POLITECNICO DI TORINO

Corso di Laurea Magistrale
in INGEGNERIA MECCANICA

Tesi di Laurea Magistrale

Terrain-Aware Navigation of an 8-DOF Off-Road Autonomous UGV Swarm over a Progressively Discovered Map

Relatore:
Prof. Mauro Velardocchia

Candidato:
Renato Cerrato

Correlatori:
Prof. Aldo Sorniotti
Prof. Antonio Tota

Academic Year 2025–2026

“If we knew what it was we were doing, it would not be called research, would it?”

— A. Einstein

a mia madre Isabella, a mio padre Roberto,
a mio fratello Alessandro, a Mineth

Abstract

This work presents the development of a MATLAB/Simulink simulation framework for a swarm of autonomous off-road ground vehicles. The operating environment is a partially known semantic map with progressive discovery. Terrain perception is handled using a Deep Learning based semantic segmentation module by assigning a terrain class to each cell of the input image. A cost map is constructed to combine the base cost per semantic class, the gradient where the class changes and the slope penalty. Furthermore, cells not yet observed receive a uniform penalty override, which is replaced by the real cost as the terrain is discovered. The architecture is a multi-agent, decentralized decision-making framework on a shared global map: the leader creates the path for itself through an enhanced A* terrain-aware algorithm, to which it is subsequently applied a path smoothing function that ensures curvature continuity. The leader is also in charge of path planning for the followers using a computationally cheaper, yet riskier method. The vehicle that holds the leader position is not always the same during the simulation, but can be updated: at each step, a ranking mechanism elects the vehicle with the lowest ETA as leader. The agents follow their defined path via a pure-pursuit path tracker, while the longitudinal velocity is regulated by an MPC whose reference is emitted by a speed reference governor that responds to the risks of failure. Each vehicle is described by an 8-degree-of-freedom dynamic model including a Pacejka tire model, roll dynamics, a static electric motor map (SPM) and four-wheel steering system. The framework has proven effective in simulations on different maps and demonstrates the consistent integration of perception, planning, coordination, and realistic vehicle dynamics, providing a flexible basis for future extensions.

Contents

List of Figures

List of Tables

List of Abbreviations

4WS	Four-Wheel Steering
ABS	Anti-lock Braking System
ACO	Ant Colony Optimization
A*	A-star search algorithm
AI	Artificial Intelligence
APF	Artificial Potential Field
BWR	Bekker–Wong–Reece model
CG	Center of Gravity
CM	Center of Mass
D*	Dynamic A-star search algorithm
D* Lite	D-star Lite incremental search algorithm
DEM	Discrete Element Method
DOF	Degrees of Freedom
DRL	Deep Reinforcement Learning
EBD	Electronic Brakeforce Distribution
ECU	Electronic Control Unit
ETA	Estimated Time of Arrival
FEM	Finite Element Method
FL	Front Left wheel
FR	Front Right wheel
ID	Vehicle identifier
LiDAR	Light Detection and Ranging
LTR	Load Transfer Ratio
LUT	Look-Up Table
mIoU	mean Intersection over Union
ML	Machine Learning
MPC	Model Predictive Control
PD	Proportional–Derivative control
PI	Proportional–Integral control
PID	Proportional–Integral–Derivative control
PSO	Particle Swarm Optimization
QP	Quadratic Programming
RC	Roll Center
RGB	Red–Green–Blue color model
RL	Rear Left wheel
RMSE	Root Mean Square Error
RPM	Revolutions Per Minute
RR	Rear Right wheel
RRT	Rapidly-exploring Random Tree
RRT*	Optimal Rapidly-exploring Random Tree
SAE	Society of Automotive Engineers
SAM	Segment Anything Model
SIM	Simulink model reference
SLAM	Simultaneous Localization and Mapping
SPM	Surface Permanent Magnet
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle

Chapter 1

Introduction

1.1 Context and Motivation

Autonomous ground vehicles are increasingly relevant in applications where human intervention is either inefficient, risky, or impossible. Typical examples include search and rescue operations, agricultural monitoring, environmental inspection, planetary exploration, and infrastructure surveillance. In structured environments, such as urban roads or industrial facilities, autonomous navigation can rely on relatively predictable conditions. Conversely, off-road environments introduce a substantially higher level of uncertainty. The terrain may be uneven, partially unknown, deformable, or characterized by spatially varying friction. Obstacles may not be known in advance, and vehicle stability may be affected by slope, roll dynamics, and local tire–terrain interaction. The use of multiple UGVs offers several advantages over a single-vehicle architecture. A swarm can improve exploration capability, increase redundancy, distribute risk among multiple agents, and maintain mission continuity even in case of partial failure. However, these benefits require a suitable integration of path planning, vehicle control, map sharing, and dynamic feasibility. This thesis addresses the development and validation of a simulation framework for a swarm of autonomous off-road UGVs operating in a partially known semantic environment. The work integrates terrain-aware path planning, leader–follower coordination, progressive map discovery, and an advanced vehicle dynamics model.

1.2 Problem Statement

The problem considered in this thesis is the autonomous navigation of a group of ground vehicles in an off-road environment represented by a semantic terrain map. While the Goal coordinates are known among the agents, the environment is only partially known by the swarm during the mission. Each vehicle contributes to the discovery of the environment through a simplified perception model, while the planner computes feasible

paths by considering terrain class, obstacles, slope, and unknown areas.

The main challenge is not only to compute a geometrically valid path, but also to ensure that the path is meaningful for vehicles with non-negligible dynamics. Therefore, the navigation problem is coupled with a vehicle model that accounts for tire forces, roll dynamics, vertical load transfer, and local terrain properties.

1.3 Objectives of the Thesis and Main Contributions

The objective of this thesis is to develop and validate a modular MATLAB/Simulink simulation environment for the analysis of a swarm of UGVs operating on off-road terrain. The proposed framework combines multi-agent coordination, terrain-aware path planning, progressive map discovery, and vehicle dynamics modeling within a single simulation architecture.

The main objectives and contributions of the thesis can be summarized as follows:

- development of a modular multi-agent simulation architecture based on MATLAB and Simulink model references;
- definition of hierarchical nonvirtual buses for structured communication between agents, planners, controllers, vehicle models, and visualization blocks;
- modeling of an off-road semantic terrain map including traversability, obstacles, friction coefficients, slope information, and progressive knowledge of the environment;
- implementation of a shared global map knowledge mechanism, allowing the explored portion of the map to increase during the simulation according to the sensing range of the vehicles;
- implementation of a terrain-aware global path planner based on an enhanced A* algorithm, in which the path cost depends on semantic terrain classes, obstacles, unknown areas, and slope-related penalties;
- integration of path smoothing techniques to generate smoother reference trajectories suitable for vehicle tracking;
- definition of a leader–follower formation strategy with dynamic role assignment, in which the leader is not fixed but is elected at run time by a ranking mechanism executed on every agent; the current leader follows the global planned path while the followers track formation targets expressed with respect to the leader frame;
- coupling of swarm-level planning and formation logic with an 8-DOF vehicle dynamics model, allowing the simulation to account for vehicle motion, speed profiles, steering commands, tire–terrain interaction effects, and local terrain properties;

- development of replay, plotting, and visualization tools for analyzing vehicle trajectories, map discovery evolution, planned paths, smoothed paths, speed profiles, and control commands;
- evaluation of the swarm behavior through simulation results, with particular attention to trajectory tracking, formation maintenance, terrain-dependent planning, and progressive environment discovery;
- creation of a flexible simulation foundation for future extensions, such as leader failure management, decentralized local maps, D* Lite replanning, collision avoidance strategies, communication constraints, and BWR-based terrain interaction models.

Chapter 2

State of the Art and Literature Review

2.1 Autonomous UGVs in Off-Road Environments

Off-road autonomous navigation differs significantly from road-based autonomous driving. In off-road scenarios, the vehicle cannot rely on lane markings, traffic rules, or structured road geometry. The terrain may include natural obstacles, steep slopes, low-friction regions, vegetation, rocks, mud, sand, or water. The main challenges are incomplete or uncertain map information, spatially varying tire–terrain interaction, reduced and variable friction, slope-induced longitudinal and lateral load transfer, obstacle avoidance in unstructured terrain, limited perception range, dynamic feasibility of planned trajectories. This work focuses on the interaction between terrain-aware planning and vehicle dynamic feasibility. The objective is not only to find a collision-free path, but also to generate paths and speed references that are meaningful for an off-road ground vehicle.

2.2 Vehicle Dynamics Models

Several vehicle dynamics models can be used for autonomous driving and unmanned ground vehicle simulation. The appropriate model depends on the required compromise between computational efficiency and physical accuracy. Low-order models are usually preferred for high-level planning and real-time control because they are simple, robust, and computationally efficient [1]. However, when the vehicle operates on off-road terrain, simplified models may become insufficient, since slope, variable friction, wheel slip, terrain irregularities, roll dynamics, and vertical load transfer can significantly affect the vehicle response.

In autonomous navigation, kinematic and bicycle models are commonly used for path planning and path tracking because they provide a compact description of the vehicle motion. A kinematic model usually describes the geometric evolution of the vehicle pose, typically through the variables x, y, ψ, and v. This type of model is useful for trajectory

generation and high-level control, but it does not explicitly represent tire forces, wheel slip, load transfer, or body attitude dynamics.

A more advanced reduced-order formulation is represented by the single-track vehicle model, also known as the bicycle model. In this model, the left and right wheels of each axle are replaced by one equivalent front tire and one equivalent rear tire. The basic dynamic bicycle model describes the planar motion of the vehicle chassis through three degrees of freedom: longitudinal motion, lateral motion, and yaw motion. These are commonly represented by the longitudinal velocity u, the lateral velocity v, and the yaw rate $r = \dot{\psi}$. The yaw angle ψ is then obtained by integrating the yaw rate.

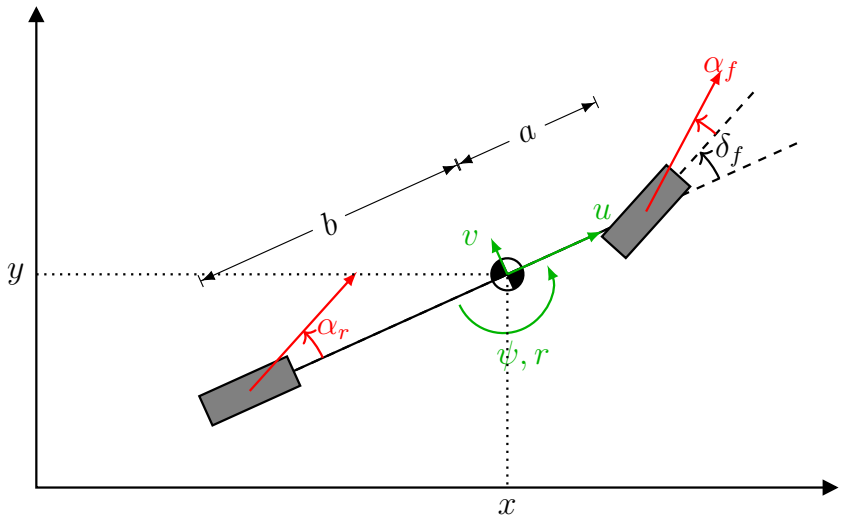


Figure 2.1: Schematic representation of the single-track bicycle model, including wheelbase geometry, velocity vectors, sideslip angle, steering angle, tire slip angles, yaw rate and curvature radius.

The 4-DOF bicycle model considered in this work is a single-track model composed of a 3-DOF chassis representation and one additional tire-related degree of freedom. The chassis dynamics describe the planar motion of the vehicle through longitudinal velocity u, lateral velocity v, and yaw rate $r = \dot{\psi}$. The additional tire degree of freedom is introduced to account for the dynamic behavior of the equivalent tires, improving the representation of tire slip with respect to a purely 3-DOF chassis model [2].

Although the single-track model uses only one equivalent front tire and one equivalent rear tire, the tire-related dynamics should not be interpreted as the dynamics of a single physical wheel. Instead, they are associated with the equivalent tires of the single-track formulation. For each equivalent tire, $i \in \{F, R\}$, the tire slip quantities can be expressed through the lateral slip angle α_i and the longitudinal slip ratio σ_i. The slip ratio can be related to the equivalent wheel angular velocity as

$$\sigma_i = \frac{R_w \omega_i - v_{x,i}}{v_{x,i}}, \qquad i \in \{F, R\}, \tag{2.1}$$

where R_w is the wheel radius, ω_i is the angular velocity of the equivalent wheel, and $v_{x,i}$ is the longitudinal velocity at the tire contact point. The corresponding wheel rotational dynamics can be written as

$$T_i - I_w\dot{\omega}_i - F_{x,i}R_w = 0, \qquad i \in \{F, R\}, \tag{2.2}$$

where T_i is the driving or braking torque, I_w is the wheel inertia, and $F_{x,i}$ is the longitudinal tire force. Depending on the adopted tire model, the additional tire dynamics may also be represented through relaxation dynamics of the tire slip variables.

Table 2.1: Degrees of freedom in a 4-DOF single-track vehicle model.

DOF	Motion	Variable	Physical meaning
1	Longitudinal chassis	u	Forward velocity of the vehicle body.
2	Lateral chassis	v	Lateral velocity of the vehicle body.
3	Yaw	$r = \dot{\psi}$	Rotation rate around the vertical axis.
4	Tire/slip dynamics	α_i, σ_i, ω_i	Additional tire-related dynamics for the equivalent tires, with $i \in \{F, R\}$.

A more detailed alternative is represented by double-track multi-degree-of-freedom models. In [3], the authors compare a bicycle model, an 8-DOF model, and a 14-DOF model for MPC-based path tracking, showing how the increase in model fidelity improves the representation of vehicle dynamics, especially when roll motion and lateral load transfer become relevant.

The 8-DOF model extends the chassis dynamics by including roll motion and the rotational dynamics of the four wheels. Therefore, it can account for individual wheel speeds, longitudinal slip, tire force generation, roll-induced load transfer, and the different behavior of the left and right sides of the vehicle. This is particularly relevant for off-road UGV simulation, where spatially varying friction, side slope, terrain irregularities, and braking or traction commands can generate different conditions at each tire.

Table 2.2: Degrees of freedom in an 8-DOF double-track vehicle dynamics model.

DOF	Motion	Variable
1	Longitudinal chassis	u
2	Lateral chassis	v
3	Yaw	ψ, r
4	Roll	ϕ, $\dot{\phi}$
5	FL spin	ω_{FL}
6	FR spin	ω_{FR}
7	RL spin	ω_{RL}
8	RR spin	ω_{RR}

The 14-DOF model further increases the physical fidelity by including six degrees of freedom at the vehicle center of mass and two degrees of freedom for each wheel. Compared with the 8-DOF model, it adds vertical motion, pitch motion, and independent vertical suspension dynamics at each wheel. This makes the 14-DOF model more suitable for rollover prediction, wheel lift-off analysis, suspension studies, and high-fidelity validation under extreme maneuvers. However, it also introduces a higher computational burden and requires a larger number of parameters, which are often difficult to identify accurately, especially for small off-road robotic platforms.

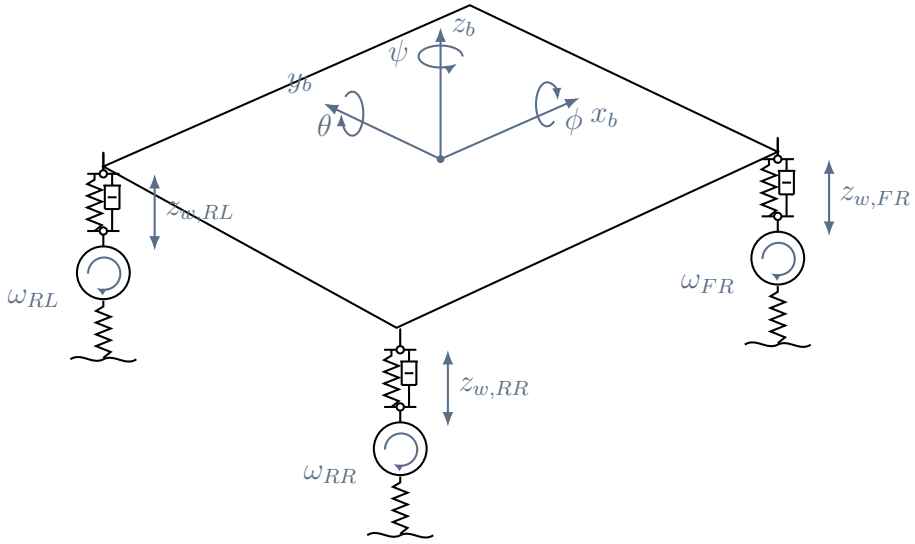


Figure 2.2: Schematic representation of the 14-DOF vehicle model

Table 2.3: Degrees of freedom in a 14-DOF vehicle dynamics model.

DOF	Motion	Variable
1	Longitudinal C.M.	u
2	Lateral C.M.	v
3	Vertical C.M.	w
4	Roll	ϕ, ω_x
5	Pitch	θ, ω_y
6	Yaw	ψ, ω_z
7	FL vertical	$z_{u,FL}$
8	FR vertical	$z_{u,FR}$
9	RL vertical	$z_{u,RL}$
10	RR vertical	$z_{u,RR}$
11	FL spin	ω_{FL}
12	FR spin	ω_{FR}
13	RL spin	ω_{RL}
14	RR spin	ω_{RR}

Table 2.4: Comparison of vehicle models for autonomous navigation.

Model	Advantages	Limitations	Typical use
Kinematic model	Very simple, fast, and robust.	Neglects tire forces, wheel slip, load transfer, and body dynamics.	Planning and preliminary path tracking.
4-DOF model	Captures planar chassis dynamics and includes tire/slip dynamics with low computational cost.	No left–right wheel distinction, no explicit roll dynamics, and no lateral load transfer.	Real-time prediction, yaw stability studies, and reduced-order control design.
8-DOF model	Captures roll, yaw, individual wheel spin, tire slip, and vertical load redistribution.	Does not explicitly include pitch, heave, or independent suspension vertical dynamics.	Dynamic validation, off-road simulation, and wheel-level control logic.
14-DOF model	Adds pitch, heave, suspension vertical dynamics, and wheel-level vertical motion.	High computational cost and higher parameterization complexity.	Rollover analysis, suspension studies, and high-fidelity validation.

2.3 Off–Road Tire–Terrain Interaction

The interaction between tires and terrain is central in off-road vehicle dynamics. Tire forces depend on vertical load, slip ratio, slip angle, friction coefficient, and terrain properties. In many vehicle dynamics models, tire behavior is represented through empirical formulations such as Pacejka's Magic Formula.

In off-road applications, additional effects may become relevant, such as sinkage, soil compaction, rolling resistance variation, and deformable terrain behavior. These phenomena are often described by terramechanics models, including BWR theory, which has been extensively studied and implemented in simulation in previous theses [4] and research articles [5]. Compared with simple friction-based models, BWR formulations provide a more physically detailed description of wheel–soil stresses, sinkage, and traction on deformable terrain. In this thesis, the terrain is represented through semantic classes and friction coefficients, while a more advanced BWR formulation is identified as a possible future extension.

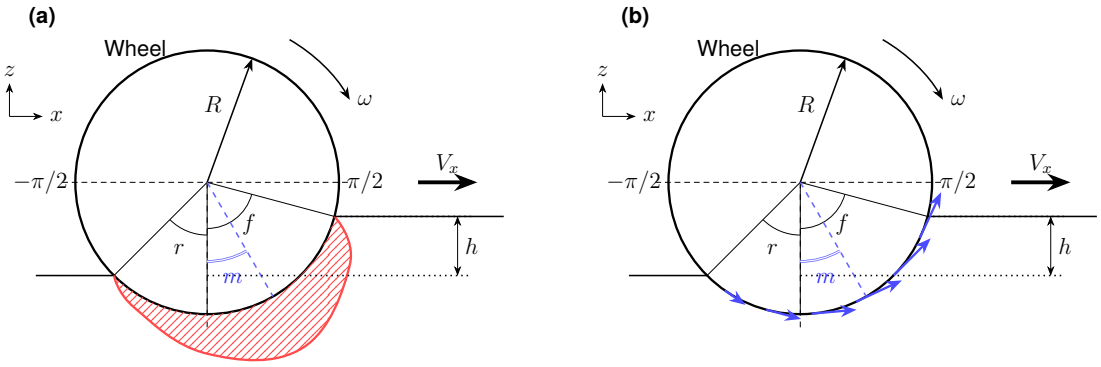


Figure 2.3: Schematic illustration of stress distribution due to wheel–soil interaction in the BWR model: (a) normal stress distribution beneath a rigid wheel; (b) tangential stress distribution beneath a rigid wheel. Adapted from [5].

More recently, high-fidelity numerical approaches have also been adopted to study tire–terrain interaction. FEM models represent the tire and the soil as deformable continua discretized into finite elements, allowing tire deformation, soil sinkage, contact stresses, drawbar pull, and rolling resistance to be estimated under different loading and slip conditions [6]. For granular terrains such as sand, coupled FEM–DEM approaches are particularly relevant: the tire is generally modeled as a deformable finite-element structure, while the soil is represented through discrete particles. This allows particle flow, local compaction, and force transmission beneath the tire to be described more explicitly [7–9]. Industrially relevant research has also investigated commercial tire concepts, including Bridgestone agricultural tires and Michelin TWEEL non-pneumatic tires, with the objective of reducing soil compaction and improving off-road mobility [10].
However, these models require detailed soil parameters, experimental calibration, and significantly higher computational effort. For this reason, they are not implemented in the present work and are instead proposed as future developments.

2.4 Semantic Traversability Mapping

This is the first step to take in the present work. The identification of the map, even before launching the planning algorithms, tracking, safety systems, and so on, is of vital importance because it generates the reference used by all of them. The mapping consists of identifying the type of terrain, slope and inclination, obstacles that the vehicle is about to traverse. It is also the most innovative section of this project from the programming point of view, because it often involves the use of artificial intelligence [11]. AI — more specifically ML — in order to be used for pattern recognition — of images, but also LiDAR detections, useful for the present case — must first be trained on a dataset. Although there exist reasonably accurate and well-developed AI models, it is important to have a dataset that associates the signal with an equivalent piece of information. It will be

the AI's task to check the input signal, try to define the equivalent information, and check how far it has strayed from the real one. Larger and more representative datasets can improve generalization, provided that label quality, class balance, and domain coverage are adequate. A well-known example is the SA-1B dataset, containing over 1 billion masks, released together with the Segment Anything Model (SAM) [12] and later extended by its successor SAM 2 [13]. As regards the semantics of the terrain obtained from satellite, it is important to cite, among many, that of SpaceNet [14], which stands out especially in the recognition of buildings and roads. The datasets [15, 16] instead are used for the recognition of semantics from UAV, with many different classes and from a fairly close distance. The autonomous vehicles considered in this work would benefit from a map recognition system implemented on board and employed, besides by camera, also by means of other sensors, such as LiDAR, which will help to evaluate distances as well as textures [17]. Research is also present on one of the possible applications of this work: extraterrestrial exploration [18]. On the recognition of the environment for Martian rovers, the reference works are [19, 20], their associated dataset is AI4Mars [21].

2.5 Path Planning Algorithms

Path Planning algorithms calculate a feasible path from a starting position to a target while avoiding obstacles and minimizing a cost function. A distinction can be made between deterministic algorithms — often graph-based, which always produce the same path for a given input — and stochastic algorithms, in which the path varies between executions.

2.5.1 Heuristic Algorithms

The most classic graph-based method is Dijkstra's algorithm [22]; while it expands nodes in order of accumulated cost and guarantees an optimal solution, it explores the graph isotropically. It proves to be a computationally heavy, slow, and often inefficient algorithm; however, many variants exist — among all Bidirectional Dijkstra — which is being studied and adapted for modern problems due to its stability [23–25]. A* algorithm [26] improves Dijkstra general efficiency giving order to the expansion using the equation

$$f(n) = g(n) + h(n) \tag{2.3}$$

where $g(n)$ is the accumulated cost from the start node to node n, and $h(n)$ is the estimated cost from node n to the goal. A* largely preserves Dijkstra's optimality while expanding far fewer nodes, provided a suitable h is used. As shown in Figure 2.4, Dijkstra finds a path with a lower traversal cost, but the number of explored cells is exponentially higher.

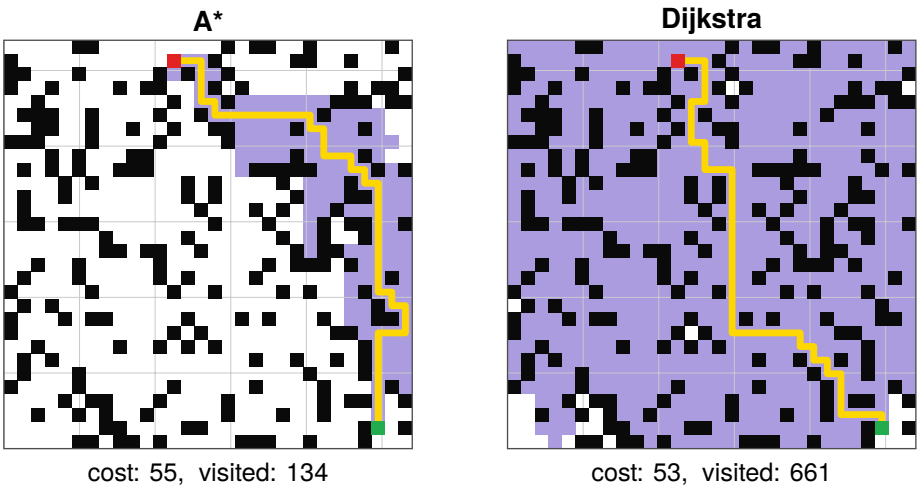


Figure 2.4: Comparison between A* and Dijkstra on the same map: A* explores a narrow band of cells around the optimal path, while Dijkstra explores almost the entire free space before reaching the goal.

In A* the balance between the path quality and research speed is regulated weighting the heuristic function h. Among the dynamic-weighting variants, taking as an example the first one to be developed [27], one uses

$$f(n) = g(n) + w \cdot h(n) \tag{2.4}$$

with $w > 1$, making the search greedier toward the goal at the cost of a suboptimality bounded by w; in the paper [28], instead, w is decreased with the depth of the search, obtaining an aggressively exploratory behavior far from the goal and an accurate one in its proximity. On grids, the angular expressiveness is increased with neighborhoods extended to 16 or 32 moves [29], which reduce the staircasing of the paths. For environments that are only partially known or discovered during motion, D* [30] and D* Lite [31] make the search incremental: instead of replanning from scratch at every discovery, they locally repair the previous solution, proving to be effective even in maps with variable obstacles [32]. In this thesis the main planner is A* re-executed periodically; the architecture is, however, set up to allow its replacement with D* Lite for more efficient replanning. This choice is due to the fact that A* is an algorithm that is simple to implement, reliable, and relatively fast to execute in a map with almost fixed obstacles.

2.5.2 Stochastic Algorithms

The most famous of the stochastic algorithms is RRT: according to its procedure, a tree is built by iteratively expanding it toward points sampled randomly in the configuration space, connecting each new sample to the nearest node of the tree. It is known for handling high-dimensional spaces and non-holonomic constraints well, and it is fairly fast

compared to its deterministic competitors. However, the path found is often not optimal, and the resulting trajectory is irregular and zig-zagging. The improvements and optimal variants of this algorithm mainly study and investigate alternative methods that speed up the convergence time [33]. RRT* is another stochastic algorithm that extends RRT by adding a rewiring step: when a new node is inserted, the algorithm checks whether connecting it through a different neighboring node reduces the total cost, and it optimizes the existing connections in the surrounding area. In this way asymptotic optimality is guaranteed: it converges to the optimal solution with an infinite number of samples and retains the advantages of RRT in complex spaces, but it is more computationally expensive because of the continuous rewiring, and the convergence to the optimum can become very slow [34]. For this algorithm too, in fact, the reason for refinement is often to speed up the convergence [35], as studied in [36, 37]. Particularly interesting for the present application is [38], which proposes to drastically speed up the algorithm by introducing an APF in the proximity of the obstacles. The substantial difference between RRT and RRT* is that while the first one rapidly finds a feasible path through a complex environment, the second is an optimized extension that guarantees finding the optimal path as the number of samples increases [39], as shown in Figure 2.5.

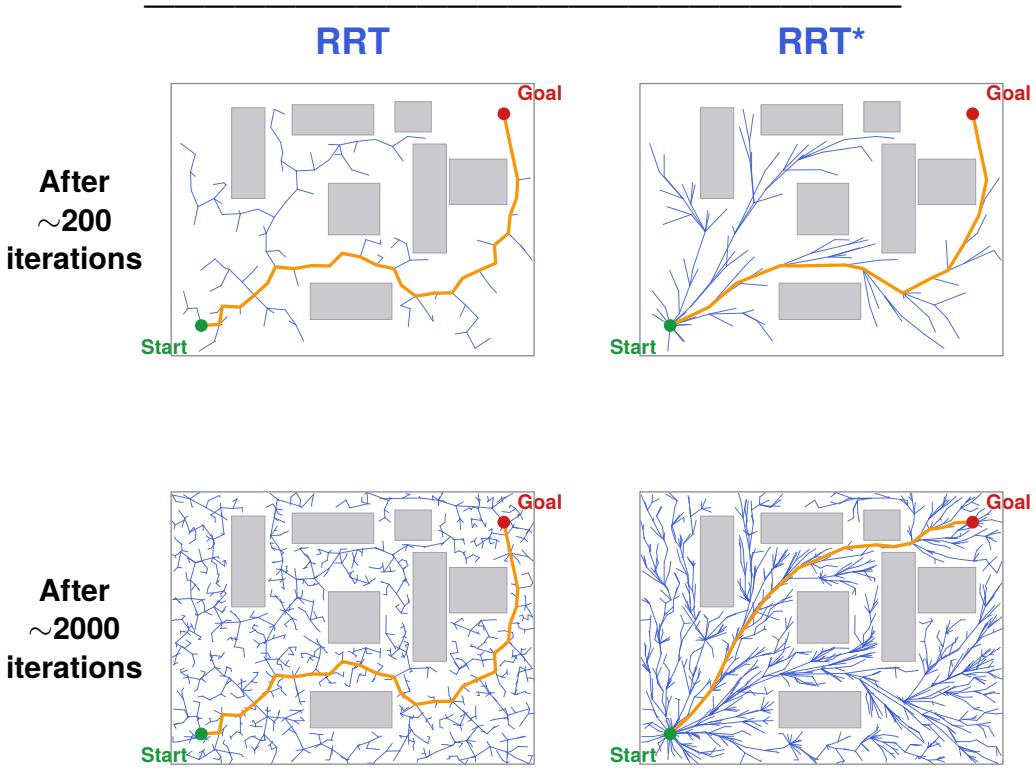
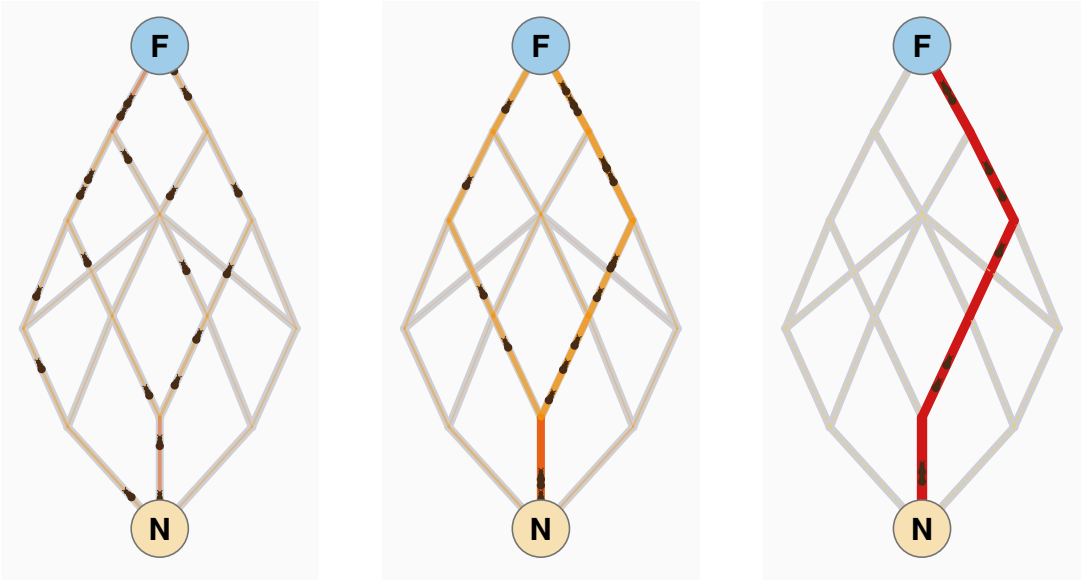


Figure 2.5: Comparison between RRT and RRT* on the same map: RRT*, thanks to rewiring, reaches asymptotically the optimal path.

ACO [40, 41] is a very interesting path planning algorithm: as its very name suggests, it is one of the most famous Bio-Inspired algorithms. It simulates the behavior

of a colony of ants that deposit pheromone on the paths they travel: the shorter paths accumulate more pheromone and become more attractive to the subsequent ants, while the pheromone evaporates over time. Its functioning is illustrated in Figure 2.6. It is a robust algorithm and particularly suited to exploration, but it presents slow convergence and does not guarantee the global optimum [42, 43]; moreover, it requires careful tuning of the parameters — evaporation, importance of the pheromone vs. heuristic: [44] created an enhanced version through a smart autotuning of the latter.



1
Exploration: ants are trying all possible paths.

2
Reinforce: pheromone is accumulating on shorter paths

3
Convergence: shortest path is dominating

Figure 2.6: Schematic presentation of ACO algorithm.

PSO also appears to be a good search algorithm: it consists of a set of particles that moves through the search space, updating velocity and position based on its own best position found and on the best position found by the entire swarm. It is simple to implement, requires few parameters to tune, and has good convergence on continuous optimization problems, but it can remain trapped in local optima, and is natively less suited to spaces with complex constraints such as the obstacles in the present case study [45–47].

2.6 Leader–Follower Formation

Formation Control is another very important aspect of swarm control [48]. While the Path Planning for a swarm can be adapted from that of a single vehicle, Formation Control is precisely what differentiates the swarm logic from the single vehicle. In leader–follower

architectures, one vehicle acts as the leader and plans the global motion, while the followers maintain relative positions with respect to the leader. It is also possible to have an architecture where there are no leaders and followers, but each vehicle acts independently, and they simply collaborate by sharing the information of the discovered map: this approach is known as SLAM, a widespread technique applied to UAVs for the coverage of a map. Initially, the only communication that could take place between the Agents in this case served to avoid collisions; however, there are studies in this field that also use swarm logic to optimize times and paths [49]. The literature has investigated the Leader–Follower formation, proposing rather innovative solutions [50], but without ever considering not only the dynamics of the vehicle, but above all the fact that the vehicles are not equipped with infinitely powerful motors or brakes and that respond instantaneously to the request: for this reason, maintaining the trajectory and the formation at the same time is extremely challenging, and it is precisely one of the innovative aspects addressed by this work. Another important challenge is trying to understand which is the best formation for the swarm: while some prioritize safety and speed (platooning) [51], others might instead be more convenient in moments when the local swarm is exploring the space (diamond, circular) [52]. It is also important to evaluate whether the chosen formation is applicable to the available clearances, which are limited because of the obstacles [53, 54].

Chapter 3

Multi-Agent Simulation Framework in MATLAB/Simulink

3.1 Mission Scenario and Problem Statement

The considered scenario consists of a group of UGVs operating in an off-road environment. The mission is to reach a predefined goal position while progressively discovering the terrain and avoiding non-traversable regions. The environment is represented by a two-dimensional grid map enriched with semantic and physical information. Each cell contains information about terrain class, friction coefficient, slope, obstacle presence, traversability, and whether the cell is known by the swarm.

3.2 System Assumptions and Requirements

The simulation relies on a set of simplifying assumptions, summarized here. Perception and communication are idealized: each UGV knows its own position, velocity and heading without noise or estimation errors, discovers the map within a fixed circular visibility radius independent of terrain, weather or lighting, and shares its knowledge through a global memory with no latency, bandwidth limits or packet loss; once a cell is discovered, it stays known for the rest of the mission. Terrain and obstacles are simplified: all terrain properties (class, friction, slope, obstacle flag) come from the semantic map. The vehicle is modeled dynamically, but some subsystems are simplified for computational tractability: friction and elevation are assumed equal at the four wheels, the local slope is applied at the center of gravity, actuators and on-board computers do not degrade, and some blocks, such as EBD (Section 5.10.1), ABS (Section 5.10.2), motor cut-off (Section 5.9.2), rely on quantities that would not be directly measurable on a real vehicle. The mission, because of the nature of A* algorithm, is deterministic with a fixed goal: the same parameters and initial conditions always reproduce the same trajectories and metrics, the leader is

elected dynamically by the ranking mechanism, and the followers track target positions defined in the leader frame through fixed geometric offsets.
The simulation is considered successful if:

- The leader reaches or approaches the target position without crossing obstacle regions; the simulation can also be considered complete if the leader steps onto a cell near the goal, depending on how the simulation is interpreted. In a real-world application of this project—given that the vehicle might have no prior knowledge of the map—it could involve passing certain checkpoints without stopping (as is the case with exploration);
- the followers maintain a coherent formation with respect to the leader, with the necessary delays or advances with respect to the designated formation caused by the uncertainty of the map, the physicality of the vehicle and all other influencing parameters;
- the known map expands consistently with vehicle motion and always increasingly;
- The path planner reacts to terrain costs and unknown regions, adjusting as the known map expands. Having a planned path at the start of the simulation that passes through as-yet-unknown obstacles also indicates that the planner is receiving the correct amount of data;
- path tracker follows the input from the planner, avoiding sharp turns and overly risky trajectories.
- vehicle states and commands remain physically meaningful, and the simulation outputs can be analyzed through quantitative metrics and visual replay. This is a necessary condition for ensuring the consistency of the vehicle's Simulink model; subsequently, when implemented with real vehicles and maps, this condition will guarantee that the sensors are properly calibrated and functional.

3.3 General Architecture

Since the structure runs at various sample times, selected to balance cost and performance, it is important to identify a legend using the Simulink Debug function. The color coding adopted by Simulink to identify the sampling rates of the model is reported in Table 3.1, where each annotation is associated with its sample time in seconds.

Table 3.1: Simulink timing legend: sample times of the model in seconds. The base discrete rate is $T_s = 0.002$ s; slower rates are integer multiples of it.

Annotation	Color	Type	Sample time [s]	Multiplier
Cont		Continuous	–	–
D1		Discrete (base rate)	0.002	$\times 1$
D2		Discrete	0.03	$\times 15$
D3		Discrete	0.1	$\times 50$
Inf		Constant	–	–
M		Multirate	–	–

The simulation architecture is based on a top-level Simulink model containing multiple agent instances. Each agent contains a vehicle model and the necessary interfaces for planning, control, and logging.

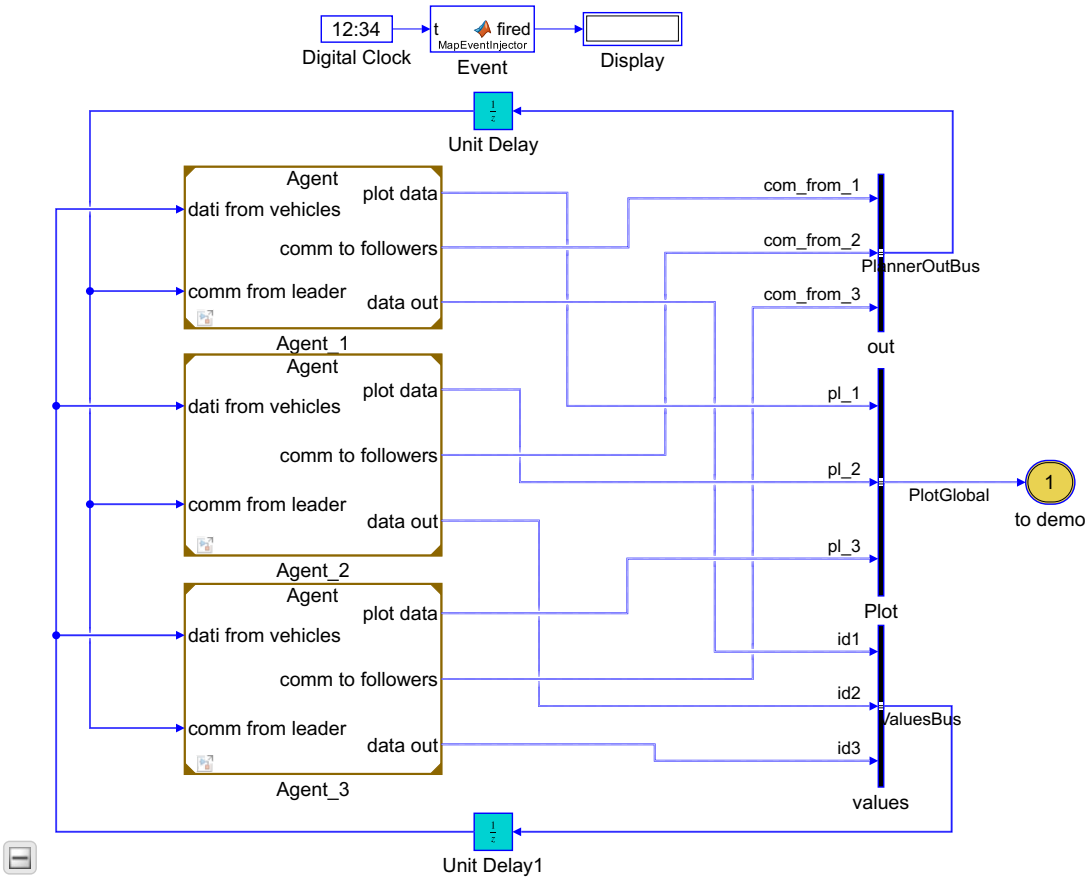


Figure 3.1: Simulink architecture of the UGV swarm. Located at `swarm_phase_5`

The three vehicles, coordinated by the architecture described in this chapter, advance in a leader–follower formation across the semantic terrain map while progressively discovering it. The complete simulation campaign is analyzed in Chapter 7.

The general structure is:

```
swarm_phase_5.slx
 Agent_1 (agent.slx)
  vehicle.slx
 Agent_2 (agent.slx)
  vehicle.slx
 Agent_3 (agent.slx)
  vehicle.slx
```

The same agent and vehicle models are reused through Simulink Model Reference blocks. Different initial conditions and parameters are assigned through model arguments.

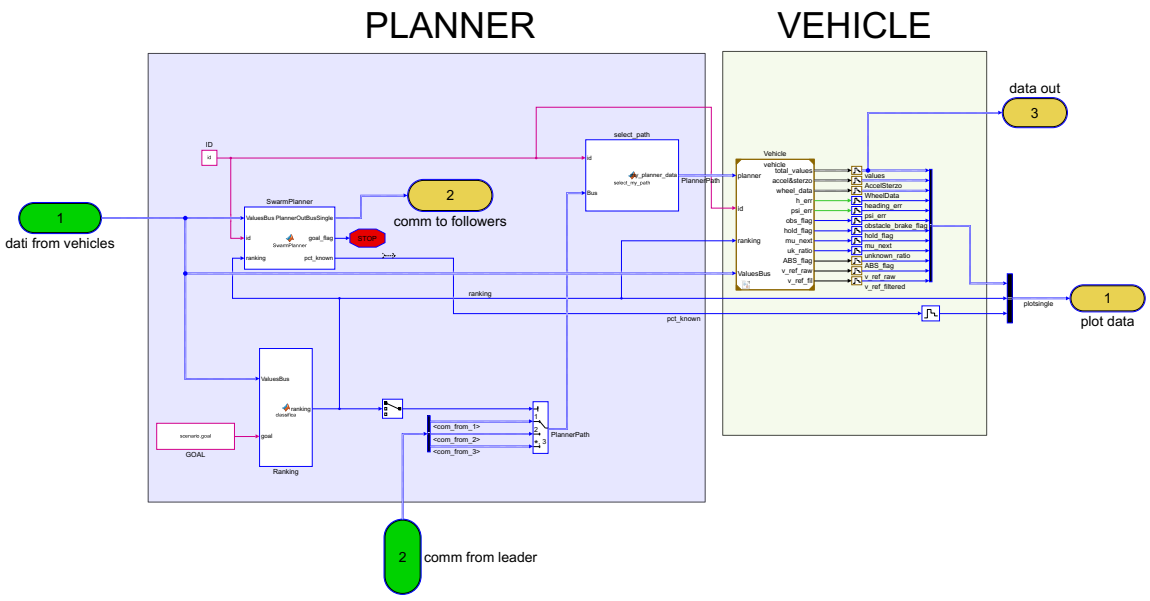


Figure 3.2: Agent structure. Located at `swarm_phase_5/(Agent)`.

Model Reference is used to avoid manually duplicating the vehicle model. This improves modularity and reduces the risk of inconsistencies between vehicle instances.The key idea is:

- all vehicles share the same model structure;
- each instance receives different initial conditions;
- parameters are passed through model arguments;
- the top model remains scalable to additional vehicles.

It is nonetheless possible to have vehicles with different parameters; one simply needs to change the arguments generated for each vehicle. For now, the only difference between the vehicles lies in their initial conditions x, y, and ψ.

