# Road Creator : Efficient Road Creating Solution for Digital Twin

Kwanho Kim[1][0009−0004−3094−0746] and Kanghyun Jo[3][0000−0001−8317−6092]

University of Ulsan, Ulsan, Korea
{aarony12, jabc1240, acejo2208}@ulsan.ac.kr

**Abstract.** Recently, there has been significant development in control systems and simulators utilizing digital twins. This paper introduces methods for constructing digital twins in outdoor environments rather than indoor settings. Building a digital twin involves not only accurately mimicking the real world but also minimizing human effort to replicate the environment within a short timeframe. To achieve this, the paper proposes an algorithm for semi-automating road construction to minimize human involvement and compares the time and quality differences between manually constructing roads and using the semi-automated algorithm. In the experiments, when using proposed algorithm, building time has been reduced by two to three times. Also this paper proposes another metric. The programs that assists developers such as Road Creator proposed in this paper can be evaluated by using Facial Expression Recognition(FER). When the developer use Road Creator, the propability of the status neutral was increased about 10% than without Road Creator. The project can be found at https://github.com/Kwan-Ho-Kim/RoadDrawer.git.

**Keywords:** Digital twin · metaverse · automation · computer vision · facial expression recognition.

## 1 Introduction

Digital twin refers to a virtual space constructed to resemble the real environment. Digital twin technology finds applications in various domains such as simulation for autonomous vehicles, control systems for ships, and optimization systems for automated factories. While advancements in computing power have activated digital twin technology, mapping the external world onto virtual space requires substantial time. This paper focuses on constructing outdoor digital twin environments applicable to autonomous driving or control systems. To overcome the significant time required for manual construction, an algorithm named Road Creator significantly enhances the efficiency of this process.

An essential metric for evaluating such algorithms is measuring the time taken to create specific roads. Additionally, comparing the completed digital twin with the real environment is crucial during the construction phase. The experiment uses visuals to compare the completeness. Another advantage of

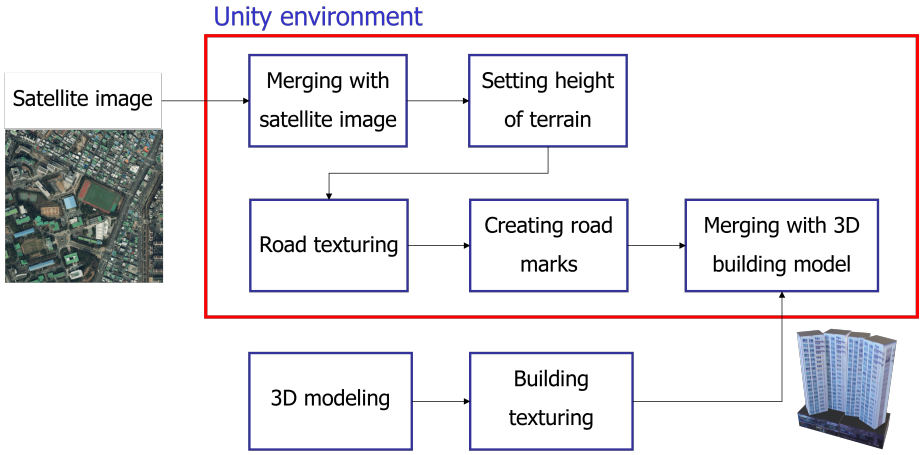


**Fig. 1.** Digital twin building process.

using Road Creator is the reduction in developer stress. Constructing digital twins through a less tedious method using Road Creator likely induces less stress compared to manual, time-consuming methods. To quantify and measure this, this paper employs Facial Expression Recognition (FER), a computer vision-based deep learning field.

The main contribution proposed in this paper lies in the Road Creator algorithm and the performance evaluation utilizing FER.

## 2 Digital twin building

The process of building digital twin environment is shown in Fig. 1. This paper uses Unity[1] to implement the environment. Fig. 1 is designed to reduce repetitive work and make work as efficient as possible. When constructing a digital twin in a different order than depicted in Fig. 1, there can be instances necessitating erasure and reworking of previously done tasks. For instance, in this paper, using Unity's basic objects, road markings were drawn on the terrain. Should the "Creating Road marks" process precede the "Setting height of terrain" process, readjusting the markings becomes necessary while altering the elevation. Additionally, progressing without drawing satellite images on the Unity terrain prohibits simultaneous work while referencing the satellite images. This section explains each process and subsequently, then the next section delves into detailing "Creating Road Marks" process which is one of the main contributions of this paper.

