Car detection system based on LiDAR and camera data fusion

Konrad Lis, Joanna Stanisz
konrad.lis11@gmail.com
joanna.stanisz.12@gmail.com

Submitted for the 2019 Digilent Design Contest Europe

11.05.2019

Advisor: PhD Tomasz Kryjak

AGH University of Science and Technology in Cracow

Table of contents

Introduction

Abstract

The purpose of this project is implementation of hardware-software object detection system based on LiDAR and camera data. Nowadays, this issue becomes more and more popular. The reason behind this is the falling price of LiDARs and the development of autonomous technologies. LiDAR point cloud processing can be divided into several stages. Firstly, ground is removed and data is filtered. Then remaining point cloud is divided into segments which are classified. Image processing takes advantage of point cloud processing results. The first step consists of projecting segments (obtained from point cloud) to image plane. The smallest rectangle which covers all of projected points is treated as region of interest. The image from ROI is resized and classified by HOG+SVM descriptor. Then data fusion step is performed. Classification probability from point cloud processing and image processing are transformed into final classification probability. The final result of this project should be detected cars marked with rectangles on the image.

Objectives

Nowadays, issue of detecting objects in real time from car sensors is more and more popular. Processing data from these sensors is often computationally complex and done only on processors. So the intention was to check if it can be efficiently done on heterogeneous platform (FPGA + ARM). The objective is a working hardware-software car detection system based on LiDAR and vision data fusion. Project requirements are presented below.

Features-in-Brief

Following features of the system were assumed:

- When user plugs in Velodyne HDL-64 LiDAR and a camera to Zybo Z7-20, system should detect objects in real time and return this information to other car components,
- System would be adjusted to car detection,
- Processing LiDAR data to get object classification results,
- Processing vision data to get object classification results,
- Data fusion of LiDAR and vision classification results,
- Higher level analysis – e.g. object tracking.

Project Summary

The major components of the project are:

- LiDAR data processing,
- Vision data processing,
- Data fusion,
- High level analysis.

Data would come directly from sensors to LiDAR and vision data processing. Then, classification results go to data fusion component. High level analysis includes output data from three prior components to do tasks such as object tracking.

The first three project components were intended to be done in PL. The last one was intended to be made in PS. The language used in PL was Verilog. In PS, C++ language was intended to be used.

As input data, KITTI database was used. LiDAR is not a cheap appliance and KITTI provides all the necessary data to test the system.

LiDAR data is being processed in a pipeline, so real-time requirements are satisfied.

If the design is fully completed, it could be commercially used as a part of ADAS system. Incomplete project can be used for educational purposes.

The design is intended for heterogeneous platform. In its final form it is highly probable that Zybo Z7-20 would not have enough resources in PL to handle all parts of the design.

Digilent Products Required

The only required Digilent product is Zybo Z7-20.

Tools Required

To run the project, following equipment is necessary:

- Hardware:
  - Zybo Z7-20,
  - 2 x HDMI cable,
  - monitor with HDMI port,
  - PC computer (with Windows OS).
- Software:
  - Vivado,
  - Prepared image frames,
  - Prepared LiDAR data.

Design Status

The project is not fully completed. However, there are significant features than can be presented.

Following features are completed:

- Classifying cars based only on LiDAR data,
- Displaying classification results on monitor.

Following features are incompleted:

- Taking data directly from sensors – in the release, processed LiDAR data was taken as input,
- Classifying cars based on vision data,
- Data fusion,
- High level analysis,
- Sending LiDAR and visual data from computer to Zybo Z7-20 by HDMI.

The main factor that held us back from completing the project was time. The estimate time to finish the project is at least one month.

Background

Why This Project?

Nowadays, topics of ADAS (Adaptive Driver Assistance Systems) and autonomous vehicles are more and more common. Both big companies and ordinary people work hard to create efficient systems for such purposes. There are still more and more cars on the roads and therefore security systems are needed to assist drivers.
In this field, one of the most important things people devote their time for is object detection. A car, which is supposed to drive by itself, must know what objects are in its surround. There are three sensors which are most commonly used for such purposes. It is LiDAR (Light Detection and Ranging), RADAR (Radio Detection and Ranging) and camera. Data from this sensors must be acquired and properly processed to detect objects. Very common approach is to fuse data from a number of these sensors. It gives higher reliability of system output information. Each of this sensors has unique draws and backs. For example data from LiDAR are almost independent of weather and time of day but camera gives us information about colors. However, the fusion of data from both sensors will eliminate the disadvantages of each of them.
Data from sensors should be processed and the appropriate detection and recognition algorithm should be used. Often, when algorithm detects object efficiently, it is also time consuming. However there is very little trials to implement such algorithms on heterogeneous platforms. In this project, we want to show that it can be done and it can bring visible profits.

The project implementation can be divided into several stages. In the first one, the related work was gathered in order to isolate methods used in each step and to compare different algorithms in terms of effectiveness and suitability for hardware-software implementation. In the second stage, a software model was created. It was implemented in MATLAB environment. The model is used for testing selected solutions and as a reference for the designed system. All parameters of the used algorithm are stated in this model. In the third stage, the system was split into the hardware resources (PL) and the processor part (PS). Point cloud and image processing are carried out in hardware. The software part would include communication and data logging.

The first element of the system is data acquisition from LiDAR sensor and camera. The data was taken from a publicly available database (like KITTI or nuscenes.org) because we do not have access to a LiDAR sensor (due to huge cost). The second phase is LiDAR point cloud processing. The first step is preprocessing, which includes ground removal, filtration and background removal. The following step is segmentation that aims to divide the remaining point cloud to segments which contain potential objects. In the end, for each segment features are extracted and classification is performed. Result of each step in point cloud processing was presented on figure 1. In the vision part of the system, the first step is to project segments evaluated by LiDAR data processing to image plane and select the corresponding ROI. The ROI is extracted and scaled. The final step is feature extraction and classification (e.g. using HOG+SVM). The last part of the algorithm is data fusion. Classification score from both systems will be fused to obtain the final object detection probability. This approach was tested in MATLAB environment and in the Vivado simulation.

We have chosen this particular approach based on literature [1][5]. It had to be then improved and adjusted. It was tested in software model in MATLAB and gave satisfactory results.

