6.6 Frontier Control Technologies

Chapter Outline

Frontier Technologies and Control Methods

Chapter 6
External Control and Trajectory Planning

6.1 External Control Fundamentals
6.2 Control Principles and Modes
6.3 Trajectory Models and Generation
6.4 Planning and Obstacle Avoidance
6.5 Tracking and Aggressive Flight Control
6.6 Frontier Control Technologies
6.7 Platform and Communication Interfaces
6.8 Examples and Experimental Validation
6.9 Exercises and Classroom Discussion

4 Units in This Section

6.6.1 Autonomous Driving and External Control
Textbook: Sec. 6.1.4.1

6.6.2 End-to-End Control and Reinforcement Learning
Textbook: Sec. 6.1.4.2

6.6.3 Foundation-Model Tasks and Tool Calling
Textbook: Sec. 6.1.4.3

6.6.4 VLA Action Generation and Adaptation
Textbook: Sec. 6.1.4.4; VLA: vision-language-action

Section Introduction: Four Approaches

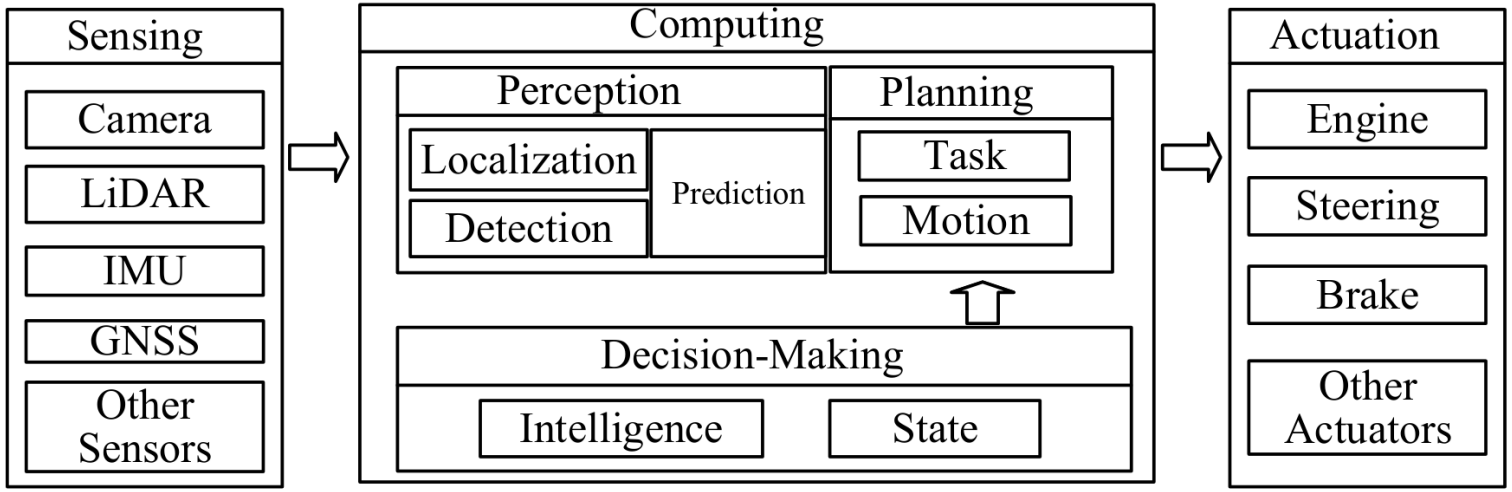
Four Approaches Change Different Parts of the Feedback Loop

Approach	Inputs and outputs	Main change
Modular	Sensor data → environment, trajectory, control quantities	Explicit module responsibilities
End-to-end learning	Selected observations → actions at a selected level	Jointly trains an input–output mapping
Foundation-model task calls	Natural language → tasks or tool parameters	Task representation and scheduling
Vision–Language–Action (VLA)	Images + language → learned actions	Language-conditioned action policy

The four designs can be combined; their order does not imply autonomy levels.
Example: VLA generates actions; conventional control executes them.

6.6.1 Autonomous Driving and External Control: Three System Modules

Sensing, Computing, and Actuation Form the Complete Task Chain



Localization: where am I?
Detection: what is nearby?
Prediction: how will it move?
Decision: select behavior via states
Planning: mission → motion

Figure 6.7 Open-Source Autonomous Driving System Framework

Without a map, simultaneous localization and mapping (SLAM) jointly estimates pose and a map.

6.6.1 Autonomous Driving and External Control: Software Stack Layers

Autoware Separates Planning from Tracking Control

Engineering module	Main inputs	Outputs
Sensing; map	Raw images, point clouds; road geometry	Observation streams and map information
Localization; perception	Observations, map, and state information	Vehicle pose; objects and predictions
Planning	Route goal, state, and environment	Currently executable trajectory
Control	Trajectory and actual state	Steering and longitudinal commands
Vehicle interface	Unified control commands; vehicle state	Chassis commands and vehicle feedback

Mission planning sets destination/route; motion planning generates trajectories.
Control continually corrects tracking error.

