3006ICT Robotics and Computer Vision

Week 8 Practical Book

Part 3: Group Project Guide

Vision-Guided Autonomous Search and Navigation in Webots

Treat it as an engineering project.

The project is worth 35 marks for the Final Report and 25 marks for the Oral Presentation.
Your goal is to develop one autonomous e-puck controller that visually determines which observation station contains the requested target, navigates safely to that station's designated observation position, and stops there.

The same submitted controller must work across missions.

Project component	Weight	What strong work looks like
Final Report	35 marks	Clear problem/design rationale, technically sound integration, systematic evaluation, robustness evidence, limitations.
Oral Presentation	25 marks	Clear explanation, convincing live demo, strong Q&A, every member understands the complete system.

Understand what the robot should do

Before writing code, rewrite the mission as a sequence of decisions. The project is not simply 'drive to a coordinate'.

Stage	Question	Output
1. Mission input	What target identity is required?	Target identity from the supplied mission interface.
2. Visual search	Which observation station actually displays that target?	Evidence from RGB camera observations.
3. Decide	Do we have enough evidence to identify the station?	Selected target station.
4. Navigate	How do we reach its designated observation position?	Route / waypoints / navigation commands.
5. Stay safe	Is the planned motion safe right now?	Local avoidance or stop/recovery action.
6. Finish	Have we reached the designated observation position?	Goal condition → STOP.

Important constraint: The map and pose information can help you navigate, but they do not tell the robot which station contains the requested target. The station identity should be determined using RGB camera observations. The project explicitly prohibits hard-coded target-to-station mappings and simulator ground-truth shortcuts.

Know what is provided

- Webots R2025a project environment and training worlds for Starts A, B and C.
- e-puck with RGB camera, proximity sensors ps0-ps7, GPS and InertialUnit.
- Eight observation stations and five navigation barriers B1-B5.
- Occupancy-grid map: 40 × 40 cells, 0.10 m per cell; 0 = free, 1 = obstacle.
- Station information and designated observation-position coordinates.
- Reference target images for the eight possible target identities.
- A mission configuration containing the required target identity.

Know what is unknown

- Which observation station contains the requested target in the current mission.
- The target-to-station assignment may change between missions.
- The correct solution therefore needs visual evidence rather than a fixed lookup table.

Start with a System Architecture

A strong group should first draw a one-page architecture diagram.

Do not begin by putting every idea into one large Python file.

Module	Responsibility	Example output
Mission	Read the required target identity.	target = '...'
Perception	Process camera images and compare observations with the target reference.	target score / detection / station evidence
Search strategy	Choose where/when to look for candidate stations.	next station / search action
Localisation	Estimate current robot position/orientation.	x, y, heading
Global planning	Choose a route to a known destination.	waypoint sequence
Waypoint controller	Convert pose error into wheel commands.	left/right velocity
Safety	Override motion when collision risk is high.	AVOID / STOP
State manager	Coordinate search, confirmation, navigation, recovery and completion.	current state
Logging/evaluation	Record what happened and why.	time, state, detection confidence, outcome

A useful architecture is layered. Perception should answer 'what do I see?'; planning should answer 'where should I go?'; control should answer 'how do I move now?'; safety should be able to override motion; the state manager should decide which behaviour is active.

Recommended high-level state machine

State	Purpose	Typical transition
SEARCH / EXPLORE	Move through candidate viewing locations and look for the requested target.	Target evidence becomes strong → CONFIRM.
CONFIRM TARGET	Check that visual evidence is sufficiently reliable.	Confirmed → NAVIGATE.
NAVIGATE	Follow route toward selected observation position.	Obstacle → AVOID; waypoint reached → next waypoint.
AVOID / RECOVER	Handle immediate collision risk or navigation error.	Path safe → NAVIGATE.
ARRIVE	Check final distance/orientation condition if required.	Goal satisfied → STOP.
STOP	Zero wheel commands and end mission safely.	Terminal state.

You do not have to use exactly these states. The point is to make the robot's behaviour explicit rather than hiding decisions inside a long sequence of nested if statements.

Organise the Group Like an Engineering Team

Because the project is worth 60%, divide work by technical responsibility, but avoid creating three isolated mini-projects. The final controller is one integrated system.

