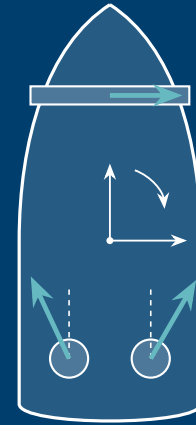


μDP

Portable Marine Control

A propulsion-agnostic manual and autonomous control stack for offshore vessel models on ArduPilot.
One operator interface, one autopilot, any hull.



Pierre-Louis Constant

Director / Lead Engineer, Oceanomatics Pty Ltd

Firmware baseline: ArduPilot Rover 4.6

Reviewed with each ArduPilot release

Abstract

μDP (micro dynamic positioning) is Oceanomatics' control stack for offshore vessel models: dynamic-positioning functionality, manual to fully autonomous, on hulls of one to two metres. It develops the control scheme designed in 2019 for a model offshore support vessel, in which a hobby radio mixed the sticks into two azimuth thrusters and a bow tunnel. The stick semantics of that scheme are retained; the mixing is replaced by a force allocator built into ArduPilot's motor library and driven by a table of effectors, so that one method covers propellers and rudders, azimuths, catamarans, water jets and Voith Schneider propellers. The stack adds RTK positioning with GNSS heading, wind feed-forward for station and track keeping, a current estimate, a vessel model identified online and persisted between sessions, a configuration tool that generates every input of the stack from a hull outline and a thruster list, and a simulation chain that runs the trial protocol before wet trials.

Contents

1	Introduction	3
2	The paradigm, and what changes	3
2.1	The 2019 approach	3
2.2	From expected outputs to equations	5
2.3	What changes	5
3	Vessel and effector models	7
3.1	Frames and notation	7
3.2	Effector types	8
3.3	The configuration matrix	12
3.4	Example configurations	12
4	Thrust allocation	13
4.1	Problem statement	13
4.2	Null space and the bias mode	13
4.3	Saturation and priority	14
4.4	Steerable effectors: flip, sector, slew, blanking	14
4.5	Speed scheduling of tunnel thrusters	15
4.6	Pivot point	16
4.7	Attainable force envelopes	16
5	Operator interface	17
5.1	Inputs, identical for every hull	17
5.2	Open-loop and rate-assisted manual	17
5.3	Idle, bias, arming	18
5.4	Radio and console hardware	18
6	ArduPilot integration	18
6.1	What Rover offers and what it lacks	18
6.2	Architecture	20
6.3	Where the demand comes from	20
6.4	Why not the scripting engine	21
6.5	The marine frame	21
6.6	Configurable layouts: effectors, holding devices, postures	22
6.7	Actuators	23
6.8	Sensors	24
6.9	Power, wiring, ingress	24
6.10	Configuration summary	25
7	Autonomous operation	25
7.1	Mode mapping	25
7.2	Station keeping	25
7.3	Ground station operation	26
7.4	Companion computer and perception	26
7.5	How far can a model go	27

8	Disturbances and a self-correcting model	27
8.1	Model with disturbances	27
8.2	Wind feed-forward for station keeping and track keeping	28
8.3	Current as a state of the stack	29
8.4	A self-correcting hull and effector model	29
8.5	Persistence	30
8.6	Where it lives	30
9	Verification	31
9.1	From paradigms to test cases	31
9.2	Software in the loop	31
9.3	Simulation levels	32
9.4	Gazebo: hull, effectors and hydrodynamics	33
9.5	ROS 2: frames, bridge and test harness	33
9.6	The configuration app	36
9.7	Bench identification	37
9.8	Wet trials	37
10	Roadmap	37
11	Conclusion	37
A	The 2019 equation set, for the record	40
B	Marine frame implementation outline	41
C	SITL physics model	46
D	Parameter reference	48
E	Indicative hardware	49

1 Introduction

In 2019 Oceanomatics designed the control system of a model offshore support vessel: two azimuth thrusters aft, one tunnel thruster forward, and a hobby radio as the only processor on board. The stick mixing was implemented in the radio and documented in [1].

Since then the requirement has widened to other hull and propulsion types, which radio-side mixing cannot cover: a more general and flexible control software is needed.

The tooling has also changed: a small boat can hold station on RTK GNSS with off-the-shelf firmware, and the radio is one of several possible operator interfaces. The new project is called μDP, for *micro dynamic positioning*: DP functionality, manual to fully autonomous, on hulls of one to two metres.

This paper:

1. restates the 2019 approach and identifies what is kept and what is replaced (Section 2);
2. generalises it: any layout of fixed thrusters, azimuths, propeller and rudder pairs, jets or Voith Schneider propellers is described by a table of effectors, and the configuration matrix and the allocation follow from it (Sections 3 and 4);
3. applies it to ArduPilot: manual, rate-assisted and autonomous operation, an implementation outline for the frame, disturbance feed-forward and a self-correcting model, a configuration tool that generates the stack's inputs from a hull outline and a thruster list, a simulation chain and a roadmap (Sections 5 to 10).

Intended readers are engineers and advanced builders familiar with vector notation and autopilot configuration, not necessarily with marine control allocation. Derivations are kept short; the standard references are [2, 3].

The methods are tested on model boats, but they are written for full-scale unmanned surface vessels as well. Allocation, disturbance handling, model identification and the configuration tool contain nothing that depends on hull scale; only the parameter values do.

The one-line summary

Retain the 2019 stick semantics (surge, sway, yaw, pivot point, bias, idle). Replace radio-side mixing by a force allocator that knows each thruster's position and limits. Use ArduPilot for modes, estimation and failsafes. Verify in simulation before wet trials.

2 The paradigm, and what changes

2.1 The 2019 approach

The 2019 study [1] covered one hull with a programmable radio and its simulator. Its six-step method is retained unchanged; the mixing it produced is not.

1. **Inputs.** Define what the operator commands: surge, sway, yaw, a pivot point, and an idle/bias switch. The set was taken from officers experienced with this vessel type and matches a commercial DP joystick.
2. **Outputs.** Name and number every actuator.
3. **Paradigms.** Sketch each manoeuvre (Figure 1): who thrusts where, in which direction, at what proportion.
4. **Equations.** Reverse the perspective: for each output, read across the paradigms and write it as a sum of curves of the inputs (Section 2.2).
5. **Programming.** Enter the equations on the target.
6. **Verification.** Check every paradigm in a simulator before touching hardware.

Three elements are retained as they were: the *pivot-point* input on the second stick, which selects the centre of rotation; the *bias mode*, azimuth pods idling against each other for immediate

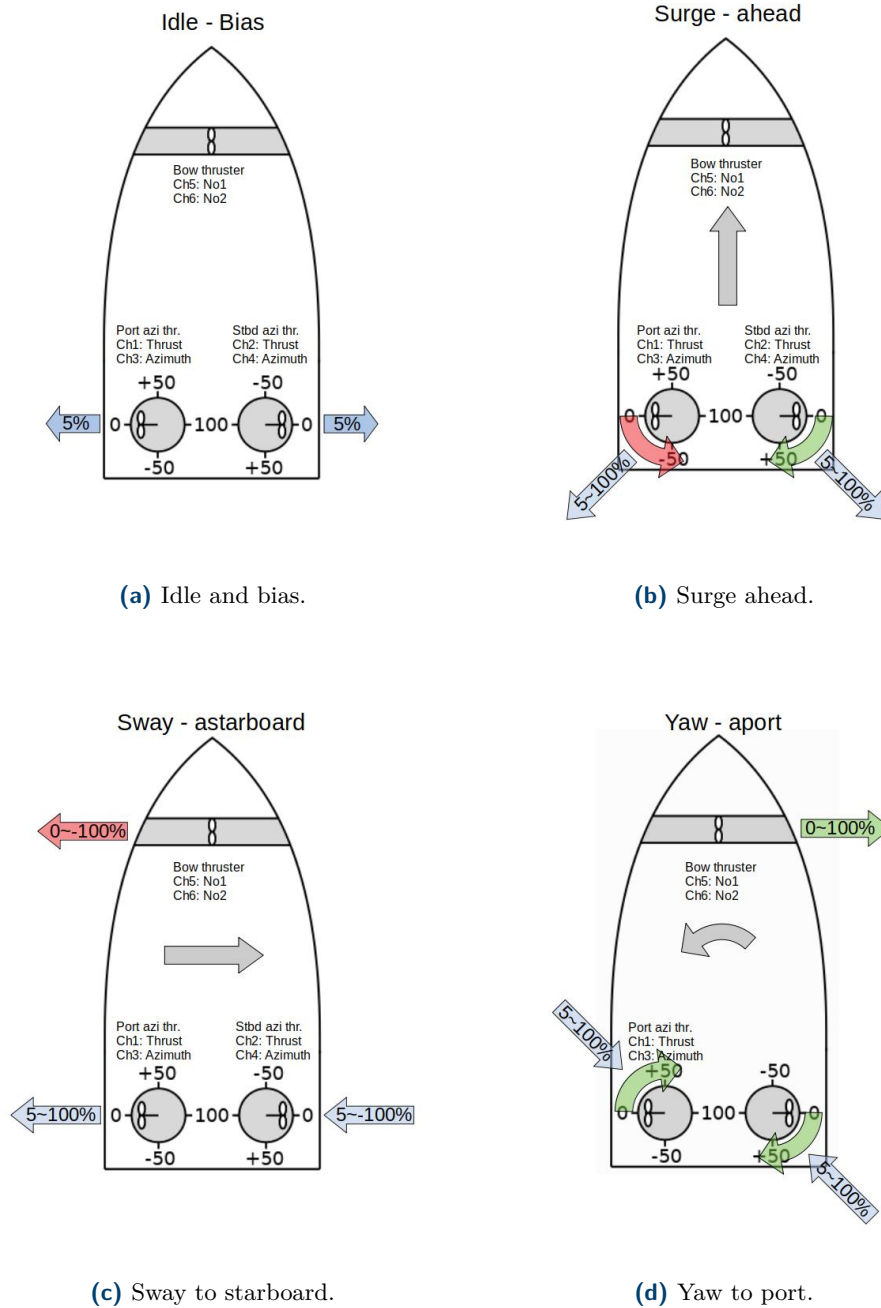


Figure 1: The qualitative manoeuvring paradigms of 2019. Green arrows are azimuth rotations, blue arrows are thrust demands. They are reused unchanged as acceptance tests (Section 9).

thrust response, kept as an option that is off by default on electric models (Section 4.2); and the *tunnel washout*, the loss of tunnel-thruster effectiveness with forward speed.

2.2 From expected outputs to equations

Step four carries the method and is independent of the target. The paradigms of step three are stated from the input side (surge stick ahead: both pods rotate aft, both thrusts increase). A mixer, a microcontroller or an autopilot needs the inverse: one expression per output.

The inversion is done with a *relational table*: outputs in rows, inputs in columns, each cell a curve from a small library ($f_{|\cdot|}(x) = |x|$, $f_+(x) = x$, $f_-(x) = -x$, a scaled line kx , the ramp $\min(x + 1, 1)$). A row gives the equation of an output; a column gives the paradigm of an input.

Table 1 reproduces the 2019 table in the notation of this paper, curves included. Each cell was entered as a curve in the transmitter’s editor; the graphical form makes a wrong sign or slope visible immediately.

Written out, every output has the same shape,

$$O_j = \sigma \left(\sum_i k_{ji} f_{ji}(I_i) + a_j \right) + (1 - \sigma) c_j, \quad (1)$$

a *matrix of curves* f_{ji} with gains k_{ji} , a bias a_j , the idle switch σ and a parking constant c_j .

The curves are structural; the gains are not. A cell’s shape encodes the paradigm: $|x|$ (thrust rises for either stick direction), x and $-x$ (two units oppose), $\min(x + 1, 1)$ (the pivot input only removes authority). These shapes hold for any hull with this layout. The gains k_s , k_n , the bias thrust t_0 and the parking angle C_0 are properties of one hull and its actuators.

The gains are semi-constants. The 2019 document printed $k_s = \frac{1}{2}$ and $k_n = \frac{1}{4}$; these were initial values. Each gain was assigned to a transmitter potentiometer as a mixer global variable and adjusted during trials. They are therefore written as symbols with initial values throughout this paper.

The per-movement weights w_{surge} , w_{sway} , w_{yaw} , set to one in 2019, are treated the same way.

Mode logic is arithmetic: the factor σ and the term $(1 - \sigma)c_j$ implement the idle switch in a mixer that has no branching, and any target can evaluate the expression.

Because the table is written per output it is independent of the target: the same table maps to mixer curves, microcontroller code or autopilot parameters.

2.3 What changes

The relational table has no knowledge of geometry. Its gains are found by trial, and combined demands add in the wrong proportions. At the initial gain $k_s = \frac{1}{2}$, surge by symmetric azimuth rotation delivered $2s \sin(s\pi/2)$ of longitudinal force for stick s (Figure 2).