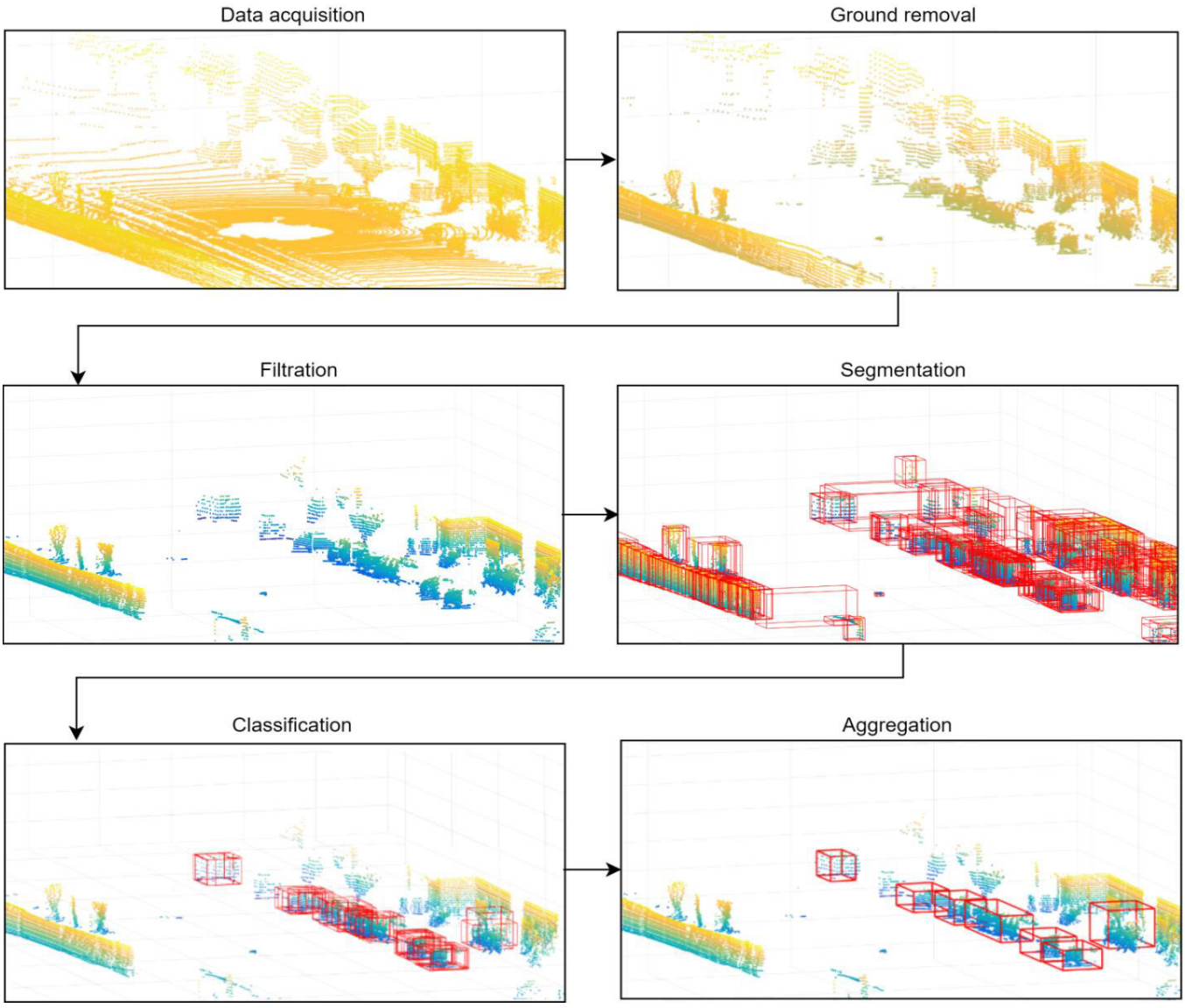

Figure 1. Result of each step in point cloud processing.

Theoretical background

LiDAR

The basis of this project is data from camera and LiDAR sensor. LiDAR are commonly used in such fields as: geology, geophysics, archeology, seismology, ground mapping or autonomous cars. LiDAR principle of operation is simple. These sensors, through a special optical system, emits the compound of the laser light with a specific wavelength and in a specific direction. Reflected from an obstacle, the beam is picked up by detectors located in the same device. Based on the time between sending the light and receiving it, the distance to obstacle is calculated.

In project we used data base KITTI Vision Benchmark Suite [2]. It provides the LiDAR and camera data gathered in urban and rural areas. Data base contains data from LiDAR Velodyne HDL-64E. This sensor is equipped with 64 canals and rotates with 10Hz frequency. Data is send by Ethernet in spherical coordinates.

SVM (Support Vector Machine)

SVM is a classifier that in the basic version divide data into two classes. Support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class, since in general the larger the margin, the lower the generalization error of the classifier. Hyperplane is described as:

$$\{x: f(x) = x \cdot \beta + \beta_0 = 0\}$$

Where:

β – normal vector to the hyperplane,

β_0 – hyperplane offset from the zero point in the space of features.

Function $f(x)$ may be treated as distance between considered point and hyperplane. Its sign depends on which side of hyperplane is x. Having a certain vector of features x, parameters β and β_0 we can determine class by the function:

$$G(x) = sign[f(x)] = sign[x \cdot \beta + \beta_0]$$

More information in [3] and [4].

Design

Features and Specifications

Final, assumed, version of the project should have features as listed in *Features-in-Brief* in *Introduction*. In simple words, it is car detection in vision and LiDAR data and tracking objects. From these, we managed to fulfill only classifying cars in LiDAR data. It can also serve as car detection system, but system with data fusion has more potential.

