

Chapter 6

Individual Project Practice Guide

Detection-Assisted Visual Tracking

From automatic detection to reliable tracking, recovery, and robot-relevant action

Griffith University

Lei Wang & Xuefei Yin

Student Practice Edition

How to use this chapter

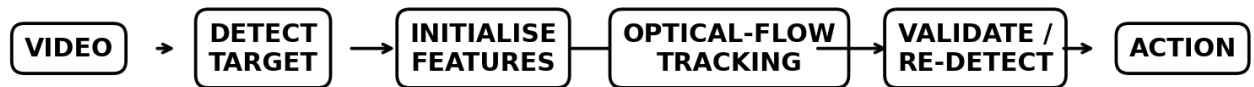
This chapter is a practical reasoning guide for building and testing the individual project. It is designed to help you connect the techniques from Workshops 1–5 into one coherent system. Use it to plan, implement, visualise, debug, evaluate and explain your own project. It is not a substitute for understanding your own code, video, parameter choices or results.

Important

Your project video, experimental results, design choices and analysis must be your own. Follow the assessment instructions, Griffith University academic-integrity requirements and the course rules for generative AI. Keep assessment materials private.

1. The project in one page

The project is an integration task. You are not being asked to invent a new detector or a new tracking algorithm. The engineering challenge is to connect several components so that the complete system can perceive a target, maintain a useful estimate, recognise when that estimate is no longer trustworthy, recover automatically, and produce a simple action.



Detection supplies semantic identity and a target region; tracking supplies efficient frame-to-frame motion.

- Detection answers: What object are we looking for, and where is it now?
- Feature initialisation answers: Which local image structures inside that target can be tracked?
- Optical flow answers: How did those image features move between consecutive frames?
- Validation answers: Do we have enough trustworthy evidence to use the tracking result?
- Re-detection answers: Can an independent detector recover the target when tracking becomes unreliable?
- Decision answers: Given the current target location and reliability state, what simple action should be produced?

Core insight

The most important architectural idea is not 'detector + tracker'. It is the separation of perception, validation, recovery and action. A numerical output is useful only when you have evidence that it is trustworthy.

2. How Workshops 1–5 come together

Workshop knowledge	Used in the project	Why it matters
Video / image handling	Read frames, inspect dimensions, write output video	A vision algorithm is part of a complete data pipeline.
Camera / image geometry	Interpret image coordinates and target centre	A pixel coordinate is an image-space quantity, not automatically a physical position.
Feature tracking	goodFeaturesToTrack + Lucas–Kanade optical flow	Provides efficient frame-to-frame target motion.
Segmentation / masking ideas	Restrict feature search to a target region	Prevents unnecessary or misleading background features.
Object detection	Faster R-CNN, class/confidence filtering	Provides semantic target identity and recovery.

3. Part 1 — Record and inspect the video

Treat the video as an experimental input, not merely a file that happens to exist. A poor experimental scene can make a good algorithm appear to fail.

3.1 Minimum inspection code

```
import cv2

VIDEO = "data/my_video.mp4"

cap = cv2.VideoCapture(VIDEO)

if not cap.isOpened():
    raise FileNotFoundError(f"Cannot open {VIDEO}")

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)
frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

print("width:", width)
print("height:", height)
print("fps:", fps)
print("frames:", frame_count)

ok, first_frame = cap.read()
if not ok:
    cap.release()
    raise RuntimeError("Cannot read first frame")

cv2.imwrite("results/first_frame.png", first_frame)
cap.release()
```

- Check that the target is clearly visible near the beginning.
- Check that the target has corners, texture or other local structure for feature tracking.
- Plan movement through left, centre and right image regions.
- Deliberately include at least two challenging conditions, such as temporary occlusion, moderate scale change, faster motion, clutter or partial disappearance.
- Keep the target identity unambiguous; normally use one instance of the selected target class.

Experimental-design insight

If the video does not create the conditions required to test recovery, no amount of code tuning can demonstrate automatic re-detection. Design the experiment first; then tune the algorithm.

