STU
SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Journal of Electrical Engineering, Vol. 77, No. 4, 2026, pp. 472-482
https://reference-global.com/journal/JEE

# Dynamics-aware local trajectory control of a tracked vineyard robot via deep reinforcement learning

**Filip Zúbek, Oliver Halaš, Vendelín František Skokan, Ladislav Körösi, Ondrej Straka, Aleš Melichár, Martin Dekan**

Autonomous navigation between narrow vineyard rows requires respecting the platform's dynamics, which trajectory optimizers such as CHOMP and STOMP resolve in batch before motion and again whenever the map changes. We instead cast local navigation as a continuous-control Markov decision process whose actions are the per-track torques of a skid-steer robot, so the platform's dynamics enter the control law itself, and the optimization is paid once, during training. Soft Actor-Critic (SAC) and Twin Delayed Deep Deterministic policy gradient (TD3), both recurrent, and a feed-forward Proximal Policy Optimization (PPO) baseline are trained over ten seeds in simulation from a real vineyard passability map. On two held-out scenarios all three produce shorter routes than a conservative weighted A* reference at higher peak but comparable average impassability, and the safety ranking of the off-policy agents reverses between scenarios.

Keywords: mobile robotics, deep reinforcement learning, motion planning, torque control, LSTM, terrain traversability, agricultural robotics, skid-steer control

## 1 Introduction

Autonomous navigation for mobile robots is a foundational problem in robotics and a prerequisite for safe operation in unstructured outdoor environments. It is classically decomposed into four coupled sub-problems: mapping, localization, path planning, and trajectory optimization together with the motion control that it executes [1]. In a simulated setting, the first two condense into a single passability map, a grid in which each cell encodes how difficult the corresponding patch of terrain is to traverse. Building such traversability fields from raw perception is itself a broad research area [2, 3]. The open question we address is not how to build the map, but how to act on it in closed loop on a platform with non-trivial dynamics.

We target the last of these sub-problems in a demanding instance: local trajectory optimization for a four-track robot, a tracked vineyard robot with independent torque control of each track. The objective is not merely to compute a route but to drive the robot along one it continuously adapts to local terrain. Vineyards are adverse for conventional planners: the rows are narrow, the inter-row terrain is irregular and time-varying, and the traction of a tracked platform changes with soil moisture, so that a geometrically valid path may be dynamically infeasible.

Classical motion planners are the established tools for this task but their limitation here is computational rather than representational. A* algorithm searches for the configuration graph for a least-cost path under a geometric cost and is in that sense genuinely blind to the robot's dynamics [4]. CHOMP and STOMP are not: CHOMP refines an initial trajectory by functional-gradient descent on a cost trading smoothness against terrain, and can respect hard constraints along the trajectory [5], while STOMP optimizes a comparable objective by cost-weighted stochastic sampling and, requiring no gradients, admits arbitrary non-differentiable terms, explicitly including constraint and motor-torque costs [6]. A dynamics-aware cost can therefore be written for either. The obstacle is what such a cost takes to optimize: both iterate over an entire trajectory before the robot moves, against a map held static for the duration of the optimization, and a map that has changed obliges them to solve again. Dynamics-awareness is thus bought with precisely the resource a robot already under way does not have, and the pipeline still ends by handing a trajectory to a separate tracking controller, which at execution time has no mechanism of its own to slow the robot when traction degrades or to redistribute torque when a track begins to slip.

Institute of Robotics and Cybernetics, Faculty of Electrical Engineering and Information Technology,
Slovak University of Technology in Bratislava, Slovakia
filip.zubek@stuba.sk

https://doi.org/10.2478/jee-2026-0045, Print (till 2015) ISSN 1335-3632, On-line ISSN 1339-309X

Learning-based methods fold these factors directly into the decision process. Deep reinforcement learning (RL) can learn closed-loop behaviors such as reactive avoidance and speed regulation directly from interaction, where they are hard to hand-engineer [7-9]. For the continuous action spaces relevant to torque control, the dominant algorithms are the off-policy actor-critics DDPG [10], TD3 [11], and SAC [12] and the on-policy PPO [13], typically with generalized advantage estimation [14]. Recurrent layers such as LSTM [15] extend these policies to partially observable settings by maintaining an internal state, and have been used in learning-based navigation such as multi-agent path finding [16]. Within agricultural robotics, learned local planners have been demonstrated for vineyard row following [17].

We learn a controller that observes a local passability map and the robot's kinematic state and outputs the four track torques directly, closing the perception-to-actuation loop. The central contribution is this dynamics-aware formulation: casting local navigation as a continuous-control MDP whose actions are per-track torques makes the platform's kinematics and dynamics part of the control law itself, rather than a constraint to be written into a cost and solved for afresh at every query. The optimization is paid once, offline, during training; at run time the policy evaluates a fixed-size local window at each control step, so its cost per step is constant and independent of route length. It can therefore run on a segment of a stored map or, in principle, online from locally sensed occupancy, where a trajectory optimizer such as CHOMP or STOMP would have to re-solve. The SAC and TD3 actors and critics embed LSTM layers for partial observability, and the reward includes a terrain-adaptive safe-speed term derived online from the local map. We compare SAC, TD3, and PPO against a conservative weighted A* reference and a straight-line geometric bound, training each agent over ten seeds and reporting full distributions rather than single runs [18, 19]. Basic A*, CHOMP, and STOMP are implemented and discussed as classical points of reference (Section 3.4) but are not evaluated quantitatively here, as it is out of the scope of this work. We find that the learned controllers produce shorter routes than the weighted A* reference at a quantified safety-efficiency cost, and that the safety ranking of the two off-policy agents reverses between our two test scenarios, which bears directly on how such controllers should be evaluated. The safe-speed term cannot induce the anticipatory slowing it was designed to produce, for reasons we trace to the formulation rather than to the learning. The off-policy agents carry recurrent memory while PPO remains feed-forward, a design we relate to prior work on recurrent policies rather than to a controlled ablation.

