

Sivan Syed - CV and Integration November 2024 - Page 1 of 5

Dance Dance Revolution - Augmented Reality (DDR-AR)

Role

Object detection and code integration

In this project, I worked on object detection with Aleena Khatum and integrating the individual parts from each member into the final cohesive product in JavaScript (JS). Specifically, I was tasked with helping develop code that would detect when a player's appendage had entered a designated region and coordinating the hitbox detection with the on-screen interface, so that falling arrows could be timed and scored.

Methodology

OpenCV

Our team decided on using OpenCV (Open Source Computer Vision Library) in order to implement object detection. OpenCV is a free, open-source software library that helps with computer vision and machine learning. It's used to process images and videos in real-time^[1]. This decision was motivated both by time constraints and ease of compatibility. Since the team had a limited amount of time for the entire project, we found it was best to draw upon previously developed resources and adapt them to our specific needs in making sure that we maintain versatility with anyone's laptop camera. We researched and found 2 primary methods of detecting movement, which make use of the foreground background subtraction^[2].

Foreground Background Subtraction

Foreground background subtraction is a computer vision technique that separates moving objects from the background in images or videos^[3]. This is accomplished by first setting the background with a static camera. When the background is first initialized, anything not moving is considered the background. By subtracting the current frame and the background, a foreground is generated, as any moving object in front of another object will be in the foreground.

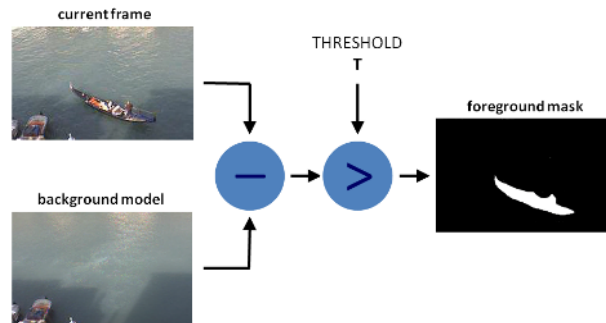


Figure 1 : Visualization of Background Subtraction from <https://docs.opencv.org/>

The primary advantage of this method is that background subtraction allowed us to only highlight the moving parts of an individual. This meant no initialization process to process and match an individual's attire, drastically reducing complexity. Additionally, since we specifically only detect movement of an object, and not just position of an object, this forced the user to be active in the game and had to properly step into the corresponding space on screen.

Segmentation

In order to properly apply this to the context of the game, we needed to segment out regions of the full frame for the video being fed through. We decided to create regions where movement would be observed. These Regions of Interest (ROIs) would be observed in order to detect when a player stepped onto a "plate" and register accordingly within the game to interact with the arrows.

Movement Detection: Contours counting

To ensure increased accuracy, we first used the method of counting the number of contours, n , within an ROI, and if $n > \text{Threshold}$, then we can consider that ROI to be "on". If there is an arrow in the corresponding lane on screen while the ROI is on, then the arrow dissolves and points are scored. Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity^[4]. Since we first apply the foreground background subtraction, resulting in a binary image, contours were simply determined by whether a long enough chain of pixels were white. Below is the result of background subtraction, where contours can then be counted.



Figure 2 : Background subtraction method and what the program “sees” after processing

Optimization: White:Black Ratio

In order to further accuracy as well as reduce hit detection time, we opted to combine both the number of contours in a region and the ratio of white pixels to black pixels. This allowed us to tweak parameters and decrease the contour count threshold for faster detection. Through several iterations, we found that a threshold ratio of just 0.11 white pixels to black pixels was enough to drastically improve the response time of the hit detection, while also seemingly not ever being accidentally activated.

Integration

The next task was to integrate each of the parts into a final working project. While Aleena and I had developed the object detection and computer vision system, one team member had created the 3D objects and object interaction in three.js. My final task for this game was to integrate the 3d objects from three.js within the game screen with the computer vision and object detection. This meant making sure that when the player was detected within a certain ROI, the appropriate lane would dissolve any descending arrows. This meant creating a flag to detect when the player activated an ROI and having the corresponding lane check for if an arrow is at the end of the lane, scoring points if there is.

User Optimization

The final task was to optimize the user interface. We mirrored the image displayed to the user so that stepping to the right in the actual world would correspond to activating the right lane in the display. We tweaked the size of the ROIs as well as the placement so that users could easily interact with individual ROIs. We also tweaked tolerances to increase responsiveness. Since there was a delay between when a user could sequentially activate an ROI, we had to ensure that an activation would be short enough so that the user could rapidly step into and out of an ROI and have the movement be detected.

Challenges

Optimization

One of the challenges we faced was optimizing the user experience, as this is a mostly subjective thing. However, through repetitive testing and validation through different people, we were able to see which parts of the process to iterate upon. One of which was the size of the ROIs. Initially, they were too small, resulting in users struggling to activate only one individual ROI.

Subsequently, the spacing between ROIs made it uncomfortable to make certain movements. As a result, we increased the spacing between ROIs as well as the vertical height of each of them.

Mirroring

After mirroring the display, we had to change the corresponding flags that would activate certain lanes. Although simple, we later discovered while in a frenzy to make final changes and submit the project, that the ROIs no longer worked. This was because the indices for them were [0,3] and not [1,4]. This initial assumption that the indices ran from [1,4] resulted in ROI and lane activations not correlating, and even lanes not activating at all. From this, I learned not to make assumptions about code and functions, especially when integrating code with each other.

Finding the best method

A large portion of time on this project was looking into object detection methods. We had discussed several options. Such as a CNN (Convolutional Neural Network), calibrating and tracking the color of the player's shoe, or actively tracking the contours of a player. After comparing the advantages and disadvantages of each of the methods, we decided that, for our challenge and the time we had, it was best that we use Foreground Background Subtraction in order to detect the player's movement. This consideration can be shown below in the provided table.

In the future, it would be ideal and most interesting to implement and compare a CNN to see how it performs versus the Foreground Background Subtraction method. This could be achieved by taking several pictures of shoes, and training a model to identify shoes, which could then be applied to segmented areas of the display.

Method	Complexity	Usability	Reliability
Foregorund Background Subtraction	Low	High	High
Shoe color detection	Medium	Low	Medium
Convolutional Neural Networks	Medium	High	High
Object Tracking	High	Medium	Low

Integration

Another challenge I faced was the integration of the object detection system and the three.js 3D modeling created by Aditya Angajala. The task resulted in me implementing a global variable “flag” which could be continuously checked in the 3d object modeling system. We faced issues of matching the correct ROI with its corresponding lane. The root of this issue was an incorrect fundamental assumption I had made in the code, which was thinking the three.js lanes ranged from [1,4], when instead values were [0,3].