Developing a simple Artificial Neural Network for fusing RGB and 3D Data

Monday 20th June, 2022 - 21:06

Rafidison Santatra Rakotondrasoa
University of Luxembourg
Email: rafidison.rakotondrasoa.001@student.uni.lu

Jose Luis Sanchez Lopez
University of Luxembourg
Email: joseluis.sanchezlopez@uni.lu

***Abstract*—This document [1] presents the Bachelor Semester Project made by Rafidison Santatra Rakotonsrasoa under the direction of his Project Academic Tutor Jose Luis Sanchez Lopez. In this project, we are interested in the implementation of a simple artificial neural network. The purpose of this project is to develop a simple Artificial Neural Network based on the fusion of RBG and 3D data. The main idea is to classify images based on the fusion of RGB images and 3D values. Deep Learning (DL) and Artificial Neural Networks (ANN) are used.**

1. Introduction

Computer vision is one of the most challenging domains in Artificial Intelligence and detecting object is one of the main focus in this domain. It has many applications such as facial recognition, self-driving cars, robot vision, Unmanned Aerial Vehicles (UAVs) known as drones, and also in other fields such as surveillance, human-computer interaction or in medical diagnosis. The program tries to recognize the image by itself using visual search technology to identify objects through a camera. Usually, the program retrieves information from several images as experience and compare them to the targeted image. Here is when we talk about machine learning, where the computer program learns from the given images and adapt to a new image without human interaction. Also, in object recognition, the use of machine learning methods is essential.

With the development of augmented reality and virtual reality, self-driving cars, three-dimensional vision problem becomes more and more important since it provides much richer information than the two-dimensional vision. Today, about ninety percent of the advances deal in computer vision and machine learning are only with two-dimensional images, stated by Justin Solomon, a senior author of the new series of papers spearheaded by Yue Wang ("our work aims to address a fundamental need to better represent the 3D world, with application not just in autonomous driving, but any field that requires understanding 3D shapes") [2].

The three-dimensional vision is used to enhance the perception of the machine. The machine would know the depth or the distance between the object and the camera. Then, the machine will have a better perception which can be advantageous in the term of accuracy when performing the data classification. We want to develop an Artificial Neural Network (ANN), probably a Convolutional Neural Network (CNN), for processing the fusion of RGB and the 3D (XYZ) data. Our idea is to integrate depth information into trivial two-dimensional digital image in the image recognition task.

2. Project description

2.1. Domains

This Bachelor Semester Project is related to different domains in computer science such as computer vision, machine learning, neural networks and point cloud. In computer science, these domains are often related to object recognition, data classification and more and more about three-dimensional computer vision.

2.1.1. Scientific. The scientific domain of our BSP are the following:

Machine learning is a subset of Artificial Intelligence that studies computer algorithms that improve automatically through experience. An algorithm is a sequence of instructions developed by programmers which are used to solve a problem. In Machine learning, learning algorithms not the programmers create the rules. This approach gives the computer instructions that allow it to learn from data instead of programming step-by-step instructions. Machine learning gives the computer the ability to learn by itself. Based on data training, the computer should be able to improve itself to give better results like a speech recognition system for instance.

Neural networks are a set of machine learning techniques, inspired by the biological neural networks that constitute human brains. It has an important role in Machine Learning and speech recognition. Neural Nets contain statistical models which are used to generate outputs or results from the model. Deep Neural Networks are more complex models of Neural Networks. A deep neural network is simply a feedforward network with many hidden layers.

Lidar is an acronym for Light Detection and Ranging and also known as laser scanning or 3D scanning. Lidar is a remote sensing method used to examine the surface on Earth. In our project, we are using data gathered by a Velodyne lidar sensor or Velodyne laser scanner which gives laser points or point cloud. The technology uses eye-safe laser beams to create a 3D representation or point cloud of the surveyed environment. We want to introduce the laser points cloud data in Object detection instead of RGB images. Nowadays, the use of Lidar is increasing due to its impact in many domains like space exploration, energy sector, environmental, construction, military and defence, energy sector and so on.

2.1.2. Technical. The technical domain of our BSP are the following:

Dataset preparation: the project is abou to code and design a simple Artificial Neural Network for fusing RGB and 3D Data. Therefore, we want to create a dataset in which we develop our model.

Model training: we want to train a neural network model with the dataset. Model training allows a machine to learn, it is commonly known as machine learning. The main goal here is to classify the fusing RGB and 3D data. We perform the training with training samples from the dataset.

Python: the technical part is related to the programming language *python*. The implementation is made entirely using python and some libraries such as numpy, matplotlib, pykitti, openCV, Python Imaging Library or PIL, tensorflow and pytorch.

2.2. Targeted Deliverables

2.2.1. Scientific deliverables. The scientific deliverables want to answer the research question: "How to develop a simple artificial neural network on the fusion of RGB and Lidar data and how to merge those data?". The scientific deliverables can be divided into two parts. On the one hand, all the information about the dataset creation needs to be shown in details. It should present how we can get six-channel data that consists of RGB and Lidar data from RGB images. As a starting point, we work on the Kitti dataset and we get the point cloud of the images in the dataset and convert those 3D point cloud into 2D camera image. On the other hand, we want to elaborate on how machine learning and neural networks work. In this part, we can also see the concept of artificial neural networks in general and convolution neural networks.