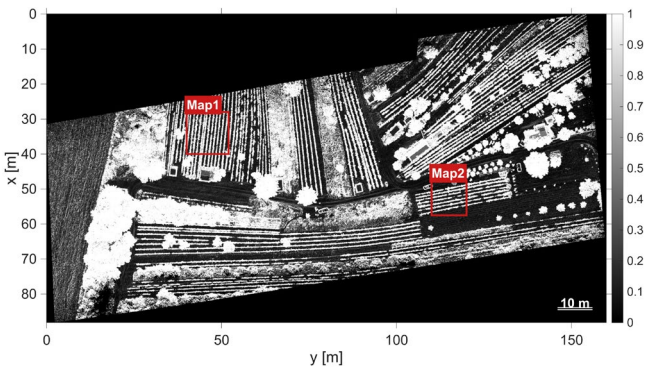


**Fig. 1.** Vineyard passability map from aerial imagery at 0.1 m per pixel. Bright regions are impassable, dark regions traversable. The marked rectangles are the segments cut for the test scenarios.

## 2 Problem formulation

The task is to drive the robot from a start pose to a goal location through passable terrain, while (i) keeping clear of impassable cells, (ii) respecting the platform's kinematic and dynamic limits, and (iii) adapting the motion online to the local terrain. Crucially, we do not seek a geometric path to be tracked later; we seek a closed-loop control policy that emits torques. Acting directly on the actuators is what admits terrain-dependent behavior at execution time, such as modulating speed or redistributing torque between tracks, without re-solving an optimization problem.

## 3 Method

We formulate torque-level navigation as a continuous-control MDP. This section introduces the robot and simulation model, the MDP itself, the three learned architectures, and the classical planners used as reference. Some of the results are rather shown in project's Github repository [21].

### *3.1 Robot model and simulation*

Training and evaluation are performed in MATLAB simulation [20] that models a simplified four-track robot whose parameters reflect the target hardware. The continuous state is $x = [x, y, \theta, v, \omega, z, \dot{z}]^{\mathsf{T}}$, where $(x, y, \theta)$ is the planar pose, $v$ and $\omega$ the forward and angular velocities, and $z$, $\dot{z}$ the chassis height and its rate.

The state is integrated with a fixed step $dt = 0.05$ s. When the angular rate is negligible the pose integrates linearly. Otherwise it follows a circular arc about the instantaneous center of rotation, which keeps the numerical trajectory error small. The control inputs are the four motor torques $M = [M_{FL}, M_{FR}, M_{RL}, M_{RR}]^{\mathsf{T}}$.

Each torque is converted to a tangential track force through the drive radius $r_s = 0.06$ m and gearbox efficiency $\eta_{drive} = 0.9$,

$$F_{tang} = \frac{M}{r_s}\eta_{drive} \tag{1}$$

and is saturated by the available friction, with friction coefficient $\mu = 0.7$ and gravitational acceleration

$$F_{max} = \mu\frac{mg}{4} \tag{2}$$

The resulting longitudinal and yaw accelerations follow from the left/right traction forces $F_L, F_R$, the track half-offset $y_{offset} = 0.25$ m, the yaw inertia $J_z = 3.0\ \mathrm{kg\,m^2}$, and linear damping terms,

$$a_{long} = \frac{(F_L + F_R) - b_v v}{m}$$
$$a_{yaw} = \frac{(F_R - F_L)y_{offset} - b_\omega \omega}{J_z} \tag{3}$$

A vertical sub-model (suspension stiffness, damping, and an active height command) reproduces dynamic shifts of the center of gravity, so that stability effects appear in the simulated response. Saturating the track force at $F_{max}$ is what makes the dynamics non-trivial: commanded torque and available traction are coupled, so beyond the saturation point additional torque produces no additional acceleration and the policy must distribute effort across the four tracks rather than simply demand more of one. Note that $\mu$ is held constant in this model, so the friction cap itself does not vary with terrain; terrain enters the problem through the passability map, which drives the collision test, the reward, and the terrain-dependent safe speed of Section 3.2.

### *3.2 Markov decision process*

We cast the navigation task of Section 2 as a Markov decision process (MDP) [7], specified below by its state, action, and reward. The remaining settings are shown in the GitHub repository [21].

**State and observation.** At time $t$ the state is the pair $s_t = (M_t, x_t)$. The local map $M_t$ is a 20×20 window of the passability map, centered on the robot and expressed in the global frame,

$$M_t \in [0,1]^{20\times 20} \tag{4}$$

The kinematic/navigation sub-vector collects orientation, heading error, normalized goal distance, and normalized velocities,

$$x_t = \left[\sin\theta, \cos\theta, \sin\theta_{\mathrm{err}}, \cos\theta_{\mathrm{err}}, \bar{d}_{goal}, \bar{v}, \bar{\omega}\right]_t^{\top} \tag{5}$$

where $\theta_{err}$ is the heading error to the goal, $\bar{d}_{\mathrm{goal}}$ the goal distance normalized by the initial trajectory length, and $\bar{v}_t$, $\bar{\omega}_t$ the velocities normalized by $5\ \mathrm{m\,s^{-1}}$ and $5\ \mathrm{rad\,s^{-1}}$. The map is flattened and concatenated with $x_t$ to form the network input,

$$o_t = \begin{bmatrix} x_t \\ vec(M_t) \end{bmatrix} \in \mathbb{R}^{407} \tag{6}$$

The map is thus consumed as a flat vector: no convolution is applied to it. At 20×20, the window is small enough that a fully connected layer over it carries a modest parameter count, and keeping the observation a single vector lets the same architecture serve all three algorithms without a separate spatial encoder.

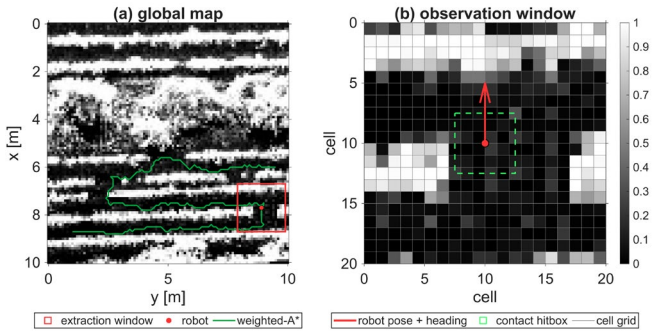


**Fig. 2.** a) Extraction of the 20×20 local passability window from a map segment. (b) The window itself, 2.0 m square, with the dashed 5×5 contact hitbox that defines $T_{diff}$

A convolutional front end would be the natural extension for larger windows and is not evaluated here. An example local window is shown in Fig. 2.

