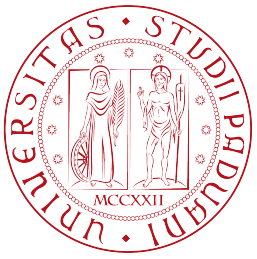

Università degli studi di Padova
Dipartimento di Ingegneria dell'Informazione
Master Degree in ICT for Internet and Multimedia

No Title - Hopefully a thesis someday

Relatore
Prof. Marco Giordani

Laureando
Rossi Valentina
matr. 1206473

Anno Accademico 2019/2020

Contents

Chapter 1

Related Works

In the last few years the concept of autonomous driving has begun to dominate automotive headlines, also thanks to the recent innovations brought by Tesla and other car manufacturers, which introduced systems for secure driving as a first form of autonomous driving.
The first real try to have an automated car was done in 1989 with CMU's ALVINN [1], with a car that taking images from a camera and a laser scanner produced as output the direction that the car had to follow in order to stay in lane, all using neural networks. It appears that for the following years the idea of self-driving cars was abandoned, probably also due to the lack of interest in neural networks in the following years, due mainly to limited computational power. As reported in [2], ANN regained popularity in the 2010s, thanks to several contribution that introduced techniques such as dropout, ReLu layers and data augmentation, that helped in solving the over-fitting training problems. It is then not a coincidence that in the same years where ANN regained popularity that DARPA, Defense Advanced Research Projects Agency of the United States government, organized three car autonomy challenges [3].
In more recent years, traditional car companies such as Ford, Volkswagen, Audi, Nissan, Toyota, Tesla and others announced their intentions in researching the autonomous vehicle manufacturing and research: in particular Testla in 2015 brought their autonomous car, and in 2017 Audi announced the first level 3 autonomous car, their A8. Moreover, companies like Google and Waymo tested driver-less car using sensor fusion, respectively in 2014 [4], [5].
Of course, there are different levels of automation. That is why the Society of Automotive Engineers (SAE) developed a classification system for defining six different levels of driving automation for motor vehicles. The first three require a driver to perform all or part of driving tasks, while in the last three the Automated Driving System (ADS) performs all of the dynamic driving tasks. In short:

- Level 0: No driving automation.
- Level 1: Driver assistance. (E.g. cruise Control OR lane centering)
- Level 2: Partial driving automation. (E.g. cruise control AND lane centering)
- Level 3: Automated driving (conditional). (E.g. automated driving in dense freeway traffic at low speeds)

- Level 4: Automated driving (high). (E.g. automated driving within a city center using geo-fenced location)
- Level 5: Automated driving (full). (E.g. automated driving everywhere)

Intermediate levels are defined as well.
To use neural networks to perform the task of autonomous driving, of course data needs to be gathered. This data can either come from sensors or from other cars. As for the latter option, it will be discussed in the following section.
Sensors can be of several different types, such as GPS, radars, cameras and scanners. In particular cameras and sensors are used for Object detection often combined. However, since laser scanners, such as LiDars, are usually really expensive, in the section below about Object Recognition some techniques will also comprise the use of stereo cameras, instead of laser scanners, to provide 3D point cloud, since they are cheaper and can work in real time, despite having lightning issues. Of course, several sensors can be placed on a vehicle at once. To couple information between lasers scanner and other passive stereo cameras, which nowadays is a popular solution to get dense depth images in real time, some methods have been proposed. One is reported in [6], which is also used on the autonomous vehicle "Pod", operating in urban environments. This paper however does not talk about the calibration method: at the time the article was published, it was done manually by an human operator. This calibration process obviously took a long time to complete. To speed it up, and to automate it, [7] proposed an algorithm to calibrate a LiDar and stereo camera setup.[1]
It must be noted that, as reported in [2], sensors have their limitations. They can be unreliable and suddenly stop to work, as it has reported to be happened to several Tesla's car, resulting in death accidents, as in [8], [9]. Also, sensors have restricted perception capacities and heavily depends on infrastructure's conditions. They also cannot detect accidents and traffic jams, which is a necessary condition to do autonomous driving in safety: to keep a steady pace the car needs to know traffic conditions in the path. That is why it is needed to have cars communicate with each other and to know information about the infrastructure itself. This is solved with networking.

In the sections below, we are going to discuss the state-of-the-art of Networking, 3D cloud Compression and Object Detection Technologies.

