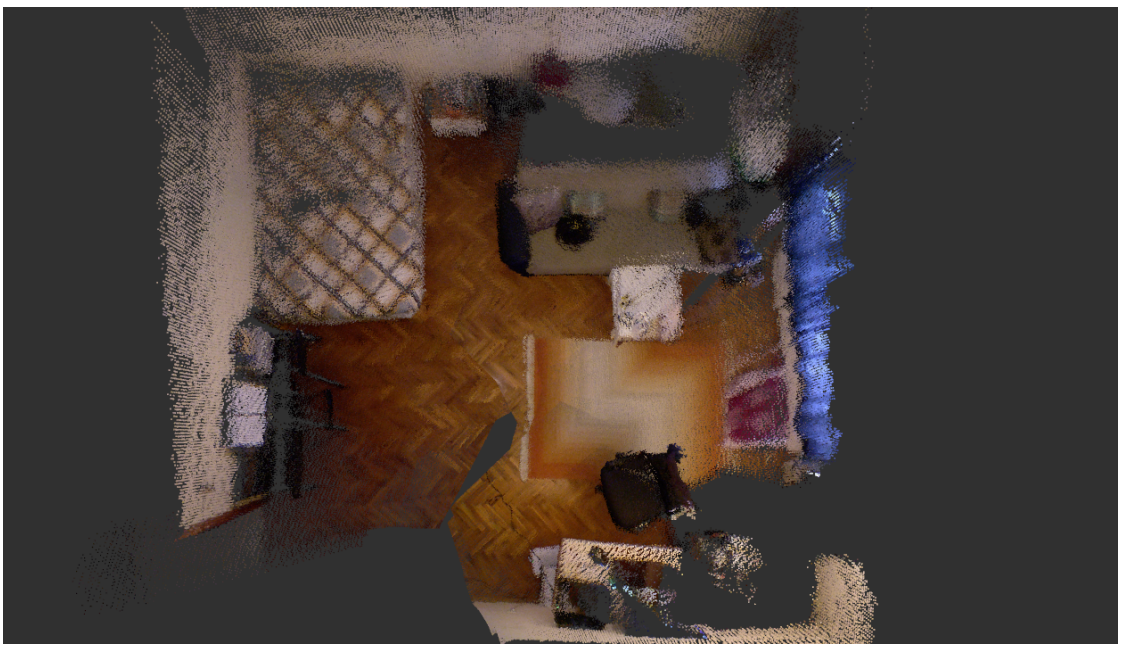

Figure 5.1. 3D map of the first room.

Figure 5.4 shows another room with the following objects: 1 bed, 2 chairs, 2 desks, 1 sofa, 1 pillow. Figure 5.5 which has the same color mapping as Figure 5.2, shows the color map of this room after running our system in it. This room is more problematic because the objects are much closer and the robot has less space to move compared to the first room. Although all the objects in the room were detected correctly, most of them created unwanted artifacts. The two chairs in front of the two desks made the system to color almost the entire desks as chairs. The walls and the floor have some incorrect coloring from the sofa and the bed. The incorrect coloring on the floor can be erased by a quick ground extraction algorithm, and the coloring on the walls can be cleared by room layout detection. This is not hard to implement because these 3D maps borders are mostly walls. The biggest failure of our system in this room was the table detection as the left side of the sofa. This is probably caused by the laptop standing on the sofa, as our system had probably trained mostly with

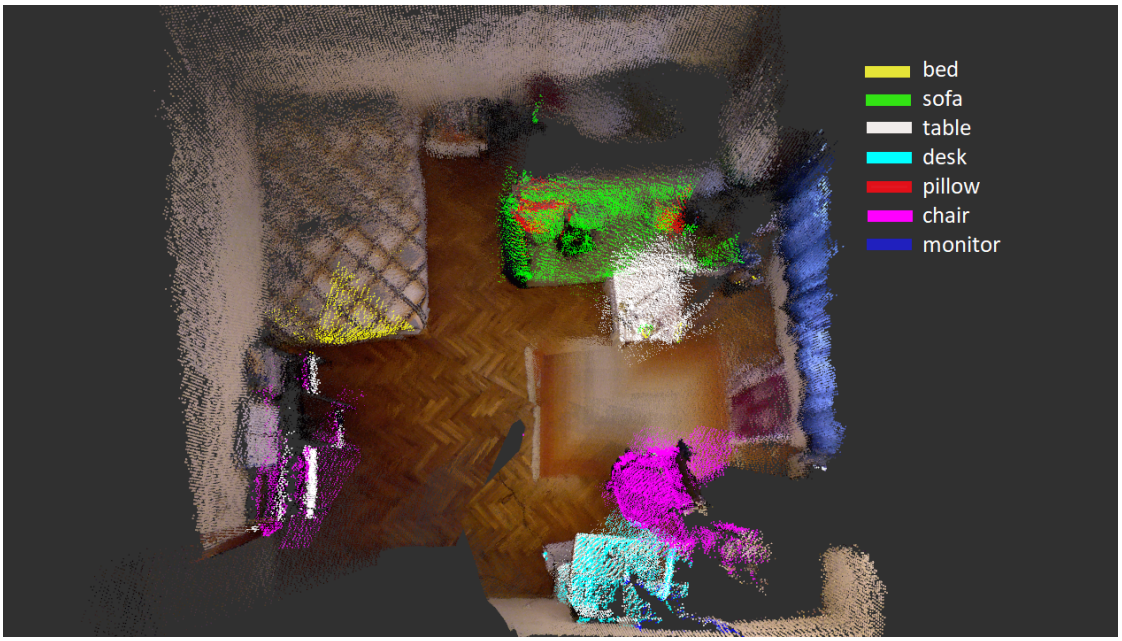

Figure 5.2. 3D semantic map of the first room with maximum thresholding.