4. Part 2 — Automatic detection and feature initialisation

This is the key bridge between the Week 5 detector and the Week 3 tracker. The detector provides a semantic target region. The feature tracker then works inside that region.

1. Run the pretrained detector on an appropriate initial frame.

2. Filter detections to the selected target class.
3. Apply the confidence threshold.
4. Select a reliable target using an explicit policy, such as the highest-confidence valid candidate.
5. Use the selected bounding box as the feature-search region.
6. Create a mask and call `cv2.goodFeaturesToTrack()`.
7. Visualise the box and feature points before moving to the full video.

4.1 The important conceptual code

```
detections = detect(model, frame)

candidates = []
for box, label, score in detections:
    if label != TARGET_CLASS:
        continue
    if score < CONF_THRESHOLD:
        continue
    candidates.append((score, box))

target_box = None if not candidates else max(
    candidates, key=lambda item: item[0]
)[1]

points = initialise_features(frame, target_box)
```

Do not hide target selection

If there are several detections, make the selection policy explicit. A detector can return several valid candidates; your application still needs a rule for deciding which one is the project target.

5. Part 3 — Optical-flow tracking

Once features are initialised, the detector does not need to run on every frame. Lucas–Kanade optical flow estimates where the selected local features move from one frame to the next.

```
p1, status, error = cv2.calcOpticalFlowPyrLK(
    previous_gray,
    current_gray,
    points,
    None,
    winSize=LK_WIN_SIZE,
    maxLevel=LK_MAX_LEVEL,
    criteria=LK_CRITERIA,
)
```

5.1 The three outputs you must understand

Output	Meaning	Engineering use
<code>p1</code>	Estimated new locations of the input points	Candidate point motion.
<code>status</code>	Whether OpenCV considers a point successfully tracked	Basic validity filter.

error

Tracking error measure returned
by the method

Additional reliability filter.

5.2 A practical reliability filter

```
status_ok = status.ravel() == 1
error_ok = error.ravel() < MAX_TRACK_ERROR

valid_mask = status_ok & error_ok

good_old = old_points[valid_mask]
good_new = new_points[valid_mask]
```

Important implementation detail: array shape

OpenCV commonly returns point arrays with shape (N, 1, 2). Before doing arithmetic, medians or drawing operations, convert them to (N, 2) explicitly. This avoids a common error such as 'only size-1 arrays can be converted to Python scalars'.

```
good_old_xy = good_old.reshape(-1, 2)
good_new_xy = good_new.reshape(-1, 2)

for old, new in zip(good_old_xy, good_new_xy):
    ox, oy = np.round(old).astype(int)
    nx, ny = np.round(new).astype(int)
```

5.3 Median target centre and displacement

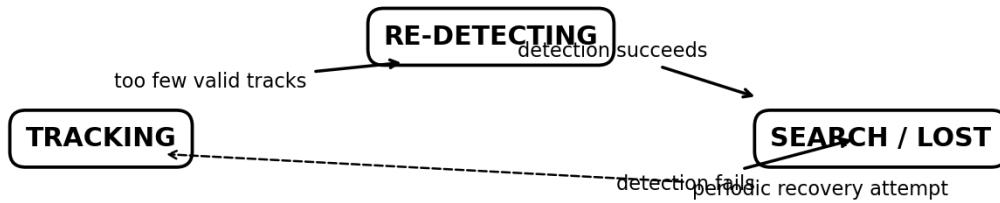
```
centre = np.median(good_new_xy, axis=0)
```

```
displacement = good_new_xy - good_old_xy
dx, dy = np.median(displacement, axis=0)
```

The median is useful because a small number of abnormal point motions should not dominate the estimate. The resulting centre is an image-plane coordinate in pixels. Do not describe it as a metric 3D position.

6. Part 4 — Validate tracking and automatically re-detect

This is the central reliability idea of the project. Tracking is an estimate. Validation determines whether that estimate has enough supporting evidence. Re-detection provides an independent recovery mechanism.