2.2.2. Technical deliverables. The technical part shall present the implementation of the required tasks which are to produce the python code for the data preparation and the data training tasks. While the machine learning and neural networks are explained in the scientific part, the design of our neural network model on the six-channel dataset that is created is shown here.

3. Pre-requisites

In the sections below, we see the main scientific and technical knowledge that is required to be known before starting the project.

3.1. Scientific pre-requisites

This project is made by a student in semester 5 in Bachelor so a similar level is recommended. In image detection task including image analysis and image processing, some basic logical computation, linear algebra, discrete mathematics or other college-level math are required. Discrete Mathematics subject, but also Intelligent Systems are recommended but not obliged. Indeed, to be able to process and manipulate images, a college-level in mathematics is required. Mathematics is the basis of image processing techniques. Matrices are used to represent images, in our case, two dimensional and three-dimensional matrices are used. Some knowledge in Intelligent Systems related to machine learning and neural networks are also necessary and recommended but not obliged since we will perform training using a neural network. The most important is the willingness to study, those prerequisites are made to understand what we will see and what we expect to be necessary to understand to be able to understand in details our project.

3.2. Technical pre-requisites

The technical part is meant to understand the implementation aspect of the project. We will use mainly the programming language *python*. It is therefore recommended to be familiar with this language and its libraries such as numpy, tensorflow, pytorch, openCV and so on.

4. A Scientific Deliverable 1

4.1. Requirements

The main objective in this section is based on the research we did which can be separated into different parts which are, first, to detect objects in images of our dataset using YOLO-You only look once which is real-time object detection, afterwards, to prepare our dataset that consists of detected RGB images and point cloud data and finally, to train a simple artificial CNN using the dataset. It is important to know that getting the best accuracy is not the goal here. This

project is meant to get the LiDAR data or point cloud data of images, merge those data with the corresponding RGB images and train our prepared dataset using a neural network model, a Convolutional Neural Network. Some topics should be covered:

– Object detection: it consists of locating or identifying objects. It generally refers the search for the location of one or more objects in an image and draws a bounding box around their extent.

– Data preparation: the steps for the conversion of the 3D point cloud data to a 2D image should be presented here.

– Data classification: it involves predicting the class of one object in an image or any other data. Here, Machine Learning and Deep Neural Networks are introduced. A Convolutional Neural Network is trained to classify images based on a particular fusion of RGB and the point cloud dataset.

All necessary explanations that are referred to the technical part should be presented here, where all the concepts and the necessary information can be understood.

4.2. Design

We decided to divide the scientific section into two subsections which are data preparation and data classification using CNN.

Data preparation

This section covers all the steps from the Kitti dataset that consists of RGB images to a dataset of RGB-XYZ six channels data. The data preparation steps can be divided into three parts which are, first, the objection detection to get the label and the position of the defined classes, afterwards, the point cloud representations to get the point cloud coordinates XYZ in two-dimensional camera images and, in the end, the dataset creation in order to create and save the six channels data.

Object detection system finds objects in the real world from an image of the world. As a starting point, we were interested in how to detect objects. To detect objects from the images in our dataset, we used YOLO. YOLO is used to detect cars, persons and bicycles and to get their positions in each image.

Point cloud or Laser Points are sets of points that describe an object or surface. To create a point cloud, laser scanning technology like Lidar is used. Each point contains an ample amount of data that can be integrated with other data sources or used to create 3D models.

With the detected objects and their respective point cloud representations, we create the dataset that consists of the fusion of RGB images of the objects and their point cloud representations. One part of the project is to merge RGB and Lidar data using images provided by KITTI or Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago.

Data classification

We use a Convolutional Neural Network (CNN). A CNN can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images, they have fewer connections and parameters and so they are easier to train. Despite the attractive qualities of CNN's, they are still expensive to apply on a large scale of data. The main limiting factor is the amount of memory available on the current machine CPU's or GPU's and the amount of training time that we are willing to tolerate. We want to develop a Deep Convolutional Neural Network and adapt it with RGB-XYZ data. Some image manipulation with the KITTI dataset is required. With KITTI, the velodyne Lidar is used to generate datasets collected with a car driving around rural areas of the city. These datasets are publicly available, and download free. In addition to the lidar 3D point cloud data, KITTI dataset also contains video frames from a set of forwarding facing cameras mounted on the vehicle. Since the velodyne scans everything around in 360 degrees and we want to project these scans in 2D images to fuse them with the images provided by the facing cameras which only took the front images.

4.3. Production

4.3.1. Data preparation. In this project, our dataset is based on images provided by KITTI Vision Benchmark Suite or KITTI [3] which contains images and videos collected with a car driving around rural areas of a city equipped with a lidar and a bunch of cameras. We decided to only take into account cars, persons and bicycles to be recognised.

Object Detection

Object detection is a computer vision task that involves both localization and classification of objects in an image. Object detection is most commonly associated with self-driving cars where systems blend computer vision, LIDAR and other technologies to generate a multidimensional representation of the road with all its participants. In our case, we decided to use YOLO to detect objects in images. YOLO has been introduced in 2016 by Joseph Redmon [4]. It predicts multiple bounding boxes and class probabilities for those boxes.

Why are we using YOLO? Detecting images with YOLO is simple and straightforward. Also, YOLO is very fast and very accurate especially for the higher version of YOLO. To detect and localize objects in images we simply run a library of YOLO with predefined weights. In order to retrieve all cars, persons and bicycles in the Kitti images in a very fast and very accurate manner, YOLO is one of the best methods to do it. In our project, YOLO is only used to detect objects,

to get its coordinates, labels and classes for data preparation purposes.