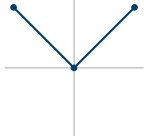
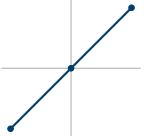
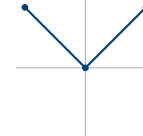
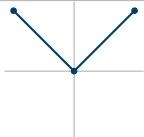
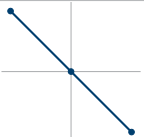
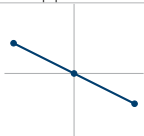
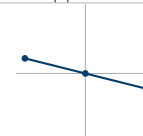
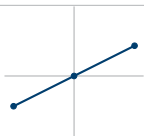
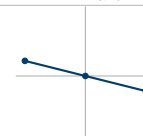
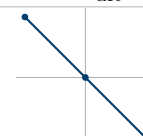
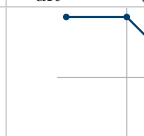
Sway induced a yaw moment unless the bow lever arm was exactly twice the stern lever arm. Per-channel clipping changed the direction of a combined demand as well as its magnitude.

The scheme was open loop; its hardware (OpenTX, Navio2, an Arduino as i-Bus decoder) is obsolete; safety consisted of one switch. The transmitter had been selected for one capability, arithmetic mixing, which no longer needs to reside in the radio (Section 5.4). The 2019 conclusion called for closed-loop control, dynamic positioning and a distributable product, none of which transmitter mixing can provide.

The change is confined to step four. An *effector table* records each thruster’s position and capability, the *configuration matrix* $\mathbf{B}$ follows from the table, and a *force-level allocator* computes each output as its row of $\mathbf{B}_W^\dagger$ applied to the demand $\boldsymbol{\tau}_d$.

The structural content of the curves moves into the geometry: the $|x|$ of the thrust rows and the $\pm k_s x$ of the angle rows are what $T = \|\mathbf{F}\|$ and $\alpha = \text{atan2}(F_y, F_x)$ produce once the pod positions are known.

Table 1: The 2019 relational table, drawn as in the original: each cell is a curve of the input at the head of its column, and each output is the sum of its row, every curve scaled by a semi-constant. Inputs: s surge, y sway, n yaw, p pivot, $\sigma \in \{0, 1\}$ idle/bias switch, t_0 bias thrust, C_0 parking angle. Gains k_s , k_n are drawn at their initial values $\frac{1}{2}$, $\frac{1}{4}$.

Output	Input s (surge)	Input y (sway)	Input n (yaw)	Input p (pivot)	Idle term
T_P port thrust	 $f_{+} = s $	 $f_{+} = y$	 $w_{aft} f_{+} = w_{aft} n $	 $w_{aft} = \min(p+1, 1)$	$+t_0$ all $\times \sigma$
T_S stbd thrust	 $f_{-} = s $	 $f_{-} = -y$	 $w_{aft} f_{-} = w_{aft} n $	 $w_{aft} = \min(p+1, 1)$	$+t_0$ all $\times \sigma$
α_P port angle	 $-k_s s$	not affected	 $-k_n w_{aft} n$	 $w_{aft} = \min(p+1, 1)$	$\times \sigma$ $-C_0(1-\sigma)$
α_S stbd angle	 $+k_s s$	not affected	 $-k_n w_{aft} n$	 $w_{aft} = \min(p+1, 1)$	$\times \sigma$ $+C_0(1-\sigma)$
T_B tunnel thrust	not affected	 $f_{-} = -y$	 $w_{fwd} f_{-} = -w_{fwd} n$	 $w_{fwd} = \min(1-p, 1)$	all $\times \sigma$

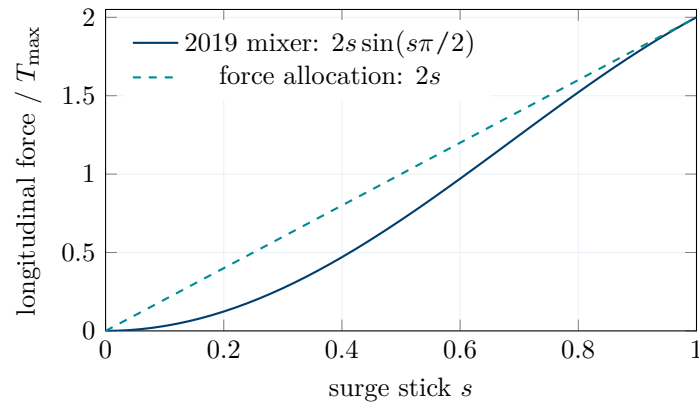


Figure 2: One measurable consequence of percent mixing: longitudinal force delivered by the 2019 surge mix, compared with an allocator that points both pods along the demand.

The semi-constants fall into two groups (Table 2). Those that stood in for geometry disappear, since the matrix computes them. Those that express a preference or an actuator property remain as parameters, editable from the ground station. The other steps keep their role with better tools (Table 3).

Table 2: Where the 2019 semi-constants go.

2019 dial	Initial value	2026
k_s , azimuth angle per unit surge	$\frac{1}{2}$	Computed from geometry by the allocator; no longer a setting.
k_n , azimuth angle per unit yaw	$\frac{1}{4}$	Computed from geometry (a , b in Figure 3); no longer a setting.
w_{aft} , w_{fwd} , pivot weights	$\min(p + 1, 1)$, $\min(1 - p, 1)$	Pivot input p maps to a lateral-force demand (Section 4.6); geometry does the split.
w_{surge} , w_{sway} , w_{yaw} , movement weights	1	MFR_KX, MFR_KY, MFR_KN: full-stick force and moment.
t_0 , bias thrust	0.05	MFR_BIAS, a null-space component (Section 4.2); defaults to zero on an electric model, still a knob on the radio if wanted.
C_0 , parking angle	$\frac{1}{2}$ (aft)	MFx_ANG, sector centre and disarmed position of each steerable effector.
Curve slope and expo	mixer curves	MFR_EXPO thrust shaping, fitted from bollard pull (Section 9.7).
Channel clipping	$\pm 100\%$	Saturation policy MFR_PRI0: uniform or yaw-priority scaling of the demand.

The control software must now cover single hulls with controllable-pitch propellers, rudders and tunnels; single hulls with azimuths; catamarans with distributed thrusters; and craft with steerable jets or Voith Schneider propellers. With mixing removed from the radio, one operator interface, one autopilot stack and one verification chain serve all of them.

A relational table can still be written for any of these hulls and is a useful paper check on an allocator, but each hull would need its own gains; the allocator needs only positions.

Consequences: an operator who can drive the azimuth model can drive the CPP/rudder model and the catamaran without retraining, because the sticks command the same forces; a new hull is a new parameter file, not new code; and every ArduPilot mode that produces surge, sway and yaw demands runs on every hull.

The 2019 equation set is transcribed, with its corrections, in Section A.

3 Vessel and effector models

3.1 Frames and notation

The body-fixed frame follows SNAME convention: x forward, y to starboard, z down, angles positive clockwise seen from above. The horizontal-plane motion is described by the velocity vector $\boldsymbol{\nu} = [u, v, r]^\top$ (surge, sway, yaw rate) and controlled through the generalised force vector

$$\boldsymbol{\tau} = [X, Y, N]^\top, \quad (2)$$

Table 3: The six steps, then and now.

Step	2019	2026
1. Inputs	Sticks and switches of one radio.	Operator demand τ_d in surge, sway, yaw plus pivot and mode, from any source: radio, GCS joystick, autopilot controller, mission. Identical for every hull.
2. Outputs	Six radio channels, named after the thrusters.	Effector table: type, position, sector, thrust limits, slew rate, output slot. Hull-specific, data not code.
3. Paradigms	Prose and sketches per manoeuvre.	Same sketches, now formalised as acceptance tests: pure sway must give $N \approx 0$, and so on.
4. Equations	One arithmetic function per channel.	Configuration matrix $\mathbf{B}$ built from the table; allocation $\mathbf{u} = \mathbf{B}_W^\dagger \boldsymbol{\tau}$ with bias, saturation and steerable-effector handling.
5. Programming	OpenTX mixes and curves.	A frame type of the ArduPilot motor library (Section 6.5), configured by parameters editable from the ground station, like the old transmitter dials.
6. Verification	Transmitter simulator by inspection.	ArduPilot SITL with a 3-DOF physics model, automated tests derived from step 3, then bench identification and wet trials.

the total longitudinal force, transverse force and yaw moment about the centre of gravity (CG). An effector i located at $\mathbf{r}_i = [l_{x,i}, l_{y,i}]^\top$ relative to the CG and producing a horizontal force $\mathbf{F}_i = [F_{x,i}, F_{y,i}]^\top$ contributes

$$\boldsymbol{\tau}_i = \begin{bmatrix} F_{x,i} \\ F_{y,i} \\ l_{x,i}F_{y,i} - l_{y,i}F_{x,i} \end{bmatrix}. \quad (3)$$

The hull responds to the total force according to the 3-DOF manoeuvring model [2],

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau}, \quad (4)$$

with $\mathbf{M}$ the mass and added-mass matrix, $\mathbf{C}(\boldsymbol{\nu})$ the Coriolis and centripetal terms and $\mathbf{D}(\boldsymbol{\nu})$ the linear plus quadratic damping.

The allocator does not use this model; the simulation (Section 9), the disturbance layer (Section 8) and the identifier (Section 8.4) do, all in this form.

Figure 3 shows the geometry of the 2019 vessel. Convention: the 2019 document measured the azimuth angle as the wash direction from the outboard position; here α_i is the direction of the *force* on the hull, measured from the bow, and $\alpha = 0$ (wash aft, push ahead) is the parking position.

3.2 Effector types

Five effector types cover the hulls considered (Table 4). Each is characterised by the number of *extended* control variables it contributes and the constraints on them.

Fixed thrusters contribute one column to the configuration matrix: $\mathbf{b}_i = [\cos \gamma_i, \sin \gamma_i, l_{x,i} \sin \gamma_i - l_{y,i} \cos \gamma_i]^\top$. A bow tunnel has $\gamma = 90^\circ$; a fixed stern propeller has $\gamma = 0$.

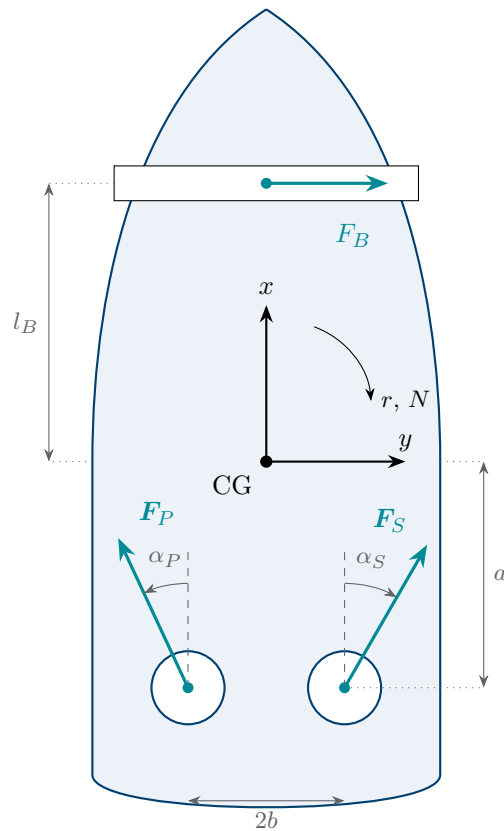


Figure 3: Geometry and sign conventions for the 2019 vessel (example A). Azimuth force angles α are measured from the bow, positive clockwise; $\alpha = 0$ pushes the hull ahead.

Table 4: Effector types, their extended variables and constraints.

Type	Variables	Force model	Constraints
Fixed thruster (tunnel, fixed-axis prop)	T	$\mathbf{F} = T[\cos \gamma, \sin \gamma]^\top$, γ fixed	$-T_r \leq T \leq T_f$; tunnels lose effectiveness with speed; on a catamaran a tunnel washing toward the other hull is ineffective, so $T_r \approx 0$ with γ set to the direction of the force it makes when washing outboard, which is toward the other hull.
Azimuth pod	F_x, F_y	$T = \ \mathbf{F}\ $, $\alpha = \text{atan2}(F_y, F_x)$	$T \leq T_f$ ahead, $T \leq T_r$ reversed; slew $ \dot{\alpha} \leq \dot{\alpha}_{\max}$; force-direction sector where the wash would hit a skeg or the neighbouring pod; $\pm 90^\circ$ servo suffices with reversible props.
Propeller + rudder (open or Kort nozzle; CPP or fixed pitch)	F_x, F_y	$F_x = T$, $F_y = -(k_T \max(T, 0) + k_R \min(T, 0) + k_u u u) \delta$	$ \delta \leq \delta_{\max}$; lateral force follows the flow, not the command: much reduced astern (open rudder next to nothing, Kort nozzle roughly a fifth of ahead, Becker flap rudder more), reversed sign with sternway; effective sector narrow.
Steerable water jet	F_x, F_y	$\mathbf{F} = T_j g(q)[\cos \delta_j, \sin \delta_j]^\top$, bucket q , nozzle δ_j	$ \delta_j \leq \delta_{j,\max} \approx 30^\circ$; reverse via bucket, $g(-1) \approx -0.5$; pump idles with bucket neutral.
Cycloidal propeller (Voith Schneider, VSP)	F_x, F_y	$\mathbf{F}$ in any direction	$\ \mathbf{F}\ \leq T_f$; very fast vectoring, no slew penalty.