### 2.1 Merging with satellite image

To begin constructing the digital twin, it is necessary to merge satellite images with terrain within Unity. This process significantly streamlines the subsequent digital twin construction phase. As depicted in Fig. 1, images of the designated area for the digital twin, obtained from websites such as Naver or Google[3, 4], offering satellite imagery, are inputted into the Unity environment. The higher the resolution of the satellite images, the smoother the subsequent processes unfold. However, there exists a trade-off: as the satellite images are zoomed in to enhance resolution, a larger area cannot be captured simultaneously. Maximizing the zoom to partition the terrain results in the highest resolution but renders the satellite images indistinguishable in the final digital twin. Therefore, it's crucial to establish a resolution that allows swift identification of building and road locations, rather than blindly escalating the resolution. This paper partitioned a single terrain into four regions, each with a resolution of 2000x2000 or higher. Through Unity's terrain properties, adjustments regarding terrain size and resolution were made.

### 2.2 Setting elevation of terrain

The first step in setting the elevation of the terrain is to raise the overall height of the entire terrain significantly. This action is necessary as Unity does not allow lowering terrain objects below zero. When constructing a digital twin at an altitude lower than the reference terrain, it's crucial to raise the terrain's height to a certain extent initially to resolve the issue of diminishing elevation. Therefore, in this paper, when establishing the terrain's elevation, the terrain's height is first raised to a certain level before proceeding with the work.

In customizing the brushes for terrain elevation settings, this paper tailored them according to their intended purposes. The default brushes provided by Unity are all circular. To attach satellite images to divided regions of the terrain, square-shaped brushes are necessary. Additionally, to create terrains with slopes more conveniently, brushes with gradation are essential. Thus, this paper customize square-shaped brushes and brushes with gradation for use. In Fig. 2, (a) and (b) represent customized brushes, while (c) and (d) depict the default brushes provided by Unity. When adding texture to the terrain or drawing roads, brushes without gradation like (b) and (c) are predominantly used. Conversely, brushes with gradation like (a) and (d) are employed when setting the elevation of the terrain.

### 2.3 Road texturing

The third step in constructing terrain within Unity involves building roads. Subsection 2.4 explains the method for marking roads and lanes. However, before delineating the lanes on the roads, it is essential to distinguish and texture the roads first. This is necessary because no matter how much we zoom in on satellite photos, there are limitations to increasing the resolution, and these images

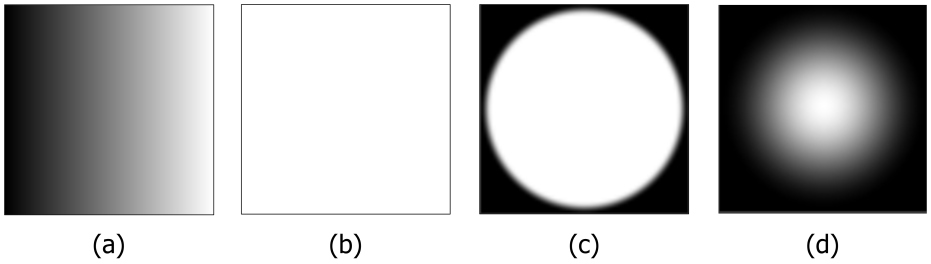


**Fig. 2.** Used brushes in Unity. (a) gradation and (b) square are customized brushes while (c) circle and (d) gradation circle are basic brushes provided by Unity.

contain unnecessary information for digital twin construction, such as cars and people. Therefore, using the brush depicted in Fig. 2-(c), this paper draw roads and add surface markings and lanes.

As mentioned earlier, since satellite photos do not appear in the final output, slight errors can be overlooked for swift operations. However, unlike the process of merging satellite photos with the terrain, the task of texturing roads will be rendered in the final digital twin and needs to be executed precisely. Hence, adjustments to rectify discrepancies are carried out during the road texturing process. The results of this Road Texturing process explained in this section can be observed in Fig. 3, in conjunction with the content of Subsection 2.4.

### 2.4 Creating road marks

The subsequent step after drawing the roads involves marking the surface indications and lanes on the roads. In Fig. 3, lane markings and surface indications are illustrated on the roads.

**Fig. 3.** The road generated by "Creating Road Markings" process.

In this paper, lane markings are implemented using Unity's 3D object "Cube". However, real-life lane markings do not cause collisions. Therefore, the default Box Collider in Unity's basic object "Cube" is removed before use. Removing the Box Collider not only prevents collisions but also conserves memory within the

Unity virtual environment, reducing computational load and optimizing resource utilization.

This process is one of the most time-consuming tasks throughout the entire procedure. While Fig. 3 depicts lane markings and surface indications on flat terrain, the workload significantly increases when marking surfaces on terrains with slopes. To minimize the manual effort of copying lane markings and attaching surface indications, this paper proposes an algorithm for semi-automating this process. A detailed explanation of the algorithm is provided in Section 3.

