Starcraft worker counting script

The motivation behind this project is to create some type of computer vision based script that can count and keep track of how many workers I have assigned to mine minerals. This is often beneficial in play due to having an adequate amount of workers to support a strong economy but not overly strong that army count and strength is compromised. It is not feasible to have some type of script that accesses game files and systems, because that would not be allowed formally. Also, to count and keep track of the workers myself manually would not be possible either.

Initially I wanted to use a somewhat higher level and complex machine vision based library such as tensor flow however a big obstacle is that typically that I know of, those machine vision models need training data or to have already been trained. Pre-trained models would work for conventional items that need detection such as animals and certain objects. In my particular case, since the minerals image and shape do not change, there is no need for any type of training. If training would be done it would be at max of 8 images, each for the corresponding base locations on the StarCraft fastest map. Another thing that the machine vision model would need to know is the image of the workers. Because of this certain and specific image that needs to be identified, the best use that was suggested from chatgpt was the use of template matching from the cv2 library.

The first step was to have the script and python, be able to use my desktop screen in a live environment to perform the object detection. Because this script needs to work while a game is running, a prerecorded video that is then analyzed is of no use. First screenshots were taken to capture the screen and then analyzed. This was done in a loop. Here the basics of the image detection algorithm was established. The image was converted to RGB from BGR and then had template matching for the minerals and workers. In the initial stages, the screen shot was captured and then resized then analyzed. This caused problems in the object detection due to the difference in sizes as the templates that were used were of full size. Thus, the live images need to be full sized as well.

From here, instead of the entire screen being captured and processed since, Starcraft only runs in 1080p with aspect ratio black bars kept on both sides of the screen, it would be a waste of space in input, processing and output to include the full resolution. Here image cropping was introduced to only obtain the center 1920 of the screen and processing would be done on this center sized portion fully to successfully implement template matching. Here, the rectangles were correctly drawn around the detected minerals. Video cropping was done by specifying the center width and height and then calculating the top left corner of the center region by subtracting from the total size of the screen.

Once it was clear that the live video object detection via template matching was working correctly, now, the workers were added to the items for object detection. This was simply just duplicating and specifying for the new objects.

The next step was to add coordinates and to keep track of the location of the minerals on screen as well as obtain mouse coordinates. This would be used later. Object counting was also added by adding the variables outside the loop as zero and to be updated within the inside of the loop if the objects are detected.

One of the first obstacles that was seen was that because we are still using screenshot based looping, the program was beginning to slow down due to having to perform object detection on three items as well as keep track of the locations of the onscreen minerals and location of the mouse. Due to all of this processing, the live video now was running slower than expected and caused the screen the blink black occasionally. Frame buffering as then implemented to help this run more smoothly. The logic behind this is to start with an initial frame capture and then enter the loop with a new frame. The old frame is then processed and shown, but while the old frame is being shown the new frame is being processed in the timeline ready to come out for viewing. At first I wanted to start with implementing a frame rate control with this double buffering technique, but found that StarCraft runs at more than 60fps. Since, StarCraft runs at unsteady frame rates, but much higher than what is the refresh of my screen it would not be possible to limit the game and double buffering to a set frame rate. Instead I just let he double buffering run without any frame rate control.

Now, flags were created to keep track of right lick events and if the mouse is in the green rectangle. Mouse in green is detected using the coordinates of the green rectangle and then comparing the coordinates of the mouse. This comparison is done inside the loop and if the conditions are met, working counts will then be increased based off the total amount of worker selected. The flags outside are triggered as true.

The first problem with this was that keeping track of the mouse right click button worked using pyautogui.mouseInfo. This would keep resulting in mouseinfo window popping up and preventing the rest of the program from continuing. It was not feasible to prevent the window from popping up and other solutions to use an alternative mouse down would instead just continuously click the right click button instead of recording when it would happen. To fix this problem, mouse input were then detected using pynput instead of the basic python mouse info. A function on click that detects if right clicks occur is written outside the loop and mouse listener is then initialized to listen for mouse right clicks.

As of here, working count is ready to be incremented by the appropriate amount of detected workers selected once the workers are right clicked onto the mineral patch thus sending them to mine. However, after a number of debugging and print statements, it is found out that when the cursor is over the minerals, object detection fails. Right click is detected but the mouse is never in the green because the green does not exist when the mouse is over it. This is due to Starcraft changing the way the cursor looks when it is hovering over minerals or gas. This object blocks the image recognition of the minerals.

The solution to this was to implement different thresholds for each object. Decreasing the threshold for the minerals allowed object detection to work regardless if the cursor on top of the minerals changed the way it looked for a moment. Lastly, the final fix was to adjust the coordinates of the pt\_candidate or the coordinates of the green box, which is taken as coordinates of the cropped area to be matched with the coordinates of the screen instead. Since the screen coordinates and the mouse are of the same origin.

The next issue came from where working counts would not be correctly updated. All flags and print functions work correctly as everything should however, assigned workers are not accounted for. The problem stems from the fact that this live video is still captured screenshots processed and output at a high rate. Sometimes it would require double or triple clicks to register the counted workers. Here was where the project shifted from image frame based to actual live video feed.

The live video was using cv2.VideoCapture. However, because this uses external cameras, the live video feed was just a black box. The solution to this was to use OBS virtual camera and set up the camera to record the screen. The output of the camera is then fed live to cv2. Once, this was complete, the next issue was the video window that popped up was extremely small. Attempts at resizing after image analyses and template matching proved to no success as the output video was the correct resolution but the image was extremely blurry and had a strange occurrence where only the right side of the screen was displayed.

Some of the trouble shooting processing was ensuring that it was not OBS as the problem. I checked with discord live streaming and it showed that OBS recorded as a virtual camera in native resolution. On top of this the video only showing the right side of the screen had to do with the cropping factor. However, after much testing it was showing that there was nothing wrong with these lines of code either and that the cropping was working as it was supposed to. It was discovered that cv2 regardless of input uses the default video as 640p. This small resolution scale caused the video to be extremely blurry when resized and also was the problem of the crop being only half of the screen. Once this was corrected things were back on track.

Next steps were to configure OBS to only capture and crop the middle section of the screen where Starcraft displays instead of the entirety of my screen since I have a wide screen monitor; there are black bars on the side of StarCraft. Fixing this and then realigning the green box that highlights the minerals to match with the new screen to recorded area coordinates of mouse was the solution. This resulted in the mouse coordinates to be subtracted by 320 pixels on the x value.

The next step is optimization now that everything is working as intended. The downside is that now we are processing live video in full 1080p resolution, this result in the program working faster than screenshot processing but it is still slow. Often times, because inputs to the game are very fast, the machine vision cannot keep up.

From here optimizations are being made. The first attempt was to utilize multi-threading. This would result in a thread for template matching which includes template matching, object counting, and updating variables related to those task including mouse detection. The second thread is the main thread which is the main program execution. This thread handles keyboard inputs to quit when needed and to display the video. However, this attempt seemed to not work and instead I went back to single thread.

The next step was to reduce the amount of looping that occurs inside the inner loop for drawing rectangles. Instead of drawing rectangles for every match, they are only drawn in intervals of 10. At this point, I realized I don’t need to draw rectangles around the workers either so that entire amount of code was removed. Overall this seemed to improve latency by a bit but not enough to be fully accurate in a live environment.

After this I wanted to see if the issue was the latency of cv2 Itself, and thus I opened a new python test file and created a simple video capture and display after a simple resize to 720p. The result was astonishingly quick and responsive, meaning that indeed all of the processing and template matching was slowing the script down considerably.

Using python time function; I recorded the time at certain intervals in the script. It was found that the time difference from entering the while loop and finishing it was nearly half a second. An intermediate time mark is the entering of the inner loop which is after template matching is complete. It was found that after entering the inner loop, it would finish in less than a hundredth of a second. Thus drawing rectangles are very light and fast. The time to enter the while loop, perform template matching, and then enter the inner loop before drawing any rectangles took over half a second.

Converting to a grayscale image and performing template matching reduced to time to around 200 milliseconds which converted to frames per seconds only equates to roughly 5. This is not fast enough still. Attempts at using time to limit the amount of template matching that occurs, still results in a 200 millisecond time. After further investigation, it is showing that the number of matches is over 100, which after a long time and realization, that number should only be equal to one since there is only one mineral patch and not over 100. SCVs are often counted as 1 or 0 or whatever the case is.

Next step in improvement was limiting the space that template matching occurs. This is implemented by created three zones, one for the minerals, which is the upper ¾ of the screen, while the multi scv takes up the entire bottom center box, and the single scv uses the left half of bottom center box. This resulted in a massive improvement of down to 70 to 80 milliseconds. Finally, converting the template matching from TM\_COEFF\_NORMED to TM\_SQDIFF\_NORMED resulted in another improvement to times of 40-60 milliseconds. Overall these are great improvements; however it is still not absolute bottom that can be done as the test file with no processing reached low times of 10 to 30 milliseconds.