Azimuth pods are the reason for the *extended thrust* representation [4, 2]. The physical variables (T, α) enter $\boldsymbol{\tau}$ nonlinearly; the Cartesian components (F_x, F_y) enter linearly. Each pod therefore contributes two columns, $[1, 0, -l_y]^\top$ and $[0, 1, l_x]^\top$, and the physical commands are recovered afterwards.

Real pods have a maximum slew rate (a hobby servo through gearing gives 150–400°/s) and sometimes sectors in which their wash hits the hull, a skeg or the neighbouring pod and the force is lost. When the propeller is reversible there are two solutions for every force, (T, α) and $(-T, \alpha + 180^\circ)$.

Propeller and rudder pairs are the common case for AHTS and PSV models, with the propeller open or in a Kort nozzle and with controllable or fixed pitch; the allocation is the same in all cases.

The rudder is a lifting surface in the propeller race, so its side force is roughly the deflection times the sum of a thrust term and a speed term. At rest an open rudder has almost no authority unless the propeller runs ahead; a rudder behind a Kort nozzle keeps a fraction of its authority astern, and a Becker flap rudder keeps more.

No controllable-pitch propeller is available at this scale. A CPP-driven hull is modelled with fixed-pitch propellers under RPM control; the allocation is unchanged, since pitch and RPM are both thrust commands, and only the thrust-versus-command curve (Section 9.7) differs.

In reverse the pair behaves the opposite way to a thruster. A thruster’s lateral force follows its command; a rudder’s follows the water flowing over it. With the propeller astern the race no longer strikes the rudder. Behind an open propeller the side force then comes only from sternway, grows with the square of that speed, and has the opposite sign to ahead for the same rudder angle; at rest going astern there is none.

A Kort nozzle draws its inflow forward over the rudder, so some authority remains astern even at rest: as a working figure, about a fifth of the ahead value, with the sign of sternway. A Becker flap rudder, whose flap roughly doubles the lift of the blade, keeps considerably more.

The k_R term carries the astern authority and the $k_u u|u|$ term the sternway, and the allocator has to respect both (Section 4.4). Both gains are measured at the bench (Section 9.7).

Following [6], we treat the pair as an azimuth-like effector with a narrow sector $|\alpha| \leq \delta_{\max}^{\text{eff}}$, at full width only for $F_x > 0$ at rest and shrunk to what k_R leaves for $F_x < 0$. The $k_u u|u|$ term extends that authority, with the opposite sign, once the hull has way on, and Section 4.4 says how the frame uses it.

Sway on such a hull is the twin-screw walking manoeuvre. One propeller goes ahead with its rudder over, and the race deflected by that rudder is what makes the side force at the stern. The other propeller goes astern to cancel the surge, and its rudder contributes little, only what the nozzle inflow gives it. The bow thruster supplies the side force forward and balances the yaw.

A linear allocator cannot produce this on its own, because at rest both rudders have little or no gain until a propeller is running ahead; Section 4.4 adds a rule for it.

Steerable water jets behave like a sector-limited azimuth with an asymmetric magnitude. The nozzle steers $\pm 30^\circ$ or so, the reversing bucket gives astern thrust at roughly half efficiency, and the jet holds zero net thrust with the pump running. The last property is the jet equivalent of the bias mode, and Section 4.2 applies.

Cycloidal propellers have the Voith Schneider propeller (VSP) as their reference design. They vector thrust in any direction within a fraction of a second and are the easiest case: two unconstrained extended variables per unit.

Figure 4 shows each type as model-scale hardware. The 2019 hull used a Bauer-Modelle Schottel 40 mm azimuth drive.



(a) Azimuth: Schottel 40 mm rudder propeller.



(b) Azimuth: Schottel 47BM-Z with motor.



(c) Fixed: 33 mm tunnel thruster with motor.



(d) Propeller + rudder: Becker flap rudder 70 mm.



(e) Steerable jet: Jet 19 BM with reversing bucket.



(f) Cycloidal: Voith Schneider eVSP65BE.

Figure 4: The five effector types as model-scale hardware. Photographs © Bauer-Modelle, Eibenstock, Germany (<https://www.bauer-modelle.com>), reproduced with the kind permission of Uwe Bauer; see [figures/bauer/SOURCES.md](#) for product pages.

3.3 The configuration matrix

Stacking all extended variables in $\mathbf{u} \in \mathbb{R}^m$ gives the linear relation

$$\boldsymbol{\tau} = \mathbf{B}\mathbf{u}, \quad \mathbf{B} \in \mathbb{R}^{3 \times m}, \quad (5)$$

where $\mathbf{B}$ is built column by column from the effector table. The vessel is *over-actuated* whenever $m > 3$ and $\mathbf{B}$ has full row rank, as it does for every layout here. Over-actuation is what allows bias modes, tolerance to a failed thruster and energy optimisation.

3.4 Example configurations

Seven layouts are used throughout the paper. The geometry is that of a 1.6 m model, with placeholder values to be replaced by measurements.

A: 2019 vessel. Two azimuths at $(-a, \mp b)$, one bow tunnel at $(l_B, 0)$ (the two physical tunnels of the 2019 hull may be driven as one or listed separately).

With $\mathbf{u} = [F_{x,P}, F_{y,P}, F_{x,S}, F_{y,S}, F_B]^\top$,

$$\mathbf{B}_A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ b & -a & -b & -a & l_B \end{bmatrix}, \quad a = 0.65, b = 0.15, l_B = 0.60. \quad (6)$$

B: single hull, twin CPP with rudders, bow tunnel. Two propeller-rudder pairs at $(-a, \mp b)$ and a tunnel. Same shape as $\mathbf{B}_A$, but the F_y columns of the pairs carry the sector and sign constraints of Table 4. Pure sway at rest comes from the tunnel and from the split of Section 4.4, one propeller ahead with rudder and the other astern. It is small, and the allocator has to be told that yaw has priority.

C: catamaran, two azimuths, one tunnel per hull. Two azimuths at $(-a, \mp c)$, one under each hull, and one tunnel per hull at $(l_B, \mp c)$. The wide spacing gives large yaw authority from differential thrust.

The catamaran has one rule of its own, and it concerns the tunnels. A bow thruster that discharges inboard, toward the other hull, is highly ineffective, because its flow works against that hull. Each tunnel is therefore useful only when it washes outboard, which makes it one-directional in practice: the port tunnel pushes the bow to starboard, the starboard tunnel pushes it to port.

We enter each as a fixed thruster whose force direction points inboard, the direction it pushes when it washes outboard, with an astern limit of zero or nearly so. For a given sway direction only one tunnel is working, which Figure 6 shows. The pods are unaffected.

With $\mathbf{u} = [F_{x,P}, F_{y,P}, F_{x,S}, F_{y,S}, T_{B,P}, T_{B,S}]^\top$, the port tunnel pushing to starboard, the starboard tunnel to port and both $T_B \geq 0$,

$$\mathbf{B}_C = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & -1 \\ c & -a & -c & -a & l_B & -l_B \end{bmatrix}, \quad a = 0.60, c = 0.35, l_B = 0.55. \quad (7)$$

D: twin reversible steerable jets, bow tunnel. Same shape as $\mathbf{B}_A$; the jets' F_y columns are limited to $|F_y| \leq F_x \tan \delta_{j,\max}$ with $F_x \geq 0$ ahead, and astern thrust is limited to $|g(-1)| T_j$. The bias analogue is a nonzero pump speed with neutral bucket, which the allocator treats as a minimum T_j rather than as a force.

E: twin cycloidal propellers aft, bow tunnel. Same shape as B_A , with the two pod columns unconstrained in direction and $\|F\| \leq T_f$. For the allocator it is the simplest layout: the sector, flip and slew steps of Section 4.4 are skipped and blanking never triggers.

F: twin CPP aft, retractable azimuth forward, bow tunnel. Layout B with a retractable azimuth added near the bow. Deployed, it is an azimuth column pair like any other, with a force sector that keeps its wash out of the hull; stowed, its columns are dropped and the hull is layout B again. The change of posture is the new element, and Section 6.6 deals with it.

G: jack-up barge, four retractable azimuths, four spud legs. Four azimuth pairs at the corners give a matrix with eight columns and a large null space, so the bias pattern is a projection rather than a pair formula. The spud legs are not effectors: when set they hold the barge and the allocation stops. This layout motivates the holding devices and postures of Section 6.6.

4 Thrust allocation

4.1 Problem statement

Given a demand τ_d , find effector commands that produce it within the thrust limits, sector limits and slew rates, for as little power as possible. In the extended variables the unconstrained version is a weighted least-squares problem,

$$\min_u \frac{1}{2} u^\top W u \quad \text{s.t.} \quad B u = \tau_d, \quad (8)$$

whose solution is the weighted pseudo-inverse

$$u^* = W^{-1} B^\top (B W^{-1} B^\top)^{-1} \tau_d = B_W^\dagger \tau_d. \quad (9)$$

The 3×3 matrix $B W^{-1} B^\top$ inverts in closed form, so the cost in the main loop of a flight controller is negligible.

The diagonal weight W expresses preferences: a heavy weight on the tunnel columns makes the allocator use the pods first. An effectiveness factor on a column of B , scheduled on speed, gives the washout of Section 4.5.

Constraints (limits, sectors, slew) are handled around the pseudo-inverse in the steps below. Among the methods surveyed in [3] this is the simplest: exact without saturation, cheap and predictable. A quadratic programme with explicit constraints [5] is planned once the frame exists (Section 6.5).

4.2 Null space and the bias mode

Because $m > 3$, B has a null space of dimension $m - 3$, and any vector u_0 in it can be added to u^* without changing the produced force:

$$u = B_W^\dagger \tau_d + (I - B_W^\dagger B) u_0. \quad (10)$$

For a symmetric azimuth pair at the same longitudinal position, the vector that has each pod push toward the centreline with equal magnitude β , wash outboard as on the real ship,

$$u_0 = \beta [0, +1, 0, -1, 0]^\top, \quad B_A u_0 = \begin{bmatrix} 0 \\ \beta - \beta \\ -a\beta + a\beta \end{bmatrix} = \mathbf{0}, \quad (11)$$

is the 2019 bias mode: pods pointing outboard, thrusts cancelling. In 2019 it was taken from bridge practice; here it is a null-space vector of $\mathbf{B}$. The magnitude β is a parameter.

The same reasoning gives the bias for a catamaran, where the two tunnels, each washing outboard, can also run against each other. It also shows that a CPP/rudder hull has no force-free bias, only a minimum-pitch or minimum-RPM idle.

The bias originates in diesel-mechanical propulsion on full-size vessels. An engine cannot change speed quickly and a controllable-pitch propeller must stay loaded, so opposed pods provide a fast change of net thrust from thrust already present.

On a small electric boat these constraints do not exist. Motor torque is available in milliseconds, the propeller spins up in a fraction of a second, and the hull responds far more slowly than either; the time saved by a pre-loaded thruster is not measurable at the hull. The cost of bias is battery, noise and thrust headroom.

Bias is therefore retained as a null-space capability with a default of zero. It is enabled for demonstration and training, where pods visibly idling outboard is expected, or where a drivetrain benefits from staying loaded (Section 6.7).

4.3 Saturation and priority

One rule comes before any scaling. A one-directional effector, a catamaran tunnel with $T_r = 0$ or a propeller that cannot reverse, may be asked by the pseudo-inverse for thrust in the direction it does not have.

Its column is then removed and the allocation is solved again without it: one step of an active-set method, at most one step per such effector. Clipping the command instead would leave an unbalanced moment and turn the produced force away from the demand, the same defect as the 2019 clipping, and scaling against a zero limit would zero the whole demand.

When an effector exceeds its limit after allocation, the demand is scaled, not the outputs. The limit is that of the direction commanded, T_f ahead or T_r astern. Two policies are provided.

- *Uniform*: $\tau_d \leftarrow \tau_d / \rho$ with $\rho = \max_i \|\mathbf{F}_i\| / T_{\text{lim},i}$. The produced force keeps the direction requested and is reduced in magnitude; this removes the 2019 clipping defect.
- *Yaw priority*: scale the translational demand (X, Y) first and preserve N as long as possible. Loss of heading is more serious than loss of speed for both operator and position controller; this is the usual DP choice [7].

4.4 Steerable effectors: flip, sector, slew, blanking

Converting (F_x, F_y) back into (T, α) for an azimuth, rudder or jet takes five decisions, in this order.