6.6.1 Autonomous Driving and External Control: Cross-Platform Comparison

Shared Interface Responsibilities, Different Physical Actions

Comparison	Car external control	Multirotor external control
Typical motion targets	Route, speed, and steering requirements	Trajectory, position, velocity, or attitude
Platform adaptation	Steering/longitudinal commands → chassis protocol	External targets → commands accepted by the autopilot
Feedback evidence	Actual speed, steering angle, and control mode	Pose/velocity, arm state, and mode
Low-level role	Closed-loop propulsion, braking, and steering	Attitude, angular-rate, and actuator response

Interfaces unify semantics; controllers handle each vehicle's physical response.

6.6.1 Autonomous Driving and External Control: Task Feedback Loop

Obstacle Detection Updates Both Behavior and Trajectory



Finite-state machine: states organize task behavior; arrows carry transition conditions.

Teaching car: initial speed 2 m/s; constant deceleration magnitude 1 m/s²; delay neglected.

$$t_{\text{stop}} = \frac{v_0}{a} = \frac{2}{1} = 2\,\text{s} \qquad d_{\text{stop}} = \frac{v_0^2}{2a} = \frac{2^2}{2 \times 1} = 2\,\text{m}$$

Feedback: actual speed below stop threshold; passage occupancy/localization remain valid.

Practical margin: extend braking distance for delay travel, state error and vehicle response.

Chapter Outline

Frontier Technologies and Control Methods

Chapter 6
External Control and Trajectory Planning

6.1 External Control Fundamentals
6.2 Control Principles and Modes
6.3 Trajectory Models and Generation
6.4 Planning and Obstacle Avoidance
6.5 Tracking and Aggressive Flight Control
6.6 Frontier Control Technologies
6.7 Platform and Communication Interfaces
6.8 Examples and Experimental Validation
6.9 Exercises and Classroom Discussion

4 Units in This Section

6.6.1 Autonomous Driving and External Control
Textbook: Sec. 6.1.4.1

6.6.2 End-to-End Control and Reinforcement Learning
Textbook: Sec. 6.1.4.2

6.6.3 Foundation-Model Tasks and Tool Calling
Textbook: Sec. 6.1.4.3

6.6.4 VLA Action Generation and Adaptation
Textbook: Sec. 6.1.4.4; VLA: vision-language-action

6.6.2 End-to-End Control and Reinforcement Learning: Input–Output Boundaries

Define Both Ends Before Calling a Controller End-to-End

Inputs: images or states; e.g., images → trajectory → conventional control.

Output	Execution chain retained	Required learning/validation
Trajectory or position	Conventional tracking control and stable inner loops	Trajectory feasibility, target updates, delay
Velocity, acceleration, or attitude	Some low-level feedback loops	Action scale, coordinates, response, limits
Actuator commands	Actuators and physical vehicle	Fast-loop stability, coupling, hard constraints

End-to-end sets learning boundaries; interface level sets control responsibility.

6.6.2 End-to-End Control and Reinforcement Learning: Benefits and Training

Task Data Underpins the Benefits of Joint Learning

Training method	What the data provides	Main tradeoff
Imitation learning	Paired observations and demonstrated actions	Collection quality matters; few samples away from demonstrations
Reinforcement learning (RL)	Observations, actions, rewards, and next observations	Reward design; exploration cost. No differentiable environment required.
Combined	Demonstrations, then interaction-based improvement	Inherits existing skills; then adapts the policy to the task

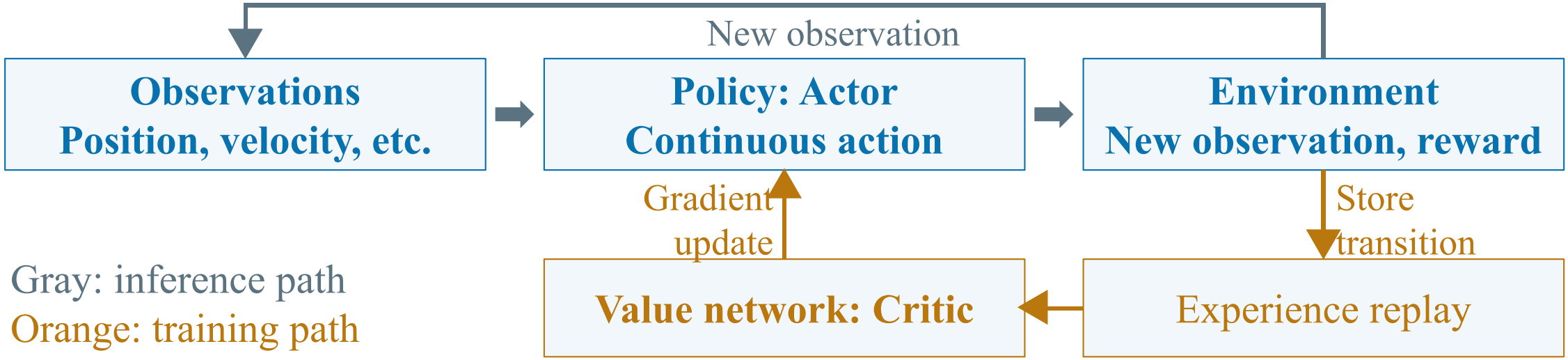
Textbook view: joint learning can treat the whole system as differentiable, reducing losses at handcrafted modules.

