

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

Abstract

A self-driving car is a solution that may solve the deadly serious traffic problems. Self-driving cars are supposed to decrease the death rate dramatically. This project aims to achieve the development of a self-driving ambulance. The project will be based on SoC design. Specifically, there will be a neural network that will be trained on a simulation environment in order to avoid collisions. This training will be based on Deep Q-Learning, which is a method of Machine Learning. In addition, there will be a route tracking feature. This trained model will be implemented on ZedBoard. The importance of this solution is improving the efficiency by using FPGA. Such a solution is proposed to have great impacts on daily life, as the ambulance will arrive at the emergency place as soon as possible, without any risk. Furthermore, when this solution is integrated into all vehicles, the time spent during traffic congestion, as well as the number of deaths caused by traffic accidents will be minimized. In terms of industrial applications, this solution can be a good reference for other self-learning and route tracking applications.

1. Introduction

The world population growth highly increases the traffic, as well as the rate of death caused by car accidents. Although there were developed partial solutions based on the idea of autonomous vehicles[1], there are still huge gaps that must be filled while trying to reach a comprehensive solution. That is why autonomous vehicle development is an active research area. This project aims to achieve the development of autonomous ambulances by using machine learning. Machine learning is a subbranch of artificial intelligence that aims to make decisions and learn from data continuously. There are various areas of machine learning such as supervised, unsupervised and reinforcement learning. This project uses reinforcement learning, which is based on finding the optimal policy under the guidance of reward. However, it requires compute-intensive functions and special hardware design is needed for the acceleration of the reinforcement learning. A neural network is trained in order to achieve the learning task by predicting the best action in each state. The prediction of the best action is based on a function of some parameters of current state that calculates q values of each action, which in turns is a function of the reward-penalty mechanism. The big deal is to find the appropriate reward-penalty mechanism for the specific problem, while there is a trade-off between the number of layers and neurons. The car not only will follow a route without colliding but will also find the shortest path leading to a specified target. The output of this project will be a real miniature prototype of the model trained in a simulation environment, which can be realized in a real ambulance in a future ideal world where only self-driving vehicles are on the roads. The main disadvantage of using neural networks in embedded systems is its high computational complexity in terms of time and energy consumption. In this project, ZedBoard Zynq™-7000 will be used as a programmable SoC platform to implement the pre-trained neural network as hardware accelerator in the programmable logic part of Zynq chip and the accelerators will be interfaced with a ARM processor being used for main control problems.

SelfBulance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBulance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

2. Method

2.1 Implementation

2.1.1 Machine Learning

Machine learning[2], which is an application of artificial intelligence, is the ability of the program to learn and improve its behavior by using its own experience, without the need for an external command. The learning process is to find patterns of data or observations, such as examples, experiences or instructions, and to improve predictions for the future. The primary objective in this field of application is to equip computers with self-learning capabilities without human intervention and assistance, and to make their actions regulate in this direction. Machine learning makes big data analysis possible and needs this data to learn. Nowadays, machines that can make faster and more accurate decisions on many issues, however, need a lot of resources such as time and large data stacks. Although there are many machine learning algorithms gathered under the main supervised and unsupervised algorithms, the search for faster, more accurate, more flexible machines that require faster learning, less data and more advanced algorithms is active in this field.

2.1.2 Reinforcement Learning

Reinforcement Learning (RL)[3] is one of the areas of machine learning. In computer science it is a generic term given to a family of techniques in which the learning system seeks to learn through direct interaction with the environment. The concept of reinforcement learning is inspired by the corresponding rewarding and punitive learning analogies encountered as living models of learning. As an example, to learn a route, a reward is assigned when the agent comes to the end of the route and a penalty is introduced every time the agent is out of the route. The purpose of the learning system is to maximize a function of the reward. The system is not guided by an external supervisor on what action to follow but it should only discover which actions are the ones to make the most profit.

Reinforcement learning was used on Deep Blue, the computer of DeepMind which had won Kasparov at chess. Also, it was used on AlphaGo and AlphaGo Zero to win the world champions of Go. Nowadays, the next step is to use reinforcement learning to be able to face an opponent on StarCraft.

2.1.3 Deep Q-Learning

Deep Q-Learning is based on a neural network[4], which approximates the q value function. The inputs are just the states of the agent and the output is q values for all possible actions for that state. For our prototype, states are just values of distance sensors.

Deep learning method uses artificial neurons. In this method, neurons are put together as layers with different numbers of neurons. Then these neurons are linked to adjacent layers' neurons with some random weights. The purpose is to make output values the same with target values. The only way of doing this is changing those weights. The learning phase is changing those weights in direction that is getting an error of output small. When the error is very small and converges to some value, then the learning is

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

supposed to be done. Next step is testing this model. When the model does a good job and the results of tests are convincing, then the learning is successful.

2.1.4 Simulation Environment

The learning method to be used in the small-scale prototype is q-learning, where certain parameters define the current state of the system, and when these parameter values are discrete, all possible states can be kept in a table. This table holds the quality values of the possible actions for each quantized state. In a small-scale prototype, the vehicle's state is the distance values detected by the distance sensors on the vehicle. A simulation environment was needed to obtain the values of these sensors and to apply the actions correctly.

Godot[5] is a game engine that has many functions such as physics calculations, collision controls, screen plotting, so it is very convenient to use as a simulation environment. However, machine learning applications cannot be done in Godot. This situation led us to find a different solution. As a result, machine learning applications are managed externally in Python and these two programs are communicated through a communication protocol. In this case, UDP[6] (User Datagram Protocol) is suitable instead of TCP (Transmission Control Protocol) [7] considering the speed requirement at the simulation and communication points. While UDP communicates with multiple agents simultaneously on a single q-table, problems such as synchronization and different data types of the two languages (Python and GDscript) have been solved by team members. Mathematical calculations of the simulation are performed on the Godot side and the necessary data is coded and that coded data is transmitted to the Python side. Here, after decoding the signal and learning operations and selecting the action, the commands such as which action to take are transmitted back to Godot and the new action is applied

2.1.5 System-on-Chip

A system on a chip (SoC)[8] is an integrated circuit that integrates an entire electronic system onto a single platform. An SoC generally includes a processing system, programmable logic, input and output ports, internal memory, as well as analog input and output blocks. Although all components are integrated into a small area, the system still can perform a variety of functions including artificial intelligence, digital signal processing and more.

One of the main motivators behind the creation of systems on a chip is the fact that moving ahead into the future, our primary goal is to reduce energy waste, save up on spending costs, as well as reduce the space occupied by large systems. With an SoC, you achieve all of those goals as you essentially size down what is normally multichip designs onto a single processor that uses much less power than before. These chips are frequently used in systems pertaining to the Internet of Things, embedded systems, as well as our own smartphones and cars.

2.2 Technical Complexity

2.2.1 Hardware

- DC Motor

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

The DC motors on the rear wheels [9] enable the vehicle to move. When the command signals are sent from the Arduino to the motor drive, the motor is moved. It is planned to use motors with a working voltage of 12V and a forced current of approximately 3A. Considering the voltage and amperage values, it is thought that the vehicle will meet the power demand.

- Distance Sensor

Ultrasonic HC-SR04 [10] distance sensor is thought to be used in this project. This sensor will be mounted on the vehicle according to the angle of view so that it can detect obstacles in front of the vehicle. Accordingly, the vehicle is expected to determine the direction. The sensor provides a non-contact measuring function of 2cm -400cm, the range accuracy can be up to 3mm and is capable of measuring at an angle of 15 degrees. If this angle is insufficient, it is considered to use more than one sensor. The sensor has ultrasonic transmitters, receiver and control circuitry.

- Motor Driver

The vehicle system is intended to use two DC motors on the rear wheels and a servo motor that controls the front wheels. For this purpose, it is planned to use a motor drive that can control both servo and DC motors in order to use a single motor drive. The motor drive intended for use is capable of operating motors in the voltage range of 4.5V-13.5V. TB6612 [11] MOSFET driver 1.2A and 3A per channel max. current capacity.

- ZedBoard

ZedBoard is a development kit that is widely used for SoC design. There are two main parts: Processing system and programmable logic. Processing system mainly includes an ARM processor, which is a hard processor, and Application Processing Unit. Programmable logic is mainly composed of FPGA fabric logic, which can be designed in terms of the application needs. Furthermore, another processor, named soft processor, can be implemented in programmable logic if needed. Finally, another important part of ZedBoard is the AXI interface, which is useful for connection of processing system and programmable logic both with each other and with peripheral devices.

- Logic Level Converter

Logic level converter is a small device that safely steps down 5V signals to 3.3V AND steps up 3.3V to 5V at the same time. It is used for the conversion of voltage level between ZedBoard and peripheral devices.

- Lithium-ion Battery

Lithium-ion battery is used to power the ZedBoard and the motor driver. They were preferred as they are lighter than any other power supply.