**Action.** The action is the four-dimensional vector of track torques, bounded by the motor limits,

$$a_t = [\tau_{FL}, \tau_{FR}, \tau_{RL}, \tau_{RR}]^{\top} \in A$$
$$A = [-10,10]^4 \subset \mathbb{R}^4(Nm) \tag{7}$$

The continuous, four-dimensional action enables differential steering and active force distribution over uneven terrain; the forward speed is additionally capped at $5\ \mathrm{m\,s^{-1}}$. All three algorithms compared here operate natively in continuous action spaces, so no discretization is required.

**Reward.** The reward at each step is the sum of a terminal term, a progress term, a heading term, and a speed penalty,

$$R = r_{fin} + r_{progress} + r_{heading} - r_{speed} \tag{8}$$

The terminal term rewards reaching the goal and penalizes failure, ending the episode when non-zero

$$r_{fin} = \begin{cases} +100 & \text{if } d_{\text{goal}} < 0.3\,\text{m}, \\ -500 & \text{collision, step limit or entrapment}, \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

The progress term rewards closing the distance to the goal

$$r_{progress} = 100\,(d_{prev} - d_{goal}), \quad v < 3\,\text{m s}^{-1} \tag{10}$$

where $d_{prev}$ and $d_{goal}$ are the goal distances at the previous and current steps. It is gated below $3\,\text{m s}^{-1}$ (above the terrain-adaptive safe speed of at most $2\,\text{m s}^{-1}$ but below the $5\,\text{m s}^{-1}$ cap) so that the agent cannot earn progress reward by moving faster than is safe. This term has the form of potential-based reward shaping with potential $\Phi = -d_{goal}$, a class of shaping that provably leaves the optimal policy unchanged [22]. The heading and speed terms below are bounded, task-specific shaping tuned to suppress degenerate behaviors rather than to redefine the objective. The heading term rewards facing the goal and explicitly penalizes facing away from it, eliminating a learned tendency to approach the goal in reverse,

$$r_{heading} = 1 - 6\,e_h^2 \tag{11}$$

where $e_h$ is the normalized heading error. Finally, the speed penalty regulates velocity against a terrain-dependent safe speed,

$$r_{speed} = 5\big(|v| - v_{safe}\big)^2, \quad |v| > v_{safe} \tag{12}$$

with the safe speed derived online from the difficulty of the terrain in contact with the robot,

$$v_{safe} = 2\big(1 - T_{diff}\big), \quad T_{diff} = mean(hitbox) \tag{13}$$

where $T_{diff}$ is the mean impassability over the cells in direct contact with the platform. The progress weight of 100 is deliberate: with $d_t = 0.05$ s the largest single-step displacement is 0.25 m at the $5\,\text{m s}^{-1}$ cap, bounding the per-step progress reward by 25, while the $3\,\text{m s}^{-1}$ gate binds first and holds the credited value below 15. Either way the term stays well scaled relative to the terminal reward. The collision penalty is five times the goal reward, prioritizing safety over speed. The component weights and thresholds were set by a combination of domain reasoning and empirical tuning, following established guidance for stable RL training [23]. Section 5.2 examines whether the safe-speed term achieves its intended effect.

### *3.3 Network architectures*

The three agents share the observation and action spaces [21] and differ mainly in whether they carry recurrent memory and in their policy class. Full layer specifications follow the original algorithms [11-13]. We state only the choices relevant to this study.

**SAC** uses a stochastic recurrent actor: a 256-unit fully connected feature extractor followed by a 128-unit LSTM that maintains trajectory context, with separate mean and standard-deviation heads. Its twin critics are likewise recurrent, and the entropy temperature is tuned automatically toward a target entropy of $-|A|$ rather than held fixed [24]. Summed over the actor and both critics, the agent has 0.96×106 parameters.

**TD3** shares the recurrent actor-critic backbone but with a deterministic actor. At 1.11×106 parameters it is the largest agent, its twin critics each carrying two LSTM layers per input branch. One further detail is relevant to its behavior in Section 5: whereas the SAC and PPO actors scale their tanh output by 10, the TD3 actor scales by 20 before the environment saturates the command at ±10 Nm. Its effective action range is therefore identical to the others, but the torque limit is reached whenever the tanh output exceeds 0.5 in magnitude, giving TD3 a markedly more saturated, bang-bang actuation than the other two agents.

**PPO** is feed-forward throughout, with an MLP actor and a separate value critic and no recurrence or twin critic; at 0.34×106 parameters it is by far the smallest agent. We attempted a recurrent PPO but did not obtain stable training. This is consistent with the documented difficulty of recurrent on-policy optimization: because PPO reuses each trajectory batch over several epochs, the stored LSTM hidden states grow stale relative to the updated policy, a phenomenon analyzed by Pleines et al. [25]. Mitigations such as refreshing the hidden states before each epoch exist but lie outside the scope of this work, and the issue does not arise for a feed-forward PPO, which carries no hidden state. We therefore adopted the feed-forward design, the principal architectural difference from the off-policy agents.