This RL needs no environment-dynamics derivatives; benefits still depend on task and data.

6.6.2 End-to-End Control and Reinforcement Learning: Training and Inference

DDPG Learns the Policy in Training and Uses It in Execution

Deep deterministic policy gradient (DDPG): an Actor–Critic method for continuous actions.



Replay stores transitions; the Critic learns action values; the Actor updates toward higher value.

Training return: optimization objective; closed-loop evidence confirms task completion.

6.6.2 End-to-End Control and Reinforcement Learning: Example Structure

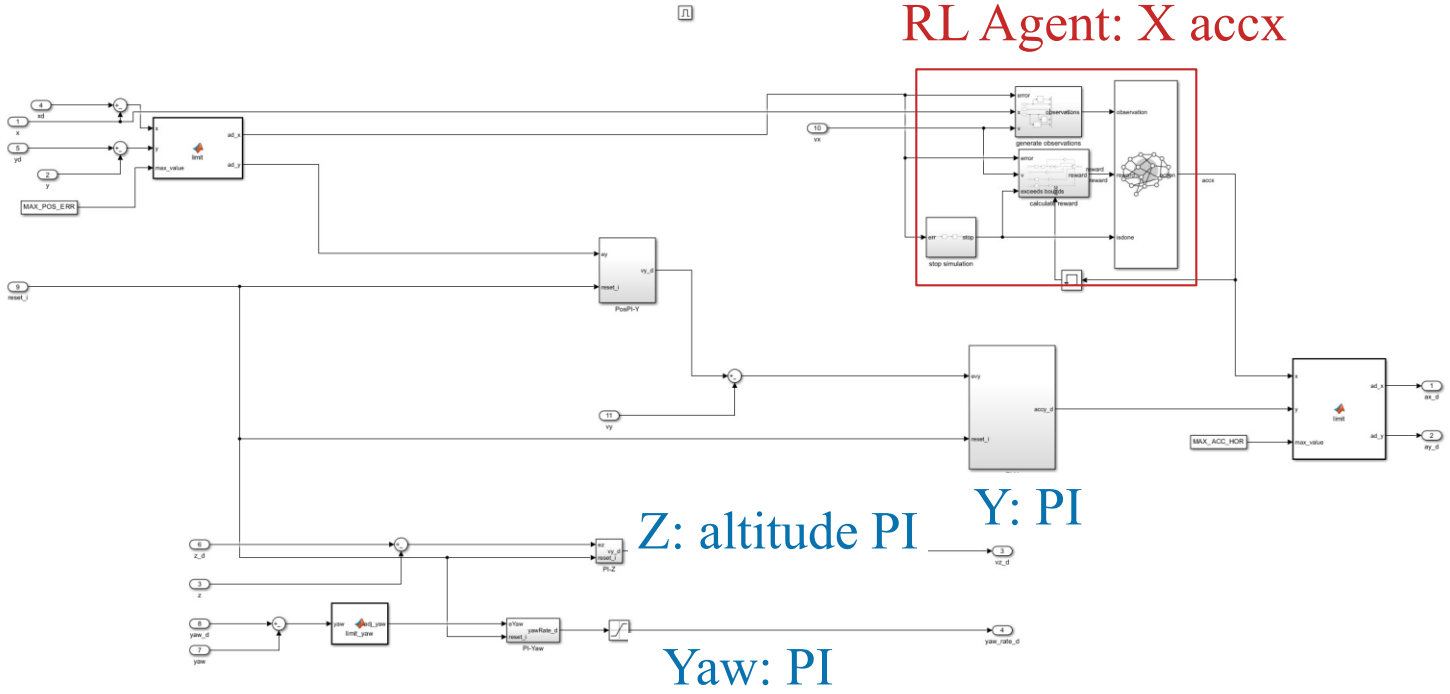
The Actual Example Replaces Only X-Axis Position Control with RL

Directory: [installation directory]/RflySimAPIs/6.RflySimExtCtrl/2.AdvExps/e5.RLExps/1.Simulink/
Training: train/rluav_init.m; validation model: rlhighDynamicAcc.slx.

Interface	Actual variables and meaning	Specification
Observations	Error integral, error, position, velocity, velocity-integral branch, velocity derivative	6D; fixed order and scale
Action	X acceleration accx	1D; range −4 to 4 m/s²
Execution	X policy + conventional Y/Z/Yaw control	Sample time: 0.02 s; other feedback loops retained

6.6.2 End-to-End Control and Reinforcement Learning: Example Model

The RL Agent Block Takes Over Only the X Channel



Position-control subsystem (screenshot from the e5 example Readme)

X: RL Agent outputs accx; a limiter produces the desired acceleration.

