BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

Measurement of Non-Electrical quantities

Multi-robot Control Systems based on Cloud Computing

Team members	Fuad Mammadzade
	Kanan Bagaliyev
	Turan Eminli
Group number	PAE 18
Project submitted on	20.12.2021 – 20 December 2021
Supervisor's name	Associate Professor, Kamala Pashayeva

BAKU HIGHER OIL SCHOOL – 2021

Contents

Abstract

Recently autonomous systems over cloud infrastructure are increasingly in demand as far as their robust and effective technology is concerned. In the pace of the algorithmic complexities, it gets harder to process all the data and store them in separate systems to operate, and transfer data with useful information. Thus, cloud computing takes the role of main computations and assist with navigation, control, and traffic management, especially in autonomous vehicles. This paper aims to depict the determination of utilizing cloud computing (or webservers) in autonomous vehicles to obtain more efficient and faster control with stronger data and predictions logged by different agents of the robotic system. The project discusses the practical application of transmitting and receiving sensor data (GPS, LiDAR, MPU-Gyroscope, Camera) and its processing in cloud provider servers. The cloud platform is a key to be used in Model Training, Simulation, High Definition (HD) Map generation, and Data storage. In this project, there are two major agents of the system: Data logger multi-robot agents (autonomous vehicles) and Cloud platform. Additionally, the paper will help to analyze the positive effects of using the cloud as a generator of the grid among multi-agent robotic systems with fully equipped sensors.

Keywords: Multi-agent sensing data, Cloud Robotics, GPS, LiDAR, MPU, real-time data transfer

Introduction

In modern multi-robotic systems research, which is a relatively new field, their implementation on cloud systems and utilizing the out-of-the-box features of cloud resources are widely investigated for their importance on task managing, data logging, and getting precise predictions for autonomous driving. The major objective of utilizing the system based on cloud computing is to alleviate the complexity of distributed computing and storage and heterogeneous computing. Since we understand that each robotic agent (an autonomous car in our project) generates enormous data from different sensing elements such as GPS for HD map generation, LiDAR for 3D mapping the environment and understanding obstacles, Camera for all the computer vision operations from sign detection to the road line determination and MPU accelerometer or Inertia Measuring Unit for detecting the behavior of the robotic system and car (Figure 1). [1]

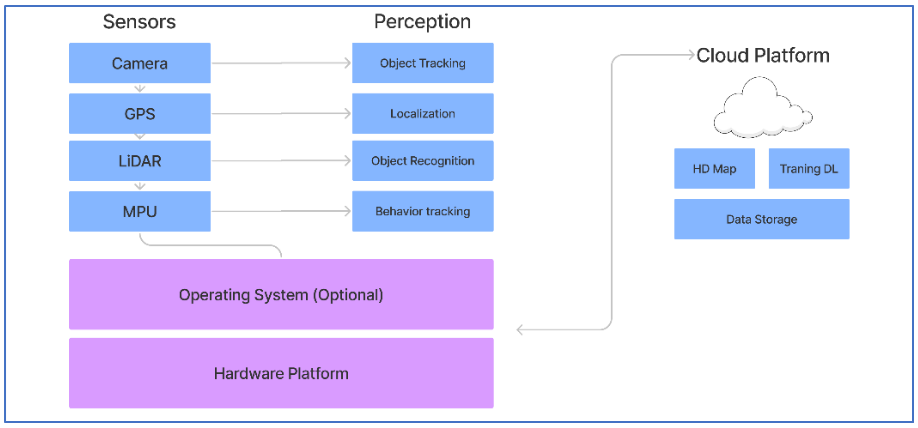


Figure 1 – Architecture of the multi-agent autonomous system

According to the project described in this paper, the different robotic agents or cars will detect different problems or advantages along the route, which is determined beforehand, with their sensors and they will transmit this collected data to the cloud platform for processing on a real-time basis. The collected data will assist other robotic systems to behave smart and more robustness could be achieved in this approach. The grid that has a dependency on remote servers, will create huge information and availability for predicting actions and defining the correct path. With the infrastructure of hardware and software introduced in this project, we will be able to utilize the real-time data on the autonomous cars with remote control and the instructions will be generated according to the data which is in the cloud platform and gained from other agents' sensor data contribution.

Theoretical background

Cloud robotics

Robot Web Tools is a set of tools meant to help Web developers and even students get started fast developing a robot Web application. Architecting a robot web application might go a number of different paths. Building web technologies on top of an existing robot framework is a popular path. One of the most widely used robot middleware is the Robot Operating System (ROS). Cloud robotics allows robots to offload duties without having to meet strict real-time deadlines by taking use of the fast growth in data transport speeds. It enables robots to take advantage of modern data centers' tremendous computing, storage, and networking resources. [5]

GPS module

A GPS receiver receives signals from a constellation of GPS satellites, which is used in the GPS-based autonomous navigation technique. In order to facilitate autonomous navigation, the receiver computes its location on the earth's surface, and a navigation algorithm evaluates additional parameters such as route and distances.

The Global Positioning System (GPS) is a constellation of satellites that provides a user with an exact position on the earth's surface. It was originally designed for military usage, but it was eventually made available to civilians. Anywhere in the globe, GPS can deliver accurate location and timing information to a user. It's a one-way system, which means that a user may only receive signals and not transmit them to the satellite. This setting is required for security reasons as well as the ability to serve an unlimited number of users. [2]

There 3 different major parts of the GPS system:

- Space
- Control
- User

A constellation of 24 satellites in fixed orbits around the globe make up the space segment. Each satellite sends GPS signals to the earth on a continual basis, which include two carrier frequencies, digital codes, and navigation messages [3]. A network of tracking stations composes the control section. These monitoring stations keep a constant eye on the satellite orbit, checking the satellite clock, atmospheric conditions, and satellite almanac, compensating for clock errors, and uploading data to satellites. All civilian and military users are included in the user segment. A GPS receiver can lock on to any visible satellite and identify its position on the surface of the planet. The GPS system was originally designed for military purposes, such as navigating through difficult terrain and coordinating military operations. It is employed in a variety of applications now that civilians have access to GPS signals. [2]

