Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 5543720, 21 pages
https://doi.org/10.1155/2021/5543720

Research Article

Multimedia Concepts on Object Detection and Recognition with F1 Car Simulation Using Convolutional Layers

Amutha Balakrishnan,[1] Kadiyala Ramana,[2] Gaurav Dhiman,[3] Gokul Ashok,[1] Vidhyacharan Bhaskar,[4] Ashutosh Sharma,[5] Gurjot Singh Gaba,[6] Mehedi Masud,[7] and Jehad F. Al-Amri[8]

[1]*School of Computing, SRM University, Chennai, India*
[2]*Department of Artificial Intelligence & Data Science, Annamacharya Institute of Technology and Sciences, Rajampet, Andhra Pradesh, India*
[3]*Department of Computer Science, Government Bikram College of Commerce, Patiala, India*
[4]*Department of Electrical and Computer Engineering, San Francisco State University, San Francisco, CA 94132, USA*
[5]*Institute of Computer Technology and Information Security, Southern Federal University, Russia*
[6]*School of Computer Science, Mohammed VI Polytechnic University, Ben Guerir 43150, Morocco*
[7]*Department of Computer Science, College of Computers and Information Technology, Taif University, P. O. Box 11099, Taif 21944, Saudi Arabia*
[8]*Department of Information Technology, College of Computers and Information Technology, Taif University, P. O. Box 11099, Taif 21944, Saudi Arabia*

Correspondence should be addressed to Mehedi Masud; mmasud@tu.edu.sa

Received 8 February 2021; Revised 7 May 2021; Accepted 10 November 2021; Published 9 December 2021

Academic Editor: Dafeng Hong

This paper presents a framework for detecting objects in images based on global features and contours. The first step is a shape matching algorithm that uses the background subtraction process. Object detection is accomplished by an examination of the oversegmentation of the image, where the space of the potential boundary of the object is examined to identify boundaries that have a direct resemblance to the prototype of the object type to be detected. Our analysis method removes edges using bilinear interpolation and reestablishes color sensors as lines and retracts background lines from the previous frame. Object contours are generated with clustered lines. The objects detected will then be recognized using the extraction technique. Here, we analyze the color and shape characteristics with which each object is capable of managing occlusion and interference. As an extension of object detection and recognition, F1 car simulation is experimented with simulation using various layers, such as layer drops, convolutionary layers, and boundary elimination, avoiding obstacles in different pathways.

1. Introduction

Object detection is the primary functionality needed by most computer and robot vision systems. The new study in this field has advanced significantly in many respects. Computer vision technology has rapidly and effectively developed. These results have been achieved with computer vision methods and with the development of new representations and models for specific computer vision issues. In machine vision, we have seen great progress. It includes the details of object detection science. To detect objects, the system tracks objects in a scene and organises them into categories. The purpose of an object detector is to detect objects regardless of size, location, position, view with respect to the camera, partial occlusion, and illumination conditions.

Object detection is the first task of many computer vision systems to learn more about the object. Once a face has been identified, more details can be gathered. This

includes (i) recognizing the face in a photograph, (ii) tracking it as it moves through an image sequence, and (iii) collecting more information on the face (i.e., the gender of the person). Global features usually are some characteristics of regions in images such as area (size), perimeter, Fourier descriptors, and moments. Global features can be obtained either for a region by considering all points within a region, or only for those points on the boundary of a region [1].

Contours can be explained simply as a curve joining all the continuous points (along with the boundary), having the same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

Object detection has been used in many applications, including (i) phone call from human computers, (ii) robotic systems, (iii) smartphones, (iv) protection systems, (v) image and video analysis and (vi) search engines, (vii) picture management, and (viii) transportation (search) (e.g., autonomous and assisted driving). Each of these applications has different specifications. This means processing time, robustness of occlusions, rotational invariance and pose shift detection. A number of applications are not assuming detection of just one class (for example faces), and some other applications are not assuming the detection of just one class from multiple viewpoints (e.g., side and frontal view of vehicles). Most systems can typically detect one object class from a restricted view set [2].

Finding meaningful regions in images has been a research topic in computer vision for many years. Researchers studied patterns at different scales, including local features like patterns on corners or intersections and long-range patterns over objects. Regions serve as the fundamental building blocks for a variety of vision applications [3–6]. For example, due to its robustness against image changes, a local region detector is critical for image matching and object recognition.

1.1. Local Features. Regions with high local prominence have made a major contribution to object detection and image recovery. Their methods provide robustness to image formation and distortion. If the objects are extracted and/or at several scales, the data will be good for recognizing algorithms [7–9]. Shape are mainly derived from high-profile texture/strength patterns, but are often fixed as triangles, ellipses, or rectangles. Eliminating the complexity of identification reduces the problem. Due to the lack of shape, the extracted features lose their representative power and generate noisy features [10]. We would like to identify types of parts and objects, so that a full-anthropic identification can be made. Multiple segmentation tries to find new segments using different criteria or by integrating neighbouring areas. Unfortunately, larger populations of voxels often produce noisy segments, and the current methods lack an accurate model of the object. Instead, bottom-up methods use past experiences with categories to create their own correct category by using fragment pieces of knowledge. We usually extract regions at both object and local level without requiring category-specific information. New regions are obtained at the local and object level, then combined in a data-driven way to make a new image-based region. In order to scale up exemplary recognition, we use the AWTV process, which uses the hierarchical structure of regions across different spatial boundaries.

2. Literature Survey

2.1. Local Region Detection. Local features are important for detecting and recognizing patterns in local regions. The extraction function combines a detection step that specifies the locations of the images to be extracted (positions, scales, shapes) and a description stage that creates a local descriptor [11]. Boundary Protection Local Area Detector and those to be tested with regional detection use various descriptors to demonstrate their applicability. Therefore, the focus of this section is type description and correspondence. Local region identification is a long-term topic of computer vision research and is important for broad baseline stereo matching. The invariant properties under image variations provide repeatability of certain instances of objects. On the other hand, generic object categories are often too sparse to capture rich visual information. Densely sampled patches provide a high level of coverage and provide reliable measurements for far higher costs [12]. Recent research shows how one sampling technique combines advantages and balances coverage with repeatability. However, unlike the various prior methods that do not know the object boundaries and shapes, our approach does not cross objects with object and context input. Algorithms that identify places based on observable characteristics are starting to use segments as the basic unit of analysis. Segments capture the structure of the object and provide broader spatial coverage. The instability of segmentation algorithms is due to the fact that they are unstable or sensitive to parameter settings. Segments can break into texture blots depending on the lighting, environment, or other background clutter [13]. Current work is focused on how reliable predictors can be selected using existing data. Many applications are randomly subsegmented to increase the likely of finding the actual object. The present work of the paper is to propose a new method, which uses new features, to solve the segmentation problem. The proposed method is shown to be effective, robust, and state-of-the-art. The proposed method supports local features such as corners, square, ellipse, rectangle, pyramid, and line. Our method does not rely on detecting those features as required in existing works on the segmentation problem. Instead, the proposed method is not based on detecting these local objects, but combines different solutions to achieve a cohesive solution for the segmentation problem.

2.2. Object Segmentation with Shape Priors. Using low-level image features such as color and texture, segmentation algorithms are often found to be unsuccessful when objects are present in similar-looking groups or are of a different color or have differing texture. Researchers are working on a top-down approach that uses model formats to segment the object. Top-down techniques merge the segmentation models with a bottom-up colors or textures to mark the type of the object [14]. Form-based objects work by detecting

objects in cluttered images. Cognitive architectures that use category-specific, top-down knowledge coupled with bottom-up evidence. In particular, category-specific priors make explicit assumptions about the observer's segmented point of view, for example, vehicle side views. We think about it this way: the key insight is that objects share their features, and this sharing lets one pass the features of a known class to an unknown class. We propose an exemplary transfer function for global forms [15]. Our approach ignores prior knowledge of the objects present in the scene, focusing instead on a shape descriptor that is able to accurately represent the class that the object is most likely of. The model-based design also eliminates the concerns of shapes that are difficult to pose. Mutual visual properties of groups of objects has been studied in several ways. Multiclass learning helps to employ the value of traditional item discrimination features in object detection and transfer information on newly learned items. From a signal-processing point of view, properties of the inputs are expressed in primitives, e.g., in units of symmetry [16]. Recently, efforts are directed at establishing foundational characteristics of these curves. The process, however, goes beyond local sharing to consider global style object-level predictions. Our effective nonparametric approach to sharing in 3D space brings greater flexibility to changes in object shapes and poses. There are a number of recently developed methods that work for our approach [17–19]. They also accept access to a database of segmented images, create category-independent object segmentation hypotheses and do the same thing we do. Previous methods focus on local bottom-up processing [20–22]. Multiple segments of the figure ground will provide multiple hypotheses for object regions. To reduce inconsistency, there is some work that aims to map multiple hypotheses to a single segmentation that provides a nonconflicting definition of the object. An ongoing effort is made to combine multiple segmentation [23]. They integrate an expert knowledge, a large vocabulary, and create a model from that by selecting pieces of phrases. Section 3 demonstrates the System Framework. Section 4 discusses the methods and materials. Section 5 presents the conclusions. Section 6 includes simulations for the F1 car using the system studied.

3. System Framework

3.1. Image Preprocessing. To further process, the relevant and usable image data is enhanced and distortions/noise introduced into the image is reduced in this step from low-level images. Here, in this work fault detection, feature extraction, and segmentation, image transformation is used typically. Intensity change and filtering an image are the main focus areas here as shown in Figure 1.

3.2. Filtering. As a preprocessing tool, noise reduction in image segmentation plays an important role.

Various methods are being developed, and almost all of them rely on the same basic method for doing this: the average. Noise is theoretically composed of other distinct pixels that are apparently apparent, and according to this information, there are adjacent pixels. Noise can be removed using the average of the actual image data in similar areas. In fact, the actual image data can share similarities in these average areas, but there is no noise in these areas. [24]. As a result, this filtering method effectively maintains and removes noise without damaging the actual image data. It is easy to define the average, but it is not easy to determine which pixel to average. Averaging multiple pixels can result in loss of image detail. Conversely, there is no average noise reduction effect with too few pixels. The media filter replaces each pixel in the image with the median of the pixels that surround the image and uses a mask of strange length to sort the pixels in the window by output intensity.

3.3. Intensity Adjustment. Improvements in images can be objective (filtered) or subjective [25]. Intensity adjustment is a technique used to emphasize an image. The goal is to emphasize the image by adjusting the value of Intensity to a new area. An important way to transform the grayscale intensity is to use the g = imadjust (f, low in; high in, low out, high out, gamma) syntax [26]. This function maps values in image f to new values in g,for example, map low in and high in map values to low out and high out values, low map values to low out, and high map values to highout [27]. All fully functional inputs are listed from 0 to 1 (double or 0 to 255 (unit 8), and gamma parameters determine the shape of the curve that maps the output intensity values from the inputs; so, gamma is less than 1 The smaller mapping is pushed higher (brighter). In the output value, the mapping is pushed lower (darker), and the map is linear [28] if the gamma is equal to 1.

3.4. Interpolation. Interpolation is the method of estimating a continuous sample function. Interpolation Image processing includes image enlargement or reduction, subpixel image register, spatial distortion correction, and image decompression. Image data is typically collected at a lower rate to reduce the noise in the image. Mapping between an unknown high-resolution image and a low-resolution image is not invertible; so, reverse mapping is not invertible [29]. The highest quality performance is needed when sampling with high-resolution cameras.

