Object Detection and Tracking in Virtual Environments

Nathan MacAdam (*Author*)
Florida Polytechnic University
Lakeland, United States

***Abstract*—3rd person video-game viewports provide a number of unique assumptions and challenges to object detection and tracking compared to real-life videos. User-interfaces, a high likelihood of object occlusion, and constant camera movement and rotation make traditional object tracking approaches difficult. However, 3rd person camera systems allow certain assumptions, like the player character being restricted to a relatively small movement area within in the camera viewport. Leveraging common assumptions about 3rd person cameras can overcome common object detection and tracking faults.**

Keywords—3rd person camera; object occlusion; environment geometry; region of interest, user interface, binarization, thresholding;

I. INTRODUCTION

A third-person camera system in a video game is defined by the functionality that the camera follows and rotates around the player character. Both camera movement and rotation challenge traditional tracking methods, as there isn't a stable background image to track moving objects against, nor the orientation of the object predictable. Especially in video games, the character is commonly occluded from the camera, and other non-environmental elements (e.g. user-interface (UI)) consume screen space within the viewport. Despite these challenges, the player character can be anticipated to remain in the relative center of the screen.

II. FEATURE IMPLEMENTATION

A. User-Interface Detection

Detecting static user-interface (UI) elements is accomplished by comparing frames for changes. Subtracting the previous frame from the current frame yields the difference between frames. By binary thresholding the frame difference, the pixels which are not static game elements are are made 0, while static elements remain 1 in the frame matrix. Multiplying this 'Static Score' matrix with subsequent frame's score matrices yields an image matrix representing the segmented UI.

Fig. 1. Calculating inter-frame difference (left to right: Black-and-white (BW) difference, Binarized difference) (threshold minimum of 225)

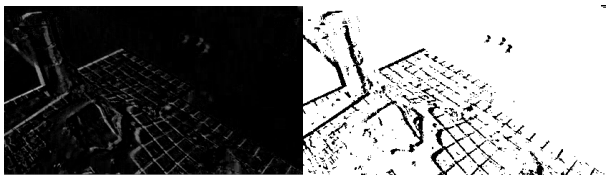


Fig. 2. UI detection from multiplied inter-frame differences (left to right: Detected UI, Dilated UI for drawing bounds)

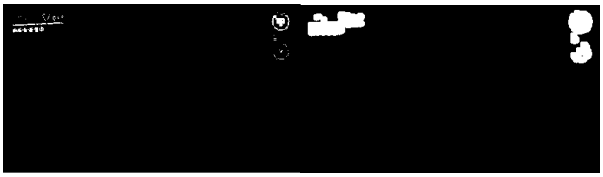


Fig. 3. UI Detection

B. Player Character Detection

With the assumption that the player character will be positioned in the relative center of the viewport throughout game play, an initial seed region of interest (ROI) is constructed at the center of the screen. By applying Canny edge detection, the contents of the ROI are filtered to approximately remove non-player character elements. The ROI is then incrementally expanded, comparing the edge-detected contents of the new ROI to the previous one until the change in the contents between ROI data is less than a provided threshold. Upon tracker failure, this process is re-stimulated to reset the tracking.

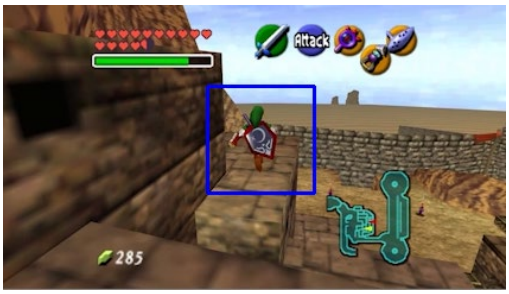
Fig. 4. ROI Detection

C. Player Character Tracking

Player character tracking is handled with OpenCV's implementation of the Discriminative Correlation Filter with Channel and Spatial Reliability (CSRT) tracking algorithm [1]. In comparison to OpenCV's other native tracking implementations, CSRT handled short durations of object occlusion and rotation of the tracking target more effectively, while also maintaining a reasonable execution time.