Role	Primary responsibility	Must also understand
Perception lead	Target/reference matching, station evidence, visual robustness.	How perception output drives search and navigation.
Navigation lead	Map, localisation, A*, waypoint generation/following.	How visual target identification selects the destination.
Control/integration lead	Differential drive, safety, state machine, integration/testing.	How every module behaves when combined.
All members	Testing, debugging, report, presentation, code review.	Complete system, limitations and failure cases.

Avoid the 'my part / your part' trap. The rubric awards technical integration and Q&A performance. Every member should be able to trace one complete run from mission input → camera evidence → station decision → route → wheel commands → safety → stop.

Team workflow

1. Agree on the architecture and interfaces before parallel implementation.
2. Define what each function/module receives and returns.
3. Use a shared repository/version-control workflow or another traceable collaboration method.
4. Merge small, working changes frequently rather than combining three large branches at the end.
5. Test the integrated controller after every significant merge.
6. Keep a short record of who implemented, tested, reviewed and improved each major component.
7. Before submission, run the complete system as a group and ensure every member can explain it.

A Step-by-Step Development Roadmap

Do not attempt the full project on day one. Build the system in layers and establish a working baseline at every stage.

Stage 0 — Make the supplied environment reproducible

- Open the provided training world and run the starter controller.
- Confirm camera, ps0-ps7, GPS and InertialUnit access.
- Confirm the supplied project configuration and relative paths work on every group member's machine.
- Do not modify the official assessment world.

Exit criterion

Every group member can run the project from a clean copy of the supplied folder without changing machine-specific absolute paths.

Stage 1 — Build low-level motion

- Implement one safe wheel-command function with velocity limits.
- Verify forward motion, left/right turning, rotation and STOP.
- Add a simple pose printout so you can see whether commanded motion matches actual motion.
- Keep wheel-speed conventions consistent throughout the project.

Exit criterion

You can command the robot to move and stop predictably without debugging this layer again.

Stage 2 — Build perception

- Start by loading and inspecting the supplied target reference images.
- Determine what visual representation makes sense for your target-matching approach.
- Test the method on images/views collected from the Webots environment before connecting it to motion.
- Measure false positives and false negatives, not just successful detections.
- Include B1-B5 distractor images in your tests.

Exit criterion

Given an image/view of a station or distractor, your perception module produces a meaningful confidence/evidence result and you understand its failure modes.

Stage 3 — Build station search

The robot is not told which station contains the target. Therefore, you need a strategy for obtaining camera evidence from candidate stations.

- Use the provided station information/map only as allowed for navigation/search planning.
- Decide whether to visit stations in a systematic order or use another defensible strategy.
- Design camera viewpoints so the station image is visible enough for recognition.
- Do not immediately commit to a station from one weak frame.
- Record which stations have been checked and what evidence was obtained.

Key design question

How do you know that 'target not detected' means the station does not contain the target, rather than 'the robot looked from a bad viewpoint'?

Stage 4 — Add target confirmation

Visual recognition in a moving robot is noisy. A strong system should separate 'possible match' from 'confirmed match'.

- Use multiple frames, multiple viewpoints, a confidence threshold, or another evidence-accumulation method.
- Consider requiring consistent evidence before changing the mission state.
- Log the evidence that caused the system to select a station.
- Test visually similar or weakly observed distractors.

Stage 5 — Build navigation to a known destination

Before connecting visual search to navigation, test navigation when the destination is already known.

- Convert the designated observation position into a navigation goal.
- Use the supplied occupancy grid if your chosen method needs it.
- If using A*, verify the path visually before trying to drive it.
- Convert grid cells to physical coordinates using the supplied project coordinate convention; do not guess the map origin.
- Implement waypoint following using current robot pose.
- Add a clear final-distance goal condition and STOP.

Exit criterion

Given a known observation position, the robot can reach it safely from the supplied training starts without relying on timed wheel sequences.

Stage 6 — Add local safety

- Read ps0-ps7 continuously during motion.
- Tune thresholds from actual sensor readings.
- Make safety override waypoint following.
- Test near boundary walls, B1-B5 and station structures.
- Add recovery if the robot becomes stuck.

Stage 7 — Integrate visual search + navigation

- Start at a supplied training position.
- Search candidate stations.
- Collect visual evidence.
- Confirm the target station.
- Select its designated observation position.
- Plan a route.
- Follow waypoints while continuously checking safety.
- Detect arrival and STOP.
- Record mission time and outcome.