### *3.4 Conventional baselines*

For reference we implement four classical planners. Of these, only weighted A* enters the quantitative comparison of Section 5, for the reasons given at the end of this section. The other three are described here to situate the learned controller against the standard alter-natives. Basic A* performs an 8-connected grid search with a Manhattan heuristic but ignores passability, so it produces a near-minimal geometric route while crossing impassable cells; this makes it useless as a safety

reference and we do not report it. Note that a Manhattan heuristic is inadmissible on an 8-connected grid whose diagonal steps cost $\sqrt{2}$, so neither A* variant used here is optimal in the strict sense; both are used as conservative references rather than as optimality bounds. Weighted A* augments the step cost with a terrain penalty

$$g(n) = g(cur) + c_{base} + c_{terr}$$
$$c_{t,err} = \alpha\,|H(n) - H(cur)| \quad (14)$$

with $\alpha = 40$, $c_{\text{base}} = \sqrt{dr^2 + dc^2}$ the Euclidean step cost, and $H$ the passability field of Section 2, so that $c_{terr}$ penalizes spatial gradients of passability. The result is a conservative, safety-prioritizing route that we use as the principal reference. It is drawn on both test scenarios (see project's Github repository [21]). CHOMP refines an initial straight-line trajectory by gradient descent, $\xi_{k+1} = \xi_k - \eta\nabla U(\xi_k)$, balancing a smoothness term against the same terrain cost. STOMP optimizes a comparable objective with cost-weighted stochastic sampling. Both are global trajectory optimizers, and their expressive power is not the limitation: CHOMP admits hard constraints along the trajectory [5] and STOMP, needing no gradients, admits non-differentiable terms such as constraint and motor-torque costs [6], so either could in principle be given a dynamics-aware cost. The limitation is what one query costs. In the configuration used here, CHOMP evaluates its gradient at 500 trajectory points over 1000 iterations, and STOMP draws 20 noisy rollouts of 300 points over 200 iterations, on the order of 106 cost evaluations before the robot moves; a richer cost multiplies that figure. Both also assume the map is fixed for the duration of the optimization, can fail to converge on the more contorted routes, and on the vineyard map tend to cut repeatedly across the vine rows. The learned controller instead pays its optimization once during training and evaluates a single fixed-size window per control step, at a cost that does not grow with route length. We do not report CHOMP and STOMP results in Section 5. Both consume an entire route in one batch and emit a geometric trajectory with no associated speed profile or actuation, so the three metrics we use are not defined for them on comparable terms. Establishing a fair comparison would require pairing each with a tracking controller on the same platform model and reporting the resulting executed trajectory, which is out of scope here, and we leave it to future work. Weighted A* is retained as the principal reference precisely because it is the one classical planner that produces a passability-aware route directly comparable on all three metrics. It should be read as a reference from the layer above: in the decomposition of Section 1 it addresses path planning, whereas the learned controller addresses trajectory optimization and motion control. The two are complementary rather than substitutes, which is what makes the hybrid arrangement of Section 5.5 natural.

## 4 Experimental setup

**Training environment.** Smaller segments are extracted from the global passability map to form five training scenarios; training on the full map would make a single episode prohibitively slow. At every episode reset the environment selects a scenario at random and samples a start and goal from its predefined route set, and the robot's initial orientation is randomized over the full 360°. This randomization forces the policy to react to the current observation rather than memorize a fixed maneuver. A target is counted as reached when the robot comes within an acceptance radius of 0.3 m of it. The test is evaluated at every control step and applies to each waypoint in the sequence, not only to the final goal.

**Hyperparameters and training.** The details of training and hyperparameters are listed in the project's GitHub repository [21]. The off-policy agents train with a replay buffer of $10^6$ transitions, a mini-batch of 2048, and an LSTM sequence length of 10, for 4000 episodes (SAC) and 7000 episodes (TD3); the first 5000 environment steps use random actions to seed the buffer. The on-policy agent (PPO) trains for 20,000 episodes with an experience horizon of 3000 and 10 epochs per update. All agents use a learning rate of $3 \times 10^{-4}$. The much larger episode and horizon budget of PPO reflects a well-documented difference in sample efficiency: off-policy SAC and TD3 reuse transitions from the replay buffer and therefore need far fewer environment interactions, whereas on-policy PPO must collect fresh trajectories for every update [12, 13]. The mini-batch sizes and gradient-clipping thresholds likewise follow established settings for each algorithm family [22].

**Statistical protocol.** Because deep RL is highly sensitive to random seeding, point estimates from a single run can misrepresent an algorithm's behavior [18]. Each algorithm is therefore trained ten times with identical hyperparameters but different seeds, and results are reported as full distributions (box plots) rather than means, exposing the inter-quartile spread and outlier runs that form the safety-critical tail for field robotics. Reporting per-seed distributions in this way meets the minimum standard recommended for reliable RL evaluation [18]. We do not additionally apply stratified estimators such as the interquartile mean or bootstrap confidence intervals [19], nor formal pairwise significance tests, which would require the full per-seed score matrix.

**Test scenarios and metrics.** Two held-out test scenarios assess generalization. Scenario 1 emphasizes long, near-straight runs with two sharp reversals and is dominated by horizontal motion along the rows. Scenario 2 is more tortuous, with frequent direction changes and one inter-waypoint gap deliberately larger than any seen in training, and is dominated by vertical motion across the map. As an absolute lower bound, we

compute the straight-line connection of the waypoints, with reference lengths of 23.33 m (Scenario 1) and 31.72 m (Scenario 2); this line is geometrically optimal but generally unrealizable. Performance is measured by three metrics: the total path length (m); the average impassability (%), the mean obstacle density in a 5×5 occupancy window centered on the robot, averaged over the run; and the maximum impassability (%), the worst such value along the run. Lower impassability indicates a safer route.

**Table 1.** Training hyperparameters for SAC, TD3, and PPO

| Scenario | Metric | SAC | TD3 | PPO | W-A* | Straight |
|---|---|---|---|---|---|---|
| 1 | Path length (m) | 24.62 [24.22–25.09] | **23.55 [23.35–23.98]** | 24.42 [23.94–24.68] | 26.91 | 23.33 |
| 1 | Avg. impass. (%) | **11.40 [10.20–12.70]** | 18.35 [17.00–19.30] | 12.50 [12.20–14.10] | 9.79 | – |
| 1 | Max. impass. (%) | **47.95 [44.40–53.27]** | 62.00 [58.50–62.50] | 49.23 [46.07–57.87] | 33.62 | – |
| 2 | Path length (m) | 36.80 [35.40–37.44] | **32.84 [32.31–33.50]** | 33.75 [33.32–34.70] | 38.52 | 31.72 |
| 2 | Avg. impass. (%) | 19.88 [19.16–21.30] | **13.80 [13.40–16.60]** | 16.50 [15.20–17.20] | 15.81 | – |
| 2 | Max. impass. (%) | 68.95 [59.96–72.90] | **57.36 [40.75–69.92]** | 57.77 [56.82–65.03] | 46.64 | – |

## 5 Results

Table 1 summarizes every method on both scenarios. Because deep RL is seed-sensitive, the learned-agent entries are medians over the ten independent training runs rather than a single rollout; Figure 5 shows the full distributions behind them. We first establish the classical reference, then analyze each learned controller, and finally compare them statistically.