1. *Force sector*. Before anything else, if the force direction lies in a sector where the wash would hit the hull, the force is clamped to the sector boundary with only its useful component kept, $T \leftarrow T \cos(\alpha - \alpha_{\text{clamp}})$, or scaled by the effectiveness parameter of that sector, for instance where the wash would strike a skeg or the neighbouring pod. This is checked on the force direction, not on the pod angle, because reversing the propeller sends the wash the same way.
2. *Flip*. For a pod with a reversible propeller, choose between (T, α) and $(-T, \alpha + 180^\circ)$ the one closest to the current angle. This halves the worst-case azimuth travel. A standard 180° servo ($\pm 90^\circ$) with a reversible ESC then covers every force direction. The 2019 hull had its Schottel-type drives modified for 360° rotation; that becomes optional, at the cost of lower reverse-thrust efficiency (Section 6.7).
3. *Mechanical sector*. After the flip, if the pod angle lies outside the servo's range, clamp it to the nearest boundary and keep only the component of the force along the clamped direction. For a rudder this reproduces the limited side force per unit thrust the pair can give.

4. *Slew.* Limit the angle step per cycle to $\dot{\alpha}_{\max}\Delta t$. Servos have finite speed, and steppers or serial-bus servos can be commanded at a known rate.
5. *Blanking.* While the actual angle is far ($> 45^\circ$) from the commanded one, reduce the thrust by a factor κ so that the pod does not push in the wrong direction during the swing, as an azimuth operator does manually.

Without angle feedback (hobby servos) the actual angle is assumed to follow the slew-limited command; serial-bus servos with position feedback remove the assumption (Section 6.7).

A propeller and rudder pair is handled differently. Reversing the propeller does not rotate its force vector: F_x flips, and F_y follows the flow over the rudder, $F_y = -(k_T \max(T, 0) + k_R \min(T, 0) + k_u u|u|) \delta$. The frame therefore computes the lateral gain $g_y = -(k_T \max(T, 0) + k_R \min(T, 0) + k_u u|u|)$ every cycle from the commanded thrust and the water-relative speed, and commands $\delta = F_y/g_y$ within the rudder range.

When $|g_y|$ falls below a threshold the pair's lateral column is scaled to zero, and sway moves to the tunnels. Going astern with sternway, g_y changes sign and the rudder command inverts on its own.

The inversion is invisible to the operator. Under the 2019 mixing, a sway demand astern would have set the rudder the wrong way.

A further rule applies to a propeller pair at low speed: the pseudo-inverse cannot create rudder authority that does not yet exist. When the sway demand exceeds what the tunnels can give and the water-relative speed is below a threshold, the frame splits the propellers before allocating.

It commands a split thrust T_s ahead on one side and the astern thrust that cancels it on the other, with T_s proportional to the sway demand up to the ahead limit.

The ahead rudder now has gain $k_T T_s$, the astern rudder only the reduced $k_R T_s$ of the nozzle inflow, with the opposite sign, and the couple $2bT_s$ that the split produces is subtracted from the demand as a known term.

The rest of the allocation is then linear as before. The ahead rudder takes the stern side force, the bow thruster takes the forward side force and the yaw balance, and the astern propeller's rudder takes the small share its k_R allows, with the sign the frame knows. The side that runs ahead is chosen so that the couple assists the yaw balance.

4.5 Speed scheduling of tunnel thrusters

Tunnel thrust deflects aft as speed increases. In 2019 this was left to the operator, who stopped using the bow thruster by pushing both sticks forward. Here the tunnel's column of $\mathbf{B}$ is scaled by an effectiveness factor scheduled on the surge speed through the water,

$$w_B(u_r) = \max(0, 1 - |u_r|/U_t), \quad u_r = u - [\mathbf{R}(\psi)^\top \mathbf{V}_c]_x, \quad (12)$$

with U_t the speed at which the tunnel stops being useful, typically 0.5–1 m/s at model scale, to be measured.

The speed is the water-relative one, because the wash deflects with the flow over the hull and not with motion over the ground: a hull holding station in a current has its tunnel washed out just the same. The estimator's ground speed serves until the current estimate $\mathbf{V}_c$ of Section 8.3 exists.

As $w_B \rightarrow 0$ the pseudo-inverse assigns no thrust to the tunnel and moves sway and yaw to the stern effectors. On a CPP/rudder hull the same mechanism can *increase* the rudder authority with speed, since $k_u u_r |u_r|$ grows.

4.6 Pivot point

Rotating a rigid body about a point P at longitudinal position x_p on the centreline means the CG has sway velocity

$$v = -x_p r, \quad (13)$$

as Figure 5 shows: turning to starboard about the bow ($x_p > 0$, $r > 0$) moves the CG to port. The pivot input therefore has two forms.

- *Closed loop* (rate-assisted manual, station keeping): command the pair $(v_d, r_d) = (-x_p r_d, r_d)$ and let the sway and yaw controllers produce Y and N . The pivot point is then honoured regardless of hydrodynamics.
- *Open loop* (direct manual): apply the yaw-generating lateral force at the end opposite the pivot. With the pivot stick $p \in [-1, 1]$ (+1 = bow) this is $Y_d = -p(2/L)N_d$, which puts the line of action of the resultant at $x_f = N_d/Y_d = -L/(2p)$: at the stern for $p = +1$, where the allocator finds the stern effectors, at the bow for $p = -1$, where it finds the tunnel, and at infinity as $p \rightarrow 0$, a pure couple. The 2019 weights $w_{\text{aft}}, w_{\text{fwd}}$ approximated this without knowing the geometry.

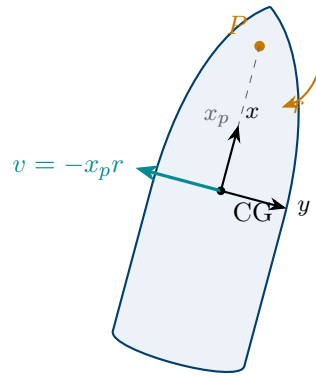


Figure 5: Pivot-point kinematics. A rotation r about P is a rotation about the CG plus a sway velocity $v = -x_p r$ at the CG.

4.7 Attainable force envelopes

The configuration matrix also gives the set of forces the hull can produce, which shows before construction whether a layout can hold station in the intended conditions. Figure 6 shows it for layouts A and C in surge and sway at zero yaw moment.

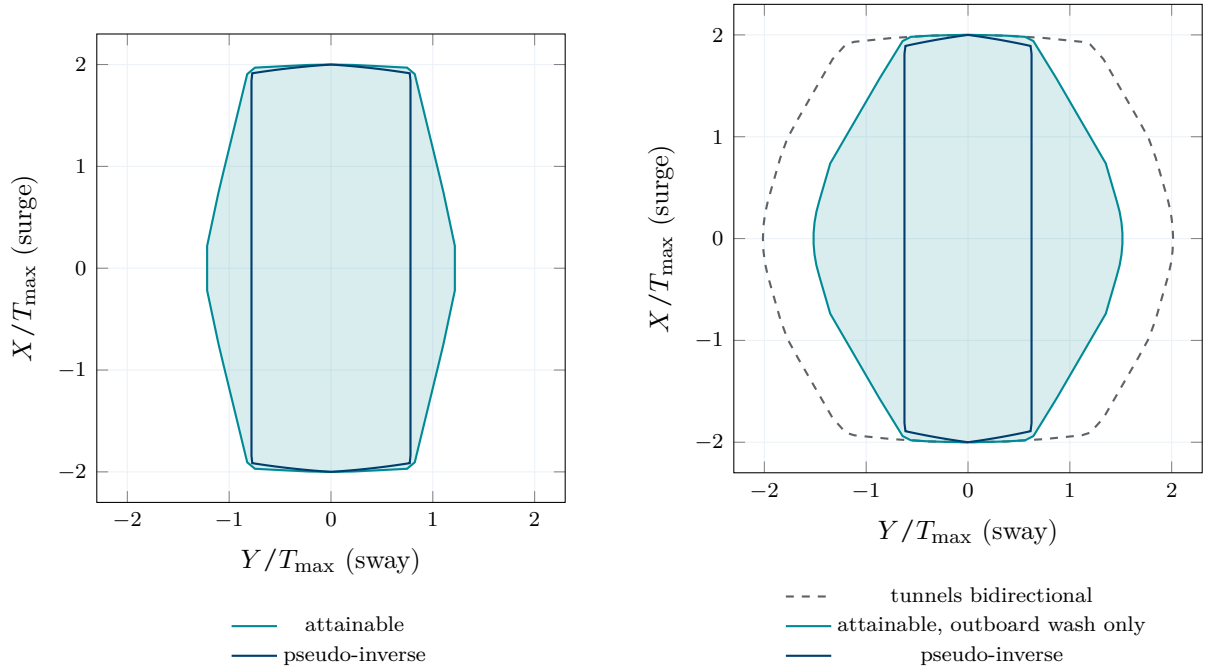
Two curves are drawn: the set the pseudo-inverse allocator reaches with uniform scaling, and the true attainable set from exhaustive sampling of the effector commands. The true set is convex, a slice of a sum of discs and segments, so its boundary is the convex hull of the samples.

The difference is the cost of the closed-form allocator. It is nil along the surge axis, where both pods point along the demand, and largest wherever the moment balance calls for opposed pod thrusts: on layout A the pseudo-inverse reaches $0.78 T_{\text{max}}$ on the sway axis against $1.22 T_{\text{max}}$ attainable, because it never sets one pod ahead and the other astern to cancel the tunnel's moment.

A constrained solver recovers the difference (Section 6.5). On both hulls sway is limited by the bow tunnel; full-size OSVs carry two or three for this reason.

For the catamaran the figure also shows the effect of the tunnel rule. With each tunnel effective only when washing outboard, one tunnel serves each sway direction instead of two, and the attainable set narrows across the beam by about a quarter while surge and yaw are untouched.

The pseudo-inverse curve is drawn with the rule of Section 4.3: the tunnel asked for thrust it does not have is dropped and the allocation re-solved, which leaves it at $0.62 T_{\max}$ on the sway axis, against $1.52 T_{\max}$ attainable.



(a) Layout A: two azimuths, one tunnel ($0.4 T_{\max}$).

(b) Layout C: catamaran, two azimuths, two tunnels ($0.3 T_{\max}$ each), with and without the rule that a tunnel washing toward the other hull is ineffective.

Figure 6: Attainable surge/sway force at $N = 0$, normalised by the thrust of one azimuth. Computed with `scripts/envelope.py`.

5 Operator interface

5.1 Inputs, identical for every hull

Table 5 defines the operator interface: the 2019 assignment with the idle switch replaced by arming. It is identical for every layout in Section 3.4.

5.2 Open-loop and rate-assisted manual

A DP joystick offers three levels: manual (the sticks command forces), auto heading (the yaw stick commands a heading rate and heading is held on release) and auto position (position is held and the sticks move the setpoint). ArduPilot Rover provides the same three.

Manual mode. Sticks pass through as τ_d : the 2019 behaviour with correct allocation. Used for bench tests and as the fallback.

Acro mode. The yaw stick commands a yaw *rate* closed on the gyro and the surge stick a *speed* closed on the estimated velocity; heading is held with the stick centred. Sway is open loop in the stock firmware; a sway-velocity loop is a small addition to the mode (Section 6.5).

Loiter mode. Position and heading are held; sticks move the setpoint. This is the DP auto-position mode (Section 7).

The pivot point behaves as in Section 4.6: open-loop weighting in Manual, kinematic command in Acro and Loiter.

Table 5: Operator inputs. Surge on the left stick, as in the 2019 channel list; any handset or console of Section 5.4 maps to the same axes.

Control	Demand	Meaning
Left stick, vertical	X_d (surge)	Ahead / astern force, or speed in rate-assisted modes.
Left stick, horizontal	Y_d (sway)	Force to starboard / port, or sway velocity.
Right stick, horizontal	N_d (yaw)	Moment to starboard / port, or yaw rate.
Right stick, vertical	p (pivot)	Centre of rotation: forward = bow, back = stern, centre = CG. Spring-centred stick or a detented slider.
Two-position switch	Arm / disarm	Replaces the 2019 idle switch. Disarmed: all thrust zero, steerable effectors parked.
Three-position switch	Mode	Manual (open loop) / Acro (rate-assisted) / Loiter (station keeping) or as configured.
Knob	β (bias)	Bias thrust, as the 2019 document intended. Off by default (Section 4.2); demonstration use; no meaning on CPP/rudder hulls.
Momentary switch	Hold	Immediate zero-force “all stop” that keeps the system armed.

5.3 Idle, bias, arming

The 2019 idle switch had two functions, safety and appearance: zero thrust, pods parked aft. Both map onto ArduPilot arming. Disarmed, the frame forces zero thrust and parks each steerable effector at its sector centre, which for an azimuth is $\alpha = 0$: wash aft, ready to push ahead.

Armed, the bias β , if set, is added as a null-space component so that the pods idle against each other; by default it is zero (Section 4.2). Arming is refused if the pre-arm checks on GNSS, estimator, battery or radio fail; the 2019 system had no equivalent.

5.4 Radio and console hardware

The 2019 transmitter was chosen because OpenTX could mix channels arithmetically; no other hobby radio could without an external microcontroller. That requirement no longer exists. In μDP the mixing is in the autopilot, and the operator device has to deliver unmixed axes and switches over a link that is fast, reliable and independent of the companion computer.

The selection criteria are link quality and latency, channel count, telemetry, failsafe behaviour, console ergonomics and openness. Table 6 rates the options [14] against them. OpenTX is discontinued; EdgeTX, its maintained fork, is the current open-source handset firmware. The newer options are not handset firmwares.