3.5. Bilinear Interpolation. The grey level in bilinear interpolation is calculated from a weighted average of four pixels closest to the input coordinate and assigned to the output coordinate [30]. Two linear interpolations are rendered first (horizontally); then, another linear interpolation is performed (perpendicular to the first) (in this paper). For one-dimensional linear interpolation, two grid points are required. For bilinear interpolation, the number of grid points required to test the interpolation function is four (linear interpolation in two dimensions). The interpolation kernel for linear interpolation is $u(s) = \{0|s| > 1$ Equation (3) and $1|s||s| < 1$, where s is the distance of the interpolated point from the grid point. Interpolation coefficients are as follows: ck = f (xk).

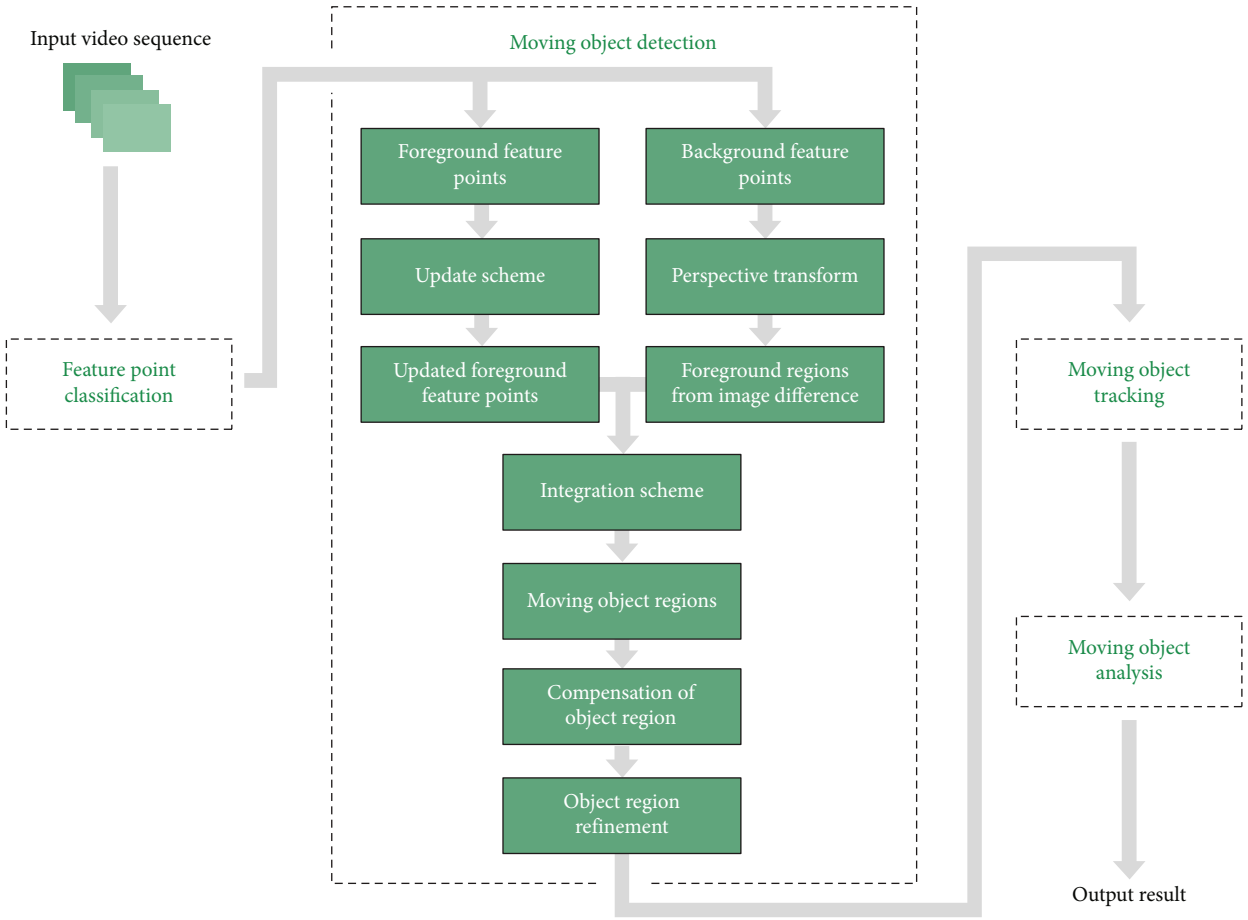

Figure 1: Object recognition system.

4. Methods and Materials

4.1. Simulation. Various segmentation methods have been proposed over the past few decades, and some classification is needed to properly present the method. Two very different approaches to segmentation may share properties that are contrary to a particular classification, but the classification seems impossible. Therefore, the classification described in this chapter emphasizes an approach rather than a strict division.

The subsequent categories are used:

(i) Threshold-based segmentation: to segment the image, histogram thresholding and slicing techniques are used. They can be combined with pre- and postprocessing techniques that can be applied directly to images

(ii) Edge0based segmentation: in this method, the edges detected in the image are used to identify the boundary of the object

(iii) Region-based segmentation: area-based technology takes the opposite approach when boundary-based technology finds the boundaries of an object and then completes and searches for the object. The boundary of an object

(iv) Clustering techniques: it is often used as a synonym for (aggregate) segmentation, but is used here to refer to techniques used in the primary analysis of high-order, measurable tendencies of search data. In this context, clustering techniques try to group similar patterns in some sense. This is similar to what you are trying to achieve when segmenting an image, and in fact, you can use a specific clustering method to segment your image

(v) Matching: if you know that the object you want to define (approximately) in the image looks like a value by a characteristic such as its perimeter length, you can use this information to find the object in the image. This splitting method is called matching datasheet values

4.2. Canny Edge Detection Algorithm. A lot of people know that the Canny edge detection algorithm is the optimal edge detector. Canny's aim was to update the many edge detectors when he started working. He achieved his target with great success, and his principles and strategies can be found in his paper, A Computational Approach to Edge Detection. The first is the lowest error rate. It is important that the edges of the image are not overlooked, and that the nonedges are not addressed. The second criterion concerns

the direction of the edge points. In other words, the distance from the detector between the edge pixels and the actual edge should be at a minimum. The third criterion is that there should be only one answer to one edge (Table 1).

The Canny edge detector smoothes out the image and noise based on these parameters. The gradient of the image is then found to highlight regions with high spatial derivatives. The algorithm follows these regions and removes any non-maximum pixels. The gradient array is now further reduced by hysteresis. Hysteresis has two threshold values and is set to zero if the magnitude is below the first threshold. If the magnitude reaches the high threshold, the edge shall be drawn. And if it is between the two thresholds, it is set to zero unless the path to a pixel with a gradient above T2 is known.

4.2.1. Step 1. The Canny edge detector requires a number of steps to implement. It is helpful to remove the noise from the original image before locating and detecting boundaries. Which makes it easy to find the Gaussian filter's ideal frequency. Convolution can be achieved by standard means when the appropriate mask has been calculated. A mask is usually much smaller than the real image. It slides over the image, adding an extra square. The lower the sensitivity, the larger the Gaussian mask. A misaligned region increases marginally as the average size increases.

4.2.2. Step 2. The next step is to find the intensity of the edge by taking the gradient of the image, lightening the image and eliminating the noise. The Canny operator performs a 2D spatial gradient image measurement. The estimated absolute gradient magnitude (edge force) can then be calculated at each point. The Sobel operator uses a variety of 3x3 convolution masks, one estimating the x direction gradient (columns) and the other estimating the y direction gradient (rows) (Table 2).

The gradient's magnitude or edge intensity is approximated by the formulation: $|G| = |\mathrm{Gx}| + |\mathrm{Gy}|$.

4.2.3. Step 3. The position of the edge is defined by the gradient x and y. However if the sum X is equal to zero, an error occurs. There must also be a restriction in the code if this occurs. Whenever the gradient in the direction of x is 0, the direction of the edge must be 90° or 0°, depending on the value of the gradient in the direction of y. If GY has a zero value, the direction of the edge is 0 degrees. If not, the direction of the edge is 90 degrees. The formula for finding the direction of the edge is correct:

$$\mathrm{Theta} = \mathrm{invtan}(\mathrm{Gy}/\mathrm{Gx}). \quad (1)$$

4.2.4. Step 4. When the direction of the edge is known, the next step is to link the direction of the edge to the direction of the image. Thus, the pixels of the 5×5 image are aligned (Table 3).

Now, we can see the pixel "a" when identifying the surrounding pixels, and there are only four possible directions-0 degrees, 45 degrees, 90 degrees, and 135 degrees. The orientation of the edge in one of the four directions must therefore be resolved on the basis of the direction nearest to zero degrees shown in Figure 2.

Table 1

+1	+2	+1
0	0	0
-1	-2	-1
Gy		

Table 2

-1	0	+1
-2	0	+2
-1	0	+1
Gx		

Table 3

x	x	x	x	x
x	x	x	x	x
x	x	a	x	x
x	x	x	x	x
x	x	x	x	x

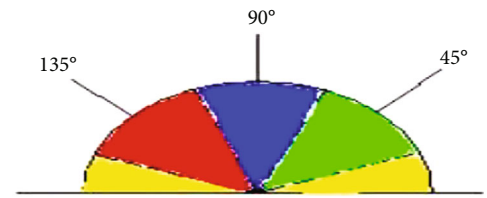

Figure 2: Edge detection based on angular coloring.

Any position of the edge within the yellow range is thus set to 0 degrees (0 to 22.5 and 157.5 to 180 degrees). Any direction of the edge falling within the green range (22.5 to 67.5°) shall be set at 45°C. The route at the edge of the blue (67.5 to 112.5°) is set at 90°. At the end, each path on the edge that falls into the red range (112.5 to 157.5°C) is set at 135°C.

4.2.5. Step 5. No full deletion must now be enforced after the edge directions have been known. Nonmaximal suppression is used to map the edge in the direction of the edge and to delete a pixel value set to 0 that is not called the edge. This gives a thin line to the output image.

4.2.6. Step 6. Hysteresis is used to clear a streak. In streaming, the edge is caused by an output that fluctuates above and below the threshold. A threshold will be applied to the image, and the edge will fall below the threshold due to noise. It is above the level, so that the rim appears dotted. Hysteresis uses two thresholds to prevent this from occurring. Any pixel with a value greater than 1 is to be an edge pixel and be labelled as such. Any pixels whose value is greater than T2 will also be selected as edge pixels. To follow a border, you need to start with a T2 gradient, but you will

not stop until you hit a T1 gradient.

$$G \text{ function with Gaussian } \sigma = 1.4. \tag{2}$$

Hence, the simple Gaussian is considered in a discrete grid a particular case of the $G(x, y)$ for narrow Gaussians.

4.3. Adjacent Weighted Threshold Variation (AWTV) Method. The goal is to minimize the error associated with classifying a background pixel as a front pixel or vice versa, and the threshold is determined by weight. This is accomplished by minimizing the area beneath the histogram for a region on the other side of the threshold. The issue is that each region lacks an AWTV; only combined regions have a weighted threshold value [31]. Understand that there is typically no minimum overlap (where the distribution's misclassified areas are equal) when the valley occurs in the combined pixel $(k-1, k)$, for example, when one cluster has a large distribution and the other a narrow distribution. One way to do this is to consider the values of both regions as two clusters; they are weightless or weightless. In other words, the mean of all pixels below the threshold may be μB (T), and all pixels above the threshold may be equal to μO (T). We want to set the threshold as follows:

$$\begin{aligned} &\forall g < T : |\mathrm{g} - \mu B(T)| < |g - \mu O(T)|, \\ &\forall g \geq T : |g - \mu B(T)| > |g - \mu O(T)|. \end{aligned} \tag{3}$$

This is a variation of AWTV's clustering pattern recognition algorithm. The basic idea is to measure the mean $\mu B\,(T)$ for pixels at four corners (assumed to be the background) and the other one to measure $\mu O\,(T)$. Set the threshold between $\mu B\,(T)$ and $\mu O\,(T)$ (so that the pixels are split according to the proximity between $\mu B\,(T)$ and $\mu O\,(T)$). Update the $\mu B\,(T)$ and $\mu O\,(T)$ estimates by actually measuring the pixel mean at both of the current thresholds. This method continues until the algorithm converges.

