DATA3900 - Bachelorproject
Final report

Gruppe 23:
Hansen, Andreas Torres (s338851)
Nguyen, Uy Quoc (s341864)
Ottersland, Anders Hagen (s341883)

Total pages: 13

Last updated:
May 16, 2022

Abstract

Short summary of the project including the result that the group reached.

Preface

This is the report of our bachelor thesis at Oslo Metropolitan University, Faculty of Technology, Art and Design. Our project was done for Accenture, and lasted from January 2022 to May 2022. We have tried to develop a solution for traffic management with self-driving cars and server communication. A physical demonstration of our proposed solution is done with the use of a Raspberry Pi computer working as a car. In this project we have documented the functionality of our system, and our process of making it.

The report is split into 7 chapters. Introduction, research areas, process documentation, implementation, results, discussion and conclusion. At the end we have added a Moscow analysis, sprint overview and our project journal as appendices. Technical terms are in appendix for those with less technical knowledge. (Kan kanskje droppe dette)

We would like to thank everyone that has contributed to our project. We would especially like to thank:

- Ivar Fauske Aasen, Solfrid Johansen and Benjamin Vallestad, representatives from Accenture.
- Dr. Jianhua Zhang, internal supervisor at OsloMet.

Contents

Implementation

1.1 Pre-project phase

After we had gotten in touch with Accenture and spoken with the supervisors and the product owner, the group had to make a few decisions regarding the direction of the project.

The first choice to make was to either build an AI for the vehicles from scratch or use the AI from the group prior. Building the AI from scratch ment that we could make an AI which integrated our system from the start, however with the time constraint of the project we decided that it would take less time to use the product from the group prior. In addition, the group had more prior experience with networking than with raspberry Pi and AI. We therefore chose to use the AI from the project prior.

During the pre-project phase there was a need for some project planning. During this planning phase we made a brief project plan. The plan contained a backlog with tasks and the estimated time period we would have to work on that current task, and with that a gantt-diagram containing those tasks.

We also used the pre-project phase to get to know Accenture, their guidelines and their workspace.

1.1.1 Choice of programming languages

For the programming languages we used python for the client and C# for the server. The group prior had used python for their raspberry Pi car. This meant that using python made it easier to extend the code from that project. The group also had prior experience with networking in python, therefore making python a prime choice of programming language for the client. C# supports multiple ASP.NET Core libraries which was useful since we were going to work with networking. We also had some prior knowledge coding in C# as well.

1.2 Implementation of Server and Client

As IoT-systems become more complex, there is a need to structure them in layers. The three fundamental layers to an IoT-system consists of the perception layer, network layer and application layer (**iot_gateway**). In this part we will discuss what our things-layer and network-layer consists of, and explain the theory behind our decisions.

1.2.1 Implementation of Client

Raspberry Pi is a small single-board desktop computer that is commonly used for IoT-projects. There is an enormous ecosystem of compatible devices that allow these computers to interact with the world in various ways. The organization states that they can be used for everything "from music machines and parent detectors to weather stations and tweeting bird houses with infra-red cameras" (**raspberrypi**). For our project we have used a Raspberry Pi Model 4, which the previous group used in their project. This is the newest and fastest model, which makes our test results as accurate as possible.

The perception layer is responsible for perceiving the world, and creating data for the network layer to collect and deliver (**iot_platforms**). Devices that contribute to this are, for example, Global Positioning Systems(GPS), cameras and sensors. The raspberry-pi models we worked with utilizes both a camera and a range detection sensor that gets processed by the image recognition algorithm running on the machine. These vehicles play the role of client in our system, as they connect to the server.

The requirements for our client was that it needed to be able to take in commands from the server and respond correctly to those commands. In addition, they had to be able to act by themselves if they were not connected to the server. For that to happen the vehicles had to send their size, position and velocity to the server when they connected initially. The server also needs to send their velocity and position to the server at a frequent rate so that the server can keep track of that information.

1.2.2 Implementation of Server

The network-layer of an IoT-system is responsible for transferring data collected by the things-layer (**iot_platforms**). In our project this is done by a centralized server that is connected to all the clients.

