AimBot using Template Matching for Object Detection

Final Project Report

Walid HADRI

Abstract

Artificial intelligence came to mimic the way human beings process problems in different areas, the applications have been increasing with the evolution of the discovered algorithms and techniques. One of the areas, where emulating the human intelligence using a computational ressource has been interesting is Video Games. The interaction between AI and Video Games, made the benefits for both sides; board games for instance (chess, Go and Atari) have always been linked since the emergence of the discipline as they provide a fertile ground to evaluate the different AI methods and algorithms developed. In the other hand, research in AI has mainly led to the improvement of video games, by creating Non-Player-Character and understanding more the user behaviour and experience.

The goal of this project is to focus on category of Video Games (FPS Games) and automate the main task of the players. The approach used is Object Detection based on template matching specifically and image processing broadly. I started from Object Detection on Images, to detection on videos and then to detection in real time. The project takes an original approach motivated by my interest in both the game and Computer Vision.

1. Introduction and Motivation

First-person shooter (FPS) is a sub-genre of shooter video games centered on gun and other weapon-based combat in a first-person perspective, with the player experiencing the action through the eyes of the protagonist and controlling the player character in a three-dimensional space. The genre shares common traits with other shooter games, and in turn falls under the action game genre. The gamer is expected to propel his avatar through the game by moving it forward, backward, sideways and so on using the game controller for PC player it would be the keyboard and the mouse. Forward movements of the controller result in the avatar moving forward through the scenery, usually with a slight left-right rocking motion to properly simulate the human gait. In order to increase the level of realism, many games include the sounds of breathing and footsteps in addition to the regular sound effects.

Figure 1. Modern FPS Game

Figure 2. Old FPS Game

The most popular and trending FPS Games are online multiplayer counter strike games with two teams competing to win. When it comes to counter strike video games, aiming at the enemies is the crux skill needed to be good at the game. These games are mostly played on computers, the keyboard is used to move and the mouse is used to aim. I am going to be developing more about aiming in the next section. But for now, aiming should be seen as placing the crosshair of the weapon on the target.

Making the process of aiming at the enemy automated has been always a way to cheat in such games, but the automation is mostly done by interacting with game and extracting the exact information about the location of the enemies on the screen, the same information used to display

them on the screen in first place. This method needs interacting with sockets comping up from the server and game developers make sure to make this interaction not possible, by restricting them and also defining banning rules regarding the use of any third party to communicate with the game so that cheating is not allowed.

The goal motivation behind the project is to develop ways based on Computer Vision to be able to detect enemies on the screen based on the same information a human player is receiving on the screen without any additional information. The detection should be enhanced so it could be performed in real time too, in different background with different frames per second rates. Creating an AI bot this way would be different from the classic cheating Bots, as what they have as inputs are the same inputs a human being receives on the screen when playing.

One of the most played counter strike video games nowadays is Valorant made by Riot Games. In this project, I will be considering this game that I know very well to make an AimBot based on Computer Vision.

2. Problem Definition

As introduced in the previous section, aiming is the main task that players perform and should be good at when playing FPS counter strike games, Valorant in particular.

2.1. The Crosshair

The crosshair is indicated as the center of the screen with some styled shape. As you can see in the figure **Indicate the figure**, the crosshair is point out with the cyan square. The crosshair is used for aiming at the targets.

Figure 3. Crosshair is the cyan square (Valorant Gameplay)

2.2. Aiming and targets

The process of aiming is moving the crosshair on the targets, in Valorant the main targets are the enemies or the bots if you are on the training session. Pro-players takes hours each day training their aim, by making the process of moving the crosshair on the targets more **accurate**, more **faster** and more natural by developing **muscle memory**.