Y, Z, yaw: conventional proportional–integral (PI) loops.

Three subsystems generate observations, reward, and termination; sample time 0.02 s.

6.6.2 End-to-End Control and Reinforcement Learning: One Reward Step

How Reward Encourages Arrival and Penalizes Motion

$$r_k = 10I_k - (1 - I_k) - 100B_k - 0.5|v_k| - 0.5|a_{k-1}|$$

$I_k = 1$ means $|e_k| < 0.1$ m; $B_k = 1$ means $|e_k| > 8$ m.
Otherwise 0; e_k: position error; v_k: velocity; a_{k-1}: previous acceleration action.

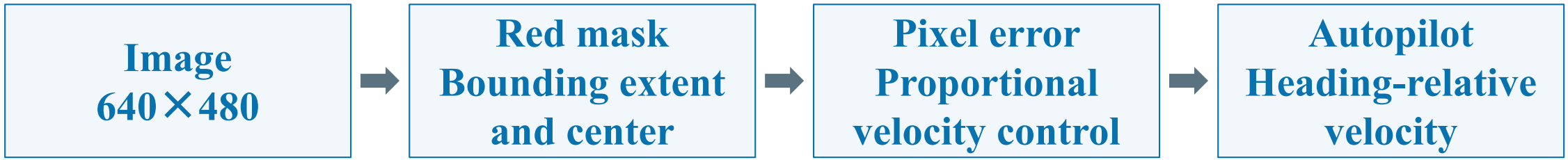
Error / m	Velocity / (m/s); prior action / (m/s²)	Step reward
0.08	0.2; 0.4; within bounds	$10 - 0.1 - 0.2 = 9.7$
0.12	0.2; 0.4; within bounds	$-1 - 0.1 - 0.2 = -1.3$
9.00	0.2; 0.4; out of bounds, terminate	$-1 - 100 - 0.1 - 0.2 = -101.3$

Conclusion: reward differs by 11 across the threshold and dominates policy preference. High reward does not prove smooth motion.

6.6.2 End-to-End Control and Reinforcement Learning: Conventional Vision Comparison

Example 2: HSV Following Uses Visual Proportional Feedback

HSV means hue, saturation, value; thresholds segment the red target.



Entry: 6.RflySimExtCtrl/2.AdvExps/e6.IsaacSimExps/RL_attctrl/circle_follow.py
Output: SendVelFRD(vx, vy, vz, 0); horizontal follows yaw; vertical is down-positive in NED (north-east-down). Units: m/s; yaw rate: rad/s.

Algorithm boundary: HSV thresholds + proportional feedback, with no learned weights; a lost target only skips that frame.

rl.py in this folder: learned policy maps 12D state to attitude/thrust (20 Hz); no images.

6.6.2 End-to-End Control and Reinforcement Learning: One-Step HSV Visual Control

How Pixel Error Becomes a Velocity Command

Given: image center (320, 240) px; target center (360, 220) px;
initial pixel height 100 px, now 80 px. All three errors exceed the 15 px deadband.
Source gains: lateral 0.009, vertical 0.005, forward/backward 0.015, in (m/s)/pixel.