Important: state determines whether a target centre is trustworthy and what action is allowed.

6.1 State-machine mental model

- TRACKING: enough valid feature evidence exists; use the estimated centre.
- RE-DETECTING: tracking has become unreliable; run the detector again.
- SEARCH / LOST: no reliable target is currently available; do not use stale target coordinates as if they were current.

6.2 Core recovery logic

```

if valid_count < MIN_VALID_TRACKS:
    state = "RE-DETECTING"

    # Run detector again.
    target_box = detect_target(...)

    if target_box is not None:
        points = initialise_features(frame, target_box)
        state = "TRACKING"
    else:
        state = "SEARCH/LOST"
  
```

Critical safety rule

When tracking becomes unreliable, do not continue to issue normal movement commands using the last known centre. That coordinate may be stale. The project deliberately requires recovery or SEARCH/LOST behaviour.

6.3 What should be measured

- Total automatic re-detection attempts.
- Number of successful re-detections.
- Frames or intervals where the main recovery events occurred.
- The trigger for each recovery, such as too few valid tracks.
- Whether the recovered target produced enough new features to resume tracking.

7. Part 5 — Robot-relevant action

```

def choose_action(target_is_reliable, cx, width):
    if not target_is_reliable:
  
```

```

    return "SEARCH"
if cx < width / 3:
    return "TURN LEFT"
if cx > 2 * width / 3:
    return "TURN RIGHT"
return "MOVE FORWARD"

```

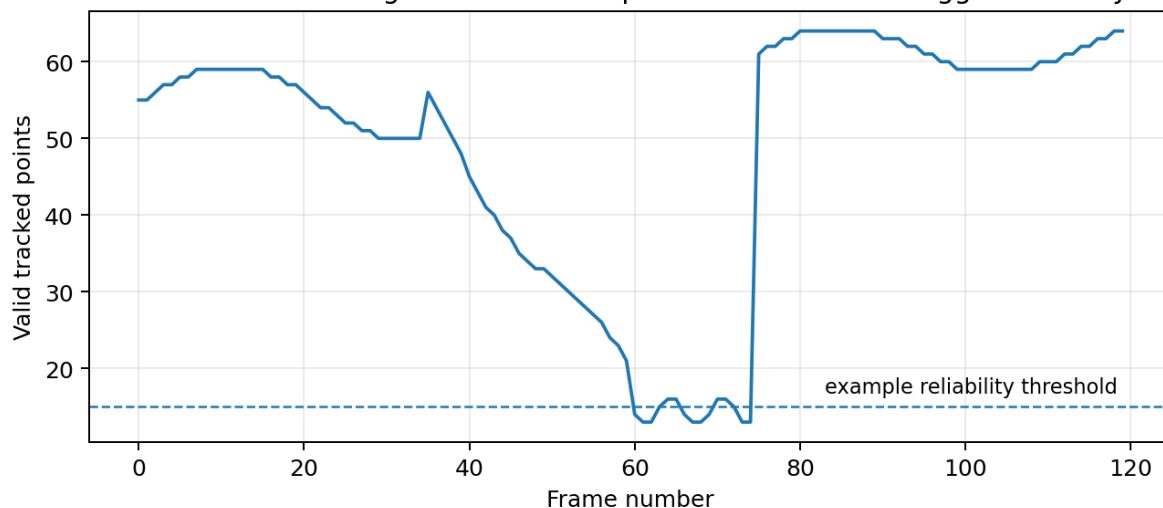
Target centre	Region	Action
$cx < W/3$	Left	TURN LEFT
$W/3 \leq cx \leq 2W/3$	Centre	MOVE FORWARD
$cx > 2W/3$	Right	TURN RIGHT
Target unreliable	Unknown	SEARCH

Perception-to-action boundary

Perception should answer 'Where is the target, and how reliable is that estimate?' The decision layer then maps that information to the required action. Keeping these responsibilities separate makes the system easier to test and explain.