Each agent represents one vehicle in the swarm. It receives global or shared information, computes or receives planner outputs, and simulates the corresponding vehicle behavior. The agent is identified by an integer ID. This ID can be used to select leader or follower behavior, route bus signals, and manage future failure scenarios. The Vehicle Model Interface is presented below in Table 3.2, summarizing the inputs and outputs of the vehicle model.

Table 3.2: Main inputs and outputs of the vehicle model interface.

Inputs	**Outputs**
Initial position and heading	Position x, y
Initial speed	Heading ψ
Initial wheel angular velocities	Total velocity
Terrain lookup tables	Accelerator/brake command
Controller parameters	Steering command
Path tracking information	Additional dynamic states, if required
Speed reference information	

3.4 Shared Map, Bus Architecture and Data Management

To reduce the number of outports and improve scalability, hierarchical Simulink buses are used.
Examples of bus structures include:

- vehicle state bus;
- planner output bus;
- actuator command bus;
- global plotting bus;
- agent-to-agent communication bus.

Table 3.3: Main bus structures used in the simulation.

Bus	**Purpose**
VehicleState	Stores position, heading, and speed.
ValuesBus	Collects vehicle states for all agents.
PlannerOutBus	Stores planned paths and planner-related outputs.
AccelSterzo	Stores accelerator/brake and steering commands.
PlotGlobal	Collects all signals required for post-processing and replay.

The simulation uses the MATLAB base workspace as a shared memory area. The map, scenario, vehicle structures, and history variables are stored globally.
This approach behaves as a blackboard architecture:

- agents and planner functions read the shared map;
- the known map is updated globally;
- the planner uses the most recent global knowledge;
- the visualization reads logged histories after the simulation.

The implemented system is not a fully decentralized swarm with independent local maps. Decision making is decentralized — each agent runs the same ranking and planning code — while the environment knowledge is shared through a single global map. It is therefore best described as a decentralized multi-agent simulation operating on shared global map knowledge. Below in Table 3.4 the Workspace hierarchical organization is shown.

Table 3.4: Workspace roles in the simulation framework.

Workspace	**Role**
Base workspace	Shared map, scenario, vehicles, and logging histories.
Top model workspace	Global parameters and model-level configuration.
Agent model workspace	Agent ID and model arguments.
Vehicle model workspace	Vehicle dynamics parameters and terrain lookup tables.

3.5 MATLAB Files

The framework is organized as a single call hierarchy rooted in the entry-point script `demo_swarm_phase4.m`. The tree below lists every MATLAB function and every Simulink model in the project: each file is indented one level to the right of the file that invokes it, so the depth of a node mirrors its position in the call stack. A `//` comment states, in pseudocode form, what the file does. Files reached through a Simulink block are tagged `[SIM]` (model reference) or `[Fn]` (MATLAB Function block); entries without a `.m` extension are MATLAB Function blocks embedded directly in the model. A file called from more than one place is repeated at each call site.

```
demo_swarm_phase4.m  // MAIN: build world, run sim, post-process
  define_swarm_buses.m  // declare Simulink.Bus objects
  load_motor_to_workspace.m  // load motor LUTs into workspaces
    motor_map.m  // build torque + efficiency maps
```

```
init_swarm_scenario.m  // assemble mission + parameters
  generate_terrain_map.m  // synth elevation, semantic, mu
  Tyre225_50_17_Comb.m  // Pacejka tire coefficient set
  build_cost_map.m  // grids -> A* traversal cost
visualize_terrain_map.m  // export terrain map PDFs
init_vehicle_agents.m  // build per-vehicle state structs
  get_local_terrain.m  // sample z, slope, mu at a pose
make_vref_filter.m  // design v_ref shaping filter
design_speed_mpc.m  // synthesize speed-tracking MPC
swarm_phase_5.slx [SIM]  // top model; sim() runs here
  MapEventInjector [Fn]  // clock-fed runtime event: obstacle spawn at t = 15 s
    apply_map_event.m  // stamp elliptical obstacle, override cost map
  Agent.slx [SIM]  // model ref: one self-contained agent
    classifica [Fn]  // weighted distance+ETA ranking, hysteresis
    swarm_planner_rank_to_id.m [Fn]  // ranks slots to IDs, run planner
      swarm_planner_helper2.m  // discovery, A*, formation logic
        build_cost_map.m  // recompute cost on known map
        update_friction_map.m  // mark wheel-compacted cells
        astar_path.m  // A* shortest path on cost grid
        smooth_path_spline.m  // spline-resample the raw path
        get_local_terrain.m  // sample terrain at probe pose
        select_local_heading.m  // obstacle-free heading choice
          get_local_terrain.m  // sample at candidate headings
          wrapToPiLocal.m  // wrap angle to (-pi, pi]
      write_rank_slot_to_id_slot.m  // reorder buffer rank -> ID
  vehicle.slx [SIM]  // model ref: 8-DOF vehicle dynamics
    PathRoleAntiUTurn [Fn]  // follower anti-U-turn guard
    PathTracker [Fn]  // look-ahead steering law
    SpeedReferenceGovernor [Fn]  // v_ref safety envelope
    obstacle_brake_from_base.m [Fn]  // emergency brake command
    md_fluxweak [Fn]  // torque deficit -> MPC measured disturbance
forceNBy4.m  // reshape logged signal to [N x 4]
getBusData.m  // read a leaf signal from a logged bus
  getBusNode.m  // descend the bus tree to a node
getBusTime.m  // read the time vector from a bus
  getBusNode.m  // descend the bus tree to a node
getfield_or.m  // nested field read with default
visualize_swarm_phase4.m  // replay snapshots + map discovery
plot_swarm_signals.m  // export logged-signal PDFs
  forceNBy4.m  // reshape logged signal to [N x 4]
  getBusData.m  // read a leaf signal from the bus
  getBusTime.m  // read the time base of the bus
plot_mu_compaction.m  // export friction-compaction PDFs
  update_friction_map.m  // build the post-compaction map
```

3.6 Execution Schedule and Orchestration

The simulation workflow is organized as follows:

1. define Simulink bus objects;
2. initialize the scenario and terrain map;
3. initialize vehicle agents;
4. populate base workspace and model workspaces;
5. configure model reference arguments;
6. run the Simulink simulation;
7. extract logged bus signals;
8. update MATLAB vehicle structures;
9. visualize trajectories, commands, and map discovery.

The plant is integrated by a fixed-step Simulink solver (`ode4`, $T_s^{\mathrm{8DOF}} = 2\,\mathrm{ms}$), while the discrete supervisory layer — ranking, planner and speed governor — runs on a slower task ($T_s^{\mathrm{sup}} = 100\,\mathrm{ms}$). Each agent is a self-contained model reference; all inter-vehicle communication flows through a single `shared_bus` delayed by one step (`UnitDelay`) to break algebraic loops. The whole scheme is decentralized: every agent runs the same code and reaches the same decisions from the broadcast state, with no central coordinator.

3.6.1 Initialization

The bootstrap phase builds the world, instantiates the agents and pushes the calibration data into each model workspace.

```
function init_demo()
// --- 1. Terrain map from an externally generated semantic image
img  <- imread("semantic_map.png")          // RGB, AI-authored
grid <- build_meshgrid(res = 1.0 [m/cell])
for each cell in grid do
class[cell] <- argmin_k || rgb(cell) - palette(k) ||   // nearest-color
    classifier
mu[cell]    <- class_to_friction(class[cell])
obs[cell]   <- class_to_obstacle(class[cell])
end
```

```
z         <- tilted_plane + sum(gaussian_dunes) + roughness   // 2.5D
    elevation
slope_deg <- atan( || grad(z) || )
map.known <- false(size(grid))  // nothing perceived yet

// --- 2. Mission + agent parameters (assignment block)
[start, goal, formation, A*-costs, 8DOF-params, Pacejka-coeffs] <-
    set_scenario()

// --- 3. Push calibration into Simulink model workspaces
load_motor_map()    // torque LUT: T = f(rpm, throttle)
for i = 1 : num_vehicles do
push_initial_conditions(agent[i])  // pose, wheel speeds, gains
end
end
```

The classifier is a minimum-distance lookup in RGB space, so it tolerates the tonal jitter in case of errors from the image recognizer from Python script. Each semantic class carries a nominal friction μ and an `obstacle` flag; the planner later turns these into traversal costs.

3.6.2 Model Arguments and Initial Conditions

Each vehicle instance receives a different set of initial conditions. This is achieved by passing model arguments from the top model to the agent model and then from the agent model to the vehicle model.
The relevant initial conditions include:

- initial x position;
- initial y position;
- initial heading;
- initial velocity;
- initial wheel angular velocities;
- initial roll-related states;
- initial terrain slope in the vehicle frame.

3.6.3 Supervisory and Plant-side Tasks

At every supervisory tick ($T_s^{\mathrm{sup}} = 100\,\mathrm{ms}$) each agent executes, in order, the ranking of Section 6.2 and the planner routine of Section 6.3: both read the `shared_bus` and write to the agent's own bus, so that every decision that does not need the fast integration rate is carried by the slow task. The plant-side blocks — collision check, path tracker, speed reference governor and longitudinal MPC, presented in Chapter 6 — run instead at the fast rate inside each `vehicle` model reference, together with the 8-DOF plant of Chapter 5.

3.6.4 Post-processing

After the run the logged signals are repacked into each agent's history and the final `known` map is recovered. A short report prints the simulated time, the explored fraction and the leader–goal distance, and an animated replay renders the perceived map with overlaid trajectories, the raw vs. smoothed paths, the speed traces and the leader's tire forces.

3.6.5 Logging, Output Extraction and Visualization

The simulation outputs are collected through a hierarchical plotting bus. This avoids the need for many independent outports and makes the architecture more scalable.
Logged quantities include:

- vehicle position (x, y);
- heading ψ;
- velocity V;
- accelerator/brake command ξ;
- steering command δ;
- known map percentage $\mathrm{pct}_{\mathrm{known}}$;
- planned paths $\mathbf{p}_{\mathrm{raw}}$ and $\mathbf{p}_{\mathrm{smooth}}$;
- known map history $\mathrm{map.known}(t)$.

The replay system provides a visual analysis of the mission evolution. It shows:

- complete semantic map and vehicle trajectories;
- progressive known map;

- raw and smoothed A* planner paths;
- vehicle speed profiles;
- accelerator/brake commands.

Chapter 4

Off-Road Environment Modeling

4.1 Overview of the Map Representations

The off-road environment is not described by a single map but by a family of grid maps, each capturing a different aspect of the terrain or of the knowledge the swarm has about it. They are all defined on the same regular (x, y) grid and share the same resolution, so that any cell can be looked up consistently across every representation. This section introduces them, while the rest of the chapter details how each one is built.

The starting point is the semantic map, which assigns a terrain class to every cell and is produced by an external AI segmentation module. From the recognized semantics and from a procedurally generated ground geometry derive the elevation map and the slope map, which describe the 2.5D shape of the ground, together with the friction map, which stores the per-cell friction coefficient μ obtained from the semantic class and corrected by the local slope. Together these layers form the truth map: the complete physical description of the terrain that the simulator integrates. The truth map is never fully available to the vehicles; each of them perceives only the cells inside its visibility radius, and the union of the perceived cells builds up the known map. Finally, the cost map condenses semantics, slope, obstacles and the unknown state into the single scalar traversal cost on which the planner operates. Table 4.1 summarizes all of these representations.

Table 4.1: Map representations used in the simulation framework.

Map	Meaning
Semantic map	Per-cell terrain class produced by the AI segmentation module (`semantic_map.png`).
Elevation map	Per-cell ground height z; the 2.5D geometry of the terrain.
Slope map	Per-cell slope angle θ_s computed from the elevation gradient.
Friction map	Per-cell friction coefficient μ from the semantic class, corrected by the local slope.
Truth map	Complete physical terrain (semantics, geometry, friction) integrated by the simulator.
Known map	Logical mask of the cells already observed by at least one vehicle.
Known history	Time history of the known map, stored for visualization and replay.
Cost map	Per-cell traversal cost used by the A* planner: semantic plus slope cost, with a penalty on the unknown cells.
Compaction map	Friction map updated by repeated vehicle passages, representing track formation.

4.2 AI-Based Semantic Map Generation

The semantic classification of the terrain is not authored by hand. It is produced by an external deep-learning module that performs semantic segmentation of an aerial image of the operating area, assigning a terrain class to every pixel. The simulation framework and this perception module are intentionally decoupled: the only interface between them is a single color-coded image, `semantic_map.png`, which the function `generate_terrain_map` reads and converts into the per-cell friction, obstacle and cost properties described above.

4.2.1 Segmentation Model and Training

The segmentation network is a SegFormer model [55], a Transformer-based architecture for dense pixel-wise prediction. The lightweight `b0` variant, pre-trained on the ADE20K scene-parsing dataset, is fine-tuned on the Semantic Drone Dataset [16], a collection of high-resolution nadir images of suburban and off-road scenes. The roughly two dozen original annotation classes — paved area, dirt, grass, gravel, rocks, water, pool, vegetation, tree, roof, wall, fence, vehicles, people and so on — are remapped onto the eight unified terrain classes used by the planner (unknown, hard_surface, bare_soil, grass, gravel_rock, water, vegetation_obstacle, obstacle) through a fixed lookup table. This remapping collapses perceptually distinct but functionally equivalent classes — for in-

stance every man-made structure becomes obstacle — and keeps the class set aligned with what the cost map actually requires.

Training is carried out at a resolution of 512×512 pixels with the AdamW optimizer, a cross-entropy loss, light data augmentation (horizontal and vertical flips, 90° rotations, brightness and contrast jitter) and an 80/20 train–validation split. The model is evaluated through the pixel accuracy and the mean Intersection-over-Union (mIoU) on the validation set, and the checkpoint achieving the best mIoU is retained.

4.2.2 Inference and Map Export

At inference time the target aerial image — in the present work a satellite frame of the chosen area — is segmented, and the resulting label map is recolored with the fixed eight-color palette of Table 4.2 and saved as `semantic_map.png`. Because a satellite frame is far larger than the network input, the image is processed as a grid of overlapping 512×512 tiles: each tile is segmented independently, the per-class probabilities are accumulated over the overlap regions, and the final label of every pixel is the arg-max of the fused probabilities. This tiled scheme preserves fine detail that a single global down-scaling would destroy. Figure 4.1 shows a representative input frame and the semantic mask predicted from it; the input frame is stored already rotated and mirrored so that its orientation matches the simulation reference frame.

The original number of classes (about twenty) was reduced to eight, because several of them were not relevant to this work and were therefore merged directly into the obstacle class.

```
function build_semantic_map(aerial_image)
// --- 1. Offline: fine-tune the segmentation network (run once)
data  <- SemanticDroneDataset()  // nadir RGB images + labels
data  <- remap_classes(data, original_to_unified)  // ~24 -> 8 terrain
    classes
model <- SegFormer_b0(pretrained = "ADE20K", num_classes = 8)
for epoch = 1 : 30 do
model <- train(model, data, opt = AdamW, loss = cross_entropy)
keep_best(model, metric = mean_IoU)  // on the 20% validation split
end

// --- 2. Online: segment the target aerial image
img  <- imread(aerial_image)  // RGB satellite/drone frame
prob <- zeros(num_classes, size(img))
for each tile (512 x 512, overlapping) in img do
logits      <- model( normalize(tile) )
prob[tile]  <- prob[tile] + softmax(logits)  // fuse logits on overlaps
```

```
end
class <- argmax_k prob  // per-pixel terrain class
rgb   <- recolor(class, palette)  // fixed 8-color palette
save(rgb, "semantic_map.png")  // read by generate_terrain_map
end
```

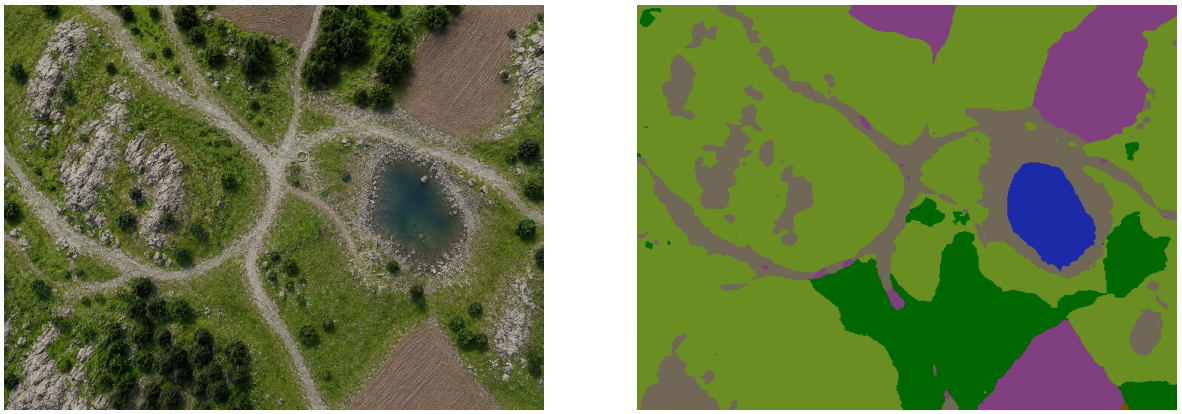
(a) Input aerial (satellite) image

(b) Predicted semantic mask

Figure 4.1: Semantic segmentation of an aerial image by SegFormer model: the RGB input frame (a) is segmented pixel-wise into the eight terrain classes (b).

The classifier on the MATLAB side is deliberately trivial — a nearest-color lookup in RGB space — so that the whole perception stage lives in the Python module and can be swapped without touching the simulator. In its current form this module is a proof of concept: it runs offline, on a single satellite frame, and with a model trained on a modest dataset, so the predicted mask still contains misclassified patches. This is acceptable here precisely because the semantic map is only an input to the framework, and the contribution of the thesis lies in the planning, coordination and dynamics layers that consume it. A further limitation is that the dataset used, although very accurate for its own purpose, does not contain enough images to train a model specialized in unpaved, off-road environments; it would instead perform better on closer-range pictures of urban scenes. For this reason the input was chosen as an image taken at a moderate distance from the ground, to which a larger scale was then applied to enlarge it, so as to make the task of the vehicles more challenging and longer. In any case, developing an AI able to interpret satellite imagery is not one of the main goals of this thesis, since more suitable solutions already exist that work with an on-board vehicle view rather than a top-down one. To show that the perception module is not tailored to one specific frame, it was run unchanged on a second, completely different aerial image. Figure 4.2 reports the input and the resulting semantic map for this second area: although the scene has different proportions of grass, gravel, vegetation and water — including an isolated pond absent from the first frame — the network still produces a coherent eight-class map that the simulator can consume directly. This confirms that the pipeline is general with respect to the

input map, and that the single frame used for the main mission was not hand-picked to make the segmentation succeed.



(a) Input aerial image (AI generated)

(b) Predicted semantic mask

Figure 4.2: Semantic segmentation of a second, different aerial image: the RGB input frame (a) is segmented into the same eight terrain classes (b). The module is run with no re-tuning, showing that it works on maps other than the one used for the main mission and is not calibrated for a single frame.

4.3 Semantic Terrain Representation

The terrain is represented by a grid map. Each cell is assigned a semantic class, such as hard surface, bare soil, grass, gravel, water, vegetation obstacle, or generic obstacle. Alternative map layouts were also considered, such as a hexagonal-cell grid or a radial-cell grid centered on the leader, which would allow a smoother planning; the classical square grid, although slightly less precise, offers the best trade-off between quality and cost. Each class is associated with physical and planning-related properties, as from Table 4.2.

Table 4.2: Semantic map classes: color legend, nominal friction, cell statistics on the simulated terrain, A* planning base cost and obstacle flag.

Color	Semantics (class)	μ	Cells (%)	Cost	Obstacle
(0, 0, 0)	unknown	0.50	0 (0.00)	100	No
(150, 80, 160)	hard_surface	0.85	20 741 (10.28)	1	No
(180, 100, 30)	bare_soil	0.55	1 735 (0.86)	10	No
(30, 110, 30)	grass	0.45	57 107 (28.32)	50	No
(120, 115, 90)	gravel_rock	0.60	0 (0.00)	30	No
(30, 60, 200)	water	0.10	0 (0.00)	10^6	Yes
(140, 165, 50)	vegetation_obstacle	0.30	21 243 (10.53)	200	No*
(240, 40, 100)	obstacle	0.00	100 840 (50.00)	10^6	Yes
Total		–	**201 666 (100.00)**	–	–

* Despite its name, the `vegetation_obstacle` class is defined with `obstacle = false` in `default_classes()`; it is therefore traversable, but heavily penalized by its base cost (200).

The nominal μ is the per-class value of `default_classes()`; the friction map applies a further per-cell slope correction. Unknown cells carry the optimistic default friction and are penalized at planning time through the `cost_unknown` overlay rather than through the base cost. The base cost is the relative `cost_per_class` value used by the A* planner; the effective cell cost also includes the semantic-boundary gradient and the slope penalty. Cell counts come from the nearest-color classification of `semantic_map.png` resampled onto the 366×551 planning grid.

These classes are only indicative: their precise definition is not critical to the present work, because the perception stage that produces the semantic map — described in the next section — is the component most readily replaced in a real deployment.

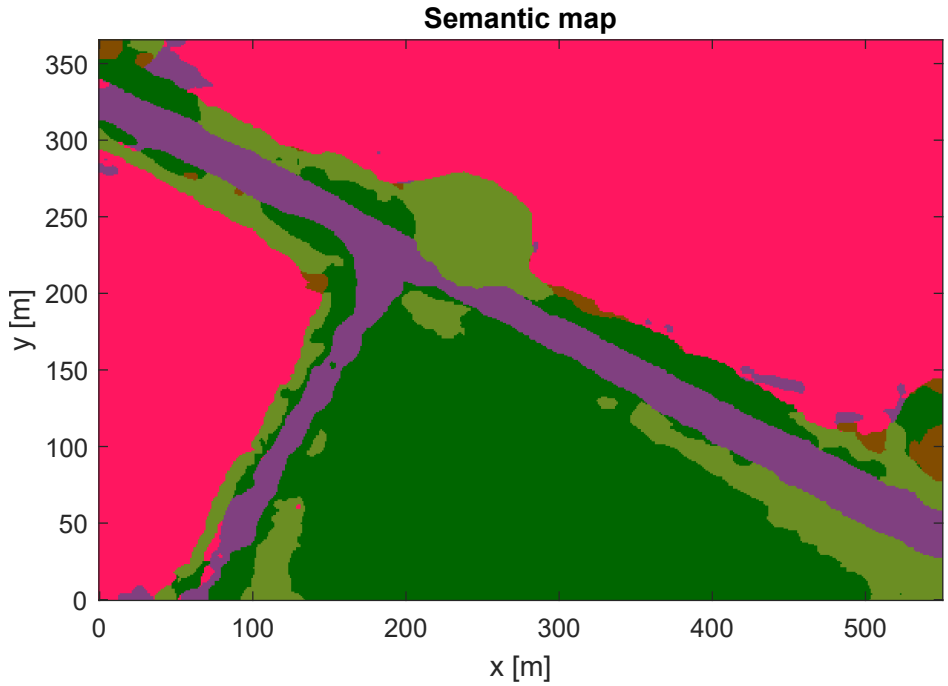


Figure 4.3: Semantic terrain map used by the simulator. The color-coded image `semantic_map.png` is resampled onto the planning grid; each color corresponds to a terrain class of Table 4.2.

4.4 Terrain Geometry and Slope

The environment is treated as a 2.5D map: each cell has an elevation value — the 3rd dimension — but the vehicle cannot, for obvious reasons, choose to navigate the map along z, since this is a coordinate constrained to (x, y). The terrain slope is computed from the elevation gradient:

$$s_x = \frac{\partial z}{\partial x}, \qquad s_y = \frac{\partial z}{\partial y} \tag{4.1}$$

The slope magnitude is:

$$s = \sqrt{s_x^2 + s_y^2} \tag{4.2}$$

and the corresponding slope angle is:

$$\theta_s = \arctan(s) \tag{4.3}$$

This information is used both for terrain traversability and for vehicle dynamic calculations.

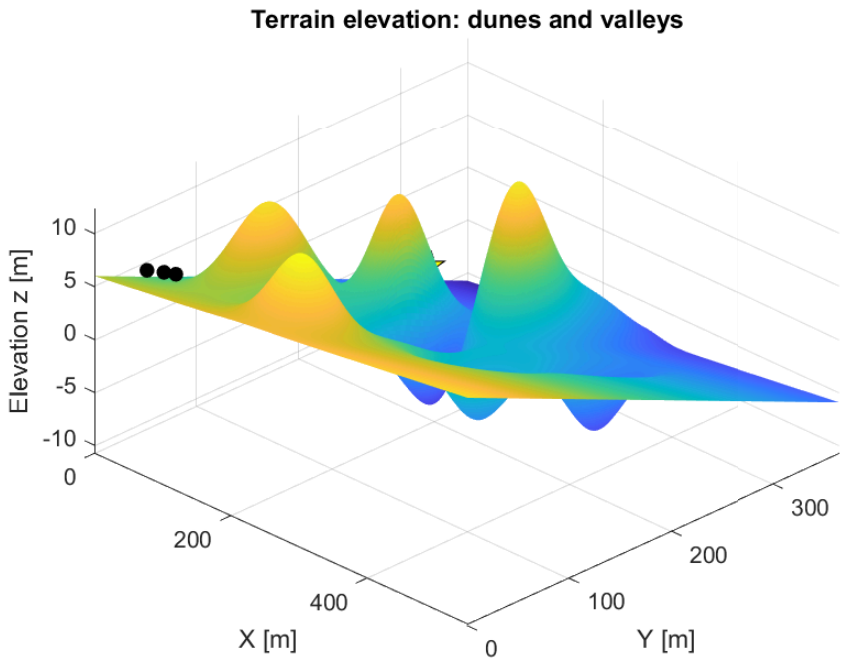


Figure 4.4: Terrain elevation of the simulated environment: a gently tilted base plane on which randomly generated dunes and valleys are superimposed.

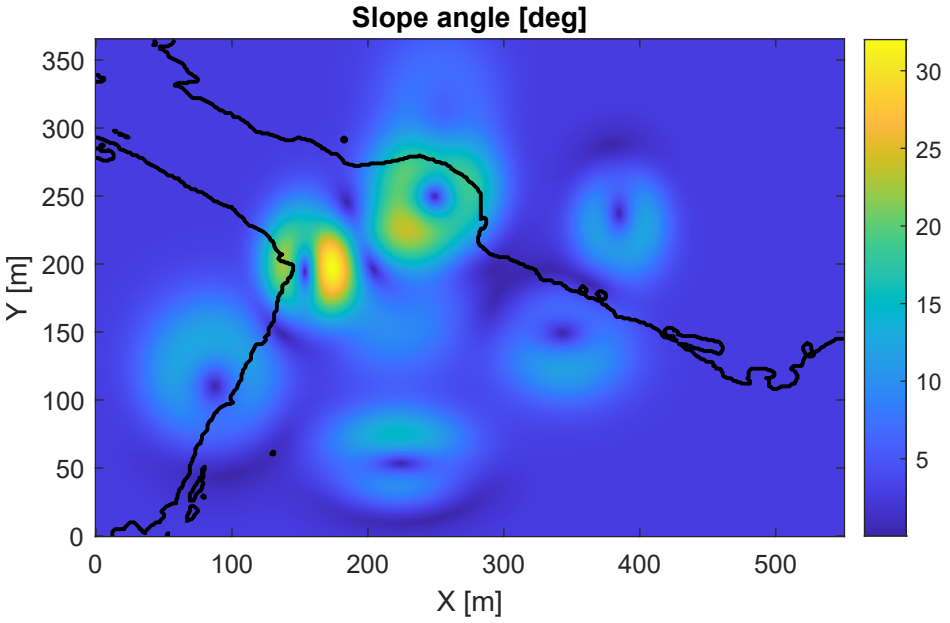


Figure 4.5: Slope angle θ_s, in degrees, obtained from the elevation gradient. The black contour outlines the obstacle regions.

4.5 Friction Map

The friction coefficient μ is assigned according to terrain class and corrected using local terrain slope. A generic expression can be written as:

$$\mu(x, y) = \mu_{class}(x, y) - k_s \theta_s(x, y) \tag{4.4}$$

where μ_{class} is the nominal friction coefficient associated with the semantic class, and k_s is a slope-related penalty coefficient, a fictitious term that further penalizes the vehicle when it crosses very steep terrain. Obstacle cells are assigned a negative or invalid friction value and are treated as non-traversable by the planner.

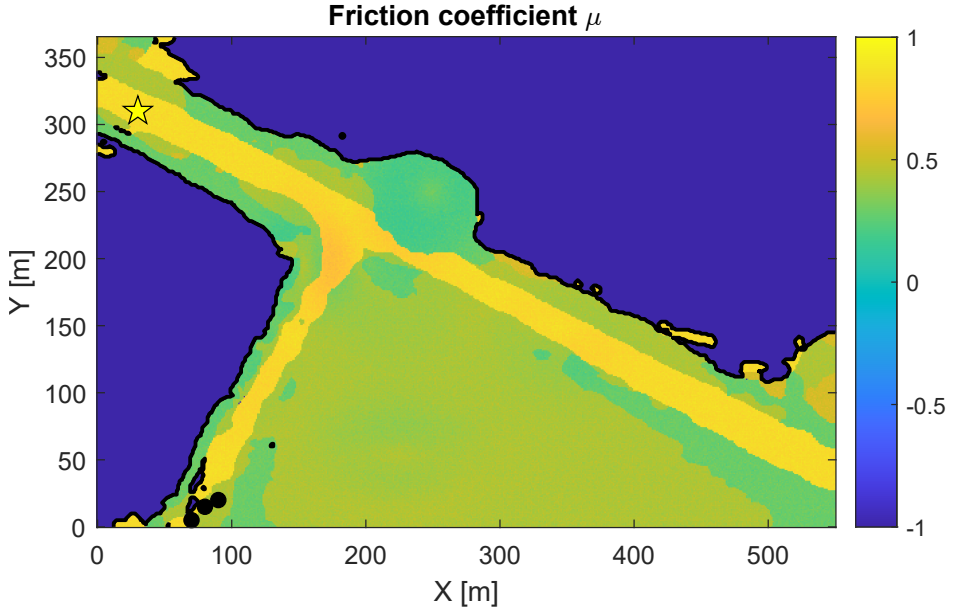


Figure 4.6: Friction coefficient μ assigned per cell from the semantic class and corrected by the local slope. The black contour outlines the obstacle regions, which carry an invalid friction value.

4.6 Known and Unknown Map

A central feature of the simulation is the distinction between the complete physical map and the map known by the swarm. The complete physical terrain is the truth map of Table 4.1; the portion of it that the swarm has actually discovered is the known map.

The known map is represented by a logical matrix:

$$K(x, y) = \begin{cases} 1, & \text{if the cell has been observed} \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

Some examples of the known/unknown map can be seen in the replay snapshots of Chapter 7.

4.6.1 Map Discovery Model

Each vehicle is assumed to observe the environment within a circular visibility radius R_{vis}, set to 30 m in all directions for the present project. A cell is considered visible from vehicle i if:

$$(x - x_i)^2 + (y - y_i)^2 \leq R_{vis}^2 \tag{4.6}$$

The global known map is updated using a cumulative logical OR operation:

$$K_{t+1} = K_t \vee V_t \tag{4.7}$$

where V_t is the set of cells visible at time t.

The known map is monotonic: once a cell has been observed, it remains known for the rest of the simulation. The known map is not exchanged between the vehicles through Simulink signals; it is updated at every step in a workspace shared by all of them, which effectively represents the knowledge sharing among the vehicles.

4.7 Cost Map

The cost map is the representation on which the A* planner actually operates. It condenses all the terrain information into a single scalar value per cell: the higher the value, the less desirable it is to drive a vehicle through that cell. It is built once, offline, by `build_cost_map` from the semantic map and the slope map, and stored in `map.cost_all`. The traversal cost of a cell is a property of the terrain, not of whether a vehicle has already perceived it: the penalty for crossing grass, gravel or bare soil is identical regardless of the order in which the cells are discovered, so there is no reason to rebuild the map at every step. For the same reason the precomputed map deliberately does not encode the `known` state, the unknown-cell penalty, or any friction term: these are applied per step inside the planner, and at run time the only quantity that keeps changing is the perceived fraction of the environment, i.e. the boolean mask `map.known`.

The routine `build_cost_map` assembles four contributions, in the order in which they are applied in the pseudocode below: a base cost per semantic class c_{class}, read from `scenario.planning.cost_per_class` and summarized in Table 4.2; a logarithmic gradient across the boundaries between classes; a slope penalty; and a final obstacle override. The base cost values are deliberately spread over several orders of magnitude, so that the planner strongly prefers hard surfaces, tolerates soil and grass, and treats water and solid obstacles as practically impassable.

```
function cost = build_cost_map(map, scenario)    // computed ONCE, offline
// --- 1. base cost per semantic class  (c_class)
```

```
for each cell (i,j) do
c_base[i,j] <- cost_per_class( class[i,j] )
end
cost <- c_base

// --- 2. logarithmic gradient across class boundaries (band R_grad = 4 cells)
num <- 0 ;  den <- 0
for each semantic class c do
d_c <- euclidean_distance_transform( class == c )  // distance, in cells, from class c
w   <- max( 0, (R_grad + 1 - d_c) / (R_grad + 1) ) // weight, decays with distance
num <- num + w * log( cost_per_class(c) )          // blend in log-space: costs span 1..1e6
den <- den + w
end
cost <- cost + ( exp(num/den) - c_base )           // signed correction ( good side +, costly side -)

// --- 3. slope penalty
cost <- cost + w_slope * max( 0, theta - theta_thr )   // theta_thr = 8 deg , w_slope = 0.5

// --- 4. obstacle override (LAST: the blend must never reduce an obstacle)
cost[ obstacle ] <- cost_obstacle                  // 1e6
cost <- max( cost, 1 )                             // numeric floor
end
```

The second contribution, the semantic gradient, makes the reference robust to small tracking errors by smoothing the otherwise abrupt jumps in cost between adjacent classes, everywhere except inside obstacles. For each cell the cost is blended in log-space with the cost of the nearby classes, using a distance-weighted average over a band of radius $R_{\text{grad}=4}$ cells, where the distance to each class is obtained with a Euclidean distance transform; a weight that decays linearly with distance gives full influence to the cell's own class and a vanishing influence at the edge of the band. The blend is performed in log-space because the costs span several decades (1 to 10^6), and its result is added to the base cost as a signed correction: it is positive on the good side of a boundary that borders an expensive class, and negative on the expensive side adjacent to a cheap class. Crucially, the obstacle override is applied after the blend, so the cost inside an obstacle is never lowered. The net effect is a soft potential that gently pushes the vehicles towards the

center of the traversable corridors: even if the tracker overshoots the reference slightly, the vehicle is biased to remain on easy terrain rather than drifting onto a costly class or an obstacle border.

The map of Figure 4.7 is the only one stored, in `map.cost_all`, and it is never rebuilt online. Later, at every replanning call, the planner routine `swarm_planner_helper2` takes a local copy of it and overlays the current perception before the A* search of the global planner is run, taking into account the cells that the swarm has not yet observed, as shown in Section 6.3. For a generic cell (x, y) the traversal cost is the sum of a semantic contribution and a slope contribution; an unobserved cell, however, cannot expose its real terrain properties and is therefore assigned a single fixed penalty:

$$C(x,y) = \begin{cases} C_{\text{unknown}}, & \text{if } K(x,y) = 0 \\ C_{\text{sem}}(x,y) + C_{\text{slope}}(x,y), & \text{if } K(x,y) = 1 \end{cases} \tag{4.8}$$

where C_{sem} is the cost associated with the semantic class of the cell — low for hard surfaces, higher for grass or gravel, and practically infinite for obstacles and water — and C_{slope} grows with the local slope angle θ_s, penalizing steep cells. The term $K(x, y)$ is the known-map mask of Equation (4.5): a cell that has not yet been observed ($K = 0$) does not contribute its true semantic or slope cost, but the uniform unknown cost $C_{\text{unknown}} = 10^3$, a value far above the base class costs, so that the search prefers already-explored corridors whenever one exists while the underlying terrain cost is left untouched.

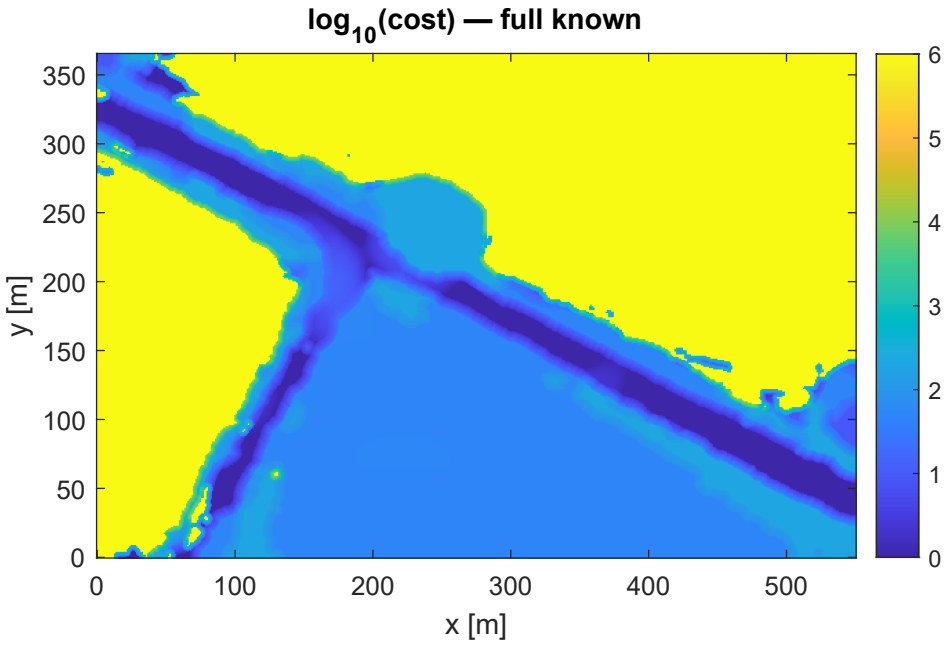


Figure 4.7: Cost map $C(x, y)$ of the simulated environment, computed with the whole terrain assumed known. Every cell carries the sum of its semantic and slope cost; obstacle and water cells carry a practically infinite, no-go cost.

The cost map of Figure 4.7 is shown for the complete environment, that is with $K(x, y) =$

1 on every cell. For this reason the C_{unknown} branch of Equation (4.8) never appears in the figure: each cell displays its true semantic-plus-slope cost. The effect of the unknown penalty can instead be observed indirectly in the replay snapshots of Chapter 7. In the early stages of a mission most of the map is still unobserved and is therefore filled with the uniform value C_{unknown}; not knowing where the obstacles are, the A* search finds no reason to deviate and simply aims straight at the target. As the vehicles advance and discover the terrain, the real per-cell costs replace the unknown value and the planned path starts to bend around the obstacles that have become visible.

4.8 Terrain Compaction and Multi-Pass Effect

The map is updated to account for repeated vehicle passages. The local friction coefficient is modified in the area covered by the vehicle footprint. This provides a simplified representation of terrain compaction or track formation, which are real and quantifiable phenomena, addressable through higher-fidelity terramechanics models of greater complexity. The update depends on:

- vehicle position;
- vehicle heading;
- footprint width;
- footprint length;
- compaction factor;
- maximum allowable friction coefficient.

The friction increase follows a multiplicative model applied at every passage over each traversable cell:

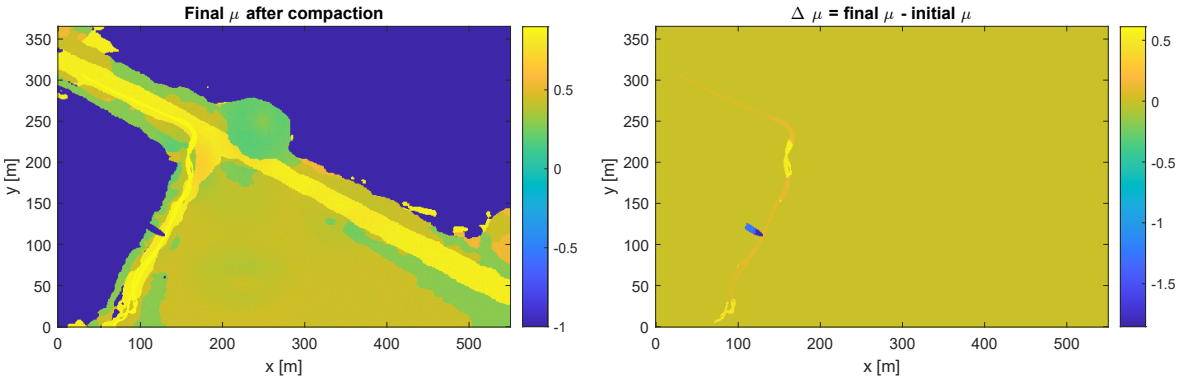
$$\mu^{(n)} = \min\left(\mu_{\max},\ \alpha \cdot \mu^{(n-1)}\right) \tag{4.9}$$

where α is the compaction factor ($\alpha = 1.20$ in this work) and n is the passage count.

```
1        function map = update_friction_map(map, pose, width, length, alpha,
     mu_max)
2        [x0, y0, psi] <- pose
3        for each cell (i,j) in map do  // global to local frame
4        x_loc <-  cos(psi)*(X[i,j]-x0) + sin(psi)*(Y[i,j]-y0)
5        y_loc <- -sin(psi)*(X[i,j]-x0) + cos(psi)*(Y[i,j]-y0)
6        end
7        mask <- |x_loc| <= length/2  AND  |y_loc| <= width/2  AND
    traversable
```

```
        mu[mask]             <- min(mu_max, alpha * mu[mask])
        visited_count[mask] <- visited_count[mask] + 1
        end
```

Although the simulation campaign is reported later, Figure 4.8 already anticipates the effect of this model on the results obtained after a complete swarm mission. It shows the friction map once the three vehicles have reached the goal: the cells repeatedly traversed by the swarm form higher-friction trails, while the rest of the map remains unchanged.



(a) Final friction coefficient μ (b) Friction increment $\Delta\mu$

Figure 4.8: Terrain compaction after a complete swarm mission — an anticipation of the post-simulation results. (a) Final friction coefficient μ updated by `update_friction_map`: the paths repeatedly traveled by the three vehicles appear as higher-friction trails. (b) Friction increment $\Delta\mu = \mu_{\text{final}} - \mu_{\text{initial}}$, which is non-zero only on the cells actually traversed by the swarm.

A $\Delta\mu$ ellipse is present in Figure 4.8b because that is a part of the map that changes over time, as explained in Section 7.1. This component is a simplified approximation and can be extended in future work using the terramechanics models presented in 2.3.

Chapter 5

Vehicle Dynamics Model

The controllers designed in the following chapters can only be as effective as the vehicle they act upon: they require a plant faithful enough to reproduce the dynamic effects that actually constrain the motion — tire friction limits, load transfer, actuator saturation — while remaining light enough to simulate a three-vehicle swarm at an acceptable computational cost. This chapter presents the 8-DOF dynamic model adopted to meet this trade-off, describing in turn its degrees of freedom and parameters, the tire model, the roll and load-transfer dynamics, the electric motor map, the braking logic and the four-wheel steering system. To provide evidence that the model behaves consistently and returns physically meaningful signals, a set of quantities is logged as a function of time and collected in Section 5.12; all the traces refer to the mission described in Section 7.2.

5.1 Role of the Vehicle Model

The vehicle model is used to evaluate the dynamic feasibility of the planned trajectories. While a kinematic model may be sufficient for preliminary path planning, an off-road UGV operating on variable terrain requires a more detailed representation of dynamics.

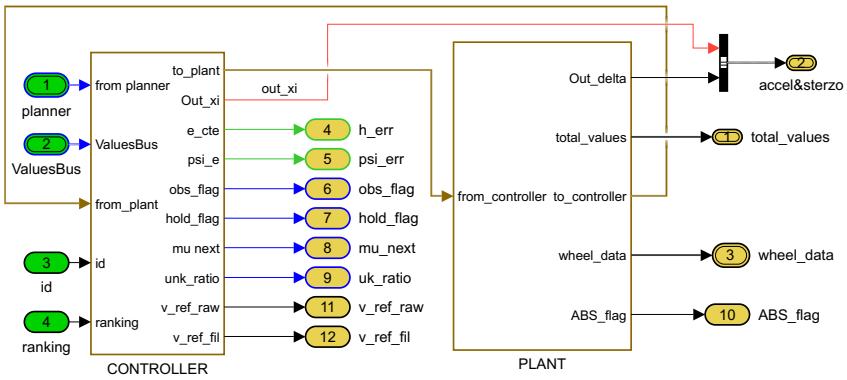


Figure 5.1: `Vehicle` block, composed of `Plant` and `Controller`. Located at `swarm_phase_5/(Agent)/(Vehicle)`.

Figure 5.1 shows the plant studied in this section, which is a simplified representation of the vehicle. Among the outputs, in addition to the values used to generate the plots presented in this work, there is a set of signals sent to the controller. In the development of a real controller, these signals would be replaced by measurements provided by on-board sensors. The vehicle model adopted in this work is used to represent the dynamic behavior of

- longitudinal motion;
- lateral motion;
- yaw dynamics;
- roll dynamics;
- wheel rotational dynamics;
- tire forces;
- vertical load transfer;
- terrain-dependent friction.