$$v_y = 0.009(360 - 320) = 0.36 \text{ m/s}$$

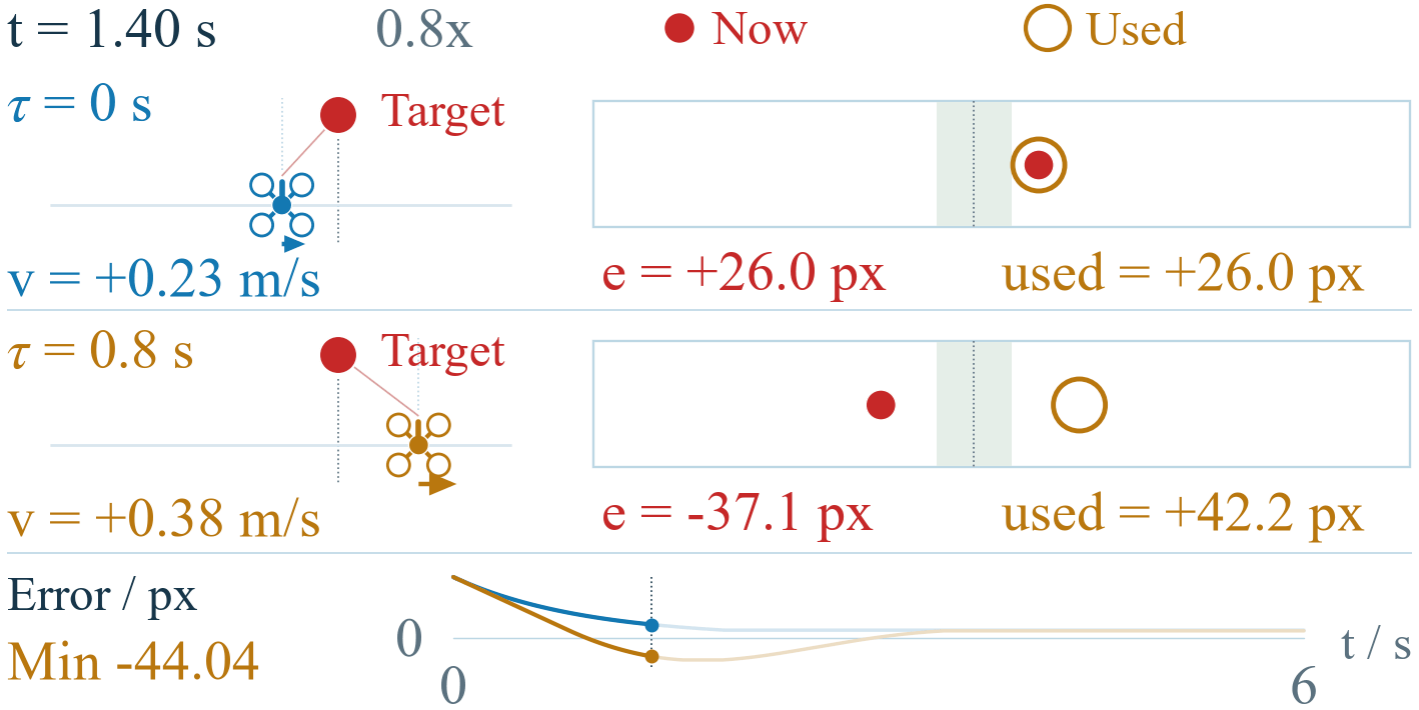
$$v_z = 0.005(220 - 240) = -0.10 \text{ m/s}$$

$$v_x = 0.015(100 - 80) = 0.30 \text{ m/s}$$

Near level: forward-right-down (FRD); horizontal follows yaw; z is NED down, so negative means up.
Pixel height is a distance cue only if target size and orientation stay nearly fixed.

6.6.2 End-to-End Control and Reinforcement Learning: HSV Following with Delay

The Same Control Law Responds Differently with Delay



Old images cause following overshoot (teaching calculation)

Fixed target; ideal response.
Proportional gain: 0.009.
Deadband: ±15 px.

No delay: final 14.75 px.
With 0.8 s delay:
minimum −44.04 px.

$$e_{k+1} = e_k - 120 v_{y,k} \Delta t$$

Pixel scale: 120 px/m;
step 0.02 s; initial error 120 px.
Flight dynamics excluded.

6.6.2 End-to-End Control and Reinforcement Learning: Attitude-Level Policy

Example 3: rl.py Maps State to Attitude and Thrust

RL_attctrl/rl.py loads Isaac Lab weights; inference: 0.05 s (20 Hz). Frames: $v^e = Rv^b$, $v^b = R^{\mathrm{T}}v^e$ (b: body FRD; e: earth NED); installed inverse wrongly uses R.

12D observations
State and target
→
Actor network
64–64, ELU
→
tanh output
4D, range ±1
→
SendAttPX4
Angles + thrust

```
roll_cmd = action[3] * 10;  thrust_cmd = hover_thrust - 0.1 * action[0]
mav.SendAttPX4([roll_cmd, pitch_cmd, yaw_cmd], thrust_cmd)   # angles in degrees
```

Designed observations: 3D each for body velocity, angular velocity, gravity projection and relative target position.

Installed uavAngVel ≠ uavAngRate: all 3 angular-rate inputs are actually zero.

Task: fixed target [100,50,−30] m; no images. Training requires Isaac Lab and a GPU.

Three outputs in order: X acceleration; heading-relative velocity; full-vehicle attitude/thrust.

6.6.2 End-to-End Control and Reinforcement Learning: Data and Generalization

Five Learning-Policy Issues Have Distinct Evidence to Check

Issue	Direct evidence to inspect
Data quality	Image/state/action time alignment; coverage of anomalies and failures
Training convergence	Return distributions across runs; closed-loop error and action magnitude on the same task
Interpretability	Logs of observations, policy actions, limits, and takeover reasons
Unseen scenarios	Independent tests of lighting, objects, initial conditions, and disturbances
Simulation-to-reality gap	Visual changes, computation delay, sensor noise, and dynamics mismatch

Chapter Outline

Frontier Technologies and Control Methods

Chapter 6
External Control and Trajectory Planning

6.1 External Control Fundamentals
6.2 Control Principles and Modes
6.3 Trajectory Models and Generation
6.4 Planning and Obstacle Avoidance
6.5 Tracking and Aggressive Flight Control
6.6 Frontier Control Technologies
6.7 Platform and Communication Interfaces
6.8 Examples and Experimental Validation
6.9 Exercises and Classroom Discussion