### *5.1 Weighted A* reference*

The weighted A* planner is conservative by design: it attains an average impassability of 9.79% on Scenario 1 (path length 26.91 m) and 15.81% on Scenario 2 (38.52 m), with peak exposure of 33.62% and 46.64% respectively. Its routes are the longest of any method. It records the lowest exposure of any method on three of the four impassability measures and is the reference the learned controllers are measured against; the exception, Scenario 2 average impassability, is taken up in Section 5.5.

### *5.2 Learned controllers*

All three RL agents drive the robot by commanding torques, so their routes emerge from physical interaction with the terrain rather than from geometric optimization (Figs. 3a and 3b).

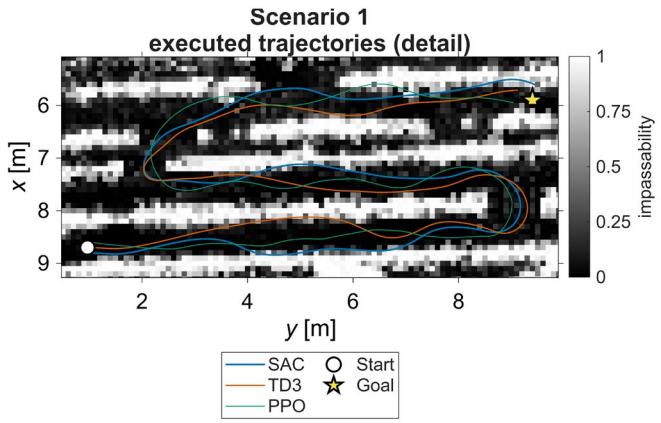


**Fig. 3a.** Executed trajectories of SAC (blue), TD3 (orange) and PPO (green). These are randomized-start single-segment rollouts, shown for route shape only, Scenario 1

**SAC** is the safest agent on Scenario 1, with the lowest median average impassability (11.40%) and the lowest median peak (47.95%), at a median route length of 24.62 m, 2.29 m shorter than weighted A*. That advantage does not carry over: on Scenario 2 it is the worst learned agent on all three metrics (Section 5.3).

**TD3** produces the shortest routes in both scenarios (23.55 m and 32.84 m median, both close to the straight-line reference). On Scenario 1 it pays for this in safety margin, with the highest median average impassability

(18.35%) and the highest peaks (62.00%), consistent with the more saturated actuation implied by its ×20 output scaling (Section 3.3). In Scenario 2, however, it is the best learned agent on every metric.

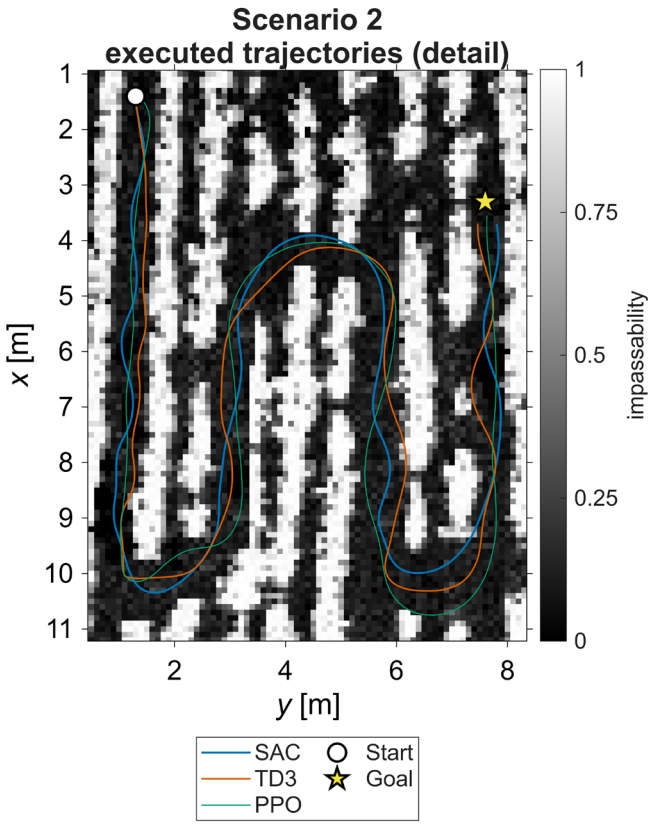


**Fig. 3b.** Executed trajectories of SAC (blue), TD3 (orange) and PPO (green). These are randomized-start single-segment rollouts, shown for route shape only, Scenario 2.

**PPO**, despite lacking recurrence, is competitive throughout: it is second of the three on every metric in both scenarios, never best and never worst. It also completes the task in substantially fewer control steps than either recurrent agent (291, against 345 for SAC and 346 for TD3, on Scenario 1; 281, against 431 and 390 respectively, on Scenario 2) at comparable route length, that is, by driving faster rather than by taking a shorter path.

**Terrain-adaptive speed.** The reward of Eqn. (8) penalizes forward speed above a terrain-dependent safe speed $v_{safe} = 2(1 - T_{diff})$, with the intention of inducing the robot to slow down before difficult ground. The term as formulated cannot express that behavior, and the measurements bear this out (PPO can be seen in Fig. 4. SAC and TD3 comparison can be seen in project's Github repository [21]). Across the six rollouts, forward speed and local impassability are positively associated at zero lag in five cases, and the deepest speed minima occur in open terrain, at the waypoint reversals, rather than on the difficult patches. Where a reduction in speed does follow a rise in impassability it appears some tens of control steps later, so any such response is reactive rather than anticipatory. The reason is structural: $T_{diff}$ is the mean over a 0.5 m×0.5 m hitbox centered on the robot (Fig. 2) and therefore carries no look-ahead at all; the only forward information is the 1.0 m radius of the observation window, roughly fourteen control steps at the observed speeds.