- Barrel Jack Connector

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

Connector is used to supply voltage to ZedBoard.

- Step-Down Switching Regulator

The Step-Down Switching Regulator allows the user to keep the voltage value to a specific value.

3. Empirical Setup

3.1. Simulation Setup

3.1.1. Training Setup

Simulation environment was created on Godot. The simulation model has three raycasts, which act like distance sensors. The values of these raycasts are used by Python in order to determine the action that needs to be taken. There are three possible actions: right, left and straight. If any of the raycasts has value 0, this means that the car collides, and a “done” signal is sent to Python in order not to choose that action in that state again. Python code is responsible for the creation and training of neural networks and for the choice of the proper action with regard to the ray casts of the car during that simulation moment.

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

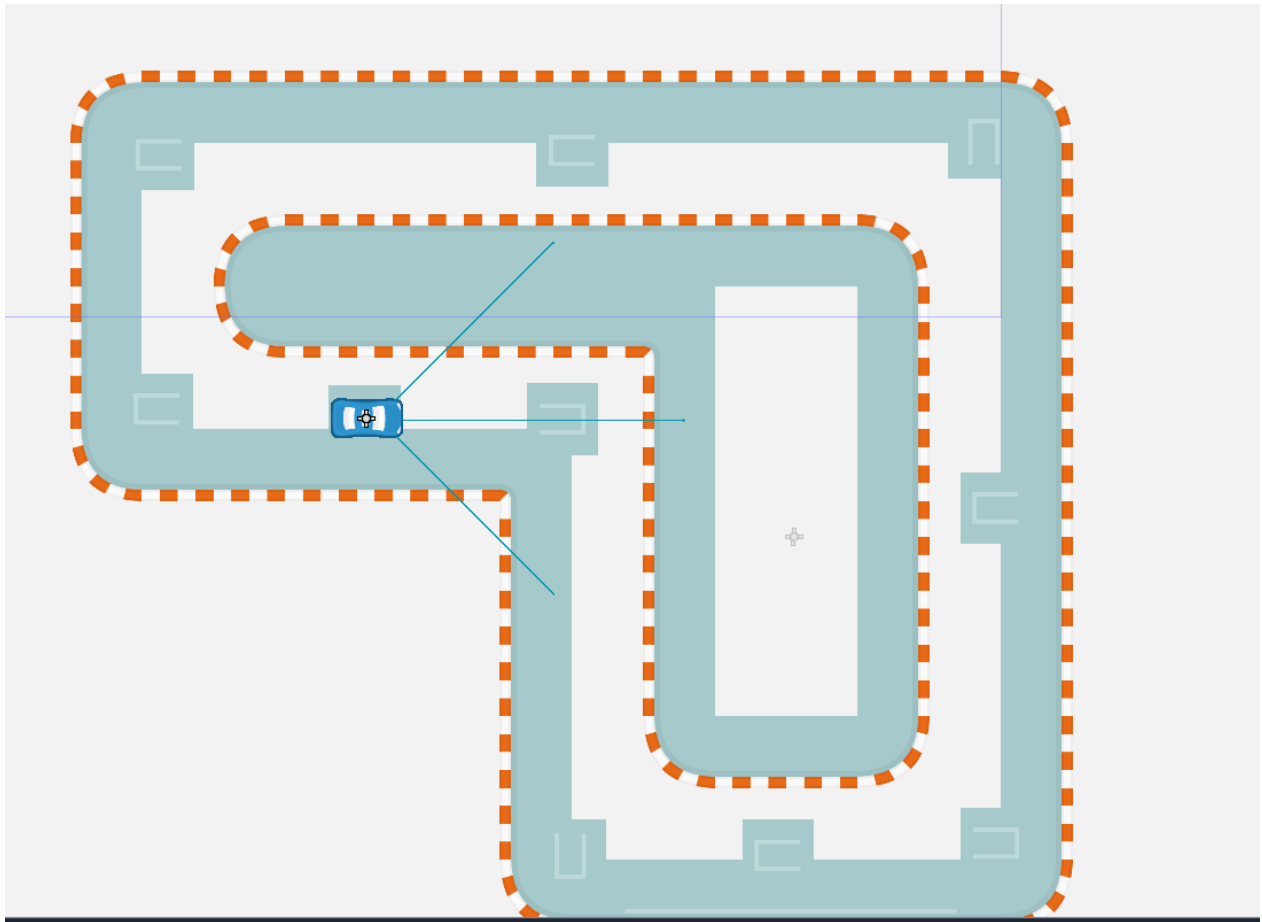


Figure 1: Screenshot of the simulation environment during training

3.1.2. Route Tracking Setup

The navigation setup is similar to the training setup, except that here there are some nodes, which are really helpful while trying to obtain the shortest path. Specifically, each node is connected with each node if they can be connected directly, without passing over an obstacle or any other collision area. Also, a target was added, which is the destination point.

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

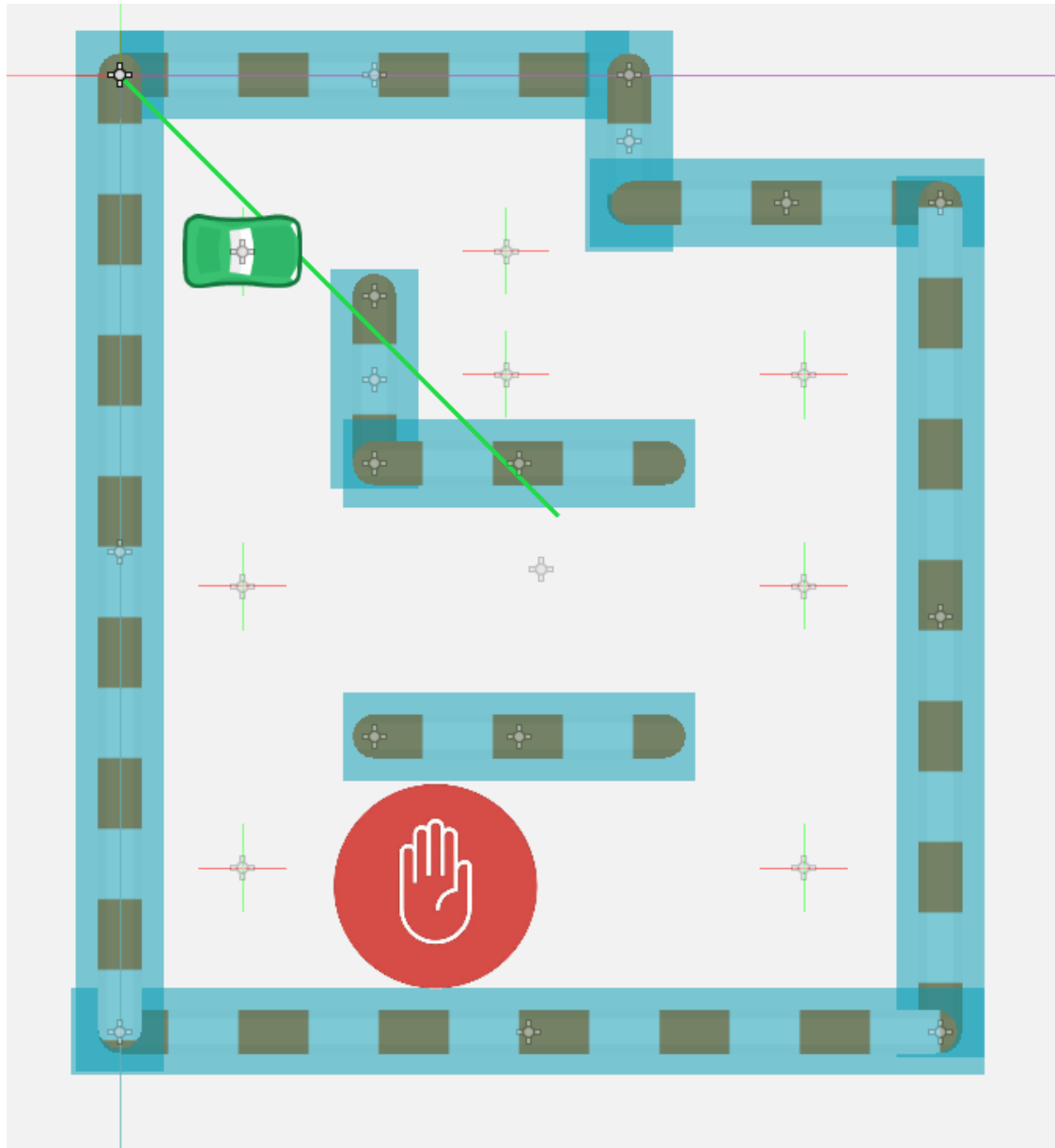


Figure 2: Screenshot of the simulation environment during route tracking

3.2. Hardware Setup

The hardware components used during experiments are the ones analyzed in the Technical Complexity section of the Method part. Specifically, there is a miniature car, with 4 DC motors where the distance sensors and the motor drivers are integrated. Motor drivers control DC motors. Motor drivers and distance sensors are controlled by ZedBoard.