8. Part 6 — Evaluation: prove what happened

Illustrative tracking evidence: a drop in valid tracks can trigger recovery



Your required tracking plot should show frame number on the x-axis and the number of valid tracked feature points on the y-axis. Use 0 when no valid tracked feature points are available. The purpose is not to make a beautiful plot; it is to provide evidence of when the tracker became weak and how recovery behaved.

```

frames = [row["frame"] for row in logs]
track_counts = [row["valid_tracks"] for row in logs]

plt.figure(figsize=(9, 4))
plt.plot(frames, track_counts)

```

```
plt.xlabel("Frame number")
plt.ylabel("Number of valid tracked points")
plt.title("Valid tracked feature points over time")
plt.tight_layout()
plt.savefig("results/valid_tracks_vs_frame.png", dpi=180)
```

8.1 Evidence checklist

- One representative successful tracking frame.
- Representative frames for the challenging conditions.
- A frame showing automatic re-detection if possible.
- A frame showing SEARCH/LOST if it occurs.
- The valid-track-count plot.
- Total re-detection attempts and successful recoveries.
- A short technical explanation of the main failure and recovery sequence.

9. Output video: make the result genuinely verifiable

A common end-to-end engineering failure is to produce an annotated image but not a valid multi-frame video. The output writer is part of the system and must be tested explicitly.

```
fourcc = cv2.VideoWriter_fourcc(*"mp4v")

writer = cv2.VideoWriter(
    str(output_path),
    fourcc,
    fps,
    (width, height)
)

# For every input frame:
writer.write(annotated_frame)

writer.release()
```

9.1 Verify the saved MP4

```
check = cv2.VideoCapture(str(output_path))

written_frames = int(
    check.get(cv2.CAP_PROP_FRAME_COUNT)
)

check.release()

print("Frames written:", written_frames)
print("Expected:", total_frames)
```

- The writer frame size should match the input frame size.
- Write one frame for each processed input frame.
- Release the writer before trying to inspect the final file.
- Do not assume that a filename ending in .mp4 proves that a valid multi-frame video was produced.
- For a recorded demonstration, make annotation text large enough to read clearly in the final video.

Recent implementation lesson

If the program crashes after writing the first frame, the resulting file may exist but contain only one frame. Always fix the first runtime error before judging the output video. Then verify the actual frame count.

10. Visualisation is your debugging instrument

Stage	What to visualise	Question to ask
Input	Representative frames	Is the target visible and suitable?
Detection	Box + class + confidence	Did the detector find the intended object?
Initialisation	Box + feature points	Are points inside the target and spatially useful?
Tracking	Point trajectories	Do points move with the target rather than the background?
Validation	Track count + state	When does the evidence become weak?
Re-detection	New detector box	Did recovery find the correct target?
Action	Centre + action + state	Does the action match both location and reliability?

11. Parameter tuning: experiment, do not guess

Parameter	If too permissive	If too selective
Detector confidence threshold	More weak/false detections may enter.	Weak true detections may be rejected.
maxCorners	May produce fewer useful points.	More points, but potentially more ambiguous/background points.
qualityLevel	Weaker corners may be accepted.	Only stronger corners survive; count may fall.
minDistance	Points can cluster.	Points spread out; fewer may remain.
Minimum valid tracks	Tracker may be trusted with little	Re-detection may occur more

evidence.

often.

Maximum tracking error

Noisier tracks may survive.

More tracks are rejected; recovery may be more frequent.

8. Choose one parameter.
9. Write down your prediction.
10. Change only that parameter.
11. Run the same video segment.
12. Inspect both the visual result and the numerical log.
13. Record what changed and decide whether the change improved the complete sequence.

12. Debugging: find the first wrong stage

Symptom	Inspect first	Likely issue
No initial target	Raw detector output	Wrong class, threshold, unsuitable frame, detector limitation.
Correct box but no features	Feature visualisation	Low texture, incorrect mask, strict feature parameters.
Features move to background	Mask + trajectories	Background points, weak texture, target-region drift.
Track count collapses	Status/error + challenging frame	Occlusion, blur, fast motion, scale change.
Re-detection never happens	Reliability criterion	Criterion never reached or state transition is wrong.
Re-detection happens constantly	Track threshold/error	Criterion too strict or features are poorly initialised.
Action is wrong	Centre + state	Coordinate interpretation or stale information.
MP4 contains one/few frames	Runtime + writer frame count	Program crashed early, writer not released, codec/path/frame-size problem.