4 Units in This Section

6.6.1 Autonomous Driving and External Control
Textbook: Sec. 6.1.4.1

6.6.2 End-to-End Control and Reinforcement Learning
Textbook: Sec. 6.1.4.2

6.6.3 Foundation-Model Tasks and Tool Calling
Textbook: Sec. 6.1.4.3

6.6.4 VLA Action Generation and Adaptation
Textbook: Sec. 6.1.4.4; VLA: vision-language-action

6.6.3 Foundation-Model Tasks and Tool Calling: Model Responsibilities

Language and Visual Understanding Do Not Execute Actions

Large language model (LLM)
Text: interprets tasks; organizes steps/tool parameters

Multimodal large language model (MLLM)
Language, images, audio, etc.: interprets task context

Vision-language model (VLM)
Images + language: locates semantic targets; answers visual questions

Vision foundation model (VFM)
Visual inputs: provides reusable visual representations

Combination: a VFM supplies visual representations and can pair with a language model; it need not itself contain one.

Action-output layer or tool-execution layer
Turns understanding into motion

All four offer understanding and representations; motion still needs an execution layer.

6.6.3 Foundation-Model Tasks and Tool Calling: Two Architectures

Offline Generation and Online Response Have Different Timing Constraints

Dimension	Offline task generation	Online instruction response
Output	Behavior tree: conditional actions; state machine or control code	Current task selection or tool parameters
When checked	Check and revise before deployment	Check state and parameters before every execution
Suitable tasks	Clear boundaries; decomposable procedures	Task steps change with new observations
Main limits	Exception-branch coverage and platform interfaces	Inference delay, network, stale observations

Models express task intent; execution needs verifiable action conditions.

6.6.3 Foundation-Model Tasks and Tool Calling: Code Counterexample

“Fly Forward” Needs Coordinates and Completion Criteria

Example model reply: ["Forward_Backward_Fly", 5, 1]
5 is distance (m); 1 is the assigned UAV ID, unused for vehicle selection in this branch.

```
target_pos = [current_pos[0] + 5, current_pos[1], current_pos[2], 0]
SendPosNED(x, y, z, yaw)
if pos_error < 0.1 or elapsed_time > command_timeout:  # original completion condition
```

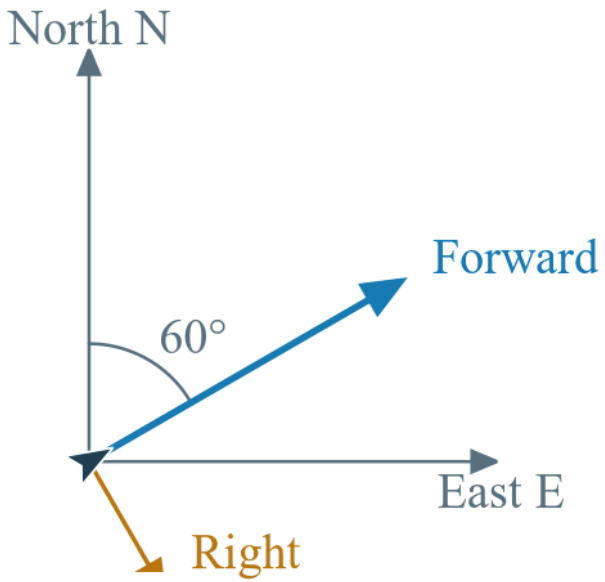
Coordinates: SendPosNED uses NED; +5 m north is not always forward along the heading.

Original defects: timeout also counts as completion; turns check position only and may finish before yaw arrives.
Remedy: test/report position, yaw, timeout separately; reply text is not a flight terminal state.

6.6.3 Foundation-Model Tasks and Tool Calling: Displacement Frames

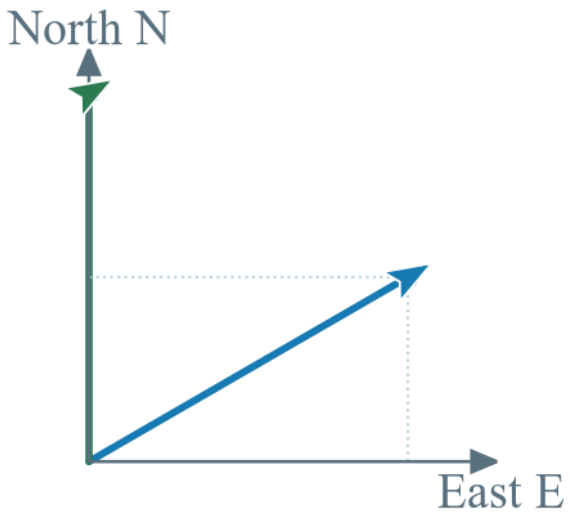
5 m Forward Along the Heading Is Not 5 m North

