electronics

Article

Virtualization of Self-Driving Algorithms by Interoperating Embedded Controllers on a Game Engine for a Digital Twining Autonomous Vehicle

Heuijee Yun [1] and Daejin Park [1,2,*]

[1] School of Electronics Engineering, Kyungpook National University, Daegu 41566, Korea; heuijee@knu.ac.kr
[2] School of Electronic and Electrical Engineering, Kyungpook National University, Daegu 41566, Korea
* Correspondence: boltanut@knu.ac.kr; Tel.: +82-53-950-5548

Abstract: Computer simulation based on digital twin is an essential process when designing self-driving cars. However, designing a simulation program that is exactly equivalent to real phenomena can be arduous and cost-ineffective because too many things must be implemented. In this paper, we propose the method using the online game GTA5 (Grand Theft Auto5), as a groundwork for autonomous vehicle simulation. As GTA5 has a variety of well-implemented objects, people, and roads, it can be considered a suitable tool for simulation. By using OpenCV (Open source computer vision) to capture the GTA5 game screen and analyzing images with YOLO (You Only Look Once) and TensorFlow based on Python, we can build a quite accurate object recognition system. This can lead to writing of algorithms for object avoidance and lane recognition. Once these algorithms have been completed, vehicles in GTA5 can be controlled through codes composed of the basic functions of autonomous driving, such as collision avoidance and lane-departure prevention. In addition, the algorithm tested with GTA5 has been implemented with a programmable RC car (Radio control car), DonkeyCar, to increase reliability. By testing those algorithms, we can ensure that the algorithms can be conducted in real time and they cost low power and low memory size. Therefore, we have found a way to approach digital twin technology one step more easily.

Keywords: autonomous driving; simulation; digital twin; lane detection; game engine; OpenCV; DonkeyCar

check for updates

Citation: Yun, H.; Park, D. Virtualization of Self-Driving Algorithms by Interoperating Embedded Controllers on a Game Engine for a Digital Twining Autonomous Vehicle. *Electronics* **2021**, *10*, 2102. https://doi.org/10.3390/electronics10172102

Academic Editor: Alexey Vinel

Received: 2 August 2021
Accepted: 27 August 2021
Published: 30 August 2021

1. Introduction

With day-by-day development of technologies for deep learning and big-data analysis, applications that use artificial intelligence are rapidly expanding. Among them, the development of algorithms using artificial intelligence is indispensable for autonomous driving. When achieved in real life, self-driving technology will be valuable for reducing traffic accidents, achieving efficiency in time, space and other resources and ease of driving for the disabled. However, as autonomous driving technology has several problems, such as stability and safety, virtual-reality simulation is required [1], and even essential. In the design of simulation programs, it is difficult to evaluate and implement all the occasions and objects in the real world [2]. If one tries to construct a simulation that is close to real phenomena, considerable money and time is needed to perform the complex manual work.

There are many factors for simulation, however, we can divide them into two, deterministic and stochastic factors. Deterministic factors are based on information that we can know in advance. Lane features such as a single dashed lane or stop line can be deterministic factors. Characteristics of roads, including safety section, curved road, road width, intersection, crosswalks, and highway can be deterministic factors. This information can be obtained in advance by GPS. Stochastic factors are information that we cannot predict. Traffic lights, pedestrians, or animals can be stochastic factors. The environment of surrounding vehicles such as distances between cars, speed of other cars, cutting off, deflection driving in the lane, and special cars can be stochastic factors. We can process deterministic factors easily, as they can be predicted. However, stochastic factors must be processed in real-time. In the real world, these factors are combined complicated. For example, in a curved road, it can be hard to recognize another vehicle in the next lane. It is difficult to identify vehicles in the next lane on the joined road, such as entry and exit sections. Vehicle recognition is also difficult on roads with slopes or roads with different heights such as underground joining sections and three-dimensional intersections. Therefore, it is necessary to process stochastic factors in consideration of these deterministic factors. To consider every element and respond properly, we have to run an accurate simulation and minimize the risk.

In this work, we suggest GTA5 (Grand Theft Auto), to test an auto-drive simulation and implement it in a real car, as shown in Figure 1. To embody the digital twin, we set two layers, the virtual layer and physical layer. The virtual layer simulates the algorithm of an autonomous vehicle using GTA5. The physical layer deploys those algorithms tested in the virtual layer and sends the data to the network. By the data in the network, we can modify and optimize the algorithm to operate closer and more accurately to reality. The reason why we chose GTA5 as the base for the simulation is that Non-Player Character (NPC)s' movements, driving, and road conditions are realistic [3]. Therefore, self-driving algorithms can be simulated sufficiently without designing the movements of other cars and the paths of nearby pedestrians. In addition, the traffic environments, such as the traffic lights, crosswalks, and signs, are very realistic, and there are police who crackdown on speeding or traffic violations. By using diverse modification (MOD) files, which are user modification files that GTA5 users made for a variety of gameplay, we can set characteristics for each vehicle, such as vehicle braking distance and maximum speed, for a more realistic vehicle customization. Using GTA5, Python-based collision prevention and lane-keeping algorithms can be simulated. We then programmed DonkeyCar, a programmable RC car (Remote Control Car), into Python and tested the algorithms simulated by GTA5 to evaluate if it could work in real-time. In addition, we developed the algorithm not only to detect people, but also to count the number of people and calculate the density.