Design Overview

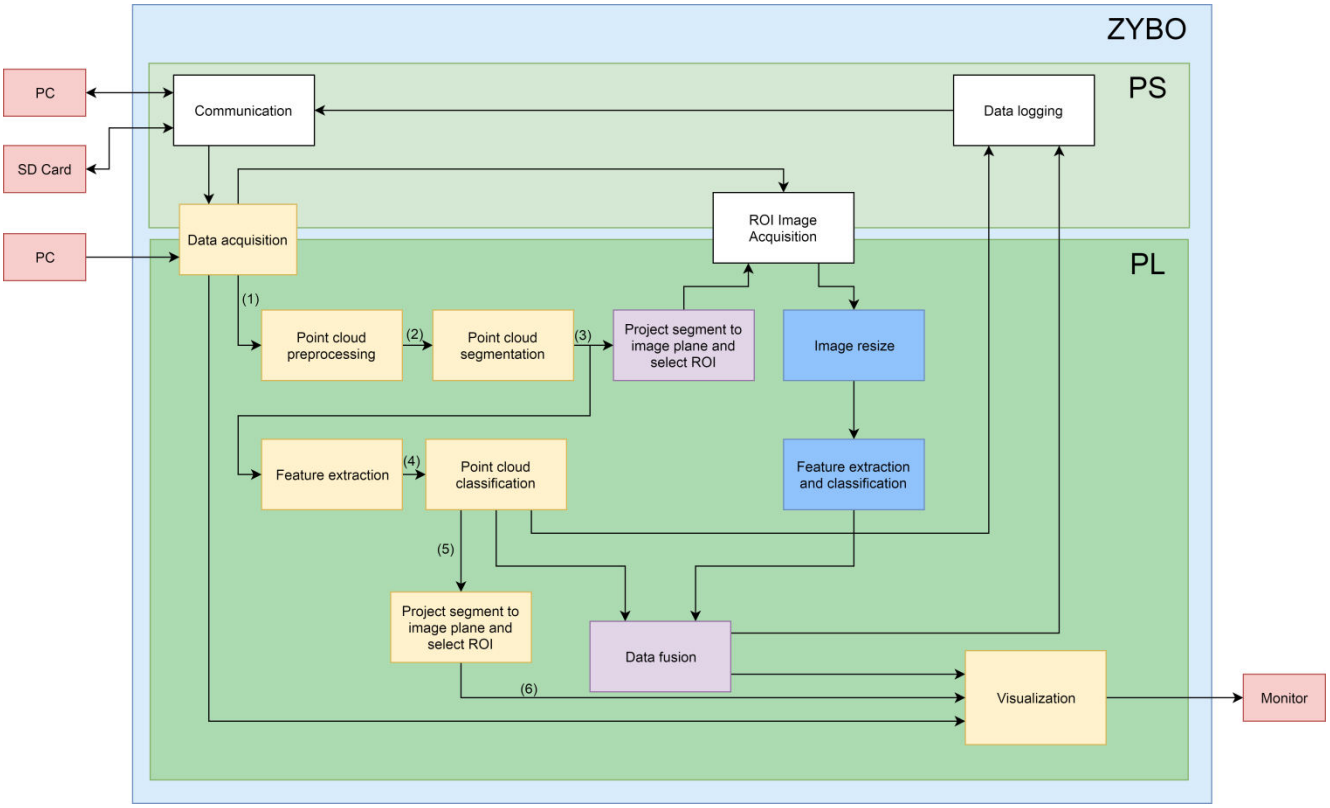

Figure 2. Top level diagram.

Top level diagram is presented on figure 2. It represents the whole assumed project design. However, not everything was finished. The design state of components are represented by their color:

- yellow – finished and taken into account in current release,
- violet – finished, but yet not used in current release (they cannot be used, because other modules are not completed),
- blue – started but not finished – due to the lack of time or some other factors,
- white – not started.

In below section, *Detailed Design Description* only modules marked yellow are described. Modules marked violet and blue are not described since they are incomplete or unused.
Module “Project segment to image plane and select ROI” is put twice in top level diagram. The reason is that the yellow one was added to present some results on monitor. The violet one has some other task, but similar behavior. It would be described in detail only once (in *Detailed Design Description*).

Table 1. Top level signals of the design.

Signal number	Description
(1)	Signal represents LiDAR data cells coming from PC – each cell consists of some number of points and a flag, which is high while points are being send (each point in one clock cycle)
(2)	Signal represents some cell features (e.g. intensity histogram), a flag which indicates if a cell was or was not removed during *Point cloud preprocessing* and a flag which indicated that data is valid. In one clock cycle, one cell features are being send.
(3)	Signal represents cell features, flag which indicated if a cell was or was not removed. It varies from signal (2), because cells are being send in packets of 9, with certain relation of neighbourhood. One such a packet of cells is called a segment.
(4)	Signal represents features for a whole segment. There are 30 features for each segment. Every feature is being send in one clock cycle. Beside this, there is a flag indicating that data is valid.
(5)	Signal represents classification probability for a segment.
(6)	Signal represents bounding box (in LiDAR cooridnates) for a segment.
(7)	Signal represents bounding box (in image coordinates) for a segment.
(8)	Image signal – one channel (gray) and synchronization signals.
(9)	Image signal – one channel (gray) and synchronization signals. It is image from (8) with marked bounding boxes.

Table 1 represents top level signals of the design. It takes into account only signals used in current release, because majority of other signals is not fully defined. Signal (1) will be described in detail in section *Input Data Description*.

Input Data Description

As input, data from KITTI database is taken. It includes both LiDAR and vision data. Vision part of the system is not completed, so only LiDAR input data will be described here.

LiDAR data consists of points. They are described by four coordinates:

- x – distance of point from LiDAR in direction of moving car,
  - positive values in forward direction,
  - type – real signed value,
- y – distance of point from LiDAR in direction of left/right hand side of driver,
  - positive values on left hand side,
  - type – real signed value,

- z – distance of point from LiDAR in a vertical direction,
  - positive values in upward direction,
  - type – real signed value,
- intensity – it measures how much light is reflected from the surface,
  - type – integer ranged from 0 to 255.

First three coordinates are perpendicular to each other.