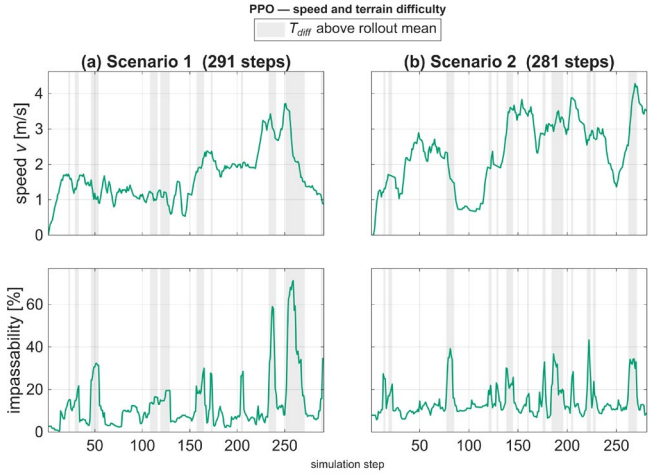


**Fig. 4.** Forward speed (upper) and impassability (lower) per agent. The intended coupling is absent: the two are positively associated at zero lag in five of six rollouts. PPO peaks near 4.3 m $s^{-1}$.

### *5.3 Statistical comparison*

Figure 5 reports the distribution of all three metrics over the ten seeds for both scenarios. Four patterns emerge. First, on path length all three learned agents fall below the weighted A* reference in both scenarios, and TD3 attains the shortest median with the tightest spread (23.55 m on Scenario 1, interquartile range 0.63 m against 0.87 m for SAC and 0.74 m for PPO). Part of the TD3 box lies below the 23.33 m straight-line reference, which is possible because the 0.3 m acceptance radius lets a rollout cut each waypoint corner rather than pass exactly through it; the reference is therefore a close approximation to a lower bound, not a strict one. Second, on maximum impassability every learned agent sits above the weighted A* reference (33.62% and 46.64%), confirming that peak exposure is a tail risk of the learned policies rather than an artifact of a single run.

Third, and most consequential, the safety ranking reverses between the two scenarios. On Scenario 1 SAC is the safest learned agent on both impassability metrics (median 11.40% average, 47.95% peak) and TD3 the least safe (18.35%, 62.00%). On Scenario 2 the order inverts completely: TD3 is best on all three metrics (32.84 m, 13.80%, 57.36%) while SAC is worst on all

three (36.80 m, 19.88%, 68.95%). Neither off-policy agent is reliably safer than the other; which one looks better depends on the scenario it is measured in. TD3's advantage on Scenario 2 peak exposure should nonetheless be read with care: its median is the lowest of the three, but its interquartile range spans 40.75% to 69.92%, close to thirty percentage points and far wider than either competitor's, so individual seeds range from clearly the safest to among the roughest. Fourth, PPO is the only agent that never occupies either extreme: it ranks second of the three on all six metric-scenario combinations. Its boxes are also among the tightest, the exception being its maximum-impassability outlier on Scenario 2.

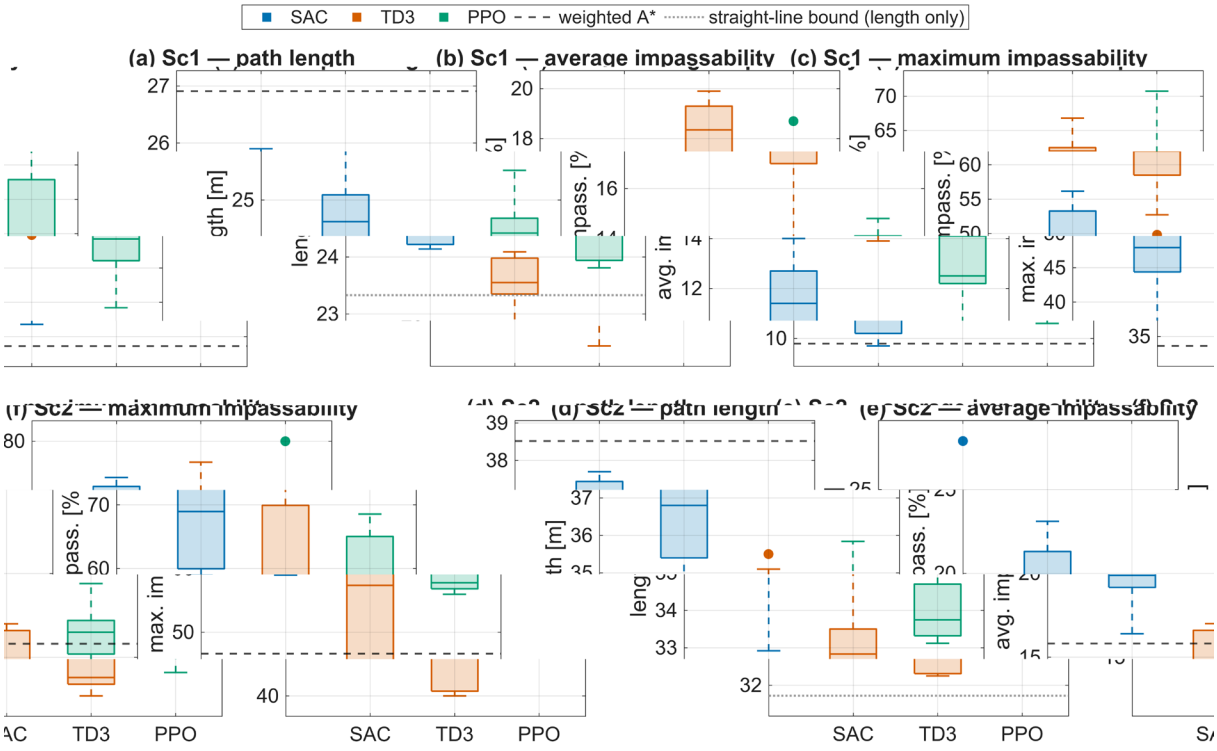


**Fig. 5.** Each metric's distribution over ten seeds. Top row Scenario 1, bottom Scenario 2: (a, d) path length, (b, e) average and (c, f) maximum impassability. Note the SAC/TD3 reversal between rows.