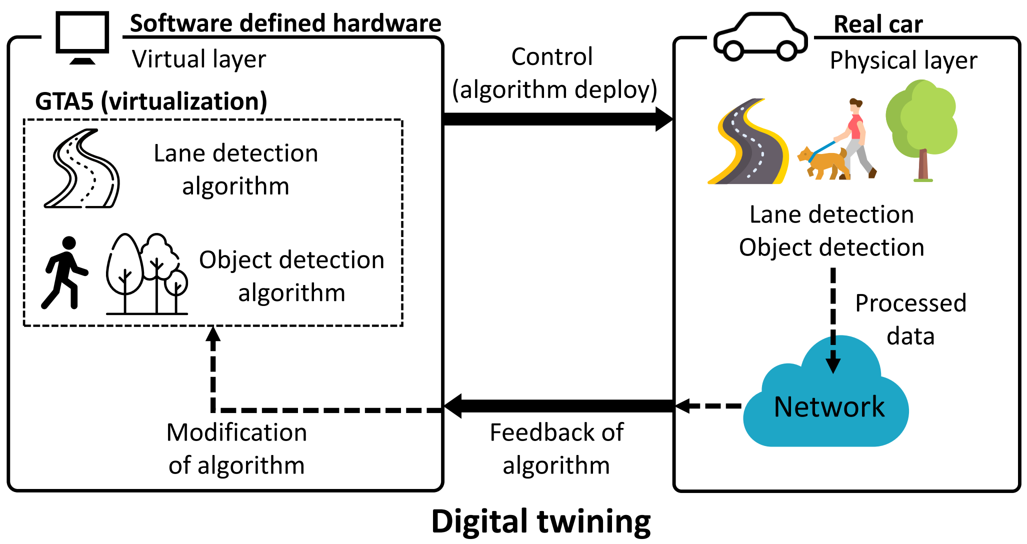

Figure 1. Summary of the structure of implementation.

2. Related Works

Digital twins [4,5], a key element of digital transformation, are technology agnostic [6]. Technology agnostic is a considerably important factor of simulation as it is unbiased toward the use of tools. As autonomous driving technology is rapidly developing, simulation technology and digital twin are becoming more important for safety and efficiency [7,8]. The digital twin has three important components, physical laws in the real world, virtual models, and connected data that links the real world with the virtual world. There are some advantages to using digital twins [9]. We can accelerate risk assessment and production time, and predict maintenance.

Electronics applied to early cars consisted only of simple devices such as ignition and charging. However the current range of applications has increased for all operating systems, ranging from engine control to chassis control to convenience control, and new electronic control technologies are continuing to be developed. Therefore, verification is required to secure the reliability of the control function of the newly developed architecture structure. HILS (Hardware In the Loop Simulation) is an abbreviation for Hardware in the loop simulation that is used to develop and test complex real-time systems. This is closely related to digital twin technology. Many experiments in automotive technology are currently conducted in the HILS method [10,11]. The HILS verification environment consists of a control system, a plant, and a HIL simulator. A control system is a variety of controllers, such as BCM (Body Control Module), IBU (Integrated Body Unit), Cluster, DDM (Driver Door Module), etc. The plant is a physical part of the controller that consists of sensors, actuators, etc. Finally, the HIL (Hardware In the Loop) Simulator is a device that can virtually build an environment to simulate a real-world automotive environment. With HILS validation, if multiple modules form a single system, performance tests can be conducted on each module before the entire system is complete. It can reduce development time and final quality improvements can be expected [12].

Co-simulation is a method that models and simulates multiple subsystems in the environment in a distributed structure. Co-simulation demonstrates the validity of multi-domain and virtual physical systems by providing a flexible solution. It is very useful for traffic conditions, as there are a lot of elements that have different domains and mathematical characteristics [13,14]. There are many factors to consider when driving a vehicle such as other vehicles, pedestrians, and the positions of traffic lights. By using co-simulation we can simulate simultaneously with these elements [15].

With the recent rapid rise of autonomous driving technology, simulation technology has also begun to develop. There are several ways to implement a simulation similar to a real situation. It is possible to build a virtual environment based on information captured from data of the real environment [16]. By using image sequence and road Geographic Information System (GIS) data for modeling simulation, the accuracy can be improved. However, collecting image data of the road at a certain point and integrating it with GIS data can be time-consuming. Several mathematical models are used to create simulations that simulate the vehicle's movement to its surroundings. Fast, low-accuracy models are based on linear dynamics and analytical solutions to equations of motion [17]. Using a data-based vehicle dynamics model and an optimization-based mobility plan, we can calculate collision-free trajectory under normal traffic conditions [18]. With the development of the simulation, the algorithms of autonomous vehicles could also be tested more accurately. Methods have been developed for a variety of dynamic systems, such as automotive-like differentials and arbitrary linear equations of motion. Collision avoidance algorithms can be implemented for various models with nonlinear equations of motion by presenting an extension of the controlling obstacle to the general mutual collision avoidance for non-linear, non-homogeneous systems [19]. It can be written in conjunction with the sign distance concept widely used in traditional trajectory generation algorithms.

3. Proposed Method

3.1. Overall Algorithms Simulated in GTA5

The overall algorithm for the proposed method is illustrated in Figure 2. To implement a virtual layer of digital twin technology, we should process data that is captured in a vehicle. Since the data recognized by a camera in the driving situation is used to process it, image data from GTA5 must be extracted for situational identification. The functions of OpenCV and NumPy (Numerical Python) [20] are used to capture and store game images as array data in real-time. Using YOLO v3 and TensorFlow [21], we analyze the images that we captured. Subsequently, we can collect data after analysis and classify the results. After sorting the data, we used the results to control the character riding in a car from GTA5 through c-type codes outside the game.

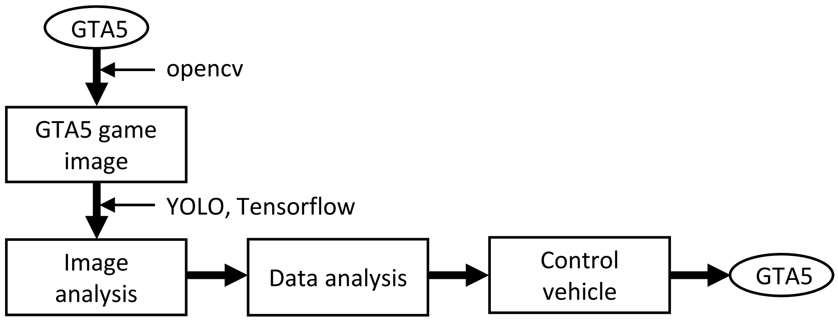

Figure 2. Overall algorithm of the autonomous driving simulation.