4.4. Clustering (The AWTV Method). Another approach that produces comparable results is to minimize duplication by clustering each cluster as closely as possible. Of course, we cannot alter the distributions themselves, but we can alter the distances between them (thresholds). When the threshold is changed in one direction, it enlarges the diffusion on one side and reduces the diffusion on the other side [32]. The goal is to choose a threshold to minimize the combined distribution. Internal distribution may be defined as the sum of the distribution weights for each cluster:

$$\begin{aligned} \sigma^2_{\text{within}}(T) &= n_B(T)\sigma^2_B(T) + n_O(T)\sigma^2_O(T), \\ n_B(T) &= \sum_{i=0}^{T-1} p(i), \\ n_{\mathrm{O}}(\mathrm{T}) &= \sum_{i=0}^{N-1} p(i), \end{aligned}$$

$\sigma^2_{\mathrm{B}}(\mathrm{T})$ = The variance of the pixels in the background (below threshold),

$\sigma^2_{\mathrm{O}}(\mathrm{T})$ = The variance of the pixels in the background(above threshold). (4)

$[0, N-1]$ is the range of intensity levels. For both groups, the difference in this class of potential each threshold requires a lot of computation, but it is easier to do so. Subtracting the variance within a class from the total variance of the combined distribution gives us the variance of

$$\sigma^2_{\text{Between}}(T) = \sigma^2 - \sigma^2_{\text{within}}(T) = n_B(T)[\mu_B(T) - \mu]^2 + n_O(T)[\mu_O(T) - \mu]^2, \tag{5}$$

where $\sigma 2$ is the sum of the variances, and μ is the sum of the means. Take note that the variance between classes is simply the weighted variance of the cluster means in relation to the overall mean. We obtain by substituting $\mu = nB(T)\mu B(T) + nO(T)\mu O(T)$ and simplifying.

$$\sigma^2_{\text{Between}}(T) = n_B(T)n_O(T)[\mu_B(T) - \mu_O(T)]^2. \tag{6}$$

So, for every potential threshold T, it is as follows:

(a) Splits the pixels into two AWTVs according to a weighted threshold

(b) Calculate the mean of each cluster

(c) Square the difference between the means

(d) Multiply the number of pixels in one cluster by the number of pixels in the other cluster

$$\begin{aligned} n_B(T+1) &= n_B(T) + n_T, \\ n_O(T+1) &= n_O(T) - n_T, \\ \mu_{\mathrm{B}}(T+1) &= \frac{\mu_{\mathrm{B}}(T)n_B(T) + n_T T}{n_B(T+1)}, \\ \mu_{\mathrm{O}}(T+1) &= \frac{\mu_{\mathrm{O}}(T)n_O(T) - n_T T}{n_O(T+1)}. \end{aligned} \tag{7}$$

I have noticed that the optimal threshold is to maximize the variance between classes (or minimize the variance between classes), but the calculations when switching to the threshold are not independent. As T increases, the average $\mu B\,(T)$ and $\mu O\,(T)$ with $nB\,(T)$, $nO\,(T)$ of each cluster can be updated as pixels move from one group to another. I can do it.

4.5. Simulation Results. A variation of the AWTV-based object recognition in any particular vehicle image is to train the features with the image, then use it to recognize the particular vehicle of the image by different structure element. This should provide more accurate recognition, as it corrects for different image features that is mean, perimeter, etc. Figure 3 shows the resulted images that are implemented using MATLAB. Following Figure 3 is the first image which is the input to this program that was designed using MATLAB graphical user interface (GUI) (Table 3).

The following Figure 4 shows background subtraction module resultant image, Figure 5 shows Canny edge detection image, and Figure 6 shows binary image conversion;

Figure 3: Car designed using MATLAB graphical user interface.

Figure 4: Background substraction module resultant image.

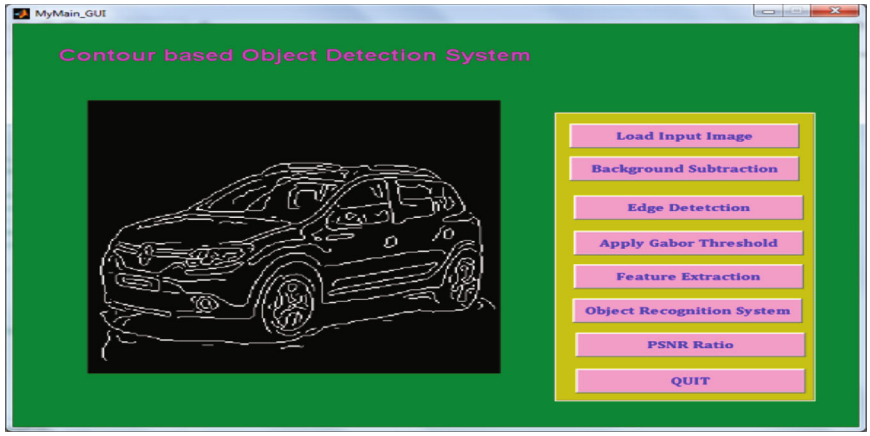

Figure 5: Canny edge detection image.

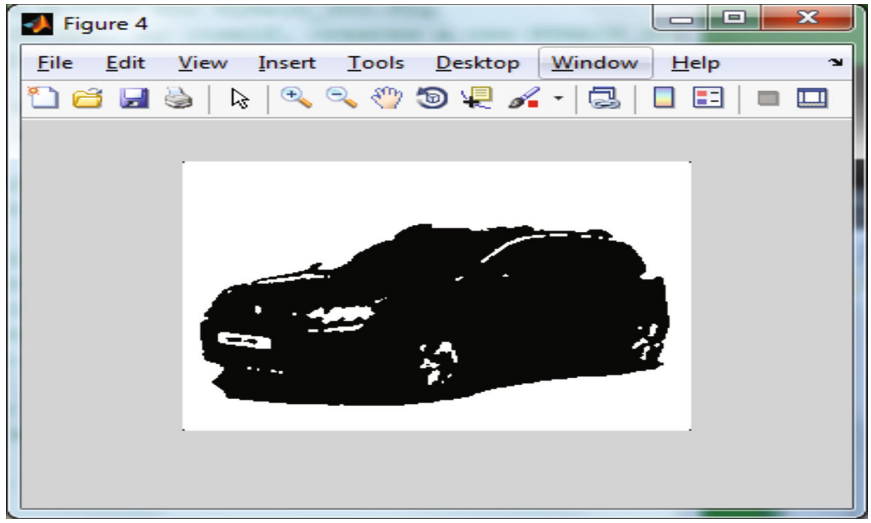

Figure 6: Binary image conversion.

then, Figures 7 and 8 represent object recognition result is shown in message box.

In Figure 9, features are shown in command window.

5. Inferences

We presented a new contour-based model for object detection. A 3D model is created by using the AWTV Method and its geometric relations. Objects are found in the test images using the most suitable object shapes. Partial occlusion is an object detection problem that is not seen in the data. The model offers geometric information for each pair of shapes; so, we expect it to handle partial occlusion. At least some samples are hardly affected by object instances in the sample dataset. The SHAPE model can handle large orientation changes, since relative rather than absolute geometric attributes are used. We can test the model with proper direction adjustments. All internal contour fragments inside the boundary are modeled, and those on the boundary are modeled as well. We will use unconnected fragments to identify patterns. Both SHAPE and relationship parameters are manually set in all experiments. For various objects, such as apples and swans, their shape and parameters may not fulfill this requirement. Our future goal is to build a machine that can determine these images automatically. We can also establish richer relationship types between type and form such as shape orientation.

6. Need for a Racing Simulator

This section deals with developing a software application or a simulator, that essentially simulates an environment for the user. The simulator developed here will be a Formula One Simulator that will enable users to experience a real-world action-packed racing environment in which the users will be able to get into a Formula One car and drive around a track or take part in several races against tough computer opponents. These races are held in some of the world renowned race tracks of the world. For example, we have reconstructed the Autodromo Nazionale Monza Race Track which is located in the city of Monza, North of Milan, Italy. In any typical racing scenario, a driver cannot simply jump into a vehicle and race around. It requires immense amounts of training and dedication [32]. The same applies for a Formula One driver. These drivers belong to an elite class of world-class drivers competing for the coveted title of "Best Driver In The World.'" The Formula One drivers are essentially tremendously fit athletes and hard trained to endure the many forces and stresses applicable to this form of motorsport [33]. First, reference images of an dimensionally accurate F1 car are set and arranged as shown in Figure 10:

Then, a primitive rectangular polygon is added superimposing the reference images on all of the different views as shown in Figure 11:

Then, this rectangle mesh is further sculpted by using modifiers (such as extrude, chamfer) and by moving, rotating, or scaling the vertices, edges, and faces of the mesh to bring about a rough outline of the body of the vehicle as shown in Figures 12 and 13.

Then, this rough outline is further sculpted to bring about a relatively low polygon model of a F1 car body. Generally, low polygon models are used in games to meet the resource constraint of the game [34–36]. The rough edges of low polygon models are covered by using normal mapping, a type of texturing which gives the illusion of depth as shown in Figure 14.

The other objects in the game are modeled similarly to Autodesk 3ds Max, formerly a 3D Studio Max, is a 3D graphics computer programme for 3D animations, models, and photographs. It was developed and produced by Autodesk Media and Entertainment. It has models, a modular plugin architecture, and can be used on Microsoft's Windows platform. It is also used by video game makers, several TV studios, and architectural visualisation studios. It is also used for film effects and film previews. The latest version of 3ds also includes shaders (e.g., ambient and subsurface dispersal), dynamic simulation, particle systems, radiosity, normal mapping and

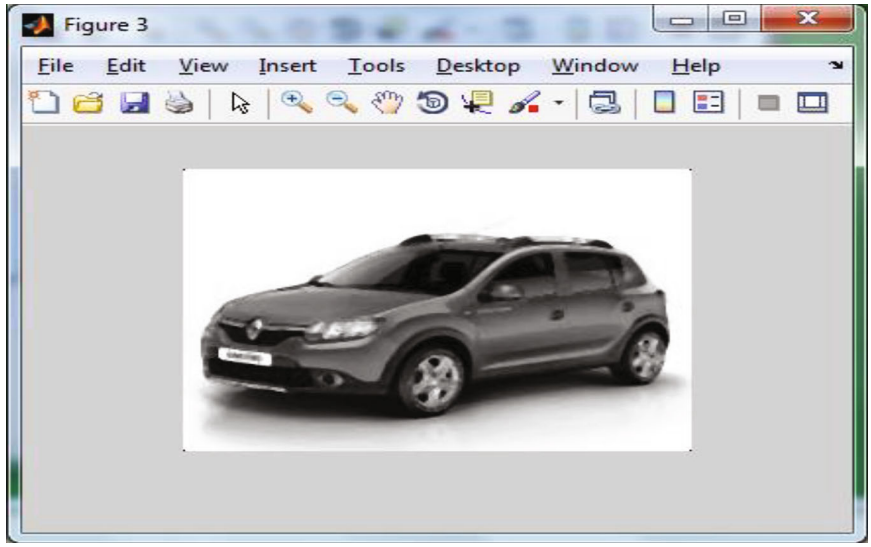

Figure 7: Object recognition result.

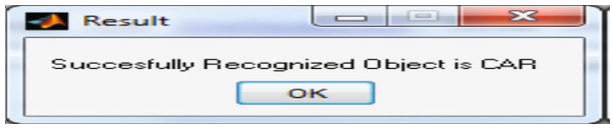

Figure 8: Object recognition result message box.

rendering, global lightset, flexible user interface, and its own scripting language, along with its modeling and animation tools. Some of the academy award winning movies designed in 3ds Max are Avatar, The Curious Case of Benjamin Button, 2012, Shutter Island, The Mummy, etc. Some of the highest grossing games designed in 3ds Max are Mass Effect, Unreal Tournament, Need For Speed, Call of Duty, etc.