The task required the client, which in our case is the cars, to receive the commands from the server. The group prior had made it such as the car made decisions based on its AI. The AI used picture recognition which could recognize speed limit signs and stop signs. The cars were also programmed to follow a path. Furthermore we needed a server which could send commands to the clients at a frequent rate, based on the information given by the clients. The server's commands need to overwrite the AI that is already on the vehicles.

Here are some requirements needed for the server:

- Needs to be able to send messages to all clients simultaneously, or a specific group of clients.
- Handle increasing traffic
- Little to no delay
- Two way communication between server and client

We initially coded a server by using API. The server was able to receive information from the clients, but for the cars to receive information from the server, they had to host their own servers. The car's main priority should be calculating decisions based on their AI, not hosting a server. It would also be harder to add a server to the code from the previous group. We therefore thought of some other solutions.

Websockets provides a two way communication over a single tcp-connection. This means that both the server and client can send and receive messages from each other. They also allow for a better efficiency than REST API because they do not require the request/response overhead for each message that is sent and received. This solved the issue we previously had with the solo API-solution.

ASP.NET Core Signal R is an open source library that simplifies adding real-time web functionality to apps (Microsoft, 2022). It supports websockets as a real time communication which is perfect for a two way communication. Signal R is often used in games, social networks and gps-apps where information to the clients are needed instantly. It also scales with increasing traffic. Signal R uses hubs where servers and clients communicate with each other. Hubs allows servers and clients to call methods on eachother. Since the server needs to send the cars commands this is perfect for our Iot-system.

The server needed some specific functionalities in correlation with the problem statement stated earlier. We did some research on what caused traffic jams and chose two problems which our server would focus on solving:

- Delayed reaction times when people stop and accelerate
- Increased traffic flow in intersections

The first point happens when a car changes velocity. Therefore the server has to check if any car's have changed their velocity. If yes, the server needs to send a command to all the car's behind a certain distance of the car that changed their velocity. The distance will be calculated by the server based on the change of velocity. The command sent out to the car's behind will be to change velocity based on distance to the car that changed their velocity and the amount of changed velocity.

To increase traffic flow in intersections we can have the server work as some traffic lights. The server can check if cars are closing in on the intersection's position and give one lane the green light and the other the red light. The lights will depend on which car's come first. The difference between a server doing this and a traffic light system is that the server can tell the vehicles to slow down before the intersection, making the traffic flow smoother.

1.3 Implementation of simulation

To show that the solution was reaching the requirements we needed to create a demonstration. A demonstration was also an important part of testing the functionalities of the IoT-system. There were two options which were viable in this case. The first one was to create a virtual simulation using unity or another graphic-program. The other option was to build a physical demonstration with two or more cars. We chose to implement a physical demonstration with two cars. This was because we already had one car finished by the previous group. In addition it was more beneficial for Accenture with a physical demo since they can show it at exhibits.

- That the server can turn off and the cars would still drive on their own.
- Cars communicating through the server and acing based on the server's decisions
- Scalability to the real world

First we built the server with only one single lane road in mind. With this solution the potential of showing off the functionalities of the IoT-system was low. We could potentially have shown that if a car in front slows down, the car behind also slows down. This would show that the system could prevent some traffic, but only in a specific scenario. However we wanted to show that the IoT system could work in more than just one single-lane road. We therefore chose to try to emulate an intersection in the physical demo.

For a physical demo to work we needed to build another car. Luckily the previous group had documented their work and we could follow their process from their final report. We also had the parts provided to us. However we were unable to get the full functionality of the second car since the TPU accelerator used by the previous group was unavailable. The TPU accelerator provided the car with the processing power needed for the AI. This meant that the car was built without the camera. But this had a minor effect on our demo since the other car was fully functional without the system. We solved this problem by letting the server do the decision making for the car with less functionality.

We also needed to emulate a road system which contained the intersection we were going to demonstrate in the demo. Since the AI had not trained to recognize intersections yet we had to code a road system into our server that contained roads, lanes and intersections. We then had to build a physical intersection for our cars with tape that correlates with the emulated intersection.

1.4 System security

Security and privacy are important topics for any IoT-system. Because these systems gather and work with huge amounts of data, they are naturally prone to being attacked. And as the systems grow and become more interconnected, with many devices around the world, the imposed risk of such an attack increases drastically [kilde på dette]. Our project is on a small enough scale that we don't expect security to be an extremely important feature for it to work. However,

because we want the system to be scalable in the future we have implemented some ideas that we think improves the data security in our system.