Debugging rule

Ask: 'What is the first stage whose output is inconsistent with what I expect?' Fix or understand that stage before changing later stages. Do not change five parameters at once.

13. Clean software architecture

You do not have to use exactly this file structure, but separate responsibilities so that each component can be tested independently.

```
my_project/
|
|-- data/
|   |-- my_video.mp4
|
|-- src/
|   |-- detector.py
|   |-- tracking.py
|   |-- decision.py
|   |-- main.py
|
|-- results/
|   |-- representative_frames/
|   |-- valid_tracks_vs_frame.png
|   |-- annotated_output.mp4
|
|-- README.txt
|-- report.docx
```

- Use reasonable relative paths rather than machine-specific absolute paths.
- Make the main entry point obvious, for example main.py or a clearly named main() function.
- Keep input data separate from generated results.
- Do not submit environment folders, downloaded model weights or unnecessary temporary files.
- Make the README explain packages and how to run the program.

14. A practical implementation order

14. Confirm the video and project file structure.
15. Run Part 1 only and verify metadata and representative frames.
16. Run the detector on the initial frame and inspect the raw detections.
17. Implement target-class and confidence filtering.
18. Select the target and visualise the selected box.
19. Initialise features inside the selected box and visualise them.
20. Run optical flow on a short segment and visualise trajectories.
21. Add status/error filtering and valid-track counting.
22. Add median target-centre estimation and image-plane displacement.
23. Add the reliability criterion and test it on a challenging segment.
24. Add automatic re-detection.
25. Add SEARCH/LOST handling.
26. Add the required action rule.
27. Write and verify the multi-frame annotated MP4.
28. Add logging and generate the required plot.
29. Only after the pipeline works, clean the code and prepare the report.

Do not build everything at once

If the complete loop is implemented before the detector, feature initialisation and optical flow have been tested separately, there are too many possible sources of failure. Build the pipeline incrementally.

15. Designing a strong demonstration video

A strong demo makes the system's state transitions visible. You are not trying to make every frame perfect; you are trying to demonstrate that the system behaves sensibly when perception is good and when perception becomes difficult.

30. Start with a clearly visible target so automatic detection can initialise.
31. Move the target or camera so the target appears on the left.
32. Move through the centre.
33. Move to the right.
34. Introduce a first challenge, such as partial occlusion or moderate scale change.
35. Allow tracking to degrade enough to trigger automatic re-detection.
36. Show successful recovery and continued tracking.
37. Introduce a second challenge, such as faster motion or partial disappearance.
38. If the target becomes unavailable, demonstrate SEARCH/LOST rather than stale movement.

Demo insight

The most convincing final demonstration is not 'look, it tracks'. It is 'look, it tracks while reliable; when evidence becomes weak, it recognises the problem, attempts recovery, and changes behaviour appropriately'.

16. Report planning guide

Report part	Explain	Useful evidence
Part 1	Scenario and why the target/video are suitable.	Video metadata + normal/challenging frames.
Part 2	Detector, class filtering, threshold, automatic feature initialisation.	Detection box + confidence + initial features.
Part 3	Optical flow, filtering, centre and displacement.	Tracking frames + track counts + parameter settings.
Part 4	Reliability criterion, state transitions and recovery.	State examples + re-detection counts + recovery evidence.

Part 5	Image-space decision rule and unreliable-target handling.	Representative action frames.
Part 6	What happened, why, failure mode, improvement and safety risk.	Track-count plot + statistics + representative frames.

