W skrócie:

Pipeline wykonuje align serii, buduje bazowy focus stack (Tenengrad argmax), segmentuje obiekty FastSAM, przypisuje segmentacje z kolejnych klatek do obiektów referencyjnych (IoU), dla każdego obiektu wybiera klatkę o najwyższej ostrości w obrębie jego maski oraz tworzy stabilną maskę finalną (union/corefill + clamp), po czym warstwowo wkleja najostrzejsze obiekty do wyniku, redukując artefakty ruchu.

Krok po kroku: jak działa pipeline

1) Wczytanie i ujednolicenie danych wejściowych

- Skrypt wczytuje obrazy z folderu (--input) wg wzorca (--pattern, domyślnie *.jpg).
- Jeśli rozmiary klatek są różne, **przeskalowuje do rozmiaru pierwszej**.

Po co: ujednolicenie rozdzielczości jest konieczne do alignu, segmentacji i stackowania.

2) Wyrównanie (alignment) serii względem pierwszej klatki

- Domyślnie uruchamia ECC (--align ecc), w trybie afinicznym (cv.MOTION_AFFINE).
- Dla przyspieszenia ECC liczy transformację na **pomniejszonych grayscale** (--ecc_scale), a potem skaluje translacje do pełnej rozdzielczości.
- Każda klatka jest „przeciągana" do układu referencji (warpAffine, WARP_INVERSE_MAP).

Po co: minimalizuje różnice wynikające z mikro-ruchów aparatu (drgania, minimalne przesunięcia), żeby segmentacja i „ostrość" były porównywalne.

3) Zbudowanie bazowego focus stacka (referencji)

- Liczona jest mapa ostrości **Tenengrad** (moduł gradientu Sobela) dla każdej klatki.
- argmax_focus_stack() wybiera **dla każdego piksela** klatkę z najwyższą ostrością i składa z tego obraz base_stack.png.

Po co: ten obraz jest „najostrzejszą" referencją globalną (ale może mieć artefakty ruchu). Służy jako:

- baza do segmentacji obiektów,
- wstępny final, który później jest „naprawiany” obiektami.

4) Segmentacja obiektów na obrazie referencyjnym (base stack)

- Model FastSAM (ultralytics.FastSAM) wykonuje segmentację na base_stack.
- Dla każdej maski zapisywane są: maska binarna, bbox, pole, score.

Następnie są 2 etapy stabilizacji:

1. **Filtracja** (filter_masks): odrzuca za małe obszary i „prawie cały kadr” (tło), robi NMS po bbox (--nms_iou), limituje liczbę masek.
2. **Scalanie podobnych masek** (merge_similar_masks): łączy maski, które wyglądają jak „ten sam obiekt pocięty na fragmenty” (blisko centroidu, podobne pole, podobny aspekt bbox, minimalny IoU bbox).

Wynik tej fazy to lista **referencyjnych obiektów** refs (RefObj), z nadanym ref_id.

Po co: potrzebujesz stabilnego zestawu „obiektów referencyjnych”, do których później przypisujesz maski z kolejnych klatek (czyli quasi-tracking).

5) Segmentacja każdej klatki + przypisanie masek do referencji (tracking przez dopasowanie)

Dla każdej klatki t:

1. FastSAM segmentuje klatkę,
2. maski przechodzą filtrację + scalanie jak wyżej,
3. następuje **przypisanie** maski z tej klatki do referencji (assign_masks_to_refs):
    a. najpierw bramka po bbox IoU (--ref_bbox_gate_iou),
    b. potem sprawdzany jest IoU masek (--ref_mask_iou_thr),
    c. pary sortowane malejąco po IoU i robione jest dopasowanie 1:1 (jedna maska nie może iść do kilku refów).

Każde przypisanie zapisuje maskę do folderu danego obiektu:

- objects/obj_XXX/frame_masks/tYYY.png,
  plus podglądy segmentacji (opcjonalnie).

Po co: to jest klucz logiki „każdy obiekt osobno” – zamiast robić globalny stack, budujesz przebieg (track) danego obiektu przez serię.

6) Budowa tracków obiektów

- tracks_from_assignments() tworzy dla każdego ref_id listę detekcji w czasie (TrackDet: czas, maska, bbox, area, score).

Po co: masz komplet: gdzie i jak obiekt występował w serii.

7) Zbudowanie maski finalnej obiektu (agregacja w czasie)

Są dwa tryby (--mask_mode):

A) Domyślny: union

1. **Union masek w czasie**: track_union_mask()
    a. OR-uje maski ze wszystkich detekcji danego obiektu,
    b. ignoruje podejrzane maski „łapiące tło" (gdy area jest zbyt duże: --union_skip_area_ratio),
    c. czyści morfologią (open/close).
2. **Wybór najlepszej klatki pod maskę union**:
    a. best_frame_for_union_mask() liczy ostrość Tenengrad **tylko we fragmencie** (union ∧ maska z danej klatki) i wybiera maksimum.
3. **Clamp (bezpiecznik)**:
    a. clamp_union_to_best() ogranicza union do okolicy maski z najlepszej klatki (dylatacja best-mask + AND).
    b. To zabezpiecza przed „rozlewaniem" union na obszary, gdzie obiekt w innych klatkach był przesunięty lub źle wycięty.

B) Opcjonalny: corefill

- Robi „głosowanie" po maskach: core (częste piksele) + fill (rzadsze, ale blisko anchor) + anchor (maska z najlepszej klatki),
- potem morfologia + wypełnianie dziur.

Po co: maska finalna obiektu ma być stabilna i odporna na chwilowe błędy segmentacji oraz ruch.

8) Klasyfikacja obiektów na tło i pierwszy plan

- Jeśli obiekt zajmuje duży procent kadru (area/(h*w) >= --bg_area_ratio), traktowany jest jako „BG”.
- Pozostałe to „FG”.
- Dodatkowo zapisuje się podsumowanie tracków (tracks_summary.txt).

Po co: kolejność wklejania warstw ma znaczenie – zwykle tło wkleja się inaczej i z innym feather niż obiekty na pierwszym planie.

9) Warstwowa kompozycja końcowa (naprawa base stacka)

- Start: final = base_stack.
- Najpierw wklejane są obiekty „BG”, potem „FG”.
- Wklejanie: hard_paste_then_feather():
    - robi twarde podstawienie pikseli src do base w masce,
    - następnie wygładza krawędzie maski rozmytym alpha (Gaussian blur) + zostawia „rdzeń” (erode) jako 100% opaque.

Wynik: final.png.

Po co: zamiast mieszać piksele wszędzie, robisz kompozycję obiektową: „weź ten obiekt z klatki, gdzie jest najostrzejszy, i wklej go do wyniku”.

Co skrypt zapisuje

- base_stack.png – bazowy stack ostrości (pixel-wise argmax).
- base/seg_base.jpg – segmentacja referencyjna na base stack (podgląd).
- frames/tXXX/seg_all.jpg, seg_assigned.jpg – podglądy segmentacji i przypisań dla klatki.
- objects/obj_XXX/ref_mask.png – maska referencyjna obiektu.
- objects/obj_XXX/frame_masks/tYYY.png – maski obiektu w każdej klatce.
- objects/obj_XXX/final_mask.png – finalna maska obiektu po agregacji.
- final.png – finalna kompozycja po „naprawie” base stacka.