In object detection, we usually use a bounding box to describe the target location. The bounding box is a rectangular box that can be determined by the x and y axis coordinates in the upper-left corner and the x and y axis coordinates in the lower-right corner of the rectangle. Another commonly used bounding box representation is the x and y axis coordinates of the bounding box center, and its width and height.

Figure 3 shows the results after using YOLO detection method in an image.

Representations for Point Cloud

In this section, we formulate the three-dimensional point to a two-dimension camera image problem. The goal is to project the 3D data to 2D points in an image. To make this conversion of 3D to 2D data, we do matrix multiplications.

First, we project the 3D provided velodyne coordinates (in */velodyne_points/data/*.bin*) to the camera coordinates using some provided rotation $R^{(0)}_{rect}$ and translation matrix t^{cam}_{velo} (in */calib_velo_to_cam.txt*). Afterwards, we project the produces camera coordinates to an image (pixel) coordinate using some provided projection matrix $P^{(i)}_{rect}$ (in */calib_cam_to_cam.txt*). Those operations are represented by the following function [5], where a 3D point x in Velodyne coordinates gets projected to a point y in the i-th camera image as

$$\mathbf{P}^{(i)}_{rect} = \begin{pmatrix} f^{(i)}_u & 0 & c^{(i)}_u & -f^{(i)}_u b^{(i)}_x \\ 0 & f^{(i)}_v & c^{(i)}_v & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\mathbf{T}^{cam}_{velo} = \begin{pmatrix} \mathbf{R}^{cam}_{velo} & \mathbf{t}^{cam}_{velo} \\ 0 & 1 \end{pmatrix}$$

$$\mathbf{y} = \mathbf{P}^{(i)}_{rect}\ \mathbf{R}^{(0)}_{rect}\ \mathbf{T}^{cam}_{velo}\ \mathbf{x}$$

where i $\in \{0, 1, 2, 3\}$ is the camera index, where 0 represents the left grayscale, 1 the right grayscale, 2 the left color and 3 the right color camera.

In the figure 4, from the red color to blue color, we can distinguish the closest and the most distant points in the depth space view of the image.

Dataset creation

To train a model using a CNN, which we will see later on in this paper, we should have at least a training and a test dataset. The training dataset is the sample of data used to fit the model that means the model learn from this data and the test dataset is used to evaluate the model. When the model is completely trained and has learned from the training dataset, we test our model with the test dataset to get its accuracy. Now, we know that the training data is for the model fitting and the test data for estimating the model's accuracy.

Using YOLO, we get the label and the bounding box coordinates of every person, bicycles and cars in all the images from KITTI. Then, instead of RGB images, our dataset is composed of the RGB XYZ six channels data. The six channels data is the fusion of detected RGB images of the detected objects and XYZ data which are the corresponding 2D camera projection of the 3D point cloud. RGB contains the values of the Red, Blue and Green colors of the image and XYZ contains the image point cloud coordinates. Those point cloud coordinates are the results of the point cloud representation that we have seen in the previous section.

4.3.2. Convolutional Neural Network training. There are two basic tasks in machine learning: classification and regression. In our case, we want to predict a class. The data training or data classification task want to answer the question "What is in the image?". In this project, we want to do a classification of three classes (person, bicycle and car). We train our model (a CNN) such that, based on our dataset that was created and explained in the previous section, our model will predict the class.

Artificial Neural Network (ANN)

An artificial neural network, on a high abstraction level, is a collection of interconnected artificial neurons. Each of these artificial neurons can perform a simple computation on their inputs, and signal the result as an output. Then, other neurons can receive these signals as input and perform their own computations. In network architectures, this hierarchy is established by making use of layers. This process, called forward propagation, is performed until the neurons at the end of the network are reached.

Artificial Neurons

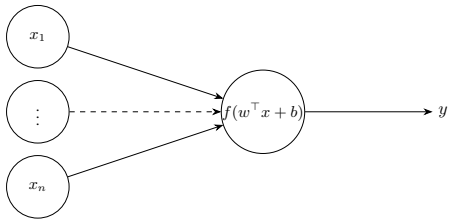

Figure 1. *The internal computation of an artificial neuron, where* $x, w \in \mathbb{R}^n$ *and* $b \in \mathbb{R}$. x *is the input vector, where as* y *is the output of the neuron.*

A single neuron in a network has three components; (1) the weights of individual inputs, (2) the bias that is added to the sum of inputs, and (3) the activation function. A neuron can be represented by using a vector:

$$x = [x_1; x_2; \ldots; x_n] \in \mathbb{R}^n \tag{1}$$

The weight values of a neuron can be represented in the form of a vector:

$$w = [w_1; w_2; \ldots; w_n] \in \mathbb{R}^n \tag{2}$$

When a neuron receives its inputs, it multiplies each input with the corresponding weight, and takes their sum. This can be simplified as a matrix product:

$$z = \sum_{i=1}^{n} w_i x_i \tag{3}$$

$$= w^\top x \tag{4}$$

After computing the weighted sums of inputs, the neuron passes the value of z to its activation function to finally compute its output;

$$y = f(z) \tag{5}$$

Activation functions

The aim of the activation function is to introduce non-linearity into the model. This function is applied for the output of the hidden layer $Z^{(h)}$ and the output of the output layer $Z^{(out)}$. We can see below some examples of activation functions:

- *Sigmoid function*: sigmoid activation function is one common activation function. This function bounds values between 0 and 1 which are often interpreted as probabilities.

$$A(z) = 1/(1 + e^{-z})$$

- *Tanh or hyperbolic tangent function*: it re-scales the sigmoid function, such that its outputs have values between -1 and 1.

$$A(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z})$$

- *Softmax function*: softmax activation function bounds values between 0 and 1. It is a normalized version of the sigmoid function used for the output layer in multiclass classification.

$$A(z_j) = e^{z_j} / \sum_{k} e^{z_k}$$

- *ReLu function*: ReLu function is a simple activation function used basically for large deep networks applications.

$$A(z) = max(0, z)$$

Convolutional Neural Network (CNN)

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. An image contains a series of pixels in a grid-like fashion that contains values to denote what color is each pixel.

In the left image, a regular 3-layer Neural Network is shown and in the right, a CNN arranges its neurons in three dimensions (width, height, depth). Every layer of a CNN transforms the 3D input volume to a 3D output volume of a neuron. Here the input layer, represented in red, the width and height represent the dimensions of the input image and the depth represents the 3 channels RGB (Red, Green and Blue). A CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer.

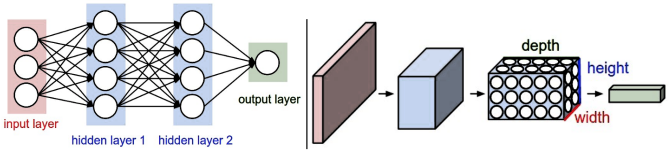

Figure 2. A regular and a convolutional neural net

A convolutional layer performs a dot product between two matrices, a kernel and a restricted portion of the receptive field in the image. The kernel is spatially smaller than an image but is more in-depth. During the forward pass, the kernel slides across the height and width of a receptive region of an image. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride.

A pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually. There are several pooling functions such as the average or the L2 norm of the rectangular neighborhood, or the weighted average based on the distance from the central pixel. However, max pooling is the most popular. It reports the maximum output from the neighborhood.

In a fully connected (FC) layer, the neurons have full connectivity with all neurons in the preceding and succeeding layer. It can be computed as a usual matrix multiplication followed by a bias effect. The FC layer helps to map the representation between the input and the output.

4.4. Assessment

The scientific part tried to explain all the concept needed to understand the technical part. The objective in this part of the project was to introduce some topics concerning the technical part, then the content presented in this section elaborate on the overall knowledge required for the following technical part of the project. Indeed, since the goal was to prepare the six channel dataset and to develop a neural network, the steps on the creation of our dataset, the fundamental knowledge in machine learning was presented and some information about the artificial neurons and deep convolutional neural network was covered.

5. Implementation of Data preparation and Data training

5.1. Requirements

In this section, each step during the implementation of our system is covered. Data preparation is the first task in our system. The idea is to prepare the data to feed our convolutional neural network. The work should be covered in this technical part are, first, to get the point cloud representation of the Kitti images and the projection of the 3D data to a 2D camera representation. Afterwards, we want to merge those data with the corresponding RGB data to create our dataset. Finally, the data training which includes the designing of deep convolutional neural networks is the last part that is required.

This technical report has been written alongside the development and implementation of our project.

5.2. Design

Since it is too difficult to implement our code from scratch, some libraries, tools, predefined algorithms, predefined system and the existing raw data from KITTI are used.

Tools

For the implementation, we are using python with the following libraries:

– `Numpy`: for the manipulation and computation made on images which are represented in large matrices. We also use numpy to save data into npy file to create our dataset.

– `Pykitti`: it is a package for working with Kitti dataset in python. This package can be useful when we work with the raw datasets and odometry benchmark datasets from Kitti. This package provides a minimal set of tools for working with the KITTI dataset in Python. In our case, we are using the raw datasets and we used pykitti to manage the provided data. We need this package to load and access the parts we need in the dataset and also the use of some advanced functions like for instance the function that loads velodyne scans as [x,y,z,reflectance].

– `OpenCV`: it is the acronym for Open Source Computer Vision Library, this library is used to read images but also for image processing.

– `Pytorch`: it is a machine learning library, this library provides a wide range of algorithms for deep learning. In our case, we only use pytorch to load YOLO, an image detection system, used to get the positions and the labels of objects in images in the data preparation process.

– `Tensorflow`: it is one of the most in-demand and popular open-source deep learning frameworks available today. We use it to design and create our concolutional neural network model and to train it with our dataset.

KITTI dataset

As mentioned before, we will use the Kitti dataset to prepare and create our dataset. The raw data described in this paper can be accessed from here [3]. The raw dataset is divided into the categories ‘Road’, ‘City’, ‘Residential’, ‘Campus’ and ‘Person’. For each sequence, the raw data provides the object annotations in form of 3D bounding box tracklets and a calibration file. The recordings have taken place on the 26th, 28th, 29th, 30th of September and on the 3rd of October 2011. The total size of the provided online data is 180 GB.