Fixed yaw 60°



Same start, different endpoints

Progress 100%



Body forward 5 m
$\Delta N = 2.50$ m
$\Delta E = 4.33$ m

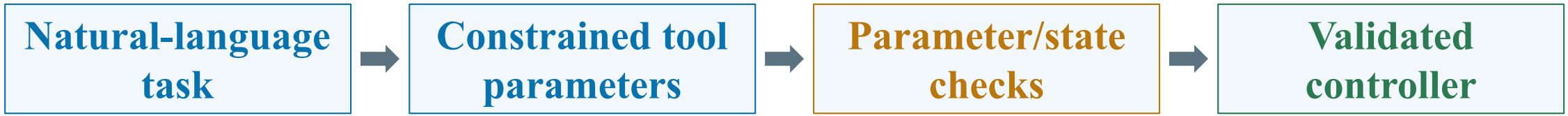
World north 5 m
$\Delta N = 5.00$ m
$\Delta E = 0.00$ m

Displacements at fixed yaw 60°; original code also sets target yaw to 0 (coordinate illustration)

Frames: earth north-east-down (NED), body forward-right-down (FRD); yaw is positive from north toward east. Forward 5 m: $\Delta N = 5\cos 60^\circ = 2.50$ m, $\Delta E = 5\sin 60^\circ = 4.33$ m.

6.6.3 Foundation-Model Tasks and Tool Calling: Tool Interface Contract

Constrained Tools Pass Task Parameters to the Executor



Teaching tool request: goto_ned(request_id="A7", x_m=5, y_m=0, z_m=-2,
yaw_rad=0, timeout_s=20)
Before execution: tool allowlist; finite values; ranges; frames; localization/control state; timestamps.

Return accepted: checks passed; log request ID, target, and acceptance time.

Return succeeded: actual pose, speed, sustained arrival and fresh feedback must jointly confirm success.

Interface contract: agreed module inputs/outputs, units, coordinate frames, rates and timing.

6.6.3 Foundation-Model Tasks and Tool Calling: Tool-State Example

Tool Results Must Distinguish Success, Timeout, and Stale Data

Example criteria: position error <0.1 m; speed <0.1 m/s; yaw error <5°; held for 1 s.
Feedback age (now − feedback timestamp) ≤0.2 s; deadline 20 s. For this example only.
Order: stale → timeout → arrival. Independent rows; defaults: age 0.04 s, elapsed 5 s.

Synthetic input	Result	Reason
0.08 m; 0.04 m/s; yaw 1.7°; held 1.2 s	succeeded	All conditions met for at least 1 s
1.2 m; elapsed 21 s	timeout	Deadline passed without arrival
0 m; feedback age 0.7 s	stale	Feedback too old to confirm state
0.02 m; yaw error 30°	running	Position reached; heading unfinished

Chapter Outline

Frontier Technologies and Control Methods

Chapter 6
External Control and Trajectory Planning

6.1 External Control Fundamentals
6.2 Control Principles and Modes
6.3 Trajectory Models and Generation
6.4 Planning and Obstacle Avoidance
6.5 Tracking and Aggressive Flight Control
6.6 **Frontier Control Technologies**
6.7 Platform and Communication Interfaces
6.8 Examples and Experimental Validation
6.9 Exercises and Classroom Discussion

4 Units in This Section

6.6.1 Autonomous Driving and External Control
Textbook: Sec. 6.1.4.1

6.6.2 End-to-End Control and Reinforcement Learning
Textbook: Sec. 6.1.4.2

6.6.3 Foundation-Model Tasks and Tool Calling
Textbook: Sec. 6.1.4.3

6.6.4 VLA Action Generation and Adaptation
Textbook: Sec. 6.1.4.4; VLA: vision-language-action

6.6.4 VLA Action Generation and Adaptation: Two-Stage Mechanism

VLA Maps Images and Language to Actions

Images + language instructions	→	Vision-language encoder	→	Action decoder
Observed content and task goals jointly condition the policy.		Shared representations retain action-relevant task features.		Discrete tokens or continuous vectors decode to platform actions.

Textbook advantages: unified perception, reasoning, and control; generalization to new tasks/platforms through large-scale data.

Training records: time-align images, language, state, and actions; action definitions must match the actual actuators.

Deployment: decode actions → convert units/frames → check constraints → low-level control → new observations.

6.6.4 VLA Action Generation and Adaptation: OpenVLA Action Interface

OpenVLA's Seven-Dimensional Actions Are Not UAV Throttle

OpenVLA (open research model, 2024): images + language → action tokens → continuous robot actions.

Example 7D action	Physical meaning	What integration must match
3 translation components	End-effector position increments	Meters, frame, incremental or absolute
3 rotation components	End-effector rotation increments	Angle representation, units, rotation order
1 gripper component	Gripper opening/closing	Action definition and platform actuator

Source parameter unnorm_key selects action normalization statistics; a new platform requires matching data and action interfaces.