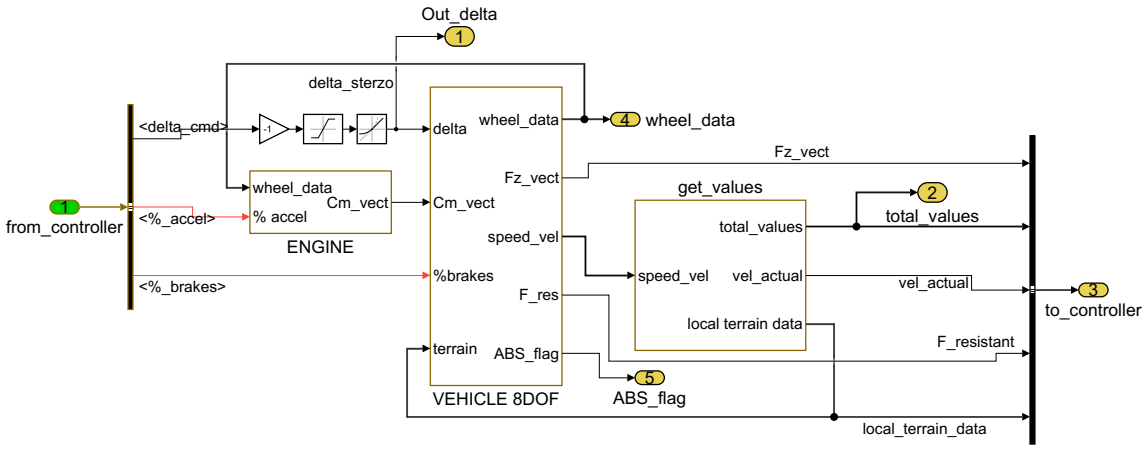


Figure 5.2: The `PLANT` subsystem of the vehicle model. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT`: the steering, powertrain and braking chains feed the 8-DOF rigid-body and wheel dynamics that turn the commands into the vehicle motion.

The `PLANT` block represents the vehicle's actual dynamic behavior, with outputs measured via on-board sensors. To ensure the highest possible fidelity to real-world behavior, the Simulink time step for this block should be as close to infinitesimal as possible; this involves actual dynamics, as the block does not emulate an ECU or electronic components.

5.2 Degrees of Freedom

The vehicle model is the 8-DOF double-track model whose degrees of freedom were detailed in Table 2.2: longitudinal, lateral and yaw chassis motion, roll motion, and the rotational dynamics of the four wheels (FL, FR, RL, RR).

For the proposed off-road UGV swarm, an 8-DOF model offers an appropriate balance between physical fidelity and computational cost: unlike a kinematic or 4-DOF bicycle model, it captures the left–right load transfer produced by lateral acceleration, side slopes, and steering, which strongly affects the stability of small vehicles with a high center of mass and a narrow track. By including roll and yaw dynamics, individual wheel spin, tire slip, and vertical load transfer, it makes it possible to assess whether a terrain-aware trajectory can actually be followed without excessive slip, instability, or unrealistic commands, and it provides the wheel-level detail required by future terramechanics extensions such as BWR soil interaction. A higher-fidelity 14-DOF model, such as the one in [3], would additionally describe pitch, heave, and suspension travel, but this complexity is unnecessary for a first implementation of the swarm framework. It is worth noting that the wheel-related degrees of freedom are not six free Cartesian coordinates but reduce, in practice, to vertical suspension travel and wheel spin, since steering is imposed

as a control input and camber, toe, and wheel-center migration are constrained by the suspension geometry.

In compact algorithmic form, the per-step update performed by the plant block is the following: the inputs are δ_{cmd}, motor torque, interpolated road slope, while the outputs are updated state $(x, y, \psi, v, \dots)$, wheel forces. For each of the four wheels the block computes slip ratio σ and slip angle α, feeds them with the vertical load F_z, friction μ and camber into the Pacejka '96 model, and integrates the resulting forces into the 8-DOF rigid body (longitudinal, lateral, yaw, roll and four wheel spins).

```
function state = vehicle_8DOF(delta_cmd, T_motor, slope)
foreach wheel in {FL, FR, RL, RR} do
[sigma, alpha] <- slip(wheel, state)
[Fx, Fy]       <- pacejka96(sigma, alpha, Fz, mu, camber)
end
state <- integrate( newton_euler(Fx, Fy, slope), Ts )
end
```

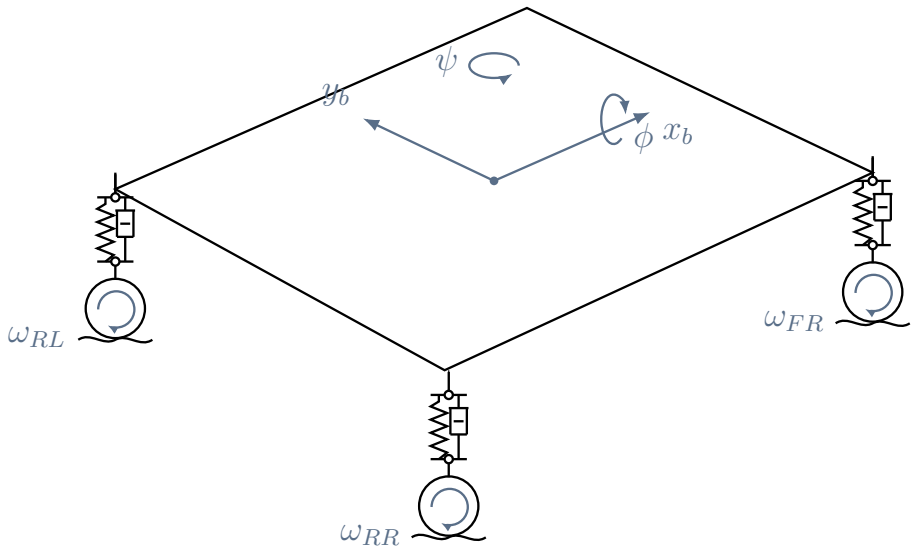


Figure 5.3: Schematic representation of the 8-DOF vehicle model

5.3 Vehicle Geometry and Parameters

The main vehicle parameters are summarized in Table 5.1. In the code these parameters are assigned through the function `init_swarm_scenario()`.

Table 5.1: Vehicle model parameters (8-DOF).

Symbol	Value	Unit	Description
Geometry, mass and inertia			
m	1300	kg	Vehicle mass
L	1.5	m	Wheelbase ($a + b$)
a	0.6	m	Distance from CG to front axle
b	0.9	m	Distance from CG to rear axle
T_f	1.5	m	Front track width
T_r	1.5	m	Rear track width
h_G	0.4	m	Height of the center of gravity
J_z	1800	kg m^2	Yaw moment of inertia
I_x	450	kg m^2	Roll moment of inertia (sprung mass)
g	9.81	m/s^2	Gravitational acceleration
Roll dynamics			
h_{Rf}	0.08	m	Front roll-center height
h_{Rr}	0.12	m	Rear roll-center height
h_{cr}	0.304	m	CG-to-roll-center distance
K_ϕ	130000	N m/rad	Total roll stiffness
K_{sf}	39000	N m/rad	Front suspension roll stiffness
K_{sr}	39000	N m/rad	Rear suspension roll stiffness
K_{af}	26000	N m/rad	Front anti-roll-bar stiffness
K_{ar}	26000	N m/rad	Rear anti-roll-bar stiffness
C_ϕ	15000	N m s/rad	Roll damping coefficient
Aerodynamics			
A_f	1.8	m^2	Frontal area
C_x	0.28	–	Aerodynamic drag coefficient
ρ	1.3	kg/m^3	Air density
h_a	0.4	m	Aerodynamic center-of-pressure height
tire			
R_0	0.317	m	Unloaded wheel radius
J_{ns}	0.8	kg m^2	Wheel moment of inertia
K_v	2.90×10^5	N/m	tire vertical stiffness
C_p	500	N s/m	tire vertical damping
L_x	0.055	m	Longitudinal relaxation length
L_y	0.5	m	Lateral relaxation length
K_{dF}	0	rad/N	Front lateral compliance-steer coeff.
K_{dR}	0	rad/N	Rear lateral compliance-steer coeff.
τ_F	-0.1	deg	Front static toe
τ_R	0.1	deg	Rear static toe
γ_{0F}	0	deg	Front static camber
γ_{0R}	-0.39	deg	Rear static camber
f_0	0.0041	–	Rolling resistance, constant term
f_2	2.051×10^{-7}	s^2/m^2	Rolling resistance, quadratic term

5.4 Wheel-Center Velocities and Slip Angles

The `Velocità Mozzi` subsystem transforms the planar body state $(u, v, \dot{\psi})$ into the longitudinal and lateral velocity components of each wheel center, by applying the rigid-body velocity transport from the center of gravity to the four wheel hubs:

$$u_{\mathrm{FL}} = u - \dot{\psi}\,\tfrac{T_f}{2}, \qquad v_{\mathrm{FL}} = v + \dot{\psi}\,a, \tag{5.1}$$

$$u_{\mathrm{FR}} = u + \dot{\psi}\,\tfrac{T_f}{2}, \qquad v_{\mathrm{FR}} = v + \dot{\psi}\,a, \tag{5.2}$$

$$u_{\mathrm{RL}} = u - \dot{\psi}\,\tfrac{T_r}{2}, \qquad v_{\mathrm{RL}} = v - \dot{\psi}\,b, \tag{5.3}$$

$$u_{\mathrm{RR}} = u + \dot{\psi}\,\tfrac{T_r}{2}, \qquad v_{\mathrm{RR}} = v - \dot{\psi}\,b, \tag{5.4}$$

where a and b denote the longitudinal distances between the center of gravity and the front and rear axles, respectively, while T_f and T_r denote the front and rear track widths. The effective steering angle at each wheel is obtained by superimposing three contributions on the steering command δ_{cmd}: the static toe angle δ_0 and a compliance-steer contribution proportional to the lateral tire force, $K_d F_y$. This term accounts for the elastic rotation of the wheel about the steering axis caused by the finite compliance of the suspension and steering linkage:

$$\delta_{\mathrm{F}\bullet} = \delta_{\mathrm{cmd},F} + \delta_{0,F} + K_{dF}\,F_{y,\mathrm{F}\bullet}, \tag{5.5}$$

$$\delta_{\mathrm{R}\bullet} = \delta_{\mathrm{cmd},R} + \delta_{0,R} + K_{dR}\,F_{y,\mathrm{R}\bullet}. \tag{5.6}$$

The effective steering angle obtained in this way, together with the longitudinal and lateral velocity components at the wheel hub, defines the tire slip angle:

$$\alpha = \delta - \arctan\!\left(\frac{v_{\mathrm{cr}}}{u_{\mathrm{cr}}}\right), \tag{5.7}$$

where v_{cr} and u_{cr} are the lateral and longitudinal velocity components of the wheel center expressed in the wheel reference frame.

The toe angle is the static angle δ_0 between the wheel plane and the longitudinal axis of the vehicle. According to the adopted convention, $\delta_0 < 0$ corresponds to toe-in, i.e. convergent wheels. In the model, $\delta_{0,F} < 0$ and $\delta_{0,R} = -\delta_{0,F} > 0$ are imposed.

The effect on vehicle balance can be interpreted through the equivalent cornering stiffness of the axle, C_α. A static toe angle preloads the tires of an axle with opposite slip angles: during cornering, the inner and outer wheels operate at different values of α, and, due to the progressive saturation of the $F_y(\alpha)$ characteristic, the useful net lateral force of the axle changes. Toe-in tends to increase the effective C_α of the axle at small slip angles, since the wheels oppose lateral deviation, whereas toe-out tends to reduce it.

By applying the understeer-gradient relation

$$K_{us} = \frac{m}{L}\left(\frac{b}{C_{\alpha F}} - \frac{a}{C_{\alpha R}}\right), \tag{5.8}$$

the resulting behavior can be inferred as follows:

	Toe-in	**Toe-out**
Front axle	$K_{us} \downarrow$	$K_{us} \uparrow$
Rear axle	$K_{us} \uparrow$	$K_{us} \downarrow$

Table 5.2: Effect of static toe on the understeer gradient K_{us}.

The adopted configuration, with front toe-in and rear toe-out, therefore shifts the vehicle balance towards oversteer on both axles, favoring turn-in responsiveness over directional stability. This choice is particularly relevant in the considered operating environment, where rapid heading correction and maneuverability are prioritized.

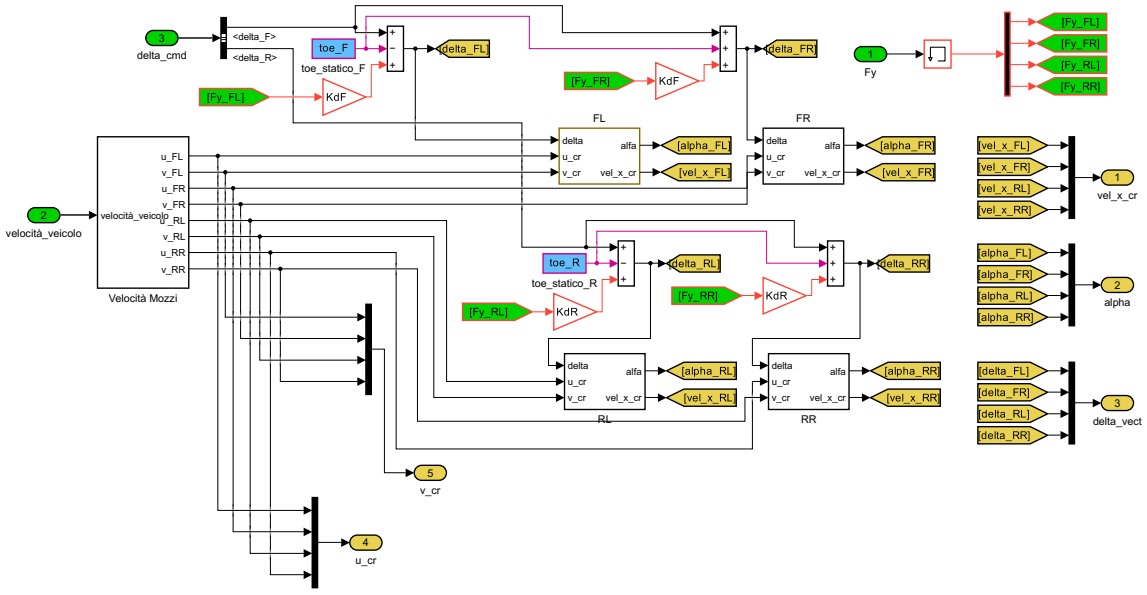


Figure 5.4: Simulink computation of the wheel-center velocities and tire slip angles: the chassis states $(u,\, v,\, r)$ and the per-wheel steering angles are combined to obtain the velocity components at each wheel center and the slip angles α_w fed to the tire model. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT/VEHICLE8DOF/VelocitaCentriRuotaeAngolidiDeriva`

5.5 Tire–Terrain Interaction

The tire–terrain interaction is a central component of the 8-DOF vehicle model. It defines the mechanism through which each wheel generates longitudinal and lateral forces as a function of the local operating conditions, namely slip ratio, slip angle, vertical load and terrain-dependent friction. Therefore, this subsystem provides the physical link between the vehicle dynamics, the control inputs and the semantic terrain map.

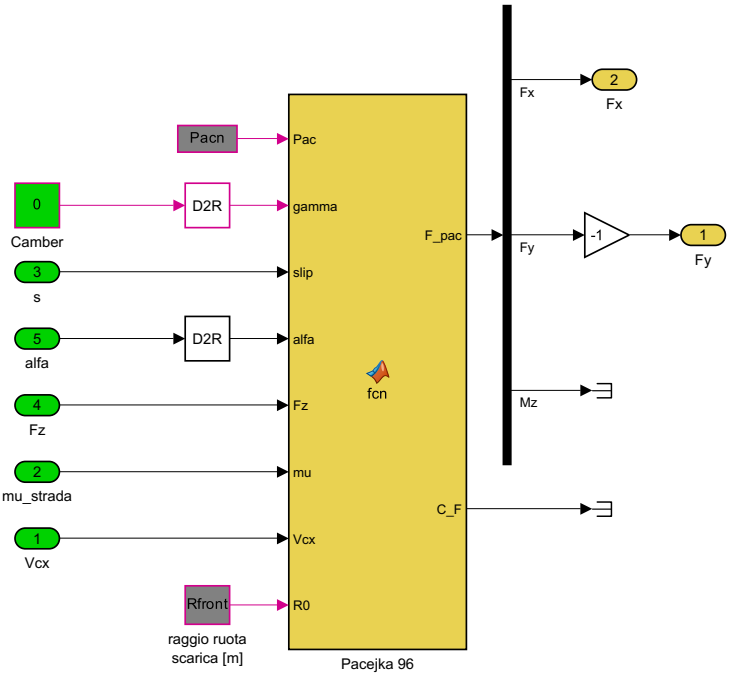


Figure 5.5: Tire–Terrain Interaction main architecture. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT/VEHICLE8DOF/InterazionePneumaticiTerreno/PneumaticoFL`; an identical structure is also present for the FR, RL and RR wheels.

while the first block calculates the slip using the formula

$$\sigma = \begin{cases} 1 - \dfrac{V_x}{\omega R} & \text{if } \omega R \geq V_x \quad \text{(traction, } \sigma \geq 0\text{)} \\ \dfrac{\omega R}{V_x} - 1 & \text{if } \omega R < V_x \quad \text{(braking, } \sigma < 0\text{)} \end{cases} \tag{5.9}$$

including through the use of ϵ to avoid division by zero, it is more interesting to briefly analyze what happens in the other two blocks.

5.5.1 Tire Model

Tire forces are computed using a tire model dependent on slip angle, slip ratio, vertical load, and friction coefficient. The terrain friction coefficient is obtained from the terrain map and used to scale tire force generation.

The generic dependency can be written as:

$$F_x = f_x(\sigma, \alpha, F_z, \mu) \tag{5.10}$$

$$F_y = f_y(\sigma, \alpha, F_z, \mu) \tag{5.11}$$

where σ is the longitudinal slip ratio, α is the slip angle, F_z is the vertical load, and μ is the local friction coefficient.

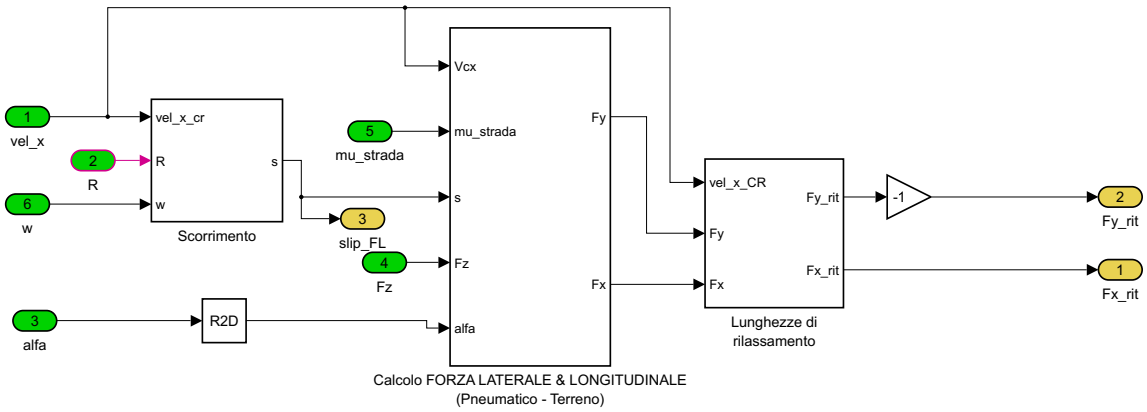


Figure 5.6: Main architecture of Pacejka Magic Formula contact forces block. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT/VEHICLE8DOF/InterazionePneumaticiTerreno/CalcoloFORZALATERALE&LONGITUDINALE(PneumaticoTerreno)`; an identical structure is also present for the FR, RL and RR wheels.

5.5.2 Relaxation Length

The tire is not an instantaneous force transducer: the carcass possesses structural stiffness k_i and viscous damping, and the contact force is established only after the tread has traveled a certain distance. The elastic component determines the relaxation length L_i ($i = x, y$), while the viscous component damps the transient.

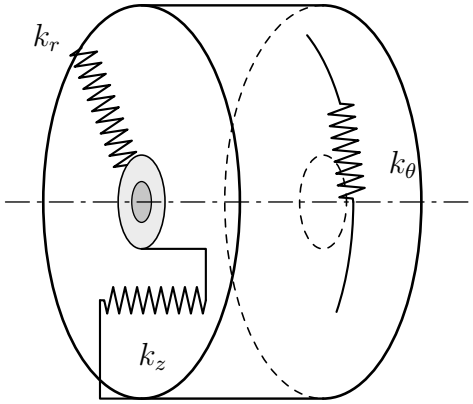


Figure 5.7: Schematic representation of tire elasticities: radial (k_r), tangential (k_θ), and axial (k_z).

The lag is modeled by a first-order differential equation in which the instantaneous force F_i tracks the steady-state value $F_{i,\mathrm{Pacejka}}$ provided by the Magic Formula

$$\tau_i \dot{F}_i + F_i = F_{i,\mathrm{Pacejka}} \qquad \Longleftrightarrow \qquad F_i = F_{i,\mathrm{Pacejka}} - \tau_i \dot{F}_i, \qquad i = x, y. \tag{5.12}$$

The time constant is the ratio of the relaxation length to the rolling speed, a value common to both directions, since the phenomenon is governed by the distance traveled

$$\tau_i = \frac{L_i}{V_x}. \tag{5.13}$$

Assuming the carcass stiffness k_i is independent of forces, speed, and deformation, the relaxation length is the ratio of the cornering stiffness C_i to the structural stiffness

$$L_i = \frac{C_i}{k_i}, \qquad C_x = \left.\frac{\partial F_x}{\partial \sigma}\right|_0, \qquad C_y = \left.\frac{\partial F_y}{\partial \alpha}\right|_0. \tag{5.14}$$

The ratio F_i/k_i, on the other hand, represents the carcass deformation within the contact patch, a quantity distinct from, and generally much smaller than, L_i. In response to a step input (driving torque for F_x, steering angle for F_y), the solution is an exponential function that approaches an asymptote. $F_{i,\mathrm{Pacejka}}$:

$$F_i(t) = F_{i,\mathrm{Pacejka}} \left(1 - e^{-t/\tau_i}\right). \tag{5.15}$$

The slope at the origin is $F_{i,\mathrm{Pacejka}}/\tau_i$—inversely proportional to τ_i—and the corresponding tangent intersects the asymptote exactly at $t = \tau_i$. After $5\tau_i$, the force has reached 99.3% of the steady-state value, and the transient phase is considered complete.

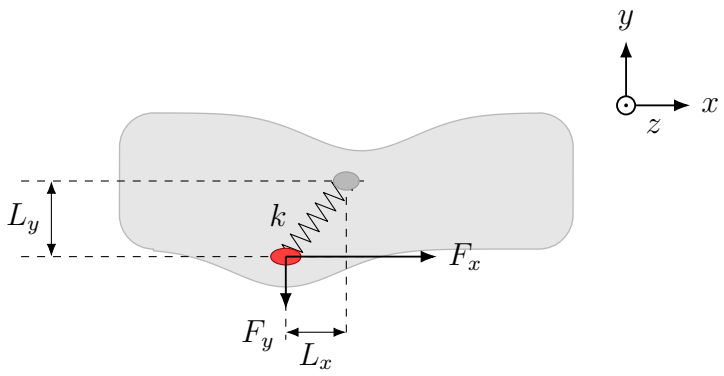


Figure 5.8: Schematic representation of tire relaxation.

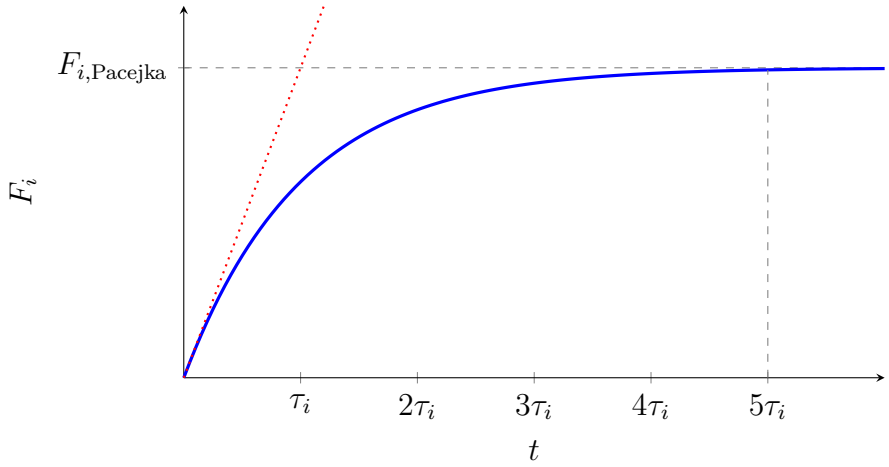


Figure 5.9: tire force response to a step input: first-order exponential behavior with time constant $\tau_i = L_i/V_x$. The tangent at the origin intersects the asymptote at $t = \tau_i$; after $5\tau_i$, the transient is considered exhausted (99.3%).

5.6 Roll Dynamics and Load Transfer

Roll dynamics are relevant in off-road conditions because lateral slope, steering maneuvers, and transient lateral acceleration can modify the vertical load distribution.

A simplified roll equation can be expressed as:

$$I_x\ddot{\phi} + C_\phi\dot{\phi} + K_\phi\phi = M_\phi \tag{5.16}$$

where I_x is the roll inertia, C_ϕ is the roll damping coefficient, K_ϕ is the roll stiffness, and M_ϕ is the roll moment.

The physical origin of this equation is illustrated by the free-body diagram of the sprung mass in Figure 5.10. The body is free to rotate by the roll angle ϑ about the roll center RC, and the distance between RC and the center of gravity G is denoted H_{roll}. The inertial contributions $m\,a_{y,G}$ and $m\,a_{z,G}$, the weight mg and the roll-inertia couple $I_{xG}\ddot{\vartheta}$

are balanced by the elastic and viscous reaction $K\vartheta + C\dot{\vartheta}$ of the suspension and by the constraint forces $F_{y,RC}$ and $F_{z,RC}$ transmitted at the roll center.

Vertical load transfer affects tire force availability and therefore the dynamic behavior of the vehicle. Figure 5.11 shows the corresponding axle free-body diagram. The roll moment M_r and the force $F_{y,RC}$ acting at the roll center, together with the roll-center height H_{RC} and the track width T, produce a lateral load transfer $\Delta F_{z,lat}$ that adds to or subtracts from the static wheel loads $F_{z,st}$, increasing the vertical force on the outer wheel and reducing it on the inner one.

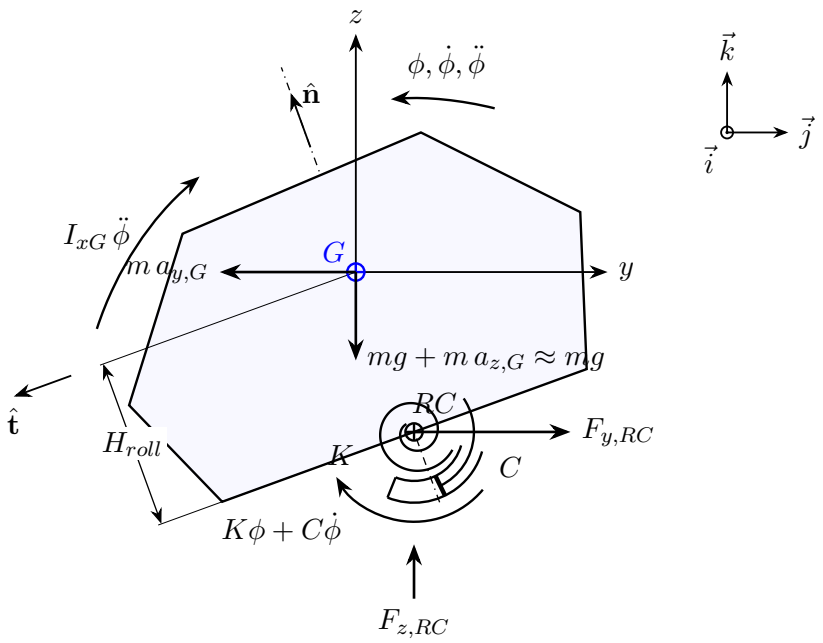


Figure 5.10: Free-body diagram of the sprung mass in roll motion. The body rotates by the roll angle ϕ about the roll center RC; the suspension is modeled by the roll stiffness K and the roll damping C.

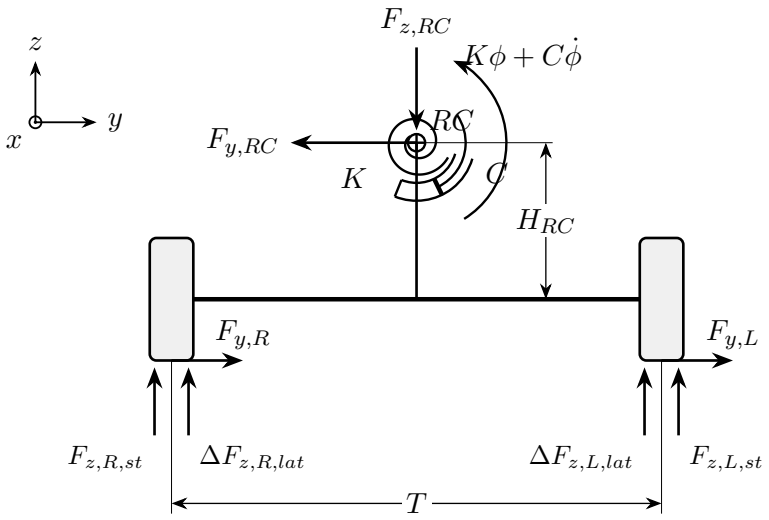


Figure 5.11: Free-body diagram of the axle. The roll moment M_r and the lateral force $F_{y,RC}$ at the roll center, the roll-center height H_{RC} and the track width T determine the lateral load transfer $\Delta F_{z,lat}$ on the two wheels.

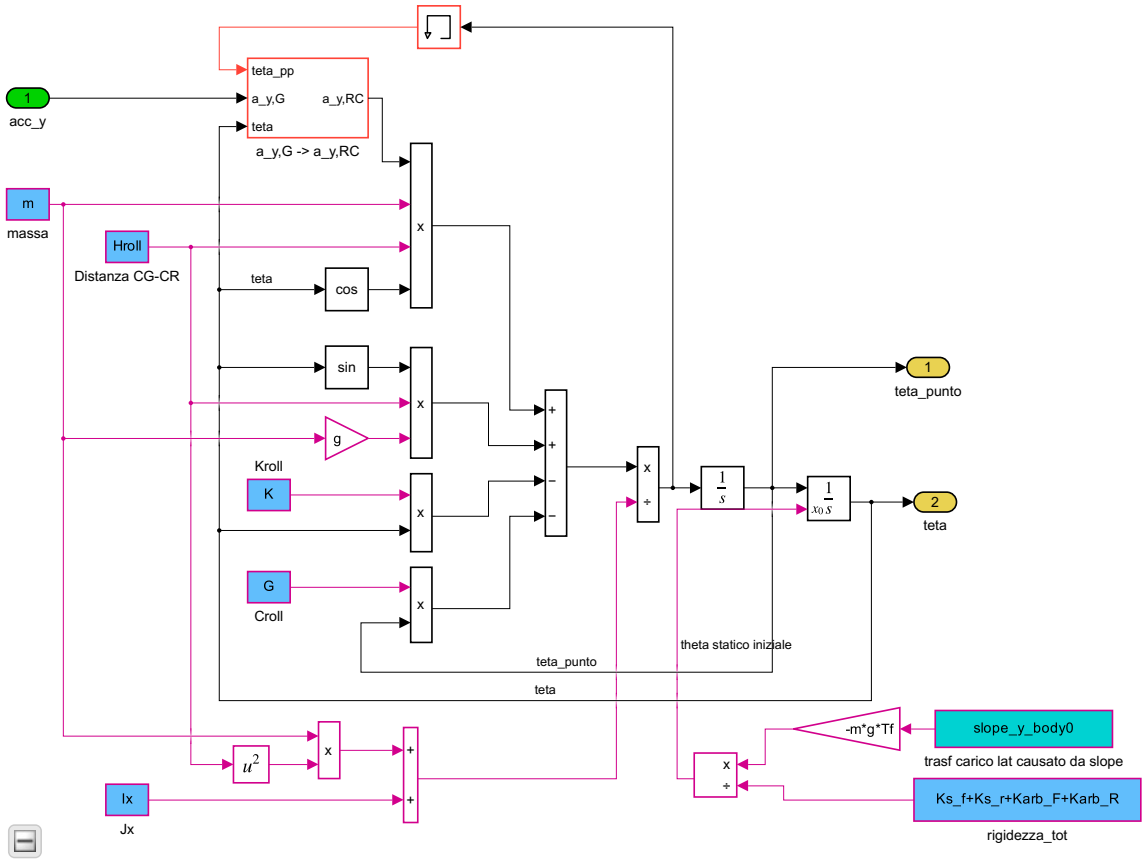


Figure 5.12: Computation of ϕ and $\dot{\phi}$ from free body diagrams. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT/VEHICLE8DOF/ModelloCassaRollio`.

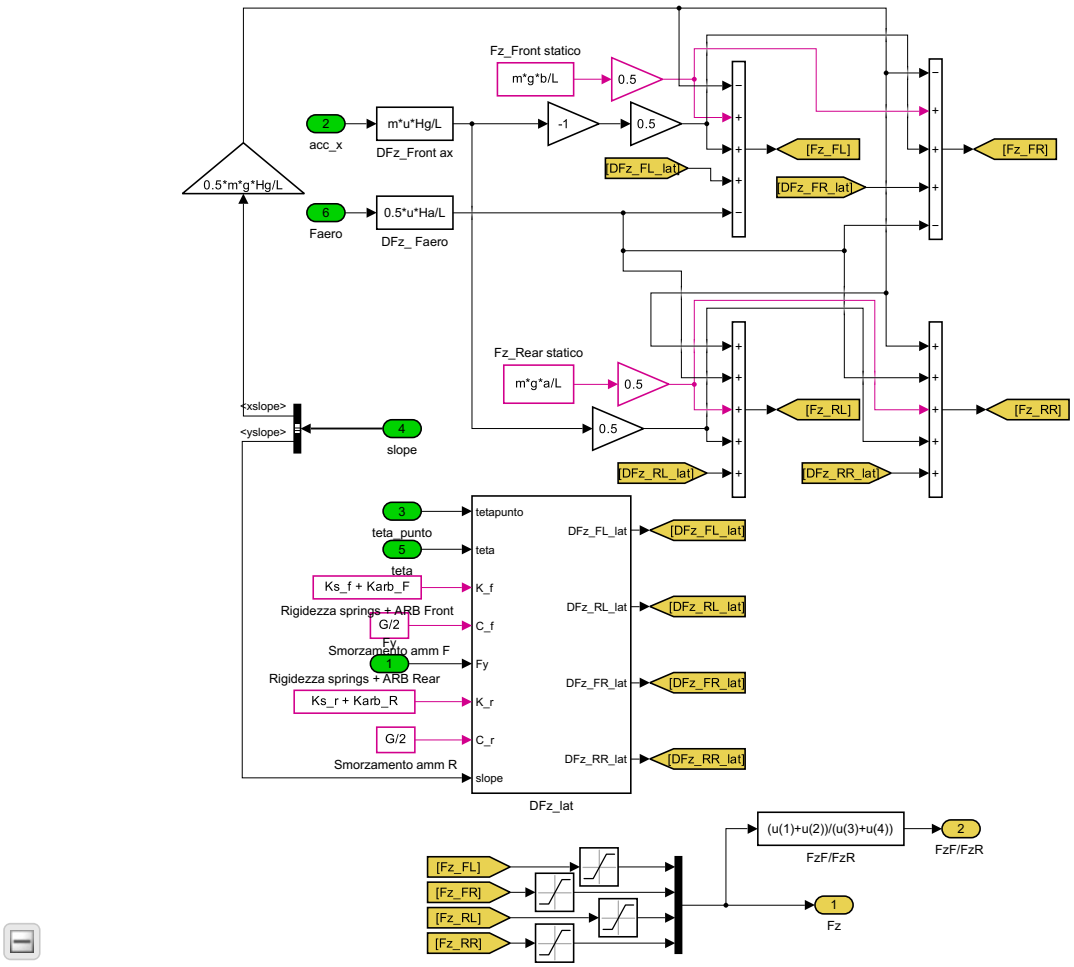


Figure 5.13: Simulink implementation of the roll dynamics and lateral load transfer: the roll state $(\varphi,\ \dot{\varphi})$ is integrated from the lateral acceleration through the suspension stiffness and damping, and redistributes the vertical loads F_z between the left and right wheels. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT/VEHICLE8DOF/CalcoloForzeNormali`.

5.7 Double Track 3DOF Model

Figure 5.14 shows the double track vehicle model in the horizontal plane, described in the body-fixed reference frame centered in the center of gravity G. Each tire exchanges with the ground a longitudinal force $F_{x,i}$ and a lateral force $F_{y,i}$, expressed in the wheel reference frame rotated by the steering angle δ_i, while the aerodynamic drag F_{aero} acts along the longitudinal direction for sake of simplicity.

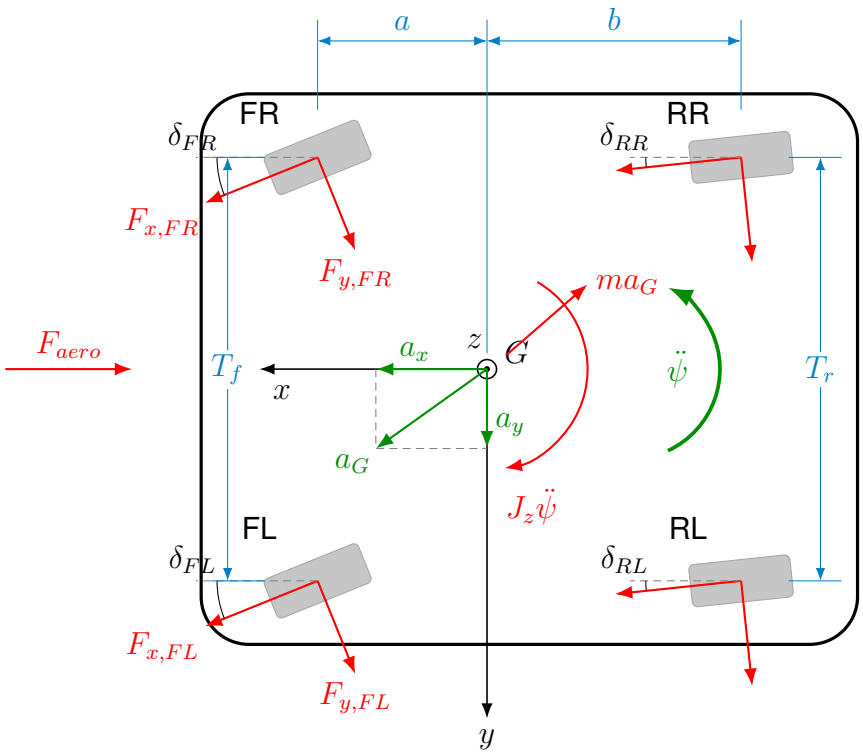


Figure 5.14: representation of double-track model

Thanks to this free body diagram is possible to write the translation equation balance

$$a_x = \frac{1}{M}\Big[(F_{x,FL}+F_{x,FR})\cos\delta_F + (F_{x,RL}+F_{x,RR})\cos\delta_R - (F_{y,FL}+F_{y,FR})\sin\delta_F + \\ - (F_{y,RL}+F_{y,RR})\sin\delta_R - \frac{1}{2}\rho\, C_x S_f u^2 \mathrm{sgn}\, u\Big] \tag{5.17}$$

$$a_y = \frac{1}{M}\Big[(F_{x,FL}+F_{x,FR})\sin\delta_F + (F_{x,RL}+F_{x,RR})\sin\delta_R + \\ + (F_{y,FL}+F_{y,FR})\cos\delta_F + (F_{y,RL}+F_{y,RR})\cos\delta_R\Big] \tag{5.18}$$

And the rotation equation balance

$$\ddot{\psi} = \frac{1}{J_z}\Big\{\frac{T_f}{2}\big[(-F_{x,FL}+F_{x,FR})\cos\delta_F + (F_{y,FL}-F_{y,FR})\sin\delta_F\big] + \\ + \frac{T_r}{2}\big[(-F_{x,RL}+F_{x,RR})\cos\delta_R + (F_{y,RL}-F_{y,RR})\sin\delta_R\big] + \\ + a\big[(F_{x,FL}+F_{x,FR})\sin\delta_F + (F_{y,FL}+F_{y,FR})\cos\delta_F\big] + \\ - b\big[(F_{x,RL}+F_{x,RR})\sin\delta_R + (F_{y,RL}+F_{y,RR})\cos\delta_R\big]\Big\} \tag{5.19}$$

Since the body frame rotates with the vehicle, it is not inertial and the acceleration of G

must account for the rotation of the unit vectors $\vec{\tau}$ and $\vec{n}$:

$$\vec{a} = a_x\vec{\tau} + a_y\vec{n} = \left(\dot{u} - \dot{\psi}\, v\right)\vec{\tau} + \left(\dot{v} + \dot{\psi}\, u\right)\vec{n} \tag{5.20}$$

so that, once a_x, a_y and $\dot{\psi}$ are known from the equations of motion, the velocity components of the center of gravity are obtained by integration:

$$u = \int \left(a_x + \dot{\psi}\, v\right) dt, \qquad v = \int \left(a_y - \dot{\psi}\, u\right) dt. \tag{5.21}$$

5.8 Terrain Grade and Bank in Vehicle Coordinates

The terrain slope is initially expressed in the global frame as s_x and s_y and is called Grade and Bank respectively. It is transformed into the vehicle body frame using the heading angle ψ:

$$s_{x,b} = s_x \cos\psi + s_y \sin\psi \tag{5.22}$$

$$s_{y,b} = -s_x \sin\psi + s_y \cos\psi \tag{5.23}$$

The grade affects traction and required propulsion force, while bank affects roll equilibrium and vertical load distribution.
As shown in Figure 5.15, the z-gradient information for each (x, y) coordinate on the map is first read from a lookup table based on the vehicle's position; it is then expressed in the vehicle's local reference frame using a rotation matrix that applies a rotation of ϕ.

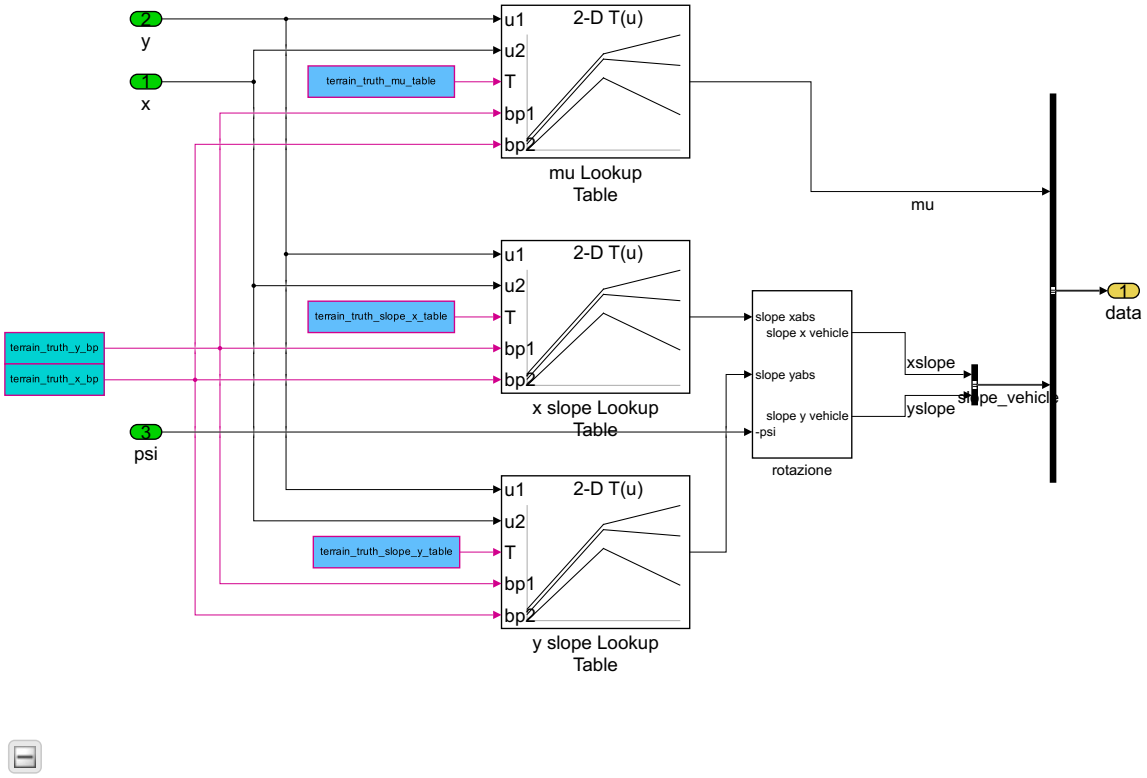


Figure 5.15: Simulink implementation of map gradient measuring. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT/get\_values/LookupTables`.