1.1 Networking Technologies

We saw above that the autopilot of an autonomous driving system relies on a large amount of data, provided by sensors and inter-vehicle communications.
As [10] highlights, the content that produced and required by autonomous cars show a strong local validity, that is, the content generated by a car will have a spatial scope of utility, and only nearby vehicles will represent the group of potential consumers. For example, to make this concept more clear, consider a car encountering a construction site on its path, which causes the car to stop, of to slow down. The information provided by that car will concern only that spatial area where the construction site is located, and only cars in that area which have that street on their path will require that information. Another property that the generated content must have is time validity:

[1] should i go more in depth on those two?

some information will have to be discarded at some point. In fact, going back to our example, the construction site at a certain point in time won't be there any longer, and the information about it will become obsolete. This also helps with the scalability of the network: useless data must not be uploaded on the Internet, since the amount of content will be already considerably high. This must also mean that there must be some discerning on which kind of content can be uploaded, and what has to be stored in-vehicle.
Another feature of vehicle applications is that they are more interested about the content itself than its provenance. This can easily be inferred by the spatial properties of the physical world: cars are geometrically scattered and travel at high speed.
After all these considerations, as said in [10], networking must be content centric, with vehicles asking information about position, speed, direction, traffic information and such to other vehicles in the area. For this reason, Information centric Networking (ICN) was conceptualized as a general form of communication architecture in Vehicular networks. To materialize the importance of the content with respect to its source, ICN uses content names instead of IP addresses. [2]
Besides other vehicles, content can come also from sensors placed in infrastructures, such as smart dust components, RFID tags and other embedded microcontrollers. All together they form the so-called Vehicle Grid, defined as "*an intelligent road infrastructure analogous to the energy grid for intelligent power generation ad distribution*" ([10]). Its instantiation would be the so-called Vehicular Cloud, that comprises the protocols and services required to operate the grid efficiently.

The most prominent cloud computing models is VANET-Cloud, presented in [11], and applied to ad-hoc vehicular networks. A VANET is defined as a set of nodes, i.e. vehicles, that are considered to be mobile and moving according to determined mobility pattern and fixed entities, called road-side units (RSUs). VANETs must be able to support vehicle to vehicle communications (V2V) and between vehicle and infrastructures (V2I).
VANET-cloud's architecture is subdivided into two sub-models. The first one is based on the conventional cloud, offering software as a service, infrastructure as a service and platform as a service to the nodes, while the second sub-model consists of a temporary cloud, made of clusters of vehicles, used to expand the fixed cloud and provide more services. Specifically, the permanent sub-model is made of stationery and virtualized data centers, interconnected with traditional networks, that offer various software, applications and platform services to clients.
Likewise, the temporary sub-model can also be stationary and mobile: in the stationary case, vehicles, passengers and RSUs lend their computing resources to other vehicles while being in an off state. In the mobile case, they can add sensing information as a new form of service. [3]
Being a relatively new architecture, there are several challenges that VANET-cloud must still face and solve. First and foremost, security and privacy issues, in particular with the temporary sub-model. The model should focus in providing data integrity, controlling the data access and avoid data loss, all assuring confidentiality of the data users.
Another issue is to coordinate the aforementioned two sub-models, also taking into account the limited lifetime of vehicles in the network. This also leads to another challenge, that is interoperablity and standardization, to allow all manufacturers of all the entities involved to be able to use the

[2] Should i add more about concrete implementations of ICN, like NDN, and the presented routing algorithms?
[3] in VANET there are also three layers, client layer, comm. layer and cloud layer. Should i describe them? i thought that for this section a simple overview was enough, since the focus to me should be the Object Detection Section

cloud.
[4]