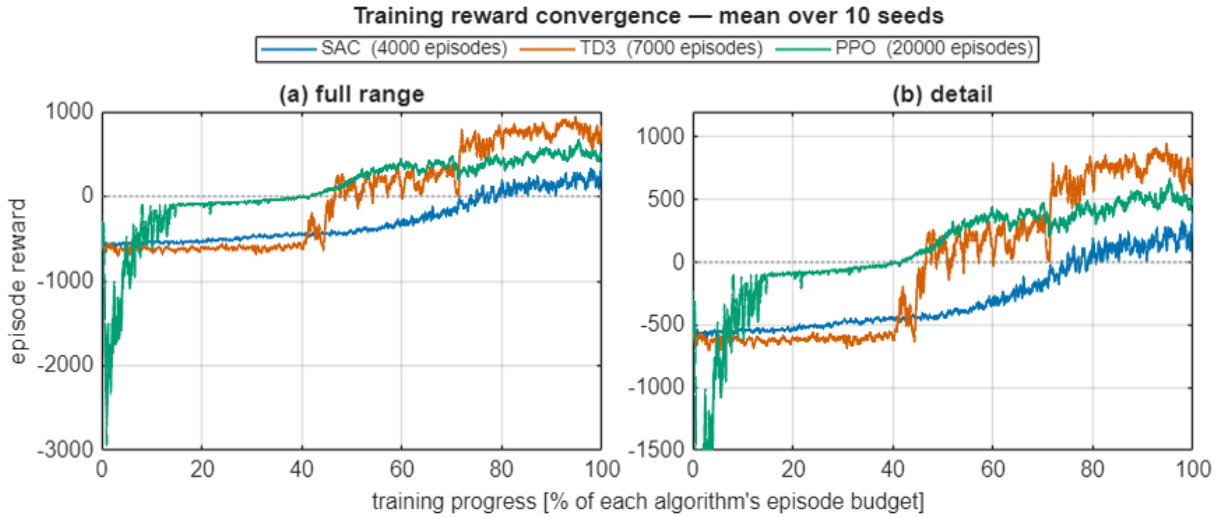


**Fig. 6.** Mean training reward over ten seeds against percent of each algorithm's episode budget: (a) full range, (b) detail. Budgets (4000, 7000, 20,000) were set from observed convergence.

### *5.4 Training convergence*

The mean reward curves, averaged over the ten seeds, reveal markedly different learning dynamics (Fig. 6). SAC converges smoothly but slowly, rising steadily from about −580 and crossing into positive reward at roughly episode 3000 of its 4000-episode budget. TD3 improves in two discrete steps rather than one: reward stays near −620 until about episode 2900, jumps to roughly +200, then makes a second jump near episode 5000 to settle between 750 and 850. PPO requires far more experience, running to 20,000 episodes; its curve is stable but slow and begins with a pronounced instability dipping to about −2900, characteristic of on-policy policy search, after which it crosses zero near episode 8300. Because final training reward orders the agents differently from the task metrics, we base all comparisons on the latter. PPO is the cheapest per update and carries under a third of TD3's parameter count.

### *5.5 Discussion*

These results must be read along two axes. On the safety metric the conservative weighted A* planner leads on peak exposure in both scenarios (33.62% and 46.64%) and on Scenario 1 average impassability (9.79%), though TD3 attains a lower average on Scenario 2 (13.80% against 15.81%). The planner's terrain penalty charges $\alpha|H(n) - H(\mathrm{cur})|$, the gradient of passability rather than its level, so it avoids terrain transitions rather than difficult terrain as such, and the route it returns is purely geometric, carrying no model of the platform's kinematics or dynamics. The contribution of the learned controllers is therefore not that they solve a safer problem but a different one. A classical planner emits a trajectory for a separate tracker to execute, resolving whatever dynamics its cost encodes before the robot moves and paying again whenever the map changes. The RL policies instead close the loop from perception to actuation at a fixed cost per control step, and produce shorter routes than the conservative reference. The trade-off is thus one of safety against efficiency: median peak impassability ranges from 47.95% to 68.95% across the three agents and two scenarios, so the learned trajectories do at times pass close to impassable terrain, which motivates a hybrid scheme in which a safety-first global planner supplies a coarse route that the learned controller locally adapts. The safe-speed result of Section 5.2 carries a design lesson.

A reward intended to shape speed against terrain should evaluate the terrain the robot is about to enter, over a horizon matched to its braking distance. We record this as the most actionable finding of the study for practitioners designing similar rewards. Choosing among the learned agents is less clear-cut than a single scenario suggests. TD3 has the strongest raw numbers, winning four of the six metric-scenario combinations, but neither off-policy agent holds its ranking across the two scenarios (Section 5.3). We therefore recommend PPO for deployment, on consistency rather than peak performance: it is the only agent never worst on any metric in either scenario, it is the cheapest by a wide margin (0.34M parameters against 0.96M and 1.11M), and being feed-forward it carries no hidden state to initialize or resynchronize on hardware. The consistency argument rests on two environments, so it establishes that PPO did not fail on any axis we measured rather than general superiority; the architectural and computational arguments do not depend on scenario count. PPO achieves its low step counts by driving fast rather than short, at speeds that would require an explicit cap for a 40 kg robot working between vine rows; it is consequently the agent whose ranking would change most if that cap were lowered. If shortest path is the sole objective and the deployment terrain resembles Scenario 2, TD3 is the better choice. We did not run a controlled recurrent-versus-feed-forward ablation, so the LSTM design follows prior evidence rather than a measured benefit here: recurrent memory aids off-policy agents under partial observability [26, 27], while recurrent on-policy PPO is difficult to stabilize [25]. These conclusions are bounded by the study's scope. It is conducted entirely in simulation with a hand-designed reward; the dynamics model only partially captures track slip and soil-moisture variability; and the observation is read directly from the simulator, whereas a physical deployment must reconstruct the passability window and kinematic vector from onboard sensing and localization. Together these define the sim-to-real gap that physical validation must close.