5.9 SPM Motor Model

The vehicle is propelled by four identical in-wheel electric motors, one per wheel, each connected to its wheel through a fixed reduction of ratio $10 : 1$. For sake of simplicity the transmission efficiency is considered 1. The machine is a SPM synchronous motor, a common choice for compact electric vehicles because of its high torque density and efficiency. The rated quantities of each motor are a peak power of $7.5\,\mathrm{kW}$ and a continuous power of $3.25\,\mathrm{kW}$, with a peak torque of $15\,\mathrm{Nm}$ and a nominal (continuous) torque of $7.5\,\mathrm{Nm}$; at the vehicle level the four units therefore provide a total peak power of $30\,\mathrm{kW}$ and a total peak motor torque of $60\,\mathrm{Nm}$. Through the $10 : 1$ reduction each motor turns ten times faster than its wheel and the wheel torque is ten times the motor torque (transmission losses neglected), so the peak torque available at each wheel is $150\,\mathrm{Nm}$.

5.9.1 Torque–Speed Characteristic

Like every electric machine, the motor exhibits two distinct operating regions, shown in Figure 5.16. Below the base (or corner) speed the limiting quantity is the maximum torque the machine can produce, so the torque envelope is flat: this is the constant-torque region. Above the base speed the limiting quantity becomes the maximum power, and the available torque decays hyperbolically as $T = P/\omega$: this is the constant-power (field-weakening) region. Two envelopes are defined for each in-wheel unit: the peak envelope ($T_{peak} = 15\,\mathrm{Nm}$, $P_{peak} = 7.5\,\mathrm{kW}$), which the motor can sustain only transiently, and the continuous envelope ($T_{nom} = 7.5\,\mathrm{Nm}$, $P_{nom} = 3.25\,\mathrm{kW}$), set by the thermal limit. Figure 5.17 reports the corresponding power–speed curves.

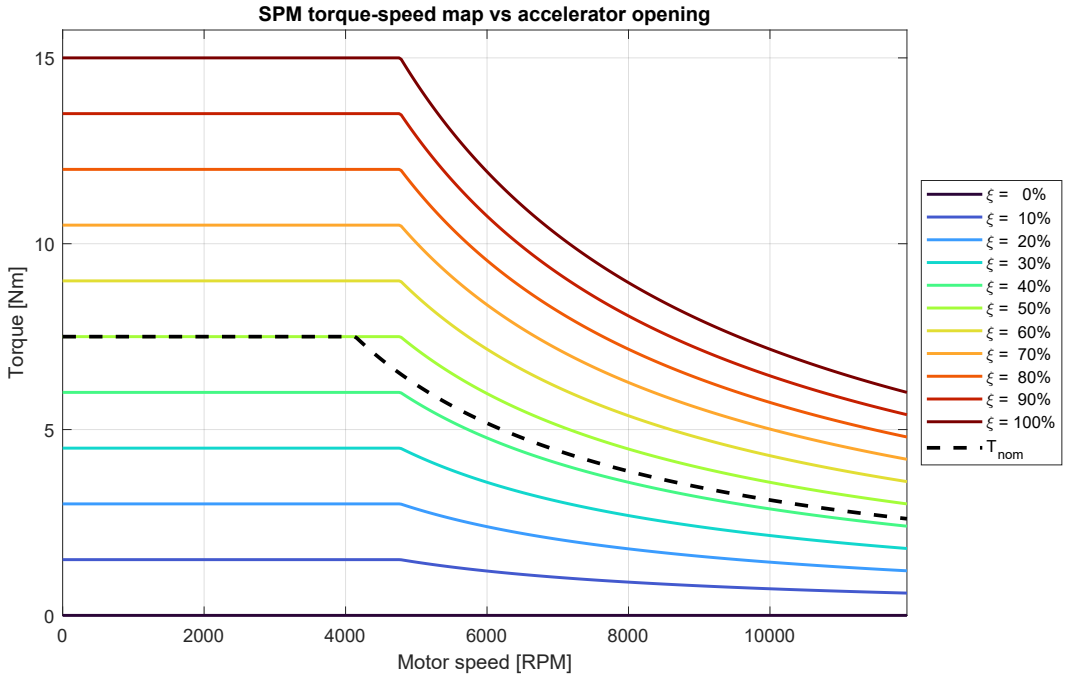


Figure 5.16: SPM motor torque–speed map of one in-wheel motor: torque delivered at the motor shaft versus motor speed, for accelerator openings from 0 to $100\,\%$. The dashed line is the continuous (thermal) torque envelope T_{nom}.

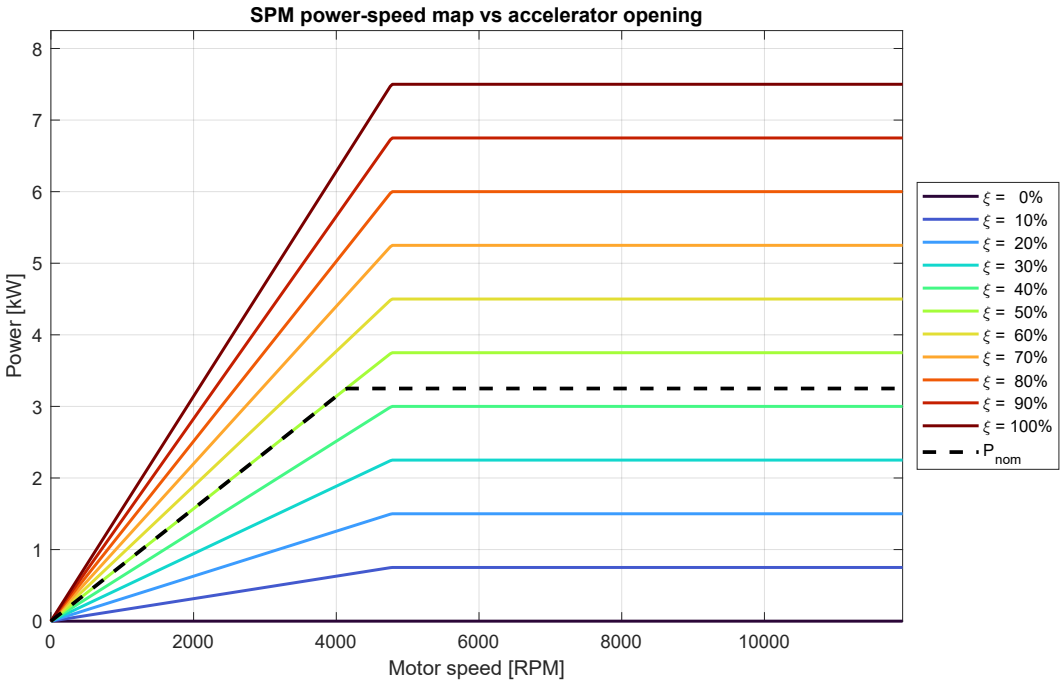


Figure 5.17: SPM motor power–speed map of one in-wheel motor: mechanical power delivered at the motor shaft versus motor speed, for accelerator openings from 0 to $100\,\%$. The dashed line is the continuous power envelope P_{nom}.

5.9.2 Slip-Based Torque Cut-Off

The engine block does not feed the accelerator command directly to the motor map: the driver request is scaled by a multiplicative factor produced by a PI regulator acting on the wheel slip error $e = \sigma^{\star} - \sigma$, where $\sigma\star = 0.10$ is the target slip and σ is the measured longitudinal slip (Fig. 5.18). With gains $K_p = 4$ and $K_i = 15$, the PI output is clamped to the interval $[0, 1]$ and its integrator is initialized at unity, so at start-up the loop is transparent and the full pedal command reaches the motor. While $\sigma < \sigma^{\star}$ the error is positive, the output saturates at one and the anti-windup clamping prevents the integrator from accumulating; as soon as σ exceeds the threshold the error becomes negative, the factor drops below unity and the effective throttle, hence the requested motor torque, is reduced until the slip returns to the admissible range. The scaled command drives the second input of the motor map, whose first input is the motor speed reconstructed from the wheel angular velocity through the gain $\mathrm{gear\_ratio} \cdot 60/(2\pi)$; the map output is finally multiplied by the gear ratio to yield the wheel torque C_m.

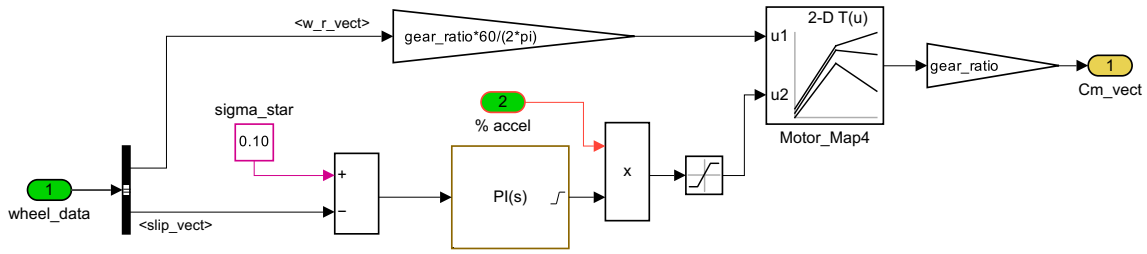
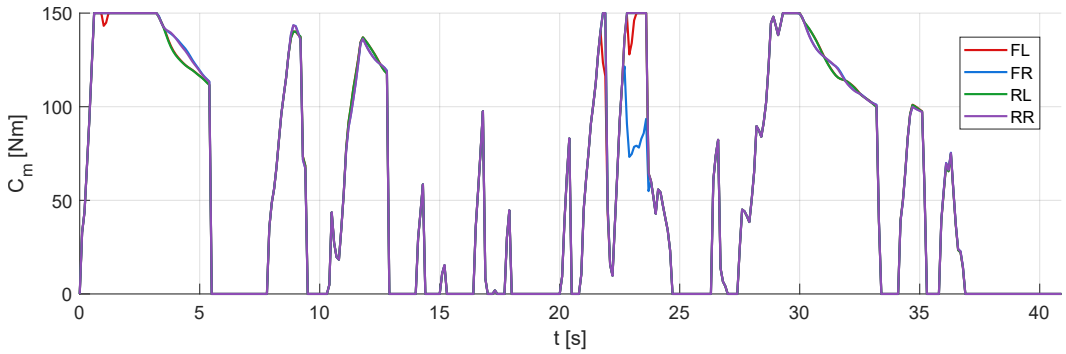
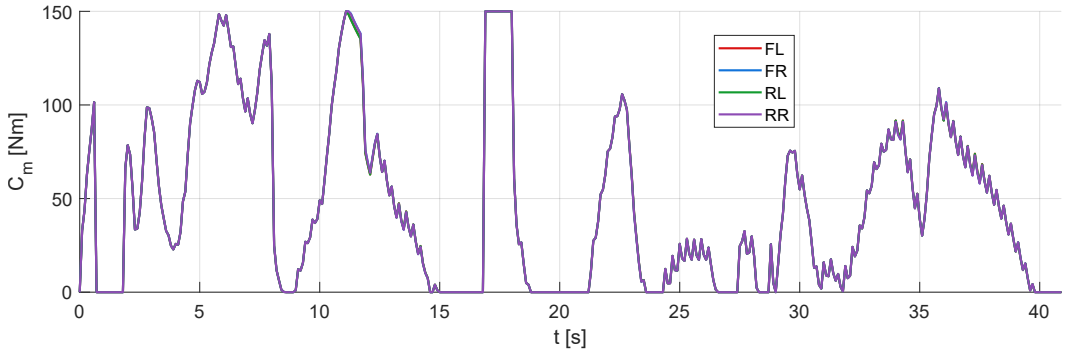


Figure 5.18: Simulink implementation del blocco motore e del cutoff mediante un controllore PI. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT/ENGINE`.

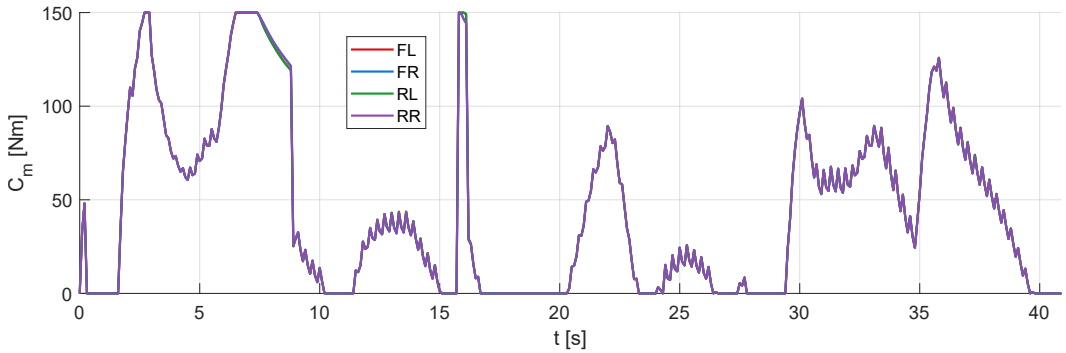
The plots of the driving torques for each wheel are shown in Figure 5.19. Comparing these with Figure 6.19 and Figure 5.30, it can be seen that while the throttle is at 100% opening, the torque drops into the flux-weakening region because the angular velocity of the wheels is increasing. Furthermore, the cut-off action on vehicle `ID1` is visible at certain moments.



(a) Driving torque — Vehicle 1



(b) Driving torque — Vehicle 2



(c) Driving torque — Vehicle 3

Figure 5.19: Driving torque at each wheel for vehicles 1, 2 and 3.

5.10 Wheel Rotational Dynamics

Figure 5.20 shows the free-body diagram of a single wheel and the adopted torque sign convention. The wheel is subjected to the driving torque T_m, the braking torque T_b, assumed positive but acting in the opposite direction with respect to the driving torque, the longitudinal tire force F_x, the vertical load F_z, and the rolling-resistance effect associated with the longitudinal offset Δx of the normal reaction. For each wheel, the simplified rotational balance implemented in the model is written as

$$J_w \dot{\omega} = T_m - T_b - F_x R_0, \tag{5.24}$$

where J_w is the wheel rotational inertia, ω is the wheel angular velocity and R_0 is the effective rolling radius. The equation is integrated to update the wheel speed ω, which is then used to compute the longitudinal slip ratio.

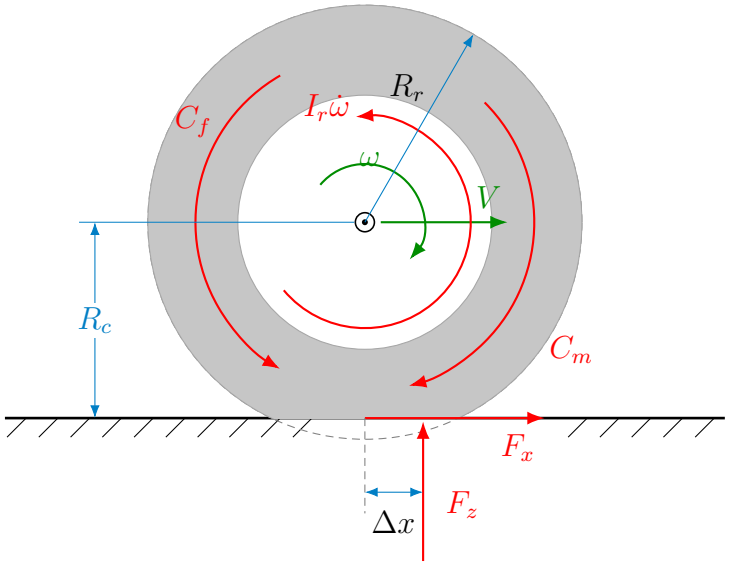


Figure 5.20: free body diagram of the single wheel

All of this is represented schematically in Simulink using the block in Figure 5.21

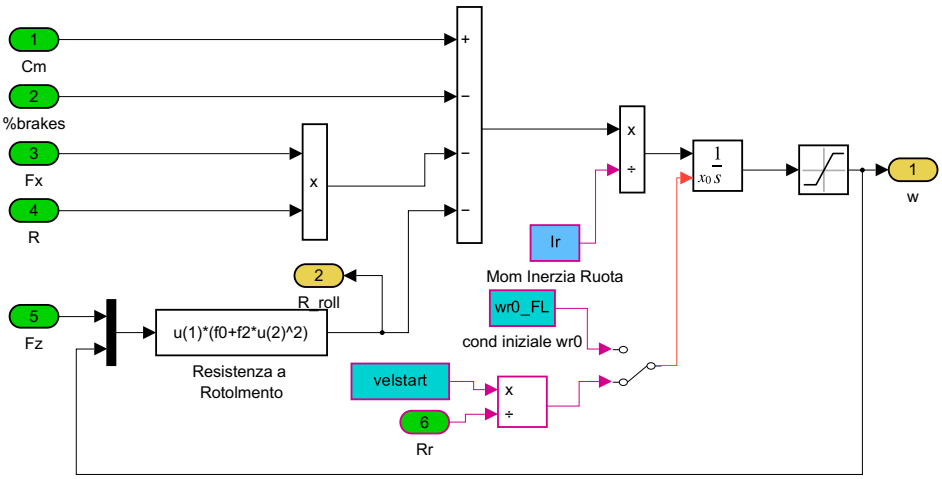


Figure 5.21: Simulink implementation of the wheel free body diagram. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT/VEHICLE8DOF/DinamicaRotazionaleRuote(EBD\&ABS)/DinamicaRuotaFL`; an identical block is also present for the FR, RL and RR wheels.

Sensors regulating braking — specifically EBD and ABS, discussed in detail in Section 5.10.1 and 5.10.2 — are connected to the wheels; the effects of these systems are shown in Section 5.12.

5.10.1 EBD

The effect of the EBD block is fairly intuitive: the braking effort commanded to each wheel is distributed in proportion to the vertical load F_z available at that wheel. As a result, both under straight-line braking and while cornering, a larger share of brake pressure is sent to the wheels that can actually sustain more braking force.

This follows directly from the fact that the longitudinal force F_x transmitted to the ground, and hence the deceleration it produces, depends on F_z: the higher the vertical load, the higher the peak F_x the tire can generate. This dependency is clearly visible in the F_x-σ curves obtained from Pacejka's Magic Formula, shown qualitatively for two values of F_z in Figure 5.22.

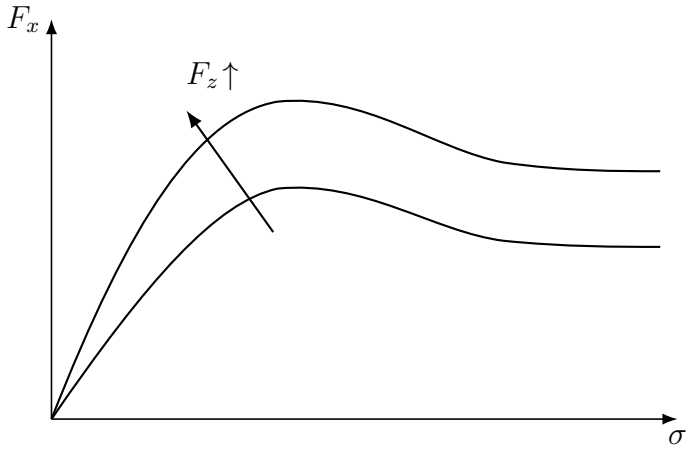


Figure 5.22: Qualitative trend of F_x as a function of the slip σ, for different values of F_z.

To formalize this idea, for each wheel $i \in \{FL, FR, RL, RR\}$ the maximum friction force it can locally sustain is estimated as

$$g_i = \mu_i F_{z,i},$$

where μ_i and $F_{z,i}$ are respectively the estimated friction coefficient and the vertical load at wheel i. Summing this quantity over the four wheels gives the total braking capacity currently available to the vehicle,

$$G = \sum_i g_i = \sum_i \mu_i F_{z,i};$$

a saturation block is placed on G to prevent division by zero whenever all four wheels are close to zero load, for instance during a wheel lift-off event. The share of the overall braking capacity attributable to wheel i is then simply

$$\xi_i = \frac{g_i}{G} = \frac{\mu_i F_{z,i}}{\sum_i \mu_i F_{z,i}}, \qquad \sum_i \xi_i = 1,$$

and it is precisely this weight that is used to split the driver's overall braking demand among the four wheels: the brake torque coefficient assigned to wheel i is

$$C_{f,i} = 1.5\,\xi_i\,(\%\text{brake}).$$

The logic of the subsystem, shown in Figure 5.23, is therefore to redistribute the total braking demand $\%$brake in proportion to the friction force each wheel can actually sustain, so that more loaded or higher-grip wheels receive a correspondingly larger share of the braking torque.

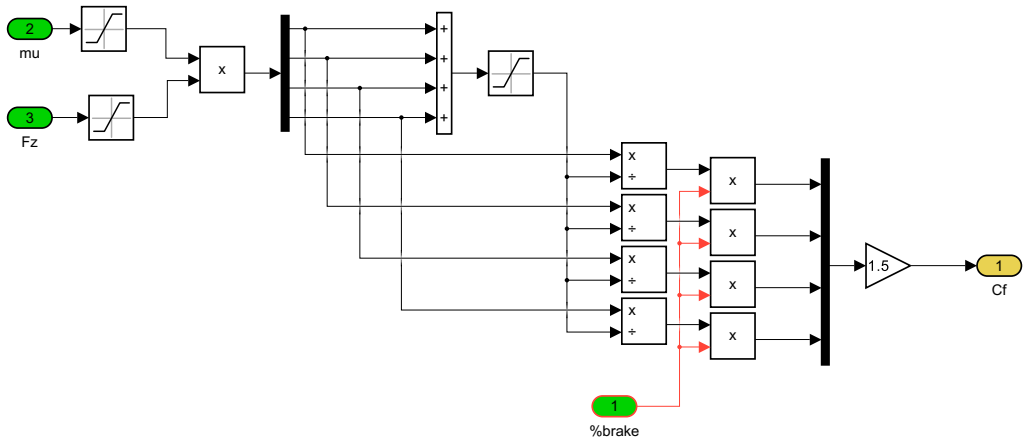


Figure 5.23: Simulink implementation of the EBD. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT/VEHICLE8DOF/DinamicaRotazionaleRuote(EBD\&ABS)/DinamicaRuotaFL/EBD`; an identical block is also present for FR, RL and RR.

The constant gain of 1.5, with units of $\mathrm{Nm}/\%$brake, is what converts the scaled percentage braking demand into the actual brake torque C_f. Its value follows directly from the input and output ranges chosen for the model: the extreme, full-braking value of the accelerator/brake signal was set to -500, while the maximum admissible brake torque per wheel was set to $750\ \mathrm{Nm}$, five times the maximum drive torque per wheel. The ratio between these two extreme values, $750/500$, is exactly 1.5, which is the gain applied at the output of the subsystem.

5.10.2 ABS

The braking logic adopted here is a simplified surrogate for an anti-lock braking system rather than a faithful reproduction of one. It keys the three operating modes (apply, hold, release) directly to the wheel slip, switching to release once the slip drops below σ_{rel} and holding the previous demand throughout the intermediate band down to σ_{hold}. This

formulation is convenient in simulation, where the slip is readily available from the vehicle model, but it departs from how a physical controller operates.

A production ABS does not act on real slip at all, since slip presupposes knowledge of the absolute vehicle speed, which cannot be measured directly on a braking vehicle and would itself have to be estimated. Instead it monitors the angular acceleration of each wheel, obtained by differentiating the wheel speed signal. A release is triggered when the wheel decelerates too sharply, typically beyond about $-100\,\mathrm{rad/s^2}$, signaling incipient lock; the brake pressure is then held and re-applied once the wheel spins back up past a positive acceleration threshold. The decision variable is therefore the angular acceleration of the wheel rather than the slip, which makes the scheme independent of any vehicle-speed estimate.

The two formulations also differ in their temporal behavior. A real ABS operates as a limit cycle, modulating each wheel at roughly 8 to $15\,\mathrm{Hz}$ with markedly asymmetric phase durations: the release phase is very short, on the order of 20 to $50\,\mathrm{ms}$, after which the controller returns to apply and attempts to build brake pressure again. The logic used here reacts at every sample but imposes no such cycle. If the slip settles inside the hold band it remains there indefinitely, whereas a physical controller would leave that state and re-enter the apply phase to keep probing for additional braking effort.

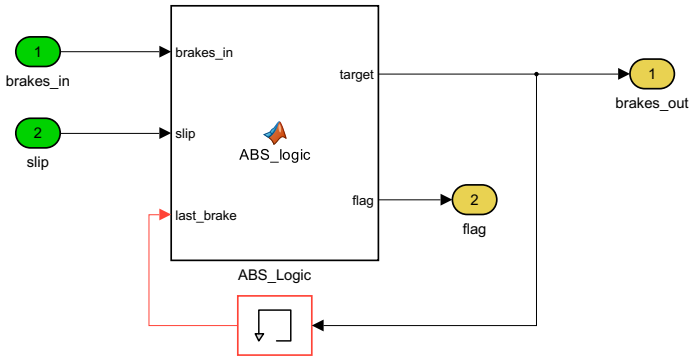


Figure 5.24: Simulink implementation of the ABS. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT/VEHICLE8DOF/DinamicaRotazionaleRuote(EBD\&ABS)/DinamicaRuotaFL/ABS`; an identical block is also present for FR, RL and RR.

The pseudocode for the `ABS_logic` function is as follows

```
function [target, flag] = fcn(brakes_in, slip, last_brake)
for i = 1 : num_wheels do
if slip[i] <= SLIP_RELEASE then  // SLIP_RELEASE = -0.20
target[i] <- 0 ;  flag[i] <- 2
```

```
        else if slip[i] <= SLIP_HOLD then  // SLIP_HOLD = -0.12
        target[i] <- last_brake[i] ;  flag[i] <- 1
        else
        target[i] <- brakes_in[i] ;  flag[i] <- 0
        end
        end
```

During the mission, the ABS did not engage for vehicles 1 and 3, whereas it activated for vehicle 2, as shown in Figure 5.25.

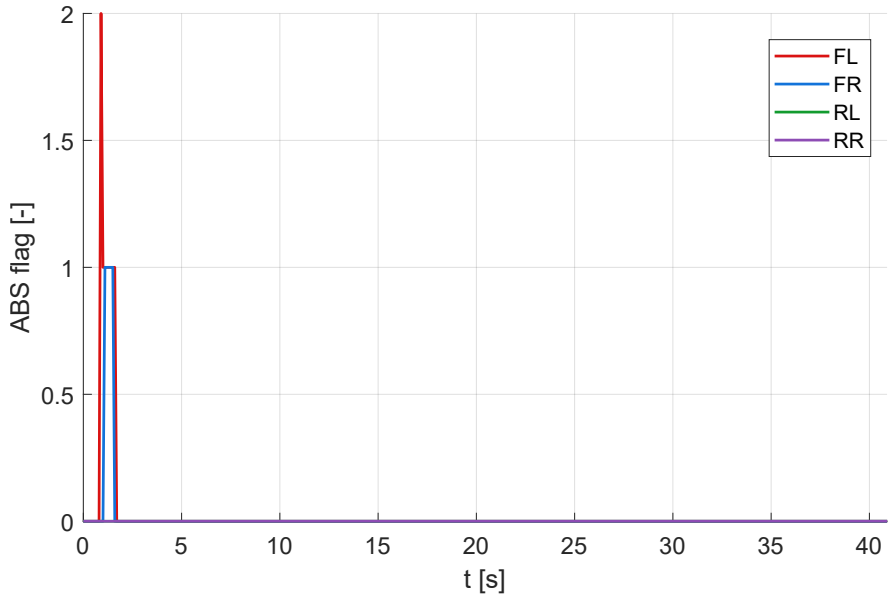


Figure 5.25: ABS flag report for each wheel of Vehicle 2: flag 0 = off, flag 1 = hold, flag 2 = release.

In a real mission on deformable terrain, the ABS thresholds would need to be recalibrated so that the system activates at higher slip values. Under these conditions, the longitudinal slip is also expected to oscillate within a wider range, since the tire–terrain interaction is less regular and the available friction varies more significantly than on rigid terrain.

5.11 4WS

Given the type of terrain, which will be very difficult to traverse in a real-world environment, it is advisable to equip the vehicles with a 4WS system. Each agent does not steer the front axle alone: the steering command produced by the path tracker is distributed to both axles, so that the rear wheels also contribute to following the planned path. The two steer angles are linked to a common steering input δ by two transmission ratios,

$$\delta_1 = K_1' \, \delta, \qquad \delta_2 = K_2' \, \delta, \tag{5.25}$$

where δ_1 and δ_2 are the front and rear steer angles, $l = a + b$ is the wheelbase, and a, b are the distances of the center of gravity from the front and rear axles. The front ratio is always positive, $K_1' > 0$, so the front wheels reproduce the steering input; the sign of the rear ratio K_2' instead selects the steering phase. A negative ratio, $K_2' < 0$, steers the rear wheels opposite to the front (out-of-phase), while a positive ratio, $K_2' > 0$, steers them in the same direction (in-phase). This distribution is realized in the block of Figure 5.26.

The role of the two ratios is read directly from the steady-state response of the single-track model. The yaw-rate and curvature gains depend on the difference $K_1' - K_2'$,

$$\frac{r}{\delta} = \frac{V}{l}\,\frac{K_1' - K_2'}{1 + KV^2}, \tag{5.26}$$

$$\frac{1}{R\,\delta} = \frac{1}{l}\,\frac{K_1' - K_2'}{1 + KV^2}, \tag{5.27}$$

where V is the forward speed, r the yaw rate, R the turn radius and K the stability (understeer) factor; setting $K_1' = 1$ and $K_2' = 0$ recovers the classical front-steering bicycle model. Out-of-phase steering makes $K_1' - K_2'$ larger and therefore raises the curvature, lateral-acceleration and yaw-rate gains: the vehicle turns on a tighter radius and reacts more promptly to the input. This is the behavior sought at low speed and in tight maneuvers, where a small turning radius and high agility matter most.

The sideslip-angle gain keeps the explicit dependence on both ratios,

$$\frac{\beta}{\delta} = \frac{b}{l}\left[K_1' + K_2'\,\frac{a}{b} - \frac{mV^2}{l}\left(\frac{aK_1'}{bC_2} - \frac{K_2'}{C_1}\right)\right]\frac{1}{1 + KV^2}, \tag{5.28}$$

with m the vehicle mass and C_1, C_2 the cornering stiffnesses of the front and rear axles. In a front-only vehicle the sideslip angle β grows increasingly negative with speed, so the rear of the body swings towards the outside of the corner. In-phase rear steering introduces a positive contribution that opposes this term and drives β towards zero. At high speed this becomes the dominant requirement: keeping the body aligned with its velocity vector suppresses the rear-end slip and damps the yaw response, so that a lane change is executed in a stable and predictable way rather than as a rotation about the center of gravity.

The two regimes are complementary, and the rear ratio is scheduled with speed to exploit both, moving from a negative low-speed value to a positive high-speed value through a logistic (sigmoid) blend,

$$K_2'(V) = K_{2,\mathrm{lo}}' + \left(K_{2,\mathrm{hi}}' - K_{2,\mathrm{lo}}'\right)\sigma(V), \qquad \sigma(V) = \frac{1}{1 + e^{-(V - V_c)/\Delta V}}, \tag{5.29}$$

with $K_{2,\mathrm{lo}}' = -0.4$, $K_{2,\mathrm{hi}}' = 0.3$, crossover speed $V_c = 10\ \mathrm{m/s}$ and transition width $\Delta V = 2.22\ \mathrm{m/s}$. The sigmoid provides a bounded, monotonic and continuously differentiable

reversal of phase across V_c, so that $K_2' \to -0.4$ at low speed (out-of-phase, agility) and $K_2' \to 0.3$ at high speed (in-phase, stability), with no abrupt change in the rear steer angle. The front ratio is reduced only mildly with speed,

$$K_1'(V) = 1 - 0.3 \, \min\left(\frac{V}{V_{\text{sat}}}, \, 1\right), \qquad V_{\text{sat}} = 28 \text{ m/s}, \tag{5.30}$$

decreasing from 1 at standstill to 0.7 above V_{sat}, which partially compensates the growth of the yaw-rate gain with speed in (5.26).

The constants in (5.29) and (5.30) were chosen empirically, so that the resulting steer angles stay within the physical range of the steering actuators over the whole operating speed range. A closed-form tuning, for instance from the constant-yaw-rate-gain criterion, would require the cornering stiffnesses C_1, C_2 of the tires on the actual off-road terrain: reliable Pacejka coefficients for off-road tires are not available and cannot be obtained from tire suppliers, so the analytical route is not viable and the parameters are fixed by simulation instead.

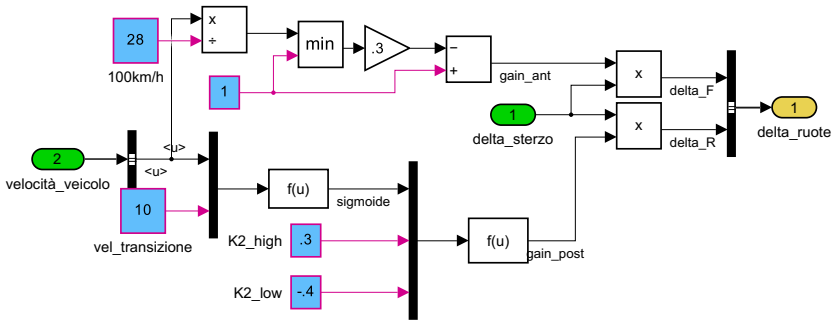
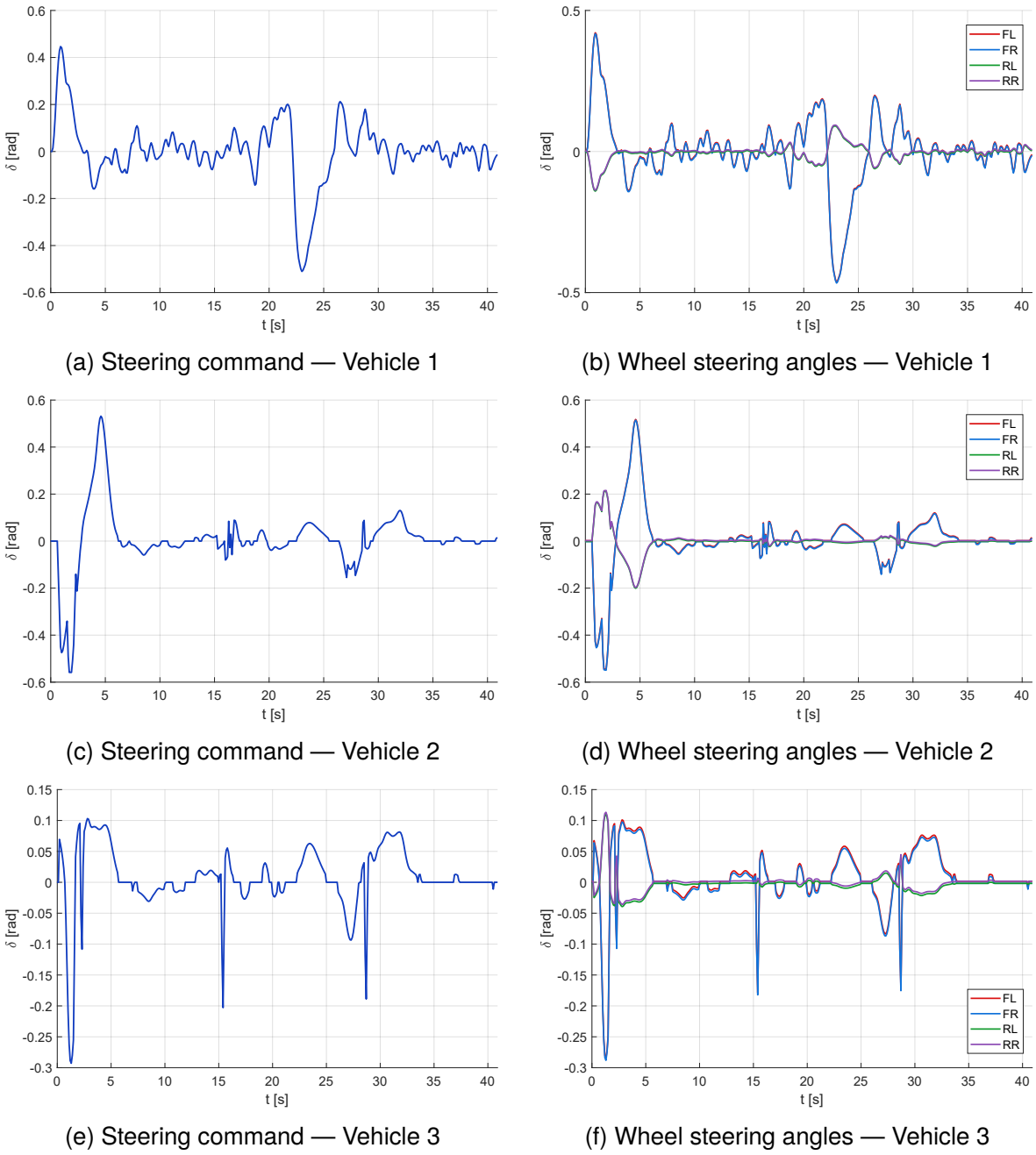


Figure 5.26: Simulink implementation of the 4 Wheel Steering system. Located at `swarm_phase_5/(Agent)/(vehicle)/PLANT/VEHICLE8DOF/4WS`.

The plots of the wheel steering delta are shown below in Figure 5.27, to be compared with Figure 6.14.



(a) Steering command — Vehicle 1

(b) Wheel steering angles — Vehicle 1

(c) Steering command — Vehicle 2

(d) Wheel steering angles — Vehicle 2

(e) Steering command — Vehicle 3

(f) Wheel steering angles — Vehicle 3

Figure 5.27: Steering-wheel command and wheel steering angles for each wheel of vehicles 1, 2 and 3.

5.12 Vehicle Dynamics Results

The mission is now examined from the vehicle-dynamics side.

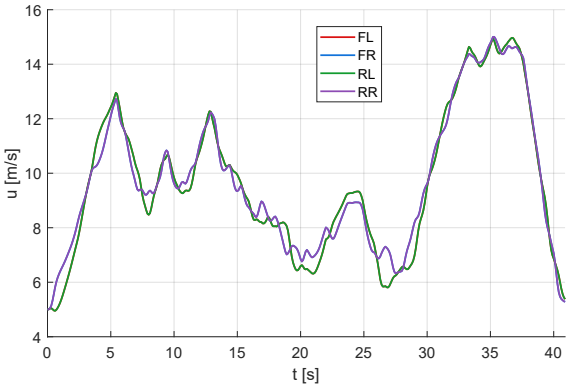
The plots are mutually consistent: an increase in ω is matched by an increase in u_{cr} as well. The time variation of F_z, together with the fact that it differs across all four wheels, confirms the effectiveness of the 8-DOF model.

The slip values always remain between -0.2 and 0.2, thanks to the cutoff of Section 5.9.2: regarding traction ($\sigma > 0$), for Vehicle 1 the slip rises between 20 and $25\,\mathrm{s}$, and

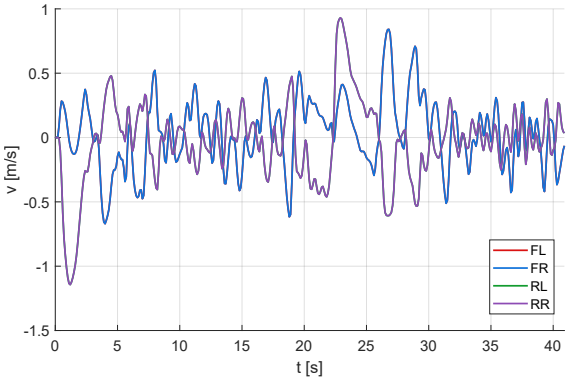
the cutoff can be seen acting on the same vehicle at the same instants, reducing C_m in the plot of Figure 5.19.

At $t = 15\,\mathrm{s}$ the effect of the obstacle-brake block is visible, which acts on Vehicles 2 and 3: its effect can be seen in Figure 6.9. Indeed, at that instant ($t = 15\,\mathrm{s}$, as established in Section 7.1), for Vehicles 2 and 3 one can observe a jump in F_x, a load transfer in F_z, and an increase in the absolute value of the slip σ.

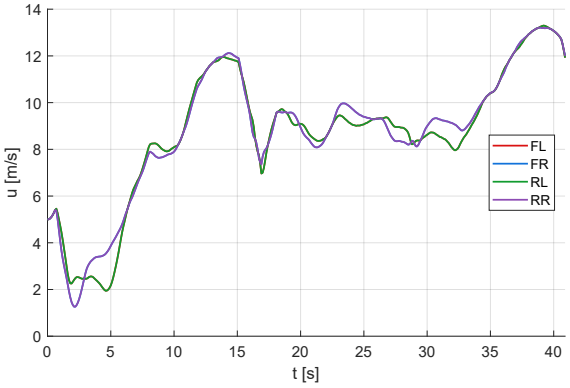
As for braking, on the other hand, Figure 5.25 shows the effect of the ABS on Vehicle 2: in the same instants in which the plot of Figure 5.25 is non-zero, the slip is seen to reach a value of about -0.2 without ever exceeding it. For Vehicle 3, instead, the slip varies within a range that is entirely acceptable for this type of simulation.



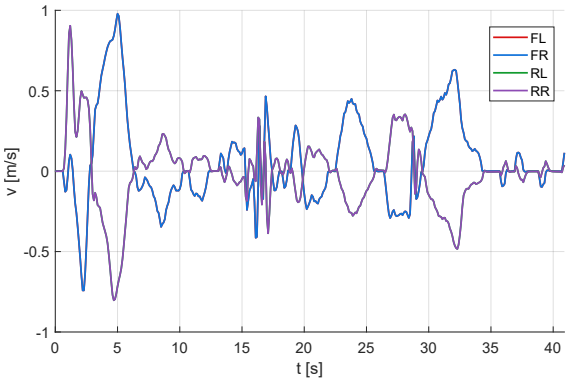
(a) Longitudinal wheel-center velocity u_{cr} ID1



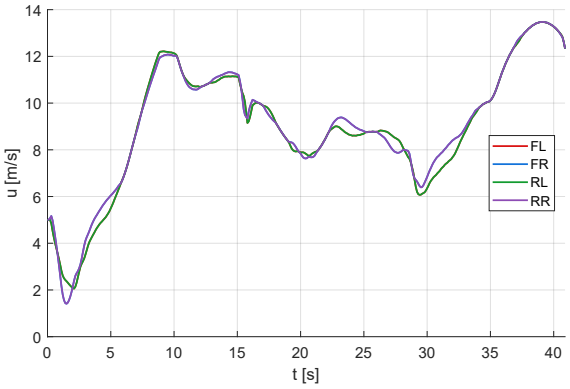
(b) Lateral wheel-center velocity v_{cr} ID1



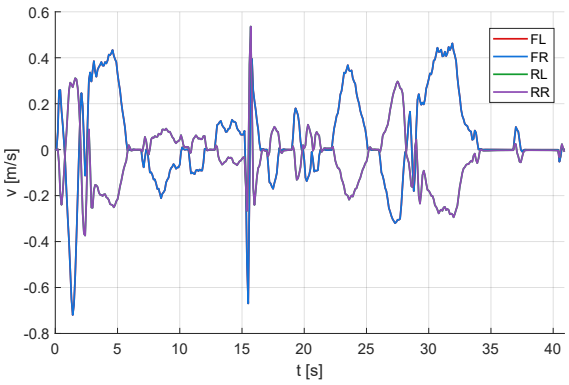
(c) Longitudinal wheel-center velocity u_{cr} ID2



(d) Lateral wheel-center velocity v_{cr} ID2

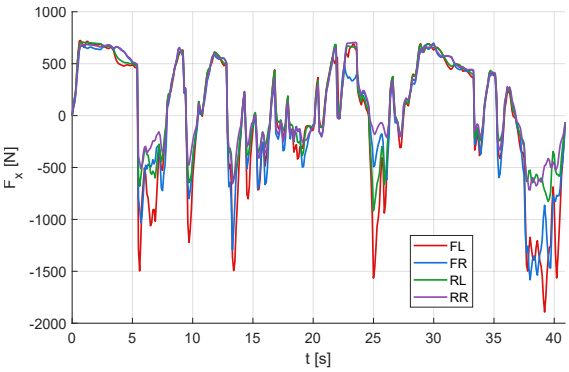


(e) Longitudinal wheel-center velocity u_{cr} ID3

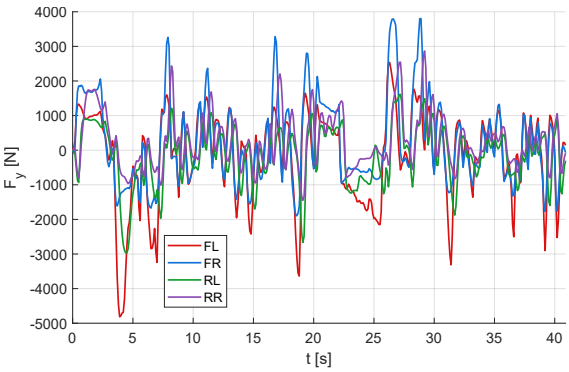


(f) Lateral wheel-center velocity v_{cr} ID3

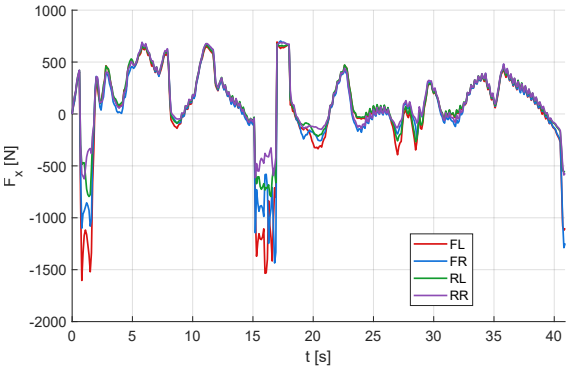
Figure 5.28: Longitudinal and lateral wheel-center velocity components for vehicles 1, 2 and 3.