Once these models are sculpted, then they are imported to the Unity3D game engine. The Unity3D game engine is a crossplatform with built in IDE. It is used to develop video games for web plugins, desktop platforms, consoles, and mobile devices. It was initially used to design games for the iOS but as of now, it can be used for multiple platforms. It is a free game engine which has a variety of powerful tools to effectively build a game. The levels in Unity are called scenes with each scene having multiple objects put together along with shader effects, lighting, scripts, cameras, activators, etc. The level building part deals with the placement of different objects with the help of the transformation, rotational, and scale modifiers. The development of the Autodromo Nazionale Monza track is illustrated below. Initially, a new empty scene is created. Then, the reference image of the Monza circuit is set as shown in Figure 15.

With the reference image set, the track is then placed superimposed on the reference image. The track consists of smaller pieces of roads that are first placed on the scene and then transformed, rotated ,or scaled to fit the track in the reference image. The pieces of the track must be exactly aligned with one another. Otherwise, collision errors might be seen later when dynamic objects are added to the scene. The track is progressively built as seen in the next two illustrations represented in Figures 16 and 17.

This piece is duplicated, and the duplicated piece is placed alongside the original. If there is a corner, then the appropriate piece is used. This continues until the track is complete. Minor adjustments have to be made at the end to close of the circuit to get the completed track as shown in Figure 18.

Once the track is completed, then a terrain is added to the scene. On the terrain, other props are added (like crowd stands) to populate the scene as shown in Figure 19.

Once that is done, then the terrain tools are used to generate a landscape. The terrain tools can be used to increase or decrease particular areas of the terrain. Then, the terrain tools are used again to generate vegetation for the terrain and add the bounds of the scene as represented in Figure 20.

Lastly ,the visual effects are added to the scene. A directional light is added to light the scene. Then, a skybox is added to add depth to the scene. Advanced effects like fog, ambient occlusion, and antialiasing are adjusted to the finest detail. Then, the scene is finally baked before any of the dynamic game objects are added. The final scene is illustrated below as shown in Figure 21.

The modules related to the F1 car is illustrated below:

6.1. End-User Car Handling. This module basically deals with controlling of the car by the end-user and also assigns a trailing viewing camera relative to the position of the car in the game world (generally behind car model). This enables the user to switch the camera views from first-person viewpoint to third-person viewpoint. First-person viewpoint refers to the graphical perspective that is rendered from the viewpoint of the player, which is the cockpit of the vehicle in this case. Third-person viewpoint refers to the graphical perspective that is rendered from a fixed distance behind and slightly above the player. This viewpoint allows

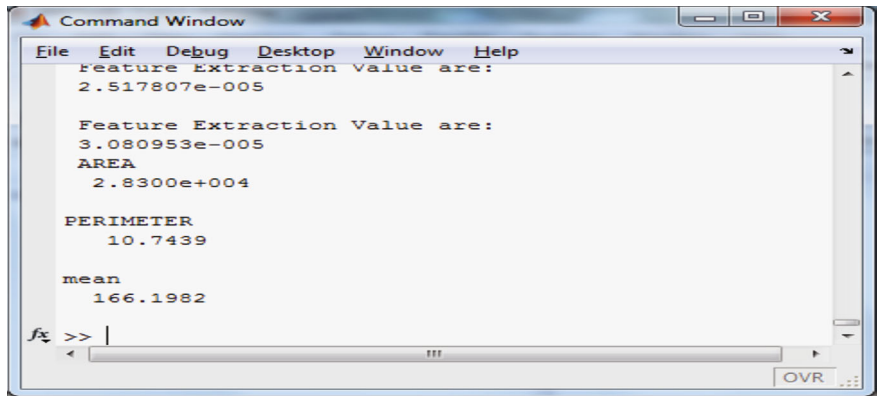

Figure 9: Image features.

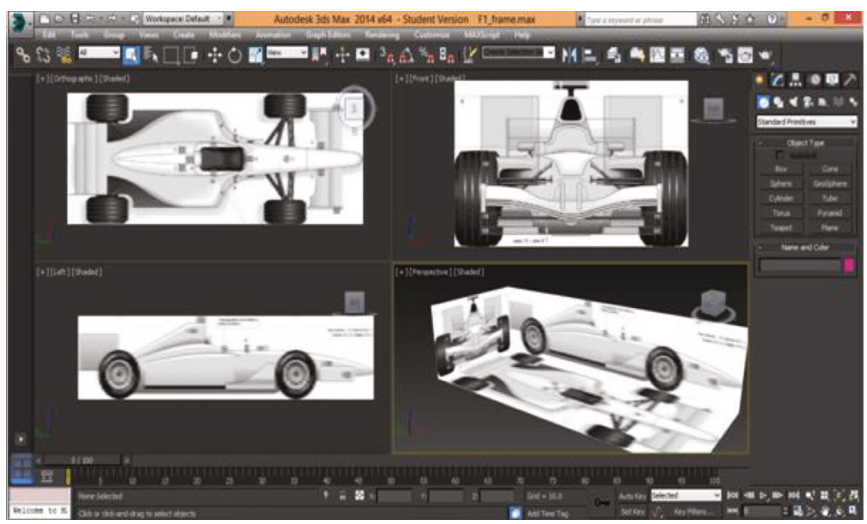

Figure 10: Reference images.

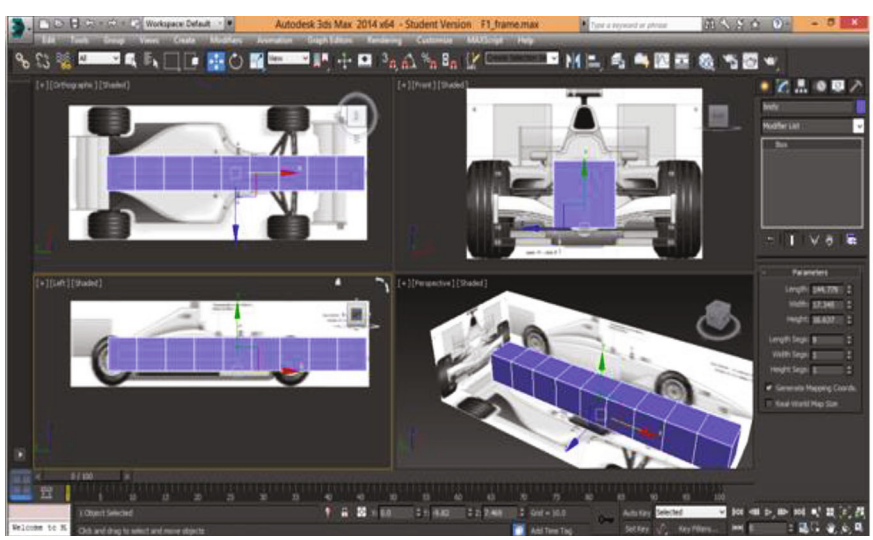

Figure 11: Adding rectangular primitive.

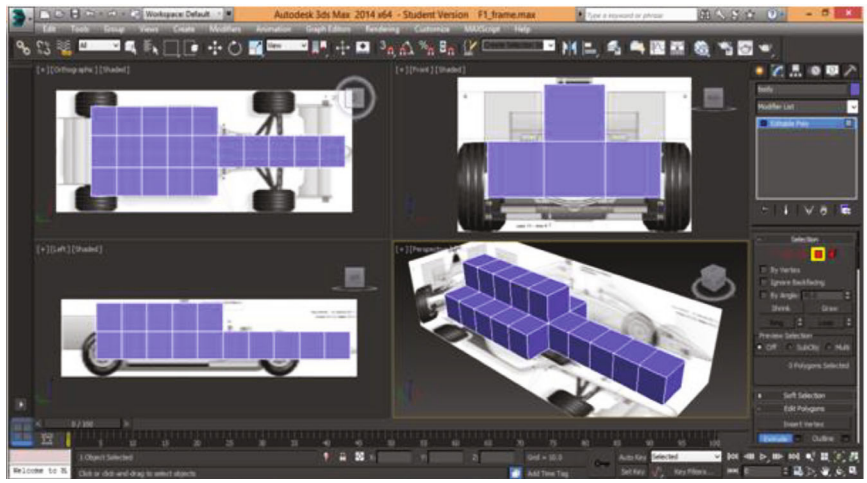

Figure 12: Selected faces are extruded.

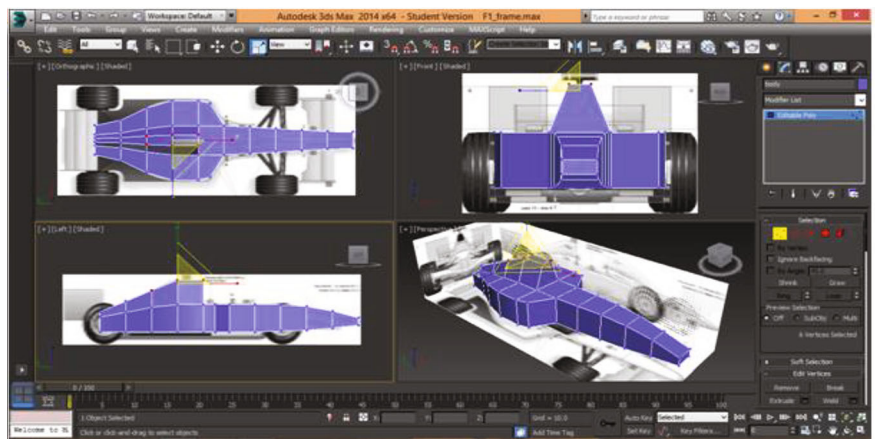

Figure 13: Vertices aligned with reference image.

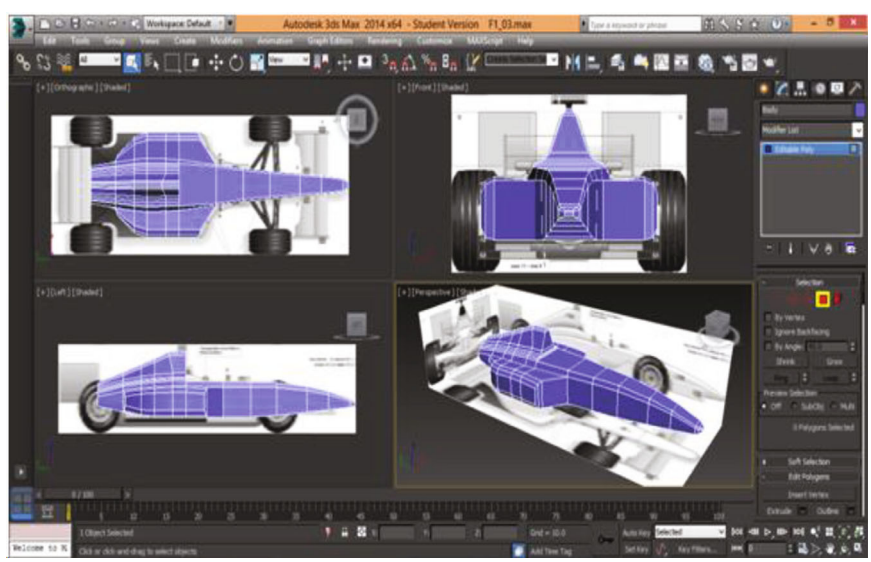

Figure 14: Fine tuning the mesh.