## 6 Conclusion

We presented a deep-RL controller that converts a local window of a precomputed vineyard passability map, together with the robot's kinematic state, directly into per-track wheel torques, making the platform's kinematics and dynamics part of the control law rather than a constraint encoded in a cost and solved for at every query. Unlike trajectory optimizers such as CHOMP and STOMP, which re-solve per query, it pays that cost once during training and thereafter evaluates one fixed-size local window per control step. Benchmarking SAC, TD3, and PPO over ten seeds on two test scenarios, all three learned controllers produced shorter routes than a conservative weighted A* reference, at consistently higher peak impassability but comparable average exposure, which defines the safety-efficiency trade-off between them. The safety ranking of the two

off-policy agents reverses between the two scenarios, so reporting full seed distributions is what makes the comparison interpretable at all. We recommend the feed-forward PPO agent, the only one never ranked worst on any metric and the smallest by a factor of three. We also report a reward-design finding: the term intended to induce terrain-adaptive slowing cannot do so, because the terrain measure it depends on carries no look-ahead. Future work will incorporate the energy cost of motion into the reward, extend the state with sensor data from the physical chassis, and validate the trained agents on real hardware, where track slip, uneven loading, and soil-moisture variability come into play. A promising direction is the hybrid architecture of Section 5.5, in which weighted A* supplies a coarse, low-risk route that the RL controller tracks and locally adapts in closed loop. Redesigning the safe-speed term to evaluate the terrain the robot is about to enter rather than the terrain beneath it is a direct and inexpensive test of whether terrain-adaptive speed control is learnable in this setting. A direct ablation of recurrent against feed-forward SAC and TD3, and an evaluation over more than two scenarios, also remain open.

## Acknowledgement

Funded by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09I05-03-V02-00039.

## References

[1] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.

[2] P. Borges, T. Peynot, S. Liang, B. Arain, M. Wildie, M. Minareci, S. Lichman, G. Samvedi, I. Sa, N. Hudson, M. Milford, P. Moghadam, and P. Corke, "A Survey on Terrain Traversability Analysis for Autonomous Ground Vehicles: Methods, Sensors, and Challenges", Field Robotics, vol. 2, no. 1, pp. 1567–1627, 2022, DOI: 10.55417/fr.2022049.

[3] C. Stachniss, Robotic Mapping and Exploration. Berlin, Heidelberg: Springer, 2009.

[4] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968, DOI: 10.1109/TSSC.1968.300136.

[5] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian Optimization for Motion Planning", The International Journal of Robotics Research, vol. 32, no. 9–10, pp. 1164–1193, 2013, DOI: 10.1177/0278364913488805.

[6] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic Trajectory Optimization for Motion Planning", in 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 4569–4574, 2011, DOI: 10.1109/ICRA.2011.5980280.

[7] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-Level Control through Deep Reinforcement Learning", Nature, vol. 518, no. 7540, pp. 529–533, 2015, DOI: 10.1038/nature14236.

[9] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement Learning in Robotics: A Survey", The International Journal of Robotics Research, vol. 32, no. 11, pp. 1238–1274, 2013, DOI: 10.1177/0278364913495721.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning", preprint arXiv:1509.02971, 2016, DOI: 10.48550/arXiv.1509.02971.

[11] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods", in Proceedings of the 35th International Conference on Machine Learning (ICML), vol. 80, pp. 1587–1596, 2018.

[12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor", in Proceedings of the 35th International Conference on Machine Learning (ICML), vol. 80, pp. 1861–1870, 2018.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms", arXiv preprint arXiv:1707.06347, 2017, DOI: 10.48550/arXiv.1707.06347.

[14] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation", arXiv preprint arXiv:1506.02438, 2016, DOI: 10.48550/arXiv.1506.02438.

[15] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997, DOI: 10.1162/neco.1997.9.8.1735.

[16] T. Wang, "CLE: An Integrated Framework of CNN, LSTM, and Enhanced A3C for Addressing Multi-Agent Pathfinding Challenges in Warehousing Systems", IEEE Access, vol. 12, pp. 88904–88912, 2024, DOI: 10.1109/ACCESS.2024.3416111.

[17] D. Aghi, V. Mazzia, and M. Chiaberge, "Local Motion Planner for Autonomous Navigation in Vineyards with a RGB-D Camera-Based Algorithm and Deep Learning Synergy", Machines, vol. 8, no. 2, pp. 27, 2020, DOI: 10.3390/machines8020027.

[18] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters", in Proceedings of the 32nd AAAI Conference on Artificial Intelligence, pp. 3207–3214, 2018, DOI: 10.1609/aaai.v32i1.11694.

[19] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, "Deep Reinforcement Learning at the Edge of the Statistical Precipice", in Advances in Neural Information Processing Systems (NeurIPS), vol. 34, pp. 29304–29320, 2021.

[20] The MathWorks, Inc., Reinforcement Learning Toolbox. Software, Natick, MA, USA, 2024. https://www.mathworks.com/products/reinforcement-learning.html.

[21] "Vineyard Impassability RL", GitHub repository. [Online]. Available: https://github.com/xHalaso/Vineyard_impassability_RL.

[22] A. Y. Ng, D. Harada, and S. Russell, "Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping", in Proceedings of the 16th International Conference on Machine Learning (ICML), pp. 278–287, 1999.

[23] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations", Journal of Machine Learning Research, vol. 22, no. 268, pp. 1–8, 2021.

[24] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft Actor-Critic Algorithms and Applications", preprint arXiv:1812.05905, 2018, DOI: 10.48550/arXiv.1812.05905.

[25] M. Pleines, M. Pallasch, F. Zimmer, and M. Preuss, "Generalization, Mayhems and Limits in Recurrent Proximal Policy Optimization", preprint arXiv:2205.11104, 2022, DOI: 10.48550/arXiv.2205.11104.

[26] M. Hausknecht and P. Stone, "Deep Recurrent Q-Learning for Partially Observable MDPs", preprint arXiv:1507.06527, 2015, DOI: 10.48550/arXiv.1507.06527.

[27] L. Meng, R. Gorbet, and D. Kulić, "Memory-based Deep Reinforcement Learning for POMDPs", in 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 5619–5626, 2021, DOI: 10.1109/IROS51168.2021.9636140.

Received 27 May 2026