The latitude and longitude information determined by the GPS system on a device without any filtering or modification is known as raw GPS data. The total distance is obtained by adding the distances between successive GPS points. The Haversine calculation for estimating the distance between two locations on a sphere is used to compute the distance between two GPS coordinates (Chopde & Nichat, 2013).

$$d(\varphi_1 \gamma_1\ \varphi_2 \gamma_2) = 2r\ \ arcsin\left(\sqrt{sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\ cos(\varphi_2) sin^2\left(\frac{\gamma_2 - \gamma_1}{2}\right)}\right)$$

LiDAR sensor

The race to develop self-driving cars and robotic vehicles has pushed lidar sensor requirements away from those of remote sensing. Long-range, high spatial resolution, real-time performance, and tolerance to solar backdrop in the daylight are all requirements for vehicle lidar imaging systems, which has pushed the technology to its limits. For different probable usage instances, such as short and long-range, or narrow and large fields of vision, multiple standards with distinct functioning principles have arisen. Using a rotating wheel arrangement at high speed and several stacked detectors, rotating lidar imagers were the first to attain the requisite performance.

The time-of-flight (TOF) measuring method is utilized for lidar imaging, in which depth is estimated by counting time delays in events in light produced from a source. Thus, lidar is an active, non-contact range-finding approach in which an optical signal is focused onto a target, and the reflected or backscattered signal is received and processed to estimate the distance, allowing the construction of a 3D point cloud of a section of the unit's surroundings. As a result, the round-trip delay of light waves traveling to the target is used to calculate the range R, or distance to the target. This can be accomplished by modulating the broadcast signal's strength, phase, and/or frequency, and then measuring the time it takes for the modulation pattern to return to the receiver.

Since light goes back and forth to the target, the observed duration is plainly representative of twice the length to the object and must be halved to provide the real range value to the target [5].

$$(R \rightarrow\ range\ to\ the\ target; c \rightarrow speed\ of\ the\ light);$$

$$t_0F \rightarrow time\ it\ takes\ for\ the\ pulse$$

$$R = \frac{c}{2} t_0F$$

Accelerometer and Gyroscope

Of course, if a project involves some kind of vehicle, it is necessary to measure physical quantities like acceleration, speed, traveled distance and rotation angle. We would need several measurement devices in order to measure these quantities. In this part, we will look at how these devices work, what are their shortcomings, and settle on an appropriate device for our project.

First, we need an accelerometer to measure acceleration in 3 different axes. Accelerometers have many uses in industry and science. Highly sensitive accelerometers are used in inertial navigation systems for aircraft and missiles. Vibration in rotating machines is monitored by accelerometers. They are used in tablet computers and digital cameras so that images on screens are always displayed upright. In unmanned aerial vehicles, accelerometers help to stabilize flight. To know how accelerometers work, it is often useful to imagine a ball inside a 3D cube:

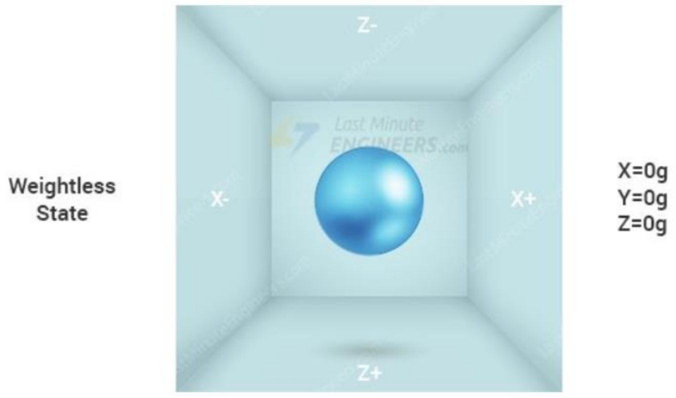


Figure 2

Suppose the cube is in outer space where everything is in weightless state, the ball will simply float in the middle of the cube. Now let’s imagine each wall represents particular axis. If we suddenly move the box to the left with acceleration 1g (A single G-force 1g is equivalent to gravitational acceleration 9.8 m/s^2), no doubt the ball will hit the wall X. If we measure the force that the ball applies to the wall X, we can get an output value of 1g on the X axis as shown in Figure 3.

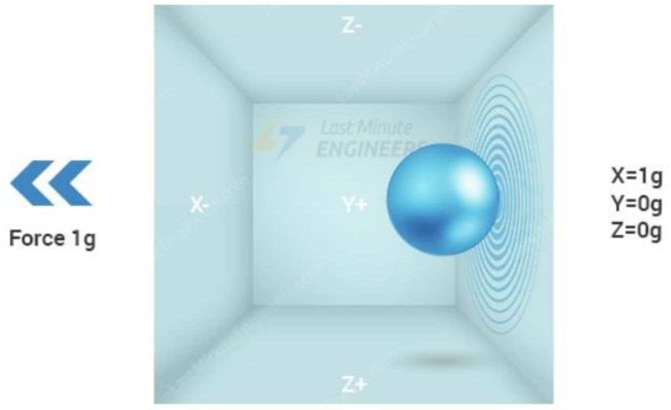


Figure 3

Identically, if we put that cube on Earth, the ball will simply fall on the wall Z and will apply a force of 1g, as shown in the picture below:

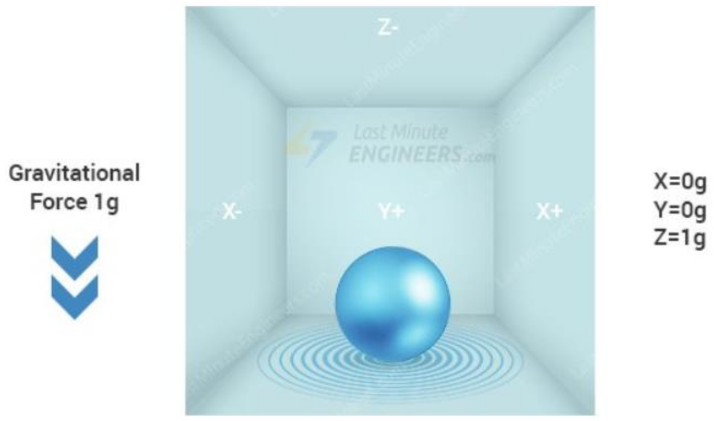


Figure 4

The gravitational force is the reason why in most accelerometers we will see an acceleration of approximately 10 m/s^2 in Z direction, even though there is no apparent motion in this direction.