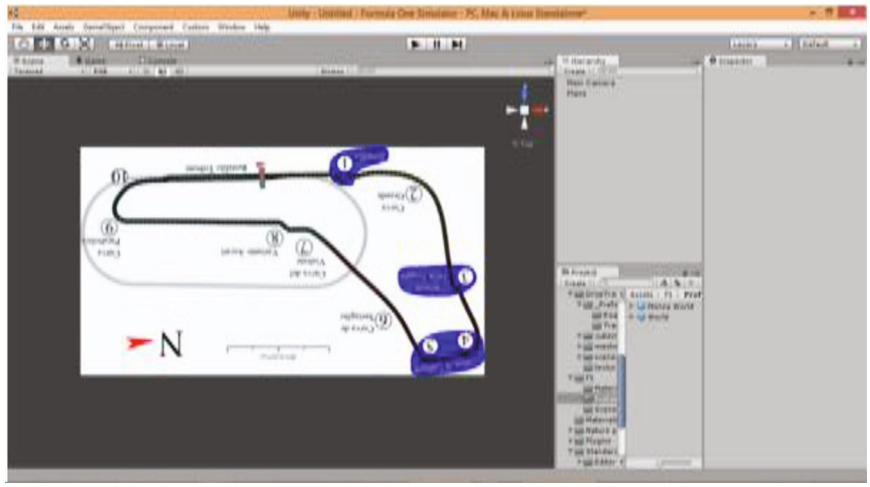

Figure 15: Adding reference image.

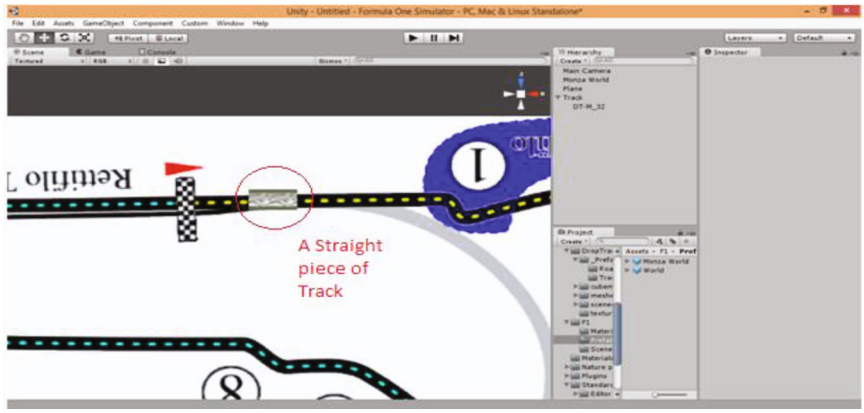

Figure 16: A piece of the track.

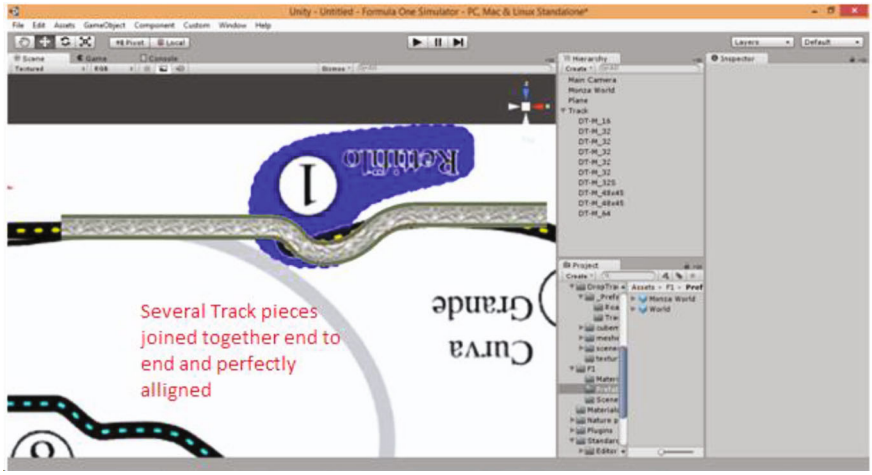

Figure 17: Adding more track pieces.

the user to see a more strongly characterized vehicle. Now, in order to make the car move, with the input from user, key mapping of various keys on keyboard is done which translates the car forward, backward or sideways. For example, we bind the forward movement of the car with "up arrow" key. When the key is pressed, the coordinates of the car are updated in the game world from say (0,0,0) to (10,0,0) which translates a forward movement. In this way, it can be done to manipulate any behavior of the car. We have added many more parameters to further enhance the handling of the end-user car to bring about more realism. For example, it is no longer possible to do cornering at high

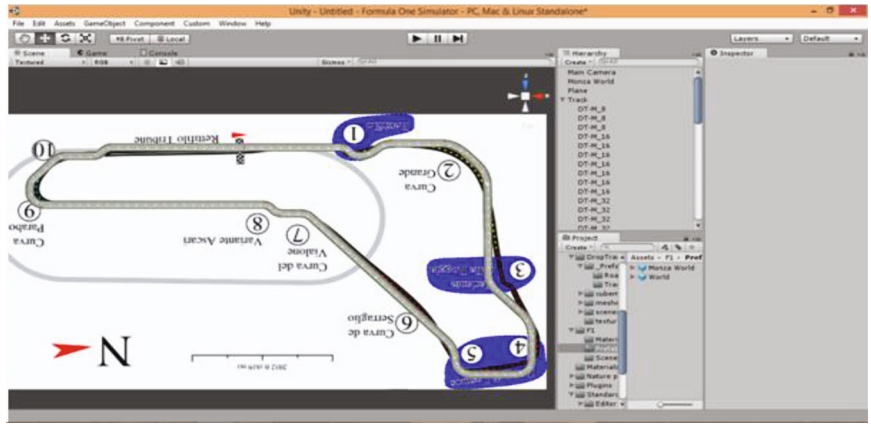

Figure 18: Completed track.

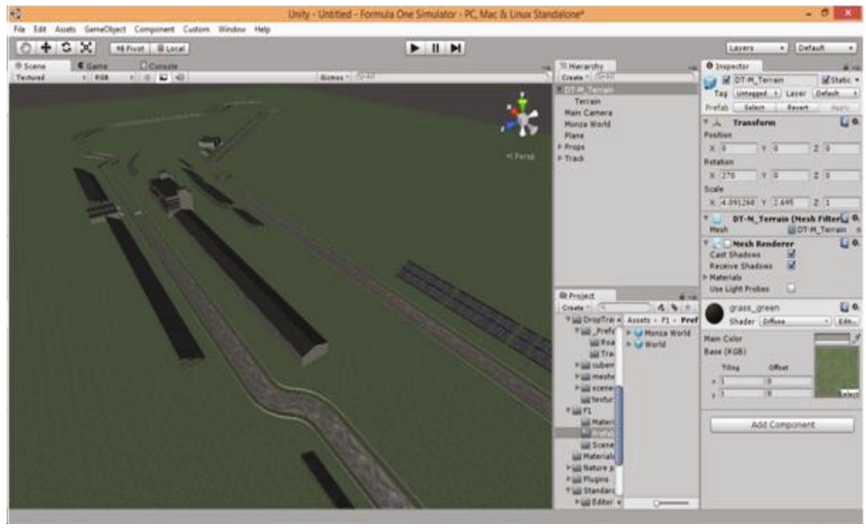

Figure 19: After adding terrain and props.

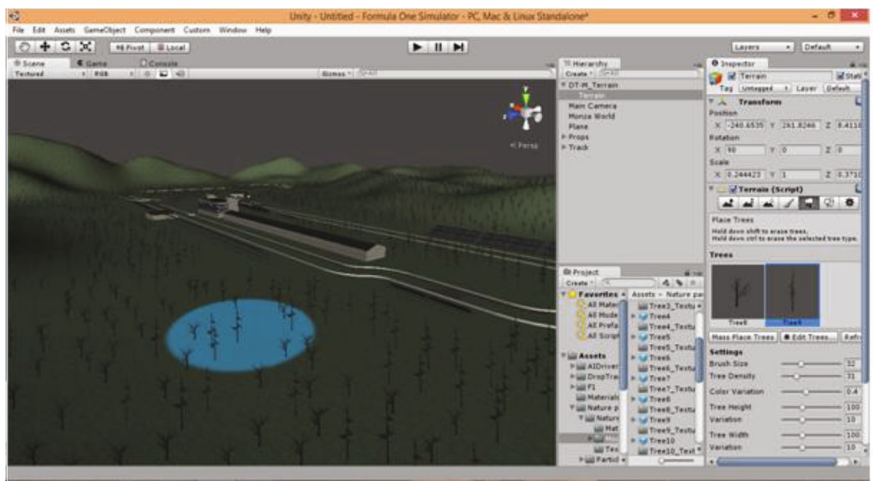

Figure 20: Adding terrain modifiers.

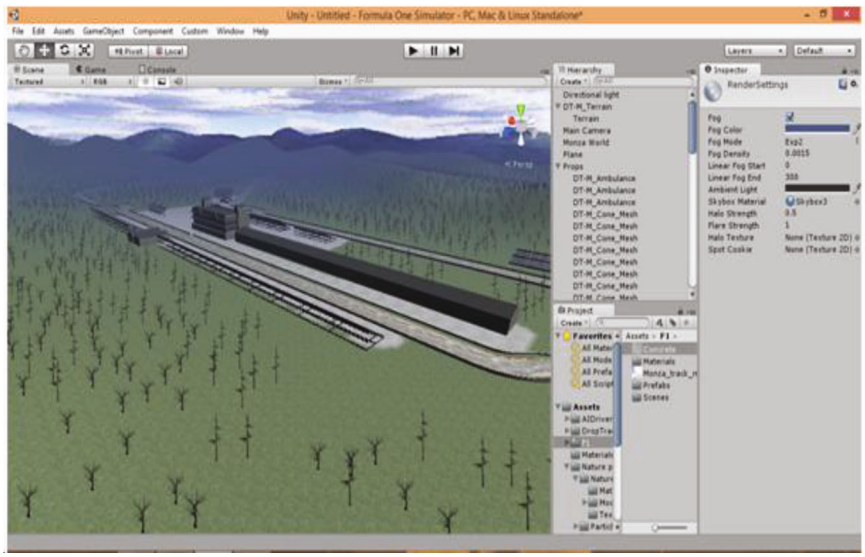

Figure 21: Adding visual effects.

speeds, and the car will skid. So, this ensures that the driver that is the end-user drives around the track in a disciplined manner. Also, collision mechanics have been added to every car; so, the user cannot simply go around hitting every other vehicles on track. Some of the parameters used here are the following:

(a) Inertia-as in example given above, due to inertia, the car will not take turns at high speed

(b) Gear ratio-the ratio of velocity applied to the car based on current gear

(c) Antiroll bar-in the unlikely chance of a vehicle flipping over, this parameter will put the vehicle back on track

(d) Collision-damage mechanics have been added to the user controlled car. So, whenever a collision occurs, the vertices of the car gets rearranged to show the aesthetics of the damage

(e) GUI-enhanced graphical user interface like speedometer and RPM meter is provided to the end user to better understand the behavior of their vehicle

6.2. AI Car Handling. This module deals with the throttle, steering, braking, and cornering of the computer or AI controlled vehicles. We shall essentially examine the behavior of the AI on a race track represented in Figure 22. More importantly its behavior with the respect to the end-user vehicle. The throttle and steering of the AI-controlled cars are taken care of with the help of scripting. The implementation language used here is the C# programming language. The AI Driver Controller script is being used to take care of the throttle and steering of the AI-controlled vehicles. For braking and cornering purposes, there are detector lines at various angles around the bot vehicle (AI controlled). The faster the car moves, the further are the range of detection lines. When these lines encounter an obstacle, then the AI will change its course accordingly. This is basically how the AI detects an obstacle on its course. As an example in the following slide, the black rectangle represents the car, the yellow lines represents the detector lines running at 0° with the axis of the car, and the brown squares depicts a unit of an obstacle. In the following example, the detection lines (colored lines) of AI driver vehicle detects another racer. When this happens, the AI driver checks the side using the angled detection lines if there is any obstacle. If one or both the sides are free of obstacles, it will then change its steer angle and deviate from its current path to avoid colliding with the other racer.