In addition to detecting a tracking failure through the implementation in the CSRT tracker, failure is also detected by checking the distance between the character origin position and the current tracking bounding box position, as well as checking that the bounding box has not grown to an exceedingly large size. If this distance exceeds a provided threshold, it will be considered a tracking failure and the tracker will be restarted.

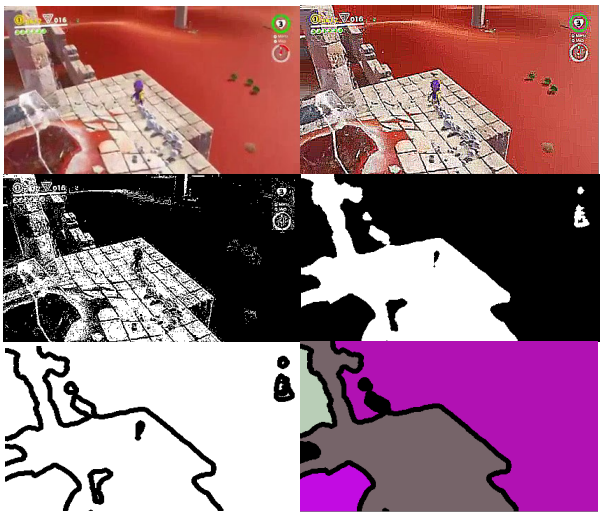
Fig. 5. CSRT Tracking Result



D. Environment Segmentation

The environment segmentation process attempts to detect segments of geometry that compose the game environment. First, applying a sharpening filter kernel to the frame image and then applying a binary threshold yields a binary image of the given frame. Then, to remove noise, resample the binary image by blurring it and apply a second threshold. Then, applying a second Laplacian filtering pass yields an edge detected image where texture-based edges, as opposed to geometry-based edges, are approximately filtered out, and the edges are continuous. Using the edge image, the contours of the original frame can be found and drawn to depict segments of level geometry, optionally removing segments with too small an area.

Fig. 6. Environment segmentation operations (left to right, then top to bottom: Original Frame, Sharpened Frame, Binarized Image, Resampled Binarized Image, Inverted Laplacian Filter, Final Segmentation)



III. Results

TABLE I. UI Detection Parameters

Binary Threshold Minimum	225
Binary Threshold Maximum	255
Dilation Kernel Structuring Shape	Morph Ellipse
Dilation Kernel Structuring Size	10 x 10

1) UI Detection across multiple resolutions (using sample gameplay from The Legend of Zelda: Ocarina of Time)

TABLE II.

Resolution (w x h, pixels)	**Detection Time Duration (frames)**	
	Frames until no extraneous data	***Real time in video capture (s)***
240 x 426	~470 frames	~15 seconds
480 x 854	~464 frames	~15 seconds
720 x 1280	~450 frames	~15 seconds

Lower resolutions yield smaller matrices and faster processing time. The overall reliability of the UI detection algorithm is scale invariant, so long as the image still maintains some detail.

2) UI Detection with sample gameplay footage featuring static UI elements

TABLE III.

Test Game Footage	Detection Time Duration (frames)	
	Frames until no extraneous data	***Real time in video capture (s)***
Super Mario Odyssey	~85 frames	~3 seconds
The Legend of Zelda: Ocarina of Time	~475 frames	~15 seconds
Star Wars: Battlefront II (PS2)	~500 frames	~20 seconds
Kingdom Hearts II	~270 frames	~15 seconds

3) UI Detection with sample gameplay footage not featuring static UI elements

TABLE IV.