17. Write analysis, not just description

Weak description	Stronger engineering analysis
The tracker failed when the object was hidden.	During the occlusion, valid tracks decreased from approximately ___ to ___. This crossed the reliability threshold of ___, triggering automatic re-detection.
The detector worked well.	At the selected threshold of ___, the detector identified the target in the initial frame and recovered it after ___ main tracking failures.
We used the median because it is good.	Median displacement reduces the influence of a small number of abnormal point motions, making the estimate less sensitive to individual tracking errors.
The action is TURN RIGHT.	The reliable target centre was $c_x = \text{___}$ in an image of width ___, placing it in the right region; therefore the specified rule produced TURN RIGHT.

18. Mini experiments before submission

39. Detector threshold: test three thresholds on the same frame and record accepted detections.
40. Feature quality: change qualityLevel or minDistance and inspect feature distribution.
41. Reliability threshold: compare a slightly lower and higher minimum-track threshold.
42. Failure recovery: identify the most challenging interval and record valid tracks before, during and after it.
43. Action correctness: choose left, centre and right frames and verify c_x and the displayed action.
44. Output verification: confirm the final MP4 contains the expected number of frames and can be played independently.

19. Responsible AI use

AI can help with implementation, debugging and explanation, but your own project decisions, video, measurements and analysis must remain your own. Use AI as an engineering assistant rather than an answer generator.

Goal	Useful prompt pattern
Understand a function	Explain <code>cv2.calcOpticalFlowPyrLK()</code> in plain English. Explain each input/output and assumptions.
Review code	Review this tracking function for logical errors. Do not rewrite it yet; identify likely problems and tests.
Debug an error	I receive a shape/unpacking error from OpenCV point arrays. Explain the array shapes and show the smallest safe fix.
Test design	Suggest controlled video conditions that would stress sparse optical-flow tracking and explain what each reveals.
Explain results	Here are my measured track counts and re-detection events. Help organise the observations into technical analysis without inventing results.

Five-question AI check

1) Does the generated code match my algorithm? 2) Are the coordinate conventions and array shapes correct? 3) What assumptions does it make? 4) What test would reveal an error? 5) Can I explain every important line I submit?

20. Final self-check

- ☐ I can explain why my selected target is suitable for feature tracking.
- ☐ My video includes left, centre and right target positions.
- ☐ My video contains at least two challenging conditions.
- ☐ Initial target detection is automatic.
- ☐ I can explain and justify my confidence threshold.
- ☐ Feature points are initialised inside the detected target box.
- ☐ I can explain the roles of `p0/p1`, `status` and `error`.
- ☐ I reject unreliable tracks rather than using them blindly.
- ☐ I estimate the target centre from valid tracked points.
- ☐ I treat the centre as an image-space coordinate.
- ☐ My reliability criterion is explicit and justified.
- ☐ Automatic re-detection occurs without manual re-selection.
- ☐ My video triggers at least one automatic re-detection.
- ☐ TRACKING, RE-DETECTING and SEARCH/LOST are visible/implemented.
- ☐ I count re-detection attempts and successful recoveries.

- ☐ An unreliable target cannot produce a normal movement command from stale coordinates.
- ☐ The final annotated output is a genuine multi-frame MP4.
- ☐ I verified the written video frame count.
- ☐ I generated the valid-track-count plot.
- ☐ My report analyses failure and recovery, not only successful frames.
- ☐ I can explain the important code and parameter choices I submitted.

21. Final engineering perspective

The deepest lesson of this project is system integration. A detector can be accurate but expensive. A tracker can be efficient but can drift or fail. A reliability check can identify weak evidence. A detector can then provide an independent recovery mechanism. Finally, the decision layer must respect the reliability state rather than treating every numerical estimate as truth.

Remember

A successful robotics vision system is not simply one that produces a number on every frame. It is a system that knows what it is estimating, measures whether the evidence is adequate, recovers when necessary, and behaves safely when it cannot see reliably.

Student Quick Reference

This section is a compact project roadmap. Use it while coding. It answers the practical questions students most often have: what each stage must produce, what to check before moving on, how the states differ, how to choose parameters, and how to prove that the final system works.