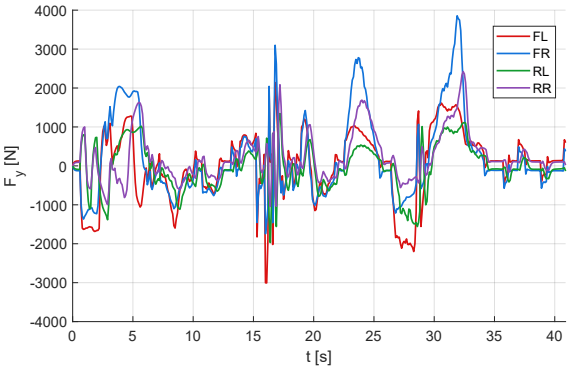
(a) Longitudinal tire force F_x — ID1



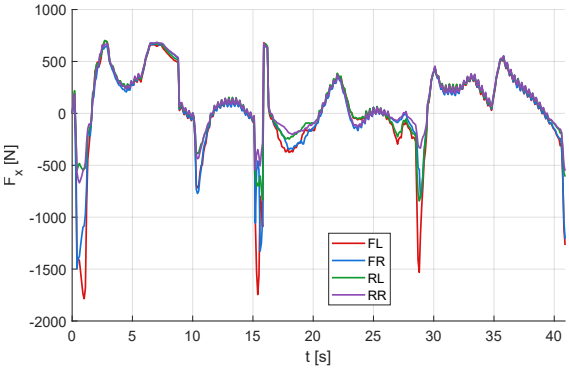
(b) Lateral tire force F_y — ID1



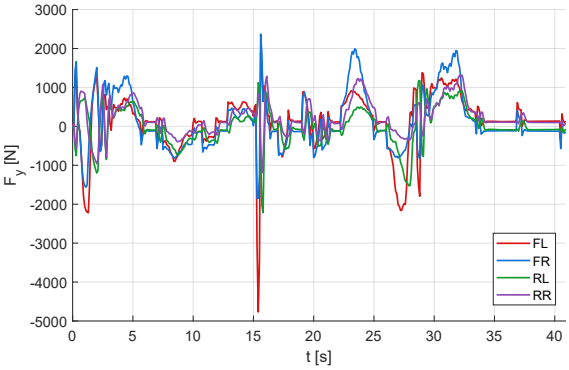
(c) Longitudinal tire force F_x — ID2



(d) Lateral tire force F_y — ID2

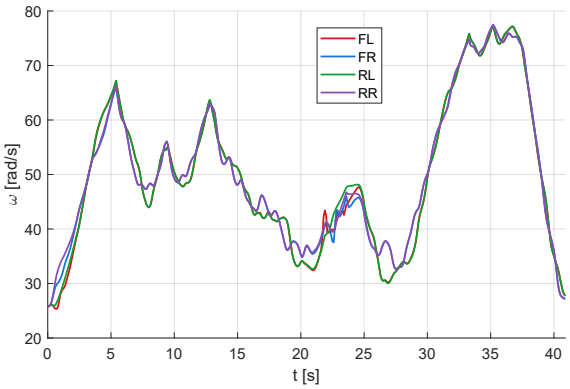


(e) Longitudinal tire force F_x — ID3

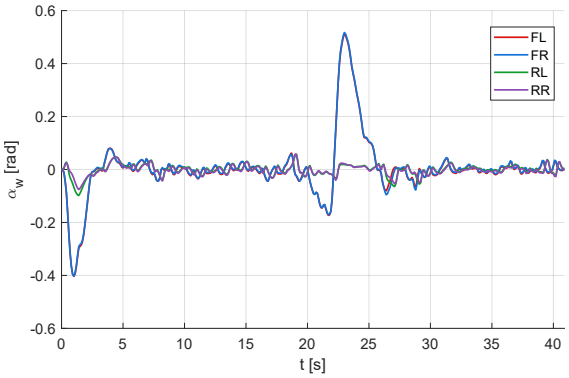


(f) Lateral tire force F_y — ID3

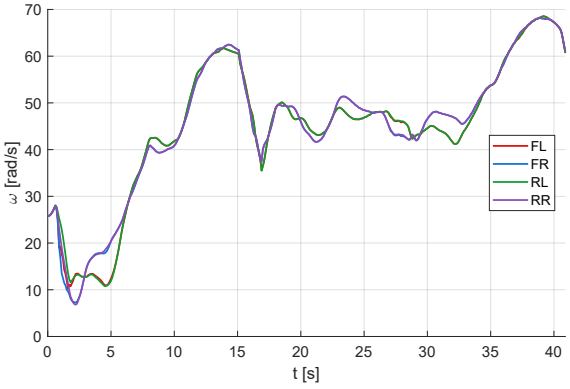
Figure 5.29: Longitudinal and lateral tire forces for vehicles 1, 2 and 3.



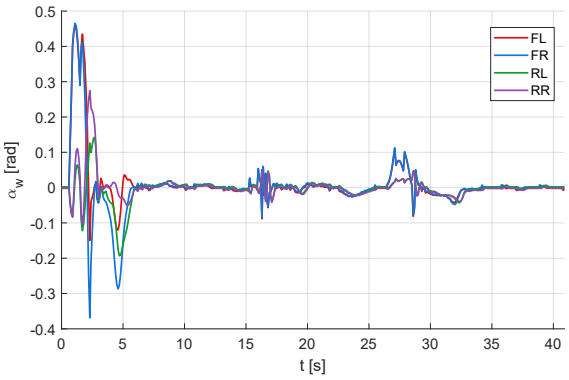
(a) Wheel angular velocities ω — ID1



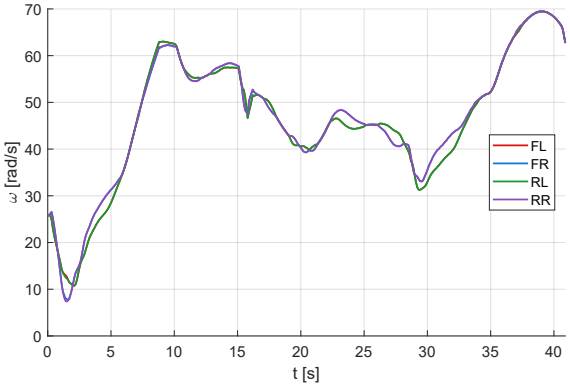
(b) Wheel slip angles α — ID1



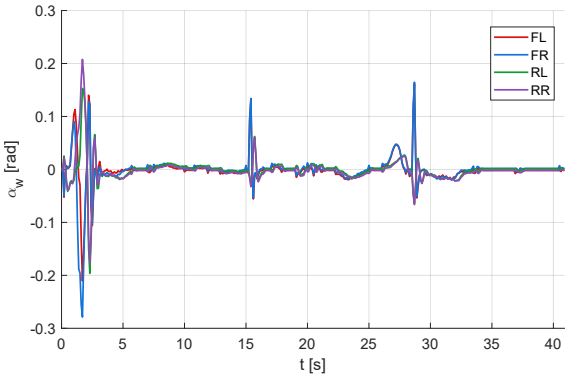
(c) Wheel angular velocities ω — ID2



(d) Wheel slip angles α — ID2

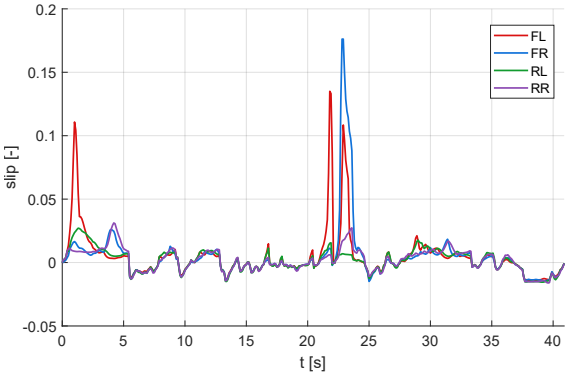


(e) Wheel angular velocities ω — ID3

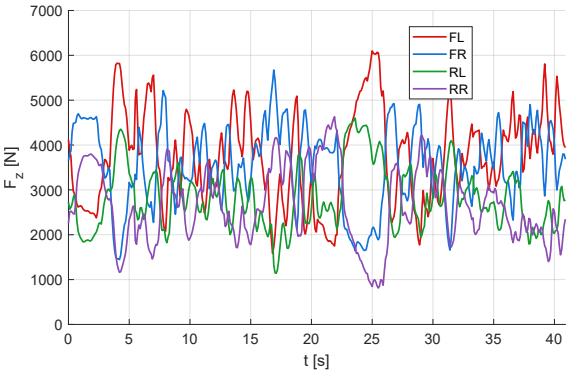


(f) Wheel slip angles α — ID3

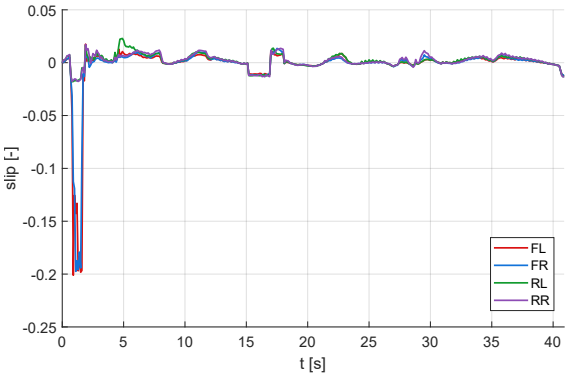
Figure 5.30: Wheel angular velocities and wheel slip angles for vehicles 1, 2 and 3.



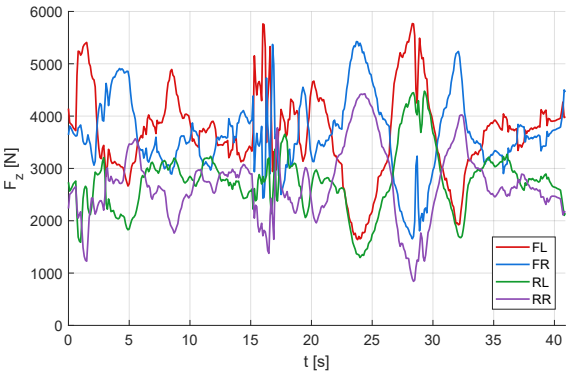
(a) Longitudinal slip ratio — Vehicle 1



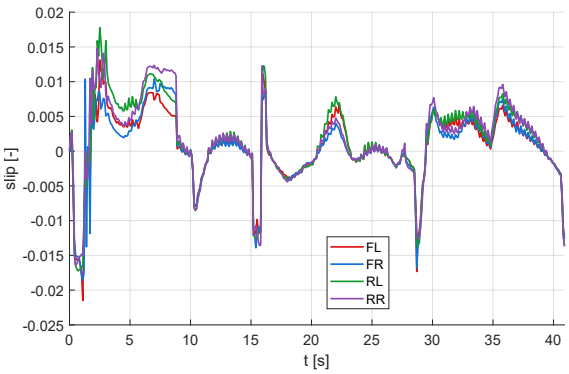
(b) Tire vertical force F_z — Vehicle 1



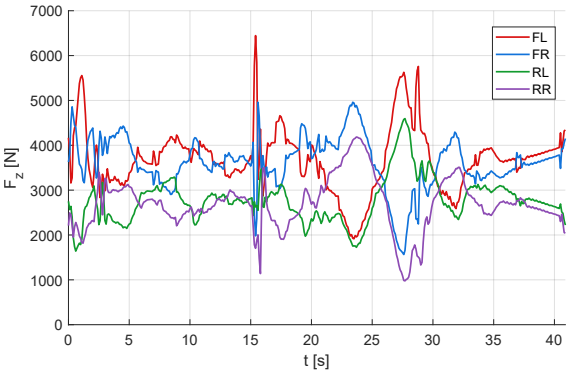
(c) Longitudinal slip ratio — Vehicle 2



(d) Tire vertical force F_z — Vehicle 2



(e) Longitudinal slip ratio — Vehicle 3



(f) Tire vertical force F_z — Vehicle 3

Figure 5.31: Wheel longitudinal slip ratios and tire vertical forces F_z for vehicles 1, 2 and 3.

Chapter 6

Path Planning, Swarm Control and Speed Governance

6.1 Overview

This chapter describes the control systems that enable the UGV swarm to navigate the map autonomously. The navigation architecture combines dynamic leader election, global path planning, formation generation, path tracking, and speed reference management. The results and plots here also refer to the simulation shown in 7.2. With reference to Fig 3.2, the following sections will analyze the operation of the Planner and the controllers within the Vehicle subsystem.

6.2 Dynamic Leader Election and Followers Ranking

The leader role is not assigned statically: at every supervisory step a ranking mechanism, executed identically on every agent, elects as leader the vehicle currently best placed to reach the goal. The elected leader computes a global path toward the goal, while the followers receive target positions defined relative to the leader and generate their own local paths toward these targets. At every supervisory tick each agent executes, in order, two decision blocks — the ranking described in this section and the path planner of the following sections. They share an identical input/output contract: they read the `shared_bus` and write to the agent's own bus. This slow task carries every decision that does not need the fast integration rate, keeping the high-frequency loop light.

The philosophy is simple: the agent that is best placed to reach the goal should lead. Leadership is therefore not hard-wired but earned each tick from a performance score that blends two normalized terms with fixed weights,

$$s_k = W_{dist}\,\frac{d_k}{\max_j d_j} + W_{eta}\,\widetilde{\mathrm{ETA}}_k, \qquad W_{dist} = 0.70, \quad W_{eta} = 0.30, \tag{6.1}$$

where d_k is the distance of agent k from the goal and the second term is the radial ETA — the time-to-goal $d_k / v_{rad,k}$ obtained by projecting the agent's velocity onto the line of sight to the target. The ETA is considered reliable only if the closing speed exceeds `V_RAD_EPS` $= 0.2\,\mathrm{m/s}$; it is normalized by the largest valid ETA in the fleet and saturated at `ETA_BAD_NORM` $= 2$, so that a stalled, lateral-moving or receding vehicle receives a moderate — not infinite — penalty and the ETA term can never overwhelm the $70/30$ weighting. A vehicle driving straight at a nearby goal scores low (good); sorting the scores in ascending order yields the ranking, with `rank = 1` denoting the leader. To suppress frequent role chatter, a hysteresis band is applied: a ranking that differs from the previous one is committed only if the gap between the two best scores exceeds a fraction `HYST_FRAC` $= 0.20$ of the runner-up score; otherwise the previous ranking is held (`persistent` state).

```
function ranking = classifica(bus, goal)
// --- 1. distance and radial ETA of every agent
for k = 1 : num_vehicles do
d[k]  <- || goal - pos[k] ||
u_los <- (goal - pos[k]) / d[k]    // unit line-of-sight
v_rad <- dot( vel[k], u_los )    // closing speed
if v_rad > V_RAD_EPS then     // V_RAD_EPS = 0.2 m/s
eta[k] <- d[k] / v_rad ;  valid[k] <- true   // radial ETA
else
valid[k] <- false     // stalled, lateral or receding
end
end

// --- 2. normalized weighted score (low = better)
for k = 1 : num_vehicles do
d_norm <- d[k] / max(d)
if valid[k] then
eta_norm <- min( eta[k] / max_valid(eta), ETA_BAD_NORM )   // cap = 2
else
eta_norm <- ETA_BAD_NORM      // moderate, non-infinite penalty
end
score[k] <- W_DIST * d_norm + W_ETA * eta_norm     // 0.70 / 0.30
end

[sorted, idx] <- sort(score, 'ascend')
new_rank      <- ids(idx)

// --- 3. hysteresis: commit a different order only on a clear gap
if new_rank == prev_rank then
```

```
ranking <- new_rank
else if | sorted(2) - sorted(1) | < HYST_FRAC * sorted(2) then
ranking <- prev_rank     // near-tie: chatter guard
else
ranking <- new_rank ;  prev_rank <- new_rank
end
end
```

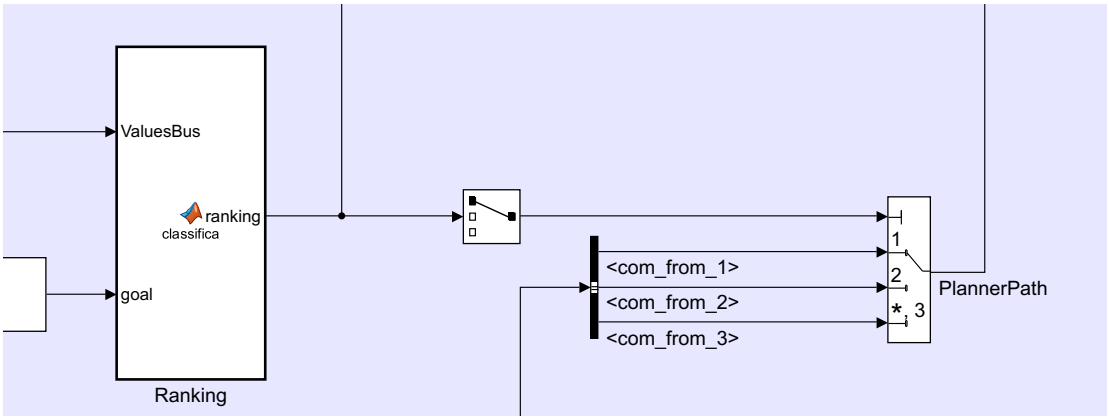


Figure 6.1: The `classifica` MATLAB Function block inside `Agent.slx`. located at `swarm_phase_5/(Agent)`: inputs are the broadcast `ValuesBus` and the goal position; the output ranking vector is shared with the planner gate, the collision check and the path trackers.

Figure 6.2 reports the ranking signal logged during a representative mission. Only the signal computed on board vehicle 1 is shown, because the three vehicles produce identical ranking signals: every agent runs the same `classifica` routine on the same broadcast state and therefore reaches the same ordering. Computing the ranking independently on every vehicle, rather than only on the leader, is a deliberate design choice in view of a future communication model with latency and packet loss. Should a vehicle lose contact with the leader, it would still know its own ranking and could elect itself leader. Having two leaders is far preferable to having one leader and a vehicle that no longer knows where its leader is: in the worst case the two leaders each run their own A* — expensive computationally, but the swarm keeps advancing and no vehicle is left behind.

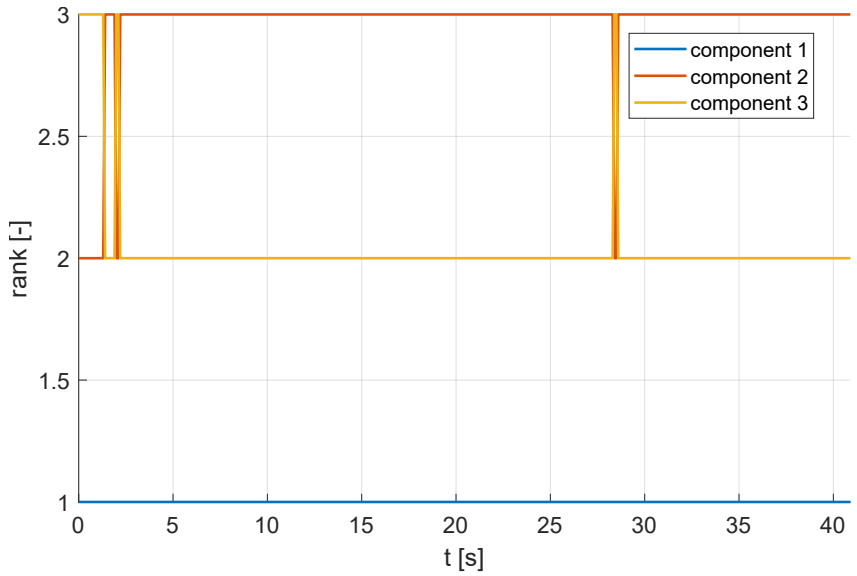


Figure 6.2: Swarm ranking logged on board vehicle 1 during a representative mission; the ranking signals of the three vehicles are identical.

6.3 Global Planner

The global planner produces the route the elected leader follows toward the mission goal. It runs the A* algorithm on the precomputed cost map of Section 4.7 — updated accordingly to what has been explored — scoring every node through $f(n) = g(n)+h(n)$, where $g(n)$ is the accumulated terrain cost defined by that map and $h(n)$ is an admissible estimate of the residual distance to the goal. Planning is centralized on a single agent — only the vehicle holding `rank == 1` runs the search, the others derive their reference from the formation logic — and it is carried by the slow planner task rather than the fast integration loop, so its cost is decoupled both from the swarm size and from the plant rate.

The planner is wrapped by the `SwarmPlanner` MATLAB Function block (Fig. 6.3), whose input and output ports are bound to the fixed vehicle slots `id1`, `id2`, `id3`, because Simulink requires ports of fixed size and order.

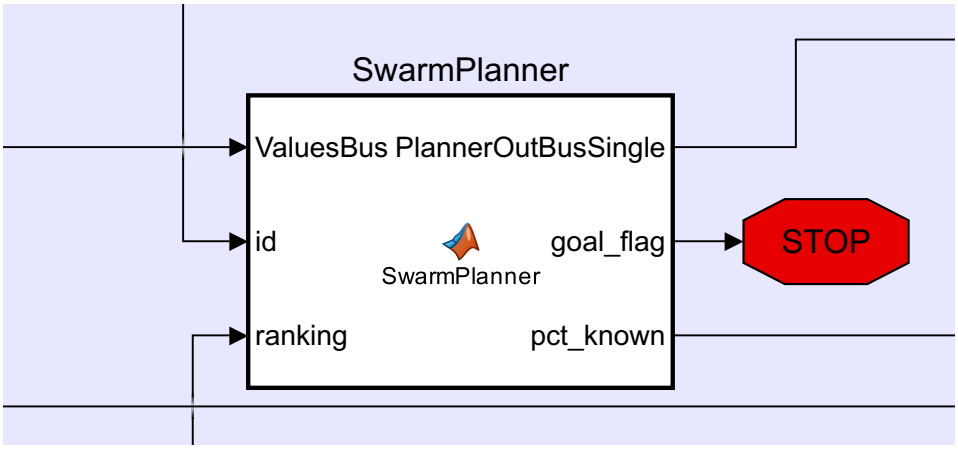


Figure 6.3: The `SwarmPlanner` MATLAB Function block, located at `swarm_phase_5/(Agent)`: inputs are the broadcast `ValuesBus`, the id of the vehicle and the position in the ranking; the outputs include a flag which stops the Simulink simulation, `pct_known` which is the observed cells, and the planned trajectories for leader and followers.

Leadership, however, is dynamic: the vehicle holding `rank == 1` changes from tick to tick (Section 6.2), so the block must route the search to whichever vehicle is currently the leader, and not blindly to `id1`. It does so in three steps. A rank-1 gate first lets only the elected leader run the search — a vehicle whose own `id` differs from `ranking(1)` returns a neutral, zero-path bus. The leader then calls `swarm_planner_rank_to_id`, which reorders the three fixed-slot states into ranking order so that the planner always receives the leader as its first argument; on return, the rank-ordered output buffers are scattered back onto the fixed `id1`/`id2`/`id3` slots by `write_rank_slot_to_id_slot`. The very same remap governs the follower references: should the ranking change between two ticks, the path-to-vehicle assignment follows automatically, with no special case inside the planner.

```
function out_bus = SwarmPlanner(values_bus, self_id, ranking)  // Simulink block, fixed id1/id2/id3 ports
out_bus <- neutral_bus       // zero paths, mu_next = 1
if ranking(1) ~= self_id then return  end      // rank-1 gate: only the leader plans
states_id <- read_fixed_slots(values_bus)   // states in fixed id order: id1, id2, id3
buf_id    <- swarm_planner_rank_to_id(states_id, ranking, L, N_REPLAN)
out_bus   <- unpack_to_id_slots(buf_id)  // id1/id2/id3: path + mu_next + unknown_ratio
end

function buf_id = swarm_planner_rank_to_id(states_id, ranking, L, N_REPLAN)
states_rank <- states_id( ranking, : )    // reorder so the leader is the first argument
```

```
buf_rank    <- swarm_planner_helper2(states_rank, L, N_REPLAN)  // plan in
    ranking order
buf_id      <- scatter_rank_to_id(buf_rank, ranking)    // map each rank
    back to its id slot
end
```

The helper function is so structured: the reordered call lands in `swarm_planner_helper2`, the routine that performs the actual work. Reading the shared map and scenario from the model workspace, it:

(i) refreshes the shared perception — each of the three vehicles marks a disc of radius R_{vis} as `known` and compacts the soil under its footprint, through `update_friction_map`, modeling the multipass stiffening of μ towards $\mu_{\max}$;

(ii) forms the per-step cost map by copying `map.cost_all` and overlaying the unknown-cell penalty from Table 4.2;

(iii) runs the A* search, `astar_path`, for the leader — but only once every `N_REPLAN` $= 2$ ticks ($0.2\,\text{s}$ at the $0.1\,\text{s}$ planner rate, tunable in `swarm_phase_5/(Agent)/SwarmPlanner`), reusing the last valid path in between;

(iv) de-stairs the raw grid path with a shape-preserving spline, `smooth_path_spline` (`pchip`);

(v) builds the follower references from the rigid formation offsets, `make_follower_path`, falling back to `select_local_heading` when the direct ray is blocked;

(vi) emits the look-ahead telemetry, `mu_next` and `unknown_ratio`, consumed by the speed governor.

The flat buffer it returns is exactly what `swarm_planner_rank_to_id` scatters back to the fixed ID slots. Finally, the planner emits two look-ahead signals per vehicle for the speed governor:

- `mu_next`, the friction of the first known cell on the path beyond $0.9\,R_{\text{vis}}$ (defaulting to a conservative 0.3 when no value is available);
- `unknown_ratio`, the fraction of unknown cells inside a frontal box (side $\approx 2.29\,R_{\text{vis}}$) aligned with the vehicle heading.

These let the governor slow down ahead of low-grip patches and unexplored terrain.

```
function planner_out = swarm_planner_helper2(states_rank, L, N_REPLAN)  //
    states_rank(1) = leader
```

```
2
3  // --- 1. shared perception (vision + soil compaction), all 3 vehicles
4  for i = 1 : num_vehicles do
5  mark_known( map.known, pos[i], R_vis )    // disc of perceived cells
6  map.mu <- update_friction_map( map.mu, footprint[i] )  // mu -> mu_max
7  end
8
9  // --- 2. per-step cost map = precomputed map + unknown overlay
10 cost_local              <- map.cost_all  // local copy, no rebuild
11 cost_local[~map.known]  <- cost_unknown      // 1e3 (>> base class costs)
12
13 // --- 3. A* for the leader, every N_REPLAN = 2 ticks; else reuse last path
14 if replan_due then
15 raw_path    <- astar_path( pos_leader, goal, cost_local, map, scenario )
16 leader_path <- smooth_path_spline( raw_path, ds = 1 [m], 'pchip' )  // de-
       stair
17 end
18
19 // --- 4. follower references: rigid formation in the leader frame
20 R <- rot( psi_leader )
21 for i = 2 : num_vehicles do
22 p_target <- pos_leader + R * offset_body[i] // rigid formation target
23 if ray_free( pos[i] -> p_target, map, step = 4 [m] ) then
24 follower_path[i] <- segment( pos[i] -> p_target, ds = 1 [m] )
25 else       // direct ray blocked
26 theta            <- select_local_heading( pos[i], p_target, map )
27 follower_path[i] <- ray( pos[i], theta, len = min(d, lookahead), ds = 1 [m]
       )
28 end
29 end
30
31 // --- 5. look-ahead telemetry for the speed governor
32 for i = 1 : num_vehicles do
33 mu_next[i]       <- mu( first known cell on path beyond 0.9*R_vis )
34 unknown_ratio[i] <- (#unknown cells in frontal box) / (#cells in box)
35 end
36 end
```

6.3.1 The A* Search

Operationally, the search runs on a grid downsampled to $\texttt{grid\_res\_m} = 4\,\mathrm{m}$, coarser than the native map because the planner does not need meter-level detail. It maintains, over the grid cells, the best known cost-to-come g and the priority $f = g + h$, together with an open set of frontier cells and a closed set of settled ones. Each iteration extracts the open cell of least f, closes it, and relaxes its 32 neighbors through Eqs. (6.2)–(6.3): a neighbor is opened, or re-opened, whenever the candidate g improves on its stored value, and the current cell is recorded as its parent. The loop terminates when the goal cell is extracted — the route is then rebuilt by following the parent pointers back to the start — or, defensively, when the open set empties (goal unreachable) or the cap $\texttt{max\_iter} = 5 \times 10^4$ is reached. Because the step cost already embeds both the terrain and the unknown-cell penalties, the returned polyline is the minimum-cost route across the perceived map, not merely the shortest one.

```
function path = astar_path(start, goal, cost, map, scenario)
downsample grid to grid_res_m ; locate start, goal cells (s, t)
g(:) <- +inf ;  g(s) <- 0 ;  f(s) <- heuristic(s, t)   // octile distance
open <- { s }
while open not empty do
  a <- argmin over n in open of f(n)        // cheapest frontier cell
  if a == t then return reconstruct(parent, t)  end    // goal reached
  move a from open to closed
  for each of the 32-neighbors b of a do
    if b in closed or out_of_bounds then continue  end
    ds    <- grid_res * sqrt(di^2 + dj^2)       // true Euclidean step
    length
    g_try <- g(a) + ds * 0.5 * ( cost(a) + cost(b) )   // average of the
    two cell costs
    if g_try < g(b) then
      parent(b) <- a ;  g(b) <- g_try ;  f(b) <- g_try + heuristic(b, t)
      open <- open + { b }
    end
  end
  if iter > max_iter then return empty  end  // safety cap = 5e4
end
return empty            // open exhausted: unreachable
end
```

Move cost. A* explores by expanding, from the current cell a, a set of reachable neighbors b; reaching b through a accumulates the running cost-to-come

$$g(b) = g(a) + c(a, b), \tag{6.2}$$

and the candidate is retained only if this $g(b)$ improves on any value found earlier. The move cost $c(a, b)$ is the geometric length of the step multiplied by the average of the two cell costs it connects,

$$c(a, b) = \underbrace{\Delta\sqrt{\Delta i^2 + \Delta j^2}}_{\text{step length}} \cdot \underbrace{\tfrac{1}{2}\big(C_a + C_b\big)}_{\text{mean cell cost}}, \tag{6.3}$$

with Δ the (downsampled) grid resolution and $(\Delta i, \Delta j)$ the index offset of the move. Averaging the departure and arrival costs charges a move for both the cell it leaves and the cell it enters: stepping from a cheap cell onto an expensive one already costs the mean of the two, so the planner feels a boundary one cell early instead of only upon arrival. Note that the cost averaging, unlike the gradient calculated in Section 4.7, is not performed logarithmically but is a standard average; this further discourages movement toward very high-cost cells (obstacles), as their cost values will heavily dominate the result.

Extended neighborhood. On a plain 8-connected grid the reachable headings are quantized to multiples of $45°$, which forces the raw path into coarse staircases. To widen the set of headings the search can express, the neighborhood is extended from the eight axial and diagonal moves to a set of 32 moves, adding knight-like jumps up to $(\pm 3, \pm 3)$ (Fig. 6.4). Since the step length in Eq. (6.3) uses the true Euclidean offset $\sqrt{\Delta i^2 + \Delta j^2}$, longer jumps are charged in proportion and none is implicitly favored by the discretization. The extra moves add eight heading families — approximately $18.4°$, $26.6°$, $33.7°$, $56.3°$, $63.4°$, $71.6°$ and their mirrors — so the polyline can follow oblique corridors far more closely before it is smoothed.

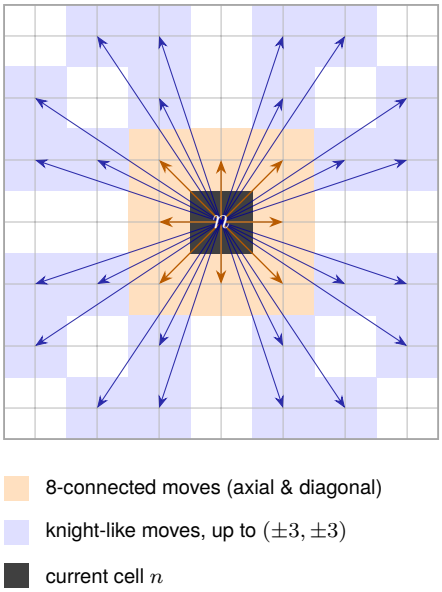


Figure 6.4: Extended A* neighborhood evaluated when the current cell n is expanded. Beyond the eight axial and diagonal steps (8-connected), the search also considers knight-like jumps reaching up to $(\pm 3, \pm 3)$, for 32 candidate moves in total. Each move is charged its true Euclidean length in Eq. (6.3), so the richer heading set reduces staircasing without biasing the search toward longer jumps.

In this way, for the same number of cells, the total cost decreases because the path is shorter, but above all, more linear and offering a wider range of heading choices, as shown in Figure 6.5.

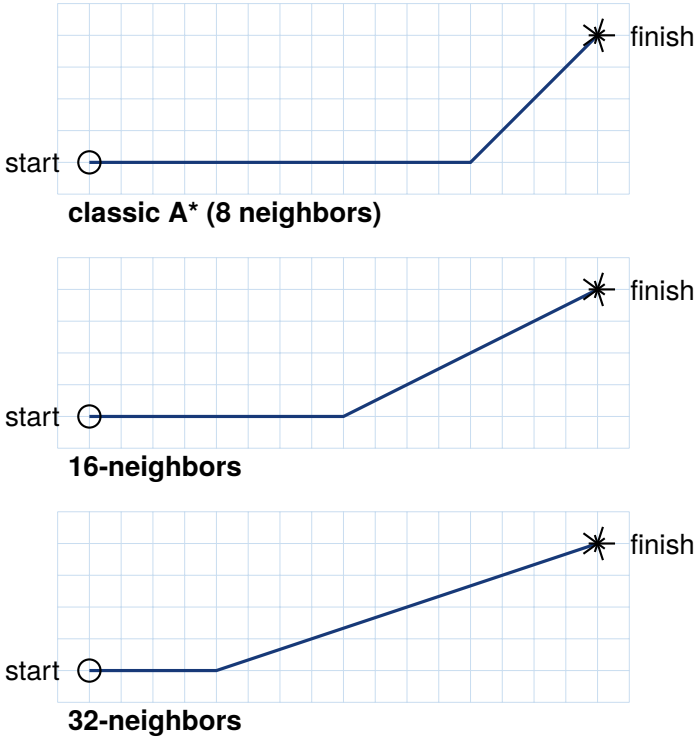


Figure 6.5: Comparison between classic, 16-neighbors and 32-neighbors A* path planning algorithm.

Heuristic. The heuristic adopted is the octile distance,

$$h(n) = \Delta \left(\max\left(|d_i|, |d_j|\right) + \left(\sqrt{2} - 1\right) \min\left(|d_i|, |d_j|\right) \right), \tag{6.4}$$

where Δ is the grid resolution and (d_i, d_j) the index distance to the goal. The octile distance is admissible on the plain 8-connected grid; with the extended knight-like moves it can slightly overestimate the residual cost, so strict optimality is knowingly traded for the finer heading resolution of the extended neighborhood.

The complete routine — the shared perception refresh, the unknown-cost overlay, the periodic A* search, the smoothing and the follower references — is summarized below.

6.3.2 Path Smoothing

The path returned by A⋆ is a holonomic, grid-constrained, piecewise-axial polyline made of unit steps along the extended neighborhood: its heading changes in discrete jumps and its curvature is, strictly, undefined at the vertices. A vehicle cannot follow such a reference cleanly, because the steering command is proportional to the path curvature and would contain spikes at every vertex. The polyline is therefore smoothed and re-sampled before it reaches the path tracker, which improves the curvature continuity of the reference, the tracking accuracy, the smoothness of the steering command and the compatibility with the vehicle dynamics.

The smoothing is performed in three stages by `smooth_path_spline`: a coarse uniform re-sampling at an intermediate arc-length step L_{res}, a number n of Laplacian smoothing passes with the two endpoints held fixed, and a final uniform re-sampling at the tracker step d_s.

```
function path = smooth_path_spline(path_in, ds_out)
if rows(path_in) < 3 then return path_in  end           // too short to
    smooth
// --- 1. coarse uniform resample (intermediate scale)
p <- resample_uniform(path_in, L_resample)              // L_resample = 6 m
if rows(p) < 3 then return resample_uniform(path_in, ds_out)  end
// --- 2. Laplacian smoothing: n_iter passes, endpoints anchored
n <- rows(p)
for it = 1 : n_iter do                                  // n_iter = 10
p_new <- p
for i = 2 : n-1 do      // interior points only
p_new[i] <- 0.50*p[i] + 0.25*p[i-1] + 0.25*p[i+1]   // [1/4 1/2 1/4] mask
end
p <- p_new     // p[1], p[n] never move
end
```

```
// --- 3. fine uniform resample for the path tracker
path <- resample_uniform(p, ds_out)                    // ds_out = 1 m
return path
end

function q = resample_uniform(path, ds)
s     <- cumulative_arclength(path)                    // s[1] = 0 ... s[
    end] = L
s_new <- 0 : ds : L   (+ append L if truncated)
q     <- interp1(s, path, s_new, "linear")             // arc-length
    parametrized
return q
end
```

Both re-sampling stages are parametrized by arc length. Given the input vertices $\{p_j\}$, the cumulative length $s_k = \sum_{j=1}^{k-1} \|p_{j+1} - p_j\|$ is computed and the curve is linearly interpolated at the uniform stations $s = 0,\, d_s,\, 2d_s, \ldots, L$, with $L = s_N$ the total length. This produces evenly spaced points and decouples the sampling density from the irregular spacing of the grid path.

The core of the routine is the Laplacian pass. At each iteration every interior point is replaced by the weighted average of itself and its two neighbors,

$$p_i^{(k+1)} = \tfrac{1}{4}\, p_{i-1}^{(k)} + \tfrac{1}{2}\, p_i^{(k)} + \tfrac{1}{4}\, p_{i+1}^{(k)} = p_i^{(k)} + \lambda \left(p_{i-1}^{(k)} - 2\, p_i^{(k)} + p_{i+1}^{(k)} \right), \qquad \lambda = \tfrac{1}{4}, \tag{6.5}$$

while the first and last points are never moved. The term in brackets is the discrete second derivative of the curve along its index, so (6.5) is one explicit Euler step of the diffusion (heat) flow $\partial p/\partial \tau = \partial^2 p/\partial s^2$ applied to the curve. High-spatial-frequency content, namely the grid staircase, decays fastest under this flow, while the low-frequency shape of the route is preserved; the fixed endpoints act as Dirichlet boundary conditions, so the start and goal positions are reproduced exactly at every replan.

Repeating the $[\frac{1}{4}, \frac{1}{2}, \frac{1}{4}]$ kernel is equivalent to convolving the curve with a Gaussian whose standard deviation, measured along the path, grows with the number of passes,

$$\sigma \;\approx\; \sqrt{2\, n\, \lambda}\; L_{\mathrm{res}} \;=\; \sqrt{\tfrac{n}{2}}\; L_{\mathrm{res}}. \tag{6.6}$$

Here σ is the effective smoothing length: features shorter than σ are removed, longer ones are kept. With the values used in this work, $L_{\mathrm{res}} = 6\,\mathrm{m}$ and $n = 10$, this gives $\sigma \approx 2.2\, L_{\mathrm{res}} \approx 13\,\mathrm{m}$, which erases the meter-scale staircase of the A* grid while leaving the tens-of-meters geometry of the route untouched.

Equation (6.6) also dictates how the three parameters are tuned. The number of iterations n is the main smoothing knob, but it cannot be increased freely: Laplacian

smoothing is also a curve-shortening flow, so each pass contracts the path slightly towards the straight chord between the anchored endpoints, cutting corners and pulling the reference towards the inside of the turns. Too many iterations therefore round off the corners that A* introduced to clear the obstacles and reduce the clearance; $n = 10$ is the compromise between a smooth reference and fidelity to the planned route. The intermediate step L_{res} enters (6.6) linearly and, in addition, sets a lower bound on the curvature radius of the output, which cannot be tighter than about L_{res}; it is set to $6\,\mathrm{m}$ so that the minimum radius stays within the turning capability of the vehicle. The diffusion weight λ is the per-iteration step of the explicit scheme and must satisfy $\lambda \leq \frac{1}{2}$ for stability: above this value the update overshoots and oscillates instead of smoothing, so it is fixed at the safe value $\frac{1}{4}$ and the smoothing strength is regulated through n alone. The tracker step d_s only sets how finely the smooth curve is sampled and is kept at $1\,\mathrm{m}$, independent of the smoothing strength: the coarse scale L_{res} removes the grid wiggles, the fine scale d_s re-densifies the curve for accurate tracking.

6.3.3 Leader–Follower Formation and Path-Planning

The leader — the vehicle currently elected by the ranking mechanism — follows the global path toward the mission goal. Each follower is assigned a desired offset in the leader body frame, so that the whole formation reorganizes consistently whenever the leadership changes.

The global target of follower i is:

$$\mathbf{p}_{i,\mathrm{target}} = \mathbf{p}_L + R(\psi_L)\mathbf{o}_i \tag{6.7}$$

where:

- $\mathbf{p}_L$ is the leader position;
- ψ_L is the leader heading;
- $\mathbf{o}_i$ is the desired formation offset;
- $R(\psi_L)$ is the rotation matrix.

In the simulations reported in this work, a line formation is adopted. The offsets are expressed in the leader body frame, with the positive longitudinal axis pointing forward along the leader heading and the lateral axis pointing to the left. The vehicle holding rank 1 is assigned the leader position, while the vehicles holding ranks 2 and 3 are placed behind it along the longitudinal direction:

Table 6.1: Formation offsets expressed in the leader body frame.

Rank	Role	Offset $\mathbf{o}_i = [x_L, y_L]^T$
1	Leader	$[0,\ 0]^T$ m
2	First follower	$[-10,\ 0]^T$ m
3	Second follower	$[-20,\ 0]^T$ m

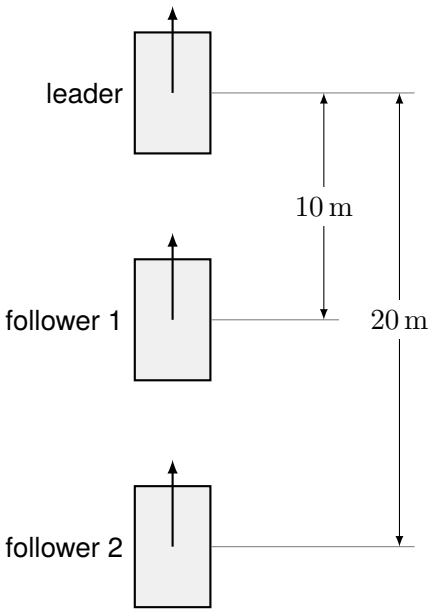


Figure 6.6: Swarm formation selected for the simulation: the two followers trail the leader along the body longitudinal axis with a $10\,\mathrm{m}$ and $20\,\mathrm{m}$ longitudinal spacing, respectively. The formation is tunable in `init_swarm_scenario`.

Other formation geometries can be implemented by modifying the offset matrix defined in `init_swarm_scenario`. For formations that are not purely longitudinal, such as triangular, diamond or circular arrangements, it is also advisable to adapt the MATLAB Function block responsible for writing the ranking, described in Section 6.2. In that case, the ranking should not only determine which vehicle is the leader, but also assign the remaining vehicles to the appropriate formation slots, for example left, right or rear positions, so that the geometric interpretation of each offset remains consistent when the leadership changes.

The followers do not plan: each follower is assigned a target point obtained by applying its rigid formation offset, expressed in the leader body frame and rotated by ψ_{leader}, to the current leader position. The reference handed to the tracker is simply the straight segment from the follower to this target, re-sampled at $1\,\mathrm{m}$: since the segment is regenerated at every planner tick, its straightness never accumulates error. Before being accepted, the direct ray is checked for obstacles by sampling the traversability grid every $4\,\mathrm{m}$; if it is blocked, an obstacle-free probe direction is chosen by `select_local_heading` and the local path is rebuilt along that heading, again re-sampled at $1\,\mathrm{m}$ and limited to

the local look-ahead length. The resulting reference is handed to the same tracker used by the leader.

