EZRA ROBOTICS

X30 Architecture & Technical Reference

How the X30 stack is built: hosts, packages, data flow, the safety chain, the protocols to the handheld and to NETRA, operations and tests

Version 1.0 • 2026-11-21

For the developers and integrators of the X30 stack and of systems that talk to it. ROS 1 (Noetic) knowledge assumed.

About this document

For whom. For the developers and integrators of the X30 stack and of systems that talk to it. ROS 1 (Noetic) knowledge assumed.

The settings and their models are in the X30 Tuning Guide, the installation in the X30 Installation Guide. Every package has its own README with the details of its nodes; the teleop protocol is `x30_bridge/PROTOCOL.md`.

The X30 documents

Installation Guide	Getting the stack, the handheld app and NETRA onto a robot and a computer, from nothing to the first patrol.
User Manual	Operating the robot: safety, the Skydroid H12, routes, patrols, inspections, the recorder and NETRA.
Tuning Guide	What every setting does, the models behind them and how to tune a robot for a site; the full parameter reference.
Architecture	How the stack is built: hosts, nodes, topics, frames, the safety pipeline, the protocols to the handheld and to NETRA.
Site Acceptance Test	The handover tests, with pass criteria and sign-off.

All of them, and the same pages as a website with search: `https://darmawanrobotics.github.io/x30_ros/`.

Revisions

Version	Date	Changes
1.0	2026-11-21	First version, for x30_ros 0.19

Contents

About this document . . . 2

1 The system . . . 5
- 1.1 What it is . . . 5
- 1.2 Hosts and network . . . 5
- 1.3 Layers . . . 6
- 1.4 Principles . . . 6

2 Packages . . . 8
- 2.1 Inside a package . . . 9

3 Data flow . . . 10
- 3.1 Topics . . . 10
- 3.2 Services . . . 11
- 3.3 Transforms . . . 12
- 3.4 Messages . . . 12

4 The safety chain . . . 14
- 4.1 Every command through one gate . . . 14
- 4.2 Stops . . . 14
- 4.3 When something falls silent . . . 15
- 4.4 What is not certified . . . 15

5 Sites, routes and the follower . . . 16
- 5.1 On disk . . . 16
- 5.2 The follower . . . 16
- 5.3 A patrol from NETRA . . . 17

6 Inspection and recording . . . 18
- 6.1 The plan . . . 18
- 6.2 At a stop . . . 18
- 6.3 The recorder . . . 18

7 Protocols . . . 20
- 7.1 The handheld: teleop UDP v1 . . . 20
- 7.2 NETRA: robot API v1 . . . 21

8 Running on the robot . . . 23
- 8.1 Services . . . 23
- 8.2 Settings files . . . 23
- 8.3 Logs . . . 23
- 8.4 Tools . . . 24
- 8.5 No internet on the robot . . . 24

9 Tests and continuous integration . . . 25
- 9.1 Tests . . . 25
- 9.2 CI . . . 25

9.3 Versions 25

CHAPTER 1

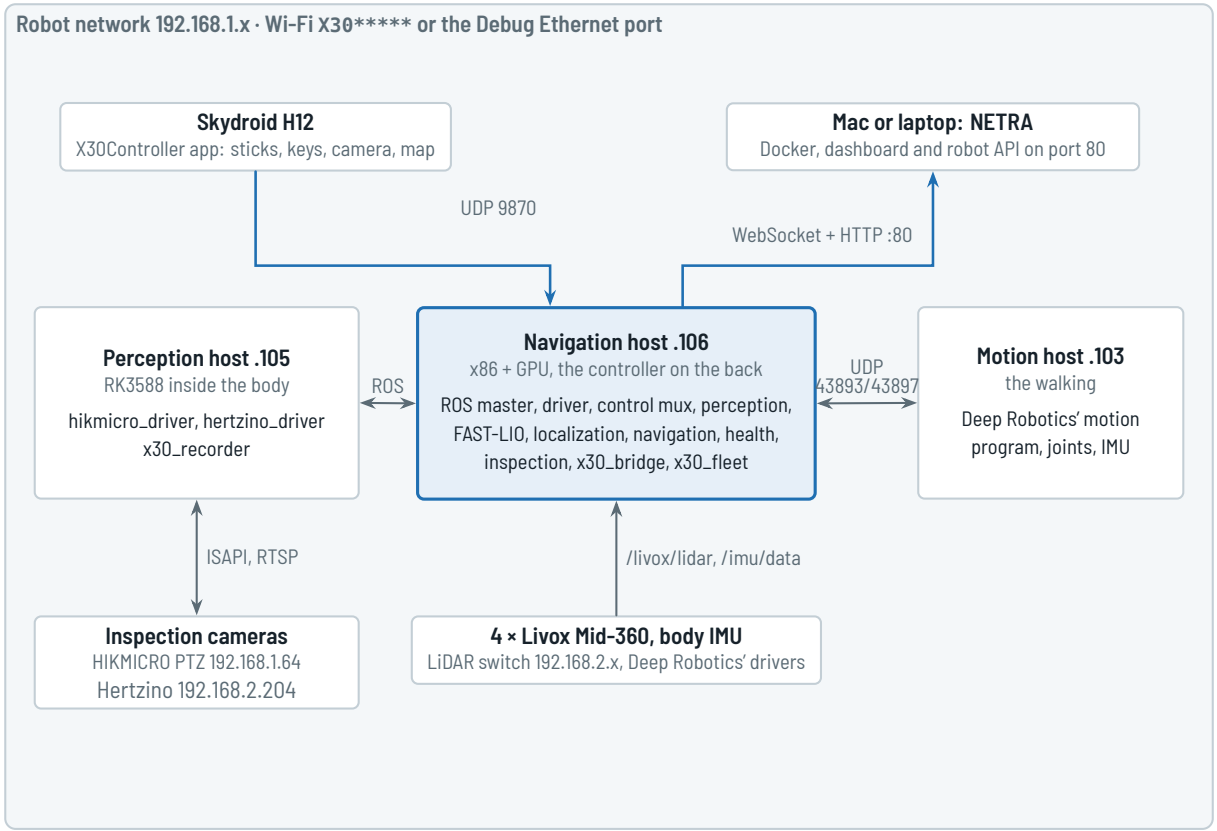
The system

1.1 What it is

`x30_ros` turns a Deep Robotics Jueying X30 Pro into an autonomous substation patrol robot: it walks routes taught by an operator, keeps clear of people and obstacles, inspects equipment with a thermal PTZ camera and an acoustic camera at the stop points, records its cameras, and reports to the NETRA command center. It runs on ROS 1 Noetic on the robot's own computers and replaces Deep Robotics' autonomy (localization, navigation, mapping), which it turns off; Deep Robotics' motion control, LiDAR and IMU drivers stay.