Figure 4. Aiming at the enemies

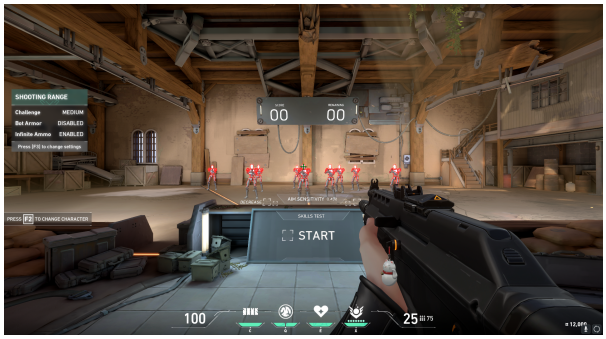

Figure 5. Aiming at the bots (used for practice and training)

However, one thing to keep in mind that aiming at different part of the body deals different damage for the same weapon. For Valorant, there are three parts of the target you can aim at:

- **The head** with the most damage that can be done
- **The body** with the second most damaged part
- **The legs** with the less damage that can be done

So, I will be interested in detecting the whole target in first and then detect the head afterwards. Then detect both of them at the same time.

So the problems the project is trying to address are:

- Target Detection: with constraints on the **accuracy** of the detection, on the **time** used to make the inference, the possibility to make the detection in real time and compete with the human gameplay.
- Make the detection on a reduced part of the screen using the help of a human player to guide the crosshair, and the goal of the AI Bot is to complete the task of

aiming on a smaller part of the screen to make the detection more faster and more accurate, this way the human player is being assisted at the hard part of the gameplay.

The main goal of the project is given an input image, we should make an algorithm that will give us the coordinates of the targets (center of the target, then we are supposed to get two coordinates (x,y)) in first place then afterwards the coordinates of the head. This algorithm should be accurate to some degree that I will define later and fast so that the inference could compete with the human gameplay.

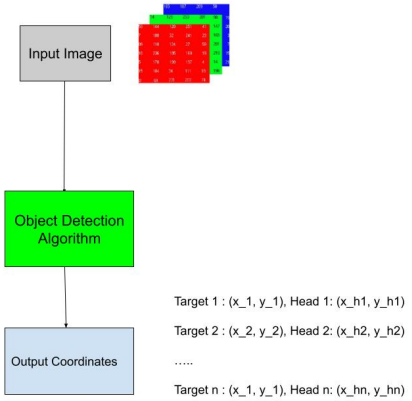

Figure 6. Formal definition of the problem

This problem falls under the umbrella of detecting the bounding boxes around the object that we are interested in. And we can formulate the problem as in the Figure[7]. In this project, we will be only interested in detecting the targets without the body parts.

3. Related Work

There are two ways to see the related work with this project, from the objective point of view of making an AI AimBot and from the technical one that is object detection on images:

3.1. From a goal point of view

As discussed in the introduction section, when it comes to making AI AimBot to automate the process of aiming in FPS Games, the most used AimBots are created based on the interaction between the server and a third party program.

The approach considered in this project does not rely on any interaction with the server or with the files of the game, we are going to having the same inputs a human player is having that are in this case the images displayed on the screen. We have different methods for comparison:

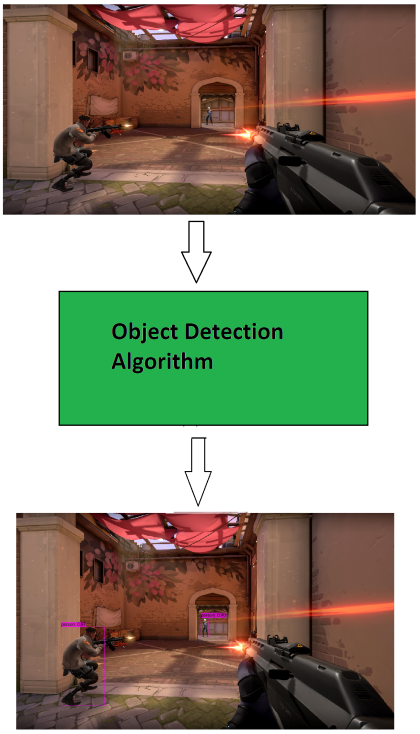

Figure 7. Illustrated formal definition by drawing the bounding boxes

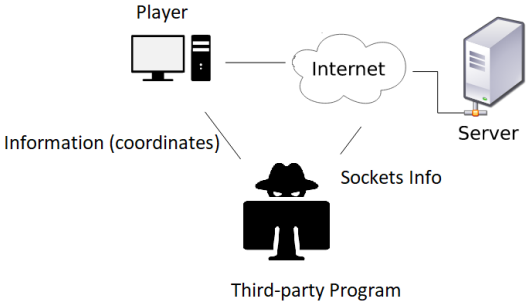

Figure 8. Third-Party program to get information the player does not have and the game is hiding