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

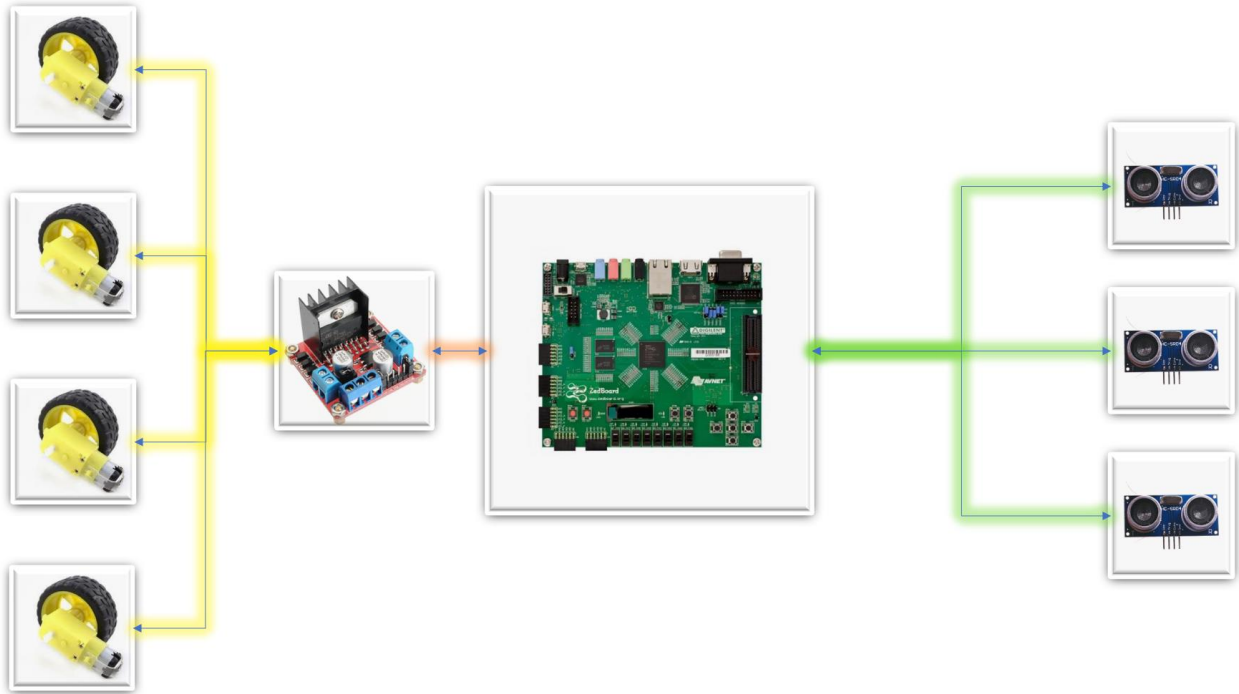


Figure 3: Block Diagram of hardware setup.

3.3. SoC Setup

The SoC setup of this project can be separated into three main parts, each of which represents a separate IP. These parts are:

1. Connection of ZedBoard with Distance Sensors
2. Neural Network Implementation
3. Connection of ZedBoard with the Motor Driver

Two of the parts are organized and analyzed under the Vivado setup section, and the last one is developed by HLS. When these three IPs are connected to the main two IPs of the Zynq processing system (Zynq IP and reset IP), the SoC part of the project is completed.

3.3.1 Vivado Setup

There are three distance sensors which work as follows: The distance sensors send a trigger signal and after that they are waiting for an echo signal. After the trigger signal is sent, when the echo signal goes high, a timer is initialized. The time elapsed from that moment to the moment where echo goes low is converted to distance by using a special formula. These distance values may be kept somewhere in order to be used while deciding for the action in the neural network. The designed IP sends a trigger signal for the 3 distance sensors, keeps the time period until receiving an echo signal and converts that value to

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

distance. Furthermore, this IP is connected with the PS part of the board, which can read that distance values via AXI interconnect.

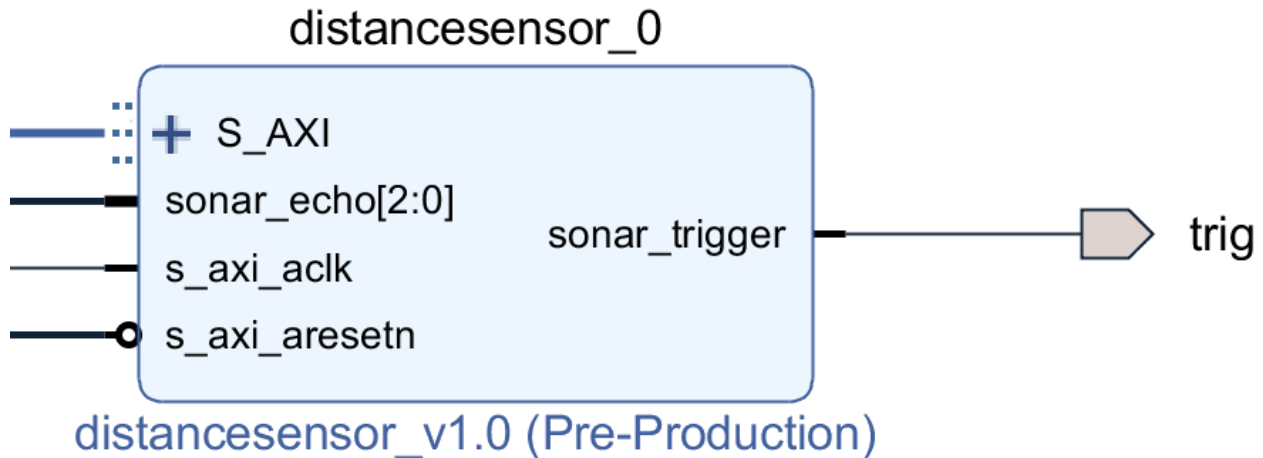


Figure 4: Distance sensor IP.

The ZedBoard, after deciding which action is the best, sends PWM signals to the motor drivers. An IP was created in order to perform this operation. The enable signals determine how fast will the motors turn, while the input signals define the direction to which the motors will turn to.

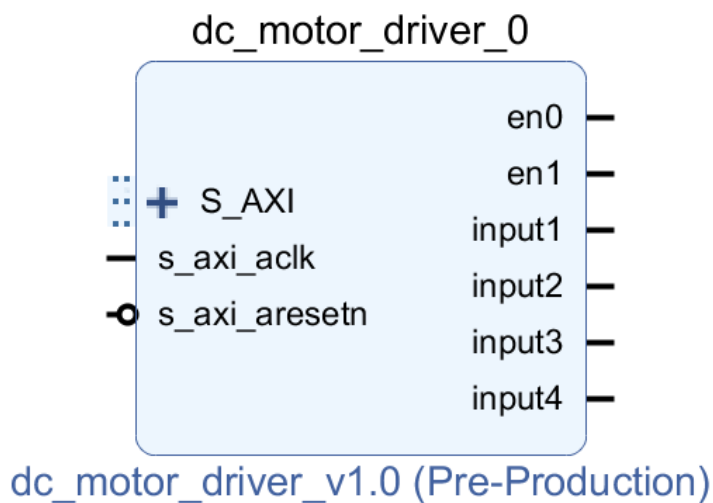


Figure 5: Motor driver IP.

SelfBulance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBulance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

3.3.2 HLS Setup

The inference of neural networks is done in HLS. HLS will use the current position of the car with respect to the environment and will tell the car which action it must choose in order not to collide. A trained model means that weights of the neural network are properly configured. For the HLS setup, those weights were used in order to compute which action is the best. Functions needed for the inference were formed. When these functions are exported to Vivado as RTL, an IP repository is created. This IP is connected to the other IPs created in Vivado. The inputs of this IP are the distance values read from the first IP, and the output is an action which will determine the PWM values needed in the third IP.