1.4.1 Privacy

Throughout our work we discussed how the solution would look in the real world. If all drivers on a road were to connect their cars to the same server it would be easy for anyone with access to that server to track any one driver. Therefore we proposed a solution based on the concept of anonymity (kilde på dette her, men vi må være på skolen for å ha tilgang). This is done by giving every vehicle that connects to the server an anonymous randomly generated identification-number that only lasts as long as the connection to the server is upheld. When the connection closes this number is deleted, and the same vehicle will get a new number the next time it connects to the server. This way we reduce the ability for an attacker to deduct any meaningful information tied to a single driver.

1.5 Phase 4 - Making a demo

At this point in the development we were sure about what kind of situation we wanted to simulate to make a satisfying product for Accenture, and to answer the problem statement we had decided on. Following the work requirements we now needed to make a demonstration that showed how the system worked. At this point we also decided that we wanted to make a physical demonstration instead of making a digital simulation, although this was a solution that would take less effort and still be a valid solution. At the start of this work phase we had started to consider making only a digital simulation, but after a meeting with our external supervisors at Accenture, in which we were advised that a physical demonstration was more in line with Accenture's goals for the project, we finally decided to make a physical demonstration.

1.5.1 Building a car

The previous group had only built one car for their project. To show a situation where two cars meet at an intersection we needed to build a new car. Luckily, Accenture kept a box of unused components from the previous group. However, we only had one Tpu, the Coral Usb Accelerator. This was an important component for giving extra processing power to the computer, and it was necessary to run the artificial intelligence the previous group had used (source from previous project).

Without this accelerator we could not run the artificial intelligence that the previous group had made. Due to the global chip shortage caused by the Covid pandemic, the accelerator was not available to purchase anywhere. This also made the camera and distance measuring sensor redundant, as these used artificial intelligence to process data. Not having two cars that utilized artificial intelligence could be a challenge, because one of the required features of our solution was that the cars should be able to override the server. We decided that as long as we had one car that could override the server commands the other car could drive solely on commands from the server. With the components, and the

product documentation of the previous project (source from previous project), we were able to build a copy of the car.

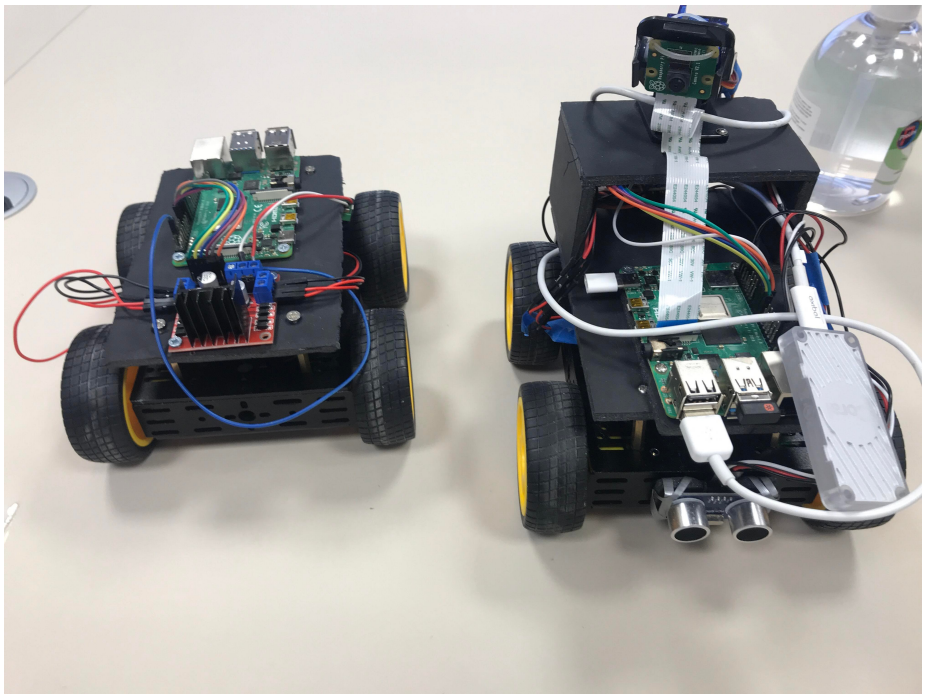

Figure 1.1: Cars with and without camera, right to left

1.5.2 Calibration of the cars

When the server, vehicles and client were implemented, and the second vehicle built, we did some testing to figure out how the car's behaved when given directions by the server. In this test the vehicles were given a specific velocity and driving distance by the server. When the vehicles arrived at their destination the server would tell them to stop. The car's drove in a straight line.