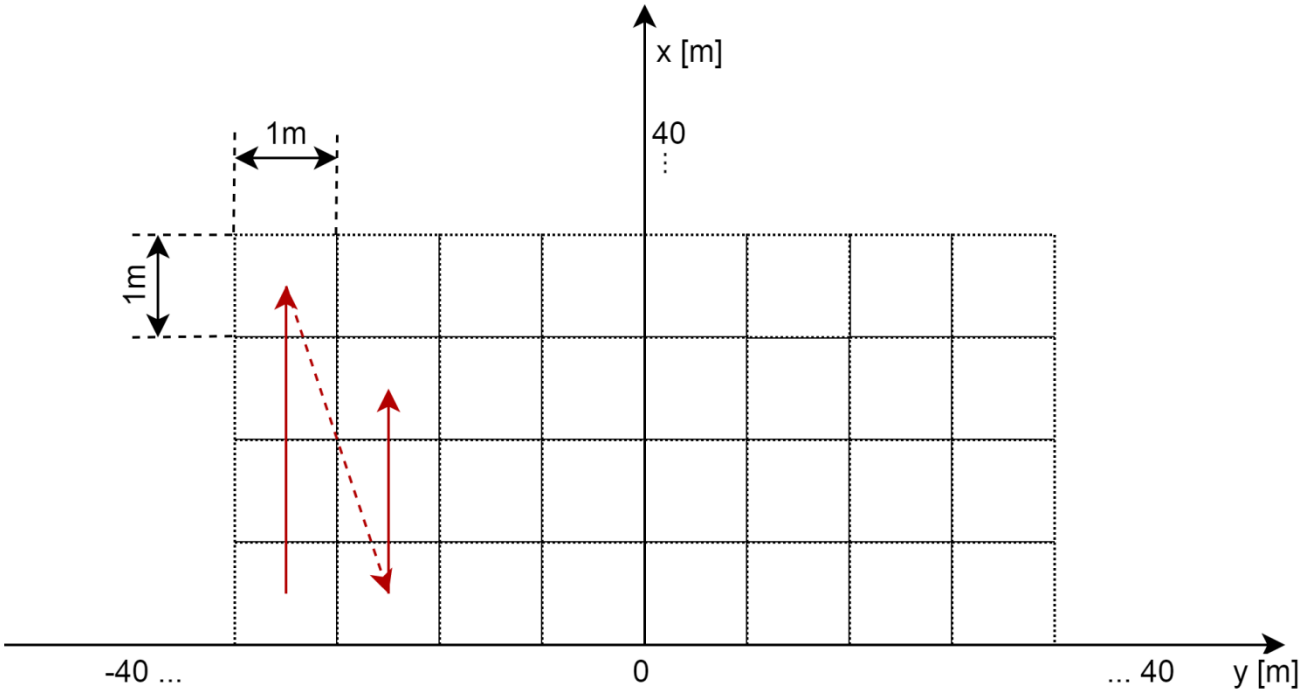

Figure 3. XY plane division into 1m x 1m cells.

XY plane is divided into non-overriding cells of size 1m x 1m (figure 3). Range in y direction is [-40m, 40m] and in x direction [0, 40m]. These values were chosen by experiment. Greater distances from LiDAR provide less detailed data and it is much harder to classify objects there. Data from back of the car (negative x's) was discarded in this release, because it is not present on image and cannot be displayed.

Sending a cell means that all of the points in cell are being continuously send in any order.

Cells are sent one by one to Zybo with constant time gaps – 1000ns. With too small gap time, some modules will not catch up with computations. The order of sending cells is showed by red arrows on figure 3.

Detailed Design Description

Data acquisition

Data acquisition consists of following modules:

- DVI to RGB Video Decoder,
- RTL HDMI data parser – parses HDMI data to obtain LiDAR points – its exact form depends on format of LiDAR data in HDMI.

Visualization

Visualization consists of following modules:

- ManageBBOXes – module which saves and replaces old bounding boxes with new ones, the output consists of 10 bounding boxes,
- Draw Bounding Box – draws on image 10 bounding boxes taken from ManageBBOXes module,
- RGB to DVI Video Decoder.

Point cloud preprocessing

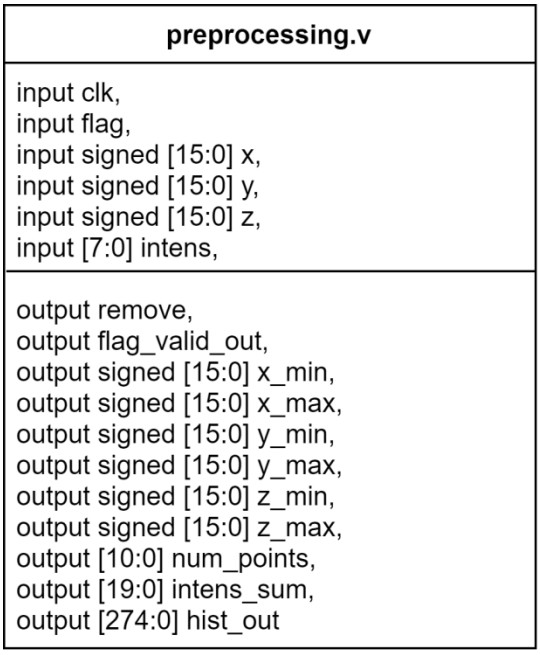

Figure 4. Preprocessing module interface.

Table 2. Preprocessing module input/output signals description.

Signal name	Width	Description
clk	1	Master clock (100 MHz)

flag	1	Flag indicating valid input data
x	16	X coordinate of one point in LiDAR data (s6c9f)
y	16	Y coordinate of one point in LiDAR data (s6c9f)
z	16	Z coordinate of one point in LiDAR data (s6c9f)
intens	8	Intensity of one point in LiDAR data (8c)
remove	1	Flag indicating that considering cell should/should not be removed
flag_valid_out	1	Flag indicating valid output data
x_min	16	Minimum of x coordinate of considering point (s6c9f)
x_max	16	Maximum of x coordinate of considering point (s6c9f)
y_min	16	Minimum of y coordinate of considering point (s6c9f)
y_max	16	Maximum of y coordinate of considering point (s6c9f)
z_min	16	Minimum of z coordinate of considering point (s6c9f)
z_max	16	Maximum of z coordinate of considering point (s6c9f)
num_points	11	Number of points in considering cell (12c)
intens_sum	20	Sum of intensity of points in considering cell (20c)
hist_out	275	Concatenation of 25 intensity histogram bins of points in cell (11c)

The first step of implemented algorithm is point cloud preprocessing. It contains ground removal, noise filtration and low foreground detection. This step is carried out by module which interface is presented on figure 4. The result is flag *remove* that indicates if considering cell should be or should not be remove. This flag is result of AND operation on another three flags: *rm_ground, rm_noise, rm_foregroud*. Figure 5 shows what these flags mean. *rm_ground* is set when mean of z coordinates of points in cell is lower that assumed threshold (*ground_thresh*). *rm_noise* is set when number of points in cell is lower than *noise_thresh*. *rm_foreground* is set when maximum of z coordinate of points in cell is greater than *zmax_thresh* or when substraction of z_max and z_min is greater than *diff_thresh*. All of this thresholds are saved as parameters of considering module.