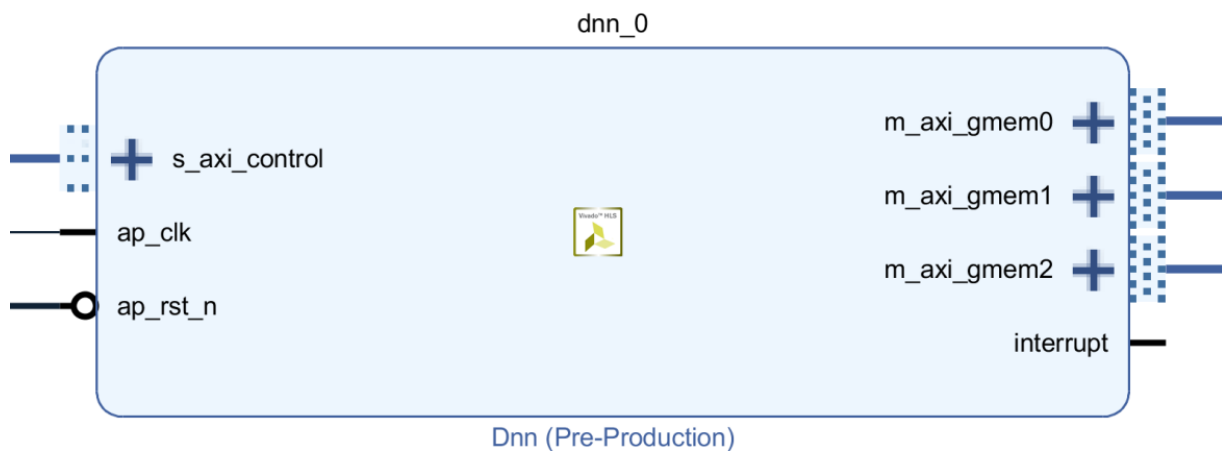


Figure 6: Deep Neural Network IP.

4. Empirical Results

4.1. Simulation Results

For the training part the big deal is choosing the appropriate reward values, while taking account of the neural network parameters. Neural network parameters are the optimizer, the loss function, the number of hidden layers, the number of neurons of each layer, as well as the number of the neural networks. For the training of this simulation model the proper parameter values are as below:

- 1) The optimizer used is Adam Optimizer
- 2) There are 3 neural networks, one for each action
- 3) Each neural network has 1 hidden layer
- 4) Each hidden layer has 14 neurons
- 5) The loss function used is mean square error

The reward values chosen according to these parameters are:

- 1) -1000 if colliding

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

2) 0, otherwise

Based on these parameters, the obtained weights are summarized below:

```
float hidden_weights0[3][8] =  
  
{{-3.84956918656826, 0.10099666798487306, -0.26281410176306963, -1.103420589119196, -  
4.610216394066811, -5.943972408771515, 4.866241738200188, -1.3819764107465744},{-  
0.0033874125801958144, 2.931508183479309, 5.079740032553673, -0.33691348135471344, -  
5.275746792554855, -2.565868243575096, 4.172136649489403, -1.4251078739762306},{-  
8.3496812582016, 8.898481786251068, -1.2333372496068478, 6.986190855503082, -  
10.993932366371155, -8.0346859395504, 7.457759141921997, -1.6926671415567398}  
  
};  
  
float hidden_weights1[3][8] =  
  
{{-4.535074800252914, 2.4278768822550774, 3.834023416042328, 0.6179178971797228, -  
0.5067280009388924, -3.793211504817009, -0.2932571657001972, -1.5097806751728058},{-  
0.241254648193717, -2.9105860590934753, 3.6026095747947693, -1.7204205617308617, -  
4.876620814204216, -2.557615488767624, 0.9422987177968025,  
0.3087730910629034},{2.8951516896486282, 5.526160150766373, -1.9037997275590897,  
1.3021478354930878, 5.0027944296598434, 7.306441426277161, -0.007503096159780398, -  
3.25497530400753}  
  
};  
  
float hidden_weights2[3][8] =  
  
{{-0.42635701037943363, -2.34987985342741, 2.2929700165987015, -1.1890034526586533,  
2.5724257305264473, 4.73761448264122, -1.0739358365535736, -  
0.08009000890888274},{1.007728859782219, 1.668719194829464, 2.541242815554142,  
3.205123007297516, 2.2805728912353516, 0.09745326871052384, 2.8159808814525604,  
4.265208497643471},{-2.137724630534649, 0.30516319535672665, -1.1025098226964474,  
0.8565045520663261, -4.191138356924057, -4.299159228801727, -5.019451394677162,  
1.7925389930605888}  
  
};
```

Figure 7: The weights of the hidden layer

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

```
float output_weights0[8] =  
{1.9137564897537231, -1.4069651365280151, 0.5737425088882446, -  
1.54334557056427, 1.9288161993026733, 1.8377007246017456, -  
1.6357892751693726, 0.3111952543258667};  
  
float output_weights1[8] =  
{-0.3641203045845032, -0.995966374874115, 0.812775194644928, -0.38561099767684937, -  
0.8587726950645447, -2.3841493129730225, 0.07397675514221191, 1.1198803186416626};  
  
float output_weights2[8] =  
{0.20045220851898193, 0.027851002290844917, 0.8216999173164368, 0.8630620837211609, 0.599  
6399521827698, 0.912432849407196, 0.8935238718986511, 0.17318430542945862};
```

Figure 8: *The weights of the output layer.*

4.1.2. Navigation Results

The navigation part of this project in the first step is using an algorithm, named AStar algorithm. In order to be able to use the AStar algorithm in each way the programmer wants, nodes were created in the simulation environment. For example, for this project, only the nodes that don't pass over an obstacle were connected. When nodes were connected, the shortest route was created and drawn on the environment. This is done by looking to the connected nodes and deciding which set of nodes lead to the shortest path. In order to have a smooth route, the connections from one node to another node were converted to Bezier curves, which give smoothness at the connection points.

It was planned to use a sensor in order to find the location of the car in real time. GPS was not used, as GPS values are not very specific and because our physical environment was small, GPS would not lead to the desired location data. Instead, we tried to use another module, which gets acceleration values. It was planned to integrate two times these acceleration values in order to obtain the position data, but the acceleration values were not as they were supposed to be, so it could not be used.

As a final solution, we decided not to use location data. Instead, when the physical ambulance has to start, the shortest path that is leading to the destination will be found in the simulation environment and a description of this route will be sent to the ambulance. Specifically, the data sent will tell the car to go some amount of meters straight, turn left, go straight again for some amount of meters etc.

The route information was sent to Python from where it was sent to the ZedBoard. Python does not do any extra operation that SDK could do, but it is needed as Godot communicates only via socket, while SDK accepts data through UART(serial communication). So Python takes the route data from Godot and sends it to ZedBoard, which in turns sends each action to the motor driver. When the data is the amount of meters that the car must go, the action sent is the straight action and delay is put on SDK, equal to the

SelfBulance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBulance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

amount of meters that the car should go. During the route tracking, if there is any obstacle on the road, Deep Q Learning algorithm is used in order to avoid it.

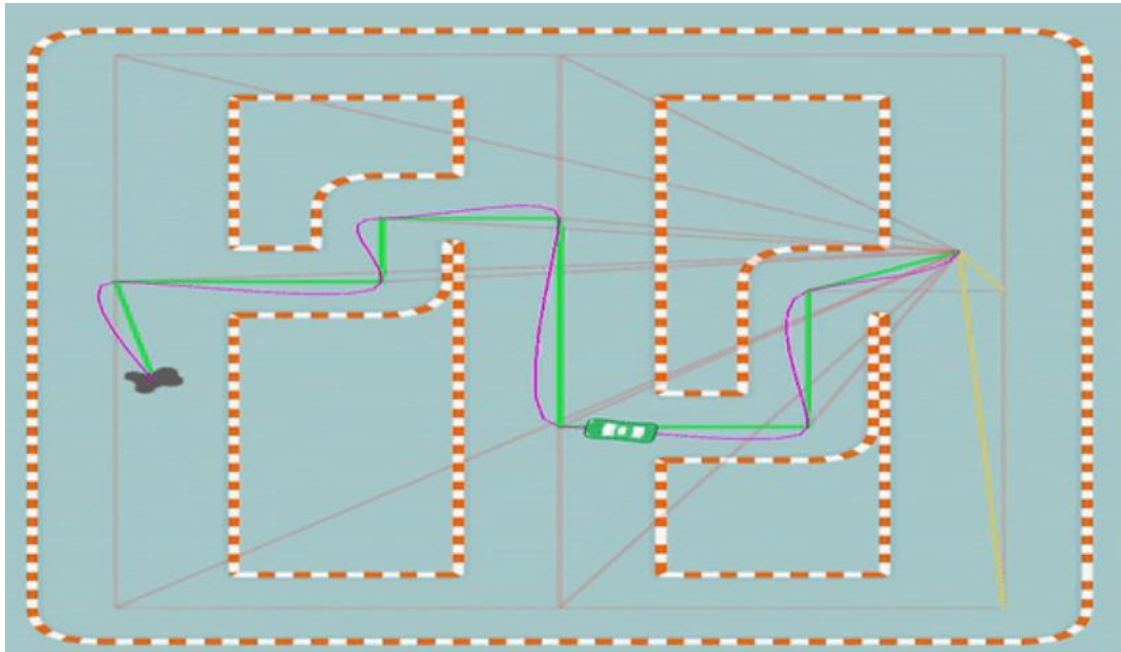


Figure 9: Screenshot of the model when following the shortest path.

