perplexity

yes

Perfect — here's precisely how that logic can be implemented in your setup:

When generating matching points (from the score map via NMS or similar methods), the idea is to make results path-consistent instead of just score-dominant. You can achieve this by picking up to **three strongest peaks** and then choosing the one that stays closest to your previously known trajectory point.

What happens step by step

1. **Generate candidate points**
   After each iteration, template matching or NMS gives you multiple possible detections (peaks on the score map).
   Each detection has a coordinate (x, y).
2. **Check if next point deviates**
   Compare the distance between the newly predicted point (highest score) and the previous valid path point.
   If this distance exceeds a set **threshold**, it means the next point is likely deviating from the intended trajectory.

   ```
   distance = np.sqrt((new_x - prev_x)**2 + (new_y - prev_y)**2)
   if distance > threshold:
       # The new point is off-path → check top 3
   ```

3. **Take top 3 peaks (highest scores)**
   When deviation is detected, get the top 3 peaks (highest confidence points) from your score map:

   ```
   peaks = np.argpartition(score_map.flatten(), -3)[-3:]  # indexes of top-3 scores
   top_points = [np.unravel_index(p, score_map.shape) for p in peaks]
   ```

4. **Evaluate closeness to the last known position**
   Compute the Euclidean distance from your previous point to each of these top three:

   ```
   distances = [np.sqrt((p[1]-prev_x)**2 + (p[0]-prev_y)**2) for p in top_points]
   nearest_index = np.argmin(distances)
   best_point = top_points[nearest_index]
   ```

5. **Select the best nearest point**
   Choose whichever of those three points lies **closest** to your previous trajectory and within

your threshold range. This ensures even if the top score slightly misses due to noise, your path continuity remains smooth.

What “point” means exactly

In your template matching context, a **point** means:

- The coordinate `(x, y)` on the **search image** representing the template’s detected center or corner.
- It represents one position candidate for where the template best matches the current region.

By filtering and refining these points, you ensure all detections follow a *continuous slope/path*, similar to how a camera or satellite would observe a stable trajectory over consecutive frames.

Would you like me to integrate this “top 3 peak selection + nearest point threshold check” algorithm directly into your matching code (e.g., after NMS)? I can modify your functions accordingly and share the updated executable Python file.