The other important quantity to measure while the car is in motion is the angular speed. It can be used to determine rotation speed when the car is turning and make appropriate decisions. The central idea in the working principle of gyroscopes is Coriolis effect. Coriolis Effect tells us that when a mass (m) moves in a particular direction with a velocity (v) and an external angular rate (Ω) is applied (Red arrow); the Coriolis Effect generates a force (Yellow arrow) that causes a perpendicular displacement of the mass. The value of this displacement is directly related to the angular rate applied as depicted on the right figure.

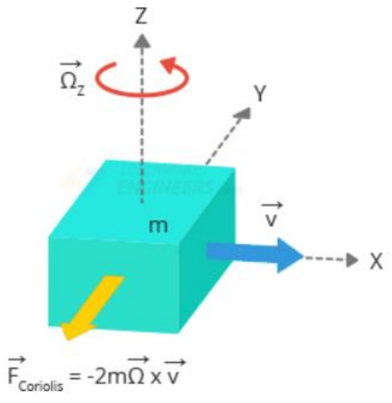


Figure 5 - Coriolis effect visualization

Now suppose that there are two masses that are kept in constant oscillating motion so that they move continuously in opposite directions. When angular rate is applied, the Coriolis effect on each mass is also in opposite directions, which results in a change in the capacitance between them; this change is sensed (Figure 6).

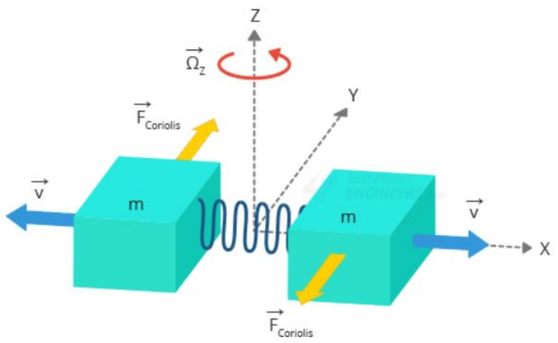


Figure 6 - Measuring angular motion using Coriolis effect

To sense and measure these 2 quantities in 3 axes (in total of 6 axes) we would need quite a bit of hardware. Instead, we used commercially graded 3-axis accelerometer and 3-axis gyroscope combined sensor module: MPU6050. At the heart of the module is a low power, inexpensive 6-axis MotionTracking chip that combines a 3-axis gyroscope, 3-axis accelerometer, and a Digital Motion Processor (DMP) all in a small 4mm x 4mm package. It can measure angular momentum or rotation along all the three axes, the static acceleration due to gravity, as well as dynamic acceleration resulting from motion, shock, or vibration.

The MPU6050 consumes less than 3.6mA during measurements and only 5μA during idle. This low power consumption allows the implementation in battery driven devices. The MPU6050 can measure acceleration using its on-chip accelerometer with four programmable full-scale ranges of ±2g, ±4g, ±8g and ±16g. The module also can measure angular rotation using its on-chip gyroscope with four programmable full-scale ranges of ±250°/s, ±500°/s, ±1000°/s and ±2000°/s.

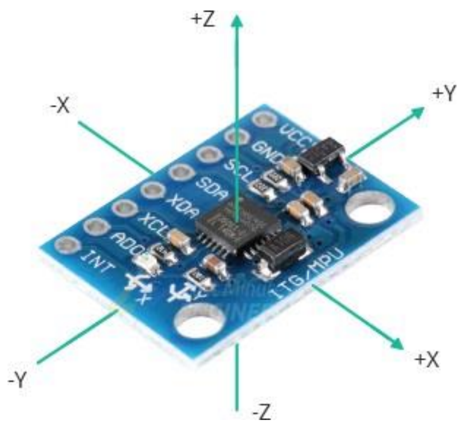


Figure 7 - MPU6050 module

Communication and connection types

As navigation sensors may create real-time data streams at high data rates (typically in the gigabits per second), cars can't send all of their sensor data within the available bandwidth. One major finding is that different types of data collected by sensors are of varying relevance. Carcel takes advantage of this by instructing the cloud to seek information at a higher resolution from only the most critical parts of the environment. These requests are sent to the car via the cloud's **request module**. It analyzes sensor data in aggregate, represented using the Octree data structure, and creates requests for different locations at specific resolutions depending on sensor data lacunae. Requests are prioritized for places that are closer to the vehicles' present positions, are on the vehicles' current trajectory, or are in the vehicles' blind spots.

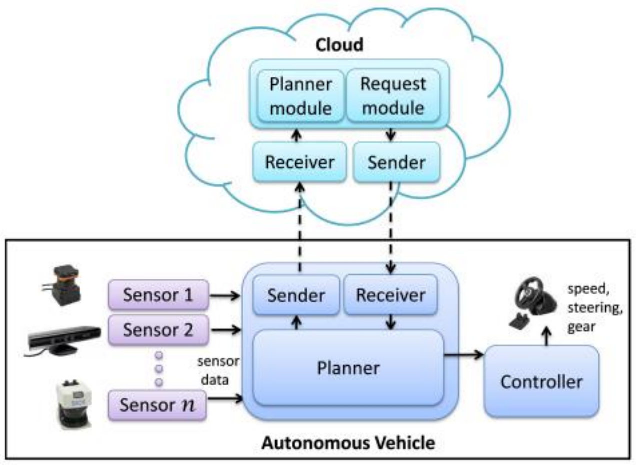


Figure 8 – Communication in the autonomous vehicle

The receiver module feeds sensor data to the **planner module**, which collects data from different autonomous vehicle and static road-side infrastructure sensors. It also keeps track of all of the cars' current locations. The planner looks at all of the sensor data it has gathered to see whether there are any impediments in the way of any vehicle's present course. The transmitter module sends information about barriers and potential alternate routes to each vehicle.