3.2. *Object Recognition and Avoidance Algorithms Simulated in GTA5*

Figure 3 shows the approximate algorithm for object detection and avoidance. Using images captured with OpenCV, we can detect objects by applying YOLO v4 and TensorFlow as shown in Figure 4. Algorithm 1 shows the pseudo-code of the object detection algorithm. First, we should set classes and weights provided by YOLO v3 to use them. To control vehicles by code, we made functions using keys 'W', 'D', 'A', 'S' that are applied in GTA5 in advance. As using CPU only consumes a lot of time, we can make the GPU operate TensorFlow CUDA (Compute Unified Device Architecture) by install TensorFlow for GPU (Graphics Processing Unit) [22]. We then load classes and weights that we set previously. To create input data, we could capture the game screen using OpenCV.

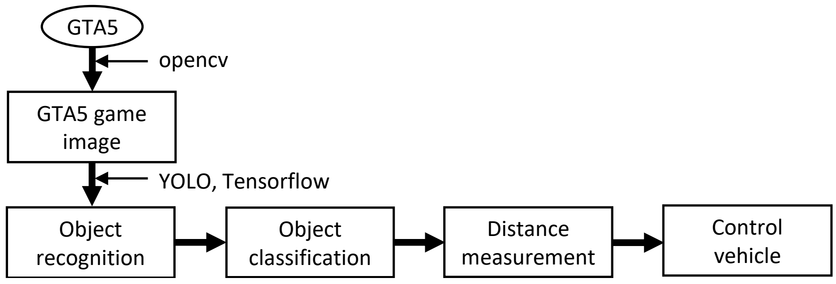

Figure 3. Object detection algorithm.

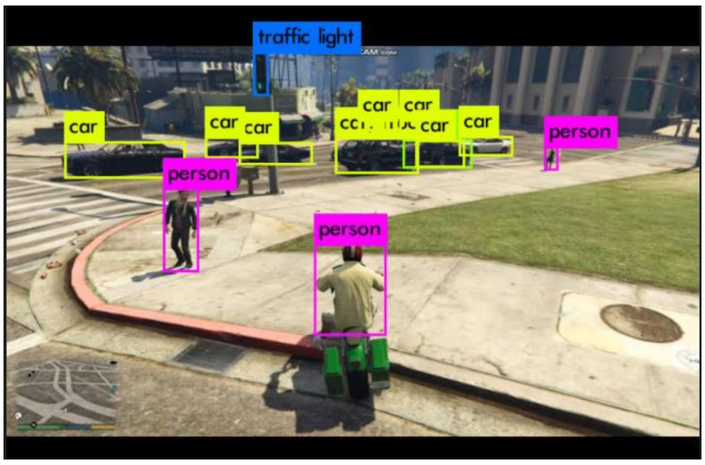

Figure 4. Object detection result.

Algorithm 1: Pseudo code of object detection algorithm.

```
1  Load classes and weights of YOLO
2  if $GO_{straight}$ then
3      PressKey(W)
4  if $GO_{right}$ then
5      PressKey(D)
6  if $GO_{left}$ then
7      PressKey(A)
8  if $GO_{back}$ then
9      PressKey(S)
10 $vid \leftarrow CapturedVideo$
11 boxes, scores, classes, nums = yolo.predict(vid)
12 foreach $i \in number\_of\_boxes$ do
13     if $classes[i] = car, bus, truck$ then
14         if $scores[i] >= 0.5$ then
15             x=(boxes[i][1]+boxes[i][3])/2
16             y=(boxes[i][0]+boxes[i][2])/2
17             distance=(1-boxes[i][3]-boxes[i][1])*4
18             if $distance <= 0.5$ then
19                 $GO\_back$
20 create a window to show result of detection
```

To make the algorithm execute in real-time, we have to reduce the needed time to detect and classify objects. Therefore, we set the resolution and size of input data as much as possible without influencing the accuracy. To analyze input data frame by frame, we can capture the video using OpenCV. As models from YOLO and TensorFlow provide confidence in objects, we can store those values in an array. By using these values, we can classify the detected objects. Objects that must be recognized while driving, such as people and vehicles, are left and categorized by using a unique array number of objects stored in the classes.

As autonomous driving requires self-controlled driving, not just obstacle recognition, box marks are placed on objects to make the process of recognition easier. To measure the distance between a vehicle and an obstacle, we measure the size of the box for the recognized object by the number of pixels that the object occupies. If the size of the box drawn on the obstacle is larger than the fixed value, the vehicle can stop until the size of the box ahead of it is reduced under a previously specified value [23].

3.3. Lane Detection Algorithms Simulated in GTA5

The algorithm for lane detection and control of the vehicle is shown in Figures 5 and 6. Algorithms 2 and 3 show the pseudo-code of the lane detection algorithm. The function 'lines' draws lines on the detected lines. From the array of 'line_data', we can extract and store data of thick lines such as slope, intercepts to an array 'line_dict'. We classified those lines extracted as one line if they have similar slopes within an error range of 20%. The final two lanes would be lines that have most similar slope detected in lines. Function 'img_process ()' deals with input data and makes it easier to process. To extract all lines in the input image, turning the image into grayscale is effective.

By using canny edge detection [24], it is possible to find most of the outlines on the screen. We can collect line data using HoughLines [25] function that is supported in OpenCV. After finding two lanes using 'lines' function created earlier, we can draw lines on the lanes for easy recognition. Figure 6 shows the thick lines that are drawn on selected lanes. To control vehicles by code, we used functions for emulating keys that are applied in GTA5. After selecting lanes, we can continue tracking their slope values. If the slope values of both lanes are positive, this means that the vehicle leans to the left. If so, we can maintain the position and direction between the lanes by steering the vehicle to the right. If the slope value is negative, we can keep the vehicle from deviating from the lane by steering it to the left.

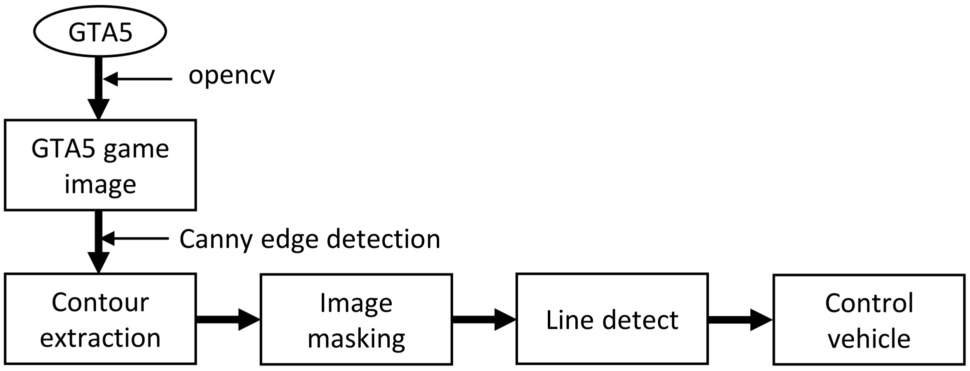

Figure 5. Lane detection algorithm.