4.2. Hardware Results

The hardware setup of this project is shown in Figure 10. Distance sensors are connected to the Zedboard, but a voltage converter exists between them, as the 5V of distance sensors must be converted to 3.3V in order to avoid any damage on the ZedBoard. ZedBoard is connected to the PC via UART communication and the motor drivers are connected directly to the ZedBoard, too.

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

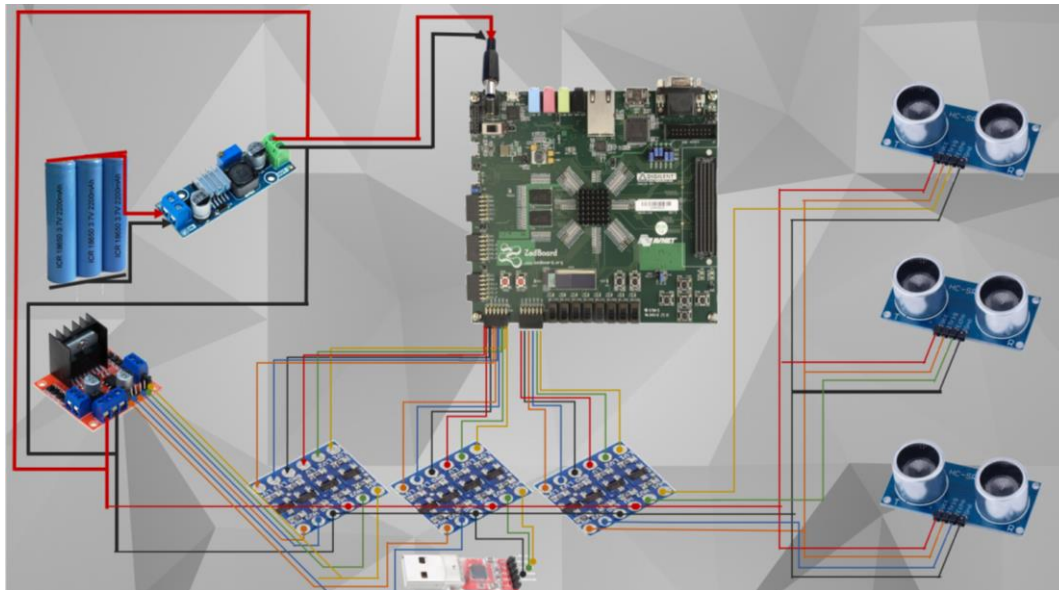


Figure 10: Block diagram of the connection of hardware components.

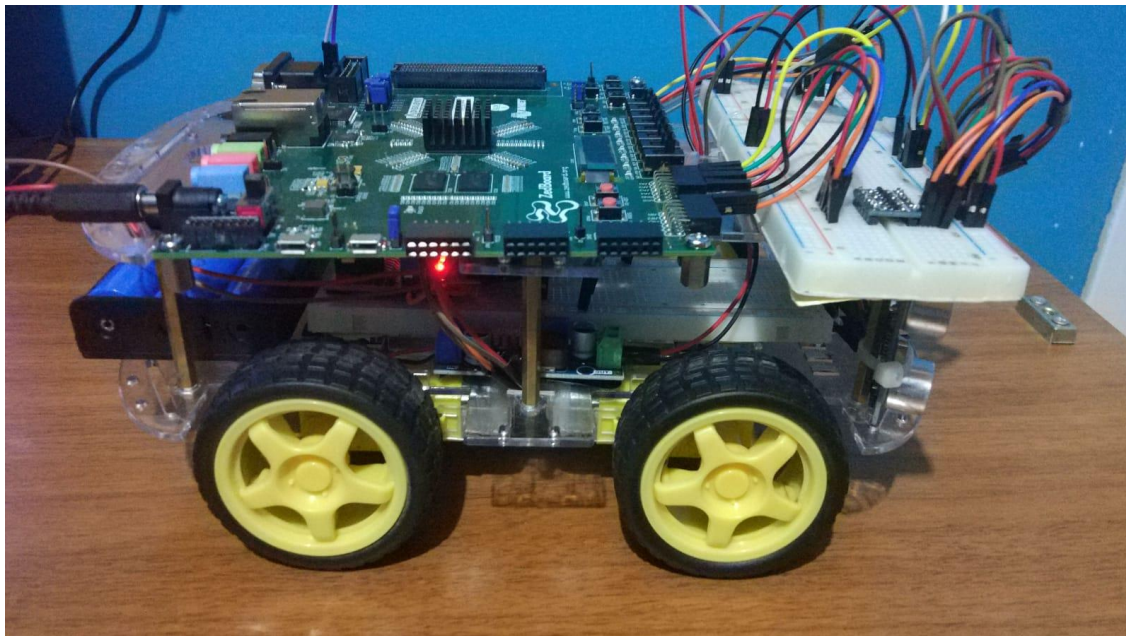


Figure 11: Hardware implementation of model car.

4.3. SoC Results

The full IP diagram of the project is shown below.

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

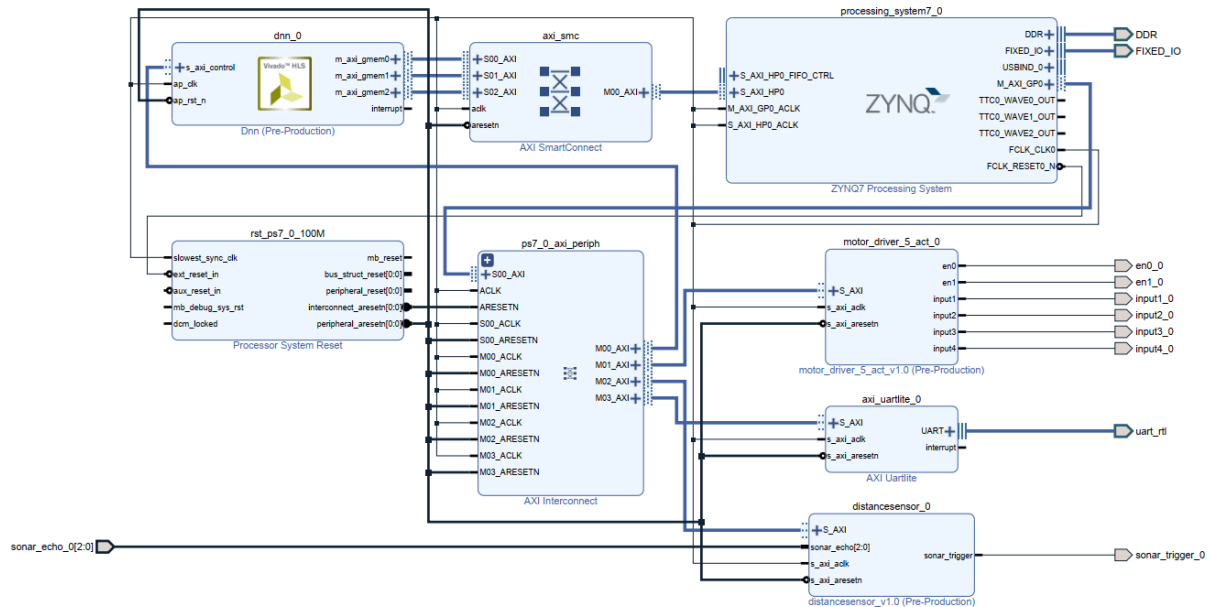


Figure 12: Total IP Diagram.

4.3.1 Vivado Results

While designing the IP related to the distance sensors, two interconnects were used. Firstly, AXI Full Interconnection was used in order to read these distances as burst data and save them onto BRAM. But this method did not help us as we had some difficulties while reading the distance values. Secondly, saving values into registers was tried, via AXI Lite Interconnection, and the desired results were obtained.

Additionally, an IP which controls only one of the three distance sensors was developed. We thought that it could be very efficient in terms of reusability, as the programmer will add as much IPs as he needs, without changing the code. But it was not efficient in terms of hardware, as each IP requires another AXI Interconnect, so it was not used during this project.