(a) Rigid body-this parameter is mostly used in the translation of the car and to provide mass

(b) Drag-the force acting against the forward motion of the car

(c) Wheel alignment-this is used to turn the wheels when the vehicle makes a turn

(d) Waypoints-the waypoints corresponds to specific coordinates in the map which defines the racing line. These waypoints are used by the AI vehicles as described below

(e) Current heading-this is used by the AI vehicles to keep track of the waypoint it currently crossed. Target heading-this is used by the AI to move to the next waypoint

(f) Waypoint container-an array where all the waypoints, installed on the track, are stored

(g) Engine torque-the force that propels the vehicle towards the next waypoint. Gear ratio-the ratio of velocity applied to the car based on current gear

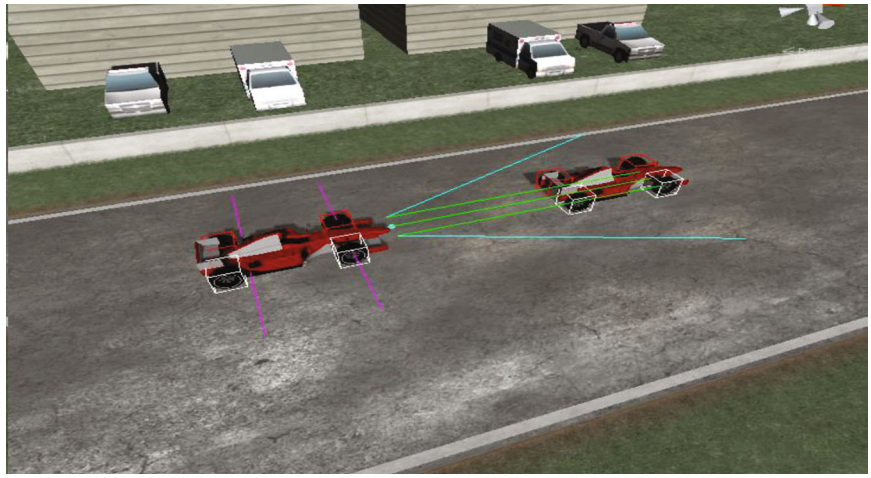

Figure 22: Detection lines of AI driver.

(h) Center of mass-this parameter is used to hold the car on the road so that it does not flip. Also, it is used to show the animations of turning. Center of mass is calculated from the rigid body parameter

(i) Wheel collider-this parameter keeps the car from falling through the track and also uses for the wheel alignment parameter for rotation

(j) Steering sharpness-this parameter decides how the AI vehicles will quickly turn around in a corner

6.3. AI Behavior. This module covers the behavioral aspects of the bots vehicles like collision, overtaking, slipstreaming, and aggression. Collision spheres or cubes which defines collision range of that vehicle will be assigned to vehicles to check for collision with vehicles or objects. In the following Figure 23, the green rectangles depicts two racing vehicles.

The bots vehicle will check for empty spaces around another race vehicle using the generated detection lines (yellow lines). If the detection lines does not register another vehicle or object (like a wall), then the bots will generate temporary overtake paths (red lines) around the vehicle in front. The bots will then determine the safest overtake path, increase its speed, temporarily divert from its racing line (blue line) to the overtake path, and attempt an overtake (in this case, path B). Alternatively, the bots can slipstream behind another vehicle and use the extra burst of speed for an overtake. Aggression will define how likely is an AI controlled bots to overtake or block an overtake attempt. All the bots will be assigned an aggression factor which will define the aggression level of a bot. Higher aggression factor means that the bots are more likely to overtake or block an overtake attempt.

6.4. Racing Lines. The racing line is the most optimized path the driver can use to conserve speed during cornering with as little braking as possible. The entire racing line was constructed and placed on the Monza track. This line provides the AI cars the most optimized guideline so they can finish the race as quickly as possible. The bots will have a path covering the length and width of the entire track in which it will have full freedom of movement (gray area).

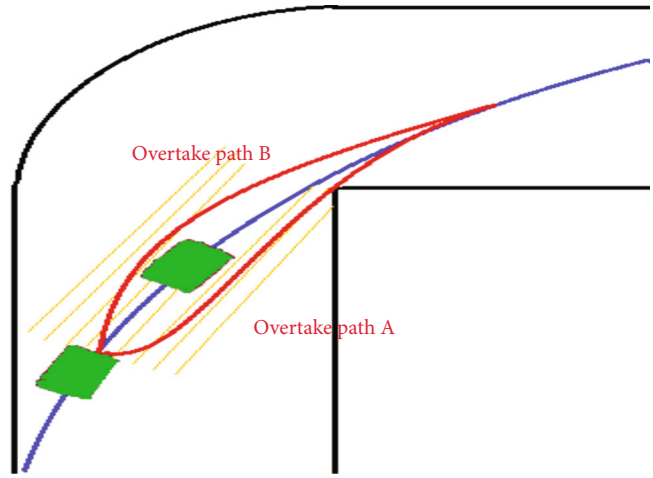

Figure 23: The AI behavior.

Each track will be set up with a racing line (colored line). The racing line is the route the vehicle must take in order to minimize the time taken to complete the course. The bots will follow the racing line unless it encounters a difficulty such as an obstacle.

The racing line can even provide visual aid to the player car so as to drive optimally. The opponent cars or AI controlled vehicles will follow the designated racing line and complete laps till the finish line. Braking zones are added to the track at necessary or critical points on the track, and the AI will break the vehicle whenever it encounters a corner as shown in Figure 24.

6.5. Obstacle Avoidance, Viewpoints, and Waypoints. The AI-controlled Formula One vehicles are modeled and programmed with the help of obstacle avoidance, viewpoint, and waypoint techniques. Obstacle avoidance techniques are used by the AI-controlled vehicles to steer away from obstacles in front and to their sides. This is done with the help of collision detectors placed on the vehicles. This helps them to avoid collision with other vehicles on track.

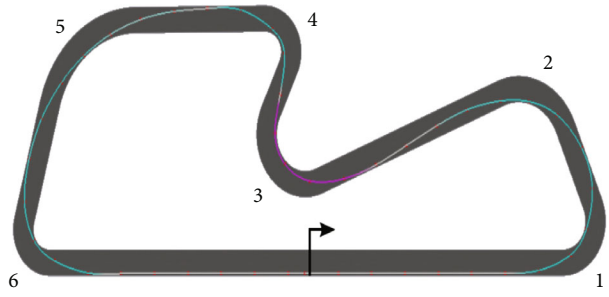

Figure 24: Racing line.

Viewpoints are essentially "eyes" for all the AI-controlled vehicles. This helps the vehicles to move forward and tackle corners. The dot at the nose of the vehicle in the picture below is a viewpoint while the colored lines are the collision detector lines. Waypoints are essentially points on the race track that are assigned individually to each AI vehicle. The vehicles may or may not follow these points. The waypoint system is mainly implemented to help the AI-controlled vehicles to recover after it has go off the race course. These techniques are implemented to recreate the actual Formula One driver behavior as an extension of object detection and recognition.

7. Implementation

Once all of the sculpting and the World Design is completed, the AI drivers and the player car must be implemented into the scene. To achieve this purpose, we again use the Unity3D game engine.

7.1. Player Car Setup. Initially, the player car in implemented into the scene. At first, the Formula One model sculpted in 3ds Max is imported into Unity3D. The axis of the F1 model must be adjusted so that it does not cause any problems in Unity3D. Also, the textures of the model have to be manually adjusted. The F1 car model will be constructed of separate parts, i.e., 4 individual wheel models and a body model that will be children of parent F1 car model. Once the model is completely imported, it is placed in the starting grid of the scene. First order of business is to assign the various collider meshes to the body and wheels of the car. Two rectangular collision meshes are placed on the car as illustrated in Figure 25. These two meshes make sure that the car does not fall through the road or go through walls and other cars. These meshes are also used to register collision between two cars or collision with the limits of the road.

Then, the wheel colliders are added to each of the individual wheels as shown in Figure 26. The wheel colliders are used to animate the wheels (like rotating). Hence, the wheel colliders must be precisely aligned to the wheel meshes otherwise the car will look like its floating during runtime.

Once all the colliders are in place, then an empty game object is created and named as center of mass. This is later used to define the centre of mass of the car. This object is placed roughly on the middle of the car. Then, the rigid body modifier is added to the Player Car. This modifier gives mass to the vehicle which is used by the code to calculate the acceleration and velocity of the car. Now, the code for the movement is assigned to the Player Car. During runtime, the code first checks for the coordinates of the player car's initial position. When the key responsible for forward movement is tapped, then the code propels the car in a particular axis by updating the position of the car. For example if the car initially is at (0,0,0), after tapping the forward movement key, the code updates the coordinates of the car to (10,0,0). Then, between the initial and the final position, the animation for the wheel rotation is applied to the wheels. Hence, it gives the impression of a forward movement of a car. When the key for sideways movement is tapped, then a parameter called steer angle calculates how long the key is tapped. Based on the duration, the steer angle parameter rotates the axis of the car according to the direction. For example, if the car is aligned at 0 degrees with global x axis and the turn right key is pressed, then the steer angle parameter will rotate the car along with the axis by say 30 degrees. During that time, the wheels will also be animated to turn by a 30 degree angle. Thus, the rotation of the car along with the turning of the wheels give the impression of the turning of a car. First, the wheels turn, then the car is rotated to provide a smooth looking turn. This is illustrated in the following Figures 27–29.

Finally, the viewing camera needs to be assigned to the Player Car. To accomplish this, the camera main object is transposed/rotated/scaled until the view of the camera is behind the player car or on the cockpit of the car as required. Then, the camera is inserted into the hierarchy of the Player Car. In this way, whenever the Player Car moves, so will its children. Hence, the camera will follow the car. Thus, the basic setup of the player car is complete. After this, the handling parameters of the car like max speed, gear ratios, forward and sideway friction, and braking force are fine tuned so as to bring the feel of the handling of a F1 vehicle.

7.1.1. AI Driver Setup. The car models for the AI's are first imported from 3ds Max into Unity and placed at their starting grid alongside the player car. These also contain separate meshes for the body and wheels. Next, the various colliders, rigid body, and center of mass are assigned to the AI driver similar to the Player Car. Then, the waypoints are assigned for each vehicle. The AI car uses these waypoints to roughly navigate through the track. Also, the AI car respawns at the last checkpoint it passed through should it go off track or collide with another vehicle. Hence, the waypoints are a very important aspect of the AI drivers. The curve drawn by these waypoints represents the racing line shown in Figure 30.