3.2. From a methodology point of view

Detecting objects on images or videos in one of the central problems in Computer Vision, even though the most advanced and accurate results are based on deep learning, using Image processing techniques like Template Matching gives pretty good results in some specific areas.

4. Methodology

4.1. Simple Template Matching

Template Matching is a method for searching and finding the location of a template image in a larger image. We simply slide the template imge over the input image (as in 2D convolution) and compares the template and patch of the input image under the template image in order to calculate some similarity score.

OpenCV comes with a function cv.matchTemplate() for this purpose. It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image. Several comparison methods are implemented in OpenCV. (You can check docs for more details). It returns a grayscale image, where each pixel denotes how much does the neighbourhood of that pixel match with template. [1]

- Template Matching Square Difference $R(x,y) = \sum_{x',y'} \left(T\left(x',y'\right) - I\left(x+x',y+y'\right)\right)^2$
- Template Matching Square Difference Normalized $R(x,y) = \frac{\sum_{x',y'}\left(T\left(x',y'\right)-I\left(x+x',y+y'\right)\right)^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$
- Template Matching Cross Correlation $R(x,y) = \sum_{x',y'} \left(T\left(x',y'\right) \cdot I\left(x+x',y+y'\right)\right)$
- Template Matching Cross Correlation Normalized $R(x,y) = \frac{\sum_{x',y'}\left(T\left(x',y'\right)\cdot I\left(x+x',y+y'\right)\right)}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$
- Matching Correlation Coefficient

$$R(x,y) = \sum_{x',y'} \left(T'\left(x',y'\right) \cdot I'\left(x+x',y+y'\right)\right)$$

where

$$\mathrm{T}'\left(x',y'\right) = T\left(x',y'\right) - 1/(w \cdot h) \cdot \sum_{x'',y''} T\left(x'',y''\right)$$
$$I'\left(x+x',y+y'\right) = I\left(x+x',y+y'\right) - 1/(w \cdot h) \cdot \sum_{x'',y''} I\left(x+x'',y+y''\right)$$

- Matching Correlation Coefficient Normalized $R(x,y) = \frac{\sum_{x',y'}\left(T'\left(x',y'\right)\cdot I'\left(x+x',y+y'\right)\right)}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$

4.1.1 Single Template Detection

Once we slide the template over the image, if we denote $(W \mathrm{x} H)$ the size of the input image and $(w \mathrm{x} h)$ the size of the template image, we get an output image of size $((W-w+1)\mathrm{x}(H-h+1))$. Based on the method used to make the comparison, we will have to find the global minimum and maximum supposing that we are looking for one single detection without multiple template matching. The extremums are searched across the whole array or, if mask is not an empty array, in the specified array region.

In this part, we are going to consider a given input and a given template extracted from the image (so we don't have the scaling or rotation issue yet)

Figure 9. The template of the Bot we are going to be detecting

Figure 10. The input image

I tried the 6 methods and compare them on this experiment. The first thing to notice is that using the Cross Correlation and the Correlation Coefficient (both without normalization), they don't give the results we are looking for. Yet, the others gave good results.

The fact that the Cross Correlation without Normalization picked the bounding box on a brighter part of the image, comes from the fact that when we don't normalize brighter patches have "advantage" over darker patches. So the normalization is the way to ensure we don't have this issue anymore. The normalization process helps with handling linear brightness variation.