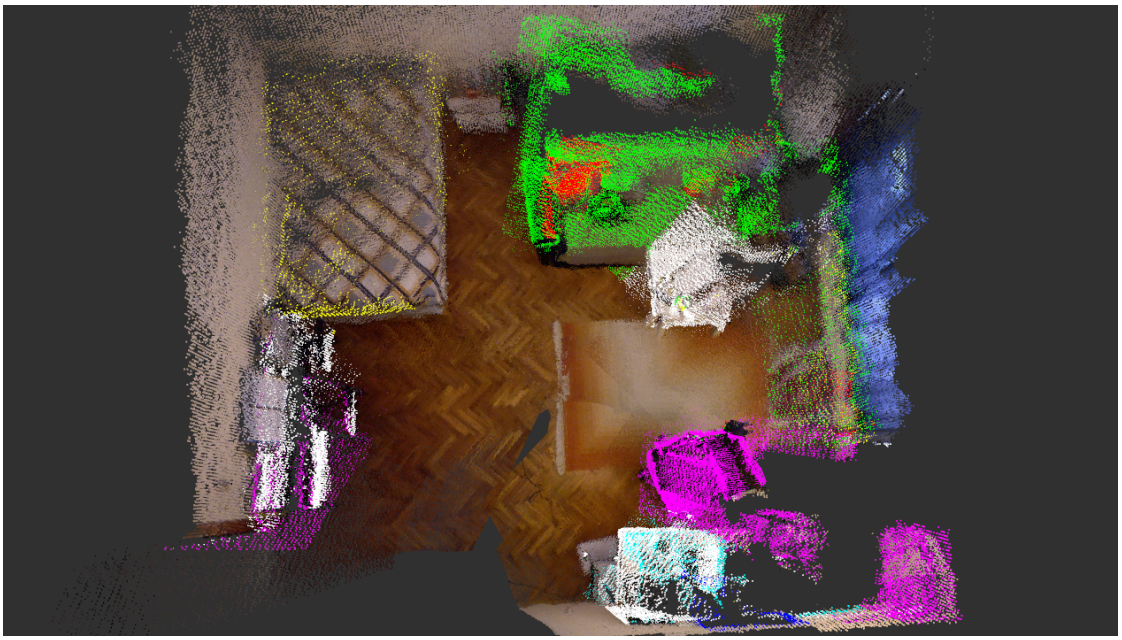

Figure 5.3. 3D semantic map of the first room without maximum thresholding.

table images having similar objects to laptops and sofa images without any objects except pillows. This wrong table detection can be recovered by interacting with the user.

Overall the second room shows that our system fails more in smaller cluttered rooms. The 3D map created in the second room can also be used by the robot but it won't be as useful as the first room's 3D map. Labeling two very close objects as object pairs can help in those situations, such as labeling chair desk pairs together.

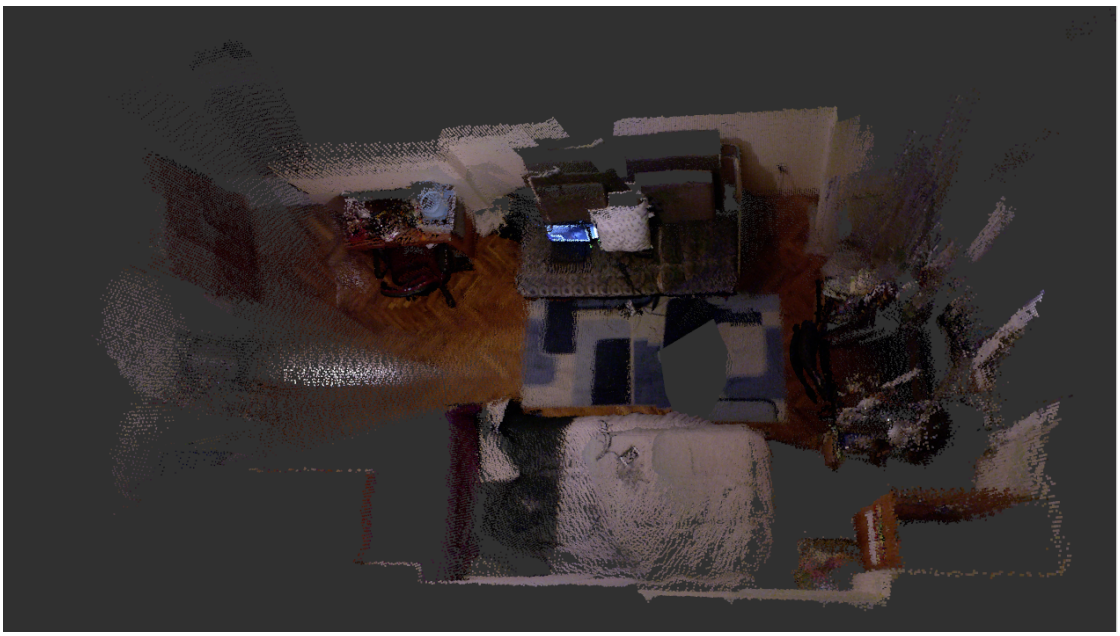

Figure 5.4. 3D map of the second room.

RTAB-Map system alone runs around 30 FPS (mostly 31 but sometimes falls to 25 when it does loop closure). Our object detection and update addition into the system does not effect this speed much because we only run our object detector for the key frames ($2-3$ key frames per second), and our object detector can run above 40 FPS on an NVIDIA GTX 1070 graphics cards.

More object detection results from the first room and 3D semantic maps created with different object detection networks can be found in Appendix A.