Stage 8 — Robustness testing

The project explicitly expects testing across the supplied missions and different starts/targets. A single successful run is not strong evidence of robustness.

- Run all three supplied student missions.
- Test different starting positions A, B and C where available.
- Test different target identities and target-to-station arrangements.
- Repeat runs when tuning parameters so you can distinguish a real improvement from luck.
- Record failure cases and diagnose their causes.

The Main Technical Challenges and How to Approach Them

Challenge 1 — Visual target identification

This is probably the central perception challenge. The robot knows the target identity, but not its station.

Risk	Why it happens	Useful response
False positive	Distractor or background resembles the target.	Use stronger visual features, confidence thresholds and multi-frame confirmation.

Risk	Why it happens	Useful response
False negative	Target is small, blurred, partly occluded or poorly viewed.	Move to a better viewpoint; use multiple observations; improve preprocessing/model.
Viewpoint variation	Station image changes size/angle with robot position.	Collect validation views at different distances and angles.
Single-frame error	One frame happens to look convincing.	Require temporal or multi-view evidence.
Confusing station identity	Robot sees an image but does not reliably know which station it belongs to.	Associate visual observations with robot pose/location and a candidate station.

Challenge 2 — Search efficiency

The 4-minute mission limit makes search strategy important. A perfect detector that searches inefficiently can still fail the mission.

- Avoid unnecessary movement between candidate stations.
- Use the map and known station information to plan an efficient search order where appropriate.
- Prefer viewpoints that provide useful visual evidence rather than merely passing near a station.
- Once the target is confidently identified, stop searching and switch to navigation.
- Measure time spent searching versus time spent navigating.

Challenge 3 — Navigation around obstacles

A shortest geometric path is not automatically the best executable path for an e-puck. Robot size, localisation error, waypoint spacing and obstacle proximity matter.

- Consider adding a safety margin around obstacles when planning.
- Do not make waypoints so dense that the controller constantly changes direction.
- Do not make waypoints so sparse that the robot cuts corners into obstacles.
- Use proximity sensors as a local safety layer even if the global path is collision-free on the map.
- Test routes from multiple starts.

Challenge 4 — Waypoint following

A* returns map cells. The robot needs physical coordinates and a controller that turns position/heading error into wheel commands.

Problem	Possible symptom	What to inspect
Heading error	Robot turns too far or in the wrong direction.	Angle convention, sign, wrap-around.
Distance threshold too small	Robot oscillates around waypoint.	Increase tolerance or reduce speed near goal.
Waypoint spacing too small	Robot makes excessive corrections.	Simplify/smooth the path.
Waypoint spacing too large	Robot cuts corners.	Add intermediate waypoints or a better path-following method.
Localisation noise	Robot appears to wander around the route.	Filter measurements or use a more tolerant controller.

Challenge 5 — Safety versus progress

The robot must make progress without sacrificing collision safety.

Use an explicit priority hierarchy.

A useful starting principle is SAFETY > NAVIGATION > SEARCH. If a collision is imminent, the robot should not continue following a route simply because the route is globally correct.

Challenge 6 — State and recovery

Many failures occur not because an individual algorithm is wrong, but because the system does not know what to do after an unexpected event.

Failure	Weak response	Better response
Target temporarily lost	Continue blindly.	Return to search/reacquisition behaviour.
Obstacle blocks planned route	Keep sending waypoint commands.	Enter avoidance/recovery and re-plan or resume when safe.
Weak target match	Commit immediately.	Accumulate evidence and confirm.
Robot stuck	Continue same command.	Detect lack of progress and trigger recovery.
No valid path	Crash or loop forever.	Report failure state and STOP safely.

Testing Strategy: Treat Testing as Part of the Project

Do not leave evaluation until the final week.

Build a small test matrix from the beginning.

Test dimension	Examples	What to record
Start	A, B, C	Success, time, path, failure reason
Target	Different target identities	Correct station? confidence?
Station type	Boundary vs interior	Recognition/navigation difficulty
Distractors	B1-B5	False-positive behaviour
Viewpoint	Near/far, different approach angles	Detection quality
Obstacle situation	Open path / tight passage	Safety behaviour
Repeated run	Same mission multiple times	Consistency

Define measurable success

Metric	Why it matters
Mission success rate	Shows whether the complete system works, not just one component.
Completion time	The project has a 4:00 simulation-time limit.
Target-station identification accuracy	Measures the critical perception decision.
Collision count / safety failures	Shows navigation robustness.
Final position error	The e-puck centre must finish within 0.20 m of the designated observation position.
Search time	Reveals whether visual exploration is efficient.
Recovery count	Shows how often the controller encounters and handles failures.