### 2.5 3D Modeling

This subsection details the modeling process for the buildings to be represented in the digital twin. This process comprises constructing the 3D model and applying textures to the constructed 3D model. The building modeling task was carried out using the 3D modeling software Blender[2] rather than Unity. Applying textures to the 3D model involves capturing images from street view.

Similar to previous processes, the time required for this phase varies significantly based on the desired level of completion. Additionally, most buildings exhibit a cuboid shape and are distinguishable through textures. To expedite the workflow, this paper primarily modeled buildings as cuboids and meticulously cropped textures for building modeling. However, to minimize disparities with the real world, buildings with distinctive shapes were crafted with attention to detail.

### 2.6 Merging 3D model

This subsection outlines the process of creating 3D buildings, constructing the entire terrain within Unity, and subsequently merging them. This phase represents the culmination of the entire workflow. Consequently, while merging, adjustments are made concerning the preceding processes. For instance, discrepancies discovered during the building construction phase, which may arise from inaccuracies during manual road drawing, are identified and rectified in this phase. Additionally, this stage involves placing trees, lampposts, and other structures. Without using Unity terrain, manually placing tree assets involves copying and pasting, whereas Unity terrain allows for a much easier and faster tree placement.

Although not an extensively time-consuming task, this phase is deemed crucial as it showcases the final output. Given the aim of simulating autonomous vehicles, this paper primarily focuses on texturing roads. However, digital twins can be utilized not only for simulating autonomous vehicles but also for pedestrian assistive systems [5] or promoting university campus facilities. Moreover, in simulating autonomous vehicles, detecting pedestrians on sidewalks and making decisions is pertinent. Hence, this paper implements pathways not only on roads but also on sidewalks leading to buildings.

# 3 Road markings creator

## 3.1 User Interface

In this paper, an algorithm called Road Creator is developed for facilitating road construction and pavement marking. This section explains the UI(User Interface) design, usage, and the process of algorithm development for Road Creator. The overall flowchart of Road Creator is depicted in Fig. 4.

**Basic usage** Road Creator is utilized by pre-arranging objects to be positioned as seen in the top left of Fig. 4. Each object is assigned to the inspector located in the top right of Fig. 4, and upon clicking the corresponding button in "Road-CreateWindow", Road Creator becomes active. Subsequently, by clicking on a gameobject within Unity's scene that possesses a collider, such as a terrain or plane, nodes are generated at that spot. For visualization purposes, nodes are represented as spheres of a specific radius. The white nodes can be observed in the image below Fig. 4. Clicking multiple locations using the mouse results in the creation of multiple nodes, and ultimately, pressing the 'Enter' key arranges the assigned game objects from the stored nodes, as depicted in the image below Fig. 4.

**Basic lanes** When not using Road Creator during regular operations, selecting 'Disable' in "RoadCreateWindow" results in no action. To implement lane markings and pavement indications through Road Creator, choosing a button other than 'Disable' is necessary. For 'White,' 'Yellow,' and 'Others', objects assigned respectively to 'White lane', 'Yellow lane' and 'Others' in the inspector are utilized. Given that the most frequent use while constructing roads involves marking lanes, 'White lane' and 'Yellow lane' are specifically named as depicted in the image. However, users can freely use different objects for 'White lane' and 'Yellow lane' according to their intent. Considering a total space to save and use objects regardless of their names might simplify the understanding to accommodate three objects. As their functions are identical among the three objects, this paper refers to them collectively as basic lanes. Subsequent subsubsections elaborate on 'Segments' and 'Modify', two functions serving different purposes. 'Others' primarily serves for indicating left/right turns, crosswalks, curbs, and similar pavement markings. Each turn indication denotes the direction from the first node to the last node, as depicted in the left image of Fig. 5.

**Segments** In Fig. 3, the yellow lane is generated by the 'Yellow lane'. In traffic systems, yellow lanes are used to indicate traffic flow in the opposite direction, while white lanes denote traffic flow in the same direction. On the other hand, segmented lanes signify the possibility of lane changes, which, if created through 'White lanes', 'Yellow lanes', or 'Others', would require considerable effort. Hence, this paper creates segmented lanes through the 'Segments' option, as illustrated in Fig. 5. Unlike basic lanes, segmented lanes require parameters