6.3.4 Follower Path Planning

Although in principle every agent could run its own planner, in the proposed architecture the followers do not plan on board. The whole planning effort is centralized on the leader: the vehicle currently elected (`rank == 1`) is the only one that executes the A* global search, and in the same supervisory call it also builds a reference path for each follower. These follower paths are written on the `shared_bus` and broadcast to the swarm, so that each follower simply receives its reference as an input signal and hands it to the same tracker used by the leader. A follower therefore never runs a search itself; it only consumes the path computed for it by the leader. This choice keeps the followers computationally trivial and, more importantly, makes the planning cost independent of the swarm size, since the search is run once per replanning cycle regardless of how many followers are present.

The reference of follower i is generated in two steps by the routine `make_follower_path`. First, the relative goal is computed: the desired formation offset $\mathbf{o}_i$, expressed in the leader body frame, is mapped to the world frame through the leader pose, giving the target point $\mathbf{p}_{i,target} = \mathbf{p}_L + R(\psi_L)\,\mathbf{o}_i$ already introduced above. Second, rather than connecting the follower to this target with a blind straight segment — which could cut across an obstacle — the routine asks `select_local_heading` for a direction that is locally free, and lays the reference along it.

The function `select_local_heading` implements a simple fan (or beam) test. Starting from the direct bearing θ_0 that points from the follower towards its relative goal, it generates a set of candidate headings spread symmetrically around θ_0 within a maximum half-span. The candidates are examined in order of increasing deviation from θ_0, so that the heading eventually retained is always the one closest to the ideal direction. For each candidate the routine casts a straight ray of length equal to the follower–target distance and checks, cell by cell on the obstacle mask, whether that segment is collision-free. The first ray that does not intersect any obstacle cell is accepted and its heading returned; if no candidate is free, the direct bearing is kept as a fallback. The selected heading then defines a straight reference segment, resampled at a constant $1\,\mathrm{m}$ arc-length, exactly as for the leader path.

```
function follower_path = make_follower_path(i, p_leader, psi_leader, pos_i,
    map, scenario)
// --- 1. Relative goal: formation offset mapped into the world frame
o_i      <- scenario.formation.offset[i]        // desired offset, leader
   body frame
```

```
p_target <- p_leader + rot(psi_leader) * o_i    // p_{i,target} = p_L + R(psi_L) o_i

// --- 2. Direct bearing and distance from the follower to its target
theta_0  <- atan2(p_target.y - pos_i.y, p_target.x - pos_i.x)
d        <- norm(p_target - pos_i)

// --- 3. Choose a locally obstacle-free heading around theta_0
theta    <- select_local_heading(pos_i, theta_0, d, map, scenario)

// --- 4. Lay the reference along the chosen heading, resample at 1 m
p_reach       <- pos_i + d * [cos(theta), sin(theta)]
follower_path <- resample_segment(pos_i, p_reach, ds = 1 [m])
return follower_path
end
```

```
function theta = select_local_heading(pos, theta_0, d, map, scenario)
span <- scenario.formation.fan_half_span    // max deviation, e.g. 60 deg
N    <- scenario.formation.fan_samples      // number of rays, e.g. 13
step <- span / floor(N / 2)

// candidates ordered by increasing deviation: 0, +step, -step, +2*step, ...
for delta in [0, +step, -step, +2*step, -2*step, ..., +span, -span] do
theta_c <- theta_0 + delta
p_end   <- pos + d * [cos(theta_c), sin(theta_c)]
if is_segment_free(pos, p_end, map) then   // straight ray vs obstacle mask
return theta_c    // first free heading wins
end
end
return theta_0     // fallback: keep the direct bearing
end
```

The helper `is_segment_free` walks the straight segment from the follower to the candidate end point and returns false as soon as it meets a cell flagged as an obstacle in `map`, and true otherwise. The mechanism is illustrated in Figure 6.7: the follower must reach the relative goal defined by the leader, but an obstacle lies in between. The candidate rays aimed near the direct bearing (dashed) all cross the obstacle and are discarded; the first ray that clears it (solid) is selected, and the follower reference is built along this direction.

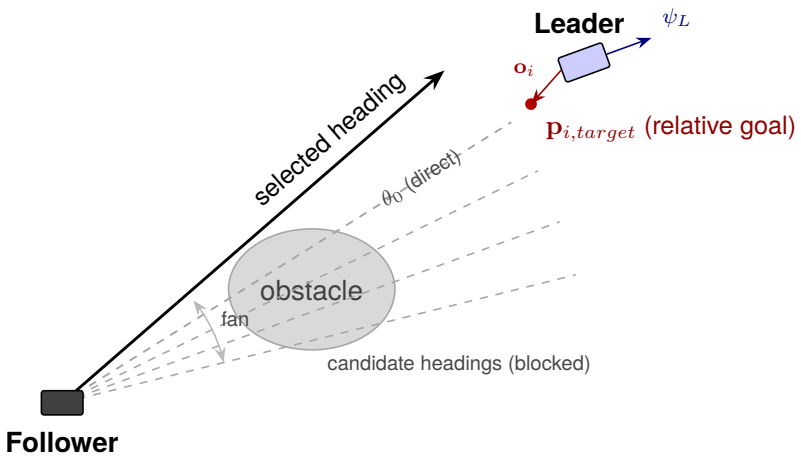


Figure 6.7: Local-heading selection performed by `select_local_heading`. The follower must reach the relative goal $\mathbf{p}_{i,target} = \mathbf{p}_L + R(\psi_L)\,\mathbf{o}_i$ assigned by the leader, but an obstacle lies on the way. Candidate straight headings are fanned out around the direct bearing θ_0; the ones that cross the obstacle (dashed) are rejected, and the first collision-free heading (solid) is retained as the follower reference direction.

6.4 Obstacle Emergency Braking

The planner keeps the reference path away from obstacles and the governor slows the vehicle ahead of unknown terrain, but neither layer guarantees that a newly discovered obstacle right on the current path is dealt with before impact. A last line of defense therefore runs at the plant rate, independently of both: at every step each vehicle scans a frontal corridor as wide as its own footprint (plus a small lateral margin) on the known obstacle map. The scan starts at the vehicle nose — a dead zone prevents the vehicle from detecting itself — and extends for the kinematic braking distance plus a safety buffer,

$$D_{check} = \min\left(d_{nose} + d_{safe} + \frac{v^2}{2\,a_{brake}},\; d_{max} \right), \tag{6.8}$$

so that the look-ahead automatically grows with the speed. If any cell of the corridor is simultaneously known and obstacle, the flag commands a hard braking ($80\,\%$ of the brake authority) that overrides the throttle channel. Unknown cells deliberately do not trigger the brake — caution towards the unexplored is the governor's job through $v_{unknown}$ — and cells outside the map borders are ignored. The intervention instants are logged by the obstacle-braking flag analyzed in Figure 6.9).

```
        function flag = obstacle_brake(x, y, psi, v, map)
        flag    <- 0
        D_check <- min( d_nose + d_safe + v^2 / (2*a_brake), d_max )
        for s = d_nose : ds : D_check do                    // scan from the
    nose forward
```

```
5         for l = -w/2 : ds_lat : +w/2 do                    // corridor =
      footprint width
6         p <- body_to_global(x, y, psi, s, l)
7         if known(p) and obstacle(p) then                   // only KNOWN
      obstacles brake
8         flag <- 1 ;  return
9         end
10        end
11        end
12        end
```

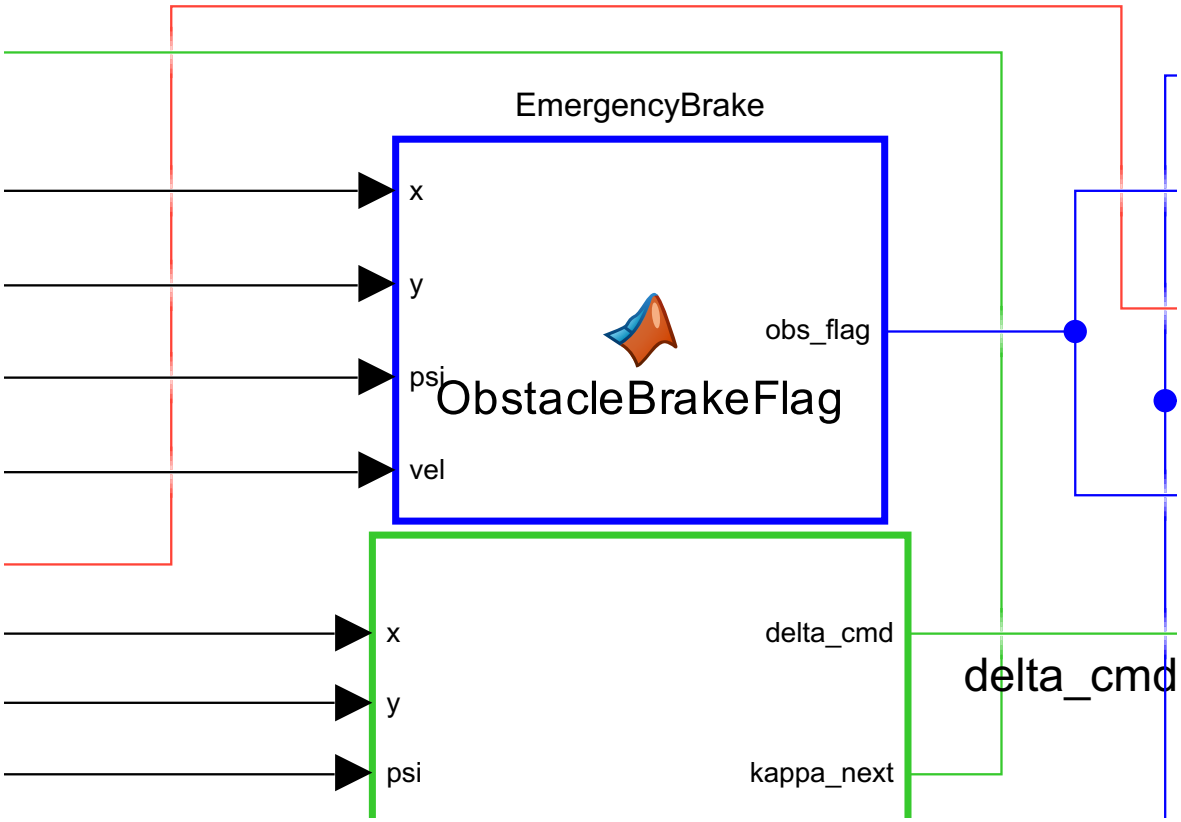
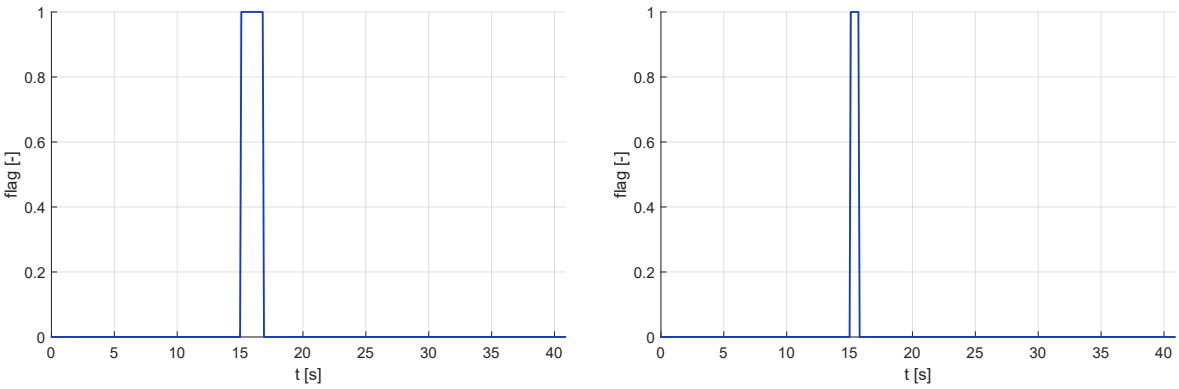


Figure 6.8: `EmergencyBrake` MATLAB fcn block. Located at `swarm_phase_5/(Agent)/(vehicle)/CONTROLLER/EmergencyBrake`.



(a) Obstacle brake flag — Vehicle 2

(b) Obstacle brake flag — Vehicle 3

Figure 6.9: Obstacle brake flag for vehicles 2 and 3. When the flag is equal to 1, the braking command is fixed to $\% = -400$, corresponding to 600 Nm of brake torque at each wheel before EBD and ABS modulation.

6.5 Anti-U-Turn Guard

A subtle failure mode of the leader–follower scheme emerges at leadership changes. When a vehicle is demoted from leader to follower, its new formation target can momentarily lie behind it; the same happens to a follower whose target is overtaken during a transient. A naive tracker would command a U-turn towards the target and, since the formation reference moves with the leader, the maneuver can re-arm repeatedly, leaving the vehicle spinning in place. Heading-based detection of this condition ("the goal is behind me") proved unreliable: while the vehicle rotates, the goal-behind flag toggles at every half-turn.

The adopted guard, applied to followers only, reasons instead on the path tangent: the target position is projected onto the direction of the final segment of the assigned path, giving a signed longitudinal coordinate s_{rel} that is invariant to the vehicle rotation. An overshoot is declared when $s_{rel} < -1\,\mathrm{m}$ while the vehicle is within $8\,\mathrm{m}$ of the target — the distance gate prevents arming on targets that are legitimately far behind, e.g. right after a leadership swap across the formation. While the guard is active the vehicle does not attempt the U-turn: its reference is replaced by a short straight stub, $6\,\mathrm{m}$ ahead along the current heading, so that it keeps moving and waits for the situation to resolve. The guard is released by the first of three conditions: the target is confirmed ahead ($s_{rel} > +1\,\mathrm{m}$) for $0.3\,\mathrm{s}$; the target keeps moving away for $2\,\mathrm{s}$ (the leader is driving off, waiting is pointless); or a maximum waiting time proportional to the distance of the new leader, $t_{wait} = 2\,d_L/\max(v_L,\,0.5)$, capped at $12\,\mathrm{s}$, expires. After release, a $4\,\mathrm{s}$ cooldown prevents immediate re-arming mid-maneuver; the current leader always bypasses the guard.

```
function path_out = anti_uturn_guard(path_in, pose, ranking)
if self == ranking(1) then return path_in  end   // leader: pass-through
```

```
3
4  t_hat <- tangent( last_segment(path_in) )      // final path direction
5  s_rel <- dot( target - pos, t_hat )      // signed longitudinal coord
6  overshoot <- ( s_rel < -1 [m] ) and ( || target - pos || <= 8 [m] )
7
8  if not guard and cooldown == 0 and rising_edge(overshoot) then
9  guard  <- true           // arm the guard
10 t_wait <- min( 2 * d_leader / max(v_leader, 0.5), 12 [s] )
11 end
12
13 if guard then
14 if (s_rel > +1 [m] for 0.3 s) or (moving_away for 2 s) or (waited > t_wait)
       then
15 guard <- false ;  cooldown <- 4 [s]     // release
16 else
17 path_out <- straight_stub(pos, heading, 6 [m])    // hold: no U-turn
18 end
19 end
20 end
```

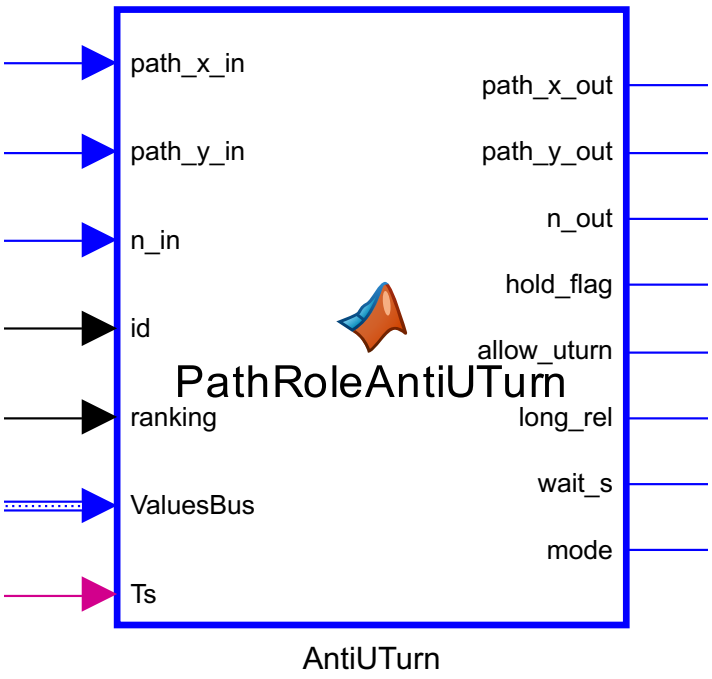


Figure 6.10: Detail of `PathRoleAntiUTurn` Simulink block. Located at `swarm_phase_5/(Agent)/(vehicle)/CONTROLLER`.

The `AntiUTurn` MATLAB function block activates if the follower starts in a position beyond the leader, or if it reaches a dead end, so to a point on the map with obstacles on

three sides—and needs to make a U-turn.

6.6 Path Tracking

Each vehicle tracks its assigned path through a look-ahead steering controller that combines a curvature feed-forward term with a feedback term acting on two geometric errors: the cross-track error and the heading error. At every step the tracker computes the closest path point, a look-ahead point, the two tracking errors, the local path curvature and the resulting steering command.

The look-ahead point is selected at a curvilinear distance L_d ahead of the path point closest to the vehicle. The look-ahead distance is scheduled with the vehicle speed and saturated within a fixed range,

$$L_d = \mathrm{clip}(L_{d,0} + k_v\,|v|,\ L_{d,\min},\ L_{d,\max})\,, \tag{6.9}$$

where $L_{d,0}$ is the base look-ahead distance, k_v a velocity-dependent gain, and $L_{d,\min}$, $L_{d,\max}$ the lower and upper bounds. A longer look-ahead at higher speed yields a smoother, more anticipatory steering action, while a shorter look-ahead at low speed keeps the tracking tight.

6.6.1 Path Tracker

The lateral controller is driven by two quantities, both evaluated with respect to the path point closest to the vehicle, $\mathbf{p}_c = (x_c, y_c)$, and to the local path heading θ_p, defined as the orientation of the path tangent at that point.

The cross-track error e_{cte} is the signed lateral distance between the vehicle position $\mathbf{p} = (x, y)$ and the path. It is obtained by projecting the position offset onto the unit vector $\hat{\mathbf{n}} = (-\sin\theta_p,\ \cos\theta_p)$ normal to the path:

$$e_{cte} = (\mathbf{p} - \mathbf{p}_c)\cdot\hat{\mathbf{n}} = -(x - x_c)\sin\theta_p + (y - y_c)\cos\theta_p. \tag{6.10}$$

It is expressed in meters. Its sign encodes the side of the path on which the vehicle lies: with the adopted convention a positive value indicates a displacement to the left of the path tangent and a negative value to the right, while $e_{cte} = 0$ corresponds to a vehicle lying exactly on the path.

One refinement applies to the leader only: since its reference is regenerated by the A* planner at every replan, the projection point used for e_{cte} is the look-ahead point rather than the closest point. Evaluating the lateral error further along the path filters the lateral jumps introduced by consecutive replans and lets the controller converge; the followers, whose formation reference is smooth, use the closest point.

The heading error ψ_e is the angular misalignment between the path tangent and the vehicle heading ψ:

$$\psi_e = \mathrm{wrap}(\theta_p - \psi), \tag{6.11}$$

where the operator $\mathrm{wrap}(\cdot)$ maps the angle to the interval $(-\pi, \pi]$, so that ψ_e is always the smallest rotation that aligns the vehicle heading with the path tangent. It is expressed in radians. A positive ψ_e means that the vehicle must rotate counter-clockwise to become parallel to the path, and $\psi_e = 0$ means that the vehicle is already aligned with it.

6.6.2 Steering Law

The steering command blends a curvature feed-forward term, which anticipates the geometry of the path, with a feedback term that regulates the two tracking errors to zero:

$$\delta_{cmd} = \underbrace{\arctan(L\,\kappa)}_{\text{feed-forward}} \underbrace{- K_y\, e_{cte} - K_\psi\, \psi_e}_{\text{feedback}}, \tag{6.12}$$

where L is the wheelbase, κ the local path curvature estimated over a short window of waypoints, and K_y, K_ψ the feedback gains on the cross-track and heading errors, respectively. Unlike the Stanley controller, the feedback term does not depend on the vehicle speed: on loose off-road terrain and at the low speeds considered in this work, a speed-dependent gain would weaken the steering action exactly where grip and controllability are most critical. A dead-band on e_{cte}, ψ_e and κ suppresses residual steering jitter on nearly straight segments, and the command is finally saturated at the maximum admissible steering angle $\delta_{\max}$.

Although the feedback term is purely proportional in both errors, it does not reduce to a simple proportional law on the cross-track error: the heading error supplies an implicit derivative action. For a vehicle traveling at speed v, the cross-track error evolves as $\dot{e}_{cte} \approx v \sin\psi_e \approx v\,\psi_e$ for small angles, so ψ_e is, up to the factor v, the time derivative of e_{cte}. Penalizing ψ_e therefore penalizes the rate at which the vehicle approaches or departs from the path: it damps the response and prevents the cross-track error from overshooting and oscillating around zero. The feedback term $-K_y\, e_{cte} - K_\psi\, \psi_e$ thus behaves as a PD controller on the cross-track error, which is why an explicit integral and derivative action, as in a full PID, is not required for path tracking.

6.6.3 Tracking Algorithm

Inputs: pose (x, y, ψ, v), `path`. **Outputs:** δ_{cmd}, κ_{next}, `goal_flag`.

```
function delta_cmd = path_tracker(pose, path)
i_near  <- argmin_i || path(i) - pos ||  // closest point
i_tgt   <- advance(i_near, arc_length = L_d) // look-ahead point
if self == ranking(1) then  // leader: project ahead,
e_cte <- signed_lateral_error(pos, path, i_tgt)  // filters A* replan jumps
else        // followers: smooth reference,
e_cte <- signed_lateral_error(pos, path, i_near)     // project on closest
    point
end
psi_e   <- wrap( heading(path) - psi )
kappa   <- mean_curvature(path, window = 12)
kappa_next <- max |kappa| over the window    // to the speed governor

delta_ff <- atan(L_wb * kappa)    // feed-forward
delta_fb <- -K_y*e_cte - K_psi*psi_e    // feedback (Stanley-like)
delta_cmd <- delta_ff + delta_fb

if |e_cte| < e_dead and |psi_e| < psi_dead and |kappa| < kappa_dead then
delta_cmd <- 0     // dead-band: no chatter
end
delta_cmd <- clip(delta_cmd, -delta_max, +delta_max)
end
```

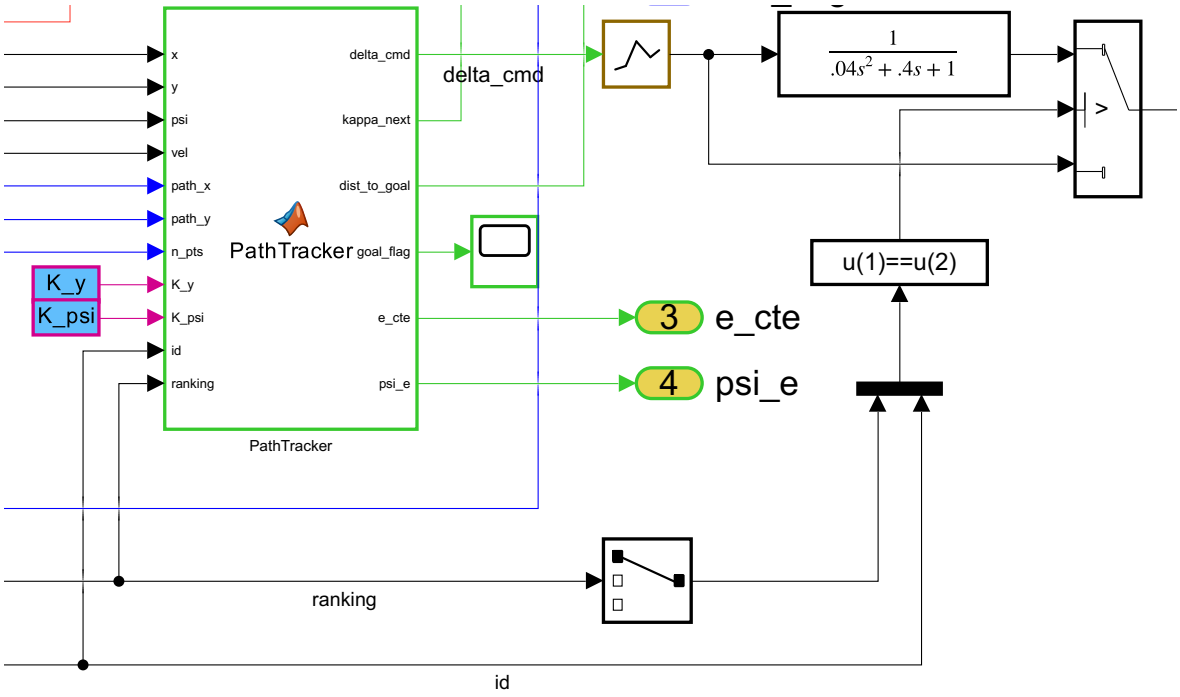


Figure 6.11: Detail of `PathTracker` MATLAB fcn block. Located at `swarm_phase_5/(Agent)/(vehicle)/CONTROLLER`.

6.6.4 Low-Pass Filter

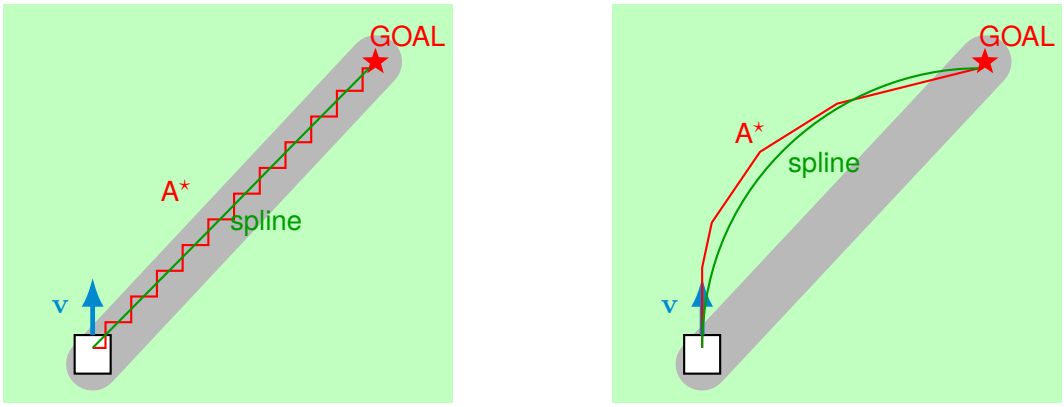
As shown in Figure 6.13, the cross-track error of the leader, which in this mission is always `ID1` as can be seen in Figure 6.2, is fairly large. The cross-track error e_{cte} is here defined as the signed lateral distance between the vehicle position and the reference path. With this kind of path tracking, and in this kind of environment, difficult and traveled at low speed, a good cross-track error would fall within 0.4–$0.5\,\mathrm{m}$.

For the followers, tracking the path planned as described in Section 6.3.4 turns out to be fairly benign. Their reference path, which is essentially a straight segment towards a formation point offset from the leader, changes at each step only as much as the relative position of the follower and its leader changes; its variation between two consecutive control steps therefore scales with the relative speed of the two vehicles and the fixed time step,

$$\|\Delta p_{ref}\| \;\approx\; \|v_L - v_F\|\, T_s, \tag{6.13}$$

so the reference evolves smoothly and the steering command it produces is smooth as well.

For the leader the situation is more involved. Although the path-smoothing stage applied to the A* output, described in Section 6.3.2, makes the path curved enough to be followed by a ground vehicle with realistic dynamics, it does not remove the underlying limitation of A*, which is a purely geometric planner. A* treats the vehicle as a holonomic point mass: it knows only the vehicle position on the map and ignores its heading and its non-holonomic constraints. As a consequence the first planned steps can start from one side of the vehicle, or even in the direction opposite to its current heading. Such a change in the plan occurs whenever the vehicle finds more favorable terrain to travel or an imminent obstacle to skirt. The fact that the cross-track error reaches values up to about four times the vehicle track width follows only from this reference re-planning: it does not mean that the vehicle is heavily oscillating about a stable, linear path.



(a) Holonomic (b) Non-holonomic

Figure 6.12: Holonomic versus non-holonomic planning over a favorable-terrain corridor (gray). The holonomic planner treats the vehicle as a point mass and ignores its current heading: the resulting path leaves the vehicle at an angle and forces it to oscillate. The non-holonomic planner accounts for the heading and the steering limits, so the path leaves the vehicle tangentially and turns towards the goal with bounded curvature.

The problem that follows is that the steering command δ sent to the wheels is as nervous as the cross-track error that drives it. Re-tuning the controller does not help, because it is the reference itself that moves rather than the tracking that is poorly tuned. To attenuate this, a second-order low-pass filter is placed on the steering command through the transfer function

$$H(s) \;=\; \frac{1}{0.04\,s^2 + 0.4\,s + 1} \;=\; \frac{1}{(1 + 0.2\,s)^2}. \tag{6.14}$$

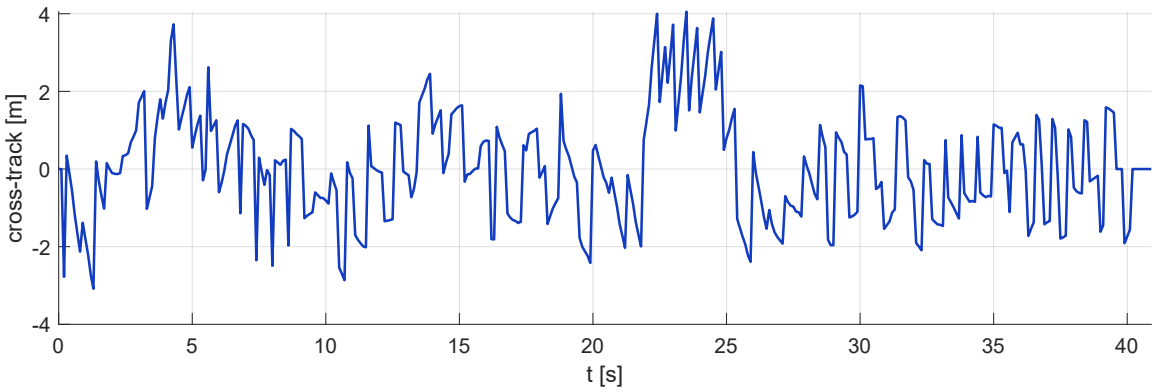
This is a critically damped filter ($\zeta = 1$) with a double real pole at $s = -5\,\mathrm{rad/s}$, that is a natural frequency $\omega_n = 5\,\mathrm{rad/s}$ and a low-frequency group delay

$$t_d \;\approx\; 2\tau \;=\; 2 \cdot 0.2 \;=\; 0.4\,\mathrm{s}. \tag{6.15}$$

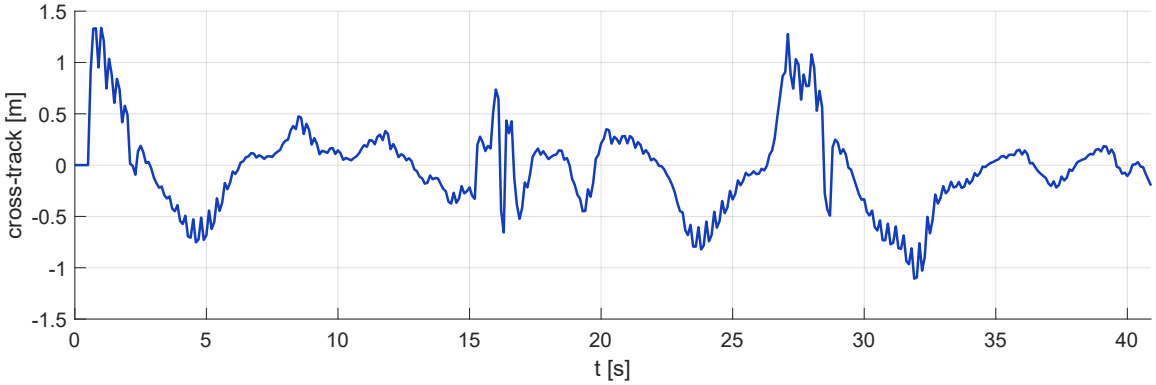
It therefore delays the steering by only about $0.4\,\mathrm{s}$ while strongly limiting the high-frequency content injected by the re-planning. A switch is placed upstream of the transfer function so that the filter is enabled only on the leader vehicle; the followers, whose reference is already smooth, receive the unfiltered command.

6.6.5 Path Tracking Error Results

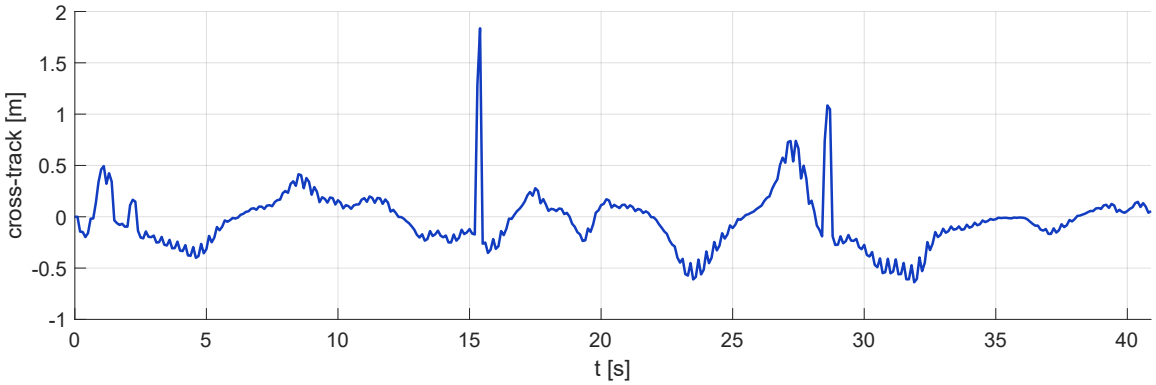
This section reports a preliminary analysis of the heading-related tracking errors logged for the three vehicles of the swarm. The data come from a single representative run and are meant to illustrate the qualitative behavior of the tracker; the campaign results are reported in the following sections.



(a) Cross-track error vehicle 1

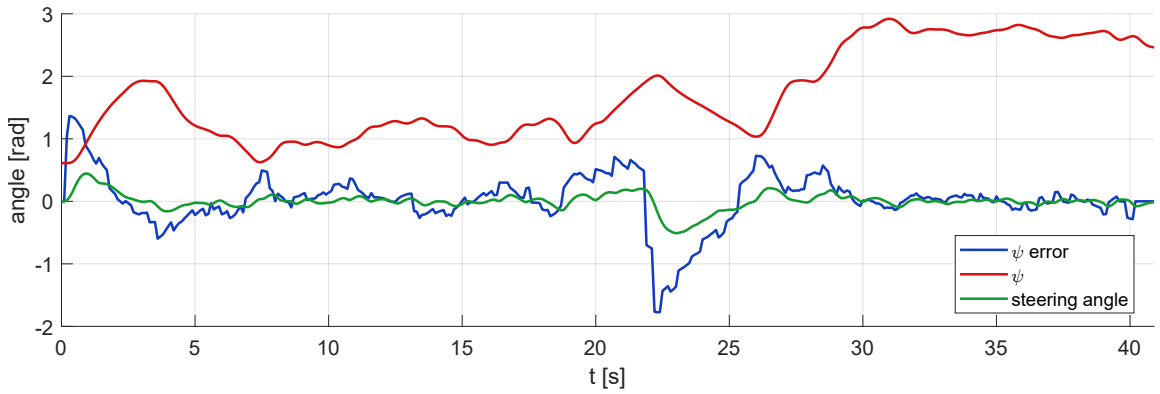


(b) Cross-track error vehicle 2

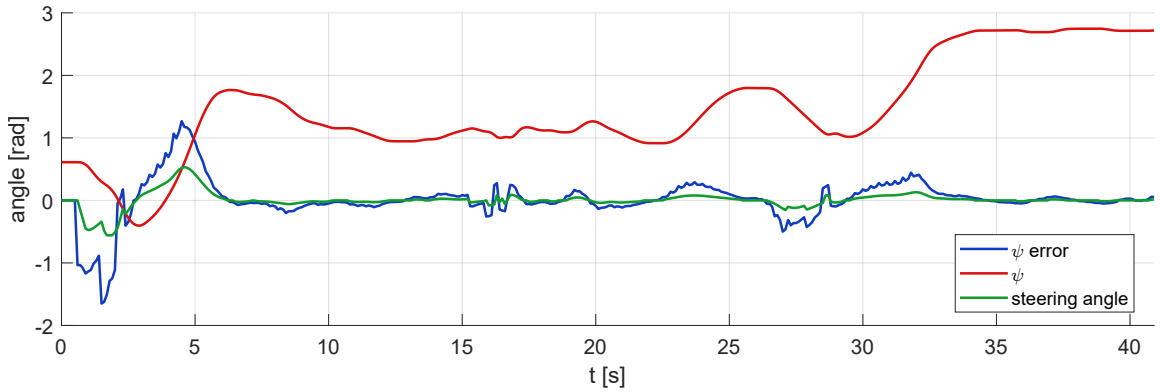


(c) Cross-track error vehicle 3

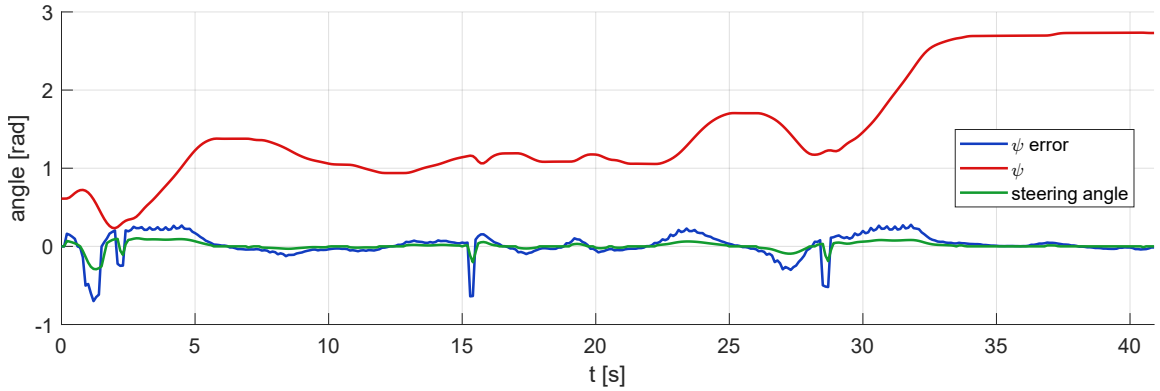
Figure 6.13: Cross-track error e_{cte} logged for the three vehicles during a representative mission; each panel is a rectangular time plot for one vehicle.



(a) Vehicle 1



(b) Vehicle 2



(c) Vehicle 3

Figure 6.14: Heading tracking error ψ_e for the three vehicles during the same mission. Each plot is overlaid with the heading angle ψ and the steering-wheel angle, so the error can be related to the vehicle motion.

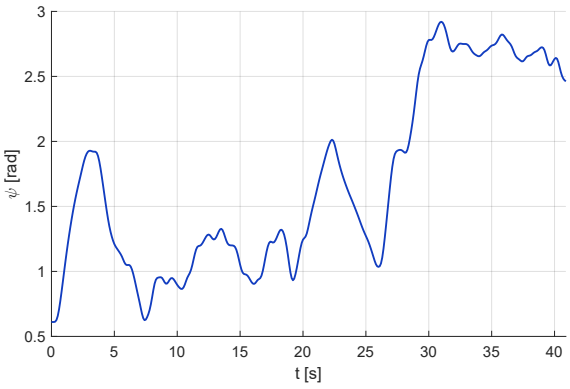
Figures 6.13 and 6.14 display a clear and consistent pattern: the followers track their reference very accurately, while the leader exhibits a much larger and oscillatory error.

The two followers keep their cross-track error within a few centimeters and their heading error close to zero for almost the entire mission. This is an important result, because it confirms that the steering controller itself is sound: the curvature feed-forward combined with the proportional–derivative feedback on e_{cte} and ψ_e follows the reference accurately whenever that reference is stable.

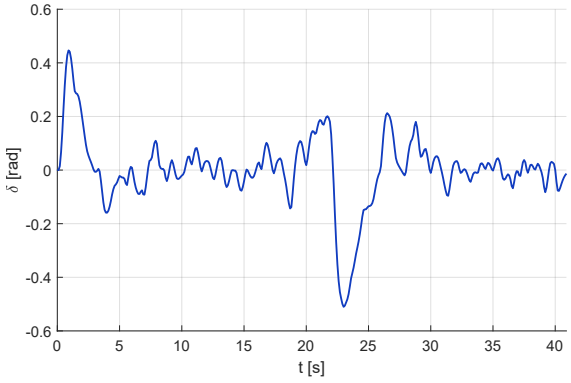
The leader behaves differently, and its larger error must not be read as a failure of the controller. The leader is the only vehicle that executes the global planner, and it therefore tracks a reference that is regenerated at every replanning step. Because the planner is grid-based, two consecutive plans can differ, so the path followed by the leader is not temporally stationary. Part of the leader error is thus a genuine tracking transient, and part is simply the cross-track error being evaluated against a reference that shifts at each replan. The followers, in contrast, track a smooth formation reference derived from the leader's smoothed path, which is far more stable; this is precisely why their error stays small.

This asymmetry is amplified by the deliberately conservative operating conditions adopted in this work: a high cruise speed combined with a small perception radius forces the planner to refresh its path frequently, and a frequently refreshed grid-based reference is intrinsically less smooth. The elevated leader error is therefore expected in this regime; reducing it is mainly a matter of choosing an adequate replanning rate — slow enough to give the tracker a stationary reference, fast enough to remain safe with respect to the perception horizon — rather than of retuning the tracker, whose quality is already demonstrated by the follower response. Furthermore, A*, as implemented in this work, is isotropic; it has no knowledge of the vehicle's heading, minimum turning radius, speed, available braking power, or any other physical parameter. Significant heading errors arise because A*, operating internally to the vehicle, generates an optimal path (post-smoothing) that may exit from the side of the vehicle. For this reason, as previously explained, the issue was mitigated by applying a low-pass filter to the controller's output, ensuring that commands based on cross-track and heading errors are not followed literally.

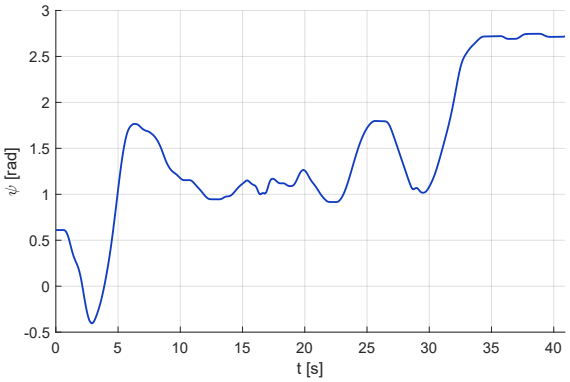
The steering activity that generates these errors is documented below: the heading histories of the three vehicles and the corresponding steering-wheel commands.



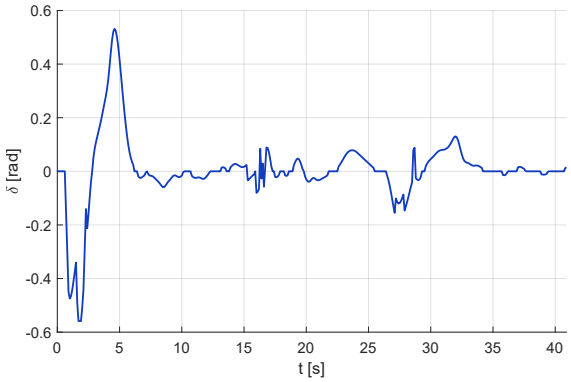
(a) Heading angle ψ — Vehicle 1



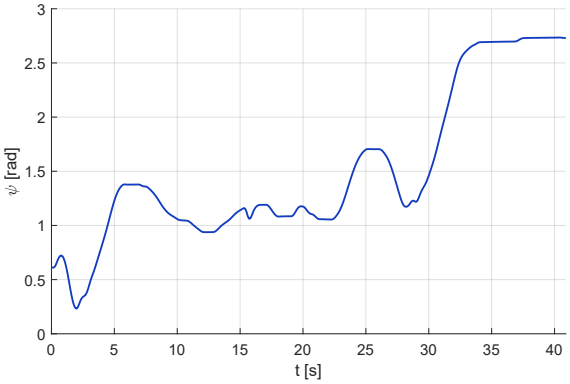
(b) Steering-wheel command δ — Vehicle 1



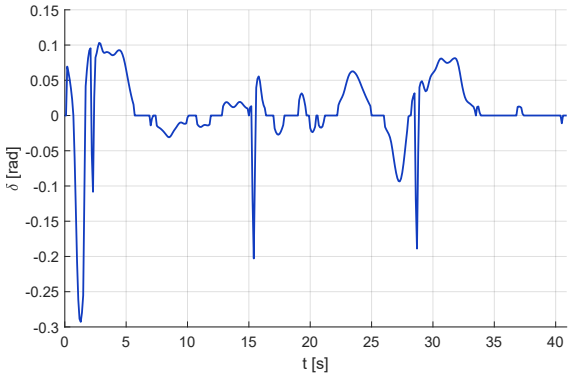
(c) Heading angle ψ — Vehicle 2



(d) Steering-wheel command δ — Vehicle 2



(e) Heading angle ψ — Vehicle 3



(f) Steering-wheel command δ — Vehicle 3

Figure 6.15: Heading angle ψ and steering-wheel command δ for vehicles 1, 2 and 3.