6.6.4 VLA Action Generation and Adaptation: One Adaptation Step

Action Decoding Must Restore Units and Coordinates

Teaching assumption: trained on flight data; output range [−1, 1], current output 0.4.
Dimension bounds from training: −0.05 m to 0.05 m; linearly recover forward displacement.

$$\Delta x_b = \frac{0.4 + 1}{2}(0.05 - (-0.05)) - 0.05 = 0.020\,\mathrm{m}$$

Convert a level body-forward increment to NED; yaw 30°:

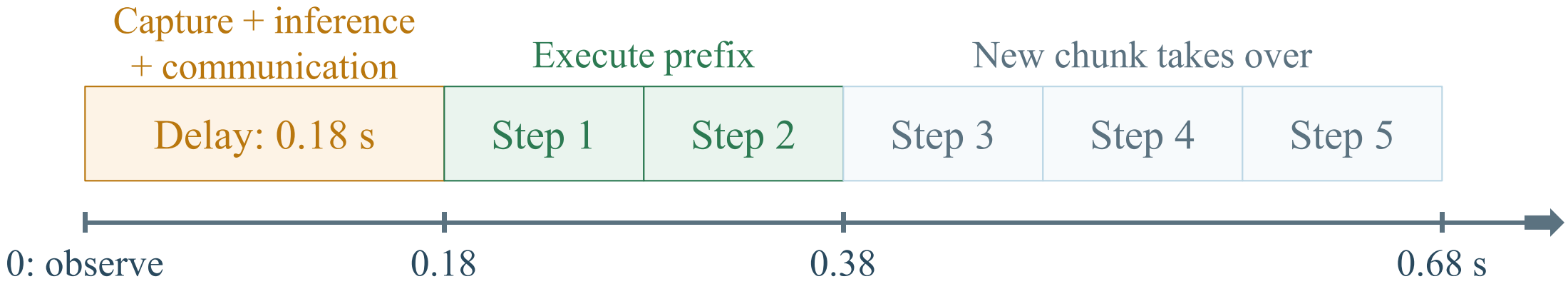
$$\Delta x_N = 0.020\cos 30^\circ = 0.0173\,\mathrm{m}, \quad \Delta y_E = 0.020\sin 30^\circ = 0.0100\,\mathrm{m}$$

The action interface contract also specifies period, relative/absolute targets, angle units, limits, and stale-action handling.

6.6.4 VLA Action Generation and Adaptation: Teaching Timeline for Action Chunking

Action-Chunk Duration Is Not an Allowable Delay

Action chunk: successive actions predicted together; all steps use the same observation frame.



Coverage: 5 steps × 0.1 s = 0.5 s. Step 1 starts only at 0.18 s, still using the image at t=0.

Travel while waiting: 2 m/s × 0.18 s = 0.36 m; the margin must cover this state change.

Feedback update: capture at 0.20 s; a new chunk takes over at 0.38 s. Set task action expiry and takeover conditions.

6.6.4 VLA Action Generation and Adaptation: Evaluation and Trends

Assess Trends through Data, Latency, and Transfer

Direction	Required data or conditions	Comparison evidence
Semantics to actions	Aligned language, images, state, and actions	Paraphrased instructions and new targets
Efficient action generation	Action chunks, parallel decoding, and observation updates	Total delay, stale actions, and takeover count
Platform transfer	Action scale, camera; controller and dynamics	Independent tests after fine-tuning on the new platform
Task reliability	Failure samples and operating constraints	Success rate, error, collision/boundary-violation records

Learned policies may join rule-based planners; constraint checks decide acceptance, replanning or takeover.

Learning Resources

Further Reading on Architectures, Algorithms, and Interfaces

Autonomous driving: architecture and vehicle interfaces (official Autoware docs)
https://docs.autoware.org/main/design/

Continuous-action policy and value learning: original DDPG paper
https://arxiv.org/abs/1509.02971

Language tasks and executable skills: SayCan research project
https://say-can.github.io/

Input format, action decoding, and transfer: official OpenVLA source
https://github.com/openvla/openvla

Classroom Exercises

Identify Control Mechanisms from Inputs, Actions, and Results

Question	Key Points
HSV pixel error → velocity: is this RL/VLA?	No: explicit detection + proportional feedback; no reward training or language-to-action map.
DDPG outputs only accx. What does it replace?	The X position-control mapping; other axes and low-level loops remain.
Model says "success"; tool returns timeout?	Record timeout; use telemetry, the deadline, and terminal state as evidence.
How can a 7D arm action connect directly to flight?	Redefine actions, scales, frames, and period; train with matching flight data.

Section Summary

New Algorithms Still Need a Verifiable Motion Feedback Loop

Modularity: perception, planning, control, and platform interfaces share responsibilities for easy replacement and checks by layer.

Learning control: first define observations, actions, rewards or demonstrations; single-axis learning is not full-vehicle end-to-end control.

Foundation models and VLA: connect semantic and action capabilities to interfaces with frames, units, periods, and conditions.

Validation: evaluate actual feedback, constraints, and terminal states; a tool timeout is not success.

Common errors: treating directory names as mechanisms, training return as control performance, or model text as executed tool actions.

Frontier methods expand task capabilities; reliable execution needs verifiable interfaces.