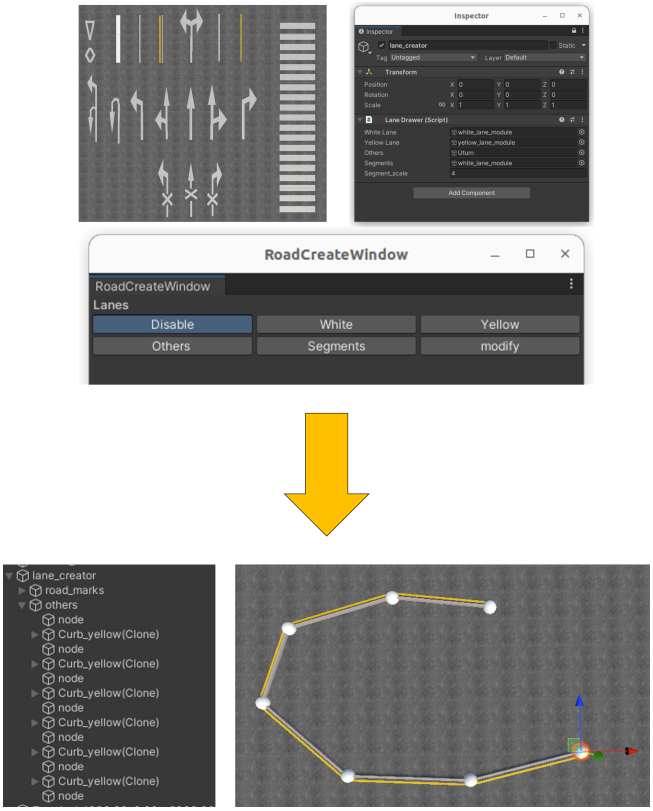


**Fig. 4.** Workflow of Road Creator

determining the sizes of these divided segments. This task is performed in the inspector. The 'Segment_scale' field in the inspector allows setting the size for a single segment.

Similar to basic lanes, 'Segments' can be used not only for the white lane but also for yellow lanes and all types of pavement markings. A thicker form akin to a pause sign can be displayed by reducing the 'Segment_scale', indicating areas where U-turns are permissible. Yellow lanes, distinguishing directions while enabling turns, can be utilized for areas allowing turns.

**Modify** The final interactive feature for users with Road Creator is 'Modify'. Even when users create lane markings and pavement indications through clicks, it's not always possible to create nodes in perfect positions. Additionally, nodes created through clicks cannot hover in the air for reasons explained in subsection 3.2. While it's unlikely to place nodes in mid-air when marking roads and pavement, if necessary according to the user's needs, they can be adjusted using the 'Modify' option. Thus, the 'Modify' option is used to adjust the node positions,

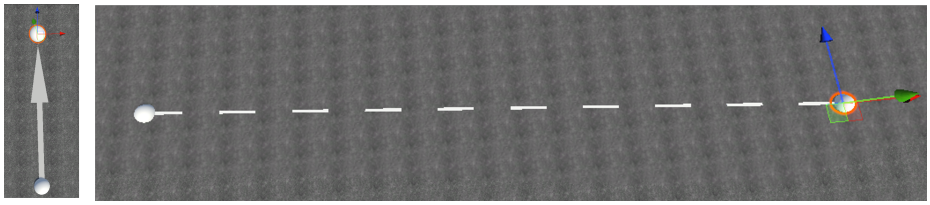

**Fig. 5.** Direction of lanes

correcting lane positions or directions for lane markings and pavement indications. Selecting 'Modify' in "RoadCreateWindow" and clicking on a node allows for modifications in the position and direction of lane markings and pavement indications. When modifying basic lanes, calculating the distance and direction between two nodes and adjusting the size suffices. However, modifying segments differs from basic lanes. If the distance between nodes widens, new segments must be created, and if it narrows, existing segments should be deleted.

The reason for magnifying nodes in Fig. 5 is to ease modifications and display where the user selected nodes during their creation. Therefore, nodes are visualized prominently. However, nodes should not appear in the final output. Therefore, nodes are only visible during 'Modify', and when "RoadCreateWindow" is set to 'Disable', all nodes remain hidden. The only instance where nodes appear without 'Modify' selected is during their initial creation. At this time, all nodes are determined using Unity tags. If a node's tag changes after creation, it will not deactivate even in 'Disable'. This approach allows convenient modification and node creation in this paper while avoiding the display of unnecessary objects in the digital twin.

### 3.2 Implementation

Algorithm 1 represents the operations performed by the Road Creator. Firstly, it initializes the variable 'node_list' that will store nodes. Next, 'OnSceneGUI' is an internal function in Unity that repeatedly executes when editing the entire screen, as long as the mouse is over the game scene. Therefore, the Road Creator algorithm continuously iterates while constructing the environment in Unity. If "RoadCreateWindow" is selected as 'Disable', an algorithm to clear all nodes is executed. Failing to clear the nodes when 'Disable' is selected would result in the nodes created upon clicking remaining in memory, thus not allowing the lanes to be created as intended.