It is the Noetic port of Dharma, the robot-agnostic stack shared with Pashupati (ROS 2, mall patrol) and HotDog (ROS 2, substation patrol on a Deep Robotics M20): the same packages, models and protocols, so a handheld or a NETRA that knows one knows the others.

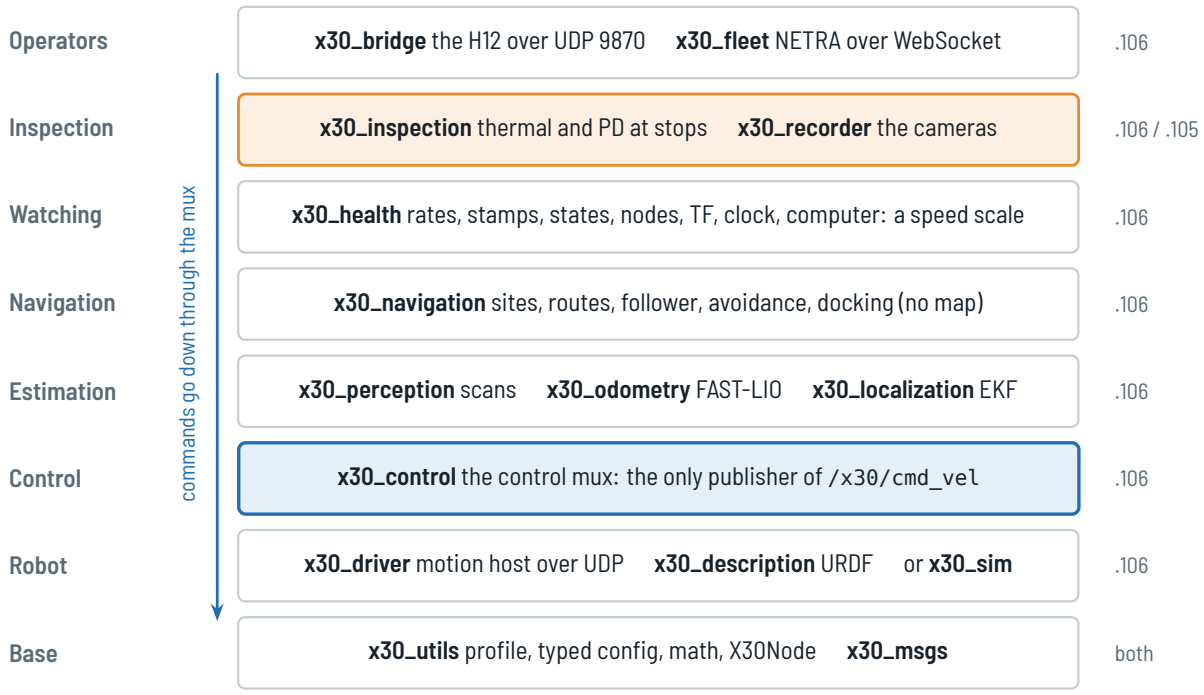
1.2 Hosts and network



Host	Address	Runs
Motion host	192.168.1.103	Deep Robotics' locomotion: gait, joints, its UDP control interface
Perception host	192.168.1.105	the inspection cameras' drivers, `x30_recorder`; Deep Robotics' elevation map and charging
Navigation host (intelligent controller)	192.168.1.106	the ROS master (Deep Robotics'), the LiDAR and IMU drivers (Deep Robotics'), and the whole x30_ros stack: driver, mux, scans, odometry, localization, navigation, inspection, health, the handheld gateway, the NETRA uplink
HIKMICRO PTZ	192.168.1.64	thermal and visible camera (ISAPI, RTSP)
Hertzino HZ-FA-371	192.168.2.204	acoustic camera, on the LiDAR network
Skydroid H12	robot's Wi-Fi	X30Controller, UDP to 9870
NETRA	a laptop or server	the command center, port 80 (HTTP, WebSocket)

Both robot hosts share one ROS master on the navigation host (`ROS_MASTER_URI=http://192.168.1.106:11311`), each with its own `ROS_IP`. The navigation host carries everything that drives or guards the robot, so a perception host busy decoding video never delays a stop.

1.3 Layers



1.4 Principles

One publisher of `cmd_vel` every motion passes the control mux: arbitration, the envelope, the protective field, the e-stop (Chapter 4).

No map the start spot is the origin; routes are taught and repeated in that frame; tags correct the drift. Nothing to build or maintain per site but the routes.

Pure cores every algorithm lives in a `core/` module without ROS, tested on its own; the nodes are thin shells around them.

Typed configuration every parameter declared once with unit, range and meaning, validated at start; live ones re-read every second; the documentation's tables are generated from them.

Secrets in the environment tokens and passwords only as environment variables, never as ROS parameters (which `roslaunch` prints and the parameter server shares).

Nothing lost offline what NETRA must get waits in a spool on disk; what the robot records stays on its disk until room is needed.