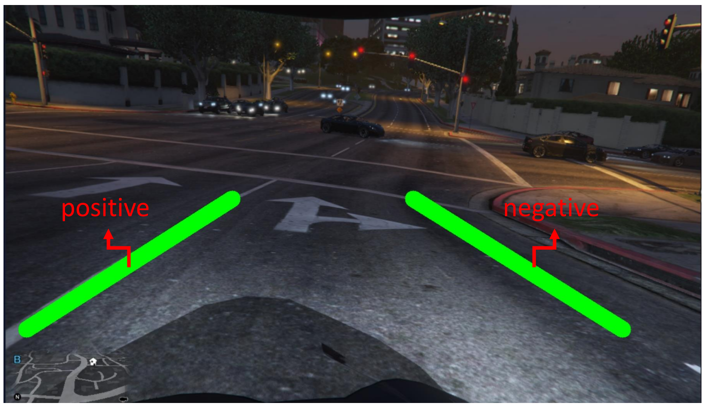

Figure 6. Lane detection result.

Algorithm 2: Pseudo code of lane detection algorithm-line detection.

```
   Input: $img, line\_data$
   Output: $lanes$
1  Function Lines($img, line\_data$):
2      foreach $i \in line\_data$ do
3          extract only the bold contours to the line
4          line_dict=[slope, coordinate]
5      foreach $i, j \in line\_dict$ do
6          if $line\_dict[i]*1.2 > line\_dict[j] > line\_dict[i]*0.8$ then
7              $final\_lane[j] \longleftarrow line\_dict[i]$
8          else
9              final_lane[i]=line_dict[i]
10     foreach $lane \in final\_lane$ do
11         two_lanes=the most similar slope detected in lanes
12         slope1=two_lanes[0]
13         slope2=two_lanes[1]
```

Algorithm 3: Pseudo code of lane detection algorithm-lane-departure prevention.

```
Input: img
Output: processed_img
1  Function img_process(img):
2      gray_img=change the image to grayscale
3      $canny\_img \longleftarrow canny(gray\_img)$
4      $processed\_img \longleftarrow GaussianBlur(canny\_img)$
5      $processed\_img \longleftarrow ROI(processed\_img)$
6      $line\_data \longleftarrow HoughLines(processed\_img)$
7      L1, L2 = lines(img, line_data)
8      draw lines
9  if $GO_{straight}$ then
10     PressKey(W)
11 if $GO_{right}$ then
12     PressKey(D)
13 if $GO_{left}$ then
14     PressKey(A)
15 while true do
16     screen = img
17     new screen = img_process(screen)
18     if slope1 <0 and slope2 < 0 then
19         $GO_{right}$
20     else if slope1 >0 and slope2 < 0 then
21         $GO_{left}$
22     else
23         $GO_{straight}$
24 create a window to show result of detection
```

3.4. Self-Driving Algorithm Implemented by DonkeyCar

To implement the physical layer of digital twin technology, the real car must be able to be programmed and controllable. Therefore we used DonkeyCar [26] as a materialize of the physical layer. DonkeyCar is a programmable car based on Python. It supports an IMX219 camera and uses a Jetson Nano board as the main board. It can be controlled by a joystick, and it provides a Unity-based self-simulation program. We can train the DonkeyCar as an autopilot with Keras, by controlling the DonkeyCar on the same road multiple times [27,28]. However, in real-world situations, self-driving cars encounter untrained road environments and obstacles, so we have to deal with image data captured with a camera. Therefore, we applied algorithms that are simulated in GTA5. We set the input image data by using an attached camera. However, commands that control the DonkeyCar's speed and direction are based on the shell, so we added some codes and compiled with a shell script to ensure smooth execution.

Algorithm 4 shows the pseudo-code of the DonkeyCar control algorithm. Figure 7 shows the structure of the DonkeyCar control algorithm based on shell script. First, we let the DonkeyCar go straight at a constant speed by using a command that controls the servo driver connected to the DonkeyCar's motor. Then, we call the python script that is programmed to detect objects. If python scripts detect an object, it returns 0, otherwise, it returns 1. Using returned values, we can control the car. If the value is 0, it can stop by changing the speed to 0; otherwise, it can move on.

To develop things better, we changed the code to count the number of people detected. Object detection code in GTA5 used quite a large range of classes and weights in YOLO. However, we narrowed the range only to detect people using TensorFlow and Tensornets. Tensornet is a TensorFlow implementation that allows us to use the specified classes and weights that we want. It is only compatible with TensorFlow v1. Algorithm 5 shows the algorithm for the people-counting program. To use Tensornet, we have to resize the image, so its resolution can be degraded. However, a slight decrease in image quality is not a serious problem in recognizing objects in real-time, and the avoidance algorithm must make DonkeyCar stop when objects are close. Thus, it is irrelevant to objects that are too small to be recognized. Whenever we detect a person, we can draw a rectangular above it and count the number of boxes. By collecting data, we can calculate the population density of the area.