Recommendation: an EdgeTX-class handset on ExpressLRS as the manual link, because its failsafe is deterministic and independent of the companion computer; the DP console on top of it, first as Lilaas controls on a BU0836X into the ground station, later as a purpose-built CRSF console where a stand-alone unit is needed. Both produce the same τ_d and setpoints.

The 2019 FrSky Taranis X-Lite runs EdgeTX and accepts a lite-bay ELRS module, so it remains usable.

6 ArduPilot integration

6.1 What Rover offers and what it lacks

ArduPilot Rover in boat class [8] provides the estimation stack (EKF3 on IMU, GNSS, compass and optionally GNSS heading), the mode logic, the heading-rate and speed controllers, a position controller, waypoint navigation, failsafes, logging and telemetry.

Table 6: Operator-device options for μDP. Mixing is no longer a criterion.

Option	Character	Role in μDP
EdgeTX handset with ExpressLRS (open source, active)	16 channels, sub-10 ms latency, kilometres of range, CRSF telemetry with the Yaapu screens, MAVLink pass-through in recent ELRS releases. Mixing and curves still there if ever wanted as a degraded mode.	The safety-critical manual link: proven failsafe, no computer in the path. Default choice for phases 1 to 3.
Proprietary handsets (FrSky Ethos, Jeti, Graupner HoTT, Futaba)	Polished, closed. Jeti and Graupner are the ship-modeller favourites for their programmability and telemetry; that programmability is now redundant.	Acceptable where a handset is already owned; the autopilot accepts S.Bus, F.Port, SRXL2 and i-Bus. Not recommended for a product.
Ground-station joystick over MAVLink (QGroundControl preferred; Mission Planner, Cockpit)	Gamepad, or industrial marine controls read through a USB HID interface: Lilaas three-axis DP joysticks and azimuth levers wired to a Leo Bodnar BU0836X board (12-bit, eight axes, 32 buttons) appear to the ground station as an ordinary joystick, with no driver. MAVLink <code>MANUAL_CONTROL</code> over WiFi, LTE, ELRS MAVLink or a serial radio. Open source; latency and failsafe depend on the telemetry link.	The DP console for phases 3 and 4: real bridge hardware, setpoints, heading wheel, click-to-go, camera, on the same screen. Needs the RC handset as fallback.
Purpose-built bridge console (open hardware)	Microcontroller reading the same Lilaas controls, heading wheel, pivot slider and switches, speaking CRSF to an ELRS transmitter module; no handset firmware at all. CRSF is documented and libraries exist.	The 2019 “Arduino Mega and potentiometers” idea, realised on a modern link. The product form of the operator interface when the console must work without a computer; the BU0836X route covers it when a ground station is present anyway.
Video-centric links (OpenHD and similar)	Digital video with RC and MAVLink multiplexed on one link; open source.	Relevant once onboard video has to reach the operator; not a substitute for the RC link at this stage.

Its motor layer supports four frame families: steering and throttle (a rudder boat), skid steering (two fixed propellers), omni-directional (three or four fixed thrusters with a fixed mixing matrix) and single vectored thrust (one steerable motor). None of these describes an azimuth pod, a propeller and rudder pair with realistic rudder behaviour, a steerable jet or an arbitrary geometry. The allocation layer of Section 4 fills that gap.

6.2 Architecture

Figure 7 shows the stack. Everything above the allocation layer is stock ArduPilot; everything below it is the effector table and the actuators. Operator, ground station and mission are all sources of the same τ_d .

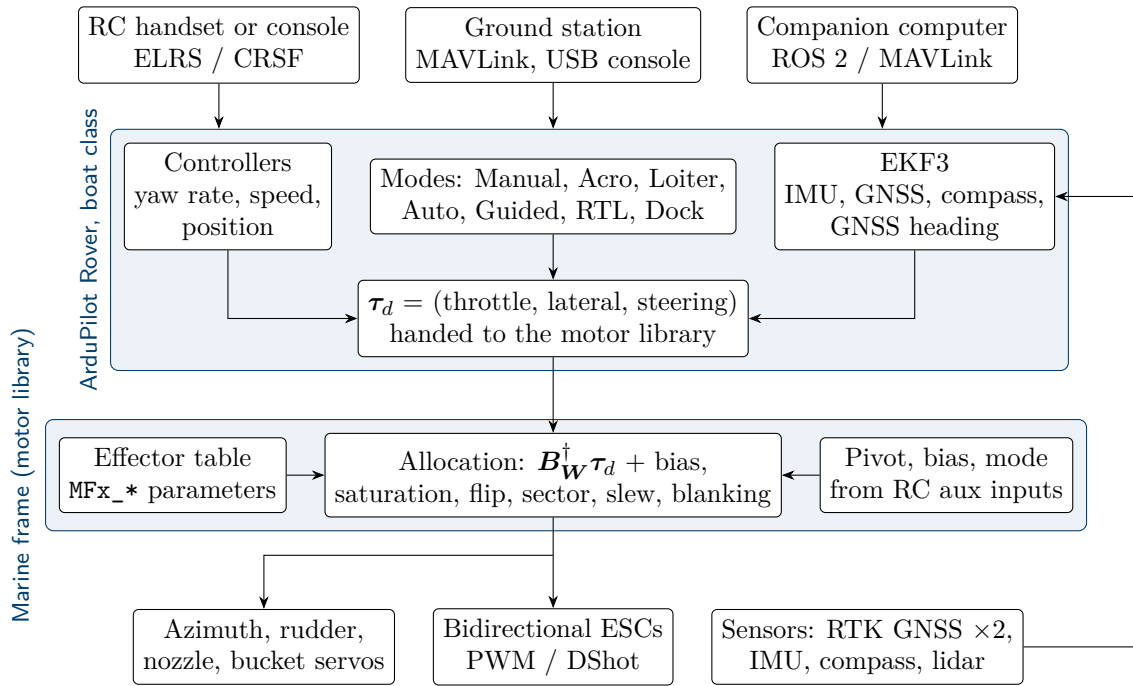


Figure 7: Control stack. The allocation layer is the only hull-specific component, and it is configured, not programmed.

6.3 Where the demand comes from

Every Rover mode ends by handing the motor library three normalised demands: throttle (X_d), steering (N_d) and lateral (Y_d). In Manual they are the sticks; in Acro they are controller outputs; in Loiter, Guided and Auto they come from the position and attitude controllers. The motor library then dispatches to the frame in use.

The marine frame is inserted at that point: it receives the three demands, allocates them across its effectors and writes the outputs. Because the frame reports lateral capability, the position controller and Loiter command sway without other changes to the vehicle code. Nothing above the motor library depends on the hull.

Firmware baseline: ArduPilot Rover 4.6

The motor library, the controllers and the integration points described in this section are those of the ArduPilot Rover 4.6 release series, against which this revision is valid. The document is reviewed periodically as ArduPilot releases evolve, and the integration points, output-function numbering and parameter names are re-confirmed in SITL at each review and before any hull-side testing on a newer branch.

6.4 Why not the scripting engine

ArduPilot includes an embedded scripting engine, and the allocator could be placed there: no firmware build, a text file on the SD card, editable in the field. It is not used in the control path, for structural reasons.

The engine was designed for extensions: telemetry, auxiliary behaviours, mission logic, sensor drivers. It runs in its own thread, below the main control loop in priority, inside a fixed heap and a per-cycle instruction budget. Placed in the only path between autopilot and thrusters, this has four consequences.

- *A script fails silently as far as the control loop is concerned.* A runtime error, memory exhaustion or an exceeded instruction budget stops it; a message goes to the ground station and nothing goes to the failsafe state machine. The vehicle stays armed, the modes keep computing demands, and nothing applies them.
- *Hold cannot stop the ship.* Every safety response of ArduPilot ends in a demand of zero force. With a dead script that demand never reaches an output; only a per-channel output timeout would, and that is a property of how the script wrote the channel, not of the failsafe logic.
- *It sits outside the machinery that makes the rest trustworthy:* no pre-arm validation of the effector table, no reaction to an error after arming, nothing in the flight log unless the script writes it, possible starvation under logging or estimator load, and a missing or corrupted SD card that leaves the vehicle armable with no allocator at all.
- *It cannot be reviewed as firmware:* no compile step, no static typing, no continuous integration against the vehicle code, no upstream review.

The allocator is therefore implemented as a frame type of the motor library. The scripting engine remains available for non-critical functions such as console feedback and is not part of the control design.

6.5 The marine frame

The frame is a new type of `AP_MotorsUGV` (working name *marine*), selected by `FRAME_TYPE` like the existing omni and vectored types. Section B gives an implementation outline in the library's style.

Parameters. One group `MFR_*` of frame settings (stick-to-force gains, bias, blanking factor, tunnel washout speed, saturation policy, thrust shaping, the split rule for propeller pairs) and one group `MFx_*` per effector (type, position, sector centre and width, ahead and astern thrust limits, slew rate, output index, and the three rudder gains for propeller and rudder pairs). The motor library registers them, so they are saved, logged, documented and editable from any ground station like every other parameter. They are the semi-constants of Table 2, and a radio knob can still be mapped to any of them. The names are working names; an upstream submission may put them under the library's own prefix.

Outputs. Thrust goes to the existing `MotorN` output functions; steering angles go to new `AzimuthN` output functions added to the servo library (for a two-pod hull the existing tilt-motor functions serve as an interim). The servo range, trim and reversal parameters, the motor slew-rate limit and the thrust linearisation of the motor library apply downstream unchanged.

Loop. Every main-loop cycle at the vehicle’s scheduler rate: build the columns of B with effectiveness factors scheduled on water-relative speed, allocate with the one-directional rule and saturation handling, add bias if enabled, resolve each steerable effector through flip, sector, slew and blanking, and write. Disarmed: thrust zero, steerable effectors parked at their sector centre.

Pre-arm checks. At least one effector; every enabled effector bound to a thrust output and, if steerable, to an angle output; limits and sectors sane; configuration matrix of full row rank. The vehicle refuses to arm otherwise, with a message that names the offending effector.

Failsafes. Disarming, Hold and every failsafe zero the outputs through the same code path they use for every other vehicle type; the hardware watchdog covers the frame like the rest of the main loop.

Logging. A frame log message per cycle with the demand, the per-effector thrust and angle and the saturation ratio, so that trials can be replayed against the envelope of Section 4.7.

Operator inputs. Pivot point and thrust bias are read through two auxiliary RC input functions and folded into the demand before allocation in open-loop modes, or into the velocity demand in closed-loop modes (Section 4.6).

Effector failure. An effector marked failed (from the ground station or an auxiliary switch) has its columns dropped and the allocation continues with the remaining ones.

Sway loop. A sway-velocity loop for Acro mode, using the body-frame velocity from the estimator, is a small addition to the mode alongside the frame.

Testing. The allocator is a plain C++ class with no hardware dependency, so the acceptance tests of Table 10 are host unit tests in the firmware’s own test suite; the same binary then runs in SITL and on the flight controller. A constrained quadratic-programme allocator replaces the pseudo-inverse in phase 4 behind the same interface.

Upstream. The frame, its documentation and a SITL model are submitted upstream so that the feature is reviewed, tested in continuous integration and carried across firmware releases.

Computing the allocation on the companion computer and sending servo values down was also rejected: it places a general-purpose computer and a data link in the only path to the thrusters.

6.6 Configurable layouts: effectors, holding devices, postures

The allocation core is indifferent to the number and position of effectors; each is one or two columns of B . Retractable azimuths and spud legs (layouts F and G of Section 3.4) add requirements around the core, handled with three tables in the frame and one manager.

Effectors. The table grows to twelve entries, each with up to three outputs (thrust, steer, auxiliary such as a bucket or a deploy actuator) and optional feedback inputs: steering angle from a serial-bus servo, a deployed or stowed limit switch, RPM from ESC telemetry.

Beyond about eight units the outputs move to DroneCAN or a serial servo bus; ArduPilot allows 32 servo channels. The bias becomes a general null-space projection of an “all pods outboard” pattern, so it no longer assumes a symmetric pair.

Deployable effectors. A retractable unit is a state machine: stowed, deploying, active, retracting, failed. While it is not active its columns are dropped, exactly as a failed effector’s are, so the allocator needs nothing new. What is new is the mechanism: the deploy output, the feedback or a timed transition, and the interlocks. No thrust unless deployed, no retraction while thrusting, no deployment above a set speed.

Holding devices. A spud leg, an anchor or a mooring line is not an effector and is not modelled as one. It is a holding device with its own state machine (raised, lowering, set, raising), an actuator and feedback.

Once set it holds the vessel: thrust goes to zero or idle, the position controller stands down, and the vessel is moored. On departure the sequence reverses, with the thrusters holding position before the legs lift. A jack-up that lifts the hull clear of the water is the extreme case, where nothing hydrodynamic applies.

