Unit 10 Reflection

This iteration aimed to align object-oriented design with measurable code quality. Radon reports **139 blocks** with an average **cyclomatic complexity of 2.85 (A)** and an average **Maintainability Index of 77.8** across the project, indicating generally simple, testable code. The standout hotspot is `Robot.tick` (**CC = 38, E-rank**) alongside a small number of C/B ranks (e.g., `GreedyPlanner` and `AStarPlanner`). These results are consistent with McCabe's view that decision density concentrates risk and maintenance effort (McCabe, 1976).

Two architectural choices demonstrably contain complexity. First, the **Strategy** pattern decouples path-planning algorithms from navigation, so algorithmic variation does not cascade into additional branches within the controller. Second, a lightweight **Observer/EventBus** isolates sensor events from consumers, reducing temporal coupling and avoiding nested control flow. Together with **dependency injection**, these practices localise complexity and improve testability, echoing the intention behind classic design patterns (Gamma et al., 1995).

For the remaining hotspot, I will refactor `Robot.tick` using a **State/Command dispatch** (e.g., `Off`, `Idle`, `Navigating`, `Charging`) combined with **guard clauses** and **extract-method** refactorings. The expected outcome is to convert the single E-rank into B or better and eliminate the few C-ranks by splitting responsibilities into small, pure helpers. This approach complements object-oriented metrics beyond CC: lowering **WMC** at class level and controlling inter-class interactions measured by **CBO/RFC** (Chidamber and Kemerer, 1994).

Finally, the metrics will be integrated into CI to guard against regressions (quality gate **MI ≥ 70** with no E/D in production code). This continuous feedback loop maintains a low decision count per method, improves cohesion, and preserves loose coupling. In sum, the measured improvements validate the design-pattern choices and provide a clear path to reduce residual hotspots, keeping the codebase maintainable, testable, and ready for further optimisation in resource-constrained IoT contexts.

References:

Chidamber, S.R. and Kemerer, C.F. (1994) ‘A metrics suite for object-oriented design’, *IEEE TSE*, 20(6), 476–493.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) *Design Patterns*. Addison-Wesley.

McCabe, T.J. (1976) ‘A complexity measure’, *IEEE TSE*, SE-2(4), 308–320.

Romano, F. and Kruger, H. (2021) *Learn Python Programming*. 3rd edn. Packt.