The project in one sentence

Detect the target automatically → initialise features inside it → track those features → decide whether tracking is reliable → re-detect when necessary → use the target position only when reliable → otherwise SEARCH.

A. What must be true at each stage?

Stage	Input	Output	Before moving on, check...
1. Video	Your recorded MP4	Frames + width/height/FPS/count	Target is visible, moves left/centre/right, and includes ≥2 challenges.
2. Detection	Initial frame	Target class + box + confidence	Correct class, sensible box, threshold recorded.
3. Features	Frame + target box	Points inside target	Enough points and useful spatial distribution.
4. Tracking	Previous/current frame + points	New points + status + error	Invalid/noisy points are rejected.
5. Validation	Valid points	Reliable/unreliable decision	Criterion is explicit and tested.
6. Recovery	Current frame	New box or failure	No manual ROI; attempts and successes counted.
7. Action	State + cx + width	LEFT/FORWARD/RIGHT/SEARCH	Unreliable target can only produce SEARCH.
8. Evaluation	Logs + output video	Plot + evidence + analysis	MP4 is multi-frame and results are measurable.

B. The three perception states — clear distinction

State	Meaning	Detector?	Use target centre for movement?
TRACKING	Current feature evidence is reliable.	Normally no.	Yes.
RE-DETECTING	Tracking has failed the reliability test; recovery is being attempted.	Yes.	No.
SEARCH / LOST	No reliable target is currently available.	May be attempted again depending on your design.	No — use SEARCH.

Easy way to remember

RE-DETECTING = an action the system is taking. SEARCH/LOST = the situation after no reliable target is available. If detection succeeds, initialise new features and return to TRACKING. If it fails, remain in SEARCH/LOST.

C. The most important tracking code

```
p1, status, error = cv2.calcOpticalFlowPyrLK(
    previous_gray, current_gray, points, None,
    winSize=LK_WIN_SIZE,
    maxLevel=LK_MAX_LEVEL,
    criteria=LK_CRITERIA
)
```

```

status_ok = status.ravel() == 1
error_ok = error.ravel() < MAX_TRACK_ERROR
valid = status_ok & error_ok

old_xy = points.reshape(-1, 2)
new_xy = p1.reshape(-1, 2)

good_old = old_xy[valid]
good_new = new_xy[valid]

```

- p1 = estimated new locations.
- status = basic success/failure flag.
- error = additional quality signal.
- valid = the points you are willing to trust.
- reshape(-1, 2) = makes OpenCV point arrays easier and safer to process.
- Do not calculate the target centre from rejected points.

D. Robust target centre

```

target_centre = np.median(good_new, axis=0)
cx, cy = target_centre

```

Use the median because a few incorrectly tracked points should not dominate the estimate. Remember: cx and cy are image coordinates in pixels. They are not a physical 3D position.

E. Reliability → recovery → action

```

if valid_count < MIN_VALID_TRACKS:
    state = "RE-DETECTING"

    new_box = detect_target(frame)

    if new_box is not None:
        points = initialise_features(frame, new_box)
        state = "TRACKING"
    else:
        state = "SEARCH/LOST"

if state != "TRACKING":
    action = "SEARCH"
else:
    action = choose_action(cx, width)

```

Key safety idea

Never let the last known centre silently continue controlling the robot-like action after tracking has become unreliable. The state must control whether the centre is allowed to influence the action.

F. Parameter choices — what you need to justify

Parameter	What it controls	What to report
-----------	------------------	----------------

CONF_THRESHOLD	Which detector predictions are accepted.	Value + brief reason based on your video.
maxCorners	Maximum number of feature points.	Value + whether enough useful points are produced.
qualityLevel	Strength required for a feature.	Value + effect on point quality/count.
minDistance	Spacing between features.	Value + whether points are well distributed.
MAX_TRACK_ERROR	Which optical-flow results are rejected.	Value + effect on noisy/valid tracks.
MIN_VALID_TRACKS	When tracking is considered unreliable.	Value + evidence that it triggers at an appropriate time.