CHAPTER 2

Packages

Package	Host	Nodes	Role
`x30_driver`	nav	`x30_driver`	the motion host's UDP protocol (API V1.0.5): `cmd_vel` as axis commands, leg odometry, IMU, joint states, battery, the robot's services
`x30_control`	nav	`control_mux_node`	the only publisher of `cmd_vel`: AUTO or REMOTE, envelope, protective field, e-stop
`x30_perception`	nav	`lidar_sector_node`, `aruco_detector_node`	sector scans for the field and avoidance; fiducial tags
`x30_odometry`	nav	`wait_still`, FAST-LIO, `odometry_monitor_node`	LiDAR-inertial odometry behind its start gate, watched
`x30_localization`	nav	`localization_node`	${}^{map}\mathbf{T}_{odom}$ from an SE(2) EKF on tags; the levelled bridge of FAST-LIO into the TF tree
`x30_navigation`	nav	`site_manager_node`, `route_recorder_node`, `path_loader_node`, `path_follower_node`, `obstacle_avoidance_node`, `docking_node`	sites and routes, teaching, following, avoidance, docking
`x30_inspection`	nav	`inspection_node`	thermal and partial discharge inspection at the stop points; reports
`x30_recorder`	perc	`recorder_node`	the cameras recorded on the perception host
`x30_health`	nav	`health_monitor_node`	one health signal: rates, nodes, states, TF, the computer
`x30_bridge`	nav	`teleop_udp_node`	the handheld's UDP gateway (teleop v1)
`x30_fleet`	nav	`fleet_uplink_node`	the NETRA uplink (robot API v1)
`x30_sim`	any	`sim_robot_node`	the X30 in software for the whole stack
`x30_description`	nav	`robot_state_publisher`	URDF from Deep Robotics' model
`x30_bringup`	both		launch files per host, the robot profile, tools
`x30_msgs`	both		messages and services
`x30_utils`	both		math, typed configuration, the robot profile, the braking model, logging, ROS helpers, test harness
`hikmicro_driver`	perc	`hikmicro_ptz_node`, `hikmicro_camera_node`	HIKMICRO PTZ: thermometry, PTZ, the visible and thermal images (own repository, `x30.repos`)
`hertzino_driver`	perc	`hertzino_acoustic_node`, `hertzino_camera_node`	Hertzino acoustic camera: partial discharge detection, its images (own repository)
FAST-LIO, Livox driver	nav		third party, as submodules (`x30.repos`)

2.1 Inside a package

```
x30_<name>/
  src/x30_<name>/core/      the algorithms, pure Python, no ROS (tested alone)
  src/x30_<name>/utils/     conversions between messages and the core's types
  src/x30_<name>/*_node.py  the nodes: parameters, topics, services around the core
  config/*.yaml             the X30's settings
  launch/*.launch
  test/unit/                tests of the core (pytest, no ROS)
  test/ros/                 tests of the nodes on a ROS master (rostest)
  README.md
```

CHAPTER 3

Data flow

3.1 Topics

Names relative to the node's namespace; the nodes run in the global namespace, the driver in `/x30`. Latched topics carry a state and arrive at once on subscription.

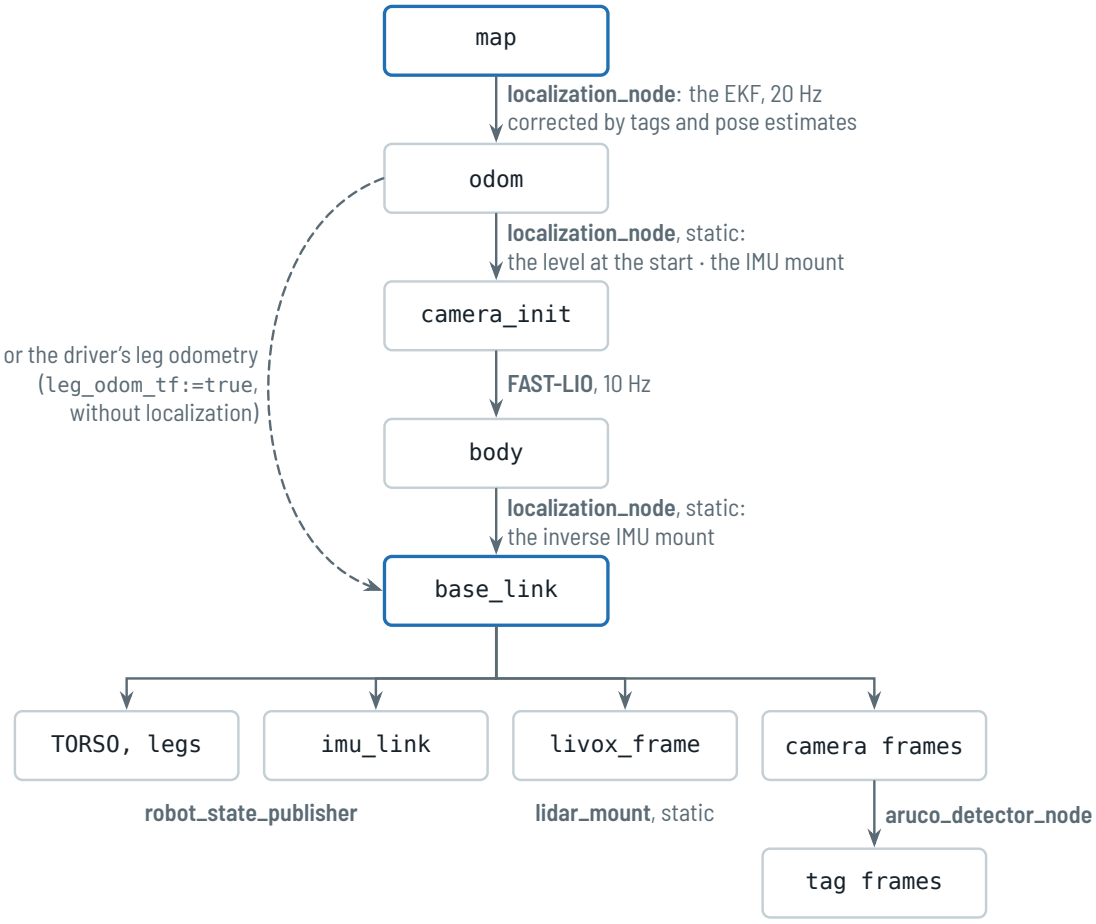
Topic	Type	From	To
Motion			
`/x30/cmd_vel`	Twist	control mux	`x30_driver`
`control/nav_cmd_vel`	Twist	path follower, docking	control mux, avoidance
`control/remote/handheld/cmd_vel`	Twist	handheld gateway	control mux
`control/estop_request`	String	gateway, uplink	control mux (latches at once)
`control/status`	ControlStatus (latched)	control mux	follower, docking, health, gateway, uplink
Sensing			
`/livox/lidar`	Livox CustomMsg	Deep Robotics' driver	scans, FAST-LIO
`/imu/data`	Imu	Deep Robotics' driver	start gate, FAST-LIO, monitor, levelling
`perception/safety_scan`	SectorScan	`lidar_sector_node`	control mux, follower, health
`perception/sector_scan`	SectorScan	`lidar_sector_node`	avoidance
`/Odometry`	Odometry	FAST-LIO	monitor, localization
`odometry/status`	OdometryStatus (latched)	odometry monitor	health
`odometry/trip`	TripStatus (latched)	odometry monitor	gateway
`odometry/restarted`	Empty	odometry monitor	localization
Where and where to			
`localization/status`	LocalizationStatus	localization	follower, route recorder, health
`localization/pose`	PoseWithCovariance	localization	recorder
`site/status`	SiteStatus (latched)	site manager	route recorder, inspection, recorder
`navigation/waypoints`	WaypointPath	path loader	follower
`navigation/avoidance`	AvoidanceCommand	avoidance	follower
`navigation/status`	NavigationStatus (latched)	follower	gateway, uplink, recorder
`navigation/stop_point`	StopPointEvent	follower	inspection
`route/saved`	String	route recorder	uplink
Inspection and the rest			
`inspection/report`	InspectionReport	inspection	uplink
`inspection/status`	InspectionStatus (latched)	inspection	gateway, uplink
`recorder/status`	RecorderStatus (latched)	recorder	gateway, uplink