To compare the time performance of the 6 methods, I ran the same experiment 100 times and I got the following results:

Figure 11. Using the Cross Correlation as a comparison method, the bounding box is the left bottom, obviously a bad result but on the bright patch of the image

Figure 12. Using the Normalized Correlation Coefficient as a comparison method

Method	Time (s)
CCOEFF	3.25
CCOEF NORMED	4.94
CCORR	2.22
CCORR NORMED	2.85
SQDIFF	2.79
SQDIFF NORMED	4.7

We notice that whenever we normalize we add more time, because there is the additional term in the denominator to calculate. The Cross Correlation seems to be the fastest one.

What we could take as a conclusion from this part is that the template matching algorithm we have till now is able to detect only one occurrence of the template and will return always one whether we have the template on the image or not and without giving us any insight about using the similarity score we calculated as a criterion to make the choice of returning an output or not.

4.1.2 Template Matching with multiple occurrences

The idea here is to keep track of the similarity scores calculated and either return for example the k-best ones (for k a given integer) or pick the outputs based on some given threshold. This way, we are able when we have multiple occurrences to detect them.

I implemented an algorithm that takes in the input image, the template image, the method, the criterion to choose the outputs whether based on a threshold or a number of outputs let's say the k best ones.

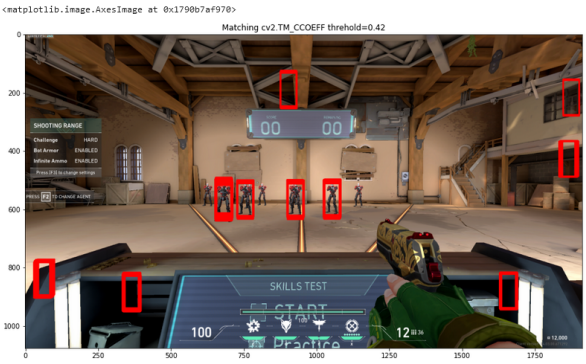

Figure 13. Normalized Correlation Coefficients with threshold 0.42

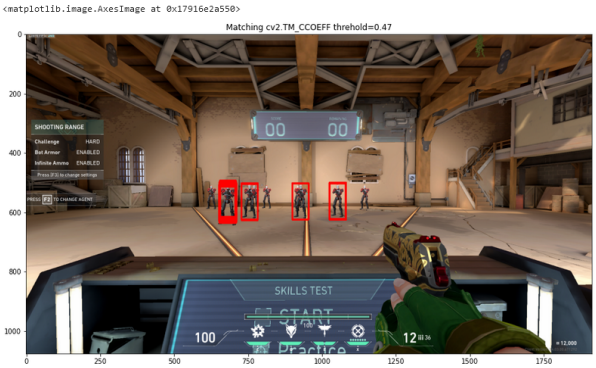

Figure 14. Normalized Correlation Coefficients with threshold 0.47

We can see that with a low threshold we are able to identify more occurrences even though they are not the exact template with the same exact size, and also other spots where we don't have the template. With a bit bigger threshold, we are able to detect some of the occurrences without having bounding box around false spots. I also show that for 200 best results, we have some bounding boxes that are overlapping. The next step is to suppress this overlapping issue with Non Maximum Suppression.

To do so, we will be using a notion called the Intersection

Figure 15. Normalized Correlation Coefficients taking the 200 best scores

over Union (IoU) metric, also referred to as the Jaccard index, which is essentially a method used usually to quantify the percent overlap between the ground truth BBox (Bounding Box) and the prediction BBox. However, in Non Maximum Suppression, we find IoU between two predictions BBoxes instead.

IOU(Box1, Box2) = Intersection_Size(Box1, Box2) / Union_Size(Box1, Box2)

The idea behind, the non maximum suppression is to loop through the bounding boxes we have in order that respects the score we have so we make sure to pick the best bounding box among the overlapping ones. We set also a threshold to decide whether two boxes are considered as overlapping or not.

Figure 16. Normalized Correlation Coefficients with threshold of 0.47 and with Non Maximum Suppression

Till here, this method we are using has issues detecting the template when it does exist on the image with some variations like the **scale**, **rotation**, the **brightness** and the **contrast**. We are going to address that in the next part. [2]

4.2. Scale Invariant Template Matching

As mentioned before, the template should have the same scale (supposing that the scale in the only variation for now), yet we want to detect the occurrences of the same template with different scales.