The vehicles were able to send information, and respond correctly to the servers commands. We also observed that the vehicles drove a different length for each velocity given even though the length was the same. This is because the velocity given to the vehicles is the amount of power going into the car's motors, not the actual velocity of the car's. We wanted the demo to be accurate so our group did some further testing where we wrote down the results.

The data Power was the velocity given by the server. Velocity was the actual velocity in our testing, which is length divided by time. As you can see the velocity was not the same as the power. We then made a graph to visualize the two values. The y-axis was the velocity while the x-axis is the power.

Figure 1.2: Graph of velocity as a function of power

We observed that the correlation between power and velocity seemed linear.

Power (?)	Length (cm)	Time (s)	Velocity (cm/s)
40	467	8.98	52.00
50	425	7.28	58.38
60	400	6.06	66.01
70	357	5.18	68.92
80	325	4.49	72.32
90	314	4.03	77.92
100	286	3.62	79.01

Table 1.1: Test text

This means we could make a specific formula that describes the correlation between the two values. We used linear regression to figure out this formula:

Figure 1.3: Graph of velocity as a function of power with linear regression

The formula we ended up with was as follows: $P = 0.4516v + 36.189$, where P is power and v is velocity, with a mean square error of $R^2 = 0.9653$. When we coded the formula into the vehicles we did another set of testing. We observed that the vehicles drove more or less the same distance for each power given. If we wanted an even more accurate formula we could have tuned the formula with the test results from our new test. Although the results were not hundred percent accurate, we concluded it was accurate enough for our demonstration.

Product documentation

2.1 Client and server

2.2 Demo

To test the solution we have worked on, we made a physical demonstration with two cars that meet at an intersection. We want to test our hypothesis that a combination of a centralized communication system and artificial intelligence can improve traffic flow. To test this, our demonstration shows

2.3 User manual

We have written a manual for people who want to recreate our demonstration. The demonstration could for example be shown off at exhibitions. The manual could also be used for people who want to further test and develop our IoT-system.

First check the IP-address of the internet you are connected to, and the usable ports. Make sure that your computer hosting the server and the vehicles are connected to the same network.

```
$ipconfig getifaddr en0
192.168.56.208
```

Then open the Server solution in your code editor, we have used visual studio. Under the folder "properties" there is a file called launchSettings.json. In that file write in the ip address and the port in the applicationUrl-section:

```
"profiles": {
       "SignalRServer": {
              "commandName": "Project",
              "dotnetRunMessages": true,
              "launchBrowser": false,
              "applicationUrl": "https://192.168.56.208:7058;http
                  ↪ ://192.168.56.208:5048",
              "environmentVariables": {
                     "ASPNETCORE_ENVIRONMENT": "Development"
              }
       },
```

After that open the Client solution. Here we used Pycharm as the text editor. Open the config.json document and write in the same ip-address and port:

```
"client": {
       "host": "192.168.56.208",
       "port": 5048,
       "delay": 0.1
},
```

If the vehicles haven't connected to that network before, they need to log on that network. To log on to a new network you will need to connect the raspBerry pi's to a screen. That could be done via the micro usb-port at the raspberry pi. When the raspberry pi has booted up, click on the internet-icon and connect to the same network as the server. If you have connected to the internet it will be saved and the vehicle should connect to that internet automatically when booting up.