After preprocessing, the algorithm proceeds to create nodes and generate lanes in earnest. It detects mouse clicks, and if "RoadCreateWindow" is selected as one of the basic lanes, it creates nodes. Nodes are generated at the point where a ray cast from the mouse first collides. The generated nodes are added to the 'node_list' created in the first line. This process continues with every user click, and all created nodes are stored in the 'node_list'

**Algorithm 1** Road Creator

```
1: Initialize node_list
2: while OnSceneGUI do
3:     if Disable then
4:         Clear node_list
5:     end if
6:     if click mouse & Basic lane  then
7:         check point
8:         create node
9:         add node to node_list
10:    end if
11:    if Press enter key then
12:        if basic lanes then
13:            for node list do
14:                calculate lanes from each nodes
15:            end for
16:        end if
17:        if segments then
18:            for node list do
19:                calculate segments from each nodes
20:            end for
21:        end if
22:    end if
23:    if Modify then
24:        activate all nodes
25:        check selected node
26:        check index of selected node
27:        if node is from segments then
28:            calculate segments from next and previous nodes
29:        end if
30:        if node is from basic lane then
31:            calculate lanes from next and previous nodes
32:        end if
33:    end if
34: end while
```

From line 11 onwards signifies the process where nodes create lanes upon the user pressing 'Enter'. If one of the basic lanes is chosen in "RoadCreateWindow", it iterates through 'node_list' to compute the lanes. It calculates the midpoint between two nodes and adjusts their size. Conversely, if "RoadCreateWindow" is set to 'segments', it iterates through 'node_list' to calculate lengths, determining the number of segments and creating lanes. Calculations for Basic lanes are explained in subsubsection 'Calculating basic lane', and those for Segments are detailed in subsubsection 'Calculating segments'.

If "RoadCreateWindow" is set to 'Modify', it first activates all nodes to enable their modification. Upon selecting a node in the scene, it identifies the index and performs the algorithm by finding the preceding and succeeding nodes. Similar

to creating basic lanes and segments upon hitting the 'Enter' key, it dynamically modifies nodes and lanes in real-time.

As mentioned earlier, the method of calculating lanes varies depending on what is selected in "RoadCreateWindow". The following two subsubsections describe how Basic lanes and 'Segments' are calculated, respectively.

**calculating basic lane** When creating a lane from two nodes, position the lane at the midpoint of the two nodes, ensuring that both nodes lie at the ends of the lane. This process is described through Equations (1), (2) and (3). Here, $P_1$ denotes the first node, $P_2$ denotes the second node, $P_l$ represents the lane's center position, and $\overrightarrow{d}$ signifies the length of the lane. Essentially, through Equation (1), the vector $\overrightarrow{d}$ directed from the first node, calculated, determines the position and size of the lane created between the two nodes.

$$\overrightarrow{d} = P_2 - P_1 \tag{1}$$

$$P_l = P_1 + \frac{\overrightarrow{d}}{2} \tag{2}$$

$$d_l = ||\overrightarrow{d}|| \tag{3}$$

**calculating segment** When creating a lane from two nodes using Equations (1), (2), and (3), the process was straightforward. However, when generating basic lanes, there is only one lane, whereas in the case of segments, the number of segments varies based on the length between nodes. Additionally, a field called 'segment_scale', signifying the length of one segment, determines the number of segments to be created. For instance, if 'segment_scale' is 3 and the magnitude of vector $\overrightarrow{d}$ is 10, two segments are generated. If 'segment_scale' is 2 and the magnitude of vector $\overrightarrow{d}$ is 10, there should be five segments. Equation (4) defines $k$ as the number of segments, $s$ as the length of one segment, i.e., the 'segment_scale' value. Equation (5) calculates the position for each segment. $P_s^i$ denotes the center position of the $i_{th}$ segment, and it iterates through the $k$ segments, placing each segment based on Equation (5). Fig. 6 illustrates the model for calculating segments.

$$k = round(\frac{||\overrightarrow{d}||}{2s}) \tag{4}$$

$$P_s^i = P_1 + \frac{s}{2} + 2(i-1)\frac{\overrightarrow{d}}{||\overrightarrow{d}||} \tag{5}$$

## 4 Experiments

### 4.1 Environment

The experiments were conducted to evaluate the performance of Road Creator. A developer assessed how efficiently roads are generated across multiple environments. The experiments involved creating flat roads, slope roads, curved flat