1.2 Compression of Point Cloud Technologies

As seen above, with autonomous driving systems, there is the need of exchanging information among vehicles. Moreover, with the spread of 3D data acquisition due to the accuracy and resolution of the available technologies, information can easily consist of very dense point clouds, representing the surrounding world. In particular, the transmission of these point clouds often is involved in remote data processing, therefore requiring real time compression, to save time and bandwidth at the same time.
Several are the real-time point-cloud compression approaches presented at this time: in this section we will present some of them, which were also made specifically taking into account the needs of autonomous driving systems and the kind of sensors used in the context.
A first approach was proposed in [12], with a cloud stream compression system that could be adjusted to match different resolutions and accuracy requirements while allowing to encode at the same time additional features, such as normals and curvature values. The system is based on octree-based point cloud compression, which is commonly used when spatial decomposition of the point set is required, and when the data to be compressed is sparse, as it is the case with point clouds generated by laser scanners and stereo camera systems. Since there will e below other compression techniques based on octree, it seems necessary to explain briefly how it works in general. An octree is a tree data structure, where each node represent a certain volume of area and has up to eight children. When the volume corresponding to a certain node is empty, the node is classified as non-existent and it referred to with a NULL pointer. The depth of the tree can be specified according to the required precision needed.
Then, to encode the data, to each node a bit is assigned, so the child configuration can be expressed within one byte. When reading the encoded stream then, the first byte specifies the number of bytes that are direct children, and their position specifies the voxel, i.e. a volumetric element associated with a region of space, occupied. This general approach allows for a compact spatial point distribution, but the main idea explained in [12] is to exploit also the temporal redundancy of adjacent frames to encode just changes with the point distribution. This was done using a double buffering octree scheme that adds a second octet set of child pointers to every node, interleaving these two sets of pointers. Then, when a new child set of nodes has to be created, a lookup to the preceding point buffer with the current one is done, to check if there are no corresponding references previously processed. In that case, a new node is indeed created, otherwise the reference is simply adopted and the new buffer will be initialized to zero. Lastly, to obtain the changes of the branches, an XOR operation between the two bytes representing a node is done. This kind of compression is obviously lossy.
The point cloud then can be reconstructed by extracting the centroids from the leaf nodes. For high

[4] Would it be too much also adding protocols use to send info for inter-vehicle communications?like IEEE 802.11 family of technologies, or Bluetooth? And also clustering techniques to manage clustering, such as platoons? I read about it, but i fear it might be out of topic a bit

resolution online compression they also propose to limit the resolution and append some additional information, such as the distance between each point and the voxel with smallest dimensions. This method reportedly increased performance by 34%.
Another, more recent, compression system using octree based system is the one proposed by [13]. In this paper the approach is however quite different since the point cloud, which is assumed to be produced by a LiDar scanner, is compressed into a text file, that is then transmitted and reconverted to the original data structure, and merged with all previous files. No other specific details were given by the paper other than using the compression technique provided by the Point Cloud Library, and that this method was tested with cars at 30km/h, allowing to send data capture by LiDar sensors in real time.
The last article in this group that used octree-based compression system is [14]. Their system is a complex hierarchical region-adaptive transform designed for color signals on 3D point clouds. They report, in fact, that the general idea was to use the color of a level of the octree to predict the color of nodes of the next level going in a bottom-up fashion. By doing so voxel are also grouped together in larger ones in three steps, across all three dimensions.
In short, to each voxel in the tree the proposed transform associates a coefficient $g_{l,x,y,z}$, that represent the average color of the voxel at level l, and a coefficient $w_{l,x,y,z}$, that is the number of voxel grouped to generate voxel $g_{l,x,y,z}$. Then, at each further grouping, a transform is applied to compute the next g coefficient and $h_{l,x,y,z}$, the high-pass coefficients that are actually going to be encoded and transmitted. Before encoding, a uniform scalar quantizer is applied to coefficients.
Results on this method have reported to outperform other method in several tests, while also being less computationally heavy.
All these methods reported above have also in common, besides the octree-based approach, the high computational effort required. To address specifically this complexity issue, [15] proposed a real-time out-of-core compression system that could also support incrementally acquired data, partial decoding and sub-sampling. Their approach uses voxels of specified size that are individually compresses. Specifically, objects in the point cloud are grouped in clusters, which are here represented by patches, parametrized over 2D domains. Over these domains, height and occupancy maps are defined, i.e. 2D bitmap images of predefined resolutions, that account, respectively, for the offset from the original geometry and holes in the point cloud. To perform this decomposition of voxels in patches, [15] proposes two approaches: the first one, the simplest, is to do a one-to-one correspondence between voxels and patches. It is very simple and achieves good data compression rates, but for a better quality compression, they also propose a data-adaptive approach that uses octrees to generate sufficiently small clusters.
Each of said patches is characterized by its position, orientation and size. The patches' normals are computed via Principle Component analysis. Height and occupancy maps are used to transform clusters' points into local coordinate systems which yield, for each height, the position in the height map. The corresponding position in the occupancy map is set to 1. In case several points are mapped in one pixel, the height's map value at that position is set to be their average. After this operation, all non-integer values are quantized, as well as height map values.
It is reported that this method achieves a compression performance of 22.7 million points per second, which could be suitable for the high rate requirements of autonomous driving applications.
The last compression approach that will be hereby reported, was specifically tailored to target the

compression of LiDar data acquisition for automotive application, with the goal of optimizing on-chip storage capacity and link bandwidth. In [16] they propose to use low-level compression approaches, to be applied during data receiving to achieve the aforementioned goal. No in-depth details of analysis were reported, but they suggested to use in particular to use the Lempel-Ziv-Markov chain algorithm ([17]) or the Golomb-Rice encoding ([18]) as compression methods. The former is a lossless dictionary-based compression method that achieves over 50% compression ratio, while the latter is based on the relation between probability and size of values.
What is actually interesting about this article, besides the initial idea of applying data compression during acquisition already, is the use in this context of Compressed sensing, that is a method which samples the signal at a rate slower than Nyquist limit, in case of sparse sampling domain, which could be the case with LiDar acquisition. [5]

1.3 Object Detection Technologies

Up to now, this chapter only reported compression techniques and network proposal to transmit information obtained from cars. This information, that it was often said to be likely coming from 3D scanners such as LiDars, has to be elaborated, to go from a general point cloud, to actual data regarding objects on the road and their position.
Before talking about the techniques used to perform the Object Detection and Localization task, it is important to talk about the KITTI benchmark suite [19]. KITTI is an extensive dataset with realistic non-synthetic images and ground truth, captured with a Velodyne (a LiDar sensor). It also provides information captured with a GPS/IMU system, that compensate the egomotion in 3D laser measurements. Ground truth is computed registering 5 frames before and 5 frames after the one of interest, using ICP algorithm. This dataset is suitable for evaluating both 3D Object Detection/Orientation Estimation and Stereo Matching, and outperforms other datasets thanks to its high resolution. Moreover, it presents three difficulty levels.

A crucial part of autonomous driving is to find and categorize road intersections in advance. In [20] a real-time intersection detection approach was proposed, and it used 3D point clouds obtained using a LiDar scanner. A drawback of this approach is that it only do road intersection recognition: in the preprocessing phase of the data, in fact, all objects that could act as obstacles are removed from the grid: this includes other vehicles and pedestrians as well. It would be arguably not efficient to have separate algorithms for object detection and intersection recognition, since it would be more computationally costly, or in general require more hardware capacity on the vehicle. However, it is still of interest to see how this problem was solved, to integrate this solution in other, more general, approaches.
The preprocessing mentioned consists of five steps. In short, as a first thing a grid map with fixed size cell is created. The variance of the elevation is computed for each cell, and based on that and on a threshold, points in the grid are set either to 0 (if below threshold) or 1 (otherwise). Then cells whose 4 connected regions are 1, are grouped into a new region, rebuilt to surround a cube. Using the cube's dimensions, pedestrians and vehicles are detected: corresponding cells will then be removed from the grid.

[5] I am not really sure to keep this article. It is just ideas, they did not do anything

After preprocessing, features are constructed using a beam model. The beams have all the same launching point, within the adaptive distance in front of the car, and they are spaced equally with 1 degree between each other. The higher the speed, the longer the distance between the launch point and the car is. Beams then scan linearly from launching point: they can be either blocked by an obstacle on the grid, or continue unlocked. In this latter case, its length will be limited by a certain constant. Then, the collected lengths will be used to detect road segments and intersections, since they present different lengths. Feature classification is then performed using labeled classes and applying Support Vector Machine. Total accuracy for classification between road segments and intersection is reported to be 93%. After this first detection, it is possible to classify between T-type or +-type intersections. Exact total accuracy was not reported, but said to be between 80% and 90%.

A more general Object Detection approach was proposed in [21]. The network proposed, called MV3D, takes multimodal data as input and predicts the full 3D extent of objects in the space. MV3D is made of two parts: a 3D Proposal Network and a Region-based Fusion Network. The Proposal Network utilizes a bird's eye view representation of the point cloud to create candidate boxes, and the fusion network extracts features by projecting the proposals to the feature maps.

Before going through the two sub-networks, the point cloud is encoded to form two representation: bird's eye view and front view representation. [6] The first part is then the 3D proposal network, which is based on the Region Proposal Network, that takes as input the bird's eye view representation. From that, 3D box proposals are generated from a set of 3D prior boxes, that are designed by clustering ground truth object sizes in the training set. These are then cleaned by removing boxes corresponding to background or are considered empty. To combine features from different resolution, then Region Of Interest pooling must be used: it is a step of the Convolutional Network used in the Fusion Network, and it performs a max pooling on inputs of non uniform size to get fixed size feature maps. After that, the Fusion Network projects the 3D proposals on the bird's eye view space, the front view space and RGB space. A deep fusion approach then fuses multi-view features in a hierarchical fashion. From the fusion features, from the proposed boxes, the network regress to oriented boxes. These are then re-projected on the bird's eye view, and IoU is computed to remove redundant boxes. Lastly, a normalization layer is added.

Results presented in the proposed paper shows that the training and testing of this approach on the KITTI benchmark has proven to give out better results than several other Object Detection Network.

Another promising approach was that of [22]. You Only Look Once is a simple and incredibly fast algorithm that achieves more than twice the mean average precision of several real time systems. Conversely to the method proposed above, YOLO looks at the entirety of the image during training and testing time, and simultaneously encodes contextual information, as well as learning generalized representations of objects.

YOLO unifies the different steps of object detection into a single neural network. This network divides the input image in a grid, where each cell containing an object is then in charge to predict bounding boxes for said object. With the bounding box, the cell must also generate a confidence score associated to it, that is computed based on how much the system believes that the box is accurate and contains an object. Of course empty cells will have a confidence equal to zero. Overall the prediction is made of five components: (x, y, h, w, c) with c confidence, h, w height and width of

[6] Should explain also how they are encoded?

the box relative to the whole frame, and x, y the box's center coordinates.
Based on this method, in [22], two implementation versions have been made. The first one uses 24 convolutional layers with 2 fully connected layer. The other implementation is a faster version with just 9 convolutional layers.
By author's own admission, YOLO has some drawbacks. First of all, since cells can only predict up to two cells, there is a limitation on how many close objects can be detected. Moreover, is has issues to generalize objects in different aspect ratio than what originally learned.
When compared to other real time systems YOLO has proven itself to be quite significantly faster, but it present less accuracy than Fast R-CNN. It is however better than the latter in terms of background detection: the authors then proposed an hybrid algorithm that uses YOLO and R-CNN together to check the bounding boxes predicted, keeping just the ones that overlap. This version resulted in an increased accuracy level.
Another approach is the one presented in [23], where the authors used Q-Learning in deep neural networks, and testing the results on a vehicle using sensor fusion. Q-Learning is an action/reward system based on a Markov Decision Process. When an action, that is drawn from a set of allowed ones, is made, the transition to a new state, caused by said action, is stored, along with its reward. This reward is estimated at each state by the Q-function, and then maximized. These stored action-state transitions are then divided in mini-batches and used at random during training. In the paper's method the Q-function is approximated with a fully connected CNN. As mentioned above, fusion sensor is used, thus there are two types on input: one coming from LiDar, one coming from a stereo system. They are processed separately with convolutional layers and successively fused using a flatten layer.
The set of defined actions that can be chosen by the agent during training is made of 5 options, each of which assigned with a reward, that in some options is computed using the system's speed as a parameter.
Conversely to previous method, this one was tested using Unity, and no comparisons with other methods have been provided. This approach however showed to be able to handle a real-time urban environment.
Up to now all deep neural network shown used CNN, so networks made of convolutional layers supported by fully connected layers at the end. However, in [24] it was chosen to use fully convolutional networks, based on the performances obtained when applied to 2D object detection tasks. The method proposed takes as input voxelized 3D data from LiDars or stereo camera systems and it outputs the objectness and the object shape in the 3D space. These two tasks are outputted as two separate maps: the first one determines whether a region belongs to an object and the other one predicts the coordinates of the bounding box. For each map a loss function is defined: an objectness loss and a bounding box loss. [7]. Only region with positive objectness are selected, and then the box predictions corresponding to these regions are selected.
Encoding on input data is done by discretizing the point cloud on square grids. Then the objectness region is defined a sphere centered in the object center. All points inside of it are then labelled as foreground. The predicted box is thus defined as offset with respect to the object center. As most of the other approaches, it is tested on KITTI and compared to other approaches. As all the other articles reported, it proven itself to be the fastest of the category of algorithms using similar ap-

[7] Should i write the expression?

proaches. For previous articles only results on the average precision achieved on the Hard mode of the KITTI benchmark were provided, so here, even though all levels results were presented, for a fair comparison only the hard mode one will be reported. Similary to others here reported, it is said to achieve an average precision of 79.2%.