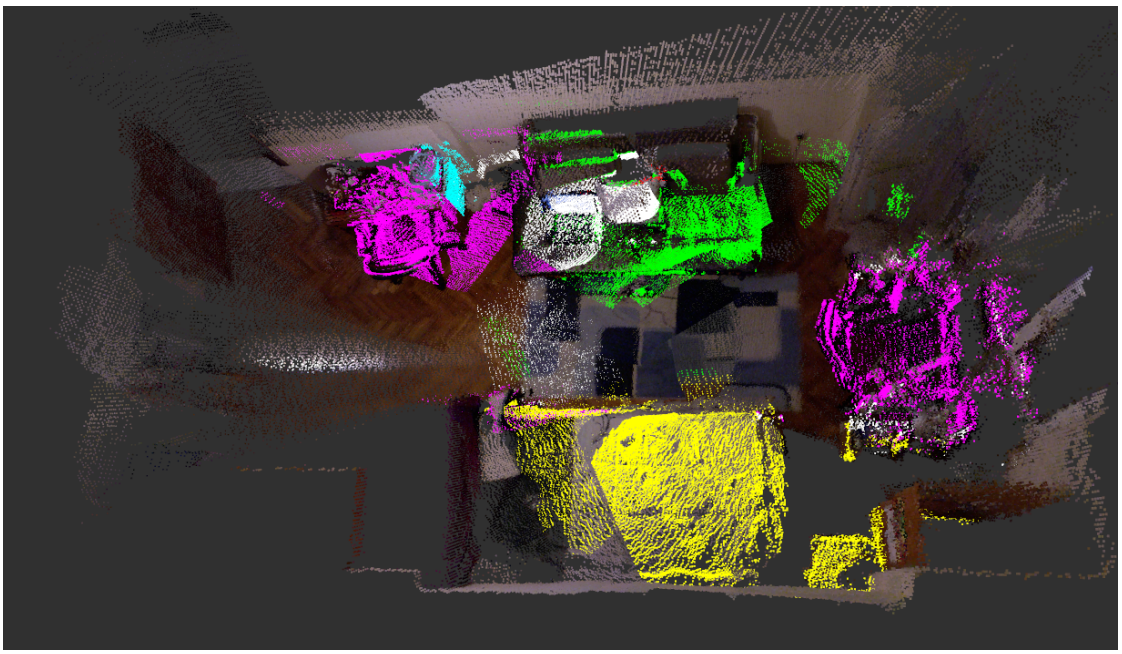

Figure 5.5. 3D semantic map of the second room.

6. CONCLUSION

Our main contribution was proving that an image segmentation system is not a must when creating 3D semantic maps. The proposed approach successfully creates usable 3D semantic maps with a 2D bounding box object detector.

SSD is more robust to transfer learning compared to Faster R-CNN. This was the reason of the differences in their network structures. Faster R-CNN uses RPN (fully-connected layers on the convolutional feature maps), which fails with feature maps coming from a different modality. On the other hand SSD uses only convolutional layers which are more robust to changes in the feature maps. That is why implementing a twin network for processing depth info with SSD had much significant improvement compared to Faster R-CNN. Although our detection level fusion with SSD object detector performed fairly well on the SUN RGB-D dataset, it may be improved further with a well adjusted fusion technique. This is left as a future work because the modification requires to change the fundamental structure of SSD by combining the two networks in multiple layers. This new fusion also needs to be trained on a much larger dataset than SUN RGB-D in order it to perform significantly better than the current SSD.

In [28] it is stated that there is a trade off between the speed of SSD and the performance for small objects comparing with other detectors. Since SUN R-GBD dataset's 19 popular object categories do not include very small objects, using SSD is advantageous in terms of both speed and accuracy. Our main goal also included detecting large objects and keeping them in memory as 3D semantic maps that is why choosing SSD performed very well. We propose different ways to use these 3D semantic maps for home assistant robots:

(i) Robot can improve its localization by using strongly detected objects as landmarks.
(ii) Robot can navigate safely by avoiding the paths closer to fragile objects such as tv, monitors or lamps detected and placed in our 3D semantic maps.

(iii) Robot can communicate with people by mentioning its knowledge about these detected objects and their locations. This will improve the interaction quality and make the people feel safer around the robot because of its awareness to the surroundings.

(iv) Robot can bring items around or on top of these large objects when a person wants it to do. Running a slower but more accurate object detector when the destination large object is reached can be the key to grasping smaller objects.

We leave improving the 3D semantic maps with different post-processing methods as a future work. These post-processing methods include passing 3D smoothing filters on the 3D map to take average of the confidence scores of nearby pixels. We believe this may decrease the artifacts since they are mostly surrounded by the pixels belonging to the background class. This may also boost the confidence scores of the detected pixels because sometimes there are some detected pixels buried under another layer of detected pixels because of the nature of applying loop closure. Averaging them together would increase the pixel confidences of the surrounding pixels. Another thing that can be added to this system is a correction system depending on the interaction with the people. Any wrong detections or missed detections can be recovered via smart interactions as explained in Subsection 5.2.3.

REFERENCES

1. Everingham, M., L. Van Gool, C. K. I. Williams, J. Winn and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*, http://host.robots.ox.ac.uk/pascal/VOC/voc2007/results/index.shtml, accessed at July 2018.

2. Song, S., S. P. Lichtenberg and J. Xiao, "SUN RGB-D: A RGB-D scene understanding benchmark suite.", *CVPR*, Vol. 5, p. 6, 2015.

3. Zhang, Z., "Microsoft Kinect Sensor and Its Effect", *IEEE MultiMedia*, Vol. 19, No. 2, pp. 4–10, Feb 2012.

4. Graichen, K., S. Hentzelt, A. Hildebrandt, N. Kärcher, N. Gaißert and E. Knubben, "Control design for a bionic kangaroo", *Control Engineering Practice*, Vol. 42, pp. 106–117, 2015.

5. Behbahani, S. B. and X. Tan, "A flexible passive joint for robotic fish pectoral fins: Design, dynamic modeling, and experimental results", *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 2832–2838, IEEE, 2014.

6. Hatakeyama, T. and H. Mochiyama, "Shooting manipulation inspired by chame-leon", *IEEE/ASME Transactions on Mechatronics*, Vol. 18, No. 2, pp. 527–535, 2013.

7. Duan, H., Y. Deng, X. Wang and F. Liu, "Biological eagle-eye-based visual imaging guidance simulation platform for unmanned flying vehicles", *IEEE Aerospace and Electronic Systems Magazine*, Vol. 28, No. 12, pp. 36–45, 2013.

8. Rosenblatt, F., "The perceptron: a probabilistic model for information storage and organization in the brain.", *Psychological review*, Vol. 65, No. 6, p. 386, 1958.

9. Alpaydin, E., *Introduction to Machine Learning*, MIT Press, Cambridge, MA, USA, 2014.

10. Chauvin, Y. and D. E. Rumelhart, *Backpropagation: theory, architectures, and applications*, Psychology Press, 1995.

11. Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, Vol. 115, No. 3, pp. 211–252, 2015.

12. Krizhevsky, A., I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger (Editors), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105, Curran Associates, Inc., 2012.

13. Dalal, N. and B. Triggs, "Histograms of oriented gradients for human detection", *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, Vol. 1, pp. 886–893, IEEE, 2005.

14. Wensley, J. H., L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak and C. B. Weinstock, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control", *Proceedings of the IEEE*, Vol. 66, No. 10, pp. 1240–1255, 1978.

15. Bay, H., T. Tuytelaars and L. Van Gool, "Surf: Speeded up robust features", *European conference on computer vision*, pp. 404–417, Springer, 2006.