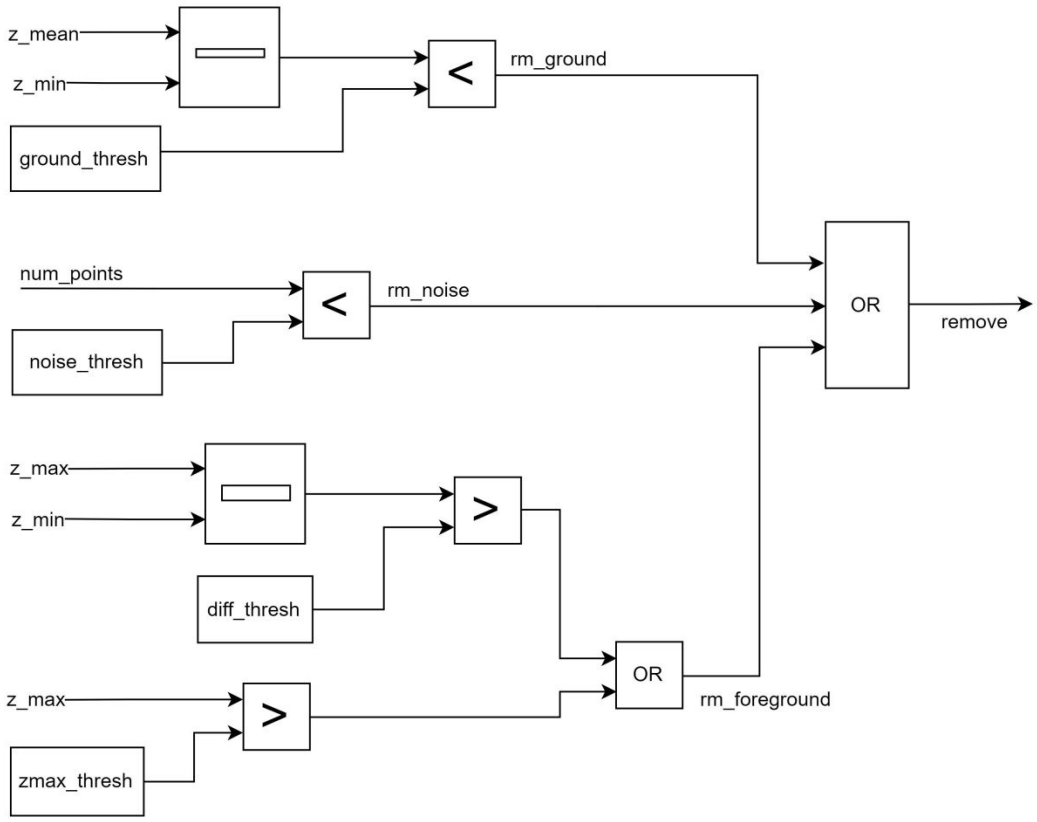

Figure 5. Flag remove computing.

The way of computing minimal and maximal coordinates is shown on figure 6. There is used register module, with *ce* and *rst* ports. Register for finding minimal and maximal value differ in initial value. On the same figure ways of computing number of points in cell, sum and mean of z coordinates are presented.

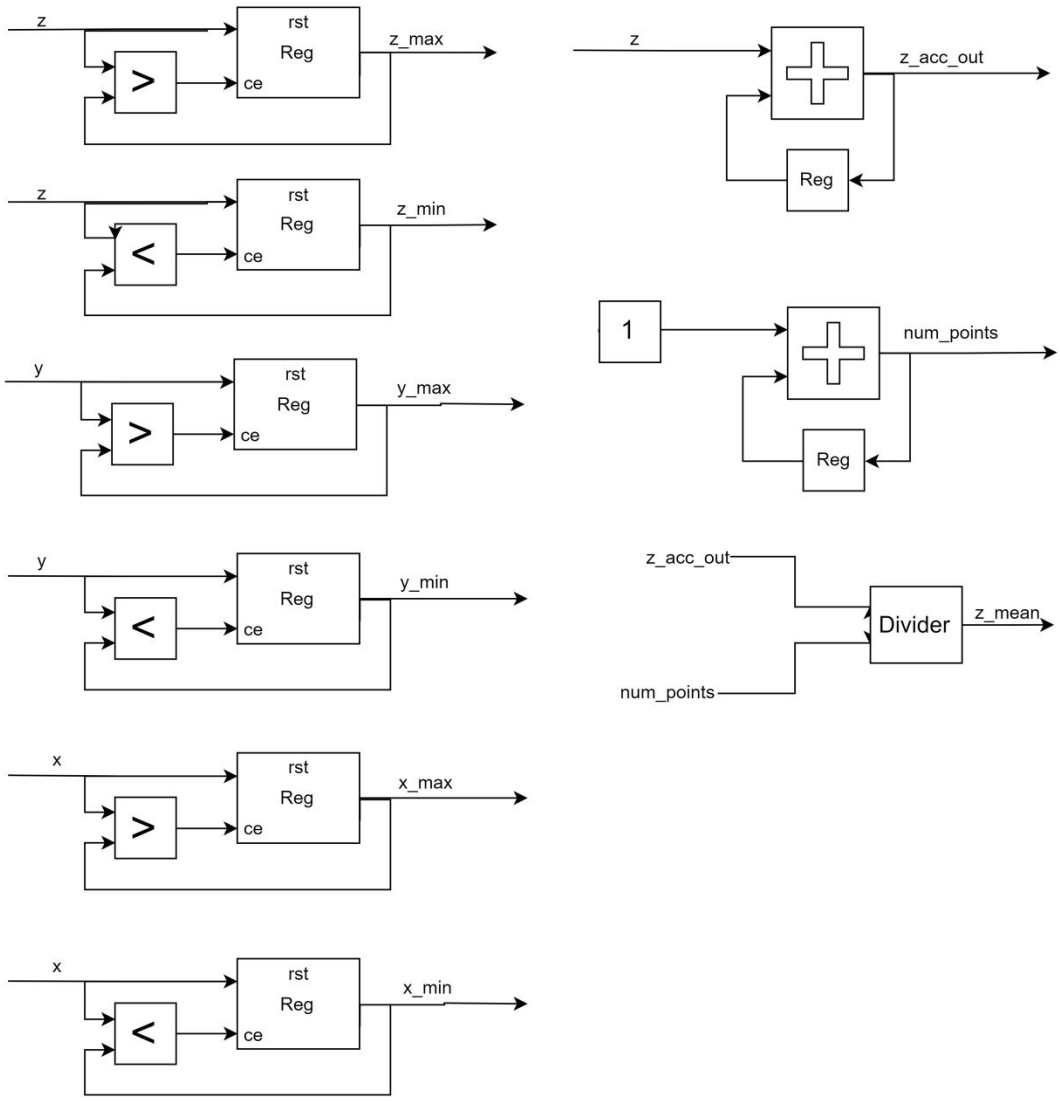

Figure 6. Computing minimal, maximal coordinates, number of points and mean of z coordinate.