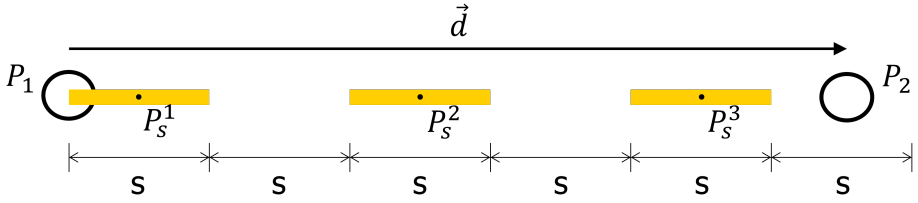


**Fig. 6.** Segments calculation model. By calculating the number of segments with round function, it is available to get expected number of segments. If the result of $\frac{||\vec{d}||}{2s}$ calculation is 2.5, resulting in the creation of two segments, only $P_s^1$ and $P_s^2$ will be generated in the figure. Conversely, if the calculation yields 2.4, leading to the creation of three segments, the $P_s^3$ segment extends beyond $P_2$

roads, curved slope roads, and rotaries. Fig. 7 illustrates samples of these environments. This paper draws each sample in advance and compare the performance when Road Creator is not used and when Road Creator is used. Conducting experiments without creating samples would likely yield higher performance in subsequent tests. For instance, if the experiment was initially conducted without using Road Creator for the Rotary environment and then measured its performance without using Road Creator, it would lead to biased results. This is because the developer already has experience building the same environment and knows the order and contents to be constructed. By creating samples, as shown in Fig. 7, and conducting experiments, regardless of the order, developers become familiar with the details and sequence of environments. The experiments were carried out using Unity version 2021.3.19f1.

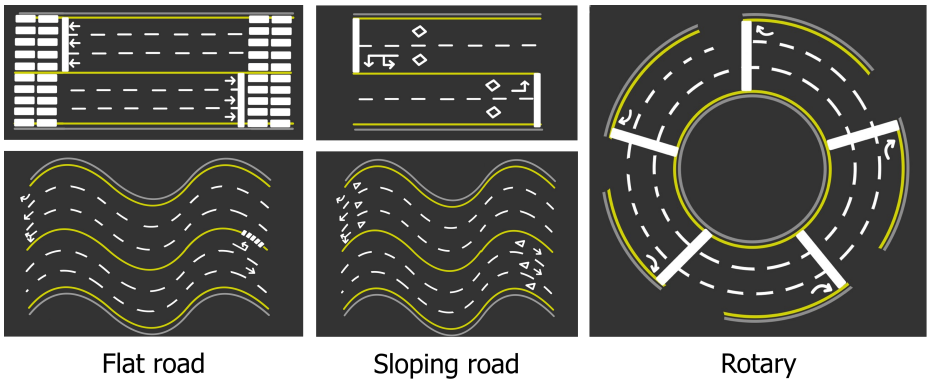


**Fig. 7.** Environment samples.

### 4.2 Evaluation metric

**Building time** The primary objective of Road Creator is to enable developers to efficiently construct roads when building a digital twin. In other words, the goal proposed in this paper for Road Creator is to create desired roads in less time compared to not using Road Creator. Therefore, the building time explained in this section is the most critical metric for measuring Road Creator's performance. All experiments start with only elevation generated, without anything else. In this paper, a user interface (UI) is created to measure the time from when the start button is pressed to when the developer completes the construction by pressing the stop button, thereby measuring the time taken to build the roads.

**Table 1.** Facial Expression Occupancy Experiment resuts: 7 expressions percentage

| **Sample** | **Method** | **Facial Expression result(%)** | | | | | | | **time (s)** |
|---|---|---|---|---|---|---|---|---|---|
| | | *angry* | *disgust* | *fear* | *happy* | *sad* | *surprise* | *neutral* | |
| Flat | Original | 18.20 | 0.00 | 1.32 | 0.00 | 2.00 | 0.70 | 77.00 | 470 |
| | Road Creator | 7.60 | 0.00 | 1.10 | 0.00 | 0.37 | 0.90 | 86.50 | 211 |
| Slope | Original | 30.30 | 2.26 | 0.96 | 3.66 | 6.85 | 2.31 | 53.57 | 887 |
| | Road Creator | 18.70 | 1.76 | 1.32 | 0.22 | 8.38 | 4.42 | 65.12 | 189 |
| Rotary | Original | 0.60 | 0.00 | 0.03 | 0.00 | 4.76 | 0.26 | 94.33 | 1107 |
| | Road Creator | 0.36 | 0.00 | 0.00 | 0.00 | 3.47 | 1.15 | 95.00 | 571 |
| Curved Flat | Original | 21.23 | 0.10 | 0.32 | 0.26 | 4.65 | 0.34 | 73.09 | 2051 |
| | Road Creator | 13.02 | 0.04 | 3.95 | 0.00 | 10.08 | 9.69 | 63.19 | 880 |
| Curved Slope | Original | 1.20 | 0.95 | 0.03 | 3.02 | 4.10 | 0.06 | 90.61 | 1376 |
| | Road Creator | 28.66 | 0.12 | 0.56 | 0.04 | 43.15 | 0.00 | 27.36 | 1019 |