Topic	Type	From	To
`health/status`	HealthStatus (latched)	health monitor	control mux, gateway, uplink
`/diagnostics`	DiagnosticArray	health monitor	gateway, uplink (the computer's figures)
`/x30/diagnostics`	DiagnosticArray	driver	gateway, uplink (the driver's status)

3.2 Services

Service	Type	Does
`control/set_mode`	SetMode	AUTO or REMOTE (with the source)
`control/estop/engage`, `release`	Trigger	the software e-stop
`navigation/start`, `stop`, `pause`, `resume`	Trigger	the follower; start waits for a pose
`navigation/set_path`	SetPath	a path given directly
`navigation/load_path`	LoadPath	a route CSV of the sites directory
`route/start`, `stop`, `mark_stop_point`	Trigger, MarkStopPoint	teach a route
`site/list`, `site/load`	ListSites, LoadSite	the sites
`localization/start`	Trigger	localize from a tag in view (no gate)
`localization/reset`, `assume_origin`	Trigger	forget ${}^{map}\mathbf{T}_{odom}$; take odom as the map
`localization/record_tags`, `load_tags`	Trigger, LoadFile	the site's tags
`docking/dock`, `undock`, `cancel`	Trigger	docking
`inspection/inspect`	Trigger	the point where the robot stands (minutes)
`inspection/add_target`	AddTarget	teach a target from the camera's current pose
`recorder/start`, `stop`	Trigger	manual recording
`odometry/trip_reset`	Trigger	the trip meter
`/x30/stand_up`, `sit_down`, `gait_walk`, …	Trigger	the driver's robot commands

3.3 Transforms



${}^{map}\mathbf{T}_{odom}$ `localization_node`: identity at the start (the start spot is the origin), corrected by tags. A correction is published as the old value just before its time and the new one at it, so a lookup between two corrections never interpolates across a jump.

${}^{odom}\mathbf{T}_{base_link}$ the localization's bridge of FAST-LIO: FAST-LIO's pose levelled by gravity at the start (the IMU's up direction gives roll and pitch), so `odom`'s z is up even when the robot started on a slope.

base_link to the sensors `robot_state_publisher` from the URDF; the LiDAR's frame from `lidar_pose`.

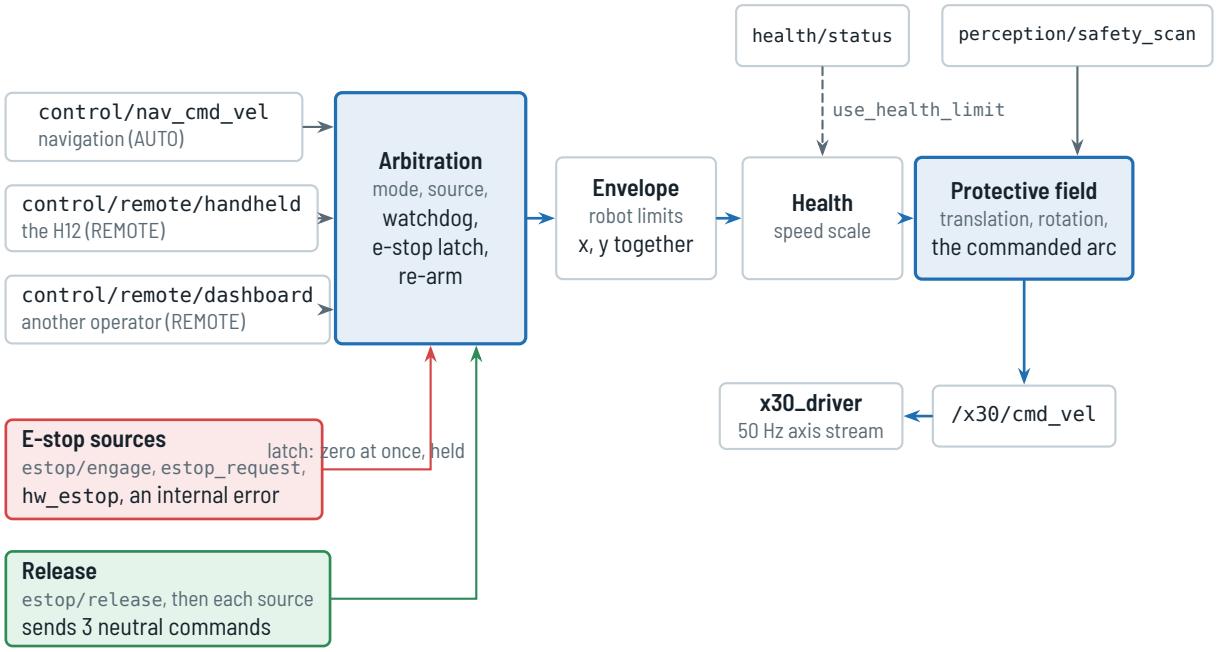
3.4 Messages

Message	Carries
ControlStatus	who drives, the e-stop and its reasons, the protective field's state
SectorScan	the nearest obstacle per angular sector (REP-103), blocked sectors
OdometryStatus, TripStatus	trust in the odometry; distance and speeds since the reset
LocalizationStatus	NONE, LOCALIZED or UNCERTAIN, the last correction's source, σ_{xy}, σ_{ψ}, the time since the last correction
SiteStatus	the active site and what it holds
Waypoint, WaypointPath	a route: poses with their dwell (stop points)
NavigationStatus	the follower's state, progress, pause reasons, message
StopPointEvent	the follower at a stop point