Preprocessing module has also other functionalities apart from calculating *remove* flag. There are computed some values used in feature extraction step later. These are: minimal and maximal cells coordinates, number of points, sum of intensity of points in cell and histogram of intensity with 25 bins (figure 7).

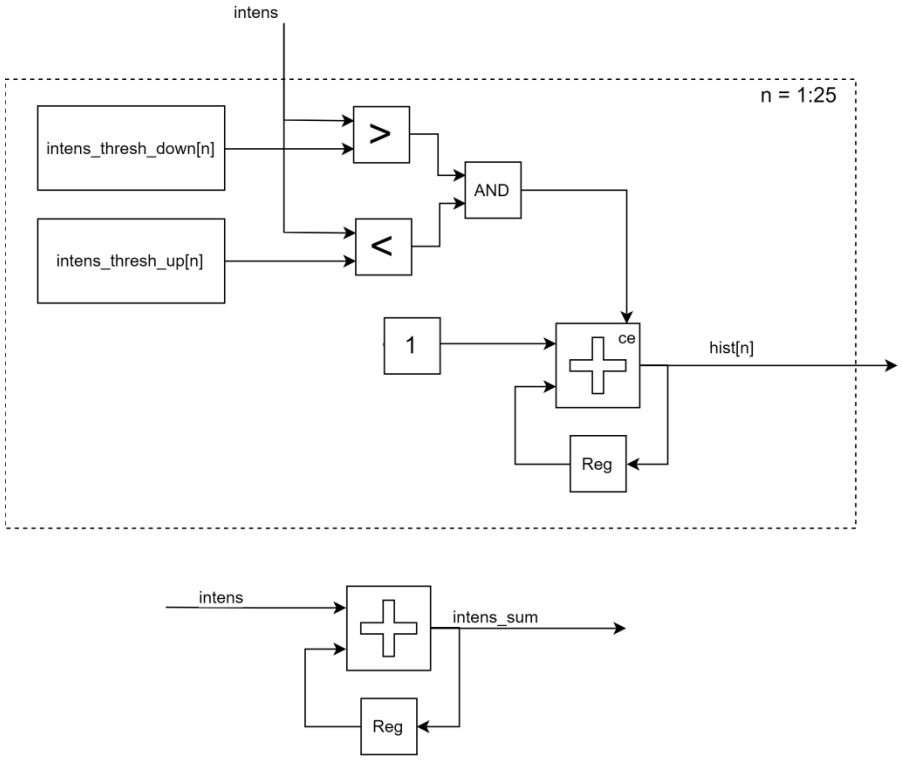

Figure 7. Computing sum of intensity and histogram .

Point cloud segmentation

Save to BRAM

save_bram_addres.v
input clk, input flag_valid_remove,
output [7:0] addra

Figure 8. Save to BRAM module interface.

Table 3. Save to BRAM module input/output signals description.

Signal name	Width	Description
clk	1	Master clock (100 MHz)
flag_valid_remove	1	Flag indicating valid input data
addra	8	Address of A port in BRAM IP

The aim of this module is to generate the address to save value in BRAM. Our approach assumes saving minimal and maximal coordinates of 1m x 1m cells, number of points, intensity sum and histogram and flag *remove*. This is all 403 bits. When *flag_valid_remove* is set this data is saved in BRAM under appropriate address. In BRAM at one time 160 cells are stored: a “rectangle” with 40 cells in x direction and 4 cells in y direction. Following cells come in x direction (in “columns”) and are successively saved to BRAM. When the first four columns arrive and the next is being send, the column address is overflowed – so the next data is saved in column of address 0, then one, two, etc.

Read from BRAM

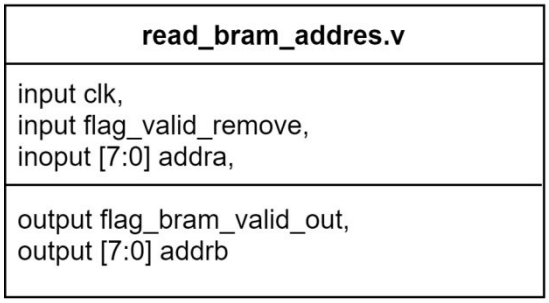

Figure 9. Read from BRAM module interface.

Table 4. Read from BRAM module input/output signals description.

Signal name	Width	Description
clk	1	Master clock (100 MHz)
flag_valid_remove	1	Flag indicating valid input data
addra	8	Address of A port in BRAM IP
flag_bram_valid_out	1	Flag indicating valid output data
addrb	8	Address of B port in BRAM IP

The aim of *read_bram_address* is to read cells from BRAM in ceratin order – to perform segmentation. Data is read in packets of nine cells. The manner data is read is presented on figure 10. Nine neighboring cells (blue box) are read in any order and sent in “one packet” to the following module. When the next cell is written to BRAM, next nine cells can be read – so that the center of blue box moves by 1 meter in x direction. When blue box comes to the end of x direction (center in 39^{th} cell) “column” it is moved in y direction by one meter and moved to the beginning of next x direction “column” (look at red arrows). Data can be started reading only if first two x direction “columns” and three cells of the next one are saved.

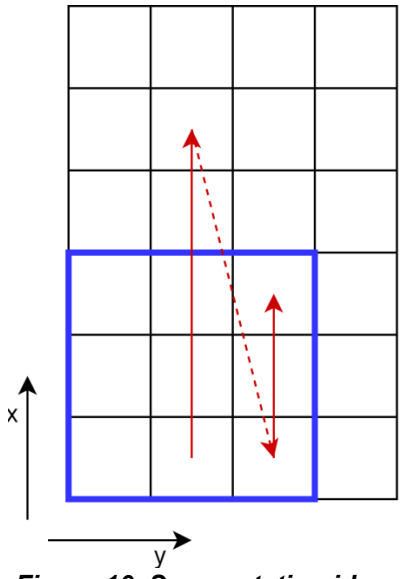

Figure 10. Segmentation idea.

Features extraction

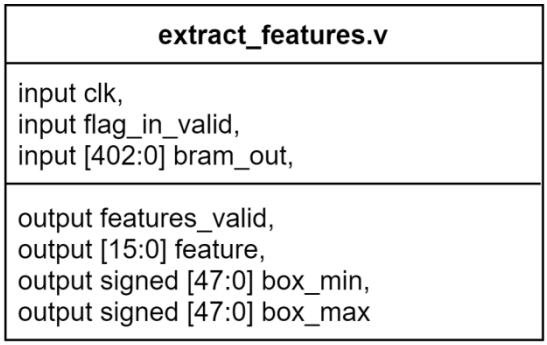

Figure 11. Feature extraction module interface.