The **receiver module** on each autonomous vehicle registers requests and advises the transmitter to transmit a higher proportion of sensor data from the requested locations at the appropriate resolution. The receiver also notifies the planner of any barriers or other paths that the cloud has reported. The transmitter module sends sensor data from several desired locations proportionate to the amount of cloud requests for each of these regions. The transmitter also broadcasts the vehicle's current location and trajectory (Figure 8). [7]

Literature review

In the recent research projects, the autonomous driving cloud is a critical component of the technological stack for autonomous vehicles and because autonomous driving technologies are still emerging, we are in the early phases of developing a cloud infrastructure for autonomous vehicles as proposed by S. Liu's research paper (2017). In the mentioned paper, it is also defended that simulation testing for new method deployment, HD map creation, and offline deep learning model training are just a few of the main cloud computing applications for autonomous driving. All of these applications need infrastructure such as distributed computing and storage. One method to do this is to adapt an infrastructure to each application, which comes with a number of drawbacks: *Management overheads Lack of dynamic resource sharing, Performance degradation* [1]. The paper by S.Kumar also proposes the solution with the cloud-assistance for autonomous driving under the hypothesis that autonomous cars rely on on-board sensors to deliver sensory information about their surroundings in order to maneuver properly and modules and ultrasonic range finders are examples of sensors that offer precise ranging information, indicating the distance between vehicles and surrounding obstructions in the domain of correctly chosen software architecture and communication between the cloud and multi-agents. In their paper, they simulated their proof of concept on the Carcel prototype which helps to transfer access data from both autonomous vehicles road infrastructure which helps to increase the efficiency and compete with conventional technologies [8].

Eun-Kyu Lee states in his paper (2016) which discusses internet of vehicles with aspect of intelligent grid, that the similar progression can be seen in the vehicle fleet, from sensor web (sensors that can be accessed through the Internet to obtain data) to Internet of Things (IoT, where components with integrated sensors are networked with one another and make intelligent use of the sensors). Vehicles, according to visionaries, will perform far better than drivers, managing more traffic with fewer delays, less pollution, and, most importantly, greater driver and passenger comfort. The difficulty of dispersed control of hundreds of thousands of autos, on the other hand, should not be underestimated [9]. The proposed claim by Rob Janssen (2016) under the concept of cloud based centralized task control for human domain multi-robot operations states that most of the robots are currently assumed to operate independently and autonomously, which means that each robot receives a private task request, collects raw sensor data from the surrounding environment, processes this sensor data using on-board algorithms, deliberates on which action to perform next, and then executes that action using its actuators (Figure 9). Sensor data may now be sent in real time thanks to the advent of high-bandwidth communication technology. Furthermore, bigdata storage facilities and cloud-based computing platforms have made it feasible to store and process this massive quantity of data. He also adds that the advantages of multi-robot systems over cloud computing include cooperatively performing tasks, globally collecting the required knowledge and fixing the failures of robots instantly without wasting time [10]. Cohen B offers the configuration with the computing environments which allow robots to use their resources effectively and creating a scalable computing architecture with containerization and enabling real-time messaging over Web Socket which will assist to communicate with Robot Operating System (ROS) in multiple channels [11].

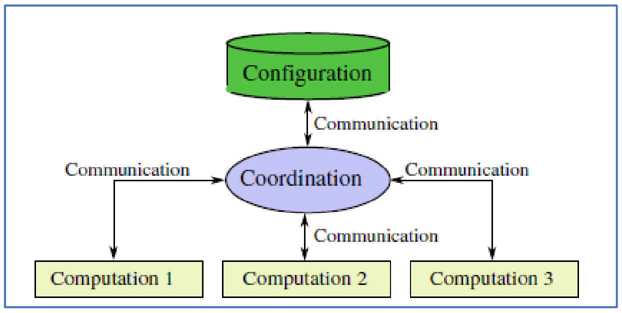


Figure 9 – Centralized communication diagram

Additionally, for practical approaches J. Zhang suggests in his research (2021), a hierarchical communication method and a resource scheduling algorithm. To evaluate connection quality for data transmission, the method considers *transmission power*, *signal-to-noise ratio*, *available bandwidth*, and other important parameters. The resource scheduling method then balances the restrictive communication circumstances and diverse computing resources, allowing the robots' most appropriate computer resources to be provided to the mobile cloud. The possible contributions that were shown in the mentioned paper about “An intern-sufficient cloud for large scale multi-robot systems” are generating lightweight messages to ensure communication channels, resource scheduling program and multitarget navigation task [12] which also helps us in our paper in the determination of HD map and other applications.

Methodology

Overview

In overall, first we create a real-time connection pool between each endpoint of the agents which generates multitude of data points to gather information. We have applied multi-robot control systems over the autonomous cars for simulating more practically and give solid output from our hypothesis. That is why, our implementation on autonomous vehicles consists of hardware sensor logging, data transmission and data receiving from cloud platform. Additionally, the methodology contains model training under the support of cloud infrastructure with GPU acceleration.

Hardware Schematics

We briefly talked about the sensors and modules utilized in the prototype in the Theoretical Background section. In this section, we will explore extensively, how each component is connected and integrated into the final hardware prototype. Let’s begin with the schematics. It is depicted in Figure 8.

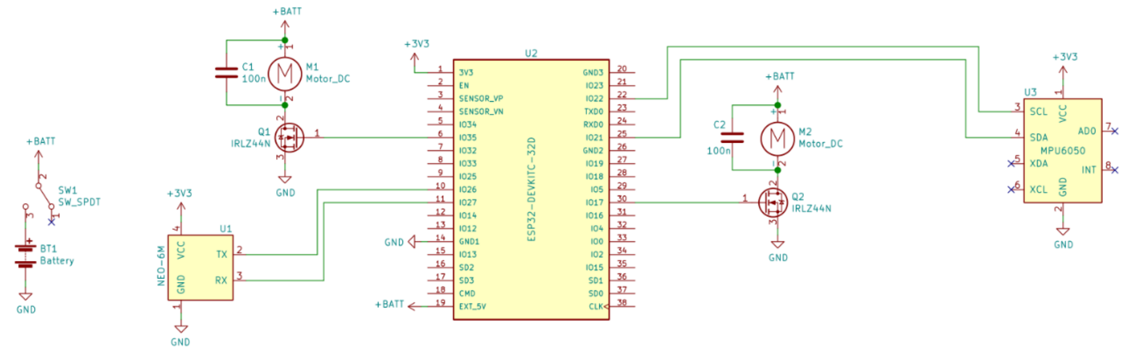


Figure 8 - Schematics of the electronics used in our prototype

Mainly, there are several different types of components:

- motors to control the vehicle,
- MOSFET to control the motor speed,
- MPU 6050 sensor for acceleration and angular speed measurement,
- NE06M GPS module for real-time positioning,
- And finally, ESP32 as the brain of the system

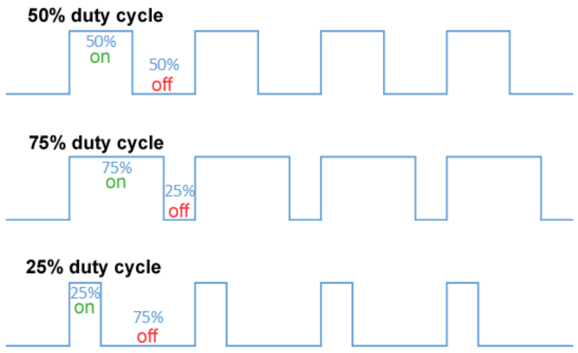


Figure 9 - PWM signals with different duty cycles

Let's elaborate on each component. First, motors are used to control the vehicle. In our model we used 2 motors for control: one as driving motor, one as controlling the direction of the vehicle. We use MOSFETs with PWM signal from the ESP32 microcontroller to control the speed of motors. PWM is abbreviation for pulse-width modulation. It is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load. An example of PWM signal is shown in Figure 9. The more the duty cycle of the PWM signal is, the more power is delivered to the load. When we increase the duty cycle of the PWM signal to the gate of the MOSFET, it also lets more current to pass through its source and drain. This results in increase in the motor speed. The reverse of this process is also working, e.g. decreased PWM duty cycle results in decreased motor speed.

ESP32 is a series of low-cost, low-power system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth. We chose this board for several reasons:

- It has wireless communication capabilities. We mainly use cloud computing techniques, which requires internet communication between the client (vehicle) and the server (cloud). ESP32 can reliably provide this communication.
- It is capable of functioning reliably in industrial environments, with an operating temperature ranging from –40°C to +125°C.
- Its cost is relatively low
- It has low-power consumption

ESP32 family has many different microcontrollers. In our prototype we used ESP32 DevKit 38 Pin.

MPU6050 sensor is connected to the ESP32 board using I2C communication interface. It is a synchronous, multi-controller/multi-target, packet switched, single-ended, serial communication bus invented in 1982 by Philips Semiconductors. It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication. A particular strength of I^2C is the capability of a microcontroller to control a network of device chips with just two general-purpose I/O pins and software. Many other bus technologies used in similar applications, such as Serial Peripheral Interface Bus (SPI), require more pins and signals to connect multiple devices.

NEO6M GPS module is connected to the ESP32 board using UART communication interface. It is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. It sends data bits one by one, from the least significant to the most significant, framed by start and stop bits so that precise timing is handled by the communication channel.

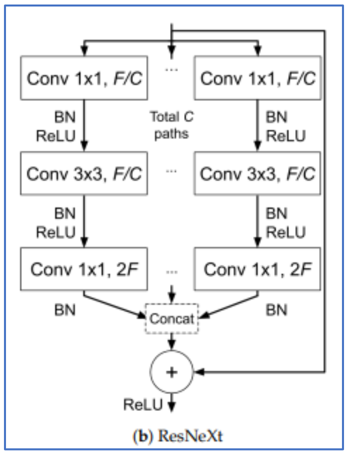
Communication between cloud and autonomous vehicles (Algorithm)

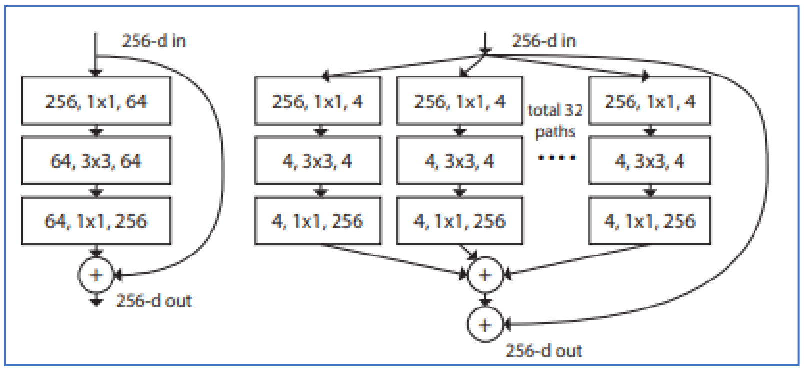
As far as optimization is concerned in the following steps, we are going to provide comprehensive information related to real-time data transfer between hardware objects which were firmly placed on autonomous cars and collect data and cloud infrastructure which contain multiple Docker containers and microservices for backend to process data faster. For this purpose, we have developed the backend service with GET and POST requests over REST (Representational state transfer) API which enables us to use the hardware data in the cloud servers.
The coming data from the hardware agent was accepted through JSON (JavaScript Object Notation) file. The JSON body will contain the logged data from the serial port of ESP32 Microcontroller which was sent through HTTP over Internet. The accepted request will be parsed in the backend side and handled by NodeJS server. After that the received data was save in the database model which runs on the NoSQL server by the vendor MongoDB. The database also stores the accepted time information. For securing our channel, authentication id was inserted in the code of the server (NodeJS) to ensure that our channel is connected to only two parties without 3rd party. This is very essential to keep the uptime of connection between our hardware and cloud infrastructure. The backend service application is running on: (Endpoint for POST request)
http://207.154.192.120/hardware/data
To move further, it is worthwhile mentioning that, it is essential to use Web Sockets in the backend side to ensure the real time data without refreshing the HTTP request. At this point, NodeJS server utilizes Socket.IO library to generate Web Socket channel between the ESP32 and our backend