In our case we will use a small part of the dataset from the 26th of September 2011 into the categories ’Road’. After downloading the raw data from Kitti, we got the folder named `raw_data`. In this folder, we can find the following data:

– `raw_data/2011_09_26/`, in which we can find the following datasets and files:

- `2011_09_26_drive_0001_sync`: The size of this dataset is 0.4 GB which contains data for just 11 seconds of driving, 114 frames, image resolution of 1392 x 512 pixels and the labels are 12 Cars, 0 Vans, 0 Trucks, 0 Pedestrians, 0 Sitters, 2 Cyclists, 1 Trams, 0 Misc.
- `2011_09_26_drive_0002_sync`: It is one of the smallest datasets out there (0.3 GB) which contains data for just 8 seconds of driving, 83 frames, image resolution of 1392 x 512 pixels and the labels are 1 Cars, 0 Vans, 0 Trucks, 0 Pedestrians, 0 Sitters, 2 Cyclists, 0 Trams, 0 Misc.
- `2011_09_26_drive_0005_sync`: The size of this dataset is 0.6 GB which contains data for just 16 seconds of driving, 160 frames, image resolution of 1392 x 512 pixels and the labels are 9 Cars, 3 Vans, 0 Trucks, 2 Pedestrians, 0 Sitters, 1 Cyclists, 0 Trams, 0 Misc.
- `calib_velo_to_cam`: this file contains the information about the rotation matrix R and the translation matrix T. R is a 3x3 matrix and T is a 3x1 matrix. We use those matrices to convert velodyne coordinates to camera coordinates.
- `calib_cam_to_cam.txt`: this file contains the information about the project mtrix P, P is a 3x3 matrix. We use P to convert camera coordinates into image coordinates or pixel.
- `calib_imu_to_velo`: We do not use this file.

– `raw_data/2011_09_26/2011_09_26_drive_XXXX_sync`: in each of the dataset above, we can find the following documents:

- `image_00`: this folder contains left (stereo) grayscale images.

- `image_01`: this folder contains right (stereo) grayscale images.
- `image_02`: this folder contains left (stereo) color images.
- `image_03`: this folder contains right (stereo) color images.
- `velodyne_points`: this folder contains all the corresponding laser points or point cloud of each image. An image named `0000000001.png` corresponds to the point cloud represented in the file `0000000001.bin`.

The Figure 5 specified the setup of the car used in KITTI. The vehicle is equipped with the Stereo camera Rig (includes four video cameras, two color and two grayscale cameras), a 360° rotating 3D Velodyne laser scanner and a combined GPS/IMU inertial navigation system. However, we will only focus on the left monochrome camera, left color camera of the camera Stereo Rig and the 64 beams Velodyne laser scanner. The point cloud is represented in 3D, each point encodes the XYZ values.

Data preparation

The purpose is to classify three different classes from our dataset. We will just consider the classes *person, bicycle and car*. For data preparation, the tasks can be divided into different parts: we project the 3D velodyne data to the 2D camera representation, and with the images in the Kitti that we have downloaded we localize and extract the coordinates of the classes that we have considered. YOLO is used as object detection to get the labels, the classes and to localize the coordinates of each detected objects. Each resulting cropped image is concatenated with the corresponding 2D point cloud data. Using the produced coordinates of each object, we crop and save the produced data into the folder with each name of the corresponding label.

Data training

For the data training or data classification, as mentioned before, we focused on classifying 3 classes which are person, bicycle and car. We developed a deep convolutional neural network in which we will train with our data that are created.

5.3. Production

5.3.1. Data preparation. We will see in this section how the dataset is created.

Data management

To manage our data we will use pykitti. This package allows us to access the part we need in the kitti dataset. The general idea is to specify what the parts you want to load and access in the dataset. The Figure 7 shows how we access the kitti dataset. Here, we loaded the dataset 2011_09_26_drive_0001.

Object detection

We used a pretrained detection system called YOLO to detect objects and to get its coordinates in images. As shown in the Figure 8. With the library *torch*, our model YOLO is loaded. The model take an image as input and returns the classes, labels and the coordinates of the detected objects.

Representations for Point Cloud

Kitti provides the images with the velodyne point cloud or point cloud. The problem is that the point cloud is represented in 3D, and it represents everything around it, i.e. in 360 degrees. It takes even the points that are behind the camera. Then we want to project only the points that are in front of the camera then we should take a fixed field of view (FOV), afterwards, we project the 3D points in 2D camera like it was explained in the scientific part of this project (you can see more details in Section 4.3.1). The representations for Point Cloud are divided in three steps, the first step is to extract points corresponding to a FOV setting, the second step is to project the 3D velodyne points in 2D camera image and the last step is to scale the 2D camera coordinates into a defined size. We can see more information about those steps in the following:

Step 1 - To extract points corresponding to a FOV setting:

The velodyne points are loaded from a binary file and converted to a matrix of N rows and 4 columns as shown in Figure 9, with XYZ values in the first three columns and the last column which we do not really need to represent the values of the reflectance. We know that the velodyne laser scanner takes the points all-around in 360°, and we want to only extract the points corresponding to a field of view (FOV). The FOV can be defined as having a vertical component (v-FOV, a Vertical Field of View) and a horizontal component (h-FOV, a Horizontal Field of View) as shown in Figure 10.

Step 2 - To project the 3D velodyne points in 2D camera:

As explained before, in Section 4.3.1, we can project the 3D velodyne data to 2D camera image by doing some matrix computations. Fortunately, the matrices that are needed in this projection are provided by Kitti. We need the rotation and translation matrices provided in the file `calib_velo_to_cam.txt` and the projection matrix provided in the file `calib_cam_to_cam.txt`. We get the rotation and translation matrices using the function presented in Figure 13 and get the projection matrix using the function presented in the Figure 14. We can see in Figure 15, how we get the 2D camera coordinates from the velodyne 3D coordinates.