From this point on, a separate group of deep neural network will be presented. This category of methods chose to train their ANN end-to-end, namely, from input image to steering angle. The output of these networks won't be anymore bounding boxes and category of object, but a steering angle, computed based on the salient areas of the image given as input. Somewhat similar to YOLO's approach , in [25] a visual attention model is used to train a convolutional network end-to-end. As YOLO, this model finds all potentially salient image areas that could affect the steering angle decision and pass this areas to the CNN recognition network, but it does so only using RGB images as input. There is also a post-process phase that clusters the attention network's network into so-called "blobs". These blobs are then masked to determine the effect on the end-to-end network. This step is used to show that these blobs can be exploited to improve the accuracy of the selected salient areas, while also making the process simpler.
Their method is made of two parts: preprocessing and convolutional feature extraction. During preprocessing input images are down-sampled and resized. In case the image does not fit the aspect ratio needed, it is cropped. Then, intensity values are converted to HSV colorspace. Lastly, a smoothing filter is applied on both current steering angle and velocity.
As for the encoder, which is the part that performs the feature extraction task, a convolutional neural network is used. The feature vector contains object description features that are used by the attention network to pay attention only the areas that were defined salient. [8] No comparisons with other methods were provided, but this method reportedly reduces by a factor of 60% spurious area in the input image using the "blobs".
A more interesting approach is the one presented in [26] and in the follow-up article [27]. This system, called PilotNet, learns significant features by only having the human steering wheel angle as input. Basically the system infers what is an important feature by itself. The setup is quite simple: some front-facing cameras are mounted on the car. The videos coming from this cameras are paired with the steering angle applied by a human driver, that is passed as $\frac{1}{r}$ to avoid issues when the angle is close to zero. This data is then augmented with some additional artificial images that show the car in some off-centered and rotated positions. This has shown to be a valid method to avoid having the car slowly drifting off the road and learning how to correct the position within the lane. The steering angle for these image is an approximation of what it was computed to be the steering value that could correct the position within two seconds. Moreover, training images consist mainly in images where the road curves, to avoid having biases in driving straight.
To train this neural network back propagation is used: images are fed to the convolutional neural network, which outputs a certain proposed steering angle. This is then compared to the ground truth one, and it is fed back to the CNN so that it can adjust its weights to make a closer prediction. To find salient objects the authors use the areas in the feature maps of the highest levels of the convolutional layers that have highest activations as masks for lower layers. To scale up feature maps to the size of the filters below, deconvolution is used. The up-scaled version of the upper layer is

[8] I did not really get their method for computing the steering angle though.

multiplied with the one of the lower layer to obtain an intermediate mask, that is iteratively scaled to fit the dimension of the layers below. The final mask is computed in a bottom up fashion, so from lower layers to input. This visualization mask then is used to highlight which parts of the original image are relevant to the system, i.e. the salient objects.

The last very recent article will focus more on the hardware side of Object Detection. Often in articles that tackle O.D. tasks authors mention that they used very powerful GPUs and computers. of the cited papers that report the hardware used, the only ones not using real fast and expensive NVIDIA GPUs were [20] and [6], respectively using an Intel dual core and a 2015 MacBook Pro with a Core i7 processor and an AMD Radeon R9 GPU, that was considered in their setup the lower power computing platform. It is however quite clear that these are still quite costly solutions, and relatively portable. For this reason it is interest to report the solution brought by [28], that used a Movidius Neural Network Stick with a Raspberry Pi3. The Raspberry O.S. was Debian 9.0 with OpenCv and Python 3.5 installed and the Movidius Stick's processor architecture is an hybrid between GPUs, DSPs and RISC, and it was specially designed to work best with computer vision and machine learning workloads. To test the suitability of this hardware, it was tested using YOLO and MobilNetSSD, with the former one seen in this section above. [9]
To test the goodness of this hardware setup, the authors confronted the computing time over 200 images on both Raspberry setup and on a virtual machine using Ubuntu 16.04, founding that Raspberry Pi3 took, with YOLO 9 seconds longer and with MobileNetSSD 2 second longer. This difference is reported to be probably due to the lack of the USB 3.0 interface on that model of Raspberry.
Even though no information is given about real time applications using the Movidius Stick, it could be possible to infer that, with a fast algorithm, it would not be impossible.