Keep a failure log

Run	Mission/start	Observed failure	Likely cause	Change made	Result
Example	Start B / target X	False positive at B3	Weak visual threshold	Raised confidence + multi-frame confirmation	Retest pending

A failure log is valuable for both engineering and the final report: it shows how design decisions were driven by evidence.

Integration Debugging: Test the Interfaces

Once individual modules work, most difficult bugs occur at the boundaries between modules.

Interface	Question to test
Perception → Search	Does the visual module clearly report confidence and station evidence?
Search → Mission state	Does the system switch to navigation only after sufficient evidence?
Planner → Waypoint controller	Are map coordinates converted correctly into Webots

Interface	Question to test
	coordinates?
Waypoint controller → Motors	Do pose errors produce the intended wheel commands?
Sensors → Safety	Does an obstacle override navigation immediately enough?
Navigation → Goal detector	Does the robot STOP at the correct observation position rather than simply near it?

Debug from the robot's decision trace

When something fails, reconstruct the sequence:

1. What did the robot see?
2. What did the perception module output?
3. What state was the controller in?
4. What goal/waypoint was selected?
5. What wheel command was generated?
6. What did the proximity sensors report?
7. What happened after the next timestep?
8. At which point did the system diverge from the intended behaviour?

Do not tune everything at once.

Change one meaningful factor, rerun the same test, and compare the result. Otherwise you cannot tell which change helped or hurt.

Appendix

Suggested Group Timeline

Phase	Main goal	Deliverable/evidence
1. Understand	Read requirements; run environment; agree architecture.	One-page architecture + task checklist.
2. Baselines	Reliable motion, camera, sensors, pose, map.	Working low-level modules.
3. Perception	Target matching and visual station evidence.	Offline/controlled recognition results.
4. Navigation	Known-goal route + waypoint following + safety.	Robot reaches known observation positions.
5. Integration	Search → identify → navigate → stop.	End-to-end training runs.
6. Robustness	Different starts/targets, distractors, failures.	Test matrix + failure log.
7. Evaluation	Three supplied missions + meaningful failure cases.	Final results table/plots/screenshots.
8. Report/demo	Explain decisions and evidence.	Final report + stable demo + Q&A preparation.

Do not postpone integration

A common project mistake is spending most of the available time making three modules individually excellent and then discovering that the interfaces do not work together. Start end-to-end testing earlier than feels comfortable.

How to Build a Strong Final Report

The report is worth 35 marks. It should tell the story of an engineering system: problem → design decision → implementation → evidence → limitations.

Rubric area	What to demonstrate
Problem understanding / design rationale — 8	Clear interpretation of the task, constraints, architecture and reasons for major design choices.
Technical approach / integration — 9	How perception, search, planning, control, safety and state management work together.
Evaluation / discussion — 8	Systematic tests, all three supplied student missions, success/failure, time and meaningful failure cases.
Complexity / robustness — 6	Non-trivial integration, robustness to changing starts/targets and evidence that the system handles difficult cases.
Writing / presentation — 4	Clear structure, figures/tables that explain the system, concise technical writing.

Strong report evidence

- Architecture diagram showing data flow and behaviour/state transitions.
- Example visual recognition results, including difficult or failed cases.
- Example planned route over the occupancy grid.
- Example robot trajectory or waypoint-following result.
- A results table covering the three supplied missions.
- Completion times and final position errors.
- Meaningful failure cases with diagnosis and what was changed.
- A contribution table showing responsibilities and integration work.
- Limitations and realistic improvements.

Do not write a success-only report.

A strong engineering evaluation does not hide failures. Explain what failed, why you believe it failed, what evidence supports that diagnosis, and whether your change improved the system.

How to Prepare for the 25-Mark Presentation

Component	Time	What to prepare
Presentation	10 min	Problem, architecture, key technical decisions, experiments and results.
Live Webots demo	Up to 4 min	Stable submitted controller; no code edits during the selected mission.
Q&A	6 min	Every member explains the complete system, decisions, limitations and results.