Step 3 - To scale the 2D camera coordinates to a defined size:

Now, each point in the point cloud are represented in 2D camera coordinates like a pixel such that each point has two coordinates. Next, we want to resize the size of the point cloud to get along the size of the image. In our case, every

image has a size 1242 x 375 pixels, then the 2D coordinates are scaled to the same size as the images. We can see the implementation in Figure 16.

Dataset creation

Now, we want to merge or concatenate the RGB images with the corresponding 2D camera point cloud. Since the images and the point cloud have the same size, we just concatenate those data to get the RGB-XYZ six channels data. The implementation of the creation of those data are shown in Figure 17.

Those six channel data are saved in *npy* file. We decided that those data are saved in npy file instead of cvs or txt file because it is the faster to load data from npy than the other format. Indeed, there is a big difference when loading huge amount of data between *.npy*, *csv* or *.txt* files [6].

All the data with the same label, a car, a person or a bicycle are located in the same folder (folder *0* for person, *1* for bicycle and *2* for car). We can see in the following the folder structure of our dataset:

– `dataset/train/0/`: contains the six channel data with a label *person*

– `dataset/train/1/`: contains the six channel data with a label *bicycle*

– `dataset/train/2/`: contains the six channel data with a label *car*

– `dataset/test/0/`: contains the six channel data with a label *person*

– `dataset/test/1/`: contains the six channel data with a label *bicycle*

– `dataset/test/2/`: contains the six channel data with a label *car*

5.3.2. CNN training. For data training, the experiments have been performed on the CNN architecture presented in the Figure 18.

5.4. Assessment

The objective in the technical part of the project was to implement and develop our data training system based on the fusion of RGB and Lidar Data.

On one hand, the tasks are done from the data preparation to the training phase, but on the other hand, the resulting accuracy of our system is still low.

Acknowledgment

I would like to thank the BiCS management and education team and the BiCS course director Mr Nicolas GUELFI.

I would also like to thank my Project Academic Tutor Mr Jose Luis SANCHEZ LOPEZ and Mr Dario CAZZATO for their help, for their support, for their time spent helping me throughout the project and for the advice concerning the missions mentioned in this report.

6. Conclusion

We were working on an object classification using the RGB-XYZ data. The dataset is provided by some predefined camera data from the Kitti Dataset which allowed us to get the lidar data with the original color images. The scientific part was to introduce and analyse some needed background relating to our goal. Some fundamental backgrounds in machine learning and image analysis were presented. For the technical part, it was mainly the implementation of data preparation and data classification.

References

[1] Nicolas Guelfi. Bics bachelor semester project report template, 2017. University of Luxembourg, BiCS - Bachelor in Computer Science, https://github.com/nicolasguelfi/lu.uni.course.bics.global.

[2] Justin Solomon and Yue Wang. Deep closest point: Learning representations for point cloud registration, 2019. https://arxiv.org/abs/1905.03304.

[3] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset, 2013. http://www.cvlibs.net/datasets/kitti/raw_data.php.

[4] Ross Girshick Ali Farhadi Joseph Redmon, Santosh Divvala. You only look once: Unified, real-time object detection, 2015. https://arxiv.org/abs/1506.02640.

[5] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset international journal of robotics research (ijrr) 32 (2013), pp. 1229-1235. https://www.mrt.kit.edu/z/publ/download/2013/GeigerAl2013IJRR.pdf.

[6] Peter Nistrup. What is .npy files and why you should use them..., 2019, May 07. https://towardsdatascience.com/what-is-npy-files-and-why-you-should-use-them-603373c78883.

7. Appendix

Figure 3. RGB and prediction image

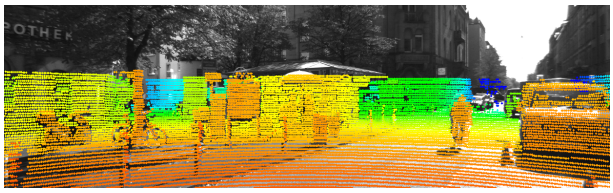

Figure 4. Point cloud representation

Figure 5. Recording platform: a fully equipped vehicle.

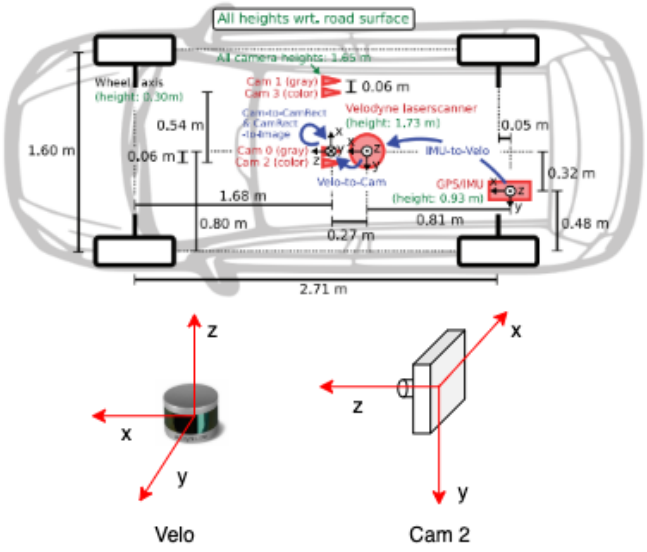

Figure 6. This figure illustrates the dimensions and mounting positions of the sensors (red).

```python
import pykitti

basedir = 'raw_data'
date = '2011_09_26'
drive = '0001'

dataset = pykitti.raw(basedir, date, drive)
```

Figure 7. Load data with kitti

```python
img = dataset.get_rgb(idx) # img[0] and img[1] are the left and right images

model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True).autos
hape()

results = model(img[0], size=640)
```

Figure 8. To detect the objects and to get the coordinates of an image

```python
def load_from_bin(bin_path):
    obj = np.fromfile(bin_path, dtype=np.float32).reshape(-1, 4)
    # ignore reflectivity info
    return obj[:,:3]
```

Figure 9. Load binary file function.