SelfBulance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBulance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

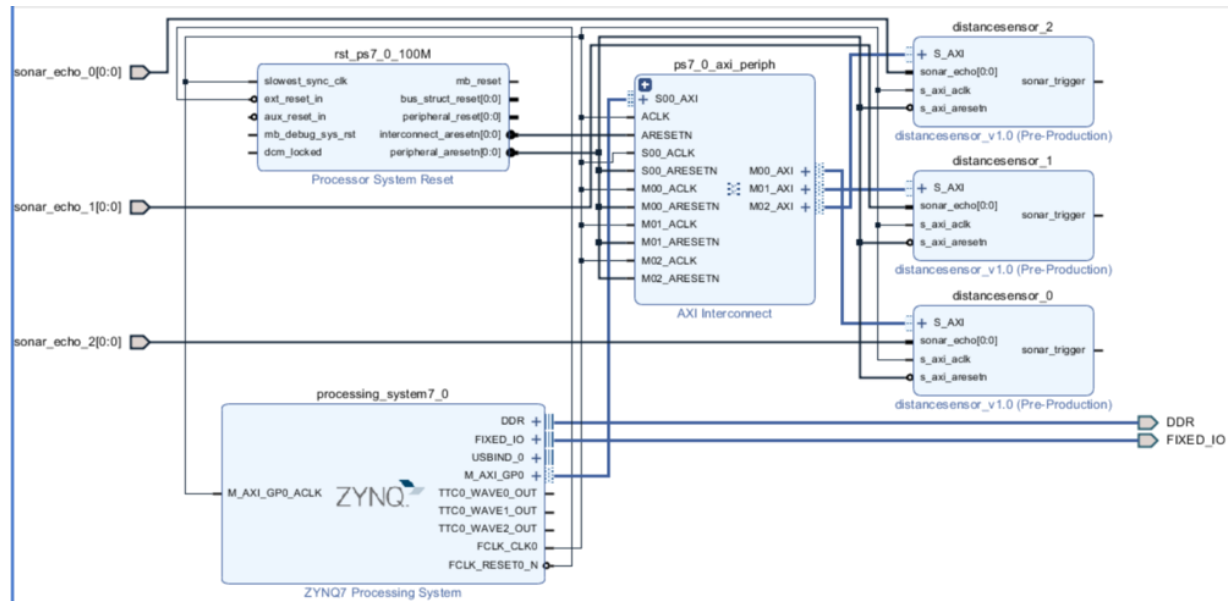


Figure 13: IP Diagram with the method of separate IP for each distance sensor.

Another optimization done for the distance sensors was using one IP, as planned in the beginning of this project, but the parameters of the IP, such as the number of sensors, will be user-defined from the customization parameters window on Vivado. This method was really helpful, and will be analyzed under the reusability section of this report.

The verification of the distance sensors functionality was done via SDK, and the results are given in Figure 14:

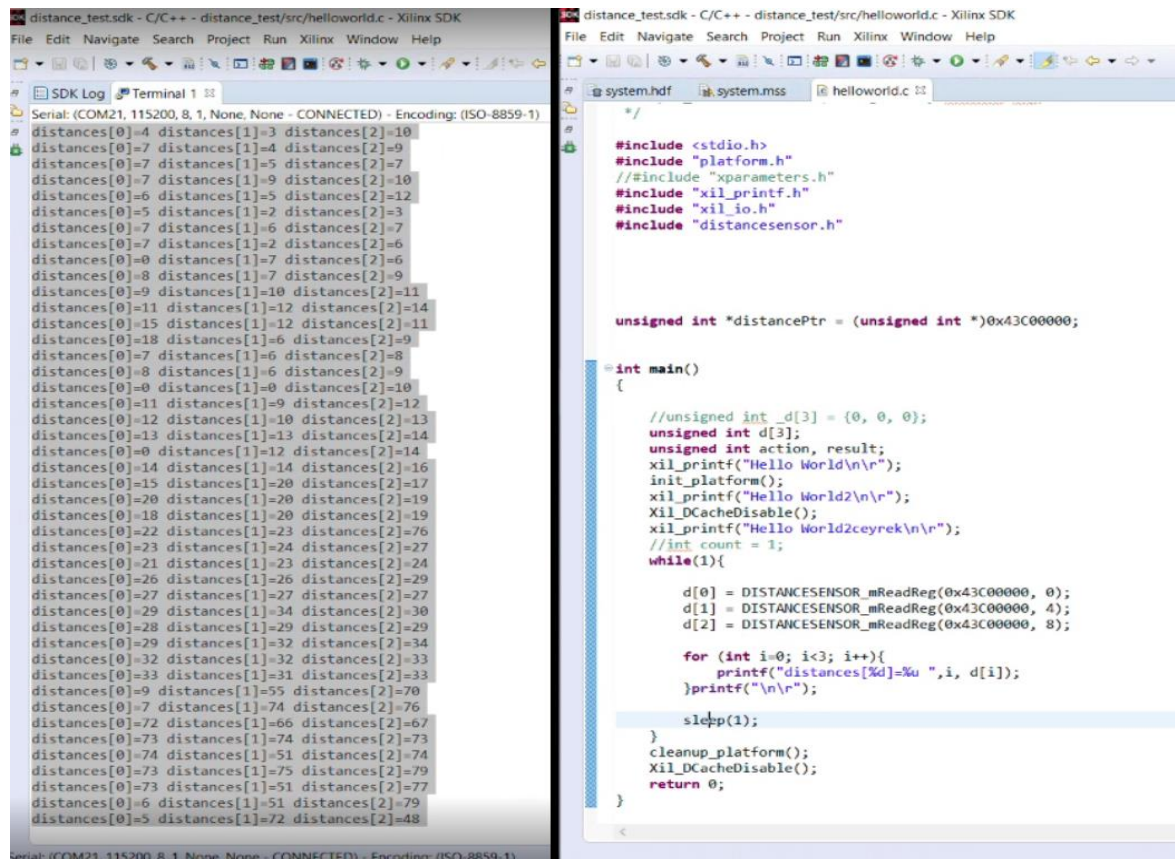
SelfBulance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBulance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>



The image shows a screenshot of the Xilinx SDK environment. On the left, a terminal window displays a stream of distance sensor data in a 3x3 grid format. On the right, a code editor shows the C source code for the application.

```
Serial: (COM21, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
distances[0]=-4 distances[1]=3 distances[2]=-10
distances[0]=7 distances[1]=-4 distances[2]=9
distances[0]=7 distances[1]=5 distances[2]=7
distances[0]=7 distances[1]=9 distances[2]=-10
distances[0]=6 distances[1]=5 distances[2]=-12
distances[0]=5 distances[1]=2 distances[2]=3
distances[0]=7 distances[1]=6 distances[2]=7
distances[0]=7 distances[1]=2 distances[2]=6
distances[0]=0 distances[1]=7 distances[2]=6
distances[0]=8 distances[1]=7 distances[2]=9
distances[0]=9 distances[1]=10 distances[2]=-11
distances[0]=11 distances[1]=12 distances[2]=-14
distances[0]=15 distances[1]=-12 distances[2]=-11
distances[0]=18 distances[1]=6 distances[2]=-9
distances[0]=7 distances[1]=6 distances[2]=8
distances[0]=8 distances[1]=6 distances[2]=9
distances[0]=0 distances[1]=0 distances[2]=-10
distances[0]=11 distances[1]=9 distances[2]=-12
distances[0]=12 distances[1]=-10 distances[2]=-13
distances[0]=13 distances[1]=13 distances[2]=-14
distances[0]=0 distances[1]=12 distances[2]=-14
distances[0]=14 distances[1]=14 distances[2]=-16
distances[0]=15 distances[1]=20 distances[2]=-17
distances[0]=20 distances[1]=20 distances[2]=-19
distances[0]=18 distances[1]=20 distances[2]=-19
distances[0]=22 distances[1]=23 distances[2]=-76
distances[0]=23 distances[1]=24 distances[2]=-27
distances[0]=21 distances[1]=23 distances[2]=-24
distances[0]=26 distances[1]=26 distances[2]=-29
distances[0]=27 distances[1]=27 distances[2]=-27
distances[0]=29 distances[1]=34 distances[2]=-30
distances[0]=28 distances[1]=29 distances[2]=-29
distances[0]=29 distances[1]=32 distances[2]=-34
distances[0]=32 distances[1]=32 distances[2]=-33
distances[0]=33 distances[1]=31 distances[2]=-33
distances[0]=9 distances[1]=55 distances[2]=-70
distances[0]=7 distances[1]=74 distances[2]=-76
distances[0]=72 distances[1]=66 distances[2]=-67
distances[0]=73 distances[1]=74 distances[2]=-73
distances[0]=74 distances[1]=51 distances[2]=-74
distances[0]=73 distances[1]=75 distances[2]=-79
distances[0]=73 distances[1]=51 distances[2]=-77
distances[0]=6 distances[1]=51 distances[2]=-79
distances[0]=5 distances[1]=72 distances[2]=-48
```

```
/*
#include <stdio.h>
#include "platform.h"
// #include "xparameters.h"
#include "xil_printf.h"
#include "xil_io.h"
#include "distancesensor.h"

unsigned int *distancePtr = (unsigned int *)0x43C00000;

int main()
{
    // unsigned int d[3] = {0, 0, 0};
    unsigned int d[3];
    unsigned int action, result;
    xil_printf("Hello World\n\n");
    init_platform();
    xil_printf("Hello World2\n\n");
    Xil_DCacheDisable();
    xil_printf("Hello World2ceyrek\n\n");
    // int count = 1;
    while(1){
        d[0] = DISTANCESENSOR_mReadReg(0x43C00000, 0);
        d[1] = DISTANCESENSOR_mReadReg(0x43C00000, 4);
        d[2] = DISTANCESENSOR_mReadReg(0x43C00000, 8);

        for (int i=0; i<3; i++){
            printf("distances[%d]=%u ", i, d[i]);
        } printf("\n\n");

        sleep(1);
    }
    cleanup_platform();
    Xil_DCacheDisable();
    return 0;
}
```

Figure 14: Testing the distance sensors.