Next, the viewpoint or just view is assigned to the AI vehicle as shown in Figure 31. The view contains the detection lines which detects if an obstacle is in front or side of the car. It behaves like the eyes of the vehicle. The view must be assigned to the nose of the vehicle. These lines detect objects in front of the Player Car. Then, views have to also be placed on both sides of the car to detect objects on the side. These are placed near the wheels so as to relay the most accurate

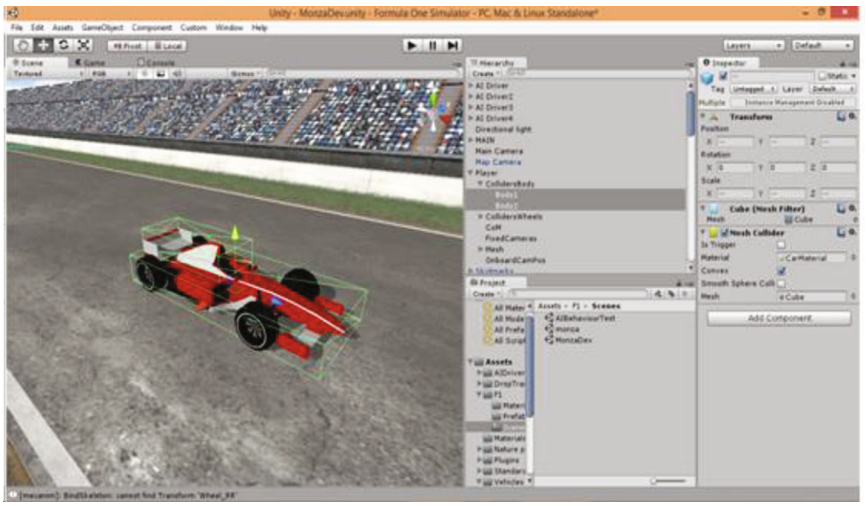

Figure 25: Rectangular colliders (green boxes).

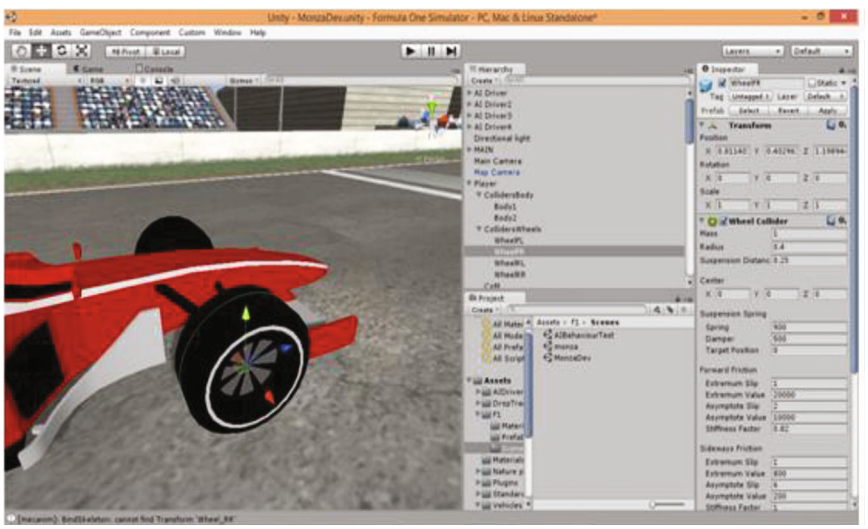

Figure 26: Wheel colliders(green circle around FR wheel).

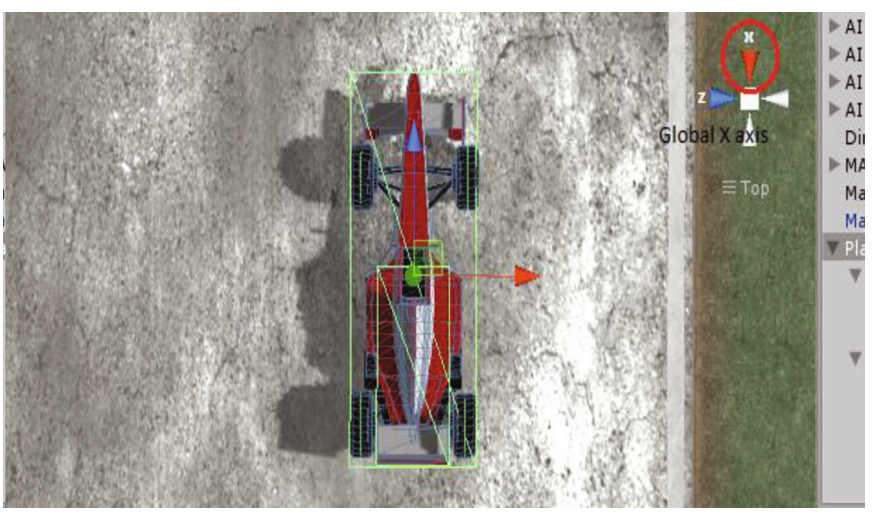

Figure 27: Aligned to global x axis.

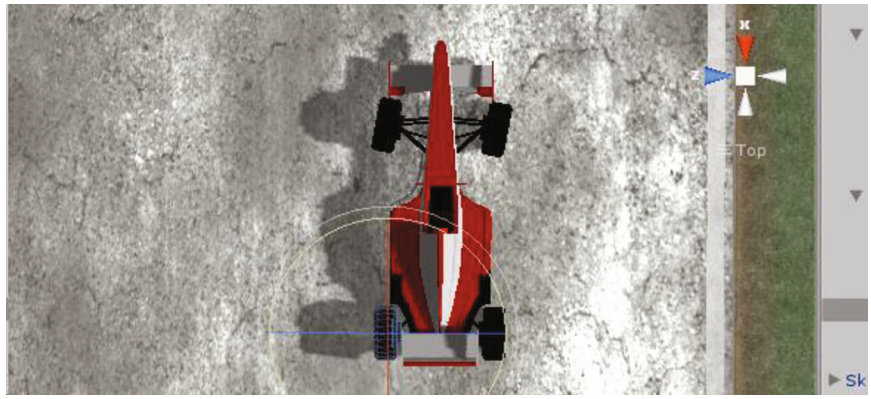

Figure 28: Turning of wheels.

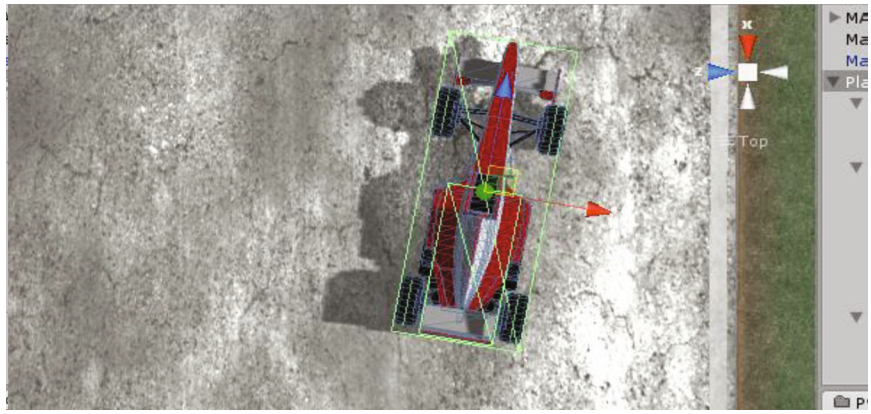

Figure 29: Turning of whole car.

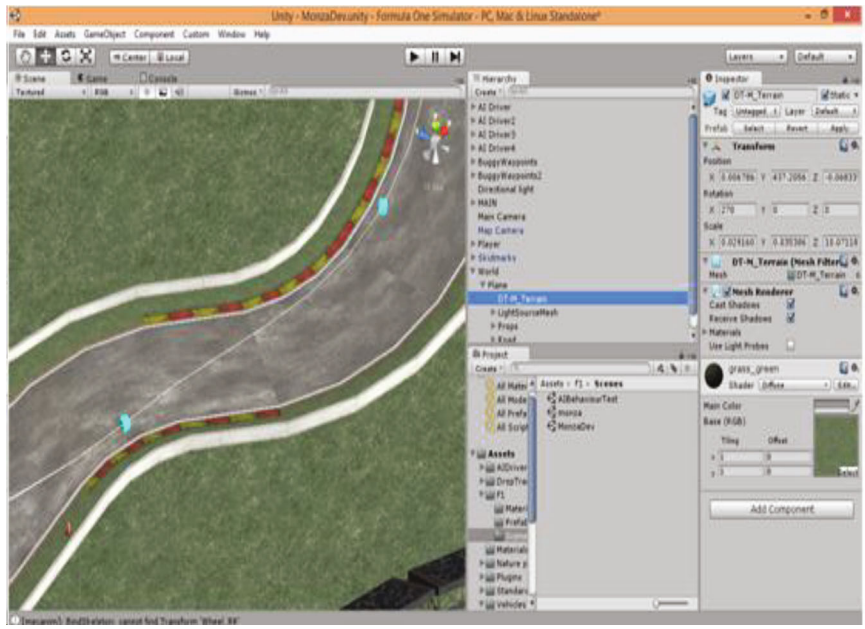

Figure 30: The small blue boxes represent waypoints and the white line represents the racing line.

position of objects at the sides and also since the wheels are used to turn the car.

Then, the code for the movement of the AI car is assigned. It works similar to the movement of the Player Car code but in this case, the velocity, acceleration, and turning of the car depends upon another piece of code. This code tells the car its speed or turn bounds. The code for the AI controller is assigned next. This code uses the detection lines

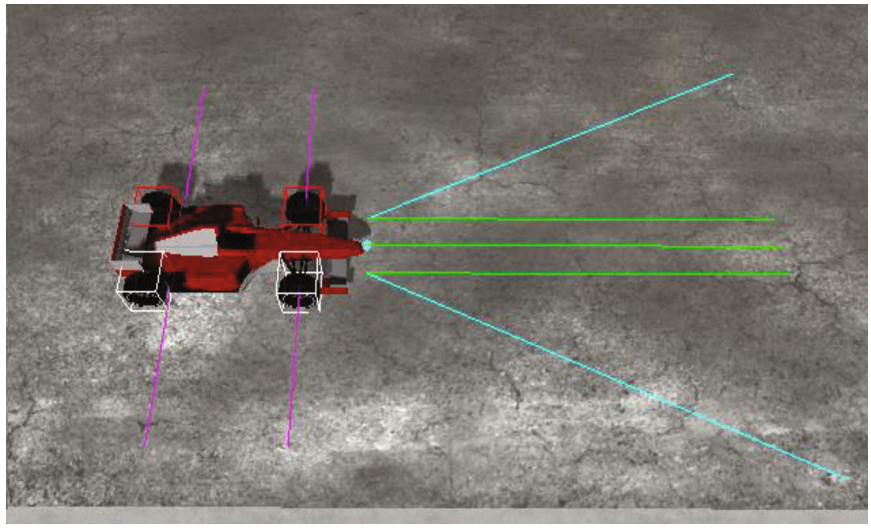

Figure 31: Viewpoint/view.

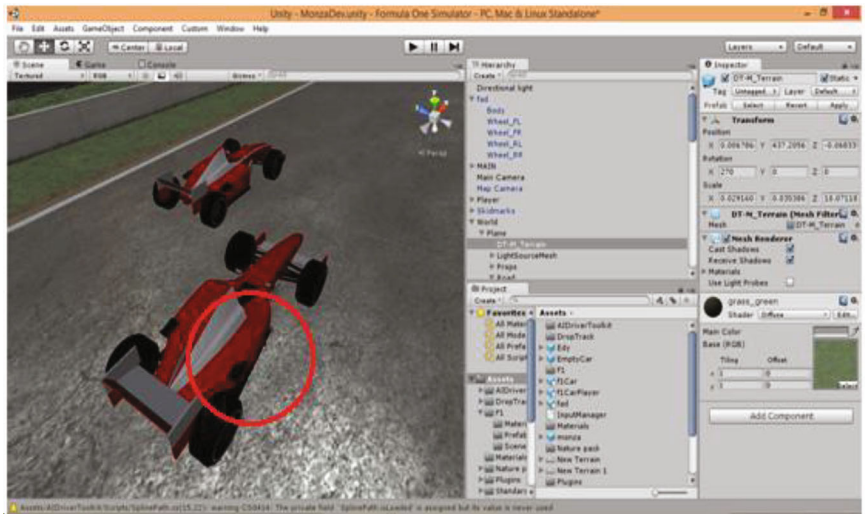

Figure 32: Deformation of mesh due to collision.

of the Viewpoint to steer the AI vehicle around the track. When the detection lines does not detect any object in its path, then the velocity of the car is set to its speed limit. Whenever there is a corner, the AI car detects the boundary, reduces its velocity by using the brake parameter, and accordingly turns itself. When it encounters another vehicle or object, it brakes and deviates from current path to avoid collision.