Algorithm 4: Pseudo code of donkeycar control algorithm.

```
value: return value of python file
Calibrate donkey with channel
Echo "speed of DonkeyCar"
Call python detect file and execute
if value=0 then
    Calibrate donkey with channel
    echo "0"
else
    Calibrate donkey with channel
    echo "speed of DonkeyCar"
```

Algorithm 5: Pseudo code of people counting algorithm.

```
preds : object that has detected classes and coordinates data
boxed : array of boxes data
inputs: Nodes of the specified image size
model = YOLOv3COCO
classes={'0':'person'}
Launch the graph in a distributed session:
cap=videocapture
while cap is open do
    img ⟵ resized_cap_image
    preds = run.model(img)
    boxes = model.boxes(preds)
    foreach j ∈ classes do
        count=0
        if boxes != 0 then
            foreach i ∈ boxes[j] do
                box=boxes[j][i]
                if boxes[j][i][4] >= 0.4 then
                    count += 1
                    draw rectangular using box coordinates
```

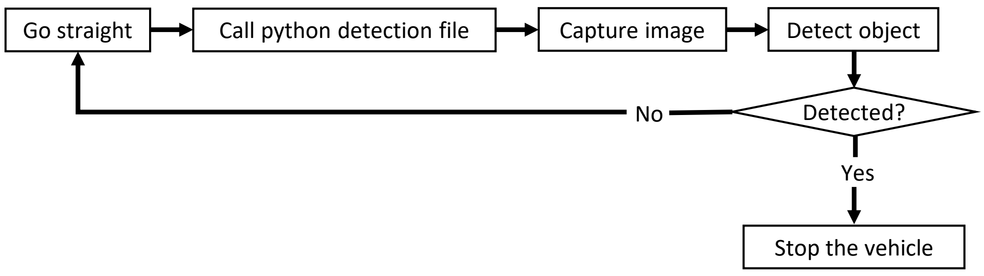

Figure 7. Donkeycar algorithm.

4. Experimental and Measurement Results

4.1. Experiment

To operate the DonkeyCar for experiments, the DonkeyCar must be assembled and connected to the main board, Jetson Nano. Figure 8a shows the assembled DonkeyCar. A voltage controller, a servo driver and a camera are connected to the Jetson Nano directly. An NiMH battery is attached with a voltage converter. Figure 8b shows a closer view of the connection of the NiMH battery and the voltage converter. Figure 9a shows a more detailed link of the Jetson Nano, the servo driver and the motor. The ground (GND) of the servo driver is associated with the GND of the Jetson Nano. The serial clock (SCL) of the servo driver is connected with the five pins, SDC1, I2C(Inter-integrated Circuit communications) bus1, of the Jetson Nano. The serial data (SDA) of the servo driver is associated with the three pins (SDA1, I2C bus1) of the Jetson Nano. The Voltage Collector (VCC) of the servo driver is attached to the 3.3 V pin (power) of the Jetson Nano.

To set up the software on the main board, Jetson Nano, we used 128 GB (GIGA Byte) microSD (Secure digital) card. We wrote Jetson Nano Developer Kit SD Card image to the microSD card that is uploaded on the Nvidia official site. Figure 9b shows the structure of Jetson Nano. We can boot Jetson Nano by inserting the microSD card containing the image and plug barrel jack for the power supply. After booting it, we can install the Python, OpenCV, TensorFlow and DonkeyCar code.

To implement a digital twin, the fusion algorithm needs to be improved in terms of robustness and applicability. In addition, parallel computing can be applied to increase computing efficiency and process large amounts of data. For better construction of cyber-physical fusion, connection and communication protocol should be standardized [29]. It should also be made predictable and reliable at the abstracted software level [30]. In order to implement the existing digital twin, the modeling of all environments had to be calculated and designed, but the efficiency was greatly increased by using it as a basis for GTA5 to which the physics engine was applied. By experimenting with validation of the algorithms using DonkeyCar, we can see how predictable and well-functioning the algorithms and simulations we designed are. Figure 10 shows the steps in the sequential method. If we use the sequential method, which is the existing method, we have to simulate it with a simulator, write the code to a real drivable car, write firmware, run it in the real environment, and then give feedback manually. Figure 11 shows a schematic diagram of a digital twin based realization. When a digital twin is used, a lot of time is reduced because the real car and the car in the virtual environment change organically through cloud-based communication. In addition, our method is reasonable to achieve digital twin because the development time is shortened, the number of interlocking codes is reduced by running the game engine, and the number of overhead iterations is small.

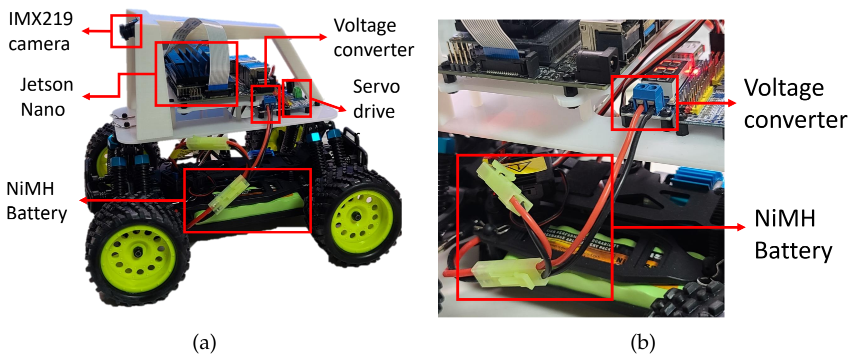

Figure 8. (**a**) DonkeyCar structure, (**b**) DonkeyCar battery connection.

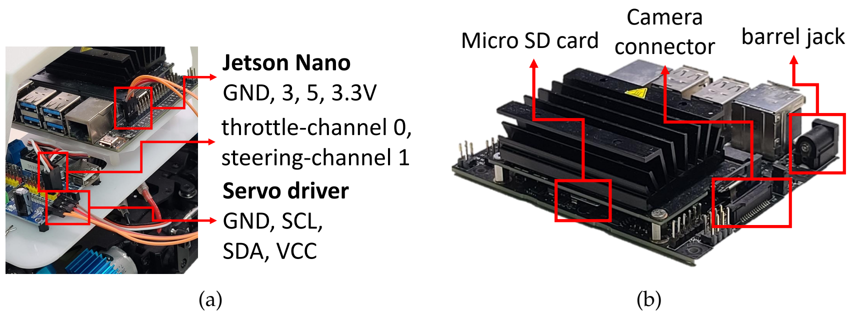

Figure 9. (**a**) DonkeyCar servo driver connection, (**b**) Jetson Nano.

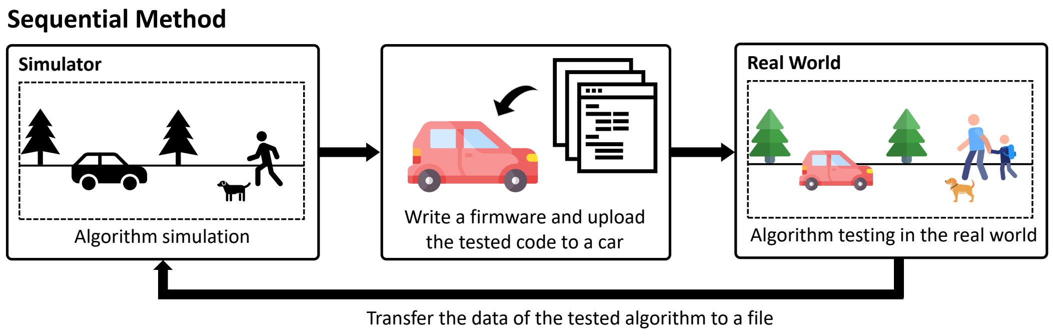
Sequential Method
Simulator
Algorithm simulation
Write a firmware and upload the tested code to a car
Real World
Algorithm testing in the real world
Transfer the data of the tested algorithm to a file

Figure 10. Schematic of sequential simulation method steps.

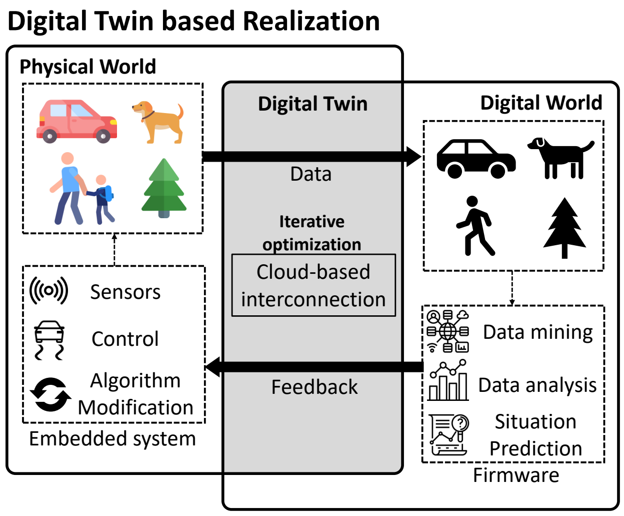

Figure 11. Schematic of digital twin based realization.

4.2. Measurement Results

Time factors also have a significant impact on digital twin technology. Figure 12a shows the execution time of the object detection, lane detection, and people counter algorithm. We added codes that log the time purely to measure the time that the algorithm is running. The total time means the entire time spent on processing, which is 0.483 s for object detection, 0.197 s for lane detection, and 0.902 s for people counter. The capture time is the time spent to capture image data from the camera and video inputs, which is 0.133 s for object detection, lane detection, and people counter. The detection time is the time taken to recognize an object from the captured image information, which is 0.417 s for object detection, 0.014 s for lane detection, and 0.692 s for people counter. The reaction time is the time spent when the car in GTA5 or the DonkeyCar reacted which is 0.002 s for object detection, 0.098 s for lane detection, and 0.078 s for people counter. We can see the car running in less than a second. In GTA5 or in real driving situations, objects or lanes do not disappear in one second. So, if the time taken is more than one second, it was considered possible to execute in real-time. Therefore, we can ensure that it can be processed in real time.

We measured memory usage during processing. 'Working set' represents the amount of physical memory currently being used by the program. 'Private bytes' indicates the amount of memory allocated to the program. This represents physical memory and swapped memory. 'Page Faults/sec' shows usage of virtual memory. Figure 12b shows the memory usage of the object detection algorithm. It shows that regardless of the number of objects that are detected, it remains the same. Figure 13a shows the memory usage of the lane detection algorithm. Figure 13b shows the memory usage of the people counter algorithm. Despite changes in the number of people recognized, memory usage is consistent. This algorithm uses about 14 MB (Mega byte) of memory, which indicates it is a very effective method. The performance comparison with previous works is summarized in Table 1. We found it effective to compare the studies of Venkatesan and Liu. First, Venkatesan's research conducted a state prediction simulation of a motor inside an automobile based on Matlab/simulink. Sensors were needed to measure the temperature and speed of the motor, and through this experiment, it was possible to predict the state of the motor and plan the execution time. However, there was a lot of computation. Liu's research combined GNSS information with Unity to simulate the reaction when changing lanes of surrounding vehicles. Two types of cameras were needed to recognize the surrounding environment and measure the distance between vehicles, and showed high accuracy and fast response time. However, when Unity and GNSS information are combined, if the GNSS information is delayed, it becomes unstable. Our research uses GTA5 as a simulator, which allows for a stable and easy simulation. You also need one camera by writing an algorithm to measure the distance in the code. GTA5 has a large capacity, but it can be overcome with the latest technology trends.

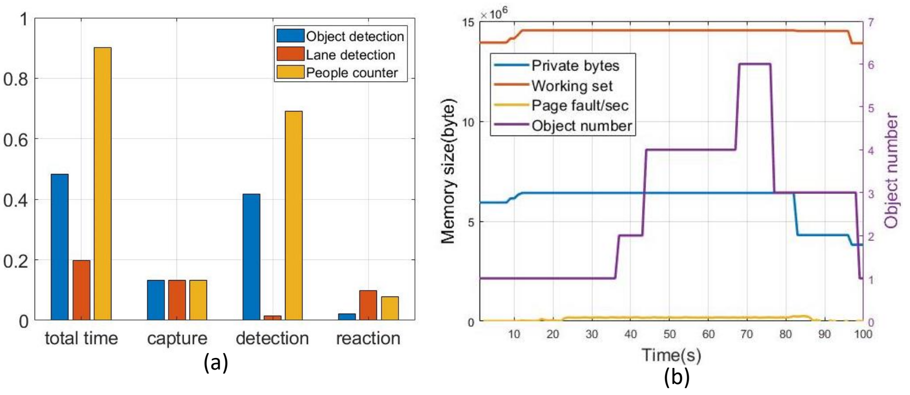

Figure 12. (**a**) Time measurement of process, (**b**) Memory usage of the object detection algorithm.

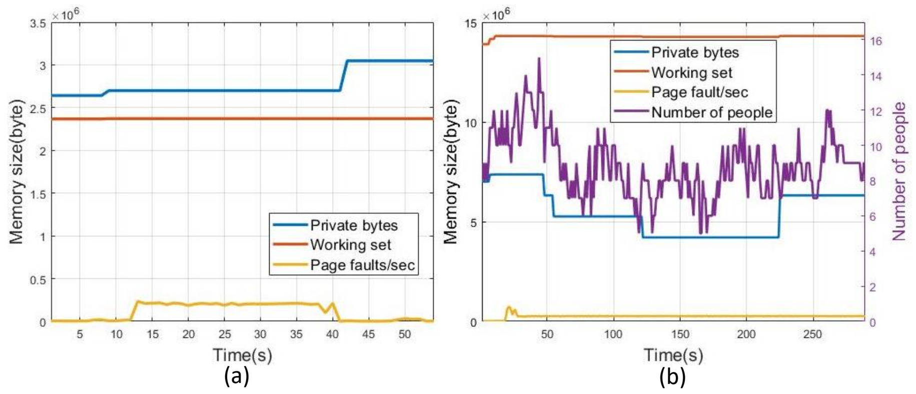

Figure 13. (**a**) Memory usage of lane detection algorithm, (**b**) Memory usage of the people count algorithm.

Table 1. Performance comparison with others.

	This Work	**Venkatesan [31]**	**Liu [32]**
Used simulator	GTA5	Matlab/Simulink	Unity combined with GNSS
Tested Algorithm	Collision avoidance, lane maintanance	Health of the motor	Predict the lane change behavior of other vehicles
Used Sensor	RGB camera	Thermometer, speedometer	RGB camera, depth camera
Advantage	Predictable response time and processing time, easy code drive	Motor condition predictable, service time-planning	High accuracy, fast response time
Weakness	GTA5's big capacity	Large amount of computation	Unstability of the simulator in case of information delay in the GNSS

5. Future Research Directions

Related works show that digital twin technology is an effective method for simulation and verification. However, depicting simulations close to the real world is difficult and burdensome because vast amounts of data must be collected, processed and communicated. Minimizing these shortcomings and burdens will be the core of future research. As mentioned earlier, there are two factors, deterministic and stochastic factors, to implement the real world in the simulation. In actual driving situations, these two factors are mixed and affect each other, therefore it is important to properly reflect them when processing data. It is necessary to develop an interface that appropriately reflects these two factors and expands them in the real world. In addition, specific evaluation criteria are needed for the development of digital twin technology. This criterion will determine the efficiency, accuracy and feasibility of the method and will be of much greater help. Furthermore, more sophisticated simulators and vehicles in the real world should be studied to communicate in real-time to detect, analyze and predict. By reflecting the predicted results in virtual reality while communicating in real-time, it will be possible to respond with a much smaller amount of data in the real world. In addition, efficiency can be increased by removing unnecessary data or traffic.

6. Conclusions

This paper proposes a method to implement digital twin technology for an autonomous vehicle. In the virtual layer, two algorithms for essential functions of autonomous driving based on GTA5 with Python were tested. For the physical layer, those algorithms conducted in the virtual layer were tested on a real car, DonkeyCar. The first algorithm is for object detection and avoidance, and the second algorithm is for lane detection. The simulation of autonomous driving is possible using GTA5 without elaborately designing the simulator. In addition, these two algorithms can be applied to various simulators that provide images, because they utilize images captured from GTA5's game screen. Furthermore, as tested on DonkeyCar, we developed code to count the number of people. These algorithms were executed in less than a second with acceptable memory usage. We can assume that they can be applied in real-time.

Author Contributions: H.Y. wrote the entire manuscript and designed core architecture and performed the software/hardware implementation; D.P. proposed main concept of the proposed architecture and designed system software architecture and also devoted his role as principle investigator and the corresponding author. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the BK21 FOUR project funded by the Ministry of Education, Korea (4199990113966, 10%), Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2019R1A2C2005099, 10%), and Ministry of Education (NRF-2018R1A6A1A03025109, 10%). This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00944, Metamorphic approach of unstructured validation/verification for analyzing binary code, 70%).

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

GTA	grand theft auto
YOLO	you only look once
OpenCV	open source computer vision
RC car	radio control car
NPC	non player character
MOD	modification
HILS	hardware in the loop simulation
BCM	body control module
IBU	integrated body unit
DDM	driver door module
HIL	hardware in the loop
GIS	geographic information system
NumPy	numerical python
CUDA	compute unified device architecture
GPU	graphics processing unit
ROI	region of interest
GND	ground
SCL	serial clock
SDA	serial data
VCC	voltage collector
I2C	inter integrated circuit communications
SD	secure digital
GB	giga byte
MB	mega byte
GNSS	global navigation satellite system

References

1. Lee, D.; Cho, J.; Park, D. Interactive simulation platform using processing-based visualization for safe collision-free autonomous driving development. In Proceedings of the 2017 IEEE Conference on Dependable and Secure Computing, Taipei, Taiwan, 7–10 August 2017; pp. 537–538. [CrossRef]
2. Paden, B.; Cap, M.; Yong, S.Z.; Yershov, D.; Frazzoli, E. A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles. *IEEE Trans. Intell. Veh.* **2016**, *1*, 33–55. [CrossRef]
3. Lee, M. An Artificial Intelligence Evaluation on FSM-Based Game NPC. *J. Korea Game Soc.* **2014**, *14*, 127–136. [CrossRef]
4. Boschert, S.; Rosen, R. Digital twin-the simulation aspect. In *Mechatronic Futures*; Springer: Cham, Switzerland, 2016; pp. 59–74.
5. Uhlemann, T.H.J.; Lehmann, C.; Steinhilper, R. The digital twin: Realizing the cyber-physical production system for industry 4.0. *Procedia Cirp* **2017**, *61*, 335–340. [CrossRef]
6. Nalam, S.; Bhargava, M.; Ringgenberg, K.; Mai, K.; Calhoun, B.H. A Technology-Agnostic Simulation Environment (TASE) for iterative custom IC design across processes. In Proceedings of the 2009 IEEE International Conference on Computer Design, Lake Tahoe, CA, USA, 4–7 October 2009; pp. 523–528. [CrossRef]
7. Rassolkin, A.; Vaimann, T.; Kallaste, A.; Kuts, V. Digital twin for propulsion drive of autonomous electric vehicle. In Proceedings of the 2019 IEEE 60th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON), Riga, Latvia, 7–9 October 2019; pp. 1–4. [CrossRef]