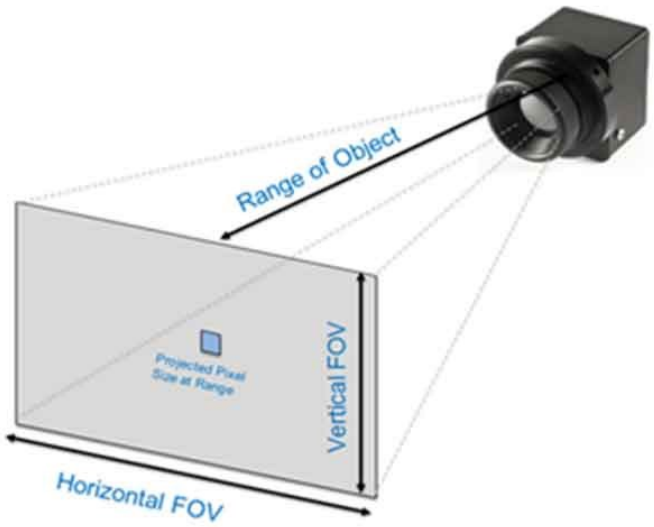

Figure 10. Set up of the field of view.

```python
def in_h_range_points(points, m, n, fov):
    """ extract horizontal in-range points """
    return np.logical_and(np.arctan2(n,m) > (-fov[1] * np.pi / 180), \
                          np.arctan2(n,m) < (-fov[0] * np.pi / 180))

def in_v_range_points(points, m, n, fov):
    """ extract vertical in-range points """
    return np.logical_and(np.arctan2(n,m) < (fov[1] * np.pi / 180), \
                          np.arctan2(n,m) > (fov[0] * np.pi / 180))

def fov_setting(points, x, y, z, dist, h_fov, v_fov):
    """ filter points based on h,v FOV  """

    if h_fov[1] == 180 and h_fov[0] == -180 and v_fov[1] == 2.0 and v_fov[0] ==
    -24.9:
        return points

    if h_fov[1] == 180 and h_fov[0] == -180:
        return points[in_v_range_points(points, dist, z, v_fov)]
    elif v_fov[1] == 2.0 and v_fov[0] == -24.9:
        return points[in_h_range_points(points, x, y, h_fov)]
    else:
        h_points = in_h_range_points(points, x, y, h_fov)
        v_points = in_v_range_points(points, dist, z, v_fov)
        return points[np.logical_and(h_points, v_points)]
```

Figure 11. FOV setting function.

```python
def velo_points_filter(points, v_fov, h_fov):
    """ extract points corresponding to FOV setting """

    x = points[:, 0]
    y = points[:, 1]
    z = points[:, 2]
    dist = np.sqrt(x ** 2 + y ** 2 + z ** 2)

    if h_fov[0] < -90:
        h_fov = (-90,) + h_fov[1:]
    if h_fov[1] > 90:
        h_fov = h_fov[:1] + (90,)

    x_lim = fov_setting(x, x, y, z, dist, h_fov, v_fov)[:,None]
    y_lim = fov_setting(y, x, y, z, dist, h_fov, v_fov)[:,None]
    z_lim = fov_setting(z, x, y, z, dist, h_fov, v_fov)[:,None]

    # Stack arrays in sequence horizontally
    xyz_ = np.hstack((x_lim, y_lim, z_lim))
    xyz_ = xyz_.T

    # stack (1,n) arrays filled with the number 1
    one_mat = np.full((1, xyz_.shape[1]), 1)
    xyz_ = np.concatenate((xyz_, one_mat),axis = 0)

    # need dist info for points color
    dist_lim = fov_setting(dist, x, y, z, dist, h_fov, v_fov)
    color = depth_color(dist_lim, 0, 70)

    return xyz_, color
```

Figure 12. Velodyne points filter function.

```python
def calib_velo2cam(filepath):
    """
    get Rotation(R : 3x3), Translation(T : 3x1) matrix info
    using R,T matrix, we can convert velodyne coordinates to camera coordinates
    """
    with open(filepath, "r") as f:
        file = f.readlines()

        for line in file:
            (key, val) = line.split(':',1)
            if key == 'R':
                R = np.fromstring(val, sep=' ')
                R = R.reshape(3, 3)
            if key == 'T':
                T = np.fromstring(val, sep=' ')
                T = T.reshape(3, 1)
    return R, T
```