Postures. Deployable units and holding devices give the vessel postures: transit (forward azimuth stowed, tunnels faded out), manoeuvring (everything deployed), moored (devices set, thrust off), jacked. A posture manager sequences the transitions with the interlocks and speed thresholds, driven by an auxiliary switch, the ground station or a mission command. The pre-arm check evaluates the configuration matrix per posture and warns where a posture is under-actuated, as transit is in sway, instead of refusing to arm.

Where it goes. This splits the frame in three: an effector and holding-device manager with the state machines and the outputs, the pure allocator, and the thin glue in the motor library. The split keeps a twelve-effector jack-up and a three-effector tug on one code base. The tables are generated by the configuration tool of Section 9.6, not entered in a parameter editor.

6.7 Actuators

Motors and reduction. The thrust quality of a small azimuth or tunnel unit is determined by the drivetrain more than by the ESC. A 40 to 70 mm model propeller needs roughly 1 000 to 3 000 rpm at full thrust; a direct-drive hobby outrunner on 3S or 4S turns three to ten times faster and operates at manoeuvring thrusts in its worst range: cogging, poor resolution near zero, uncertain starting in reverse.

The recommended drivetrain is a low-Kv brushless motor (100 to 400 rpm/V, gimbal or large-diameter outrunner type) with a 2:1 to 4:1 gear or toothed-belt reduction. The geared and belt-driven Bauer-Modelle azimuth units (Figure 4) are built this way.

The motor then runs fast enough for clean sensorless commutation while the propeller turns slowly; the reduction multiplies torque, so reversal is sharp and low thrust repeatable; the low-Kv motor is quieter and draws less current at equal shaft power; the belt isolates it from propeller shock loads.

Thrust resolution near zero, where station keeping operates, improves by the reduction ratio. A sensed or field-oriented ESC is then a refinement, not a requirement.

ESCs. Bidirectional ESCs, as recommended in 2019, remain the choice: brushed bidirectional units for brushed motors, brushless units in 3D mode otherwise. Reverse thrust of a fixed-pitch propeller is typically 50–70 % of ahead thrust and is entered as `MFx_TR`.

DShot with bidirectional telemetry gives RPM feedback and therefore a thrust estimate; PWM at 50–400 Hz is sufficient otherwise. With bias off by default (Section 4.2) no unit has to hold a few per cent of thrust for long; if bias is used for demonstration, the reduction drivetrain makes it clean.

Azimuth actuators. In increasing order of capability: a 180° hobby servo through gearing, relying on the flip logic of Section 4.4; a multi-turn serial-bus servo with position feedback, continuous rotation and a known slew rate, driven from an autopilot serial port; or a small stepper or gimbal motor with a magnetic encoder behind a microcontroller speaking a simple serial protocol. With feedback the blanking logic uses the measured angle, and a jammed pod is detected.

Rudders, nozzles, buckets. Ordinary servos with `SERVOx_MIN/MAX/TRIM` calibrated so that the frame's ± 1 maps to the mechanical range. Water-jet buckets have a strongly nonlinear thrust-versus-position curve that should be measured and stored as a short table.

6.8 Sensors

Station keeping at model scale needs decimetre position and degree-level heading at zero speed. Position comes from an RTK GNSS receiver (u-blox F9P class) with a fixed base station or an NTRIP correction service.

For heading, a well-calibrated compass is the baseline. Mounted high, away from the ESCs and motor leads, with the autopilot's calibration and motor-interference compensation applied, it gives heading at zero speed to a few degrees.

That is enough for transit, early trials and manual work. It is not enough for station keeping, which needs the degree-level heading stated above: the compass alone serves phase 2, and from Loiter onward the second receiver below is required.

A moving baseline is better. A second RTK receiver and antenna on the centreline let the autopilot derive heading from the baseline between the two antennas, unaffected by motor currents, ballast or the hull's own field, and fused with the compass rather than replacing it. A hull that must hold station and dock gets the second receiver; for transit only, the compass is acceptable.

At RTK precision the attitude estimate also serves as the vertical reference for the antennas. An antenna sits on a mast above the centre of gravity, and every roll and pitch swings it sideways by the mast height times the sine of the angle. On a 1.6 m model with the antenna 0.4 m up, 10° of roll moves it 7 cm, several times the RTK error.

Uncorrected, this appears as position motion and the station-keeping loop responds with the thrusters.

The estimator corrects it as long as each antenna's body-frame position is entered in the antenna-offset parameters (the `gnss_a`, `gnss_b` frames of Table 12). The fix is then moved from the antenna to the centre of gravity using roll and pitch before it is fused, and the moving-baseline vector is projected to the horizontal before it becomes a heading. The IMU offset is entered for the same reason.

These offsets are measured on the finished hull, with the CG found by balancing (Section 9.7), and recorded in the hull description file.

For docking and obstacle avoidance a short-range lidar or an onboard camera provide relative measurements to the autopilot or to the companion computer.

6.9 Power, wiring, ingress

Separate the servo rail from the flight-controller supply with its own regulator. Size the main battery for all effectors at full thrust plus 30 %. Monitor current with the autopilot's battery monitor so that a failsafe can act.

Put the receiver, autopilot and GNSS antennas as high and as far from the ESCs as the hull allows. Waterproof connectors between the removable deck and the hull are justified. These are the “minor additional requirements” anticipated in the 2019 conclusion.

6.10 Configuration summary

Table 7 collects the ArduPilot parameters that define the stack for layout A. A full parameter file per hull is a deliverable of phase 2 (Section 10).

Table 7: Key ArduPilot configuration for layout A. Values in italics are hull specific.

Parameter	Value	Purpose
FRAME_CLASS	Boat	Boat behaviours (no crash check on drift, loiter semantics).
FRAME_TYPE	Marine	The marine frame of Section 6.5; reports lateral capability to the position controller.
SERVO1..5_FUNCTION	Motor1, Azimuth1, Motor2, Azimuth2, Motor3	Port thrust, port angle, stbd thrust, stbd angle, bow tunnel.
SERVOx_MIN/MAX/TRIM	<i>calibrated</i>	Mechanical range of each actuator; trim = zero thrust / wash aft ($\alpha = 0$).
RCMAP_*	as Table 5	Throttle = surge, roll = yaw, yaw = sway.
RCx_OPTION	Arm/Disarm, Mode, Pivot, Bias	Arm switch, mode switch, pivot and bias inputs of the marine frame.
MFR_KX/KY/KN	<i>6, 3, 2</i>	Full-stick force and moment (N, N, N m).
MFR_BIAS	0	Bias thrust fraction; off by default.
MF1_TYPE..OUT	2, -0.65, -0.15, 0, 180, 5, 3, 180, 1	Port azimuth (type, l_x , l_y , park angle $\alpha = 0$ with force ahead, sector half-width, T_f , T_r , slew, output index).
MF2_*	2, -0.65, +0.15, 0, 180, 5, 3, 180, 2	Starboard azimuth.
MF3_*	1, 0.60, 0, 90, 0, 2, 2, 0, 3	Bow tunnel, force to starboard positive.
GPS_TYPE, GPS_TYPE2	moving-base pair	Dual RTK, GNSS heading.
EK3_SRC1_YAW	GNSS with compass fallback	Heading from the antenna baseline, compass retained.
ATC_STR_RAT_*, ATC_SPEED_*	<i>tuned</i>	Yaw-rate and speed loops for Acro and above.
PSC_*, LOIT_*	<i>tuned</i>	Position controller and station-keeping radius.
FS_*	Hold	RC, GCS, battery, EKF failsafes to all-stop.

7 Autonomous operation

7.1 Mode mapping

Table 8 maps the Rover modes to what the operator of an OSV model expects. The 2019 phase 3 called for dynamic positioning, reference systems, navigation scripting and path planning; all four are stock functions once the allocation layer exists.

7.2 Station keeping

Loiter with a small radius is a basic DP mode. The position controller produces surge and sway velocity demands, the attitude controller turns them and the heading error into τ_d , and the allocator distributes them.

Performance depends, in this order, on the heading source at zero speed (Section 6.8), the tuning of the yaw-rate and speed loops, and the accuracy of the thrust model in the allocator

Table 8: Rover modes as seen from the bridge.

Mode	Behaviour	Bridge equivalent
Manual	Sticks are forces.	Individual levers / manual joystick.
Acro	Yaw rate and speed closed loop, heading held on release.	Joystick with auto heading.
Steering	Heading-rate control tuned for transit.	Autopilot follow-up steering.
Hold	Zero force, armed.	All stop.
Loiter	Position and heading held within a radius; sticks nudge the setpoint.	Auto position (DP).
Auto	Waypoint mission with speed and heading control.	Track pilot.
RTL / SmartRTL	Return to launch, directly or retracing the path.	Return to safe position.
Guided	Position, velocity or heading targets from a ground station or companion.	DP with external reference / remote console.
Dock	Approach a visual target and stop against it.	Auto berthing.
Follow	Maintain offset to another vehicle.	Escort / shadowing.
Circle	Orbit a point at set radius and speed.	Survey pattern.

(deadband, shaping, reverse efficiency). The loiter parameters offer heading-relative or bow-to-target behaviour; for an OSV model the heading last commanded is held. The disturbance layer of Section 8 provides the remaining improvement.

7.3 Ground station operation

QGroundControl is the preferred ground station. It provides click-to-go (Guided), missions (Auto), geofences, live tuning, video and a joystick layer that accepts the BU0836X console of Section 5.4 as a standard USB device. Mission Planner and MAVProxy remain useful for configuration and automation.

A DP console is a thin layer on top: heading and position setpoints sent as Guided targets, with the joystick mode as manual backup. Parameter files and mission files together are the distributable product that the 2019 transmitter model file was intended to be.

QGroundControl is a general-purpose vehicle station with three limitations for marine work: it manages one vehicle at a time in practice, it has no radar picture, and it uses aeronautical or street maps rather than nautical charts.

Oceanomatics is developing a marine ground station that will replace QGroundControl as it matures. It coordinates several units (several μDP hulls, or a hull and its escorts) on one display with shared setpoints and formation logic. It composites radar returns from the vessels and from shore units into one picture.

It also displays S-57 and S-100-series (S-101) electronic navigational charts, so that station keeping, docking and missions are planned on the chart a full-size operation would use. It uses the same MAVLink and ROS 2 interfaces as Section 9.5; nothing on the vessel changes with the station.

7.4 Companion computer and perception

A Raspberry Pi 5 or Jetson-class computer connects to the autopilot over MAVLink or, in recent ArduPilot releases, over DDS as a ROS 2 node. Onboard vision, lidar processing, docking-target detection and multi-vessel coordination run there. It sends a docking target so that Dock mode can berth against a quay wall, feeds proximity data to the autopilot's avoidance so that Auto

and Guided steer around obstacles, and runs higher-level behaviours (anchor-handling sequences, escort patterns) that issue Guided targets.

The preferred hardware form is an integrated bundle of a Pixhawk-class flight controller and a Raspberry Pi compute module in one enclosure (Holybro's is the best known; Oceanomatics has no affiliation with Holybro): one board stack, one power input, internal links, and a companion running Oceanomatics software on ordinary Linux beside stock ArduPilot with the marine frame.

The condition is sealing. These bundles are built for aircraft and ground vehicles and are not waterproof; on a model exposed to spray, condensation and occasional swamping they need a sealed enclosure with glanded cable entries, a breather membrane and conformal coating, which must be weighed against the bundle's cost.

Where that cost is not justified, a discrete flight controller and a separately sealed Raspberry Pi, linked by a short serial or Ethernet cable, provide the same functions with cheaper replacement parts.

7.5 How far can a model go

DP class notation [7] asks, from equipment class 2 upward, for redundant position and heading references, defined thruster-failure behaviour and a capability plot; class 1 tolerates a single failure that ends in loss of position. At model scale all four can be imitated.

Two GNSS receivers give heading redundancy against the compass and position continuity if one fails, though not the independent second reference a classed vessel needs, since both share the constellation and the correction link. The allocator degrades gracefully when an effector is marked failed (its columns leave $\mathbf{B}$), and Section 4.7 is the capability plot.

No equivalence with a classed system is claimed. The result is a faithful teaching model of one, which was the 2019 requirement.

8 Disturbances and a self-correcting model

Station keeping and track keeping are limited by wind, current and model error. ArduPilot's controllers absorb constant disturbances with their integrators, which is adequate for first trials but not for a product: the integral state is not a physical quantity, is not shared between modes, is reset at every mode change, lags every gust and turn, and is lost at power-off.

This section adds three elements of a full-size DP system: wind feed-forward from a sensor, a current estimate carried as a state of the stack, and a vessel model identified online and persisted between sessions. Figure 8 shows their place in the loop.

8.1 Model with disturbances

The manoeuvring model of Eq. (4) is completed with the two environmental terms:

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r = \boldsymbol{\tau} + \boldsymbol{\tau}_{\text{wind}} + \mathbf{R}(\psi)^\top \mathbf{b}, \quad \boldsymbol{\nu}_r = \boldsymbol{\nu} - \mathbf{R}(\psi)^\top \mathbf{V}_c, \quad (14)$$

where $\mathbf{V}_c$ is the current velocity in the North-East frame, so that damping acts on the water-relative velocity $\boldsymbol{\nu}_r$; $\boldsymbol{\tau}_{\text{wind}}$ is the wind load; and $\mathbf{b}$ is a slowly varying force bias in the earth frame that collects what the model does not know (second-order wave drift, hull asymmetry, a fouled propeller). This is the standard DP plant model [2].