6.7 Speed Reference Governor

The speed reference governor limits the desired speed according to dynamic, environmental, and mission-related constraints.

The final reference speed can be expressed as:

$$v_{ref} = \min\left(v_{max}, v_{rollover}, v_{goal}, v_{unknown}, v_{LTR}\right) \tag{6.16}$$

where:

- v_{max} is the global maximum mission speed;
- $v_{rollover} = \sqrt{a_{y,roll}/|\kappa_{next}|}$ is the cornering limit from the static roll-over bound $a_{y,roll} = \eta_{roll}\, g\, T_{track}/(2h_G) - g\,|\sin\theta_{lat}|$ ($\eta_{roll} = 0.80$), evaluated on the look-ahead path curvature κ_{next} and corrected for the lateral road bank θ_{lat};
- $v_{goal} = \sqrt{2\, b_{eff}\, \max(d_{goal} - r_{goal},\, 0)}$ is the braking-distance limit to the goal, with an effective deceleration $b_{eff} = \min(\eta_{brake}\, \mu_{next}\, g\cos\theta + g\sin\theta,\ b_{max})$ that accounts for the look-ahead friction and the longitudinal slope;
- $v_{unknown} = \max(v_{max}\,(1 - 0.6\, r_{unknown}),\ v_{min})$ is a linear penalty on the unknown ratio ahead, floored at $v_{min} = 1\,\mathrm{m/s}$ so that the swarm keeps exploring instead of stalling;
- v_{LTR} is a staged cut driven by the lateral load-transfer ratio computed from the four vertical tire loads: the speed is cut to $0.7\,V$ above $LTR = 0.65$ and to $0.4\,V$ above $LTR = 0.80$, acting as an early roll-over precursor.

The load-transfer ratio that drives the staged cut is computed from the four vertical loads as

$$LTR = \frac{|(F_{z,FL} + F_{z,RL}) - (F_{z,FR} + F_{z,RR})|}{F_{z,FL} + F_{z,RL} + F_{z,FR} + F_{z,RR}}. \tag{6.17}$$

A friction-circle limit $v_\mu = \sqrt{a_{lat}^{\max}(\mu_{next})/|\kappa_{next}|}$, with a reserve subtracted for longitudinal control, is also computed; in the current implementation it is excluded from the committed minimum, the cornering envelope being entrusted to the roll-over bound (the two nearly coincide at nominal grip), while the look-ahead friction μ_{next} still shapes the envelope through the braking term b_{eff}.

Table 6.2: Main speed governor constraints.

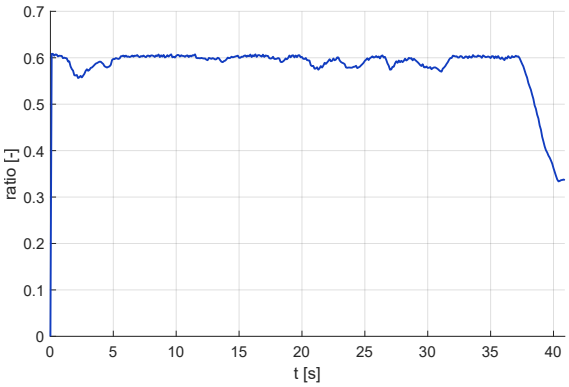
Constraint	Effect
Rollover risk	Limits lateral acceleration on the upcoming curvature, corrected for road bank.
Braking to goal	Ensures the vehicle can stop within the goal distance, aware of look-ahead friction and slope.
Unknown terrain	Linear speed cut with the unknown ratio ahead, floored at $1\,\mathrm{m/s}$.
Load transfer	Staged cut on the measured LTR ($0.7V$ above 0.65, $0.4V$ above 0.80).
Global limit	Mission-level maximum speed v_{max}.

The complete governor law, including the asymmetric rate limiter applied downstream of the binding constraint, is summarized in the following pseudocode.

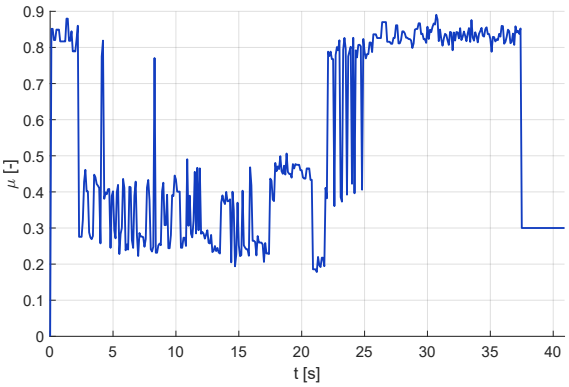
```
function v_ref = speed_governor(v_max, kappa_next, mu_next, d_goal,
    unknown_ratio, Fz, ...)
v_mu    <- sqrt( a_lat_allowed(mu_next) / |kappa_next| )  // friction
    circle (monitor only)
v_roll  <- sqrt( a_roll_safe(T_track, h_CG, bank) / |kappa_next| )
v_goal  <- sqrt( 2 * b_eff(mu_next, slope) * max(d_goal - r_goal, 0) )  //
    brake to goal
v_unk   <- max( v_max * (1 - 0.6 * unknown_ratio), v_min_explore )      //
    floor 1 m/s
v_ltr   <- staged_cut( LTR(Fz) )              // 0.7V above 0.65, 0.4V
    above 0.80

v_raw <- min( [v_max, v_roll, v_goal, v_unk, v_ltr] )   // binding
    constraint

// rate limiter on the rise only: climb gently, drop immediately
if v_raw > v_ref_prev then
v_ref <- min( v_raw, v_ref_prev + a_up * Ts )            // a_up = 3 m/s^2
else
v_ref <- v_raw                                           // immediate
    decrease
end
v_ref <- clip(v_ref, 0, v_max)
end
```

The asymmetric rate limiter (gentle acceleration, prompt deceleration) and the final saturation guarantee a smooth yet conservative reference, which the longitudinal MPC con-

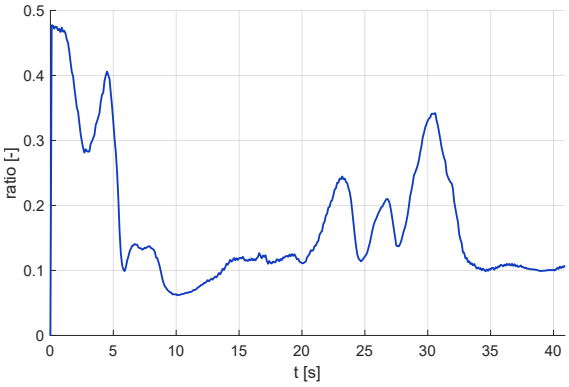
troller then tracks through the throttle and brake command.



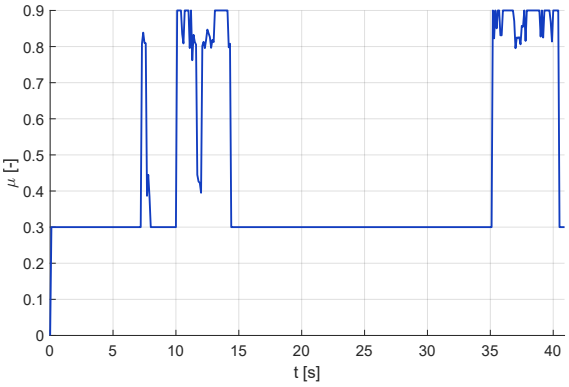
(a) Unknown ratio r_{unknown} — ID1



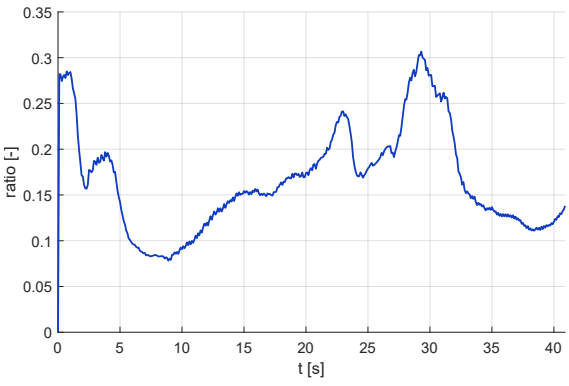
(b) Look-ahead friction μ_{next} — ID1



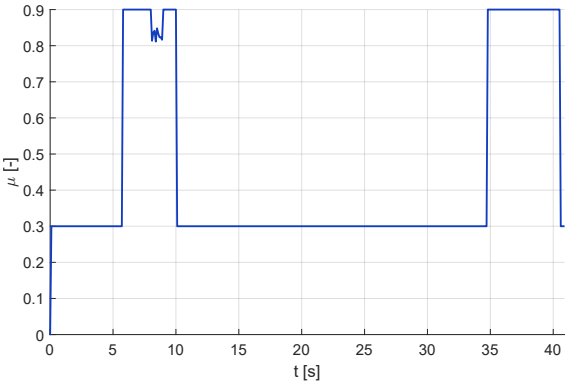
(c) Unknown ratio r_{unknown} — ID2



(d) Look-ahead friction μ_{next} — ID2



(e) Unknown ratio r_{unknown} — ID3



(f) Look-ahead friction μ_{next} — ID3

Figure 6.16: Unknown ratio r_{unknown} and look-ahead friction coefficient μ_{next} for each vehicle, used to compute v_{unknown} and v_{goal}.

6.8 Longitudinal Velocity Control via Model Predictive Control

6.8.1 Motivation

The longitudinal velocity loop was first closed with a PID acting on the velocity error. This worked, but two structural limitations kept surfacing. The controller is purely reactive: a change in the resistance to motion, such as a slope transition, produces a velocity error first and a correction only afterwards. Moreover, the actuator limits live outside the controller, enforced by downstream saturation blocks, so whenever the vehicle operates near those limits the integrator winds up and tracking degrades.

The PID was therefore replaced by an MPC, which removes both problems at the root. The actuator constraints become part of the optimization itself, so the controller never plans a command it cannot deliver, and the resistive force enters the prediction model as a measured disturbance: the controller reacts to a slope when the slope appears, not when the velocity error does.

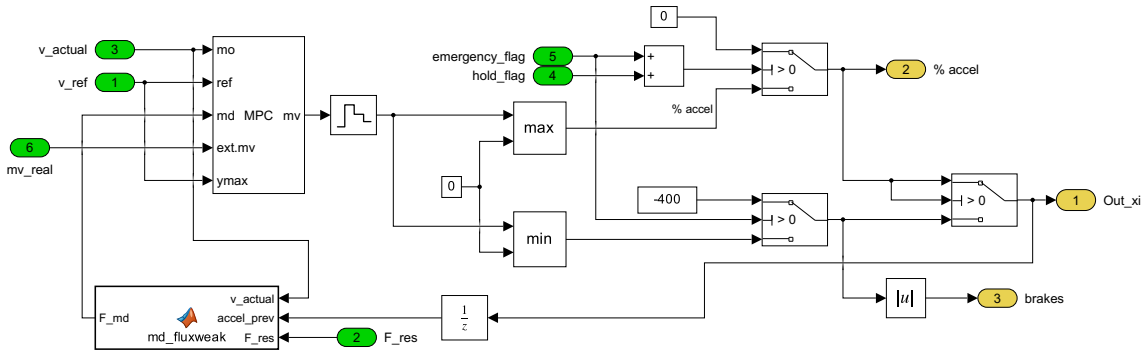


Figure 6.17: Detail of `Throttle & Brake` subsystem. Located at `swarm_phase_5/(Agent)/(vehicle)/CONTROLLER/Throttle\&Brake`.

6.8.2 Prediction Model

The MPC relies on an internal model of the plant to generate predictions. The longitudinal dynamics are described by the force balance on the equivalent translating mass:

$$m_{\text{eff}}\,\dot{v} = F_{\text{trac}} - F_{\text{res}} \tag{6.18}$$

where m_{eff} accounts for the rotational inertia of the wheels referred to the translational

degree of freedom:

$$m_{\text{eff}} = m + \frac{4\,J_{\text{wheel}}}{R_0^2} \tag{6.19}$$

The traction force at the wheel is related to the throttle opening $\xi\,[\%]$ through the motor map. Linearizing, $F_{\text{trac}} \approx k_v\,\xi$, with the combined motor–transmission–wheel gain

$$k_v = \frac{\tau_{\text{g}}\,k_\xi}{R_0} \tag{6.20}$$

where τ_{g} is the gear ratio, R_0 the rolling radius, and k_ξ the motor torque per unit throttle percentage. The slope k_ξ is taken from the constant-torque plateau of the motor map, so the linear gain implicitly assumes that peak torque is available at every speed; the validity of this assumption at high speed is discussed below. The resistive force collects rolling resistance, the gravitational slope component, and the flux-weakening correction term:

$$F_{\text{res}} = \left(f_0 + f_2\,v^2\right) mg + mg\sin\theta + F_{\text{fw}} \tag{6.21}$$

Choosing the velocity as the state ($x = v$), the throttle as the manipulated variable ($u = \xi\,[\%]$) and the resistive force as a measured disturbance input ($d = F_{\text{res}}$), the continuous-time state-space model is

$$\dot{x} = A\,x + B_u\,u + B_d\,d, \qquad y = C\,x \tag{6.22}$$

with the scalar matrices of a first-order, single-state system:

$$A = 0, \qquad B_u = \frac{k_v}{m_{\text{eff}}}, \qquad B_d = -\frac{1}{m_{\text{eff}}}, \qquad C = 1 \tag{6.23}$$

The negative sign of B_d reflects the fact that the resistive force opposes the motion. The continuous model is discretized via zero-order hold at the MPC sampling time $T_s = 0.03\,\text{s}$, yielding the discrete-time model used internally for prediction.

The term F_{fw} in (6.21) compensates for the main limit of the linear input gain. Above the base speed the motor enters flux weakening (Figures 5.16 and 5.17), and the torque actually delivered for a given ξ falls below the value that k_v, built on the peak-torque plateau, attributes to it. The deficit between the torque the internal model expects and the torque the envelope makes available is converted into an equivalent force at the wheel and injected through the measured-disturbance channel as an additional resistive term, so that the prediction model subtracts what the real motor cannot provide. This computation is carried out in the `md_fluxweak` MATLAB Function block, upstream of the MD port.

Cost Function

At each sampling instant t, given the current velocity measurement $v(t)$ and the current resistive force $F_{\mathrm{res}}(t)$, the MPC solves the following quadratic program over a prediction horizon of p steps:

$$\min_{\xi_0,\dots,\xi_{m-1}} \sum_{k=0}^{p-1} \Big[q \left(v_k - v_{\mathrm{ref}} \right)^2 + r_\Delta \left(\Delta \xi_k \right)^2 + r_u \, \xi_k^2 \Big] \tag{6.24}$$

where v_k are the predicted outputs, v_{ref} the filtered velocity reference, and $\Delta \xi_k = \xi_k - \xi_{k-1}$ the control increment. The tracking weight q sets how aggressively the reference is followed. The move-suppression weight r_Δ penalizes abrupt throttle changes, playing inside the optimization the role that a rate limiter would play outside it. The weight r_u on the absolute throttle value is set to zero: climbing a sustained slope requires a sustained throttle opening, and penalizing it would introduce a systematic tracking offset.

Penalizing $\Delta \xi$ rather than ξ produces an implicit integral action: the control law converges to whatever constant throttle drives the steady-state velocity error to zero, without an explicit integrator term.

Constraints

The actuator limits are imposed directly inside the QP at every step of the control horizon of m steps:

$$\xi_{\min} \le \xi_k \le \xi_{\max}, \qquad k = 0, \dots, m-1 \tag{6.25}$$

with $\xi_{\min} = -500\,\%$ and $\xi_{\max} = +100\,\%$. The positive range is the throttle opening; the negative range is mapped downstream to the brake actuator, and the asymmetry reflects the fact that the braking torque available at the wheel is five times the peak drive torque. No rate constraint is imposed on $\Delta \xi$. For an autonomous UGV there is no comfort argument for one, and the only physical delays involved, brake actuation and motor torque build-up, are negligible on the time scales of the simulation. A rate limiter used in earlier experiments was removed for this reason: it slowed the braking response without any physical justification.

The controller also reads back, through the `ext.mv` port, the command actually applied to the plant, which can differ from the optimal one computed by the QP because of the downstream overrides — the slip-based cut-off, the ABS logic and the emergency brake. Feeding the true input back keeps the state estimator consistent with what the plant really received, avoiding the wind-up that would arise from assuming the optimal command was applied.

6.8.3 Low-Pass Filter on the Velocity Reference

The velocity reference v_{ref} generated by the planner exhibits rapid fluctuations arising from the path-following logic. Fed directly to the controller, each local fluctuation is interpreted as a permanent reference change and produces an aggressive, bang-bang switching of the throttle command.

A first-order low-pass filter is therefore inserted upstream of the MPC reference port:

$$G_f(s) = \frac{1}{\tau_{\mathrm{f}}\, s + 1}, \qquad \tau_{\mathrm{f}} = 1\,\mathrm{s} \tag{6.26}$$

The filter is implemented as a State-Space block rather than a Transfer Fcn block, because the latter enforces zero initial conditions. The controllable canonical realization, obtained via `tf2ss`, is

$$A_f = -1, \qquad B_f = 1, \qquad C_f = 1, \qquad D_f = 0 \tag{6.27}$$

and the matrices are assigned by the function `make_vref_filter()`.

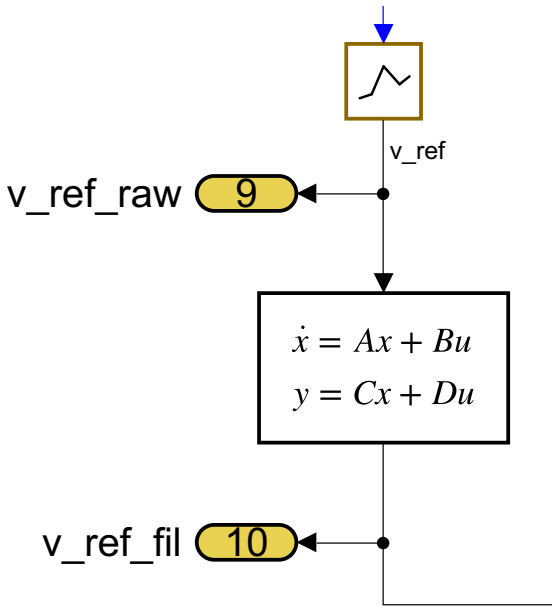


Figure 6.18: Filtering of reference speed: the transfer function receives `v_ref_raw` and gives `v_ref_fil` as output. Located at `swarm_phase_5/(Agent)/Vehicle(vehicle)/CONTROLLER`

The block initial condition is set to `scenario.vehicle8.velstart`, so the filter starts at the correct value and introduces no transient at $t = 0$. The MPC then tracks the trend of the reference and rejects the local fluctuations, at the cost of a lag of about one second on reference changes, which is acceptable because the planner never commands instantaneous velocity steps.

6.8.4 Parameter Tuning

The quantities monitored were the tracking RMSE on the filtered reference, the amplitude and regularity of the command $\xi(t)$, and the behavior at slope transitions, where the quality of the measured-disturbance feedforward becomes visible. The deployed configuration, reported in Table 6.3, favors reactivity. The tracking weight is high ($q = 1000$) and the move suppression moderate ($r_\Delta = 1$), enough to keep the command regular without slowing the braking response. The control horizon is kept deliberately much shorter than the prediction horizon ($m = 4$ against $p = 30$): on a first-order plant, letting m approach p with a small move-suppression weight collapses the optimization into a deadbeat law, and the resulting command is needlessly nervous; the gap between the two horizons restores the coupling between successive moves. Reference preview is not used, because in deployment only the filtered reference is available and the future trajectory of the raw one cannot be reconstructed. The reference is therefore held constant over the prediction window, so lengthening p brings no anticipation benefit; $p = 30$ steps ($0.9\,\mathrm{s}$) proved sufficient.

Steady-state accuracy is not entrusted to the horizon but to the estimator. The persistent velocity error caused by the mismatch between the first-order internal model and the 8-DOF plant, which lumps together brake-force distribution, tire grip and the neglected drivetrain dynamics, is treated as an output disturbance. The internal model is augmented with a near-integrating output-disturbance channel,

$$G_d(s) = \frac{20}{s + 0.05},$$

driven by white noise, and the measurement-noise variance on the velocity output is set low (0.01), so that the estimator attributes the prediction innovation to the disturbance rather than to the sensor. The disturbance estimate drifts until the predicted output matches the measurement, and the optimizer responds by commanding the additional throttle or brake needed to cancel the offset. This is integral action on the tracking error, but delivered through the constrained QP rather than by a free integrator: the estimate is shaped by the same actuator limits that bound the command, so the unbounded accumulation of classical windup cannot occur. A pure integrator $1/s$ was deliberately avoided as disturbance model: on a plant that is itself an integrator it renders the augmented state unobservable and the Kalman design fails, while the slow pole at -0.05 restores observability and keeps the corrective action nearly as strong.

A second mechanism acts on the output side. A soft upper bound on the predicted velocity is set at run time to the current reference through the `ymax` port, with $\mathrm{MaxECR} = 10^{-3}$, which makes the constraint nearly hard. Where the disturbance model corrects the tracking error after it appears, the output ceiling prevents the optimizer from planning trajectories above the reference in the first place. The two mechanisms are not redundant,

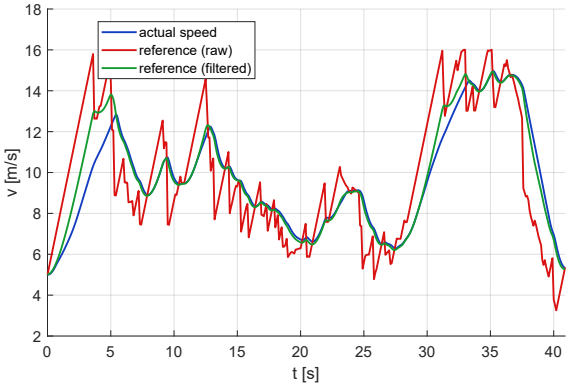
since they address different failure modes: model-mismatch windup the first, overshoot above v_{ref} the second, and removing either degrades the response.

Table 6.3: MPC parameters used in the simulation

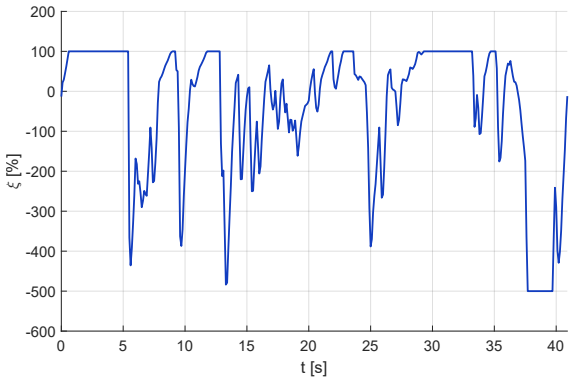
Parameter	Symbol	Value
Sampling time	T_s	$0.03\,\mathrm{s}$
Prediction horizon	p	30 steps (0.90 s)
Control horizon	m	4 steps
Tracking weight	q	1000
Move suppression	r_Δ	1
Absolute throttle	r_u	0.0
Throttle bounds	$[\xi_{\min},\, \xi_{\max}]$	$[-500,\, +100]\,\%$
Rate constraint	$\Delta\xi$	unconstrained
Output soft ceiling	$y_{\max}$	soft, $\mathrm{MaxECR} = 10^{-3}$
Output disturbance	$G_d(s)$	$20/(s + 0.05)$
Estimator meas. noise	–	0.01
Nominal velocity	v_0	$5\,\mathrm{m/s}$
Reference filter	τ_{f}	$1\,\mathrm{s}$

6.8.5 MPC Results

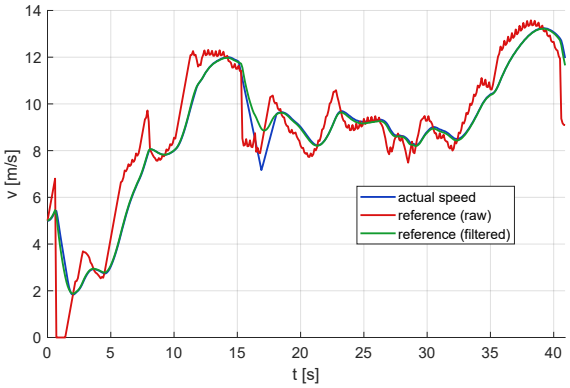
Figure 6.19 reports, for each vehicle, the actual speed against the raw and filtered reference produced by the governor of Section 6.7, and the corresponding throttle/brake command ξ generated by the MPC. The plots are logged from a representative mission: for each vehicle, the left panel shows the speed tracking and the right panel the command.



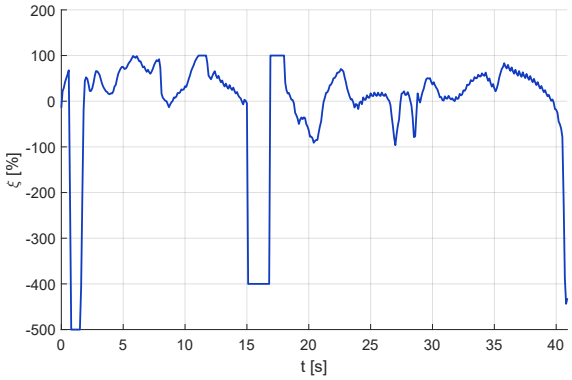
(a) Vehicle speed — Vehicle 1



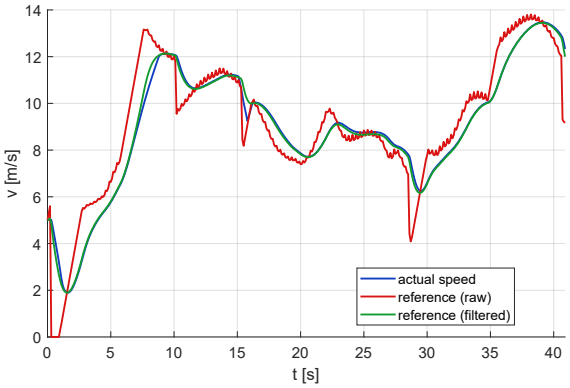
(b) Throttle/brake command ξ — Vehicle 1



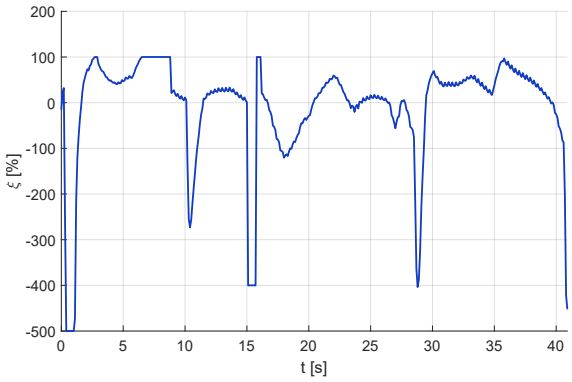
(c) Vehicle speed — Vehicle 2



(d) Throttle/brake command ξ — Vehicle 2



(e) Vehicle speed — Vehicle 3



(f) Throttle/brake command ξ — Vehicle 3

Figure 6.19: Vehicle speed and throttle/brake command ξ produced by the MPC for vehicles 1, 2 and 3.

Chapter 7

Mission Replay

While Section 5.12 analyzes the logged signals and physical quantities quantitatively, this chapter documents the simulated missions in visual form. Two sequences are reported. Figure 7.6 shows, for the same mission and the same instants, the internal state of the planner: the raw A* polyline, the smoothed spline, the path actually traveled and the progressive extent of the discovered area. Figure 7.9 finally repeats the experiment on a different semantic map with a different goal, with no change to the model or to the controller parameters. Each frame is reproduced at large scale, two per page, so that the terrain classes, the formation geometry and the planned paths remain readable.

7.1 Runtime Obstacle Event

The replays reported in this chapter unfold, by default, on a map that stays fixed for the whole mission. To exercise the reactive layer of the architecture — and, in particular, to show that the collision checker works in practice — a controlled runtime event was added to the reference scenario: at $t = 15\,\mathrm{s}$ a rigid elliptical obstacle is spawned on the map, as if a tree or a boulder suddenly dropped onto the path. The event is deterministic and well localized in space and time, so that its effect can be isolated cleanly in the logged signals instead of being confounded with the normal evolution of the mission.

The obstacle is injected directly inside `swarm_phase_5.slx`, as it is visible in 3.1 by a dedicated MATLAB Function block, `MapEventInjector`, driven by the model `Clock`; an adjacent `Display` block shows the `fired` flag switching from 0 to 1 at the trigger instant, giving an immediate visual confirmation while the simulation runs. The block invokes the routine `apply_map_event` as an extrinsic function, so the map mutation executes once, in interpreted MATLAB, exactly at the event time.

Figure 7.1: `MapEventInjector` MATLAB fcn block: `Digital Clock` provides simulation times, the generated event is shown with a flag in `Display` block. Located at `swarm_phase_5`.

The ellipse (semi-axes $18 \times 4\,\mathrm{m}$, oriented at $-30°$) is stamped onto the perceived world as a class-8 rigid obstacle. A subtle but essential detail concerns the cost map: since it is precomputed only once in `init_swarm_scenario` (Section 4.7) and never rebuilt online, the routine must explicitly override `map.cost_all` on the affected cells to `cost_obstacle` $= 10^6$; without this override the A* search would happily plan straight through the freshly created obstacle. The obstacle is deliberately not revealed everywhere at once (`MAKE_KNOWN` $=$ `false`): the swarm discovers it through the usual visibility radius — the realistic case — so that only cells already explored react immediately. A fire-once latch (`persistent applied`) guarantees the event is applied a single time, and the pre-event semantic background, together with the trigger instant, is recorded so that the replay renderer can switch the map background at $t = 15\,\mathrm{s}$.

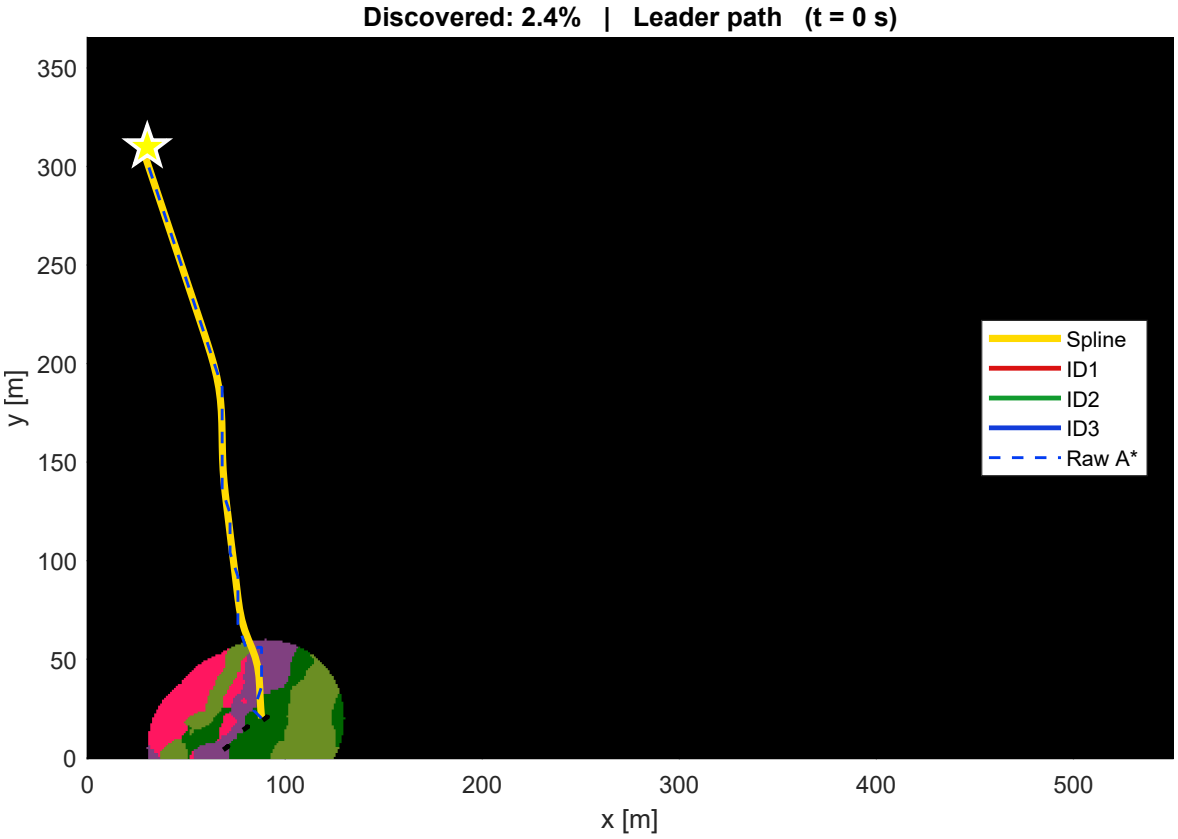
```
function fired = MapEventInjector(t)
coder.extrinsic('apply_map_event')
fired = apply_map_event(t)
end
function fired = apply_map_event(t)
if event already applied or t < T_EVENT then
return // fires at T_EVENT = 15 s
end
E <- cells inside the obstacle ellipse
mark E on the map as an impassable obstacle   // terrain, semantics and
    cost updated together
publish the updated map to the running simulation
mark event as applied ;  fired <- 1
end
```

This single event is the direct cause of the throttle reduction observed on vehicles 2 and 3 at $t \approx 15\,\mathrm{s}$: as the newly spawned obstacle enters their perception, the collision checker intervenes and the two give-way followers ease off the accelerator, exactly as the safety logic is meant to do. It is important to note that if the map contains such instances of vehicle failure—resulting in the spawning of obstacles—one must necessarily reduce speed or accept the risk of losing a vehicle, perhaps compensating by increasing

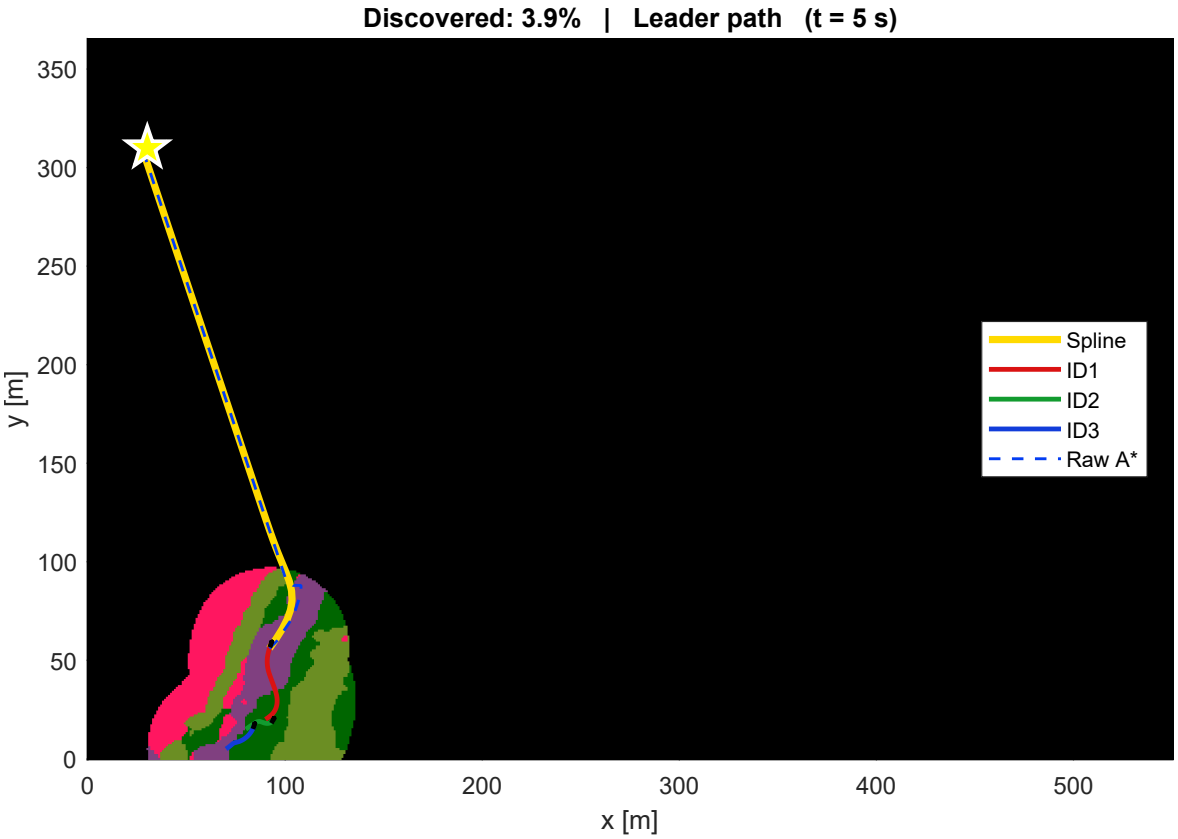
the swarm size. Yet, this represents the very trade-off offered by the swarm and the core objective of this entire study: if a vehicle fails, the swarm continues regardless. The second vehicle keeps moving because, in a simulation, vehicle failure is not typically factored in. In a real-world environment involving vehicle-to-vehicle communication, however, a vehicle is excluded from the ranking if no data is received from it or if it fails to move within a certain timeframe (indicating it has become stuck or broken down).

7.2 Planner State and Map Discovery

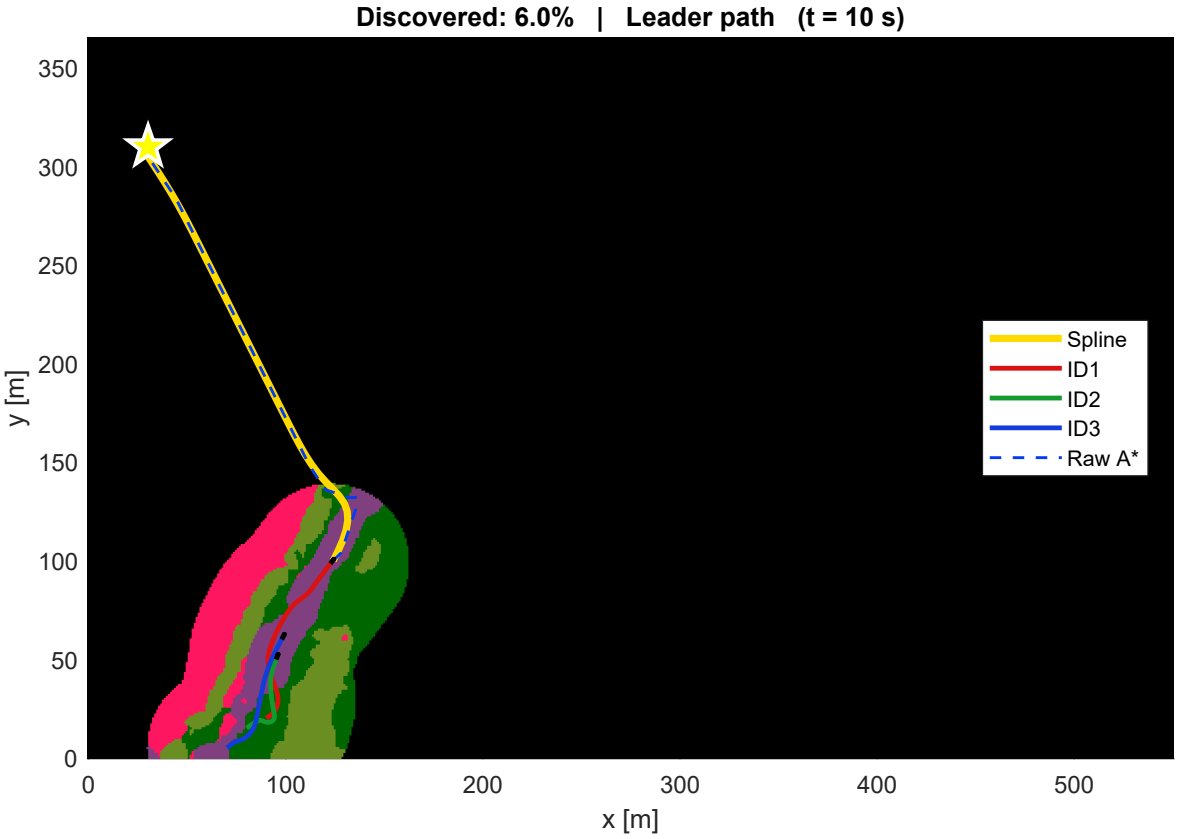
Figure 7.6 reports the planner state at the same instants of the previous sequence: each frame overlays the raw A* polyline, the smoothed spline actually handed to the tracker and the path traveled so far, together with the extent of the area discovered by the swarm. The early frames show the path aiming straight at the goal through the still-unknown region; as the terrain is discovered, the plan bends around the obstacles that have become visible.