[9] Should add MobileNEtSSD since they found out that it outperforms YOLO quite a bit in both time and accuracy!!

Bibliography

[1] D. A. Pomerau. Alvinn: An autonomous lan vehicle in neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[2] Z. Zhao, P. Zheng, S. Xu, and X. Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, Nov 2019.

[3] K. Mahelona, J. Glickman, A. Epstein, Z. Brock, M. Siripong, and K. Moyer. Darpa grand challenge. *unknown*, July 2007.

[4] N. Spinrad. Google car takes the test. *Nature*, 514:528, Oct. 2014.

[5] Andrew J Hawkins. Waymo is first to put fully self-driving cars on us roads without a safety driver. *The Verge*, 7, 2017.

[6] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach. Real-time compression of point cloud streams. In *2012 IEEE International Conference on Robotics and Automation*, pages 778–785, May 2012.

[7] C. Guindel, J. BeltrÃ¡n, D. MartÃn, and F. GarcÃa. Automatic extrinsic calibration for lidar-stereo vehicle sensor setups. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, Oct 2017.

[8] D. Yatron and D. Tynan. Tesla driver dies in first fatal crash while using autopilot mode. *The Guardian*, 2016.

[9] A. Efrati. Uber finds deadly accident likely caused by software set to ignore objects on road. *The Information*, 2018.

[10] M. Gerla, E. Lee, G. Pau, and U. Lee. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 241–246, March 2014.

[11] S. Bitam, A. Mellouk, and S. Zeadally. Vanet-cloud: a generic cloud computing model for vehicular ad hoc networks. *IEEE Wireless Communications*, 22(1):96–102, February 2015.

[12] R. B. Rusu S. Gedikli M. Beetz W. Steinbach J. Kammerl, N. Blodow. Real-time Compression of Point Cloud Streams. *2012 IEEE International Conference on Robotics and Automation*, pages 778–785, 2012.

[13] B. Anand, V. Barsaiyan, M. Senapati, and P. Rajalakshmi. Real time lidar point cloud compression and transmission for intelligent transportation system. In *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pages 1–5, April 2019.

[14] P. A. Chow R. L. de Queiroz. Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform. *IEEE transactions on image processing*, 25(8):778–785, Aug 2012.

[15] T. Golla and R. Klein. Real-time point cloud compression. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5087–5092, Sep. 2015.

[16] *Review of LiDAR Sensor Data Acquisition and Compression for Automotive Applications.*, volume 852, 2018.

[17] E.J. Leavline and D.A. Singh. Hardware implementation of lzma data compression algorithm. In *International Journal of Applied Information Systems (IJAIS)*, volume 5, March 2013.

[18] R. Rice and J. Plaunt. Adaptive variable-length coding for efficient compression of spacecraft television data. *IEEE Transactions on Communication Technology*, 19(6):889–897, December 1971.

[19] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, June 2012.

[20] Q. Zhu, L. Chen, Q. Li, M. Li, A. NÃ¼chter, and J. Wang. 3d lidar point cloud based intersection recognition for autonomous driving. In *2012 IEEE Intelligent Vehicles Symposium*, pages 456–461, June 2012.

[21] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6526–6534, July 2017.

[22] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, June 2016.

[23] A. R. Fayjie, S. Hossain, D. Oualid, and D. Lee. Driverless car: Autonomous driving using deep reinforcement learning in urban environment. In *2018 15th International Conference on Ubiquitous Robots (UR)*, pages 896–901, June 2018.

[24] B. Li. 3d fully convolutional network for vehicle detection in point cloud. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1513–1518, Sep. 2017.

[25] Jinkyu Kim and John Canny. Interpretable learning for self-driving cars by visualizing causal attention. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[26] D. Dworakowski et al. M. Bojarski, D. Del Testa. End to End Learning for Self-Driving Cars. *IEEE transactions on image processing*, April 2016.

[27] D. Dworakowski et al. M. Bojarski, D. Del Testa. Explaining how a deep neural network trained with end to end learning steers a car. *IEEE transactions on image processing*, April 2017.

[28] A. Pester and M. Schrittesser. Object detection with raspberry pi3 and movidius neural network stick. In *2019 5th Experiment International Conference (exp.at'19)*, pages 326–330, June 2019.