16. Girshick, R., J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

17. Szegedy, C., A. Toshev and D. Erhan, “Deep Neural Networks for Object Detection”, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Q. Weinberger (Editors), *Advances in Neural Information Processing Systems 26*, pp. 2553–2561, Curran Associates, Inc., 2013.

18. Uijlings, J. R., K. E. Van De Sande, T. Gevers and A. W. Smeulders, “Selective search for object recognition”, *International journal of computer vision*, Vol. 104, No. 2, pp. 154–171, 2013.

19. Felzenszwalb, P. F., R. B. Girshick, D. McAllester and D. Ramanan, “Object detection with discriminatively trained part-based models”, *IEEE transactions on pattern analysis and machine intelligence*, Vol. 32, No. 9, pp. 1627–1645, 2010.

20. Erhan, D., C. Szegedy, A. Toshev and D. Anguelov, “Scalable object detection using deep neural networks”, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2147–2154, 2014.

21. He, K., X. Zhang, S. Ren and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, *european conference on computer vision*, pp. 346–361, Springer, 2014.

22. Girshick, R., “Fast R-CNN”, *International Conference on Computer Vision (ICCV)*, 2015.

23. Everingham, M., L. Van Gool, C. K. Williams, J. Winn and A. Zisserman, “The pascal visual object classes (voc) challenge”, *International journal of computer vision*, Vol. 88, No. 2, pp. 303–338, 2010.

24. Ren, S., K. He, R. Girshick and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 39, No. 6, pp. 1137–1149, June 2017.

25. Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the inception architecture for computer vision", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

26. Redmon, J., S. Divvala, R. Girshick and A. Farhadi, "You only look once: Unified, real-time object detection", *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

27. Sadeghi, M. A. and D. Forsyth, "30Hz Object Detection with DPM V5", D. Fleet, T. Pajdla, B. Schiele and T. Tuytelaars (Editors), *Computer Vision – ECCV 2014*, pp. 65–79, Springer International Publishing, Cham, 2014.

28. Liu, W., D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD: Single Shot MultiBox Detector", B. Leibe, J. Matas, N. Sebe and M. Welling (Editors), *Computer Vision – ECCV 2016*, pp. 21–37, Springer International Publishing, Cham, 2016.

29. Gupta, S., R. Girshick, P. Arbeláez and J. Malik, "Learning Rich Features from RGB-D Images for Object Detection and Segmentation", D. Fleet, T. Pajdla, B. Schiele and T. Tuytelaars (Editors), *Computer Vision – ECCV 2014*, pp. 345–360, Springer International Publishing, Cham, 2014.

30. Gupta, S., P. Arbeláez and J. Malik, "Perceptual Organization and Recognition of Indoor Scenes from RGB-D Images", *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 564–571, June 2013.

31. Song, S. and J. Xiao, "Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images", *CoRR*, Vol. abs/1511.02300, 2015, `http://arxiv.org/abs/1511.02300`.

32. Song, S. and J. Xiao, "Sliding Shapes for 3D Object Detection in Depth Images", D. Fleet, T. Pajdla, B. Schiele and T. Tuytelaars (Editors), *Computer Vision – ECCV 2014*, pp. 634–651, Springer International Publishing, Cham, 2014.

33. Dame, A., V. A. Prisacariu, C. Y. Ren and I. Reid, "Dense Reconstruction Using 3D Object Shape Priors", *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1288–1295, June 2013.

34. Sun, L., C. Zhao and R. Stolkin, "Weakly-supervised DCNN for RGB-D Object Recognition in Real-World Applications Which Lack Large-scale Annotated Training Data", *CoRR*, Vol. abs/1703.06370, 2017, `http://arxiv.org/abs/1703.06370`.

35. Thrun, S. and M. Montemerlo, "The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures", *The International Journal of Robotics Research*, Vol. 25, No. 5-6, pp. 403–429, 2006, `https://doi.org/10.1177/0278364906065387`.

36. Labbé, M. and F. Michaud, "Online global loop closure detection for large-scale multi-session graph-based SLAM", *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2661–2666, Sept 2014.

37. Sünderhauf, N., T. Pham, Y. Latif, M. Milford and I. D. Reid, "Meaningful Maps - Object-Oriented Semantic Mapping", *CoRR*, Vol. abs/1609.07849, 2016, `http://arxiv.org/abs/1609.07849`.

38. Zhao, C., L. Sun, P. Purkait and R. Stolkin, "Dense RGB-D semantic mapping with Pixel-Voxel neural network", *CoRR*, Vol. abs/1710.00132, 2017.

39. Hermans, A., G. Floros and B. Leibe, "Dense 3D semantic mapping of indoor scenes from RGB-D images", *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2631–2638, May 2014.

40. Xiao, J., A. Owens and A. Torralba, "Sun3d: A database of big spaces reconstructed using sfm and object labels", *Computer Vision (ICCV), 2013 IEEE International Conference on*, pp. 1625–1632, IEEE, 2013.

41. Janoch, A., S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko and T. Darrell, "A category-level 3d object dataset: Putting the kinect to work", *Consumer Depth Cameras for Computer Vision*, pp. 141–165, Springer, 2013.

42. Silberman, N., D. Hoiem, P. Kohli and R. Fergus, "Indoor segmentation and support inference from rgbd images", *European Conference on Computer Vision*, pp. 746–760, Springer, 2012.

43. Geerse, D. J., B. H. Coolen and M. Roerdink, "Kinematic Validation of a Multi-Kinect v2 Instrumented 10-Meter Walkway for Quantitative Gait Assessments", *PLOS ONE*, Vol. 10, No. 10, pp. 1–15, 10 2015, `https://doi.org/10.1371/journal.pone.0139913`.

44. Keselman, L., J. I. Woodfill, A. Grunnet-Jepsen and A. Bhowmik, "Intel RealSense Stereoscopic Depth Cameras", *CoRR*, Vol. abs/1705.05548, 2017, `http://arxiv.org/abs/1705.05548`.

45. Bradski, G., "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

46. Gupta, S., R. Girshick, P. Arbeláez and J. Malik, "Learning Rich Features from RGB-D Images for Object Detection and Segmentation", D. Fleet, T. Pajdla, B. Schiele and T. Tuytelaars (Editors), *Computer Vision – ECCV 2014*, pp. 345–360, Springer International Publishing, Cham, 2014.