Message	Carries
AvoidanceCommand	reactive avoidance's bend of the command, its emergency
DockingStatus	docking onto a tagged charger
InspectionReport, InspectionFinding	everything measured at a point; one criterion's result
InspectionStatus	ready, busy, the last point's severity
RecorderStatus	what the perception host records, where, the free disk
HealthStatus	the overall level, the issues, the speed scale

CHAPTER 4

The safety chain



4.1 Every command through one gate

Only `control_mux_node` publishes `/x30/cmd_vel`. Navigation (`control/nav_cmd_vel`) and the operator sources (`control/remote/<source>/cmd_vel`) publish to it; it forwards one of them, AUTO or REMOTE, after the envelope, the health speed scale (optional) and the protective field, and holds zeros during an e-stop. The follower has its own guard on the same scan, with margins at least the field's: in normal operation the field never has to act.

4.2 Stops

Stop	Path	Effect
Hard emergency stop (robot)	Deep Robotics' hardware	the joints off: the robot drops
H12 STOP	gateway: `control/estop_request` at once, then `control/estop/engage`	zeros held; a patrol pauses; release by a held STOP, then neutral sticks
NETRA **E-stop**	uplink: the same two	as above
`emergency_stop` command	the driver's service; the e-stop latched too	the robot's own emergency stop
`control/hw_estop`	a hardware input (optional)	engaged while true or silent for `hw_estop_timeout_sec`
Deep Robotics' handheld	takes over the robot (manual mode)	the stack's commands no longer reach the legs until the robot is back in its mode

4.3 When something falls silent

Silent	After	Then
The handheld (stick)	0.3 s	the mux sends one zero; the gateway drops the handheld after 2 s
Navigation	0.5 s	the mux sends one zero
The safety scan	0.5 s	the field: STALE (AUTO stops, REMOTE creeps); the follower holds
The LiDAR (no cloud)	0.5 s	every sector BLOCKED
The robot's pose	0.5 s	the follower holds
The mux	0.5 s	the driver sends zeros, then stops streaming
The driver	1 s	the robot drops the axis by itself
A critical node, the ROS master	watchdog	`x30_watchdog.py` restarts the host's launch

4.4 What is not certified

The e-stops of the H12 and NETRA, the protective field and the guard are software on a computer without a safety rating. They reduce risk; the hard emergency stop on the robot, Deep Robotics' handheld and a responsible operator are the protection, and the site's risk assessment decides where the robot may walk.

CHAPTER 5

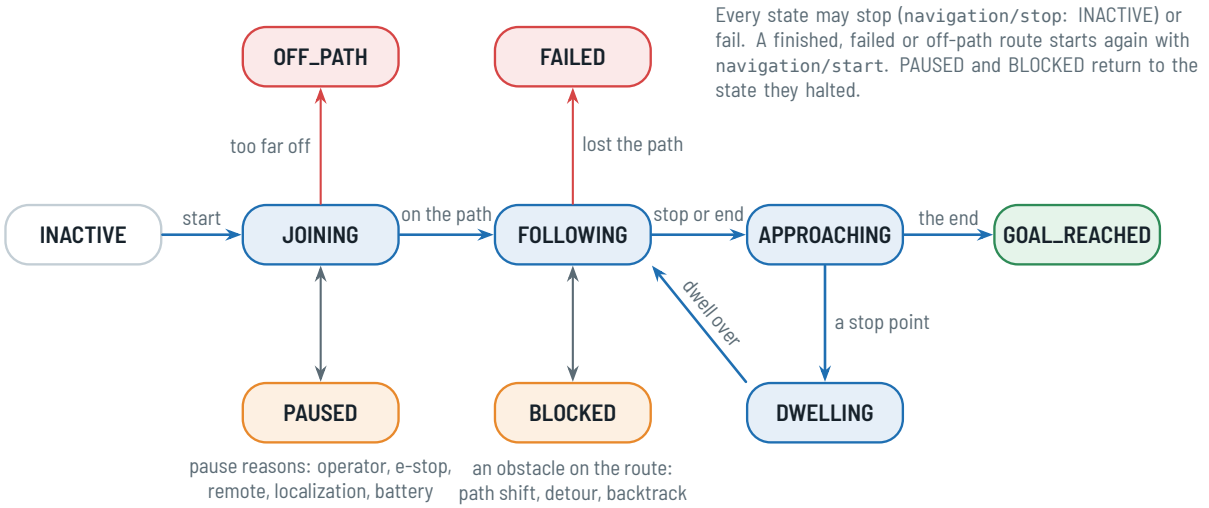
Sites, routes and the follower

5.1 On disk

```
~/x30/map/<site>/              a site: a substation (1-48 of a-z, 0-9, _ and -)
  tags.json                    recorded tags: their map poses
  routes/<name>.csv            taught routes: x, y (m, map frame), yaw_deg, dwell_sec
  inspection.yaml              the site's own inspection points and targets
  netra/<route_id>.csv         routes NETRA started a patrol on
  netra/inspection.yaml        NETRA's plan for them (points netra_stop<k>, targets netra_<
    id>)
~/x30/reports/<day>/...        inspection reports
~/x30/recordings/<day>/<time>_<reason>/   recorder sessions (perception host)
~/x30/spool/                   uploads waiting for NETRA
~/x30/logs/                    each node's JSON lines log
```

A row with `dwell_sec` above zero is a stop point. Files are written atomically (a temporary file renamed), so a power cut never leaves half a route.