Do not claim that a parameter is 'optimal' unless your experiments support that claim. A better explanation is: 'I tested ___ and ___ on the same video segment and selected ___ because it gave enough reliable features while avoiding frequent false recovery.'

G. Video resolution — no need to chase 640×360

There is no requirement to reduce the video to 640×360 for marks. If the system works reliably at the recorded resolution, you may keep it. Lower resolution can improve speed, but it can also reduce target detail and useful feature points. Choose a resolution that gives a reliable, practical system and explain the choice if relevant.

H. What makes the final demonstration convincing?

- Start with clear automatic detection.
- Show the target moving through left, centre and right regions.
- Show normal tracking before the difficult event.
- Show at least two challenging conditions.
- Show valid-track count falling or otherwise becoming unreliable.
- Show RE-DETECTING when recovery is attempted.
- Show successful recovery and return to TRACKING, if your video permits it.
- If the target cannot be recovered, show SEARCH/LOST rather than stale movement.
- Show the action text changing consistently with the target position and reliability state.

Best demonstration story

Do not aim only for 'perfect tracking'. A stronger engineering demonstration is: reliable perception → difficult condition → reliability drops → recovery attempt → successful recovery or SEARCH/LOST → safe action.

I. Minimum evidence to save

Evidence	Why it matters
Video metadata	Shows that the input was inspected.
Initial detection frame	Shows automatic target selection.
Initial feature frame	Shows correct feature initialisation.
Normal tracking frame	Shows the tracker working.
Challenge frame 1	Shows behaviour under difficulty.
Challenge frame 2	Shows another difficulty.
Re-detection frame	Shows automatic recovery.
SEARCH/LOST frame, if reached	Shows correct behaviour when recovery fails.
Valid-track plot	Provides numerical evidence of tracking quality.
Final MP4 frame-count check	Shows that the output is genuinely multi-frame.

J. Report: answer 'what, how, why, evidence'

For each part, answer...	Example of the level of detail
What did you implement?	Automatic detection followed by masked feature initialisation.
How does it work?	Filter target-class detections by confidence, select the highest-

	confidence candidate, then detect features inside its box.
Why did you choose it?	The detector provides semantic identity; local features are cheaper to track frame-to-frame.
What evidence supports it?	Show the detection box, feature points, track-count plot and recovery frames.
What happened when it failed?	State the measured track-count change, trigger, recovery result and main limitation.

Avoid a common report mistake

Do not simply describe what the code does. Connect implementation → parameter choice → observed result → engineering interpretation.

K. Fast debugging decision tree

If you see...	Check this first
No target detected	Target class, confidence threshold, initial frame, detector output.
Box correct but no points	Mask, texture, qualityLevel, minDistance.
Points quickly move to background	Mask, feature distribution, optical-flow filtering.
Track count suddenly drops	Challenge frame, status, error threshold.
Never re-detects	Reliability criterion and state transition.
Re-detects constantly	Criterion too strict, weak initial features, or noisy tracking.
Wrong LEFT/RIGHT action	cx, image width, and state.
SEARCH becomes movement	State centre is probably being used; make state part of the action condition.
MP4 has one/few frames	Runtime crash, writer release, frame size, codec, or output path.

L. Final 30-second self-test

- Can I explain why my target has enough texture/corners for tracking?
- Can I explain why detection is not run blindly on every frame?
- Can I explain status, error, valid_count and median centre?
- Can I explain exactly what triggers RE-DETECTING?
- Can I explain what happens when re-detection fails?
- Can I prove that my video triggered at least one automatic re-detection?
- Can I explain why SEARCH is safer than using the last known centre?
- Can I explain my main parameter choices?
- Can I show the valid-track plot and interpret it?
- Can I verify that my MP4 contains the expected number of frames?
- Can I explain every important part of my submitted code?

Final reminder

The goal is not to make the computer produce a coordinate on every frame. The goal is to build a vision system that knows when its estimate is useful, recognises when it is not, recovers automatically when possible, and avoids unsafe decisions when perception is unreliable.