47. Alpaydin, E., *Machine Learning: The New AI*, MIT Press, Cambridge, MA, USA, 2016.

APPENDIX A: INDOOR OBJECT DETECTION VIDEO DATASET

A.1. Detection Results

In this section we will compare detection boxes of three different networks (SSD-RGB, SSD-RGB fused with SSD-D, and SSD-RGB fused with SSD-D+). Since SSD-D+ includes our overlaying technique, this comparison will also evaluate it.

Figure A.1, Figure A.2, Figure A.3, Figure A.4, and Figure A.5 follow the same order: SSD-RGB detections at the top left, SSD-RGB and SSD-D detections at the top right, SSD-RGB and SSD-D+ detections at the bottom.

In Figure A.1 RGB network detected the desk as table and missed the pillow. Depth network detected the desk correctly but missed the pillow also. RGB overlaid depth network found the pillow using the strength of our overlaying technique but labeled the desk as both table and desk.

In Figure A.2 all three networks missed the night stand. RGB network missed the bed and the others missed the sofa also. There are some false chair detections, and we think it is caused by the excess amount of chair instances in the SUN RGB-D dataset on which we trained our depth networks without any pretraining.

In Figure A.3 all of the networks missed the bed, 2 pillows, and the sofa. Only the SSD-RGB network detected the table correctly. Although RGB overlaid depth data helped our network to detect the night stand it affected the sofa to be labeled as table.

In Figure A.4 all of the networks detected the chair successfully but the depth ones detected parts of the chair as another chair. This can be overcome by using a higher NMS threshold.

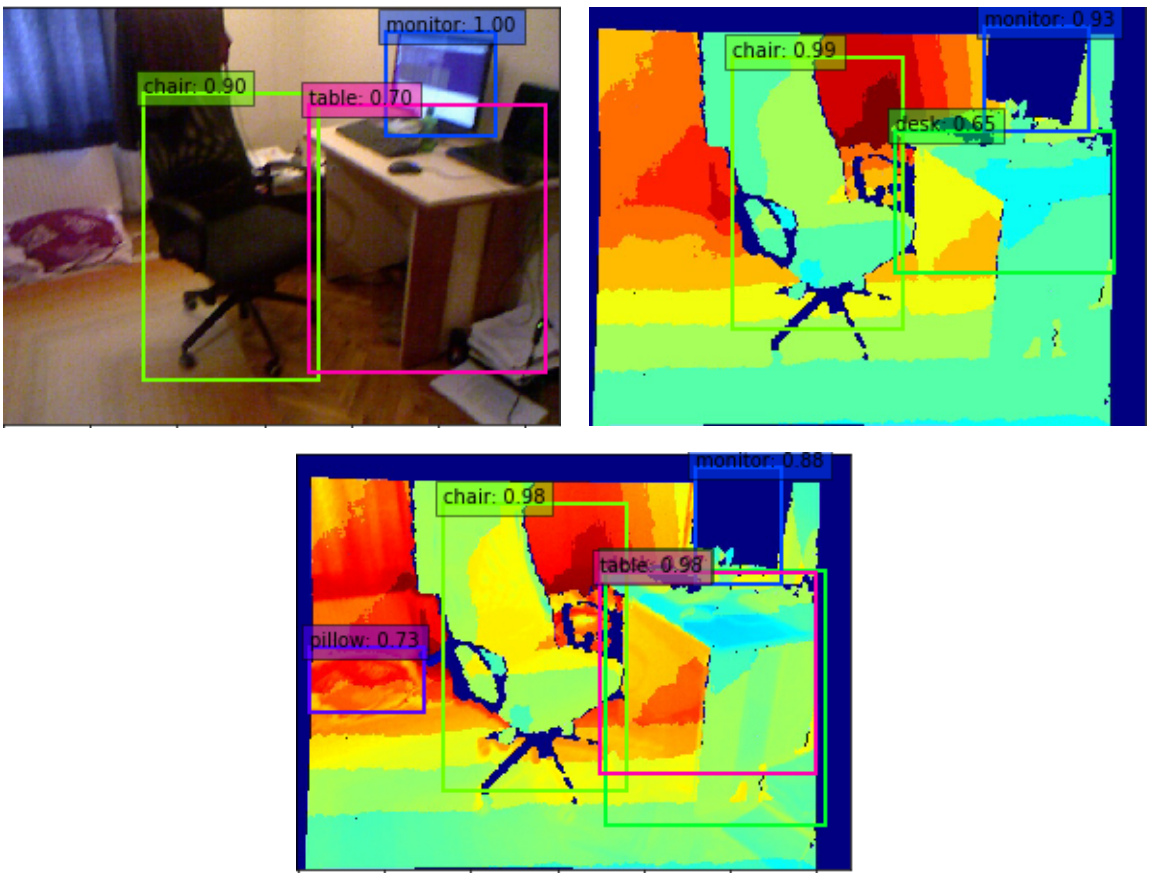

Figure A.1. Chair, table, monitor, desk, and pillow detection boxes from our dataset.

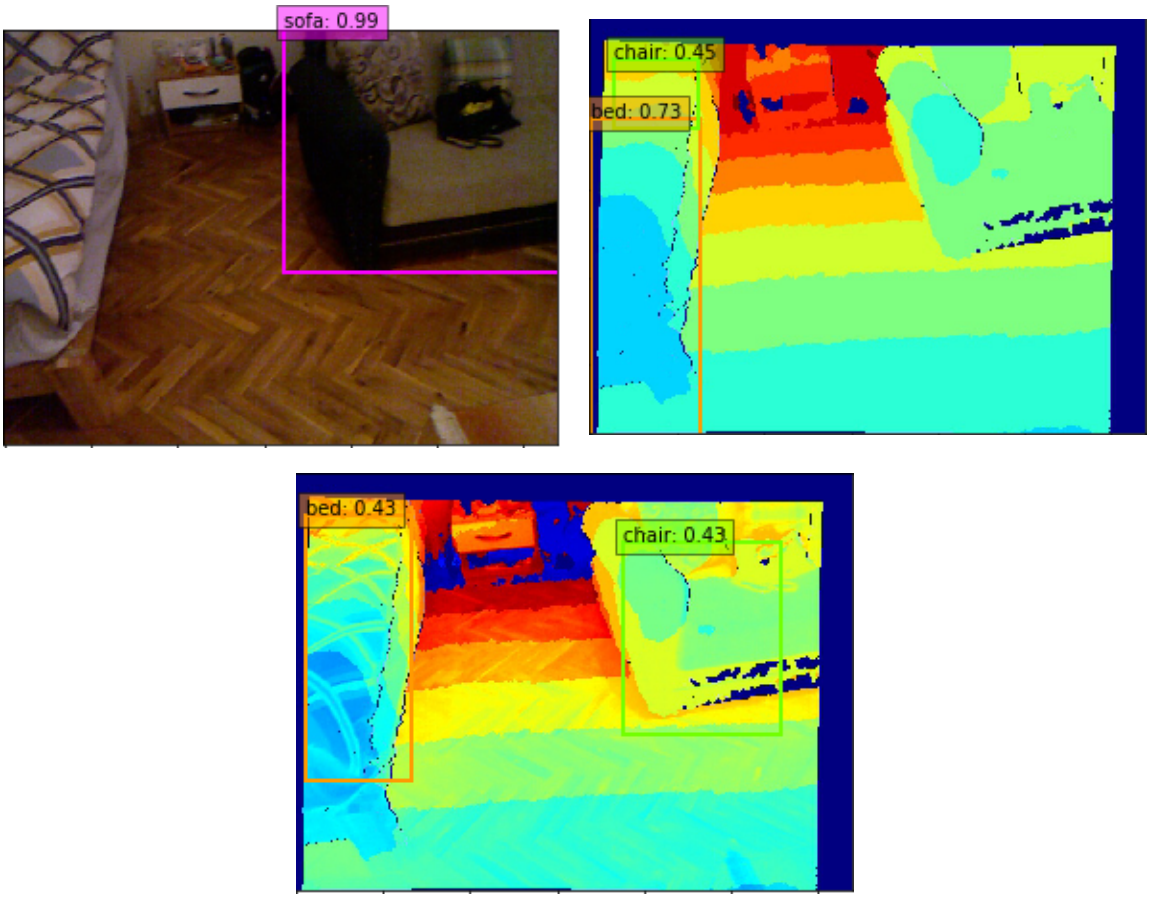

Figure A.2. Sofa, chair, and bed detection boxes from our dataset.

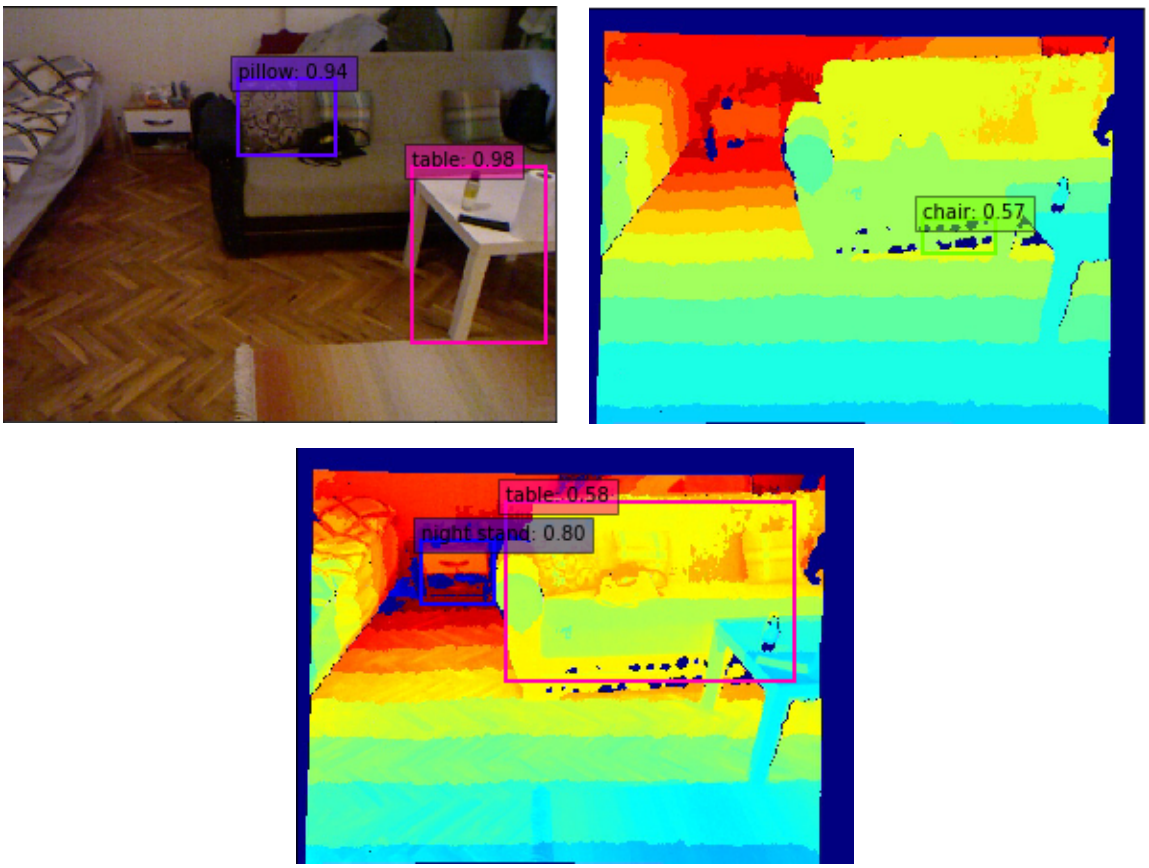

Figure A.3. Pillow, table, night stand, and chair detection boxes from our dataset.

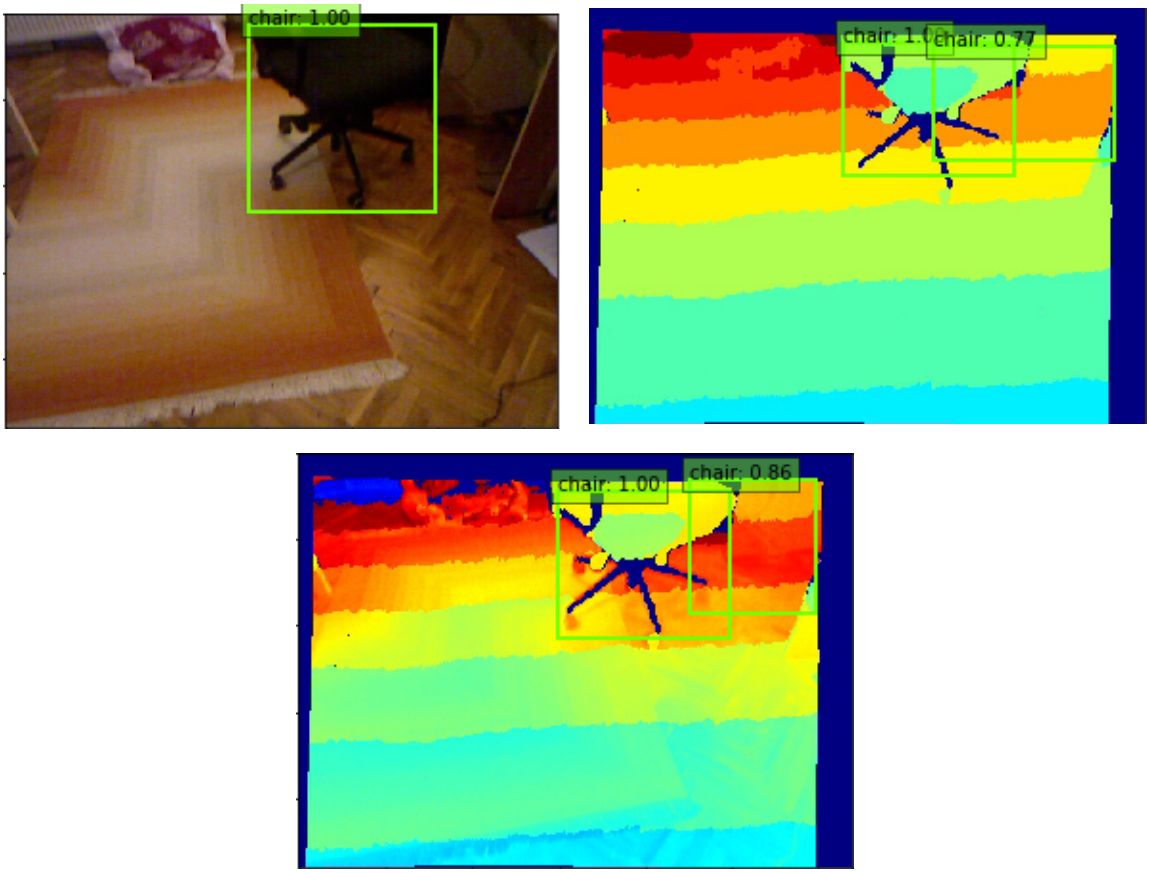

Figure A.4. Chair detection boxes from our dataset.

In Figure A.5 we see the false detection boxes of our depth networks one for pillow and one for table class. Chair is partially detected which is still a false detection. RGB is more stable around these dark areas, which is strange because the depth data should be more robust to illumination changes. Our guess is the reflective surface of the chair caused depth data to fail at those locations.

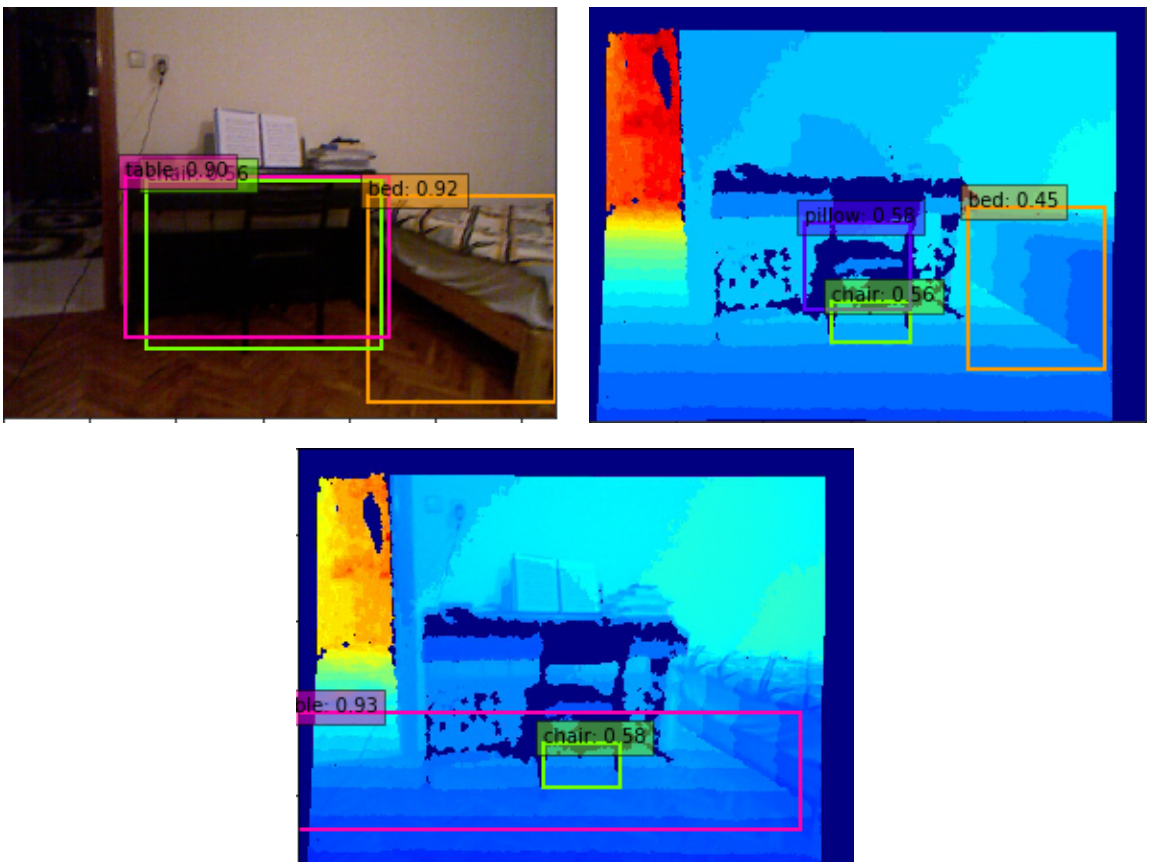

Figure A.5. Table, chair, bed, and pillow detection boxes from our dataset.