5.2 The follower

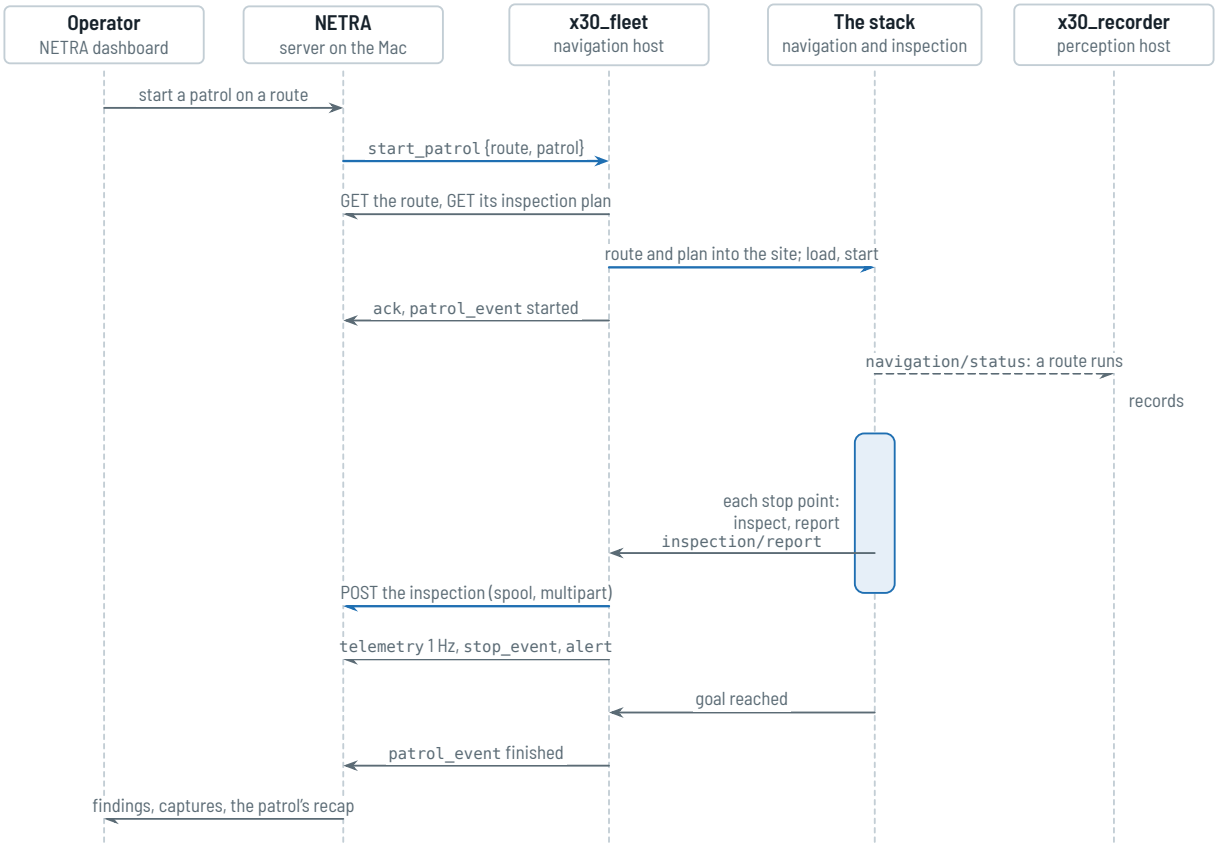


State	Meaning
INACTIVE	no route running
JOINING	walking onto the route from up to `join.max_distance` away
FOLLOWING	on the route, at the speed profile's speed
APPROACHING	converging onto a stop point or the route's end
DWELLING	standing at a stop point: the inspection runs
PAUSED	a pause reason holds it: the operator, the e-stop, REMOTE, localization (NONE), the battery, disarmed

State	Meaning
BLOCKED	standing still too long: an obstacle path shift, detours and backing up could not pass
OFF_PATH	farther from the route than `off_path_stop`
GOAL_REACHED	the route's end
FAILED	a problem the follower cannot leave by itself

The follower is a pure state machine: `Follower.tick(inputs) -> Decision` every 50 ms, with the scan, the pose, the avoidance output and the time as its inputs; the node only gathers them and publishes the decision. Every state may go to INACTIVE (stop) or FAILED.

5.3 A patrol from NETRA



CHAPTER 6

Inspection and recording

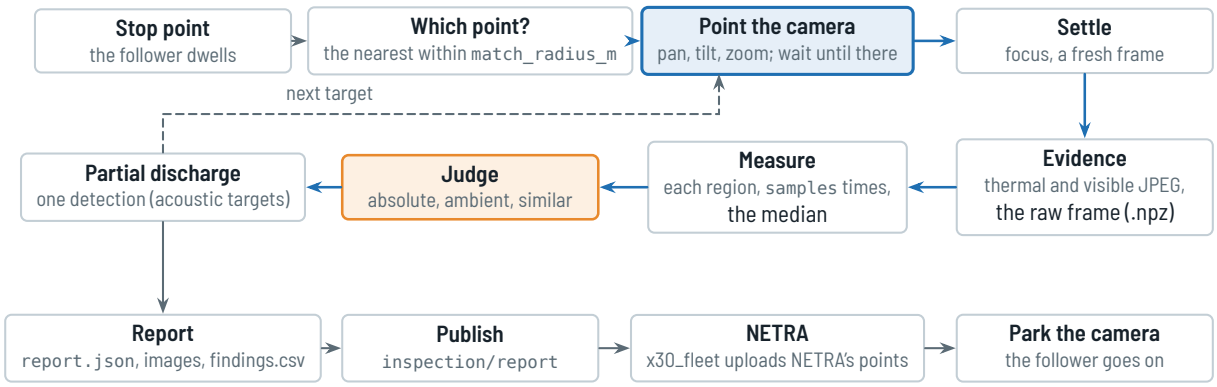
6.1 The plan

Inspection points are places where the robot stands (map frame), each with its targets: where the PTZ camera looks (pan, tilt, zoom) and what is checked there. Points are matched by position, not by route: every route that stops within `match_radius_m` of a point inspects it.

```
points:
  - name: bay1_disconnector
    x: 12.40
    y: -3.10
    targets:
      - name: bay1_ds_clamps
        pan_deg: 35.0
        tilt_deg: -10.0
        zoom: 4.0
        thermal: {enabled: true, strips: 3, max_temp_c: 0.0}
        acoustic: {enabled: true, duration_sec: 5.0}
```

The site's own file and NETRA's plan are merged; a NETRA point never replaces a point of the same name in the site's file.

6.2 At a stop



`inspection_node` answers `navigation/stop_point`: for each target it moves the camera (`hikmicro_ptz_node` on the perception host), waits until it settles, takes the temperatures of the regions several times (median), a visible and a thermal image and the raw temperatures, runs the acoustic detection, judges the findings (Tuning Guide, "Inspection") and publishes one `inspection/report`. The report folder is written before the message goes out, so the uplink always finds the files it names.