service. The Socket.IO helps to create artificial room that is possible to join with backend and data transmitter side.
Another scalable microservice will be needed to store our Deep Learning model to process our car tracking, HD map generation and object recognition. This microservice will be programmed on Flask framework which is developed in Python and allows us to use Neural Network Model easily: loading in Python code and utilizing the function over the input video.

Training Data

Since we employ a variety of deep learning models in autonomous driving, it's critical to deliver updates that increase the models' efficacy and efficiency over time. However, because the quantity of raw data collected is so large, we wouldn't be able to train models quickly with only one server. In order to implement model training, we have worked on $ResNeXt$ architecture with 2 epochs whereas it is recommended to increase it until 7 or 8 to obtain more precise answers. As our computation power was limited in our testing and due to debugging time, we proceeded with 2 epochs which were also satisfactory for detecting cars and their tracking. An ensemble of multiple $ResNeXt$ architectures won second place in the ILSVRC competition with a top-5 error rate of 3.03 percent. In comparison to Inception modules, the architecture is straightforward in design. $PyTorch$ module is used to load our model and generate our model after training and fitting data.





1) $ResNet$ 2)$ResNeXt$

Model and results

Hardware

First, let's start with the hardware prototype. You can see our created vehicle prototype in Figures 9 and 10. The main problem during the making of the model was related to the accuracy of the sensor measurements. We used a technique called inertial measurement unit. An inertial measurement unit (IMU) is an electronic device that measures and reports a body's acceleration, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and

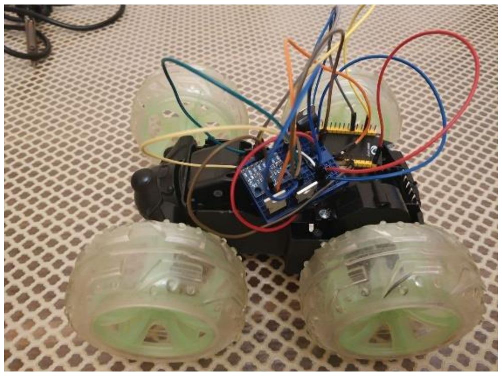


Figure 10 - Vehicle prototype without any enclosure

sometimes magnetometers. Using accelerometer data, we find the velocity of the vehicle using integration. Because we have 3-axis acceleration data, we can calculate the velocity vector in 3D space by integrating each accelerometer axis data separately. The problem with this method is even smallest drifts and inaccuracies in the sensor gets amplified by integration. That is why we eventually drift off from the real value of the velocity. This results in error-prone data, which causes wrong decisions from the cloud server AI.

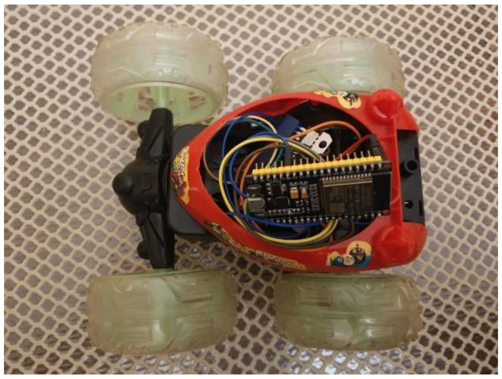


Figure 11 - Vehicle prototype with an enclosure

Cloud-based artificial intelligence

In our model training results, we observed that our resnet18 and resnext50 networks experience almost the same patterns whilst we are loading pretrained resnext50. For obtaining different result, we have defined the pretrained flag as false to get multitude of results in comparison. As we mentioned above, the epoch number is enormously affecting the runtime of our application to train images with labeled cars. Despite this, multiprocessing was applied to the system to increase the time efficiency and with help of Python multiprocessing library we were able to generate pool for running multiple applications at the same time. On the other hand, we could not increase the number epochs as we intended. In order to optimize our code, we have used RAdam, and after that we have predicted on the given input datasets which were given on the images with the labeling the cars. Here, the circle denotes the center, and squares were drawn according to the pixel of the images.