Each term is handled by a different mechanism, and all enter the frame the same way, as a feed-forward added to the controllers' demand before allocation:

$$\boldsymbol{\tau}_d = \boldsymbol{\tau}_{\text{ctl}} + \boldsymbol{\tau}_{\text{ff}}, \quad \boldsymbol{\tau}_{\text{ff}} = -\hat{\boldsymbol{\tau}}_{\text{wind}} - \mathbf{R}(\psi)^\top \hat{\mathbf{b}} + \mathbf{D}(\boldsymbol{\nu}_{r,d})\boldsymbol{\nu}_{r,d} + \mathbf{M}\dot{\boldsymbol{\nu}}_d, \quad \boldsymbol{\nu}_{r,d} = \boldsymbol{\nu}_d - \mathbf{R}(\psi)^\top \hat{\mathbf{V}}_c. \quad (15)$$

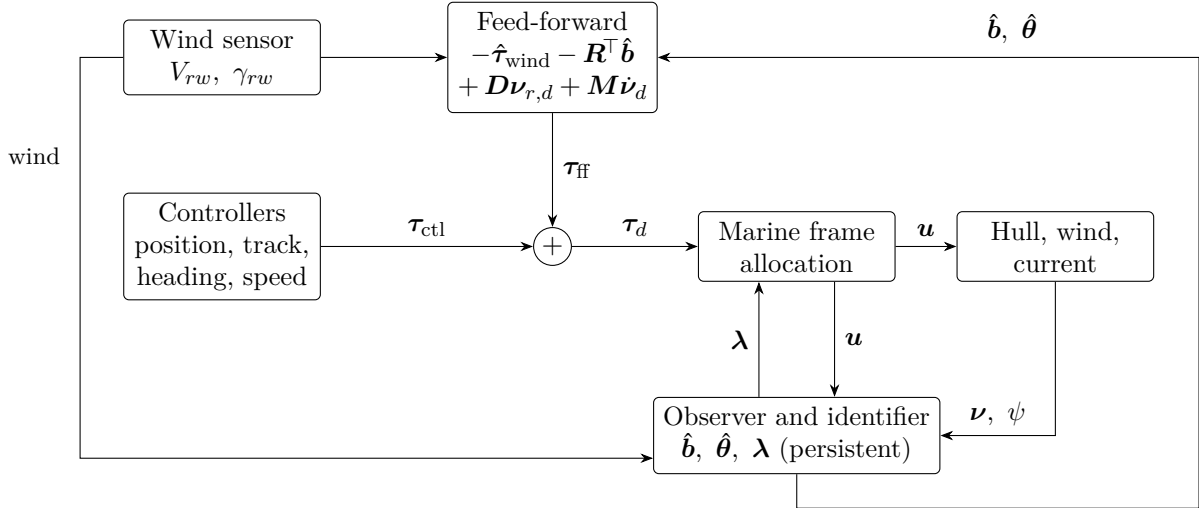


Figure 8: Disturbance feed-forward and model adaptation around the marine frame. The controllers see only the residual; the observer and the identifier learn from the allocated force $\mathbf{u}$ and the estimator's motion.

Here $\mathbf{v}_{r,d}$ is the desired velocity through the water, and $\hat{\mathbf{b}}$ is the part of the bias that the current estimate does not explain (Section 8.3), so that the current is counted once, in the damping, and not again in the bias.

The controllers then regulate only the residual: station keeping and track keeping are achieved without high gains, and the integrators stay near zero, so mode changes are free of transients.

8.2 Wind feed-forward for station keeping and track keeping

The wind sensor serves two functions.

In *station keeping*, wind is the largest and fastest-changing disturbance at model scale. A gust arrives within a second; an integrator responds in tens of seconds, by which time the hull has moved. With the wind load computed from the sensor and cancelled before allocation, the position loop sees only the feed-forward error, and the excursion on a gust is limited by the thrusters' response time.

In *track keeping* (Auto and Guided transits), a beam wind produces sway and a yaw moment. Without feed-forward the vessel leaves the track until the cross-track integrator has built up a correction, and repeats this at every leg change and wind shift. With feed-forward the allocator meets the load directly; the mechanism depends on speed.

At slow speed the heading is maintained. The bow stays on the track bearing and the tunnels or azimuths supply the side force that holds the track; a rudder hull cannot do this and uses the split of Section 4.4.

At cruising speed the vessel holds a heading a few degrees off the course to compensate for wind and current, and the hull's own resistance provides the side force. This is a small heading offset, not crabbing; with feed-forward it is set from the known load rather than found by the integrator on each leg.

Head and following winds enter the speed loop the same way, so speed over the leg is held.

A wind sensor on the model gives the apparent wind speed and direction (V_{rw}, γ_{rw}) relative to the bow. The wind load is

$$\boldsymbol{\tau}_{\text{wind}} = \frac{1}{2} \rho_a V_{rw}^2 \begin{bmatrix} C_X(\gamma_{rw}) A_F \\ C_Y(\gamma_{rw}) A_L \\ C_N(\gamma_{rw}) A_L L \end{bmatrix}, \quad (16)$$

with A_F , A_L the frontal and lateral projected areas above the waterline and L the length.

The coefficient functions are initialised from published superstructure coefficients. Written as a short Fourier series in γ_{rw} they enter Eq. (16) linearly, so the hull identifier of Section 8.4 identifies them on the water as well.

At model scale the sensor must be small: an ultrasonic anemometer of a few tens of grams, or a vane potentiometer with a cup or propeller anemometer on a pulse output. Placement matters more than the instrument.

The sensor must be unshadowed: above the superstructure and clear of mast, funnel and antennas. A sensor in the wake of a deckhouse measures a wind the hull is not exposed to, and the feed-forward then acts in the wrong direction. The masthead, or a dedicated short pole, is the correct position; the identifier references the wind coefficients to that height.

The apparent-to-true conversion, needed for logging, the ground station and transit planning, uses the estimator's velocity.

ArduPilot's wind-vane library, from its sailboat support, reads analogue vanes, pulse and NMEA anemometers and computes true wind from apparent; the marine model uses its output rather than a second driver.

The feed-forward is applied with a confidence weight that ramps from zero to one as the coefficients converge, so that an uncalibrated sensor cannot drive the hull the wrong way. Until the identifier of Section 8.4 exists (phase 3), the weight is a fixed conservative value set from bench and published coefficients.

8.3 Current as a state of the stack

Current cannot be sensed like wind. A water-speed or Doppler log does not work on a hull this size: it sits within a metre of the thrusters, and their wash reaches it whenever the vessel manoeuvres, which is when the current matters. A log reading thruster wash as current would feed a wrong value into the feed-forward. The current is therefore estimated, not measured.

The estimated quantity is the bias $\mathbf{b}$ of Eq. (14). A DP observer of the Fossen type propagates the model with the allocated force, compares it with the estimator's position, velocity and heading, and corrects a slowly varying earth-frame force bias with a time constant of minutes.

With the damping identified, or taken from the bench until then, the bias converts to an equivalent current through the linear part of $\mathbf{D}$,

$$\hat{\mathbf{V}}_c \approx \mathbf{R}(\psi) \mathbf{D}^{-1} \mathbf{R}(\psi)^\top \hat{\mathbf{b}}, \quad (17)$$

which the ground station displays as set and drift.

Once converted, only the remaining bias enters the feed-forward of Eq. (15). A water-speed log, if fitted, serves only as a cross-check in steady transit with the thrusters running steadily, where $\mathbf{V}_c = \mathbf{v}_{\text{ground}} - \mathbf{R}(\psi) \boldsymbol{\nu}_r$ is reliable; it is not a source during station keeping.

The current is a state on the ArduPilot side: a North-East velocity vector with covariance, owned by the estimation layer and independent of the mode. The frame reads it for feed-forward, the modes for planning (a transit against a known set is planned at the correct speed and heading), and the logger and telemetry report it.

The stock firmware holds the same information in one controller's integrator. As a state it can be displayed, validated, saved and compared between sessions.

8.4 A self-correcting hull and effector model

The parameters of Eq. (14) and Eq. (16) are measured once on the bench (Section 9.7) and drift afterwards: battery voltage sags, propellers foul, ballast moves, and the wind coefficients are not measured at all. Instead of manual retuning after each trial, the vehicle identifies them from its own data in operation.

The model is linear in its parameters once the regressor is chosen. The allocated force $\boldsymbol{\tau}$ is known because the frame computed it, and the motion $\boldsymbol{\nu}$, $\dot{\boldsymbol{\nu}}$ comes from the estimator. Collecting the unknowns in $\boldsymbol{\theta}$ (diagonal mass and added mass, linear and quadratic damping, wind Fourier coefficients),

$$\boldsymbol{\tau} = \boldsymbol{\Phi}(\boldsymbol{\nu}, \dot{\boldsymbol{\nu}}, V_{rw}, \gamma_{rw}) \boldsymbol{\theta} + \mathbf{R}(\psi)^\top \mathbf{b}, \quad (18)$$

solved at the loop rate by a recursive least-squares estimator with a forgetting factor [9, 10].

Effector effectiveness, the ratio of delivered to commanded thrust, is identified alongside as a factor λ_i per effector. With the hull parameters held, the force the hull actually received, $\boldsymbol{\Phi}\hat{\boldsymbol{\theta}} + \mathbf{R}^\top \hat{\mathbf{b}}$, is regressed on the commanded effector forces, $\boldsymbol{\Phi}\hat{\boldsymbol{\theta}} + \mathbf{R}^\top \hat{\mathbf{b}} = \mathbf{B} \text{diag}(\mathbf{u}) \boldsymbol{\lambda}$, again linear.

The allocated force cannot be used here, since it equals $\mathbf{B}\mathbf{u}$ by construction. A λ_i below its bound indicates a fouled propeller or a failing ESC; it is flagged to the operator and applied in the allocation immediately, so the other effectors take up the load.

Four rules constrain the estimator. *Excitation*: it updates only when the regressor is informative. Manual and Acro trials provide that; quiet station keeping does not, so the update is gated on a measure of excitation and the forgetting factor stays close to one. *Bounds*: every parameter is confined to an interval around its bench value, and a step outside it is rejected and logged, never applied.

Confidence: the covariance trace is reported as a model-quality figure, and the model-based feed-forward of Eq. (15) is blended in as it falls. *Consistency*: the identified parameters are those of the simulation models of Section 9, so the physics model and the Gazebo world converge towards the real hull as it is sailed, and the envelope of Section 4.7 is redrawn from identified thrusts rather than nominal ones.

8.5 Persistence

The quantity to persist is the identified hull model. Identification requires an excitation run and a period of sailing, and need not be repeated at every controller restart.

The identified parameters, the effectiveness factors and the last current estimate are stored in a parameter group (working name MDL_*) written to storage on disarm, with a quality figure and a timestamp, using the mechanism ArduPilot already applies to learned compass offsets.

At the next start the model is available immediately, the feed-forward is active from the first minute, and the estimator refines the existing values.

A persistent model is validated with the test used in DP proving trials and FMEAs: all position references are switched off for a few minutes, the system holds station on the model alone (dead reckoning from allocated forces, wind and current estimate), and the drift-off is measured when the references return.

A small drift-off confirms the model; the direction of a large one identifies the erroneous term.

The test is run in simulation as a scenario and on the water during trials.

Three safeguards apply: the ground station can freeze the model (for demonstrations or after a hull change), reset it to bench values, or export it; every saved set is also in the flight log, so a regression can be traced to the session that produced it; and the hull description file of Section 9.5 is updated from the exported set.

8.6 Where it lives

Table 9 places each element in the stack. The allocator's interface is unchanged: the frame receives a demand and a set of effectiveness factors.

Table 9: Components of the disturbance and model layer.

Element	Where	Inputs and outputs
Wind sensor	Existing wind-vane library; new sensor placement in the transform tree.	Apparent wind speed and direction; true wind for logging, telemetry and transit planning.
Wind feed-forward	Marine model library, called by the modes before the frame.	Coefficients from the identifier; $\hat{\tau}_{\text{wind}}$ with confidence weight, into the position, track and speed loops.
DP observer	Marine model library, at the loop rate.	Allocated force, estimator state; earth-frame bias $\hat{\mathbf{b}}$ and covariance.
Current state	Estimation layer; new message in log and telemetry.	Bias and identified damping (a log only as a transit cross-check, since thruster wash corrupts it when manoeuvring); $\hat{\mathbf{V}}_c$ with covariance, shown as set and drift.
Identifier	Marine model library; recursive least squares, excitation-gated.	Motion, force, wind; hull parameters $\hat{\boldsymbol{\theta}}$, effector effectiveness $\boldsymbol{\lambda}$, quality figure.
Effector effectiveness	Marine frame.	$\boldsymbol{\lambda}$ applied in allocation; low values flagged.
Persistence	Parameter group MDL_*, saved on disarm; hull YAML export.	Learned state across sessions; freeze, reset, export from the ground station.