4.2.1 DownSampling and UpSampling with Gaussian Pyramids

We will be using the image pyramid that is is a collection of images - all arising from a single original image - that are successively downsampled or upsampled until some desired stopping point is reached.

There are two common kinds of image pyramids:

- **Gaussian pyramid:** Used to downsample images
- **Laplacian pyramid:** Used to reconstruct an upsampled image from an image lower in the pyramid (with less resolution)

For the Gaussian pyramid, it can be seen as a set of layers in which the higher the layer, the smaller the size.

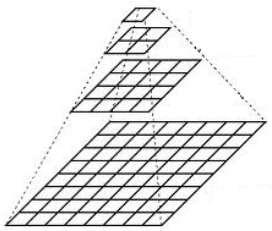

Figure 17. Gaussian Pyramid

Every layer is numbered from bottom to top, so layer $(i+1)$ (denoted as G_{i+1} is smaller than layer i (G_i).

To produce layer $(i+1)$ in the Gaussian pyramid, we convolve G_i with

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Remove every even-numbered row and column. The resulting image will be exactly one-quarter the area of its predecessor. Iterating this process on the input image ${}_G0$ (original image) produces the entire pyramid.

For Upsizing, upsize the image to twice the original in each dimension, with the new even rows (we fill columns with zeros), the we perform a convolution with the same kernel shown above (multiplied by 4) to approximate the values of the "missing pixels" (where we added the zeros).

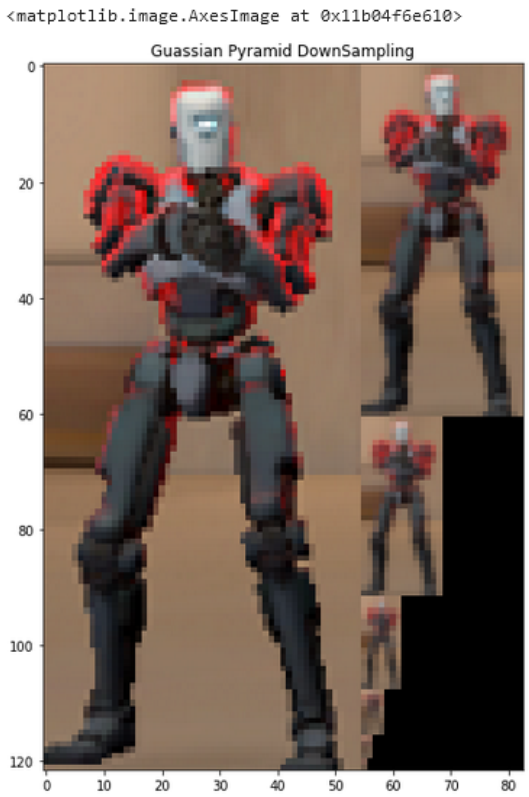

Figure 18. Guassian Pyramid applied to the template

I ran the previously implemented algorithm in the previous section that takes into account the non maximum suppression, and I iterated over the images in the pyramid. Th results shows that an additional occurrence was detected, yet not all of them, this is coming from the fact we downsample by one-quarter each time, yet this downsampling is way more faster than a simple resizing and also includes applying a filter.

Figure 19. Template Matching using Gaussian Pyramid Downsampled templates

4.2.2 More precise resizing

In this part, we make a more precise resizing, especially that with Pyramid we lose the size in no more than two 2 iterations. This is why the idea of making a more precise resizing would be very beneficial for the detection. The results confirm it. The experiment done is to resize the template with scales between 0.4 and 1 with 100 sample, for each resized template, perform the template matching implemented before, do non maximum suppression to get the final bounding boxes. Using this method we get to detect all the occurrences on the input image.

Figure 20. Template Matching with multi-scaling

4.3. SIFT

The idea behind Scale invariant feature transform is that keypoints of objects are first extracted from a set of reference images and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on some distance of their feature vectors.