Suggested 10-minute story

- Problem and constraints — 1 min.
- System architecture — 1.5 min.
- Visual target identification — 2 min.
- Navigation and waypoint control — 2 min.
- Safety/recovery and integration — 1 min.
- Experimental results — 1.5 min.
- Limitations and key lesson — 1 min.

Q&A preparation

- Why did you choose this perception method?
- How do you know the target-station decision is reliable?
- How do you handle false positives and false negatives?
- Why did you choose your search order?
- How does A* or your planner interact with local safety?
- How do you convert a planned route into wheel commands?
- What happens if the target is temporarily lost?
- What happens if the robot gets stuck?
- What was your most important failure case?
- What would you improve with more time?

Assessment Rules You Must Not Accidentally Violate

Do	Do not
Use the RGB camera to determine the target station.	Use Webots Camera Recognition or simulator ground truth.
Use the supplied map, GPS, orientation and station information for appropriate navigation.	Use map/coordinates as a substitute for visual target identification.
Keep one controller working across missions.	Hard-code a target-to-station mapping or mission-specific route.
Use copies of training worlds for development.	Modify the official assessment world geometry/assets.
Use reproducible relative paths.	Depend on machine-specific absolute paths.
Acknowledge external code/models/resources as required.	Submit code you cannot explain or justify.
Test the complete submitted controller.	Edit the controller after the assessment mission is selected.

Mission constraints

Each assessed mission has a 4:00-minute simulation-time limit. The final e-puck centre must be within 0.20 m of the designated observation position, with no collision or manual intervention.

Academic Integrity and Responsible AI Use

The project materials state that groups must understand and be able to explain the submitted system, acknowledge external code/models/resources appropriately, and comply with course and Griffith University guidance for generative AI.

Useful AI assistance	Unsafe project practice
Explain an unfamiliar Webots/OpenCV/Python concept.	Generate a complete system and submit it without understanding it.
Suggest debugging hypotheses from your observed logs.	Invent experimental results or claim an untested method works.
Compare alternative algorithms.	Use AI to bypass the need for your group to design and evaluate the system.
Review your explanation for clarity.	Submit code containing components no member can explain in Q&A.

Your test is simple:

If the examiner points to any important function and asks 'Why is this here?', 'What assumption does it make?', or 'What happens if it fails?', every member should be able to answer.

The Most Important Design Principles

Principle	Meaning in your project
Separate perception from action	A detector should report evidence; the controller decides what to do with it.
Use confidence, not blind certainty	Visual recognition can be wrong; confirmation reduces accidental

Principle	Meaning in your project
	commitment.
Plan globally, control locally	A planner chooses a route; feedback control executes it safely.
Safety can override the goal	Collision avoidance must be able to interrupt navigation.
Design for failure	Target loss, false detection, blocked paths and localisation errors are normal engineering cases.
Measure performance	Use time, success rate, final error, collision/failure counts and recognition evidence.
Integrate early	End-to-end tests reveal interface problems that unit tests cannot.
Keep the controller reproducible	The final package should run without manual path editing or mission-specific changes.
Know the whole system	The presentation and Q&A assess individual understanding as well as group output.

Group Project Final Checklist

☐ We can run the supplied project environment on every group member's machine.
☐ We have a clear architecture and state/behaviour design.
☐ Our wheel commands have safe limits and a reliable STOP state.
☐ Our visual system uses the RGB camera to identify the target station.
☐ We have tested distractor images and difficult visual cases.
☐ We do not rely on a fixed target-to-station mapping.
☐ We do not use simulator ground truth, Camera Recognition, .wbt parsing or metadata shortcuts.
☐ We can navigate to a known observation position before integrating target search.
☐ We have local collision safety that can override navigation.
☐ We have recovery behaviour for important failure cases.
☐ We have tested the same controller across the supplied student missions.
☐ We record success/failure, completion time and final position accuracy.
☐ We have meaningful failure cases and can explain them.
☐ We have a contribution record for the group.
☐ Every member understands the complete system.
☐ The submitted code uses reproducible relative paths and contains only required files.
☐ We have tested the exact controller/package intended for submission.
☐ We can explain our main design decisions in the presentation and Q&A.