If you want to change the velocities of the vehicles it can be done here in the server:

```
public class VehiclesHubDatabase : IVehiclesHubDatabase
{
       private readonly Intersection _intersection;

       private readonly HashSet<Vehicle> _vehicles = new();
       private readonly HashSet<Lane> _lanes = new();
       public int Count => _vehicles.Count;

       private readonly Dictionary<string, Vehicle>
           ↪ _connectionIds = new();
       private readonly Dictionary<Vehicle, string>
           ↪ _vehiclesConnectionId = new();
       public Dictionary<Vehicle, Thread?> VehicleThreads { get;
           ↪ }= new();
       public Thread? GlobalThread { get; set; }
       public Stopwatch Clock { get; } = new();

       public double SpeedLimit => 80;
```

The file is located at VehicleHubDatabase under the Database folder. The variable you want to change is the SpeedLimit.

Right under you can change the length of the roads:

```
public VehiclesHubDatabase()
{
        _intersection = new Intersection().
                AddRoad(new Road {Length = 300}.
                        AddLane(null, true).
                        AddLane()).
                AddRoad(new Road {Length = 300}.
                        AddLane(null, true).
                        AddLane());

        _intersection.ConnectedLanes().
                ForEach(lane => _lanes.Add(lane));
}
```

We have written more about the specifics of the system in our product documentation. Then place the vehicles down at the start of the track, turn on the server and connect to the power banks. The vehicles should automatically connect to the server and start driving after 20-30 seconds.

Discussion

3.1 Evaluation of process

In our process documentation we have described the way the project progressed from beginning to end. In this section we will discuss positive and negative takeaways, with the goal of aiding further work on the project. This will include our approach to work methodology, communication and team building.

3.1.1 Work methodology

As previously mentioned, we chose to apply an agile work methodology based on the fact that our project seemed to be prone to many changes. We included rituals like daily stand-ups, sprint planning meetings and retrospective meetings after sprints. Doing this helped our team understand what difficulties the other members were facing, and gave us time to discuss our obstacles daily.

There were also some elements of agile development that we did not incorporate, either because of lack of experience or lack of time. As explained in the work of Skyttermoen, T. and Vaagasar, A.L.; agile development teams regularly have meetings with the product owner, who give feedback on the strengths and weaknesses of the deliveries (Skyttermoen & Vaagaasar, 2017, pp.120). Our group only had a few meetings with the product owner during the project period, to update him on how far along we were. We did, however, have biweekly meetings with our external supervisors at Accenture who provided us feedback on the work we had done. Because we had the freedom to choose the type of solution we wanted, as long as it met the requirements that were set to us, we didn't need the product owner to be in our biweekly meetings.

3.2 Evaluation of results

3.3 Further work

How to implement self-driving in society in the best way is a question that will take a long time to answer. Through our work with this proof-of-concept we believe we have made an addition to this discussion. Due to time constraints, we have chosen not to implement some features that would make the product

work in a more complex environment. These features could be explored if this project were to be further developed.

3.3.1 Scalability

When the knowledge and research has come further regarding AI, there may be a possibility for such an IoT system to be scaled for the real world. There is a need for less traffic in the cities and our results from this project shows that an IoT-system where cars can communicate through a server will increase traffic flow. Scalability is therefore important which we had in mind while coding the server and the client.

Our demo only contains two cars but our server is built in a way where multiple vehicles can connect to it. There are also possibilities to connect other devices to the server, for example traffic lights. However there are no specific functionalities regarding traffic lights on the server as of now. Scaling the IoT system for functionalities with traffic lights is a task for future development. Adding roads and making a more complex road system would also be a task for future development.

3.3.2 Extendability to the real world applications

Edge computing

In the future when self-driving vehicles become more prominent, and the 5G network becomes more available there could be a possibility for IoT-systems handling traffic management. Our group therefore did some research regarding how the system will extend to the real world's applications.

The IoT systems usually follow the fog or edge computing architecture with distributed or even decentralized concepts (**iot_platforms**). This is to prevent overloading on servers handling a lot of data.

One solution to distribution of data is that each road has one server responsible for their respective road. If a road is long it will be split into geographical areas where one server has responsibility for their geographical area. Intersections will have a server handling information from both road's respective servers, since information from both servers are needed to make decisions in intersections.

In the traffic there are a lot of unforeseen situations that can happen. Traditional coding will not be able to cover every outcome in a traffic situation, therefore there will be a need for AI on the servers in addition to the cars.

3.3.3 Risk management

Conclusion