4.3.2 HLS Results

The weight values of each network of the trained model were obtained from Keras and saved on header files. The same input values were given both to Python and to C codes in order to verify that C codes are working properly. The maximum q values of both codes were the same. After that, C synthesis was done, and the results shown in Figure 14 were obtained. The area utilizations during this experiment are small, as only the inference part of the model was included.

SelfBulance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBulance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

Performance Estimates					Utilization Estimates					
▣ Timing (ns)					▣ Summary					
▣ Summary					Name	BRAM_18K	DSP48E	FF	LUT	
Clock	Target	Estimated	Uncertainty		DSP	-	-	-	-	
ap_clk	10.00	7.47	1.25		Expression	-	-	0	8	
▣ Latency (clock cycles)					FIFO	-	-	-	-	
▣ Summary					Instance	-	15	1545	2778	
Latency	Interval				Memory	3	-	192	21	
min	max	min	max	Type	Multiplexer	-	-	-	15	
549	549	549	549	none	Register	-	-	5	-	
						Total	3	15	1742	2822
						Available	280	220	106400	53200
						Utilization (%)	1	6	1	5

Figure 15: C Synthesis results of the neural network functions in HLS.

As the verification of the header files was done, the next step was taking the distance values saved on the slave registers and inserting them as input to the neural network. The action was calculated in terms of the maximum q values. In order to test the functionality of that IP, it was exported, and it was added to the already existing IP diagram. Both the distance sensors and the maximum q values with the action were written to RAM. Address was assigned from the address editor tab, and an HP port was used in order to read from and write to RAM without stalling the PS tasks. In SDK, the verification of this IP was done. The results are shown below:

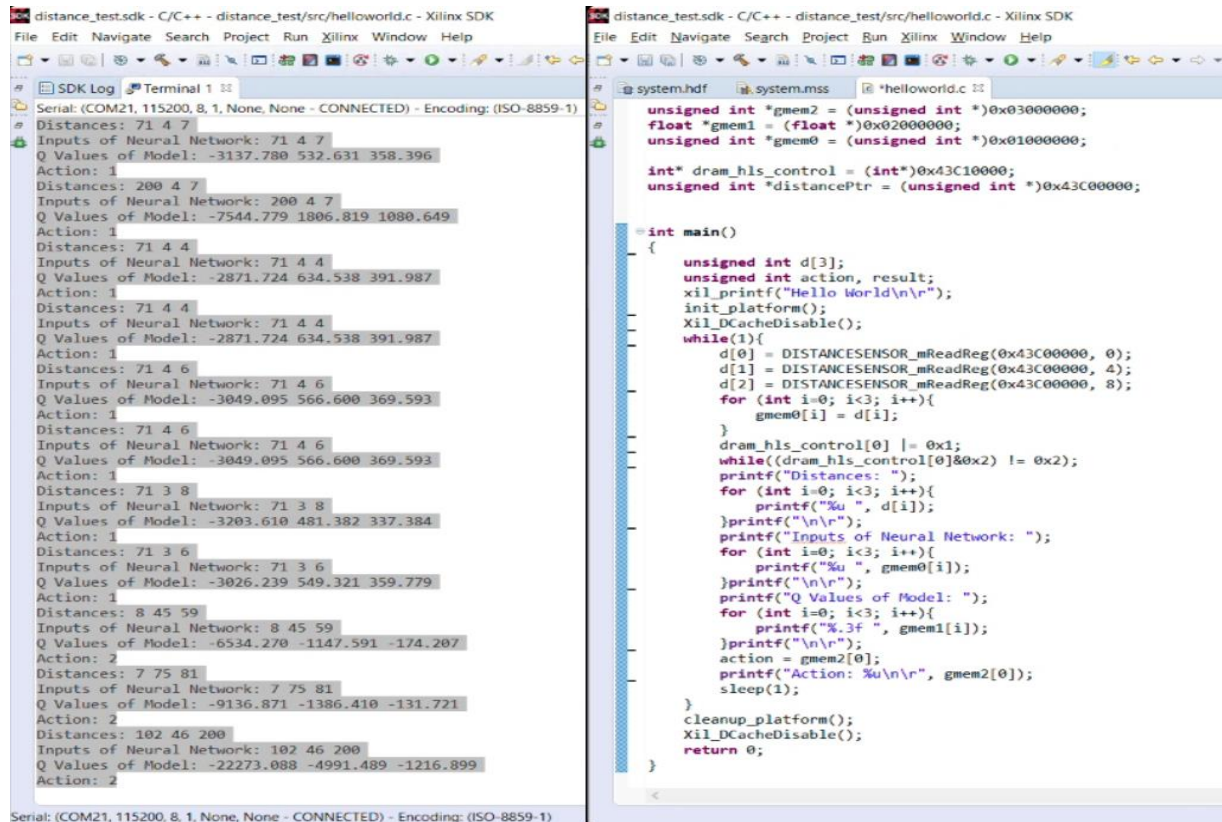
SelfBulance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBulance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>



```
distance_test.sdk - C/C++ - distance_test/src/helloworld.c - Xilinx SDK
File Edit Navigate Search Project Run Xilinx Window Help

Serial: (COM21, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
SDK Log Terminal 1
Distances: 71 4 7
Inputs of Neural Network: 71 4 7
Q Values of Model: -3137.780 532.631 358.396
Action: 1
Distances: 200 4 7
Inputs of Neural Network: 200 4 7
Q Values of Model: -7544.779 1806.819 1080.649
Action: 1
Distances: 71 4 4
Inputs of Neural Network: 71 4 4
Q Values of Model: -2871.724 634.538 391.987
Action: 1
Distances: 71 4 4
Inputs of Neural Network: 71 4 4
Q Values of Model: -2871.724 634.538 391.987
Action: 1
Distances: 71 4 6
Inputs of Neural Network: 71 4 6
Q Values of Model: -3049.095 566.600 369.593
Action: 1
Distances: 71 4 6
Inputs of Neural Network: 71 4 6
Q Values of Model: -3049.095 566.600 369.593
Action: 1
Distances: 71 3 8
Inputs of Neural Network: 71 3 8
Q Values of Model: -3203.610 481.382 337.384
Action: 1
Distances: 71 3 6
Inputs of Neural Network: 71 3 6
Q Values of Model: -3026.239 549.321 359.779
Action: 1
Distances: 8 45 59
Inputs of Neural Network: 8 45 59
Q Values of Model: -6534.270 -1147.591 -174.207
Action: 2
Distances: 7 75 81
Inputs of Neural Network: 7 75 81
Q Values of Model: -9136.871 -1386.410 -131.721
Action: 2
Distances: 102 46 200
Inputs of Neural Network: 102 46 200
Q Values of Model: -22273.088 -4991.489 -1216.899
Action: 2
Serial: (COM21, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)

distance_test.sdk - C/C++ - distance_test/src/helloworld.c - Xilinx SDK
File Edit Navigate Search Project Run Xilinx Window Help

system.hdf system.mss helloworld.c
unsigned int *gmem2 = (unsigned int *)0x03000000;
float *gmem1 = (float *)0x02000000;
unsigned int *gmem0 = (unsigned int *)0x01000000;

int* dram_hls_control = (int*)0x43C10000;
unsigned int *distancePtr = (unsigned int *)0x43C00000;

int main()
{
    unsigned int d[3];
    unsigned int action, result;
    xil_printf("Hello World\n\r");
    init_platform();
    Xil_DCacheDisable();
    while(1){
        d[0] = DISTANCESENSOR_mReadReg(0x43C00000, 0);
        d[1] = DISTANCESENSOR_mReadReg(0x43C00000, 4);
        d[2] = DISTANCESENSOR_mReadReg(0x43C00000, 8);
        for (int i=0; i<3; i++){
            gmem0[i] = d[i];
        }
        dram_hls_control[0] |= 0x1;
        while((dram_hls_control[0]&0x2) != 0x2);
        printf("Distances: ");
        for (int i=0; i<3; i++){
            printf("%u ", d[i]);
        }
        printf("\n\r");
        printf("Inputs of Neural Network: ");
        for (int i=0; i<3; i++){
            printf("%u ", gmem0[i]);
        }
        printf("\n\r");
        printf("Q Values of Model: ");
        for (int i=0; i<3; i++){
            printf("%3f ", gmem1[i]);
        }
        printf("\n\r");
        action = gmem2[0];
        printf("Action: %u\n\r", gmem2[0]);
        sleep(1);
    }
    cleanup_platform();
    Xil_DCacheDisable();
    return 0;
}
```

Figure 16: Testing the DNN IP.