**Facial Expression Recognition** Another benefit to expect when using Road Creator is reducing developer fatigue. To verify how much less fatigued a developer becomes when utilizing Road Creator, this paper employ a field actively used in computer vision, namely FER (Facial Expression Recognition). FER falls within the domain of deep learning and is used for classifying human expressions. This paper utilize the user-friendly state-of-the-art model, the Residual Masking Network [6]. This model classifies human expressions into a total of 7 classes: angry, disgust, fear, happy, sad, surprise, and neutral.

Although the accuracy of FER in deep learning models is currently around 70%, the results of expression recognition are probabilistic. During the construction of a digital twin, by taking averages for each class across numerous frames, one can observe the trend of expression changes between using Road Creator and manually constructing roads by the developer. Using a moving average filter during an experiment, average probabilities for each class are computed. Fig. 8 visualize the trend of expression changes in graphs to depict the changes in facial expressions during the experiments.

A single experiment can sometimes take more than 30 minutes to conduct. Due to potential memory issues when calculating averages for all frames and

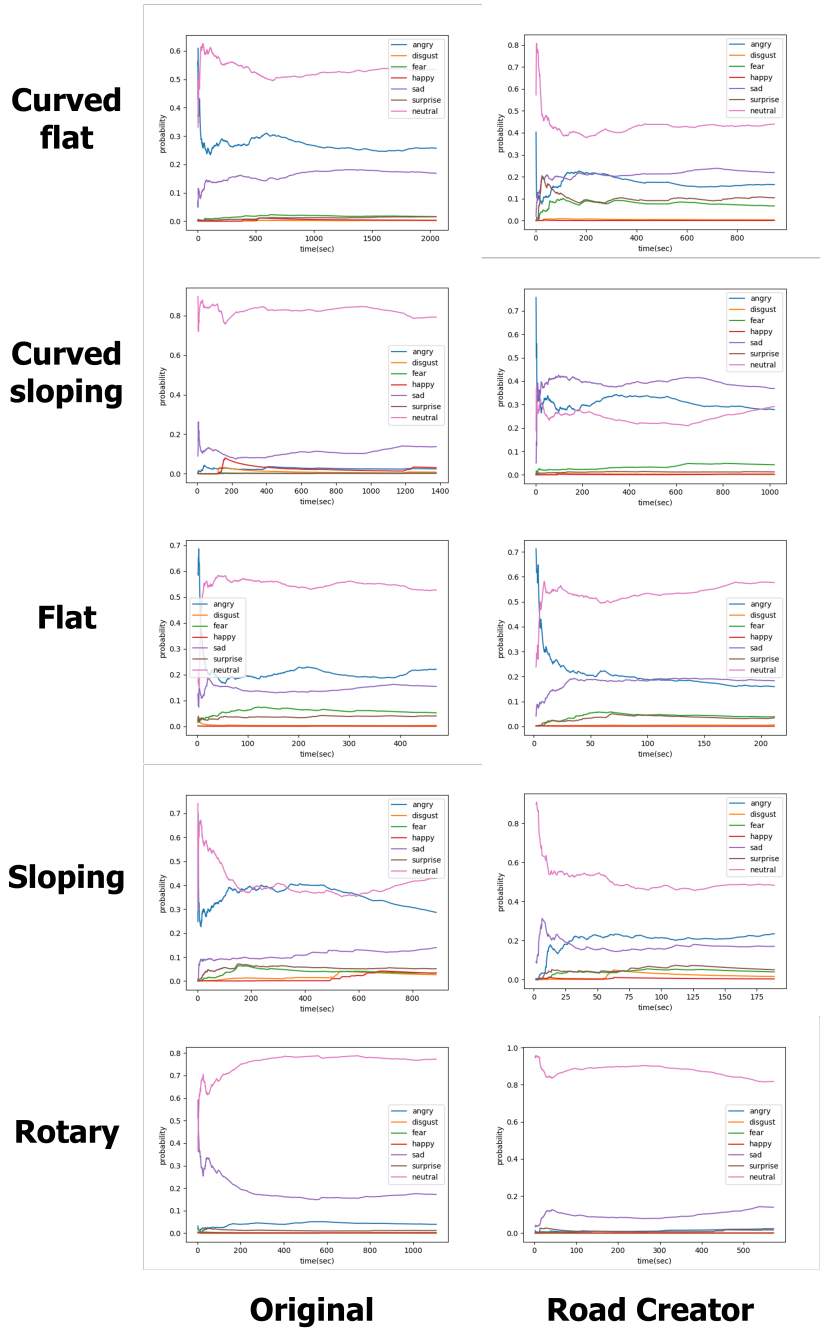


**Fig. 8.** FER graphs. Each graphs represent the change during creating road markings with Road Creator or without Road Creator.