6.3 The recorder

`recorder_node` on the perception host writes a session per patrol or request:

```
~/x30/recordings/<YYYYMMDD>/<HHMMSS>_<patrol|manual>/
  session.json              reason, start and end, site, what was recorded (rewritten as
    it grows)
  thermal/*.npz             a raw frame each: degC per pixel, the pose, pan, tilt, zoom
  video/<stream>_*.mp4      the camera's streams, segment by segment (ffmpeg, RTSP), and
    their logs
  acoustic/*.npz            the acoustic camera's data in chunks, and detections.jsonl
```

Files go to the disk through a bounded queue; a full queue drops and counts rather than blocks. The video is copied as the camera encodes it (no re-encoding on the robot).

CHAPTER 7

Protocols

7.1 The handheld: teleop UDP v1

The X30 speaks the teleop protocol v1 of Pashupati: the same datagrams, so a handheld of either robot drives the other; the X30's additions are ignored by a v1 handheld. One UDP port on the navigation host (9 870); the handheld sends JSON, the robot answers JSON (control) and chunked binary datagrams (streams). The full reference is `x30_bridge/PROTOCOL.md`.

7.1.1 Session

1. The handheld sends `hello` (again every second when idle). The robot answers `welcome` (its commands, the stick's limits, the camera names) and the route, then streams `state` at 10 Hz.
2. One driver at a time: another device's `hello` is answered `busy` while the current one lives; after 2 s of silence the robot drops it and zeroes the stick.
3. With a shared secret (`X30_TELEOP_TOKEN`) every message carries `"k"`, compared in constant time; a `hello` without it is answered `denied`, anything else dropped.
4. Discovery: a broadcast `{"t": "discover"}` is answered by every robot's `beacon` (name, `secured`, `in_use_by`, mode, battery), without a token.

7.1.2 Messages

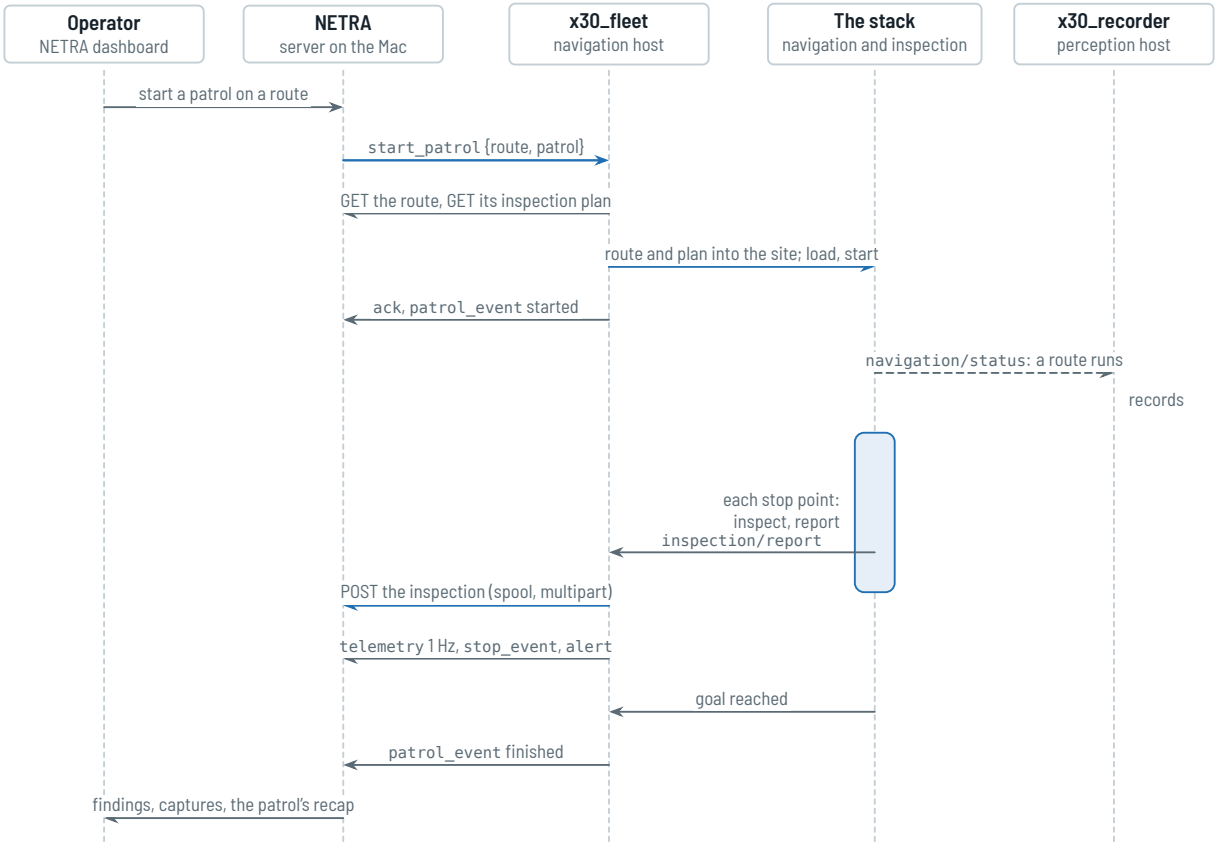
`t`	Fields	
`hello`	`name`, `streams`	register, keep alive
`joy`	`vx`, `vy` (m/s), `wz` (rad/s)	the stick, 20–30 Hz; drives in `remote` only
`cmd`	`id`, `name`, `args`	a command; resent every 200 ms until its `ack`; runs once per `id`
`streams`	`camera`, `lidar`, `video`, `camera_fps`, `camera_name`	which streams
`ack`	`id`, `ok` (`null`: pending), `msg`, `data`	a command's answer; long ones (`inspect`, `nav_start`) answer pending first
`state`	mode, e-stop, field, nav state, pose, battery, health, trip, speed, **X30**: camera, recorder, substation	the robot

Commands map one to one onto the stack's services (Chapter 3): `estop`, `set_mode`, `nav_start`…, `record_start`, `mark_stop`, `list_routes`, `load_route`, `localize`, `record_tags`, `dock`, `nav_speed`, `trip_reset`, and the X30's `recorder_start`, `recorder_stop`, `inspect`, plus the driver's robot commands.