Test Game Footage	Detection Time Duration (frames)	
	Frames until no data present in UI frame detection	***Real time in video capture (s)***
Rachet and Clank (PS4)	~200 frames	~7 seconds
Animal Crossing: New Horizons	~970 franes	~30 seconds

The time duration until UI has been completely segmented from the frame background is dependent on how varied the image content of the frames is.

TABLE V. CHARACTER DETECTION & TRACKING PARAMETERS

Detection Seed Width	40
Detection Seed Height	40
Detection Seed Vertical Offset	-25
Detection ROI Expansion Rate	10
Detection ROI Delta Threshold	15
Detection ROI Maximum Size	100
Tracker Maximum Distance Threshold	100
Tracker Maximum Bounding Box Size	150

TABLE VI. BACKGROUND SEGMENTATION PARAMETERS

Sharpen Kernel	((1,1,1),(1,-8,1),(1,1,1))
Binarization Threshold Minimum	40
Binarization Threshold Maximum	255
Resampling Gaussian Blur Size	15 x 15
Resampling Gaussian Blur Sigma X	15
Resampling Binarization Threshold Type	Otsu Binarization
Edge Dilation Kernel Structuring Shape	Morph Ellipse
Edge Dilation Kernel Size	3
Contour Minimum Area Threshold	1000

IV. LIMITATIONS

The enormous set of variables, parameters, and environments (background, illumination, characteristics of tracked objects) make it nearly impossible to develop a universal system for object detection and tracking [2].

A. User-Interface Detection

Non-static UI elements and semi-transparent UI elements do not separate from the frame image as well. Modifying the minimum threshold value can provide better results in these circumstances, but at the cost of taking longer to remove noise from the data set.

The UI detection also has no way to recover from a complete screen change (e.g. fade-to-black, opening a pause menu, etc.).

B. Player Character Detection

In the scenario where the player character cannot be separated from the background elements of the frame, the detection system will provide an ROI that is too large, and may result in poor tracking. This behavior is typically corrected by the tracking system resetting.

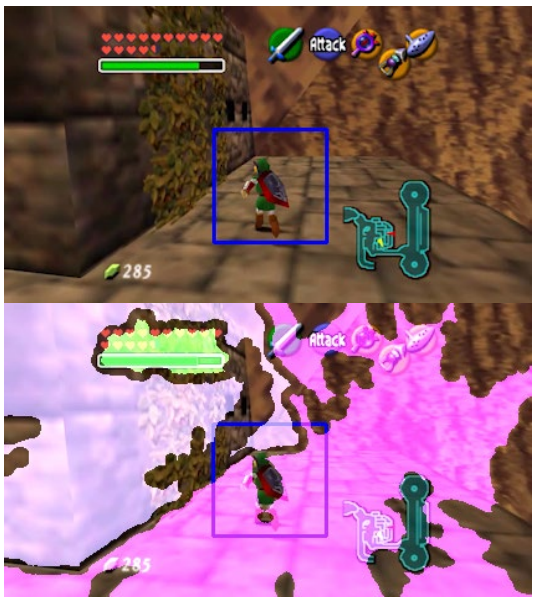
C. Player Character Tracking

Since the tracking system attempts to automatically detect a ROI, it is possible for the system to track a non-player entity. In most cases, this corrected after the entity leaves the tracking systems boundaries, but if the ROI captures a UI element this issue can persist over longer durations.

D. Environment Segmentation

Highly textured environments lacking sharp geometry definition and color contrast do not segment as well.

Fig. 7. Example of a poorly segmented background



REFERENCES

[1] A. Lukežic, T. Vojír, L. Zajc, J. Matas, M. Kristan, "Discriminative Correlation Filter with Channel and Spatial Reliability." *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, doi:10.1109/cvpr.2017.515.

[2] P. Janku, K. Koplik, T. Dulík, I. Szabo, "Comparison of Tracking Algorithms Implemented in OpenCV." *MATEC Web of Conferences*, vol. 76, 2016, p. 04031., doi:10.1051/matecconf/20167604031.

Sample game play footage used for this report is available here