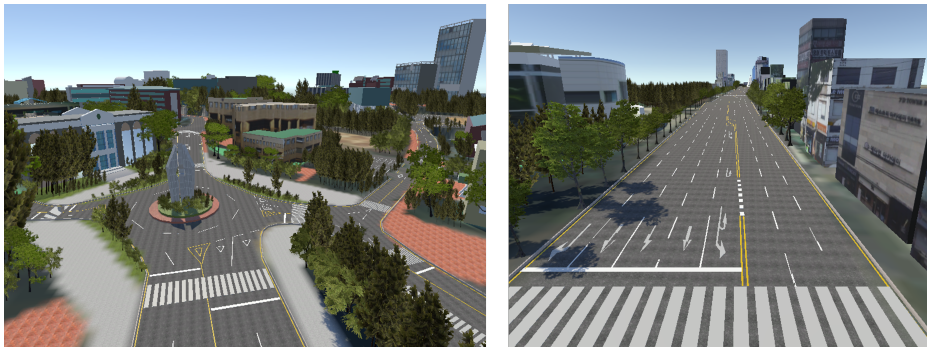

**Fig. 9.** Digital twin building results.

plotting graphs for every frame, this paper sample only 2 frames per second to compute probabilities for each expression.

### 4.3 Results

The experimental results are listed in Table 1. Experiments were conducted on a total of 5 samples mentioned in Fig. 7. The numbers highlighted in red indicate the building time when using Road Creator. For all samples, the experiments completed using Road Creator were approximately 2 to 3 times faster. Experiments regarding FER took place over a span of 2 days. Consequently, there might be significant differences depending on the developer's mood right before conducting the experiment. While this could imply potential inaccuracies in the experimental results, it also signifies that FER can serve as a performance metric for developer assistance systems like Road Creator. Examining the building time for constructing Curved Slope roads in Table 1 reveals little difference when using Road Creator compared to not using it, unlike when constructing other roads. This is due to the developer being significantly fatigued when using Road Creator for Curved Slope roads. As evidenced by the FER metric, the probability for the angry class is 28.66%, for the sad class is 43.15%, and for the neutral class is 27.36%, indicating a notably lower probability for the neutral class compared to other experiments. However, in other experiments conducted with relatively similar psychological states, the probability for the neutral class concerning Road Creator appeared higher.

## 5 Discussion

This paper proposes Road Creator which is an efficient tool to build digital twin. Without Road Creator, developer had to build digital twin by copy-paste each road markings such as lane, curb, turn direction, etc. However, Road Creator make developers can only create some nodes and press Enter to create desired lanes. Experiments show that using Road Creator can decrease building time

drastically. Also this paper proposes novel method to evaluate the performance of Road Creator. FER is a type of research field in computer vision. By using FER algorithms, the performance of programs that assist developers or office workers can be evaluated as well as Road Creator proposed in this paper.

Fig. 9 shows the constructed digital twin environment. All roads were created using the Road Creator. In this paper, the resulting digital twin can also be utilized for building autonomous driving datasets. For instance, rather than manually constructing digital twins, they can be generated through deep learning networks. This necessitates the availability of datasets constructed through digital twins. The digital twins proposed in this paper could serve as datasets for constructing generative networks for future digital twin generation. Furthermore, it can be employed in automating the construction of datasets essential for autonomous driving. Traditionally, during deep learning training, datasets have largely been built targeting the real world. However, by constructing digital twins within virtual environments, an extensive range of scenarios can be created, aiding in preparing datasets that can correspond to real-world situations.

**Acknowledgments.** This result was supported by "Regional Innovation Strategy (RIS)" through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(MOE)(2021RIS-003)

## References

1. Unity Real-Time Development Platform, https://unity.com/kr, last accessed 2023/12/16
2. Home of the Blender Project, https://www.blender.org/, last accessed 2023/12/16
3. Naver Map, https://map.naver.com/p?c=15.75,0,0,1,dh, last accessed 2023/12/16
4. Google Map, https://www.google.com/maps, last accessed 2023/12/16
5. Hyeonkwon Son, Kwanho Kim, Kanghyun Jo: Digital Twin-based Pedestrian Route Guidance System, KSIC 2024
6. Pham, Luan and Vu, The Huynh and Tran, Tuan Anh: Facial expression recognition using residual masking network, ICPR 2020, pp. 4513–4519, IEEE(2021)