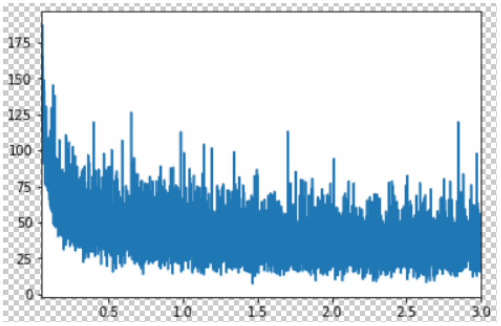


Figure 12 - Model train loss

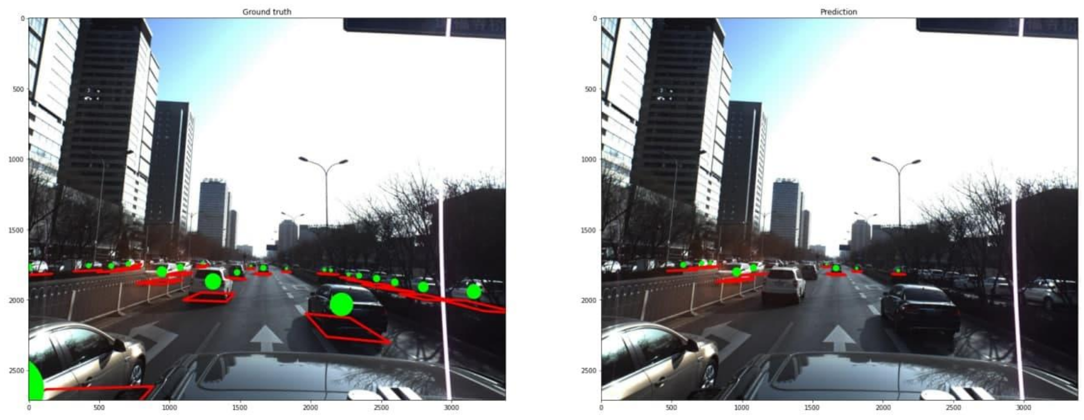


Figure 13 - Visualization of predictions

Simulation

For simulating our technical part, we have used JavaScript. Many autonomous vehicle motion planners are implemented close to the metal in C or C++, or they utilize computing platforms like CUDA or OpenCL to generate plans. Using WebGL, we can implement similar parallel planning algorithms just in the browser without installing any packages. With Three.js, the motion planner can be executed in real-time 3D simulated scenarios.

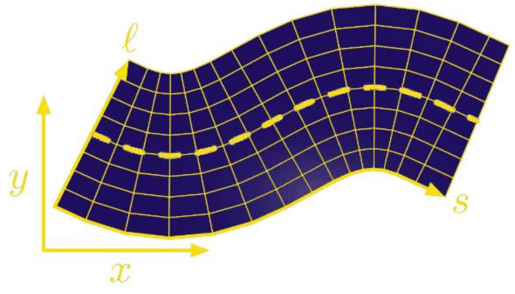


Figure 14

The simulation run in standard XY coordinate system. But sometimes we use Station-Latitude (SL) coordinate system. The station is the longitudinal distance along the road from some initial point, and the latitude is the lateral offset from the centerline.

In this project we have used just static obstacles. Obstacles are drawn to an obstacle cost grid with WebGL. To ensure safe movement around obstacles, their sizes on the grid are expanded into two zones: the smaller collision zone and the larger hazard zone. Paths traveling through the collision zone have infinite cost and are pruned during the graph search. Paths traveling through the hazard zone have an increased cost but are still feasible.

To create worked prototype as we mentioned before WebGL and Three.js were used. WebGL which is JavaScript API used for rendering interactive 3D and 2D graphics in web browsers without using any plugins. The main reason why we have used WebGL is that because it can work any platform that support OpenGL. Also, it has huge library, so we don’t need to create some basic stuffs like car, obstacles and so on from scratch. WebGL is blindingly fast and fully utilizes hardware acceleration, making it suitable for games or complex visualizations.

Another tool we have used in our project is Three.js. Three.js is a cross-browser JavaScript library which is used mostly to create animated 3D graphics in web browser. Three.js uses WebGL. With the use of WebGL, Three.js enables our browser to render smooth 3D animation and being cross-browser, it provides immense leverage of multi-dimensional objects across the web. One of the advantages of Three.js is perspective. Three.js library provides two different cameras: orthographic camera and perspective camera.

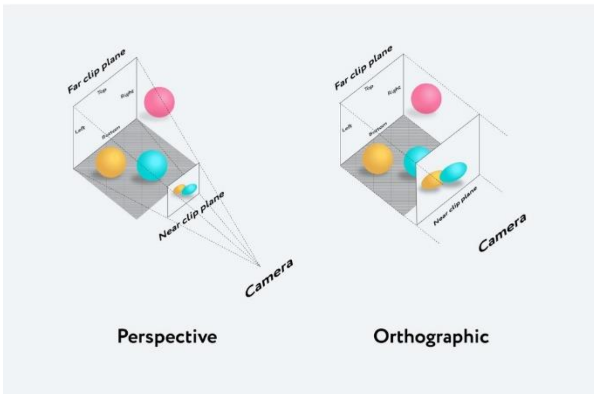
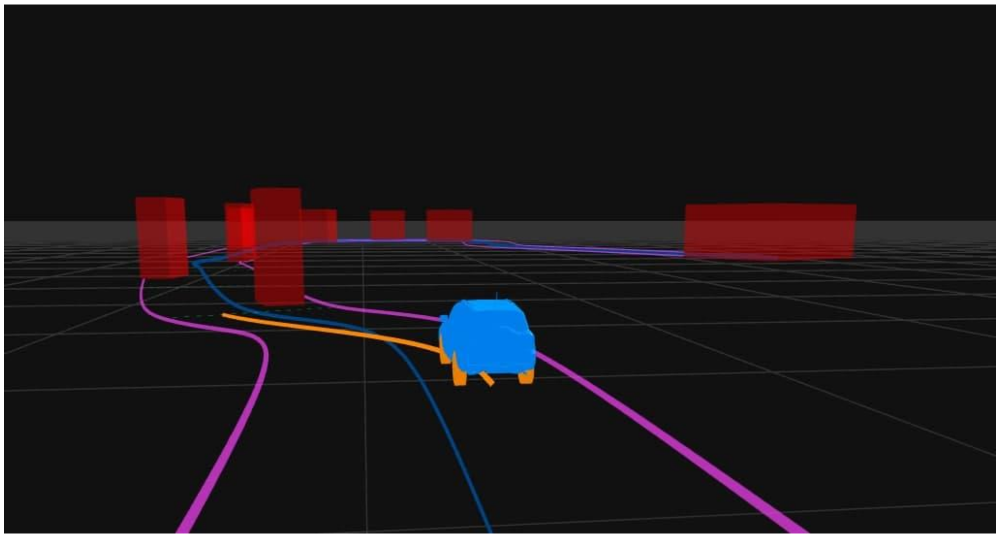