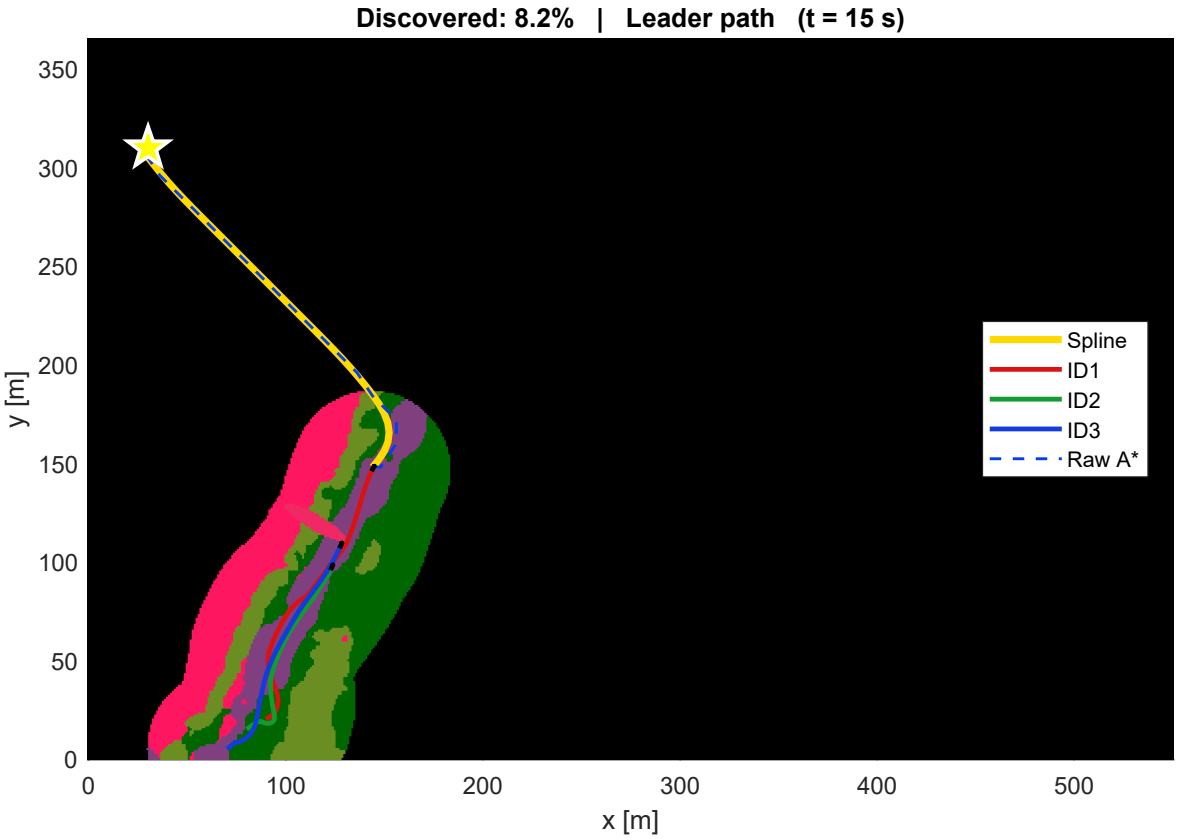
(a) $t = 0\,\mathrm{s}$



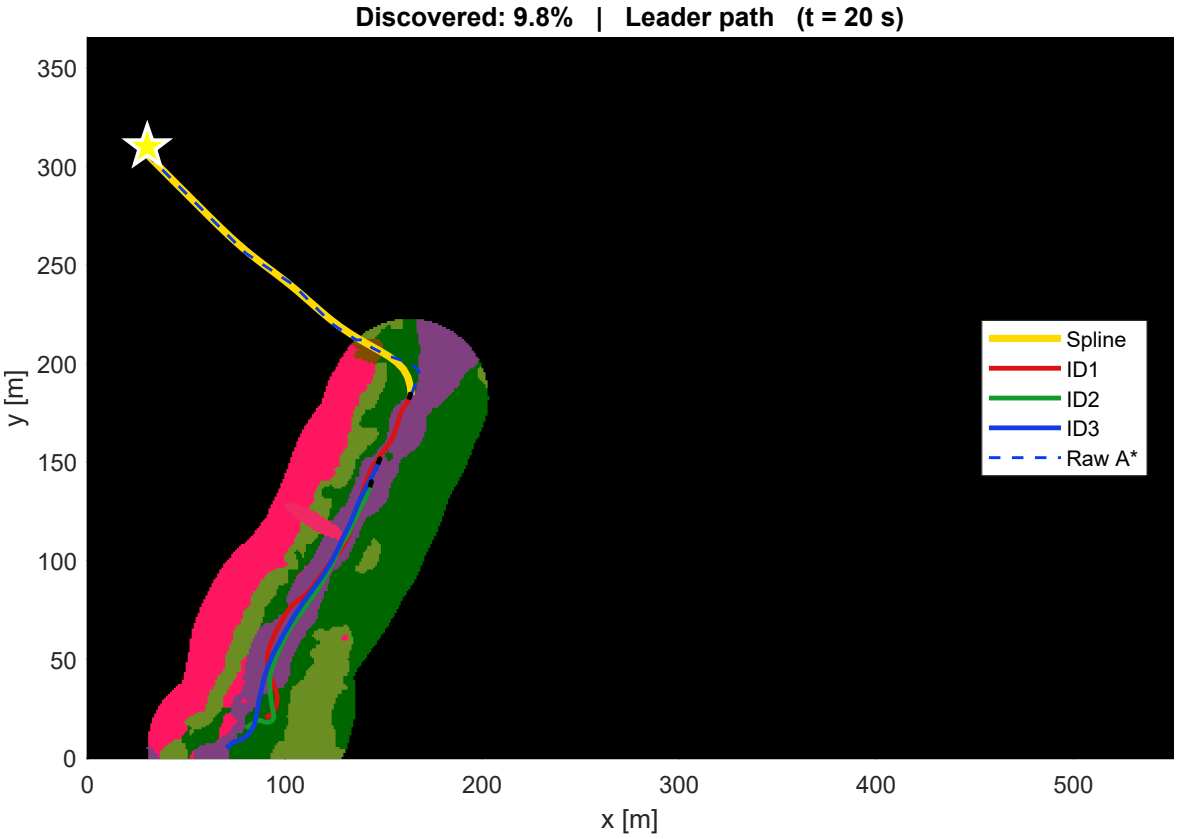
(b) $t = 5\,\mathrm{s}$



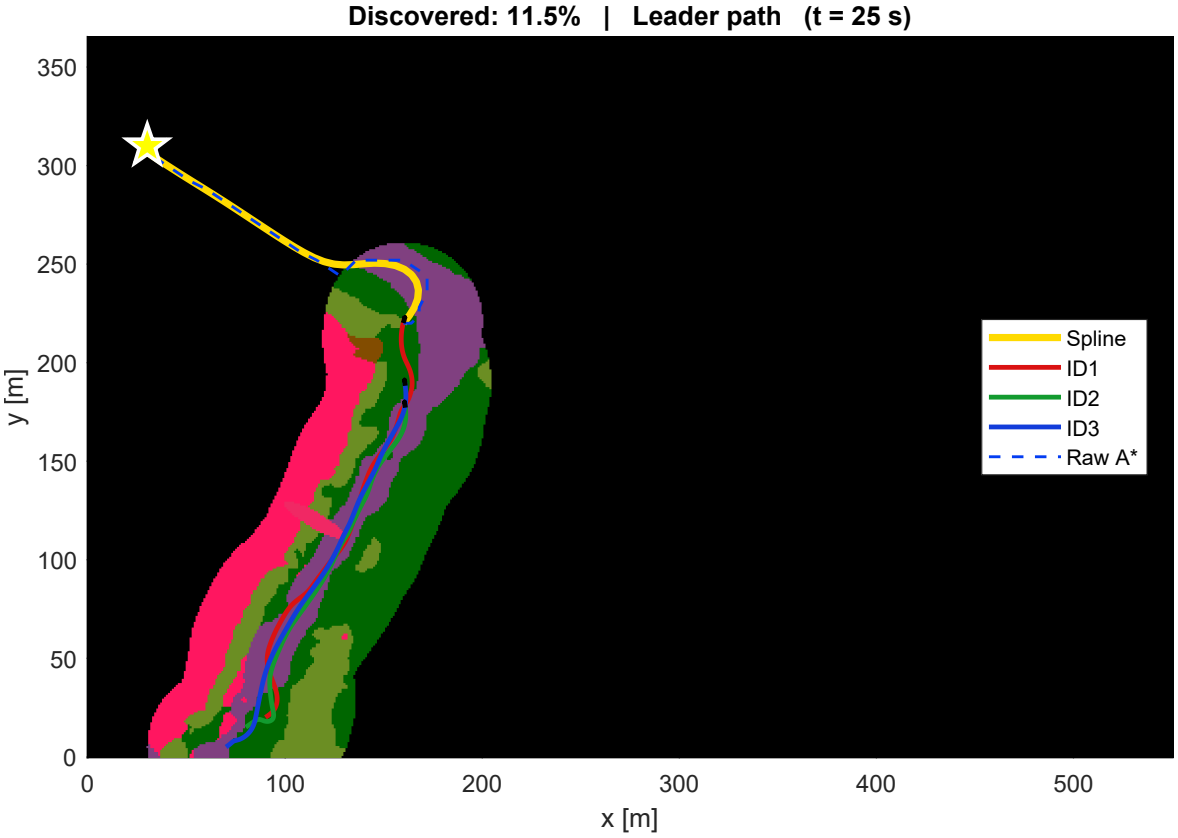
(a) $t = 10\,\mathrm{s}$



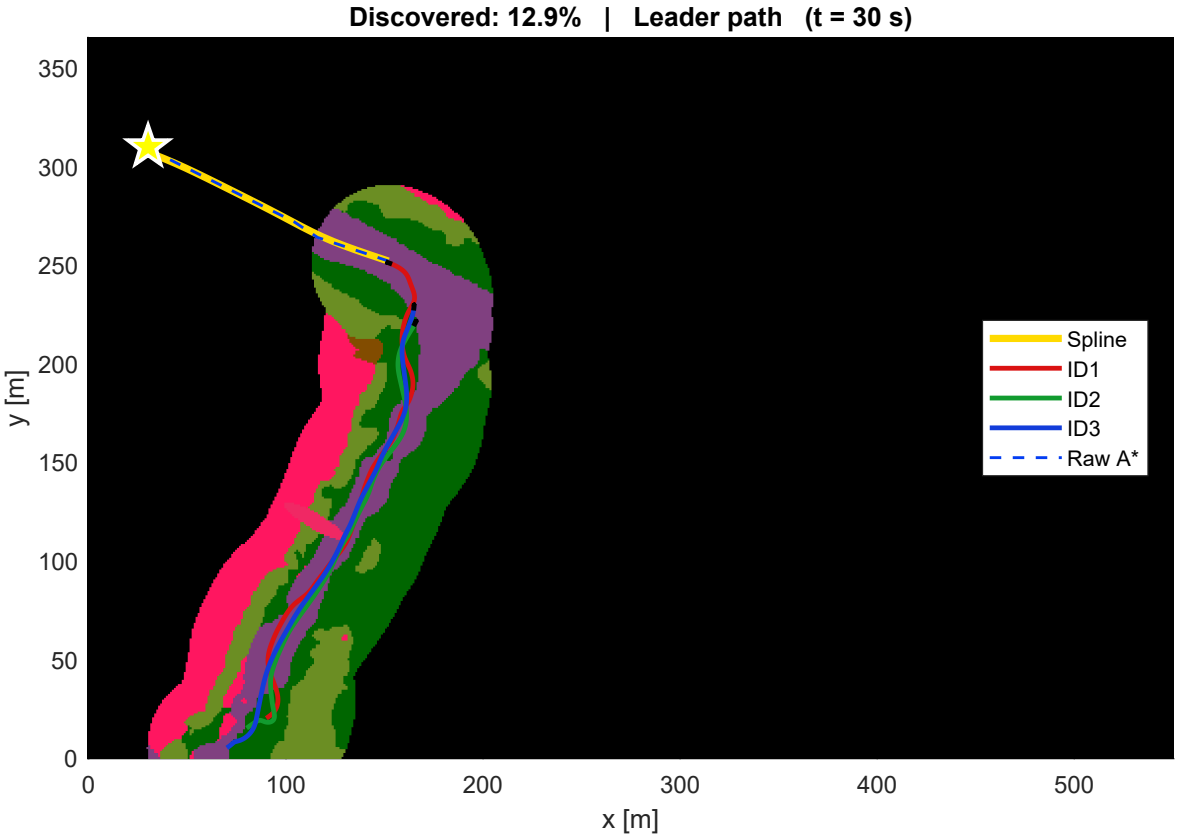
(b) $t = 15\,\mathrm{s}$



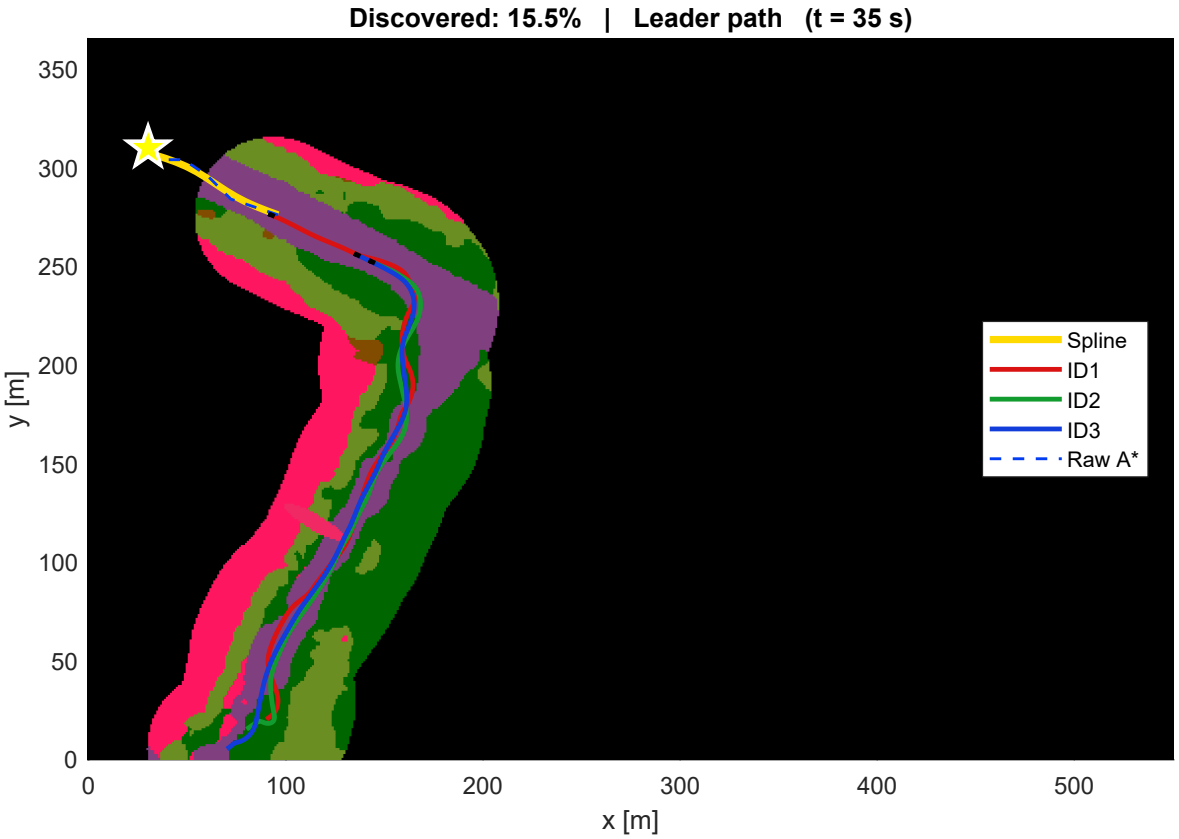
(a) $t = 20\,\mathrm{s}$



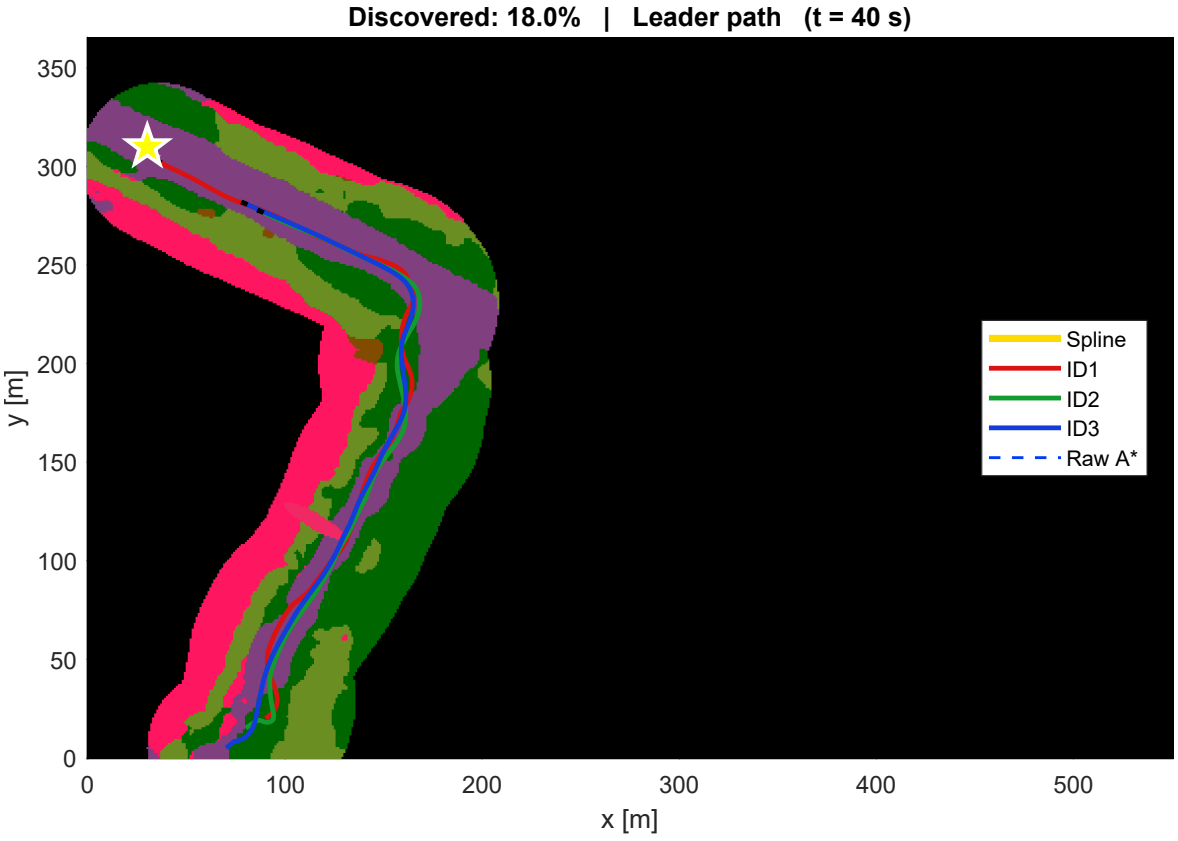
(b) $t = 25\,\mathrm{s}$



(a) $t = 30\,\mathrm{s}$



(b) $t = 35\,\mathrm{s}$



(a) $t = 40\,\mathrm{s}$

Figure 7.6: Mission replay — A* planner state (raw path, smoothed spline, traveled path and discovered area).

7.3 Replay on a Different Map

To verify that the dynamic leader election of Section 6.2 behaves as intended on a different terrain, the ranking signal logged during this mission is shown in Figure 7.7. As before, the three vehicles produce identical signals — each agent evaluates the same radial-ETA score on the same broadcast state — so only the signal of one vehicle is reported.

The point of interest is the contrast with the first scenario of Section 7.2. There, the vehicle best placed at the start kept the lowest radial ETA for the whole mission, the rank-1 assignment never changed, and the leader was effectively fixed. On this map the situation evolves differently: at some stage the leading vehicle's closing speed onto the goal drops — for instance because it must detour around an obstacle — while another agent gains a more direct line of sight and reaches a lower ETA. The ranking re-orders and the lead passes to a different vehicle, as the switch of the rank-1 trace in Figure 7.7 makes visible. The hysteresis band keeps the handover clean: it occurs once, when the ETA gap between the two best candidates exceeds the threshold, without any chattering around the crossing. This is exactly the behavior the mechanism of Section 6.2 was designed to produce, and the formation reorganizes consistently around the new leader, confirming

that role assignment, formation generation and re-planning all respond correctly to the change of leadership.

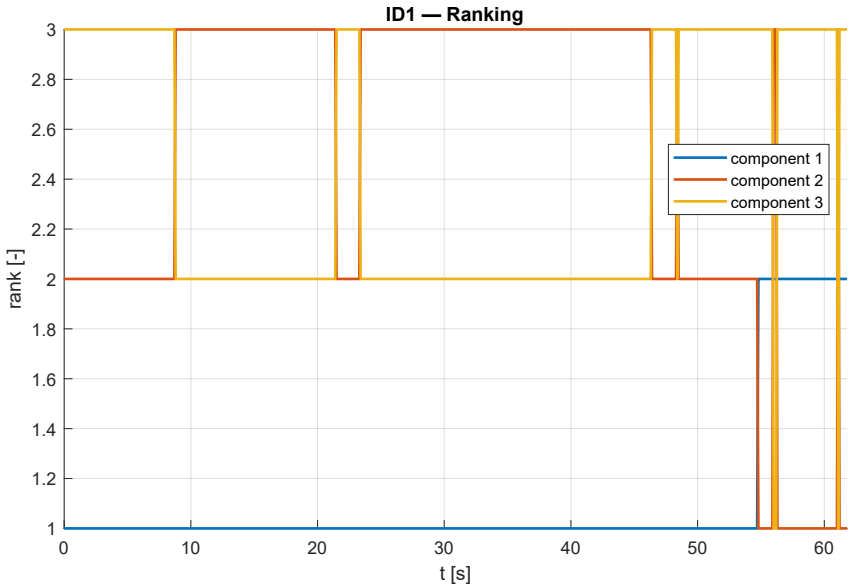
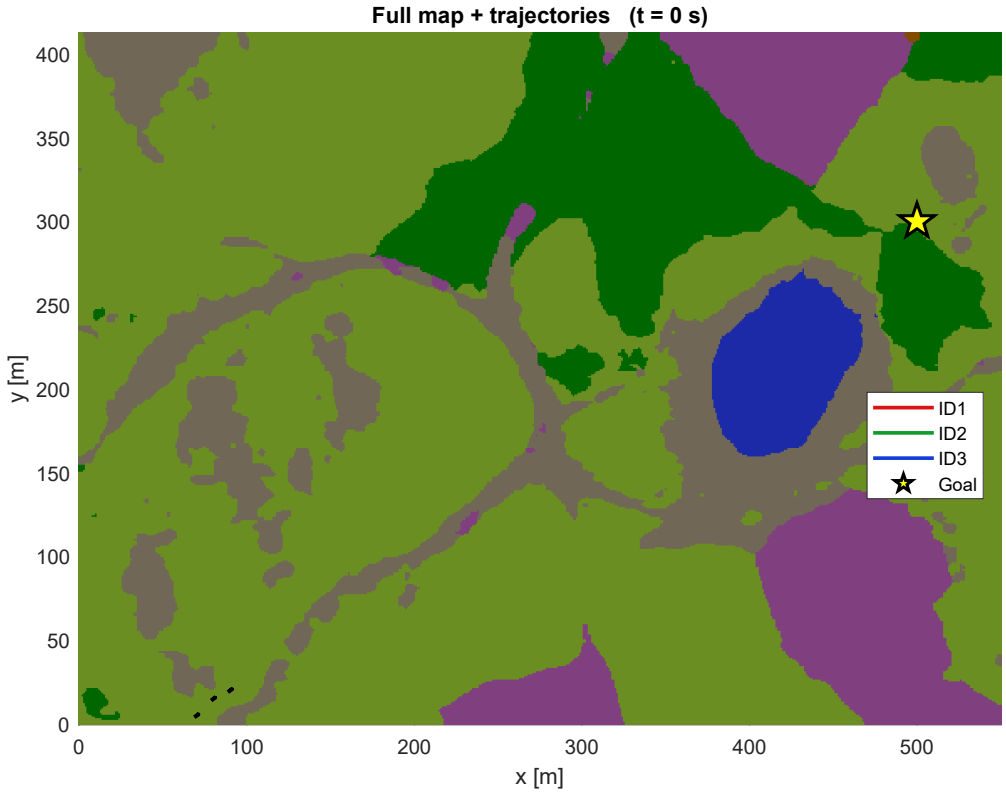
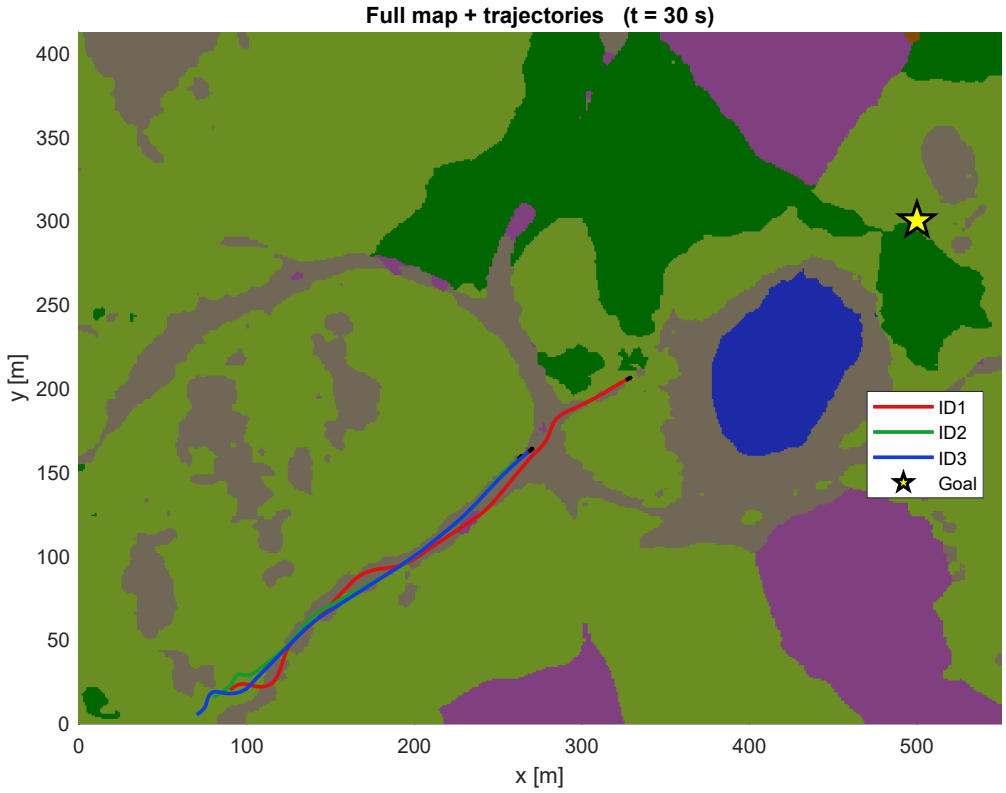


Figure 7.7: Swarm ranking logged on board one vehicle during the mission on the second map; the signals of the three vehicles are identical. Unlike the first scenario, the rank-1 assignment switches during the mission, so the leader changes.

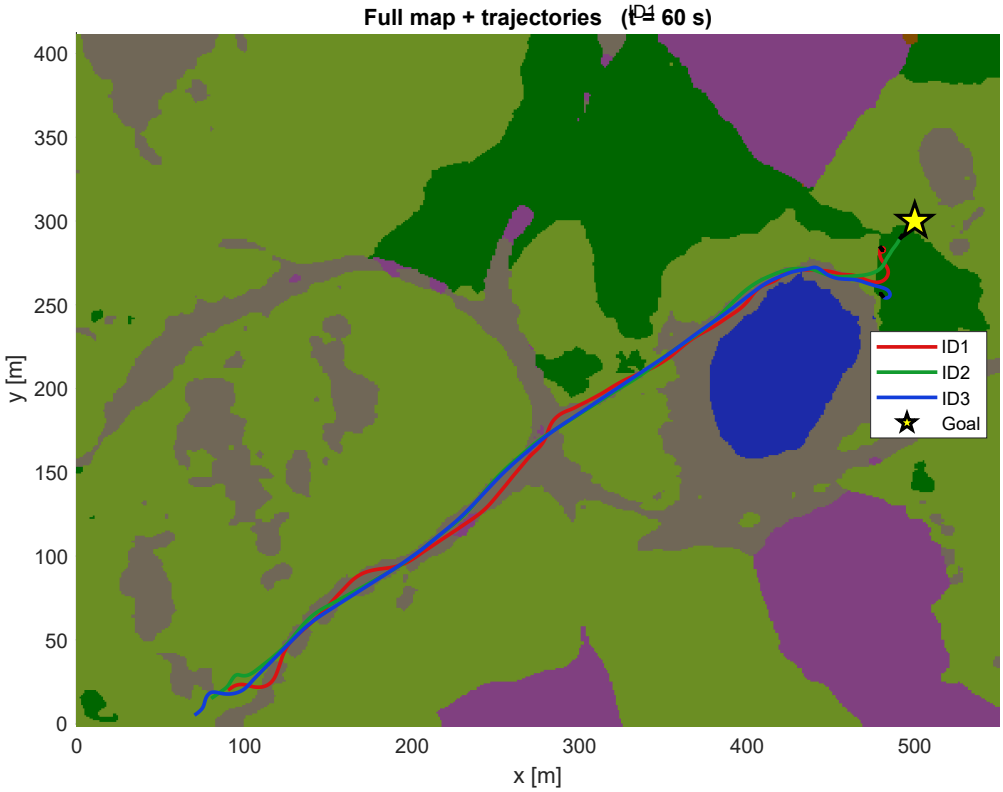
To show that the framework is not tuned to a single scenario, the very same swarm — with no change to the model, the controllers or their parameters — was run on a different semantic map, the one of Figure 4.2, with a different goal location. The frames of Figure 7.9, sampled at $t = 0$, 30 and $60\,\mathrm{s}$, show the three vehicles progressing across the new terrain and reaching the new target: the simulator accepts an arbitrary input map and goal, and the model was not calibrated for the first scenario.



(a) $t = 0\,\mathrm{s}$



(b) $t = 30\,\mathrm{s}$



(a) $t = 60\,\mathrm{s}$

Figure 7.9: Replay on a different map — semantic map with the trajectories of the three vehicles.

Chapter 8

Critical Discussion, Conclusion and Future Work

8.1 Interpretation of the Results

The simulation results should be interpreted by considering the interaction between planning, map discovery, control, and vehicle dynamics. A successful mission requires more than reaching the goal: the trajectory must be dynamically reasonable, the known map must evolve consistently, the followers must remain coordinated, and the speed profile must respect terrain and safety constraints.

The use of a dynamic vehicle model exposes information that a purely kinematic model could not reveal, in particular the speed limitations due to tire friction, the roll-related stability concerns, the actuator saturation, the transient behavior during steering, and the mismatch between the planned path and the dynamically feasible motion.

8.2 Engineering Trade-Offs, Contributions and Limitations

The project involves several trade-offs:

Table 8.1: Main engineering trade-offs.

Choice	Advantage	Limitation
A* planner	Simple and interpretable	Not incremental
Shared global map	Easy cooperation	Idealized communication
8-DOF vehicle model	Acceptable dynamic realism	Not complete for uneven real test
Model Reference architecture	Modular and reusable	Requires careful parameter management
Semantic terrain costs	Easy to tune	Approximate physical meaning

The main achievements on which the original contributions of this work rest are:

- a semantic off-road terrain map and its progressive update during the mission;
- terrain-aware path planning with A* and the generation of smoothed, trackable paths;
- a leader–follower formation strategy with dynamic, ranking-based role assignment;
- the integration of the planning and coordination layers with an 8-DOF vehicle dynamics model;
- a modular Simulink architecture based on Model Reference;
- a replay visualization and quantitative analysis framework.

The main limitations of the current work are the following.

- The perception of each vehicle is modeled as a circular visibility region, an idealized sensor footprint.
- Communication is idealized through a single shared workspace, with no range limit, latency, or packet loss.
- The generative AI module responsible for image recognition was trained on a database that was not specifically optimized for this task. Furthermore, it cannot estimate the terrain elevation, which is instead assumed and randomly generated.
- Path planning uses A* with periodic replanning. The search is direction-agnostic and carries no steering penalty, so the first segment of a path may leave the vehicle sideways: a planner conceived for kinematic systems is here applied to a dynamic vehicle.

- The treatment of the unknown map relies on heuristic cost penalties rather than on a probabilistic terrain model.
- The tire is described by Pacejka coefficients calibrated on a road tire, with a nominal radius and a tread that is not an off-road one. A representative off-road tire would behave very differently, and reliable coefficients for it are not available.
- The steering command is low-pass filtered only on the leader output: the leader oscillates while the followers' reference changes little. Filtering the path itself, locally, would be a cleaner solution.
- The EBD and ABS logic reads quantities that cannot be measured on a real vehicle, such as the vertical loads F_z and the wheel slip σ. Their behavior is therefore ideal, free of the latency and of the measurement and computation errors of a real system.
- Terrain deformation is represented by a simplified update (a uniform increase of μ) rather than by a full terramechanics model.
- The follower obstacle avoidance is a greedy, memoryless heading fan. This is the classic local-minimum weakness of reactive, potential-field-like methods, known since the critique of Koren and Borenstein; here it is mitigated by stacked safety layers (U-turn detection, collision check, motor cutoff) rather than removed.
- Vehicle failure management has not yet been implemented, partly because the signals are assumed to be perfect. Instead, if a vehicle gets stuck in the terrain or crushed by an obstacle, it will drop to the bottom of the rankings and be treated as the worst of the followers.

8.3 Future Work

Several developments can extend the present framework.

- Replace the periodic A* replanning with D* Lite, or another incremental planner, reusing previous planning information when new map data becomes available; the choice should be supported by the literature and by a discussion of its advantages in partially known and time dependent environments;
- Furthermore, as explained in Section 6.6.4, a search algorithm is required—whether A* or otherwise—that is inherently non-holonomic and accounts for the vehicle's physical constraints, such as its turning radius and inertia when changing direction; an example of its development can be found in [50];

- Assign each agent its own local map and let the agents exchange information through a communication model that includes range, latency, and packet loss, moving towards a genuinely decentralized swarm;
- Add vehicle-failure management: detect the failure and propagate a failure flag between the planner and the vehicles, so that a follower takes over the planning role using the map accumulated up to that point;
- Replace the simplified terrain update with a BWR terramechanics model (sinkage, soil compaction, rolling resistance, traction limits), with per-wheel sinkage and elevation differences instead of a uniform μ, and visualize the soil compaction left after each passage;
- Extend the simulation to a larger number of agents (>50) and more extensive maps, thereby allowing for the swarm to split into smaller clusters. The larger map and longer mission durations would enable the implementation of an exploration versus exploitation logic; this helps local cluster leaders determine whether to head for the goal or explore the map to provide greater certainty for clusters traversing the area later. Such logic, which is complex, would not have been effectively demonstrated in the current work, as it was developed using an overly small map;
- Introduce torque vectoring, electronic stability control, and a realistic ABS. The four independent in-wheel motors already allow an asymmetric left–right torque distribution, which generates a direct yaw moment at the wheels that actively assists the steering — a capability particularly valuable on loose off-road terrain, where the steer angle alone often saturates the available lateral grip. The traction control would then no longer merely cut the motor torque when a wheel slips (Section 5.9.2), but redistribute it across the wheels to help the vehicle turn.
- Extend the perception module beyond the present offline, single-frame proof of concept towards continuous, on-line semantic segmentation trained on a larger dataset, and add probabilistic obstacle and class priors for the unexplored cells, so that the cost function can anticipate, for example, the likely extent of an obstacle from its class.
- Extend the dynamics to a 14-DOF model and add a per-wheel slope and elevation disturbance, for instance on the gravel class to emulate stones, rather than a single global slope.
- Make the controller deployable on real hardware: the `coder.extrinsic` and `evalin` constructs used here are valid only for desktop simulation and are not compatible with code generation, so they must be converted before the controller can run on a real ECU.

- Exploit the known motor efficiency map to plan energy-optimal routes, in addition to terrain-optimal ones.
- Inflate the obstacles with a safety margin, or segment them conservatively already at the perception stage, so that the follower planner — which currently sees only a binary obstacle occupancy — preserves clearance.

Bibliography

[1] Amit Ailon and Shai Arogeti. Trajectory Tracking Control for a Kinematic Bicycle Model. In *2020 28th Mediterranean Conference on Control and Automation (MED)*, pages 526–531, September 2020. doi: 10.1109/MED48518.2020.9183161.

[2] Karl Berntorp, Björn Olofsson, Kristoffer Lundahl, and Lars Nielsen. Models and methodology for optimal trajectory generation in safety-critical road–vehicle manoeuvres. *Vehicle System Dynamics*, 52(10):1304–1332, October 2014. ISSN 0042-3114. doi: 10.1080/00423114.2014.939094.

[3] Shuping Chen, Huiyan Chen, and Dan Negrut. Implementation of MPC-Based Path Tracking for Autonomous Vehicles Considering Three Vehicle Dynamics Models with Different Fidelities. *Automotive Innovation*, 3(4):386–399, December 2020. ISSN 2522-8765. doi: 10.1007/s42154-020-00118-w.

[4] Lorenzo Mancardi. Implementation of an off-road tyre model for real-time dynamic driving simulator. Master's thesis, Politecnico di Torino, Torino, Italy, 2020.

[5] Keita Kita, Tomoya Arai, Kei Tsuchiya, Seiji Kon, Masahiro Katayama, and Shingo Ozaki. Terramechanics analysis of climbing behavior of wheel against rigid obstacle on soft terrain. *Discover Mechanical Engineering*, 3(1):16, June 2024. ISSN 2731-6564. doi: 10.1007/s44245-024-00046-7.

[6] Sh. Taheri, C. Sandu, S. Taheri, E. Pinto, and D. Gorsich. A technical survey on Terramechanics models for tire–terrain interaction used in modeling and simulation of wheeled vehicles. *Journal of Terramechanics*, 57:1–22, February 2015. ISSN 0022-4898. doi: 10.1016/j.jterra.2014.08.003.

[7] M. Grujicic, H. Marvi, G. Arakere, and I. Haque. A finite element analysis of pneumatic-tire/sand interactions during off-road vehicle travel. *Multidiscipline Modeling in Materials and Structures*, 6(2):284–308, August 2010. ISSN 1573-6105. doi: 10.1108/15736101011068037.

[8] Kaiming Xia. Finite element modeling of tire/terrain interaction: Application to predicting soil compaction and tire mobility. *Journal of Terramechanics*, 48(2):113–123, April 2011. ISSN 0022-4898. doi: 10.1016/j.jterra.2010.05.001.

[9] Chun-Lai Zhao and Meng-Yan Zang. Application of the FEM/DEM and alternately moving road method to the simulation of tire-sand interactions. *Journal of Terramechanics*, 72:27–38, August 2017. ISSN 0022-4898. doi: 10.1016/j.jterra.2017.04.001.

[10] Juthanee Phromjan and Chakrit Suvanjumrat. Non-pneumatic tire with curved isolated spokes for agricultural machinery in agricultural fields: Empirical and numerical study. *Heliyon*, 9(8), August 2023. ISSN 2405-8440. doi: 10.1016/j.heliyon.2023.e18984.

[11] Tsion Fekadu Deressu, Amanuel Kumsa Bojer, Taye Girma Debelee, Worku Gachena Negera, Saralees Nadarajah, and Kena Wendimu Gebissa. Enhancing land use and land cover classification with deep learning-based satellite imagery segmentation. *International Journal of Applied Earth Observation and Geoinformation*, 144:104839, November 2025. ISSN 15698432. doi: 10.1016/j.jag.2025.104839.

[12] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment Anything, April 2023.

[13] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. SAM 2: Segment Anything in Images and Videos, October 2024.

[14] Adam Van Etten, Dave Lindenbaum, and Todd M. Bacastow. SpaceNet: A Remote Sensing Dataset and Challenge Series, July 2019.

[15] Ye Lyu, George Vosselman, Gui-Song Xia, Alper Yilmaz, and Michael Ying Yang. UAVid: A semantic segmentation dataset for UAV imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 165:108–119, 2020. ISSN 0924-2716. doi: 10.1016/j.isprsjprs.2020.05.009.

[16] Friedrich Fraundorfer et al. Semantic drone dataset, 2018.

[17] Paulo V. K. Borges, Thierry Peynot, Sisi Liang, Bilal Arain, Matthew Wildie, Melih G. Minareci, Serge Lichman, Garima Samvedi, Inkyu Sa, Nicolas Hudson, Michael Milford, Peyman Moghadam, and Peter Corke. A Survey on Terrain Traversability Analysis for Autonomous Ground Vehicles: Methods, Sensors, and Challenges. *Field Robotics*, 2:1567–1627, July 2022. ISSN 2771-3989. doi: 10.55417/fr.2022049.

[18] Cuebong Wong, Erfu Yang, Xiu-Tian Yan, and Dongbing Gu. Adaptive and intelligent navigation of autonomous planetary rovers — A survey. In *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 237–244, Pasadena, CA, USA, July 2017. IEEE. ISBN 978-1-5386-3439-4. doi: 10.1109/AHS.2017.8046384.

[19] Brandon Rothrock, Ryan Kennedy, Chris Cunningham, Jeremie Papon, Matthew Heverley, and Masahiro Ono. SPOC: Deep Learning-based Terrain Classification for Mars Rover Missions. https://arc.aiaa.org/doi/10.2514/6.2016-5539, 2016.

[20] Deegan Atha, R. Michael Swan, Annie Didier, Zaki Hasnain, and Masahiro Ono. Multi-mission Terrain Classifier for Safe Rover Navigation and Automated Science. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–13, March 2022. doi: 10.1109/AERO53065.2022.9843615.

[21] Robert Swan, Masahiro Ono, and Deegan Atha. AI4Mars: A dataset for terrain-aware autonomous driving on mars, October 2022.

[22] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959. ISSN 0945-3245. doi: 10.1007/BF01386390.

[23] Gintaras Vaira and Olga Kurasova. Parallel Bidirectional Dijkstra's Shortest Path Algorithm. In *Databases and Information Systems VI*, pages 422–435. IOS Press, 2011. doi: 10.3233/978-1-60750-688-1-422.

[24] Hongzhuan Zhao, Jianyu Mao, Luchen Wang, Ruijue Tian, Tao Wang, Dan Zhou, Yicai Zhang, Changhe Nong, Qiuyu Liu, and Chuling Zhao. Research on improved bidirectional Dijkstra shortest path based on heap structures. September 2025. doi: 10.1117/12.3078757.

[25] Bernhard Haeupler, Richard Hladík, Václav Rozhoň, Robert E. Tarjan, and Jakub Tětek. Bidirectional Dijkstra's Algorithm is Instance-Optimal. In *2025 Symposium on Simplicity in Algorithms (SOSA)*, Proceedings, pages 202–215. Society for Industrial and Applied Mathematics, January 2025. doi: 10.1137/1.9781611978315.16.

[26] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968. ISSN 2168-2887. doi: 10.1109/TSSC.1968.300136.

[27] I. Pohl. *First Results on the Effect of Error in Heuristic Search*. MIP-R-. Edinburgh University, Department of Machine Intelligence and Perception, 1969.

[28] I. Pohl. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. 1973.

[29] Jiabo Huang, Chunmei Chen, Junjie Shen, Guihua Liu, and Feng Xu. A self-adaptive neighborhood search A-star algorithm for mobile robots global path planning. *Computers and Electrical Engineering*, 123:110018, April 2025. ISSN 0045-7906. doi: 10.1016/j.compeleceng.2024.110018.

[30] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3310–3317, San Diego, CA, 1994.

[31] Sven Koenig and Maxim Likhachev. Fast replanning for navigation in unknown terrain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3473–3479, Washington, DC, 2002.

[32] Chadhurbala R V, Naveen US, Nipun B V, Supriya. P, and Suyampulingam A. Comparative study of Bidirectional A*, D* and D* Lite for Path Planning. In *2024 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pages 1–6, July 2024. doi: 10.1109/CONECCT62155.2024.10677080.

[33] Xinda Wang, Xiao Luo, Baoling Han, Yuhan Chen, Guanhao Liang, and Kailin Zheng. Collision-Free Path Planning Method for Robots Based on an Improved Rapidly-Exploring Random Tree Algorithm. *Applied Sciences*, 10(4):1381, January 2020. ISSN 2076-3417. doi: 10.3390/app10041381.

[34] João Braun, Thadeu Brito, José Lima, Paulo Costa, Pedro Costa, and Alberto Nakano. A Comparison of A* and RRT* Algorithms with Dynamic and Real Time Constraint Scenarios for Mobile Robots. In *Proceedings of the 9th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pages 398–405, Prague, Czech Republic, 2019. SCITEPRESS - Science and Technology Publications. ISBN 978-989-758-381-0. doi: 10.5220/0008118803980405.

[35] Iram Noreen, Amna Khan, and Zulfiqar Habib. Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions. *International Journal of Advanced Computer Science and Applications*, 7(11), 2016. ISSN 21565570, 2158107X. doi: 10.14569/IJACSA.2016.071114.

[36] Fahad Islam, Jauwairia Nasir, Usman Malik, Yasar Ayaz, and Osman Hasan. RRT$*$-Smart: Rapid convergence implementation of RRT$*$ towards optimal solution. In *2012 IEEE International Conference on Mechatronics and Automation*, pages 1651–1656, August 2012. doi: 10.1109/ICMA.2012.6284384.

[37] Wei Wang, Hongli Gao, Qize Yi, Kaiyuan Zheng, and Tengda Gu. An Improved RRT* Path Planning Algorithm for Service Robot. In *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 1824–1828, Chongqing, China, June 2020. IEEE. ISBN 9781728143903. doi: 10.1109/ITNEC48623.2020.9085226.

[38] Tian Guan, Yi Han, Michuang Kong, Siyu Wang, Deyuan Feng, and Wei Yang. An improved artificial potential field with RRT star algorithm for autonomous vehicle path planning. *Scientific Reports*, 15(1):16982, May 2025. ISSN 2045-2322. doi: 10.1038/s41598-025-00694-z.

[39] Sertac Karaman and Emilio Frazzoli. Incremental Sampling-based Algorithms for Optimal Motion Planning, May 2010.

[40] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, February 1996. ISSN 1941-0492. doi: 10.1109/3477.484436.

[41] Marco Dorigo, Gianni Di Caro, and Luca M. Gambardella. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(2):137–172, April 1999. ISSN 1064-5462. doi: 10.1162/106454699568728.

[42] Hayati Mamur, Mehmet Ali Üstüner, and Mohammad Ruhul Amin Bhuiyan. Future perspective and current situation of maximum power point tracking methods in thermoelectric generators. *Sustainable Energy Technologies and Assessments*, 50:101824, March 2022. ISSN 22131388. doi: 10.1016/j.seta.2021.101824.

[43] Marco Dorigo and Christian Blum. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2):243–278, November 2005. ISSN 0304-3975. doi: 10.1016/j.tcs.2005.05.020.

[44] Peng Li, Lei Wei, and Dongsu Wu. An Intelligently Enhanced Ant Colony Optimization Algorithm for Global Path Planning of Mobile Robots in Engineering Applications. *Sensors*, 25(5):1326, January 2025. ISSN 1424-8220. doi: 10.3390/s25051326.

[45] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, November 1995. doi: 10.1109/ICNN.1995.488968.

[46] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, June 2007. ISSN 1935-3820. doi: 10.1007/s11721-007-0002-0.

[47] M. Shahab Alam, M. Usman Rafique, and M. Umer Khan. Mobile Robot Path Planning in Static Environments using Particle Swarm Optimization, August 2020.

[48] Kwang-Kyo Oh, Myoung-Chul Park, and Hyo-Sung Ahn. A survey of multi-agent formation control. *Automatica*, 53:424–440, March 2015. ISSN 0005-1098. doi: 10.1016/j.automatica.2014.10.022.

[49] Patrik Schmuck, Thomas Ziegler, Marco Karrer, Jonathan Perraudin, and Margarita Chli. COVINS: Visual-Inertial SLAM for Centralized Collaboration, August 2021.

[50] Tobias Recker, Malte Heinrich, and Annika Raatz. A Comparison of Different Approaches for Formation Control of Nonholonomic Mobile Robots regarding Object Transport. *Procedia CIRP*, 96:248–253, January 2021. ISSN 2212-8271. doi: 10.1016/j.procir.2021.01.082.

[51] T. Balch and R.C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, December 1998. ISSN 2374-958X. doi: 10.1109/70.736776.

[52] Seyed Mohammad Mahdi Seyed Sajadi and Hajar Atrianfar. Dynamic Circular Formation Of Multi-Agent Systems With Obstacle Avoidance And Size Scaling: A Flocking Approach, January 2023.

[53] Zihao Deng, Peng Gao, Williard Joshua Jose, Christopher Reardon, Maggie Wigness, John Rogers, and Hao Zhang. Coordinated Multi-Robot Navigation with Formation Adaptation, March 2025.

[54] Xinmiao Sun and Christos G. Cassandras. Optimal Dynamic Formation Control of Multi-Agent Systems in Environments with Obstacles, August 2015.

[55] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers, October 2021.