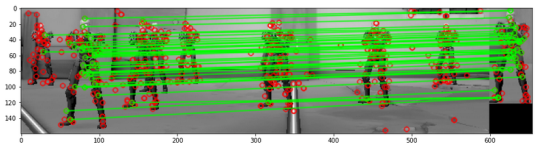

Figure 21. Using SIFT descriptors

4.4. HOG descriptors

HOG, or Histogram of Oriented Gradients, is a feature descriptor that is often used to extract features from image data. It is widely used in computer vision tasks for object detection.

Some important aspects of HOG that makes it different from other feature descriptors:

- The HOG descriptor focuses on the structure or the shape of an object. In the case of edge features, we only identify if the pixel is an edge or not. HOG is able to provide the edge direction as well. This is done by extracting the gradient and orientation (or you can say magnitude and direction) of the edges.

- The orientations are calculated in 'localized' portions. This means that the complete image is broken down into smaller regions and for each region, the gradients and orientation are calculated. We will discuss this in much more detail in the upcoming sections

- the HOG would generate a Histogram for each of these regions separately. The histograms are created using the gradients and orientations of the pixel values, hence the name 'Histogram of Oriented Gradients'

The HOG feature descriptor counts the occurrences of gradient orientation in localized portions of an image.

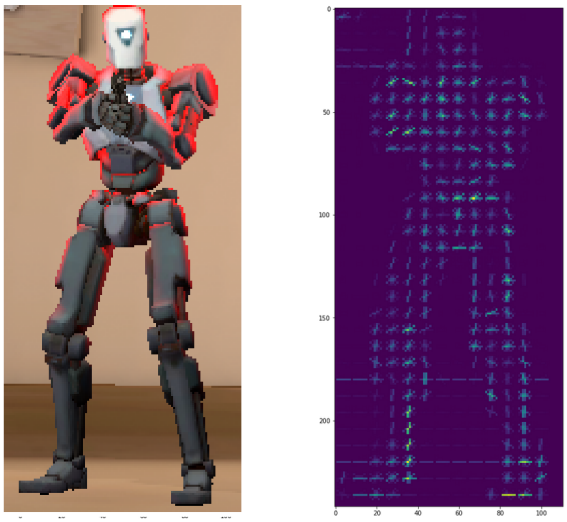

Figure 22. Using SIFT descriptors

The idea is to use the same brute force template matching algorithm, but now instead we are trying to find matches with respect to the HOG descriptor.

5. Evaluation

To evaluate the results we have different choices of scores, we will be using the IoU.

The Intersection Over Union (IoU) is used to evaluate the object detection algorithm. It is the overlap between the ground truth and the predicted bounding box, i.e it calculates how similar the predicted box is with respect to the ground truth.

Method	Score
Simple Template Matching threshold 0.42	0.36
Simple Template Matching threshold 0.47	0.64
Template Matching with Pyramid Scaling	0.74
Template Matching with precise scaling	0.99
SIFT	0.85
HOG	0.78

The fact that we the template Matching with the precise scaling outperforms the others comes from the fact that we compare the image and the template also in the color space, and we know that the template occur in the input image with the same appearance just with different scales.

6. Conclusion

Using techniques from image processing to create features in order to do object detection make us aware of what some layers learn when we are using deep learning. Some features descriptors are enough to perform the task of detecting the object, especially when it comes to template matching with a limited number of matches to find within some input images. However, the results given by deep learning outperform in large datasets, especially when it comes to generalizing and spending less time find the features ourselves. This is what has been showed in the two projects combined from the course Deep Learning and VIC on the same project of detecting the targets on given images.

7. References

1. Object Detection based on Fast Template Matching through Adaptive Partition Search `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7219760`

2. Grayscale Template-Matching Invariant to Rotation, Scale, Translation, Brightness and Contrast `http://www.lps.usp.br/hae/PSIVT07_Hae_Sidnei_Grayscale_Template_Matching_Invariant_RSTBC.pdf`

3. Distinctive Image Features from Scale-Invariant Keypoints`https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf`