

**yes**

Here is the fully integrated timing code snippet added into the main processing loop of your inria\_test\_v2\_DYNAMIC\_WITH\_VISUALIZATION.py script, showing exactly where to place it:

from time import time  
  
# ... initial setup code before loop ...  
  
print("Starting camera simulation...")  
camera\_fps = args.fps  
camera = SimCamera(points\_array=points\_array, image=args.source, fps=camera\_fps, skip\_pts=3)  
camera.start()  
print("Camera started!\n")  
  
print("="\*70)  
print("DYNAMIC TEMPLATE MATCHING STARTED")  
print("="\*70)  
print("Search region will PROPAGATE based on detected matches.")  
print("Cyan rectangle shows the active search region.")  
print("Press 'q' in the CV window to quit.\n")  
  
# Start measuring total processing time here  
tic = time()  
frame\_count = 0  
match\_count = 0  
last\_match\_global = None  
  
while True:  
 if camera.frame\_q.empty():  
 print(time(), " Frame Q Empty")  
 sleep(1/camera\_fps)  
  
 if not camera.running.empty() and camera.process.is\_alive():  
 continue  
 else:  
 print("\nCamera closed. Press any key in CV window to close.")  
 break  
  
 fid, frame\_x, frame\_y, crop = camera.frame\_q.get()  
 print(f"\n{'='\*70}")  
 print(f"FRAME {fid}")  
 print(f"{'='\*70}")  
  
 frame\_count += 1  
  
 # Apply noise and blur (if any)  
 crop = noise(crop, args.noise)  
 crop = blur(crop, args.blur, (args.blur\_filter, args.blur\_filter))  
  
 if args.resize != 100:  
 crop = cv2.resize(crop, new\_size)  
  
 # Load template and perform matching  
 data = data\_loader.load\_template(crop)  
 score = run\_one\_sample\_2(model, template=data['template'], image=data['image'])  
 scores = np.squeeze(np.array([score]), axis=1)  
  
 mb\_boxes, mb\_indices = nms\_multi(scores, np.array([data['template\_w']]), np.array([data['template\_h']]), [data['thresh']], multibox=True)  
  
 if len(mb\_indices) > 0:  
 match\_count += 1  
  
 local\_x, local\_y = mb\_boxes[^0][^0][^0], mb\_boxes[^0][^0][^1]  
 global\_x, global\_y = data\_loader.map\_to\_global\_coords(local\_x, local\_y)  
  
 print(f"\n{'='\*70}")  
 print(f"✓ MATCH FOUND! (Match #{match\_count})")  
 print(f"{'='\*70}")  
 print(f" Local coords (search region): ({local\_x}, {local\_y})")  
 print(f" Global coords (full image): ({global\_x}, {global\_y})")  
  
 last\_match\_global = (global\_x, global\_y)  
  
 # Dynamic INS update logic here...  
 if args.dynamic\_update:  
 new\_ins\_point = (int(global\_x), int(global\_y))  
 data\_loader.set\_ins\_point(new\_ins\_point, (args.search\_region\_width, args.search\_region\_height))  
 current\_ins\_point = new\_ins\_point  
  
 # Visualization and display code...  
  
 else:  
 print(f"\n✗ NO MATCH FOUND")  
 print(f" INS point unchanged: {current\_ins\_point}")  
 print(f" Search will continue in the same region (prevents drift)")  
 print(f"{'='\*70}")  
  
 # Display windows and handle keypress (exit on 'q')  
 cv2.imshow("result", cv2.resize(scaled\_image, (scaled\_image.shape[^1] // source\_image\_scale\_factor, scaled\_image.shape[^0] // source\_image\_scale\_factor)))  
 cv2.imshow("v-camera", crop)  
 if cv2.waitKey(1) == ord('q'):  
 break  
  
toc = time()  
total\_time = toc - tic  
  
print(f"\n{'='\*70}")  
print("PROCESSING COMPLETE - STATISTICS")  
print(f"Total processing time: {total\_time:.3f} seconds")  
print(f"Total frames processed: {frame\_count}")  
print(f"Total matches found: {match\_count}")  
print(f"Match rate: {match\_count/frame\_count\*100:.1f}%")  
if last\_match\_global:  
 print(f"Final match position: {last\_match\_global}")  
print(f"Search region size: {args.search\_region\_width}x{args.search\_region\_height}")  
print("="\*70)  
  
cv2.waitKey()  
cv2.destroyAllWindows()

**Summary of changes:**

* Placed tic = time() before the main processing while True: loop.
* Initialized frame\_count and match\_count counters before the loop.
* Incremented frame\_count within the loop every frame processed.
* Incremented match\_count whenever a match is found.
* After the loop (once exited), recorded toc = time(), calculated total processing time, and printed consolidated statistics.

This code will give you total runtime along with useful frame and match statistics integrated cleanly into your existing script flow. Let me know if you want it adapted for another specific part of your code or need help with finer-grained timing!

⁂