After the verification, possible optimizations were done in the HLS part. Firstly, we had three neural networks, one for each action, and for each neural network there was a separate function, as the weights were not equal, and each weight set was saved on a separate header file. These header files were merged, and the function, too. The part of the code which had taken the weight values was modified. Each time a weight value must be taken, it will choose the appropriate weight set and take weight values from there.

Also, each parameter used was converted to variable. Number of actions, number of distance sensors, number of neural networks were all defined as variables. This improvement was really efficient in terms of reusability.

Below, you can see the utilization results of the first dnn IP(Figure 17) versus the utilization results of the last dnn IP(Figure 18), which includes all the optimizations.

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

- **Summary**

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	300
FIFO	-	-	-	-
Instance	6	30	6366	10029
Memory	-	-	-	-
Multiplexer	-	-	-	1069
Register	-	-	161	-
Total	6	30	6527	11398
Available	280	220	106400	53200
Utilization (%)	2	13	6	21

- **Detail**

- **Instance**

Instance	Module	BRAM_18K	DSP48E	FF	LUT
grp_doit0_fu_169	doit0	0	10	1574	2593
grp_doit1_fu_145	doit1	0	10	1544	2589
grp_doit2_fu_157	doit2	0	10	1544	2589
topfunction_control_s_axi_U	topfunction_control_s_axi	0	0	36	40
topfunction_fcmp_ncg_U35	topfunction_fcmp_ncg	0	0	66	239
topfunction_fcmp_ncg_U36	topfunction_fcmp_ncg	0	0	66	239
topfunction_gmem0_m_axi_U	topfunction_gmem0_m_axi	2	0	512	580
topfunction_gmem1_m_axi_U	topfunction_gmem1_m_axi	2	0	512	580
topfunction_gmem2_m_axi_U	topfunction_gmem2_m_axi	2	0	512	580
Total	9	6	30	6366	10029

- **DSP48**

Figure 17: Utilization results of the first DNN IP.

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

- **Summary**

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	173
FIFO	-	-	-	-
Instance	6	10	3474	4831
Memory	-	-	-	-
Multiplexer	-	-	-	251
Register	-	-	158	-
Total	6	10	3632	5255
Available	280	220	106400	53200
Utilization (%)	2	4	3	9

- **Detail**

- **Instance**

Instance	Module	BRAM_18K	DSP48E	FF	LUT
grp_doit_fu_150	doit	0	10	1836	2812
topfunction_control_s_axi_U	topfunction_control_s_axi	0	0	36	40
topfunction_fcmp_kbM_U27	topfunction_fcmp_kbM	0	0	66	239
topfunction_gmem0_m_axi_U	topfunction_gmem0_m_axi	2	0	512	580
topfunction_gmem1_m_axi_U	topfunction_gmem1_m_axi	2	0	512	580
topfunction_gmem2_m_axi_U	topfunction_gmem2_m_axi	2	0	512	580
Total	6	6	10	3474	4831

- **DSP48**

Figure 18: Utilization results of the last DNN IP.

5. Reusability

5.1 Reusability of the Training Setup

Sources regarding the training are published on the GitHub repository. All setup can be used by other developers who are interested in developing autonomous vehicles by using reinforcement learning. Setup can be reused in different approaches, as both small changes in parameters like number of hidden layers and neurons, or large changes such as the reward mechanism or number of sensors on the vehicle, can lead to different results.

5.2 Reusability of the HLS Setup

By defining some parameters with pragmas, neural network code can be reused by simply changing these parameters. Specifically, when these parameters are defined as pragmas, reusability is allowed. Anyone that wants to infer a fully connected neural network with a single hidden layer can easily use this code.

SelfBalance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBalance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

5.3 Reusability of the Vivado Setup

Both distance sensor IPs and motor driver IPs are designed in a way which can be reused from any project where motor driver and distance sensors will be used. As can be seen in Figure 19, in order to reuse the IP cores, necessary parameters such as sensor numbers and input/output size of the network will be generic for our IP cores. One that wants to use these IPs for different needs can easily integrate these IPs into that project. A distance sensor used with this IP can have a range of values between 0-10 meters.

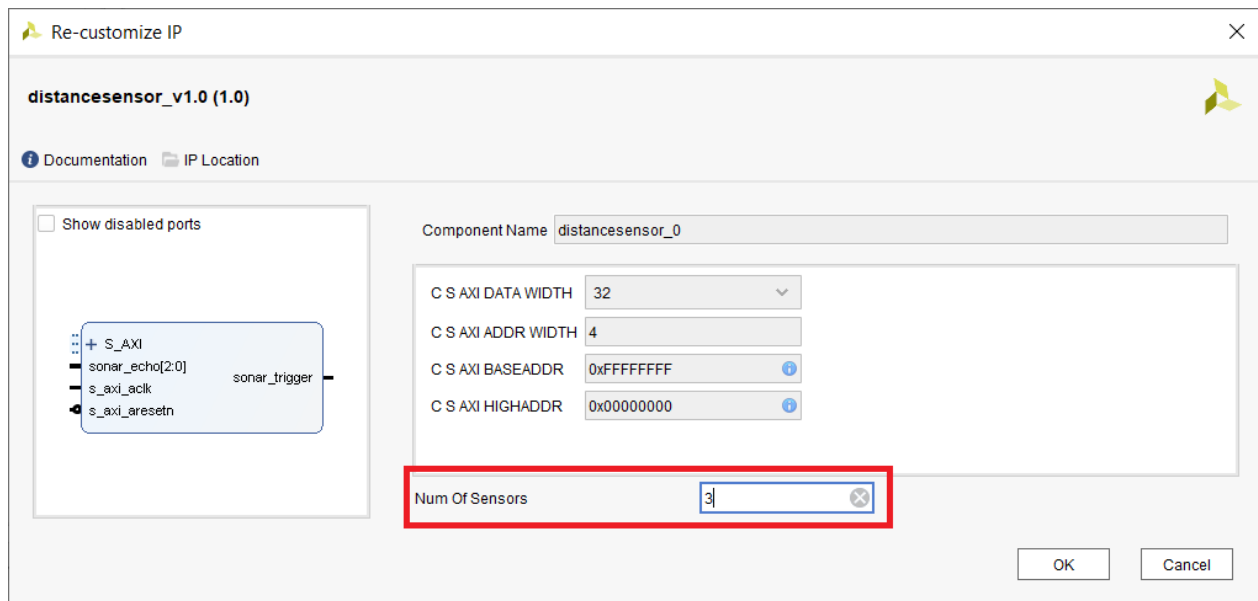


Figure 19: Parameterized number of sensors. The IP can now be used for different scenarios.

6. Conclusion

While trying to contribute to the design and development of autonomous vehicles by implying this project, some results have been obtained. Deep Q Learning turned out to be a good choice while implementing neural networks. Also, SoC is really helpful, especially for accelerating the inference, as there are three neural networks that must run concurrently in order to achieve better performance.

References

- [1] Chohra, A., Farah, A. & Benmehrez, C. Neural Navigation Approach for Intelligent Autonomous Vehicles (IAV) in Partially Structured Environments. *Applied Intelligence* 8, 219–233 (1998).
- [2] Smola, Alex, and S. V. N. Vishwanathan. "Introduction to machine learning." *Cambridge University, UK* 32 (2008): 34.
- [3] Kaelbling, Leslie P.; Littman, Michael L.; Moore, Andrew W. (1996). "Reinforcement Learning: A Survey". *Journal of Artificial Intelligence Research*. Cilt 4, s. 237–285
- [4] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M.

SelfBulance: A Self Driving Ambulance via Deep Q Learning: An SoC Solution

Asli Zoumpoul & Yasin Alptekin

Academic Advisor: Assist. Prof. Dr. İsmail San

Source Codes: <https://github.com/yasinalp/SelfBulance>

Presentation Video: <https://www.youtube.com/watch?v=iiC7OrYKr7g>

(2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[5] <https://godotengine.org/>

[6] UDP, User Datagram Protocol, and Datagram Sockets. "User Datagram Protocol." (1980).

[7] Stulp, F., & Verbrugge, R. (2002). A Knowledge-based Algorithm for the Internet Transmission Control Protocol (TCP). Bulletin of Economic Research, 54(1), 69-94.

[8] Crockett, Louise H., et al. The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc. Strathclyde Academic Media, 2014.

[9] <https://www.motorobit.com/urun/12v-3000rpm-16mm-reduktorlu-dc-motor-cizgi-izleyen-robot>

[10] <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>

[11] <https://pdf.direnc.net/upload/adafruit-motor-shield-v2-for-arduino.pdf>