8. Marra, F.; Sacchetti, D.; Pedersen, A.B.; Andersen, P.B.; Træholt, C.; Larsen, E. Implementation of an electric vehicle test bed controlled by a virtual power plant for contributing to regulating power reserves. In Proceedings of the 2012 IEEE Power and Energy Society General Meeting, San Diego, CA, USA, 22–26 July 2012; pp. 1–7.
9. Gilchrist, A. *Industry 4.0: The Industrial Internet of Things*; Springer: Berlin/Heidelberg, Germany, 2016.
10. Cho, J.; Hwang, D.; Lee, K.; Jeon, J.; Park, D.; Kim, Y.; Joh, J. Design and implementation of HILS system for ABS ECU of commercial vehicles. In Proceedings of the ISIE 2001, 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No. 01TH8570), Pusan, Korea, 12–16 June 2001; Volume 2, pp. 1272–1277.
11. Lee, C.H.; Kim, S.S.; Jeong, W.H.; Lee, S.H. Development of Real Time Vehicle Dynamics Models for Intelligent Vehicle HILS. *Trans. Korean Soc. Automot. Eng.* **2006**, *14*, 199–206.
12. Zheng, S.; Tang, H.; Han, Z.; Zhang, Y. Controller design for vehicle stability enhancement. *Control Eng. Pract.* **2006**, *14*, 1413–1421. [CrossRef]
13. Dietz, S.; Hippmann, G.; Schupp, G. Interaction of vehicles and flexible tracks by co-simulation of multibody vehicle systems and finite element track models. *Veh. Syst. Dyn.* **2002**, *37*, 372–384. [CrossRef]
14. Spiryagin, M.; Simson, S.; Cole, C.; Persson, I. Co-simulation of a mechatronic system using Gensys and Simulink. *Veh. Syst. Dyn.* **2012**, *50*, 495–507. [CrossRef]
15. Kim, N.; Karbowski, D.; Rousseau, A. A modeling framework for connectivity and automation co-simulation. *SAE Int. J. Engines* **2018**, *11*, 1-010.
16. Zhang, C.; Liu, Y.; Zhao, D.; Su, Y. RoadView: A traffic scene simulator for autonomous vehicle simulation testing. In Proceedings of the 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), Qingdao, China, 8–11 October 2014; pp. 1160–1165.
17. LaValle, S.M. *Planning Algorithms*; Cambridge University Press: Cambridge, UK, 2006.
18. Best, A.; Narang, S.; Barber, D.; Manocha, D. Autonovi: Autonomous vehicle planning with dynamic maneuvers and traffic constraints. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 2629–2636.
19. Bareiss, D.; Van den Berg, J. Generalized reciprocal collision avoidance. *Int. J. Robot. Res.* **2015**, *34*, 1501–1514. [CrossRef]
20. Oliphant, T.E. *A guide to NumPy*, 2nd ed.; Continuum Press: North Charleston, SC, USA, 2015
21. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv* **2016**, arXiv:1603.04467.
22. Smilkov, D.; Thorat, N.; Assogba, Y.; Yuan, A.; Kreeger, N.; Yu, P.; Zhang, K.; Cai, S.; Nielsen, E.; Soergel, D.; et al. Tensorflow. js: Machine learning for the web and beyond. *arXiv* **2019**, arXiv:1901.05350.
23. Hong, S.; Park, D. Lightweight Collaboration of Detecting and Tracking Algorithm in Low-Power Embedded Systems for Forward Collision Warning. In Proceedings of the IEEE ICUFN 2021, Jeju Island, Korea, 17–20 August 2021.
24. Rong, W.; Li, Z.; Zhang, W.; Sun, L. An improved CANNY edge detection algorithm. In Proceedings of the 2014 IEEE International Conference on Mechatronics and Automation, Tianjin, China, 3–6 August 2014; pp. 577–582.
25. Chan, T.; Yip, R.K. Line detection algorithm. In Proceedings of the 13th International Conference on Pattern Recognition, Vienna, Austria, 25–29 August 1996; Volume 2, pp. 126–130.
26. Subedi, S. AI in Robotics: Implementing Donkey Car, IdeaFest 81. 2020. Available online: https://red.library.usd.edu/idea/81/ (accessed on 1 August 2021).
27. Zhang, Q.; Du, T.; Tian, C. Self-driving scale car trained by deep reinforcement learning. *arXiv* **2019**, arXiv:1909.03467.
28. Mahmoud, Y.; Okuyama, Y.; Fukuchi, T.; Kosuke, T.; Ando, I. Optimizing Deep-Neural-Network-Driven Autonomous Race Car Using Image Scaling. In *SHS Web of Conferences*; EDP Sciences: Aizu-Wakamatsu, Japan, 2020; Volume 77, p. 04002.
29. Tao, F.; Zhang, H.; Liu, A.; Nee, A.Y. Digital twin in industry: State-of-the-art. *IEEE Trans. Ind. Inform.* **2018**, *15*, 2405–2415. [CrossRef]
30. Lee, E.A. Cyber physical systems: Design challenges. In Proceedings of the 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), Orlando, FL, USA, 5–7 May 2008; pp. 363–369.
31. Venkatesan, S.; Manickavasagam, K.; Tengenkai, N.; Vijayalakshmi, N. Health monitoring and prognosis of electric vehicle motor using intelligent-digital twin. *IET Electr. Power Appl.* **2019**, *13*, 1328–1335. [CrossRef]
32. Liu, Y.; Wang, Z.; Han, K.; Shou, Z.; Tiwari, P.; Hansen, J.H. Sensor fusion of camera and cloud digital twin information for intelligent vehicles. In Proceedings of the 2020 IEEE Intelligent Vehicles Symposium (IV), Las Vegas, NV, USA, 19 October–13 November 2020; pp. 182–187.