7.1.2. Collision Damage. After setting up all the cars, the damage mechanics are added. The code for damage is assigned to both the Player and the AI cars. When the collider meshes of two cars come in contact (occupy the same world space), then it is registered as a collision. The code then effectively rearranges the vertices of the mesh by a small factor to show the damage to the car. Even in the case of a collision with a wall the vertices of the car mesh are deformed a little. However, the vertices of the wall are not deformed as shown in Figure 32. This is due to performance reasons. The collision mechanics uses a lot of system resources. The track has separate modular meshes that are arranged to form the track. So, adding the damage code to all of the separate meshes could result in a significant drop in FPS during runtime. The damage mechanics also effects the performance of the Player Car. The more the damage it takes, the harder it becomes to effectively control the car. For example, if the front left wheel takes damage, then the car will turn a little towards the right even when the "turn right" key is not tapped. This adds some depth to the handling of the Player Car.

7.1.3. Final Game Build. Once all the game objects are integrated with their respective codes, these are then added all together to the scene. The scenes exist only inside the game engine that is Unity. So, the next order of business is to make it into a stand-alone software application. To achieve this, we use the inbuilt build feature of the Unity engine. The build feature takes in all the separate scenes and along with the visual effects generates an executable file of .exe format.

There are two modes available to operate the simulator: training mode and automatic mode. In training mode, the user uses virtual control to drive the car using the steering angle, the throttle, and the brakes. You can do this with a keyboard or a mouse. During training mode, the simulator will start saving driving data to a specific spot. Driving data consists of each image on the left, right, and front sides of the vehicle and the corresponding steering, throttle, and brake measurements.

In order to collect training data, the car was driven in training mode around the virtual track for five full laps using the centre lane driving behavior, in order to avoid reaching the road boundaries on both sides. The data was further improved by numerous selective driving acts around corners to account for oversteer and understeer, with several obstacles in different directions approaching to hit the vehicle at different angles. A total of 5,000+ data points were collected from the training run. After the rise, the total number of data points was more than 10,000. We used Keras deep learning library. The control angle was the only measure used as the output for the model; the convolutionary neural networks (CNNs) were used to map raw pixels from the front-facing camera to the steering controls for the self-driving car. Training data was 80% and 20% of the data used for validation. As expected, the mean squared error loss of both the training and validation sets decreased with the number of iterations At the end of the training process, the model was able to steer the vehicle on the track without leaving the route, thus avoiding road obstacles.

8. Conclusion and Future Scope

The final build of the scenes has produced a simulation program for Formula One cars. In this simulator, the end user will be able to take the wheel of one of the F1 cars. The AI cars will run along with the Player Car and behave like a real world Formula One car. It will try to overtake the player whenever it sees an opportunity. It will also avoid collision with other objects or vehicles. The simulator will contain with several racing circuits all of them providing various racing challenges for the budding racer. The circuits contains various type of corners like hairpin bends and U-turns. The circuits will also contain visual aids for the player to drive effectively in the track-like kerbs. To get the actual feel of the F1 car, there will also be a cockpit camera so that the users can view the track from the eyes of a racer.

While designing F1 simulator, we should make a note of behavior of virtual traffic and weather conditions, and the road layout can be manipulated (offline or in real time) as a function of the training needs or research aims in the future. We have to consider "safety" related features in the future research.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request (mmasud@tu.edu.sa).

Conflicts of Interest

The authors declare that they have no conflicts of interest to report regarding the present study.

Authors' Contributions

Amutha Balakrishnan contributed to the conceptualization, data curation, formal analysis, methodology, writing-original draft, and software. Kadiyala Ramana contributed to the data curation, writing-original draft, investigation, resources, validation, and software. Gaurav Dhiman contributed to the supervision, writing–review and editing, project administration, and visualization. Gokul Ashok contributed to the conceptualization, investigation, resources, validation, and software. Vidhyacharan Bhaskar contributed to the supervision, writing–review and editing, and visualization. Ashutosh Sharma contributed to the supervision and writing–review and editing. Gurjot Singh Gaba contributed to the writing–review and editing and investigation. Mehedi Masud contributed to the writing–review and editing and funding acquisition. Jehad F. Al-Amri contributed to the writing–review and editing and funding acquisition.

Acknowledgments

Taif University Researchers Supporting Project number (TURSP-2020/211), Taif University, Taif, Saudi Arabia.

References

[1] B.-H. Chen and S.-C. Huang, "An advanced moving object detection algorithm for automatic traffic monitoring in real-world limited bandwidth networks," *IEEE Transactions on Multimedia*, vol. 16, no. 3, 2014.

[2] X. Bai, Q. Li, L. Latecki, W. Liu, and Z. Tu, "Shape band: a deformable object detection approach," in *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, 2009.

[3] S. Bhattacharya, P. K. R. Maddikunta, I. Meenakshisundaram et al., "Deep neural networks based approach for battery life prediction," *CMC-COMPUTERS MATERIALS & CONTINUA*, vol. 69, no. 2, pp. 2599–2615, 2021.

[4] A. Kaur and G. Dhiman, "A Review on Search-Based Tools and Techniques to Identify Bad Code Smells in Object-Oriented Systems," in *Harmony search and nature inspired optimization algorithms,*, pp. 909–921, Springer, Singapore, 2019.

[5] M. Garg and G. Dhiman, "Deep convolution neural network approach for defect inspection of textured surfaces," *Journal of the Institute of Electronics and Computer*, vol. 2, no. 1, pp. 28–38, 2020.

[6] D. S. Rajput, A. P. Singh, Y. Agarwal, P. K. Reddy, and G. T. Reddy, "Feature selection analysis for multimedia event detection," in *IOP Conference Series: Materials Science and Engineering*, vol. 263, no. 4p. 042004, IOP Publishing, 2017.

[7] O. Danielsson, S. Carlsson, and J. Sullivan, "Automatic learning and extraction of multi-local features," in *International Conference on Computer Vision*, Kyoto, Japan, October 2009.

[8] V. Ferrari, F. Jurie, and C. Schmid, "From images to shape models for object detection," *International Journal on Computer Vision*, vol. 87, no. 3, pp. 284–303, 2009.

[9] J. Shotton, A. Blake, and R. Cipolla, "Multiscale categorical object recognition using con-tour fragments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 7, pp. 1270–1281, 2008.

[10] J. De Winter and J. Wagemans, "Perceptual saliency of points along the contour of everyday objects: a large-scale study," *Perception & Psychophysics*, vol. 70, no. 1, p. 50, 2008.

[11] P. Felzenszwalb and J. Schwartz, "Hierarchical matching of deformable Shapes," in *Computer Vision and Pattern Recognition*, pp. 1–8, Minneapolis, MN, USA, June 2007.

[12] V. Ferrari, T. Tuytelaars, and L. Van Gool, "Object detection by contour segment networks," *Lecture Notes in Computer Science*, vol. 3953, p. 14, 2006.

[13] J. Shotton, A. Blake, and R. Cipolla, "Contour-based learning for object detection," in *International Conference on Computer Vision*, pp. 503–510, Beijing, China, 2005.

[14] S. Hermann and R. Klette, "Global curvature estimation for corner detection," *Image and Vision Computing New Zealand*, 2005.

[15] D. Martin, C. Fowlkes, and J. Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 530–549, 2004.

[16] T. Sebastian, P. Klein, and B. Kimia, "Recognition of shapes by editing shock graphs," *IEEE International Conference on Computer Vision*, pp. 755–762, 2001.

[17] R. Kumar and G. Dhiman, "A comparative study of fuzzy optimization through fuzzy number," *International Journal of Modern Research*, vol. 1, no. 1, pp. 1–14, 2021.

[18] I. Chatterjee, "Artificial intelligence and patentability: review and discussions," *International Journal of Modern Research*, vol. 1, no. 1, pp. 15–21, 2021.

[19] P. K. Vaishnav, S. Sharma, and P. Sharma, "Analytical review analysis for screening COVID-19 disease," *International Journal of Modern Research*, vol. 1, no. 1, pp. 22–29, 2021.

[20] G. Jianhong, H. He, Z. Wenxuan, and Y. Liang, "Research on real-time collision detection for vehicle driving in the virtual environment," in *IEEE international conference on information and automation*, China, June 2008.

[21] C. I. Tan, C.-M. Chen, W.-K. Tai, and S.-J. Yen, "An AI-tool: Generating paths for racing game," in *Seventh International Conference on Machine Learning and Cybernetics*, Kunming, July 2008.

[22] T. P. Hartley and Q. H. Mehdi, "In-game adaptation of a navigation mesh cell path," in *17th International Conference on Computer Games*, Louisville, KY, USA, 2012.

[23] V. Sezer and M. Gokasan, "A novel obstacle avoidance algorithm:"follow the gap method"," *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1123–1134, 2012.

[24] J. Oroko and B. Ikua, "Obstacle avoidance and path planning schemes for autonomous navigation of a Mobile robot: a review," *Sustainable Research and Innovation Proceedings*, vol. 4, 2012.

[25] Y. Zhu, T. Zhang, J. Song, and X. Li, "A new hybrid navigation algorithm for mobile robots in environments with incomplete knowledge," *Knowledge-Based Systems*, vol. 27, pp. 302–313, 2012.

[26] A. Sgorbissa and R. Zaccaria, "Planning and obstacle avoidance in mobile robotics," *Robotics and Autonomous Systems*, vol. 60, no. 4, pp. 628–638, 2012.

[27] C. L. Kumari, "Building algorithm for obstacle detection and avoidance system for wheeled mobile robot.," *Global Journal of Research Engineering*, vol. 12, no. 11-F, 2012.

[28] S. K. Kalmegh, D. H. Samra, and N. M. Rasegaonkar, "Obstacle avoidance for a mobile exploration robot using a single ultrasonic range sensor," in *2010 International Conference, Emerging Trends in Robotics and Communication Technologies (INTERACT)*, pp. 8–11, Chennai, India, December 2010.

[29] A. Yufka and O. Parlaktuna, "Performance Comparison of Bug Algorithms for Mobile Robots," in *Proceedings of the 5th international advanced technologies symposium*, Karabuk, Turkey, May 2009.

[30] http://www.fritz-hut.com/vector-field-histogram-vfh/.

[31] I. Ulrich and J. Borenstein, "Vfh∗: local obstacle avoidance with lookahead verification," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, pp. 2505–2511, San Francisco, CA, USA, 2000.

[32] G. Li, A. Yamashita, H. Asama, and Y. Tamura, "An efficient improved artificial potential field based regression search method for robot path planning," in *2012 IEEE International Conference on Mechatronics and Automation*, pp. 1227–1232, Chengdu, China, August 2012.

[33] K. H. Chen and W. H. Tsai, "Vision-based obstacle detection and avoidance for autonomous land vehicle navigation in outdoor roads," *Automation in Construction*, vol. 10, no. 1, pp. 1–25, 2000.

[34] T. R. Gadekallu, N. Khare, S. Bhattacharya, S. Singh, P. K. R. Maddikunta, and G. Srivastava, "Deep neural networks to predict diabetic retinopathy," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–14, 2020.

[35] Y. Liu, Q. Sun, A. Sharma, A. Sharma, and G. Dhiman, "Line monitoring and identification based on roadmap towards edge computing," *Wireless Personal Communications*, pp. 1–24, 2021.

[36] T. Reddy, R. M. SP, M. Parimala, C. L. Chowdhary, S. Hakak, and W. Z. Khan, "A deep neural networks based model for uninterrupted marine environment monitoring," *Computer Communications*, vol. 157, pp. 64–75, 2020.