Table 5. Feature extraction module input/output signals description.

Signal name	Width	Description
clk	1	Master clock (100 MHz)
flag_in_valid	1	Flag indicating valid input data
bram_out	403	Data saved in BRAM
features_valid	1	Flag indicating valid output data
feature	16	One of 30 features describing the segment
box_min	48	Concatenation of minimum x, y, z coordinates
box_max	48	Concatenation of miximum x, y, z coordinates

The result of *extract_features.v* module is 30 features sent in sequential clock cycles. Features are presented in table 6. In this part of implementation some values computed in preprocessing step are used. For example to calculate height, width and length of the segment there was used minimal and maximal coordinates for each 1m x 1m cells. This module receives data from 9 such cells so to

calculate 1-3 features (table 6) it is necessary to use register for finding the minimal and maximal coordinates for the whole segment. The same is about number of points in segments – accumulate number of points in 9 small cells. In similar way intensity features are calculated.

Table 6. Features description.

Number	Description
1	Height of the segment
2	Width of the segment
3	Length of the segment
4	Number of points in segment
5	Mean of intensity of points in segment
6-30	Histogram of intensity with 25 bins

Point cloud classification

SVM classification

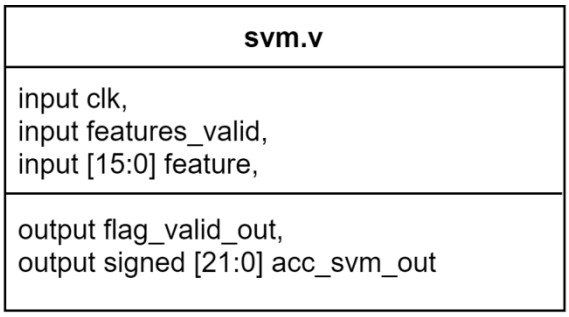

Figure 12. SVM classification module interface.

Table 7. SVM classification module input/output signals description.

Signal name	Width	Description
clk	1	Master clock (100 MHz)
features_valid	1	Flag indicating valid input data
feature	16	One of 30 features describing the segment (1c15f)
flag_valid_out	1	Flag indicating valid output data
acc_svm_out	22	SVM classification score (s8c13f)

Features vector is multiplied by the normal vector to the hyperplane. Hyperplane model (betas and bias) were determined in MATLAB during the classifier learning process. This model returns the distance of the segment from hyperplane. A distance sign informs that segment is car or not. The scheme of svm model is presented on figure 13.

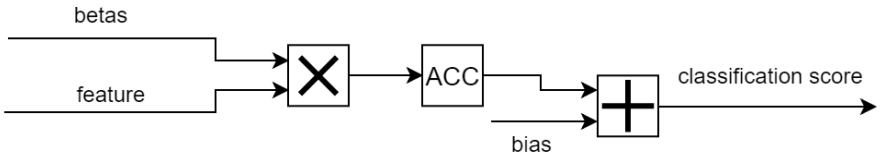

Figure 13. Computing of SVM score classification.

Calculate probability score

calc_score.v
input clk, input flag_valid_svm, input signed [21:0] acc_svm_out,
output valid_score, output [15:0] svm_score_out

Figure 14. Calculate probability score module interface.

Table 8. Calculate probability score module input/output signals description.

Signal name	Width	Description
clk	1	Master clock (100 MHz)
features_valid_svm	1	Flag indicating valid input data
acc_svm_out	22	SVM classification score (s8c13f)
flag_valid_score	1	Flag indicating valid output data
svm_score_out	16	SVM classification probability score (1c15f)

The final probability SVM score is calculated in module *calc_score.v*. Module returns probability that considering segment is a car. Probability is computed with use of sigmoidal function:

$$f(x) = \frac{1}{1 + \exp(Ax + b)}$$

A and b coefficients were obtained with MATLAB built-in function (*fitSVMPosterior*), after SVM learning process. From the range [0;1] 64 values (probabilities) with a fixed step were selected. They were saved to the LUT module. With the use of inversed sigmoidal function, *x* values corresponding to probabilities were calculated. They were the base for creating 64 sections of classification scores. So for every section in classification scores, there's a corresponding probability.

So the probability calculation goes as follows (figure 15): classification score is compared to 64 constants. Section number is obtained from encoder. The corresponding probability is computed. The final classification probability is (due to specific MATLAB behavior) one minus probability for the section.

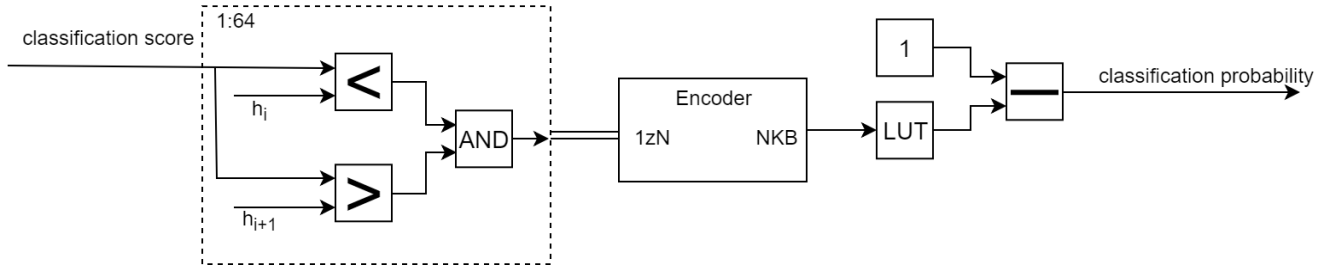

Figure 15. Scheme of calculating probability score.

Project segment to image plane

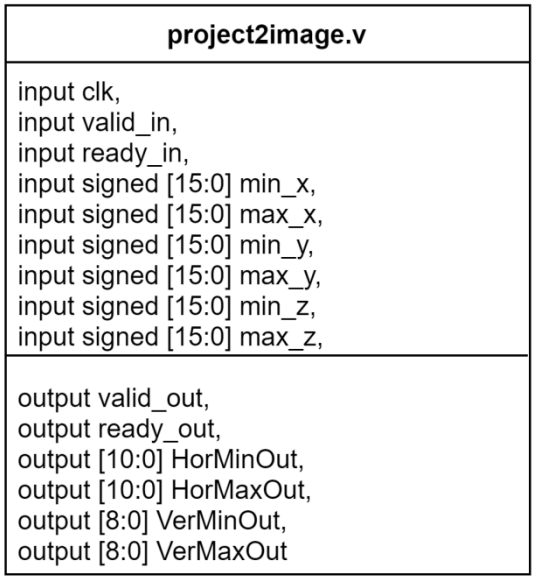

Figure 16. Project segment to image plane module interface.