7.1.3 Binary streams

Each datagram starts with a 13-byte little-endian header: magic `PSH1`, `u8` stream (1 camera, 2 LiDAR, 3 route), `u32` frame, `u16` chunk index, `u16` chunk count; chunks of at most 1400 bytes. The camera is one JPEG per frame, the LiDAR $N \times$ (`int16` x, y, z in cm, base_link), the route a JSON list of $[x, y,$ dwell] in the map frame. The robot subscribes to a camera and the cloud only while a handheld wants them.

7.2 NETRA: robot API v1



7.2.1 The WebSocket

`/api/v1/robot/ws` with `Authorization: Bearer <token>` and `X-Robot-Id`. The robot sends `hello`, `telemetry` (1 Hz), `patrol_event` (started, lap, finished), `stop_event`, `alert` (blocked, off the path, failed, the e-stop), `ack` and `time_sync`; events wait while NETRA is unreachable and the connection comes back with a backoff. NETRA sends commands, acknowledged with the service's answer:

Command	The stack
`start_patrol {route_id, patrol_id}`	the route and its plan into the site, then `navigation/load_path` and `navigation/start`
`stop_patrol`, `pause`, `resume`	`navigation/stop`, `pause`, `resume`
`localize`	`localization/start`
`estop`, `estop_release`	`control/estop_request` at once, then `control/estop/engage`; `release`
`dock`, `undock`	`docking/dock`, `undock`
`recorder_start`, `recorder_stop`	`recorder/start`, `stop`
`inspect_point {point}`	one NETRA point where the robot stands; refused while moving

7.2.2 Downloads and uploads

`GET /api/v1/robot/routes/<id>/file` the route CSV, into `<site>/netra/`.

`GET /api/v1/robot/routes/<id>/inspection` NETRA's inspection points of the route, by stop: each stop a point `netra_stop<k>`, each NETRA point a target `netra_<id>` (its PTZ pose, its regions, its temperature limit).

`POST /api/v1/robot/inspections` one NETRA point's result, **always multipart**: a `meta` part (JSON: `uid` `<robot_id>.<report>.<point>`, the time on NETRA's clock, the patrol, the pose, the ambient, the readings) and the images `thermal`, `visual`, `acoustic`. The `uid` makes a retry idempotent.

`POST /api/v1/robot/routes` a route taught on the robot, for NETRA's route database.

Uploads go through the spool (`~/x30/spool`): an item leaves the robot's disk only once NETRA answered it, so nothing is lost offline or across restarts.

CHAPTER 8

Running on the robot

8.1 Services

Each host runs its part as a systemd service, `x30-navigation` and `x30-perception` (`tools/x30_host.sh install <role>`):

`Type=notify`, `WatchdogSec=60` `x30_watchdog.py` runs the host's launch file and reports to systemd while the ROS master answers and every critical node is registered; it restarts the launch when the master restarts or a critical node stays away.

`Restart=on-failure`, `StartLimitIntervalSec=0` never given up on.

Order the navigation host's service starts after Deep Robotics' services (`jy_*`) and the one running the ROS master; it never starts a master of its own (Deep Robotics' LiDAR and IMU drivers depend on that master).

Deep Robotics' autonomy `tools/dr_services.sh off` at every start, and a sweep every 30 s by the watchdog: what Deep Robotics restarts late goes again (the journal names it).

8.2 Settings files

`/etc/x30/<role>.env`, mode 600, owned by root and the robot's user: `X30_LAUNCH_ARGS` (the launch arguments, shell words), the secrets (`X30_TELEOP_TOKEN`, `X30_NETRA_TOKEN`, `HIKMICRO_PASSWORD`, `HERTZINO_PASSWORD`), `X30_PYTHON` for NeuPAN, the log settings. `install` never overwrites an existing file.

8.3 Logs

The journal `journalctl -u x30-navigation`: the service, the watchdog's verdicts.

Each node's JSON lines `~/x30/logs` (`X30_LOG_DIR`): one event per line, with the node, the level, the message and its fields; easy to filter with `jq`.

ROS's own logs pruned at every start to `X30_ROS_LOG_MB` (2 048 MB), the oldest runs first, the newest five kept.

The black box (`blackbox:=true`) the last hours of status, commands, pose and log in a rolling rosbag.

8.4 Tools

Tool	Does
`x30_host.sh env|check|start|install`	the ROS network of a host, a full check of both hosts, the launch by hand or as the service
`dr_services.sh status|off|on`	Deep Robotics' own localization, navigation, mapping and bridge off, now and at boot
`check_deps.sh`	what the build and the stack need, checked offline
`pack_sources.sh`	the stack, its repositories and the Livox SDK in one archive, on a computer with internet
`install_neupan.sh`	NeuPAN's environment for the `neupan` controller
`x30_check.py`	the ROS rows of `x30_host.sh check`: the master and every node reachable, the other host's clock, the cameras' ports

8.5 No internet on the robot

The robot never downloads: `pack_sources.sh` archives the sources on a computer with internet, `scp` carries the archive, `check_deps.sh` checks the apt packages offline (or through an SSH proxy of the computer). NETRA builds its images once with internet and starts offline after.

CHAPTER 9

Tests and continuous integration

9.1 Tests

Package	Tests	Package	Tests
x30_navigation	261	x30_health	62
x30_driver	129	x30_inspection	60
x30_sim	95	x30_localization	50
x30_control	93	x30_bridge	32
x30_perception	83	x30_fleet	32
x30_utils	82	x30_recorder	23
x30_odometry	72	x30_description	6
x30_bringup	66	**all**	**1146**

Unit tests (`test/unit`, pytest) of the pure cores: the braking model, the field, the EKF, the follower's state machine, path shift, the criteria, the protocols, without ROS.

Node tests (`test/ros`, rostest) on a ROS master: the nodes with their real parameters, topics and services, the simulated X30 driving routes end to end, a fake NETRA for the uplink, the gateway with a simulated handheld (`teleop_rig.py`).

Style `ruff` on every package (`ruff.toml`), `shellcheck` on the tools.

9.2 CI

GitHub Actions on every pull request and on `main`, in the `ros:noetic-ros-base` container: the dependencies from the packages' manifests, `catkin_make` and every test with its failures as annotations, the URDF checked, the shell scripts; a second job builds and tests the NeuPAN controller in its environment. The documentation's workflow builds the PDFs and the website from `docs/` and publishes the site on GitHub Pages.

9.3 Versions

One version for the whole stack, in every `package.xml` and in `CHANGELOG.md` (Keep a Changelog, Semantic Versioning).