Figure 13. Velo to camera calibration function.

```python
def calib_cam2cam(filepath, mode):
    """
    If your image is 'rectified image' :
        get only Projection(P : 3x4) matrix is enough
    but if your image is 'distorted image'(not rectified image) :
        you need undistortion step using distortion coefficients(5 : D)

    in this code, I'll get P matrix since I'm using rectified image
    """
    with open(filepath, "r") as f:
        file = f.readlines()

        for line in file:
            (key, val) = line.split(':',1)
            if key == ('P_rect_' + mode):
                P_ = np.fromstring(val, sep=' ')
                P_ = P_.reshape(3, 4)
                # erase 4th column ([0,0,0])
                P_ = P_[:3, :3]
    return P_
```

Figure 14. Camera calibration funciton.

```python
def velo3d_2_camera2d_points(points, v_fov, h_fov, vc_path, cc_path,mode='00'):
    """ print velodyne 3D points corresponding to camera 2D image """

    R_vc, T_vc = calib_velo2cam(vc_path)
    RT_ = np.concatenate((R_vc, T_vc),axis = 1)

    P_ = calib_cam2cam(cc_path, mode)

    # xyz_v - 3D velodyne points corresponding to h, v FOV in the velodyne
    # coordinates
    # c_    - color value(HSV's Hue) corresponding to distance(m)

    xyz_v, c_ = velo_points_filter(points, v_fov, h_fov)

    # convert velodyne coordinates(X_v, Y_v, Z_v) to camera coordinates
    # (X_c, Y_c, Z_c)
    for i in range(xyz_v.shape[1]):
        xyz_v[:3,i] = np.matmul(RT_, xyz_v[:,i])

    # xyz_c - 3D velodyne points corresponding to h, v FOV in the camera
    # coordinates

    # convert camera coordinates(X_c, Y_c, Z_c) image(pixel) coordinates(x,y)
    for i in range(xyz_c.shape[1]):
        xyz_c[:,i] = np.matmul(P_, xyz_c[:,i])

    # xy_i - 3D velodyne points corresponding to h, v FOV in the image(pixel)
    # coordinates before scale adjustment
    # ans  - coordinates of every point cloud

    xy_i = xyz_c[::]/xyz_c[::][2]
    ans = np.delete(xy_i, 2, axis=0)

    return ans, c_, xyz_c
```

Figure 15. Velo to camera projection function.

```python
def in_range_points(points, size):
    """ extract in-range points """
    return np.logical_and(points > 0, points < size)

""" Adapt the point cloud coordinates to a defined scale of 2D image """
width = 1242
height = 375
w_range = in_range_points(ans[0], width)
h_range = in_range_points(ans[1], height)
xyz_c_x = xyz_c[0][np.logical_and(w_range,h_range)][:,None].T
xyz_c_y = xyz_c[1][np.logical_and(w_range,h_range)][:,None].T
xyz_c_z = xyz_c[2][np.logical_and(w_range,h_range)][:,None].T
ans_x = ans[0][np.logical_and(w_range,h_range)][:,None].T
ans_y = ans[1][np.logical_and(w_range,h_range)][:,None].T
c_ = c_[np.logical_and(w_range,h_range)]

xyz_c = np.vstack((xyz_c_x, xyz_c_y, xyz_c_z))

ans = np.vstack((ans_x, ans_y))
```

Figure 16. Scale velo points to a defined height and width.

```python
def create_RGB_XYZ(velo_points, v2c_filepath, c2c_filepath, image):
    """ create 6 channels data (RGB_XYZ) """
    ans, c_, xyz = velo3d_2_camera2d_points(velo_points, v_fov=(-24.9, 2.0),
                   h_fov=(-45,45), vc_path=v2c_filepath, cc_path=c2c_filepath)
    coord = ans
    points = xyz

    arr_3 = np.zeros((image.shape[0], image.shape[1], 3))
    image_6 = np.concatenate((image,arr_3), axis=2)

    for i in range(coord.shape[1]):
        image_6[np.int32(coord[1][i]), np.int32(coord[0][i]), 3] = points[0][i]
        image_6[np.int32(coord[1][i]), np.int32(coord[0][i]), 4] = points[1][i]
        image_6[np.int32(coord[1][i]), np.int32(coord[0][i]), 5] = points[2][i]

    return image_6
```

Figure 17. RGB-XYZ data creation.

conv2d_5 (Conv2D)	(None, 54, 54, 96)	69792
batch_normalization_5 (Batch	(None, 54, 54, 96)	384
max_pooling2d_3 (MaxPooling2	(None, 26, 26, 96)	0
conv2d_6 (Conv2D)	(None, 26, 26, 256)	614656
batch_normalization_6 (Batch	(None, 26, 26, 256)	1024
max_pooling2d_4 (MaxPooling2	(None, 12, 12, 256)	0
conv2d_7 (Conv2D)	(None, 12, 12, 384)	885120
batch_normalization_7 (Batch	(None, 12, 12, 384)	1536
conv2d_8 (Conv2D)	(None, 12, 12, 384)	147840
batch_normalization_8 (Batch	(None, 12, 12, 384)	1536
conv2d_9 (Conv2D)	(None, 12, 12, 256)	98560
batch_normalization_9 (Batch	(None, 12, 12, 256)	1024
max_pooling2d_5 (MaxPooling2	(None, 5, 5, 256)	0
flatten_1 (Flatten)	(None, 6400)	0
dense_3 (Dense)	(None, 4096)	26218496
dropout_2 (Dropout)	(None, 4096)	0
dense_4 (Dense)	(None, 4096)	16781312
dropout_3 (Dropout)	(None, 4096)	0
dense_5 (Dense)	(None, 3)	12291

Figure 18. CNN architecture