A.2. 3D Semantic Mapping

Figure A.6 and Figure A.7 show that our system performs worse with any depth information, because there is not enough data to teach our detector any generic features of objects. That is why using RGB detector for the 3D pixel-wise confidence update creates better 3D semantic maps.

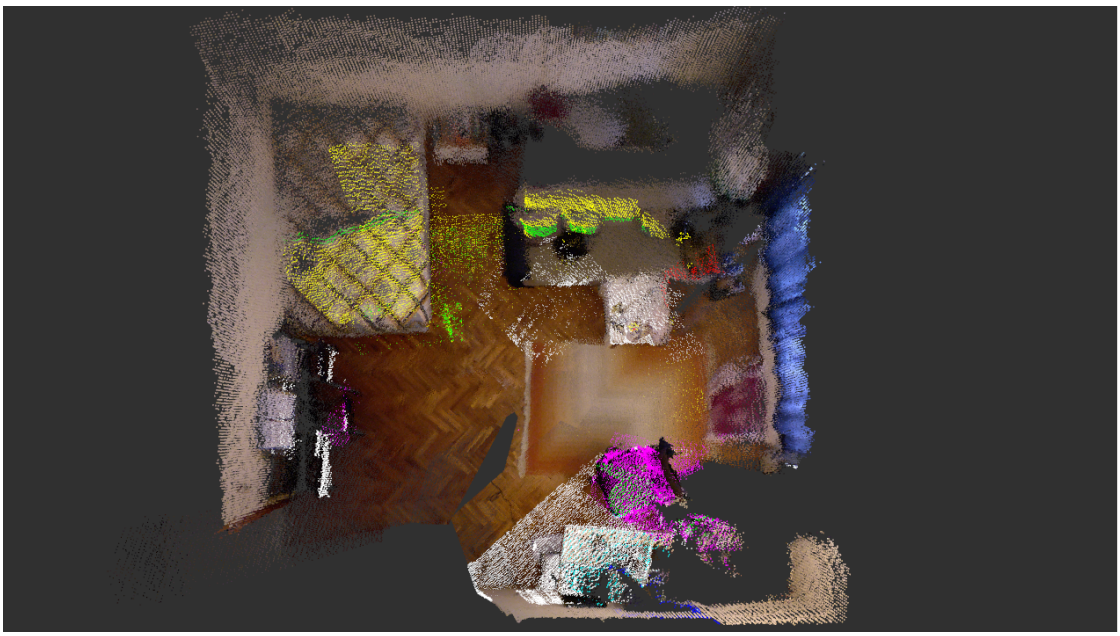

Figure A.6. 3D semantic maps of the first room using the SSD-RGB and SSD-D fusion as the object detector.

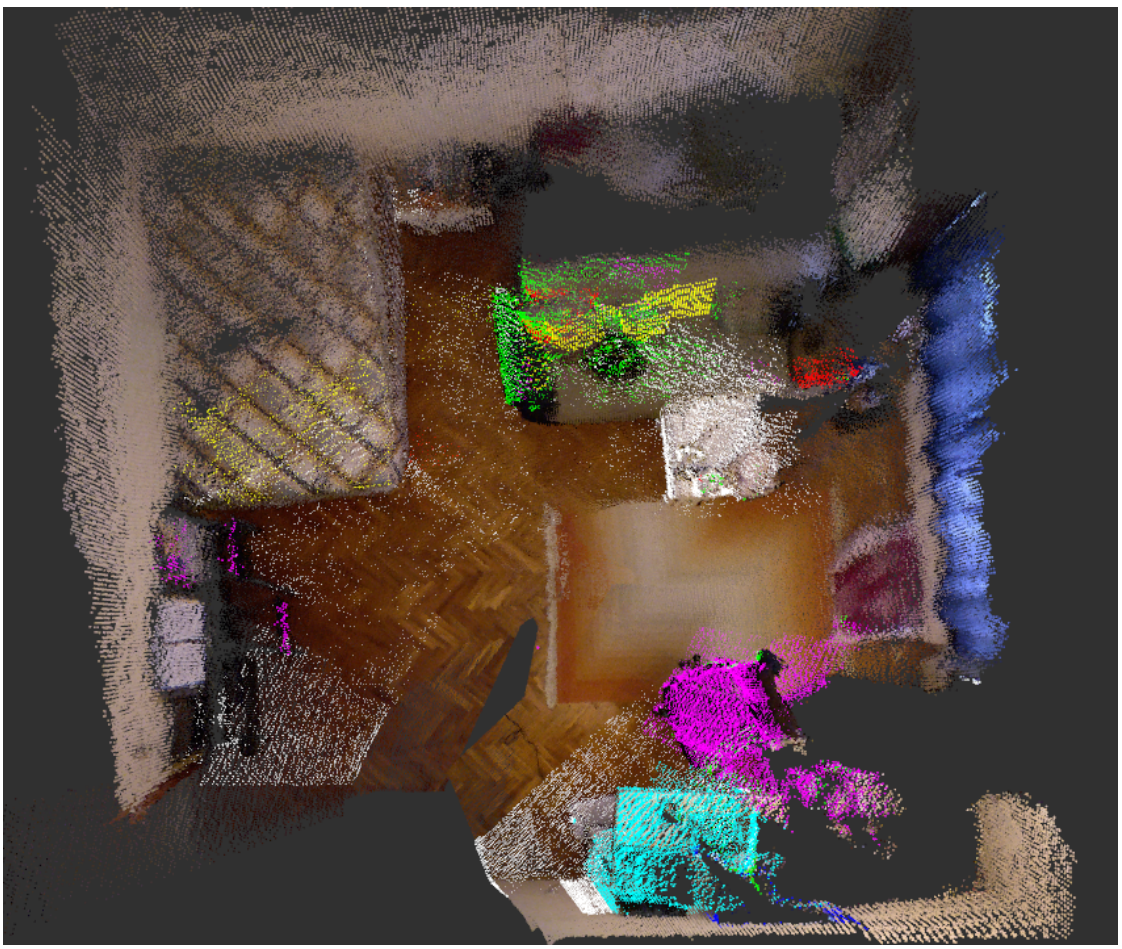

Figure A.7. 3D semantic maps of the first room using the SSD-RGB and SSD-D+ fusion as the object detector.