Table 9. Project segment to image plane module input/output signals description.

Signal name	Width	Description
clk	1	Master clock (100 MHz)
valid_in	1	Flag indicating valid input data
ready_in	1	Flag indicating that the next module is ready to fetch the data.
min_x	16	Minimum of x coordinate of considering point (s6c9f)
max_x	16	Maximum of x coordinate of considering point (s6c9f)
min_y	16	Minimum of y coordinate of considering point (s6c9f)
max_y	16	Maximum of y coordinate of considering point (s6c9f)
min_z	16	Minimum of z coordinate of considering point (s6c9f)
max_z	16	Maximum of z coordinate of considering point (s6c9f)
valid_out	1	Flag indicating valid output data
ready_out	1	Flag indicating that this module is ready to fetch the data from the preceding one
HorMinOut	11	Bounding box minimum – horizontal direction
HorMaxOut	11	Bounding box maximum – horizontal direction
VerMinOut	9	Bounding box minimum – vertical direction
VerMaxOut	9	Bounding box maximum – vertical direction

This module projects bounding box in LiDAR coordinates to bounding box in image pixel coordinates. There are following steps to do it:

- generate all 8 points of LiDAR coordinate bounding box based on two edge points (minimal and maximal),
- project all of these points on image plane – multiplying them by hardcoded matrix (it can be obtained from calibration of sensors – in our case, it was prepared in KITTI dataset) – and

then, for each point, two from three obtained values have to be divided by the third remaining one,
- given image resolution (constant), check if all of the points are in image bounds – if not, discard the input bounding box – it is not fully present on image,
- compute minimal and maximal pixel coordinates from obtained set of projected points – return them as a bounding box in image coordinates.

The component first saves output bounding box in a FIFO and sends it to the next module when flag *ready_in* is in high state. When the component starts calculations, it sets *ready_out* flag to zero – for information that it is not able to get any data. Flag *ready_out* is set to one when the component finishes calculations.

Detailed Not-Completed Design Description

Discussion

Problems Encountered

From beginning to this moment of the project there were dozens and dozens of problems. They will be enumerated below:

- Problem with literature material – only a few designs were found that described LiDAR processing in a useful manner. Only one was done on FPGA (+ARM), but it was not described well from implementation and experimental point of view. However it has given a few ideas how to preprocess, segment and classify the data (with no clue how to do it on FPGA). Literature describing data fusion from LiDAR and camera was not easy to find either. There was even less materials than for LiDAR. Given no good explanation how to do it, we had to fuse these approaches and come with a new way to do it.
- Problem with processing LiDAR data in spherical coordinates on PL – raw data from physical LiDAR comes in spherical-like coordinates – user has to make some simple operations to make it spherical. So the natural way to make preprocessing and segmentation is to use spherical coordinates. However, it was nowhere described. So we firstly modified literature approaches to spherical coordinates (it took some time and effort). In software model (in MATLAB) the results were satisfying. We moved to implementation in PL. We have made preprocessing with much effort. The next thing to do was segmentation. We assumed segments should be approximately of size 3m x 3m (they should be of car-like size). We came with a complex way to divide some spherical region to segments of size approx. 3m x 3m. Points in each segments should be first collected and then sent to feature extraction module. The biggest problem here (and unsolved) was point collection. It is impossible or almost impossible to do it with 630KB of BRAM. Much more is necessary. We tried to use PS to do it, however in PL we used clock 100MHz (lower frequency would cause system not going in real time) and it is too fast for generation of interrupts to processor with 667MHz clock. We were stuck at this moment and started the project from scratch with Cartesian coordinates.
- Processing LiDAR data in Cartesian coordinates on PL – it is “unnatural” coordinate system for LiDAR, however easy to use while processing LiDAR data on PL. The main problem here is to transform coordinates so that following cells in Cartesian coordinates are filled with points and real time requirements are satisfied. We did not resolve this problem either. In our release we use data prepared by KITTI.

- Problem with managing product of several comparisons – when we were doing spherical approach, we wanted to determine in what cell the point actually is (there were approx. 100 cells). So we made a net of comparators – if one of them returned one, the point belong to that cell. However, it was necessary to determine the number of that comparator. To do it in pipeline we came with idea to use encoder. At post-implementation timing simulation we have seen that it takes 20 ns to establish encoder response. With clock 100MHz it was far too long, so we increased latency. We came up with design that is easily physically separable and we inserted there some flip-flops. The time shrinked down to 5 ns and latency was 5. We have then used this trick several times.
- Problem with sending data from PC to Zybo by HDMI – the problem is synchronization of LiDAR and visual data. We decided to use only one image channel and use two other to send LiDAR points. To pack all LiDAR data alongside with one image, resolution 1920 x 1080 was necessary to use. However, this solution is not well tested and it could error prone. If LiDAR data for some point cloud taken from KITTI database would be too big, higher resolution is necessary or LiDAR data would have to be cut.

Engineering Resources Used

Engineering time: 800 hours (not completed yet).

Marketability

This project has high marketing potential. In its final form it would be a product ready to sell. ADAS systems and autonomous cars are precursor issues in modern technology. Object detection systems based on car sensors are inseparable parts of ADAS and autonomous vehicles. A lot of companies struggle to develop their own systems. FPGA + ARM duet has potential to process LiDAR and vision data with high speed – in real time and as far as we know, only a few people work on such ideas.

References

[1] Q. Cabanes i B. Senouci. „Objects detection and recognition in smart vehicle applications: Point cloud based approach”. W: 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN). 2017, s. 287–289. DOI: 10.1109/ICUFN.2017.7993795.

[2] Andreas Geiger i in. „Vision meets Robotics: The KITTI Dataset”. W: International Journal of Robotics Research (IJRR) (2013).

[3] John Platt. „Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods”. W: Adv. Large Margin Classif. 10 (June 2000).

[4] Trevor Hastie, Robert Tibshirani i Jerome Friedman. The Elements of Statistical Learning. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[5] W. Jun i T.Wu. „Camera and lidar fusion for pedestrian detection”. W: 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR). 2015, s. 371–375. DOI: 10.1109/ACPR.2015.7486528.