Figure 15

At the end we want to mention that this simulator is still a work-in-progress. You may see bugs, WebGL crashes, strange vehicle behavior, and any number of other failures. Currently, it is functional only in Google Chrome with hardware acceleration enabled and experimental canvas features enabled. The target frame rate is 60fps, but it may run slower depending on how long planning takes on your hardware. Several screenshots from the running simulation is provided in Figure 16:



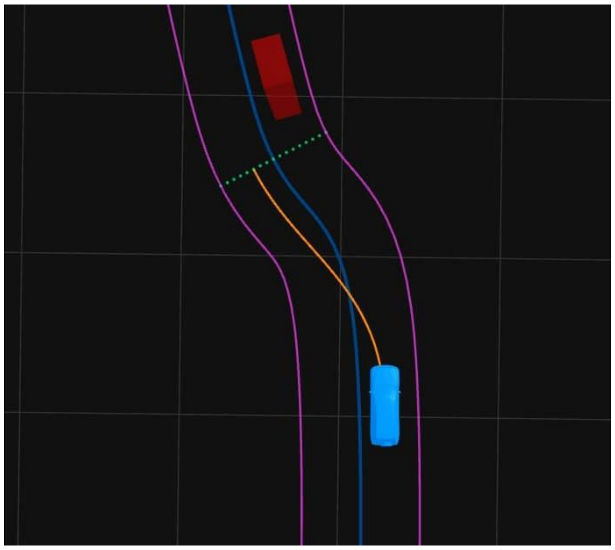


Figure 16 - Simulation images

Discussion

In this paper, we presented the architecture of utilizing cloud infrastructure for the enhancement of multi-robotic systems especially autonomous vehicles. Thus, in our results we depicted multitude of figures and images that strongly lead to the belief that, using remote servers and machines for more powerful design and architecture will be more useful in autonomous and robotic world. However, further study is required to fully exploit the cloud's potential. Autonomous cars, for example, frequently demand more precise localization than GPS can offer. Vehicles can possibly receive this localization information from the cloud by using the locations of recognized landmarks detected in the area. Similarly, the cloud may provide a variety of additional services for autonomous cars, such as real-time traffic, congestion, diversions, and accident information, among others.

The simulation of autonomous vehicles which is deployed on the cloud server was the plain example of the usage internet protocols in decision making and transmitting data. The main problem in developing such a system is assuring minimal latency and high availability across a wireless channel that is inherently unstable. We propose that a thorough cloud-assisted design of autonomous vehicular systems may considerably enhance autonomous vehicle safety and dependability, bringing commercial autonomous vehicles closer to reality.

Conclusion

All things considered; the research paper suggests that cloud computing is increasingly in high priority to rocket the efficiency of multi-robotics systems and autonomous vehicles. In this paper, we have provided different types of approaches and solutions with different methodologies by other researchers as well as depicted some algorithmic scheme and diagrams to illustrate more visually the architecture which is intended. Utilizing cloud resources in equivalent load balancing techniques we can conclude with better design perspective which is claimed mainly in our paper. Thus, many agents that followed by cloud platform and were connected to the remote server can transfer data and then benefit from the common data storage and predictions in the cloud system.

Our research could be extended to different cycles accordingly to its further development in the prototyping and generating scalable infrastructure on different cloud resource providers. This will assist to get more precise data for autonomous machines, robots and vehicles to communicate with network with more quality and faster responses. Existing Wi-Fi connection protocols are unable to provide the real-time performance assurances that these controllers require. As a result, the architecture is not totally centralized, in the sense that some control "decisions" are still made locally on the robot. We assume, however, that as Wi-Fi real-time capabilities improve, such as those proposed in, even high-rate motion controllers can be deployed on the Cloud, and that their functional implementations (such as those used for position, velocity, and force control) can be run there and successfully reused for a wide range of connected platforms and applications. Additionally, implementing encryption in Octree Representation using a 2-level Octree could also handle the previously mentioned problem in higher extent to obtain non-loss systems. To conclude, our results from this project can conclude with the statement that we could obtain significantly essential outcome to prove that multi-robot systems under the cloud network could solve the problem of scaling and low computation resources in each agent with separating the load over different machines and servers.

Additionally, our project updates are available in the following link with the document of this paper and simulation of our research:

http://207.154.192.120/

References

[1] S. Liu, J. Tang, C. Wang, Q. Wang, and Jean-Luc Gaudiot, (2017) Implementing a Cloud Platform for Autonomous Driving, (pp 1-8) Fellow, IEEE

[2] A. El-Rabbany, (2006) Introduction to GPS: the Global Positioning System 2nd Ed., Boston, MA, Artech House

[3] Y. Ye, (2011) GPS Controlled Autonomous Vehicle: An Interesting Approach to GPS Guided Autonomous Vehicle Navigation LAP LAMBERT Academic Publishing

[4] Gyula Mester, (2015) Cloud Robotics Model (pp 3-8)

[5] S. Royo (2019) An Overview of Lidar Imaging Systems for Autonomous Vehicles (pp 1-4)

[6] Alejandro Gonzalez, (2017) A Low-Cost Data Acquisition System for Automobile Dynamics Applications (pp 1-2)

[7] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun. Towards fully autonomous driving: Systems and algorithms.

[8] S.Kumar, S.Gollakota, D.Katabi (2012) A Cloud-Assisted Design for Autonomous Driving, MIT

[9] Eun-Kyu Lee, (2016), Internet of Vehicles: From intelligent grid to autonomous cars and vehicular fogs, International Journal of Distributed Sensor Networks

[10] Rob J (2016) Cloud based centralized task control for human domain multi-robot operations

[11] Cohen B (2013) Paas: new opportunities for cloud application development. IEEE Computer 46(9):97–100

[12] Jingtao Zhang, Jun Zheng, Xiaodong Zhang, Kun Zhang, Pengjie Xu and Qirong Tang (2021) An intern-sufficient cloud for large-scale multi-robot systems and its application in multitarget navigation, International Journal of Advanced Robotic Systems