9 Verification

9.1 From paradigms to test cases

The 2019 manoeuvring paradigms become automated acceptance tests on the allocator, run against **B** without physics (Table 10). They detect sign and geometry errors in seconds and are re-run whenever a parameter file changes.

9.2 Software in the loop

ArduPilot SITL runs the real firmware on a workstation and accepts an external physics model over a JSON UDP interface: the autopilot sends the 16 servo outputs, the model returns position, attitude, velocity and IMU readings. The frame code is the same binary in SITL and on the flight controller.

A 3-DOF manoeuvring model with mass and added mass, linear and quadratic damping, wind and current inputs and the same effector table as the allocator (Section C) is sufficient to exercise every mode with a real ground station and a real or simulated radio, including failsafes and station keeping. It replaces the transmitter simulator of 2019 with a moving hull.

The Gazebo bridge implements the same JSON interface, so the model can be replaced by a full Gazebo world without changes on the autopilot side (Section 9.3).

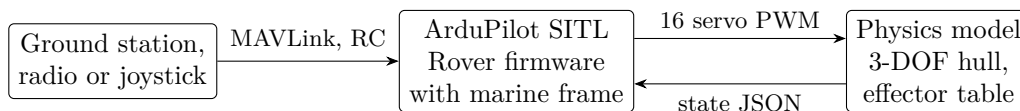


Figure 9: Software-in-the-loop chain. The same firmware and parameter file later go onto the flight controller.

Table 10: Allocation acceptance tests, all layouts.

Demand	Expected
Disarmed	All thrust zero; steerable effectors at sector centre.
Armed, zero demand	$\mathbf{Bu} = \mathbf{0}$ within tolerance; only bias present, none unless enabled.
Pure surge $\pm$	$Y \approx 0$, $N \approx 0$; azimuths at 0° or flipped; tunnels off.
Pure sway $\pm$	$X \approx 0$, $N \approx 0$; tunnel and stern effectors in proportion to lever arms.
Pure yaw $\pm$, pivot centre	$X \approx 0$, $Y \approx 0$; couple only.
Yaw, pivot bow / stern	Lateral force applied aft / forward, $Y_d = -p(2/L)N_d$.
Combined at full stick	Produced force parallel to demand (uniform) or N preserved (priority).
Water-relative speed above U_t	Tunnel column scaled to zero; sway and yaw from stern only.
Step in azimuth demand	Angle slews at the configured rate; thrust blanked while far.
Effector marked failed	Remaining effectors produce the demand or the closest scaled version.
One-directional effector asked the wrong way	Column dropped and re-solved; produced force parallel to demand, $N \approx 0$; no clipped output.

9.3 Simulation levels

Three simulation levels are used, in increasing order of fidelity and cost (Table 11). All share one interface: SITL communicates with any physics backend through the same JSON protocol, so the Python model of Section C and a Gazebo world are interchangeable behind the same autopilot, frame and parameter file.

Level 0 runs headless in seconds and is used in continuous integration. Level 2 adds waves, current, cameras and lidar, and exercises perception, docking and the ROS 2 side.

Table 11: Simulation levels.

Level	What runs	What it is for
0. Allocation only	The frame's allocator class as a host unit test against $\mathbf{B}$, no autopilot, no physics.	Acceptance tests of Table 10; seconds; every parameter change and every firmware build.
1. SITL + 3-DOF model	ArduPilot SITL with the marine frame, Python JSON model (Section C), ground station or scripted MAVLink.	Modes, controller tuning, failsafes, station keeping in calm water; headless CI.
2. SITL + Gazebo + ROS 2	ArduPilot SITL with the marine frame, Gazebo Sim with hydrodynamics, buoyancy, thruster and joint plugins, <code>ardupilot_gazebo</code> bridge, AP_DDS to ROS 2, <code>ros_gz</code> bridges for sensors.	Waves, current and wind; camera and lidar for vision, docking and avoidance; companion-computer software; demonstrations.

9.4 Gazebo: hull, effectors and hydrodynamics

The Gazebo world is built from the same effector table as the allocator. The hull is one link with mass, inertia and a collision volume for buoyancy; each effector is a link, plus a joint for the steerable types, placed at (l_x, l_y) from the CG. Four Gazebo Sim systems provide the physics.

- *Hydrodynamics* applies the Fossen-style added-mass and damping terms to the hull link. Its coefficients are the entries of $\mathbf{M}_A$ and $\mathbf{D}$ of the 3-DOF model ($\mathbf{xDotU}$, $\mathbf{yDotV}$, $\mathbf{nDotR}$ for added mass; $\mathbf{xU}$, $\mathbf{xUabsU}$, $\mathbf{yV}$, $\mathbf{yVabsV}$, $\mathbf{nR}$, $\mathbf{nRabsR}$ for linear and quadratic damping), so the Python model and the Gazebo world are identified once and stay consistent. The same system accepts an ocean-current vector, which is how station-keeping under disturbance is tested.
- *Buoyancy* (graded) gives heave, roll and pitch restoring from the hull volume, which the 3-DOF model ignores and which matters for tunnel-thruster immersion and for camera motion.
- *Thruster* models each propeller with the propeller law $T = \rho K_T n |n| D^4$ from a thrust coefficient, diameter and fluid density, with optional reverse asymmetry; one instance per azimuth, tunnel or shaft propeller. A water jet is represented by a thruster with a restricted nozzle joint and a bucket joint that scales the thrust between +1 and $g(-1)$.
- *Joint position controller* drives the azimuth, rudder, nozzle and bucket joints at the slew rate and range of the real actuators. Rudder lift, when it is not folded into the allocator's sector model, is added with the lift-drag system on the rudder link.

The `ardupilot_gazebo` plugin closes the loop. It implements the SITL JSON interface on the Gazebo side and maps each of the 16 servo outputs to a joint or thruster command through a `<control>` block (PWM range, multiplier, offset, command type). The mapping is the last column of the effector table, so the Gazebo model and the frame parameters agree on which output drives what without hand editing.

Wave fields for sea-keeping tests are taken from the Virtual RobotX Gazebo work [11] and the wave plugins derived from it.

9.5 ROS 2: frames, bridge and test harness

Frames. ROS follows REP 103 and REP 105 [12, 13]: world frames are East-North-Up, body frames are Forward-Left-Up, angles are right-handed about z up. Marine control, ArduPilot's internal state and the allocator of Section 4 use North-East-Down and Forward-Right-Down (SNAME). Both are needed; the conversion must be done in one place only.

Table 12 fixes the tree with one conversion only, the fixed rotation between `base_link` (FLU) and `base_link_frd` (FRD), which is a 180° roll:

$$\mathbf{p}_{\text{FRD}} = [x, -y, -z]_{\text{FLU}}^\top, \quad \mathbf{p}_{\text{NED}} = [y, x, -z]_{\text{ENU}}^\top, \quad \psi_{\text{NED}} = \frac{\pi}{2} - \psi_{\text{ENU}}. \quad (19)$$

Every effector gets a frame at its position on the hull with z along the steering axis. The azimuth angle α is then the joint angle, and `tf` shows the pods turning.

Bridge. ArduPilot supports ROS 2 natively through AP_DDS over a micro-XRCE agent on a serial port or UDP, in SITL and on the flight controller alike. It publishes the filtered pose and twist, GNSS, IMU, battery, time and the static transforms it knows about, and it subscribes to velocity commands and external pose, with services for arming, mode changes and pre-arm checks.

In level 2 the Gazebo sensors (camera, lidar, ground-truth pose) reach ROS 2 through the `ros_gz` bridges, and the simulation clock is published so that every node runs on simulated time.

The ROS 2 side consists of three nodes, none of which touches the allocator:

- a *frame publisher* that reads the effector table and sensor placement file and publishes the static transforms of Table 12, plus the effector joint angles read back from the autopilot or from Gazebo;

Table 12: Transform tree for μDP. Static transforms are generated from the effector table and the sensor placement file.

Frame	Parent	Convention	Origin and use
map	—	ENU	ArduPilot EKF origin (home); AP_DDS filtered pose; Gazebo world with spherical coordinates set to the same origin.
odom	map	ENU	Continuous local frame from the EKF; identical to map unless an external odometry source is fused.
base_link	odom	FLU	Hull CG, REP 105 body frame; what ROS nodes and Gazebo see.
base_link_frd	base_link	FRD	Hull CG, SNAME body frame; the frame of τ , of the effector table and of the allocator.
gnss_a, gnss_b	base_link	FLU	Antenna phase centres; their baseline is the moving-baseline heading, and its offset is what ArduPilot needs in its GPS position parameters.
imu_link	base_link	FLU	Flight-controller IMU; must match the autopilot’s board orientation parameter.
eff_i	base_link	FLU, $z =$ steering axis	Effector i at $(l_x, -l_y, 0)$; child eff_i_rot carries the joint angle.
camera_link, camera_optical	base_link	FLU / optical	Onboard camera; optical frame per REP 103 (z forward, x right, y down).
lidar_link	base_link	FLU	Proximity sensor for avoidance and docking.
wind_link	base_link	FLU	Wind sensor; its yaw offset is what the wind-vane library needs to reference apparent wind to the bow.

- a *DP console* that turns operator or perception setpoints (hold this position and heading, approach that quay, follow that vessel) into Guided-mode velocity or position targets through AP_DDS, the ROS 2 counterpart of the ground-station console of Section 7.3;
- the *perception* nodes (camera vision, lidar processing) that produce docking targets and obstacles for the autopilot’s Dock mode and avoidance.

One source of truth. The effector table exists in three places: the ArduPilot parameter file (MFx_*), the Gazebo SDF (links, joints, thruster and control blocks) and the transform tree. All three are exported from one hull description, so a change of geometry or output slot cannot leave them inconsistent. The hull description defines a hull in μDP and is produced in the configuration tool of Section 9.6.

Test harness. Levels 1 and 2 are driven by ROS 2 launch files with **launch_testing**, so that a scenario is a test case with a pass criterion, and by **rosbag2** recordings and the autopilot’s own logs for replay.

The scenarios of Table 13 are the wet-trial protocol run in advance. Each is run for every hull layout of Section 3.4; the station-keeping cases are run against the capability plot of Section 4.7, so that position is lost at the current speed the envelope predicts and not before.

Table 13: Simulation scenarios with pass criteria (values are placeholders for a 1.6 m hull).

Scenario	Setup	Pass criterion
Step responses	Manual mode, unit step on each axis, calm water.	Force direction within 5° of demand; no yaw induced by sway beyond 0.02 rad/s.
Heading hold	Acro mode, 90° heading step, then current 0.3 m/s on the beam.	Settles within 10 s, overshoot $< 10^\circ$, steady error $< 2^\circ$.
Station keeping	Loiter, current ramped 0 to 0.6 m/s from four directions, waves optional.	Position error < 0.3 m up to the envelope limit, the envelope recomputed with the tunnel faded on water-relative speed; graceful drift beyond.
Pivot point	Acro, yaw at each pivot setting.	Measured centre of rotation within 0.15 m of the commanded point.
Effector failure	Station keeping, one effector marked failed mid-run.	No loss of heading; position error bounded by the reduced envelope.
Failsafes	RC loss, GCS loss, low battery, EKF variance, each in turn.	Configured action within 1 s; thrust zero in Hold; pods parked when disarmed.
Mission	Auto, four-waypoint box with speed changes.	Cross-track error < 0.5 m; tunnel column zero above U_t through the water.
Docking	Dock mode, visual target on a quay in Gazebo.	Contact speed < 0.1 m/s, heading within 5° of the quay.
Wind, station keeping	Loiter, wind step 0 to 6 m/s on the beam, feed-forward off then on.	Peak position excursion halved with feed-forward on; no wrong-way push with uncalibrated coefficients.
Wind, track keeping	Auto, four-leg box in a steady 5 m/s wind, feed-forward off then on.	Cross-track error after each leg change settles in under 5 s with feed-forward on; speed over the leg within 10 % of setpoint.
Current estimate	Loiter, current ramp with the sensor absent.	$\hat{V}_c$ within 0.05 m/s and 15° of the simulated current after 3 min; drawn on the ground station.
Identification	Manual and Acro excitation run, then station keeping.	Identified mass, damping and effectiveness within 10 % of the simulator's values; fouled propeller ($\lambda = 0.5$) flagged within 60 s.
Persistence	Power cycle of SITL after identification.	Learned parameters and current estimate restored; feed-forward active from the first minute.
Model test (drift-off)	Station keeping, all position references switched off for 3 min, then restored.	Drift-off below 1 m in calm water and below 3 m in 0.3 m/s current; the identifier's quality figure predicts which.
Posture change	Layout F: transit to manoeuvring and back at several speeds.	Forward azimuth deploys only below the set speed; no thrust on it until deployed; sway capability changes with the posture as the pre-arm report predicts.
Mooring sequence	Layout G: station keeping, spuds set, thrust off, spuds raised, station keeping resumed.	Position held within 0.3 m through both transitions; no thrust while set; controllers resume without transient.

9.6 The configuration app

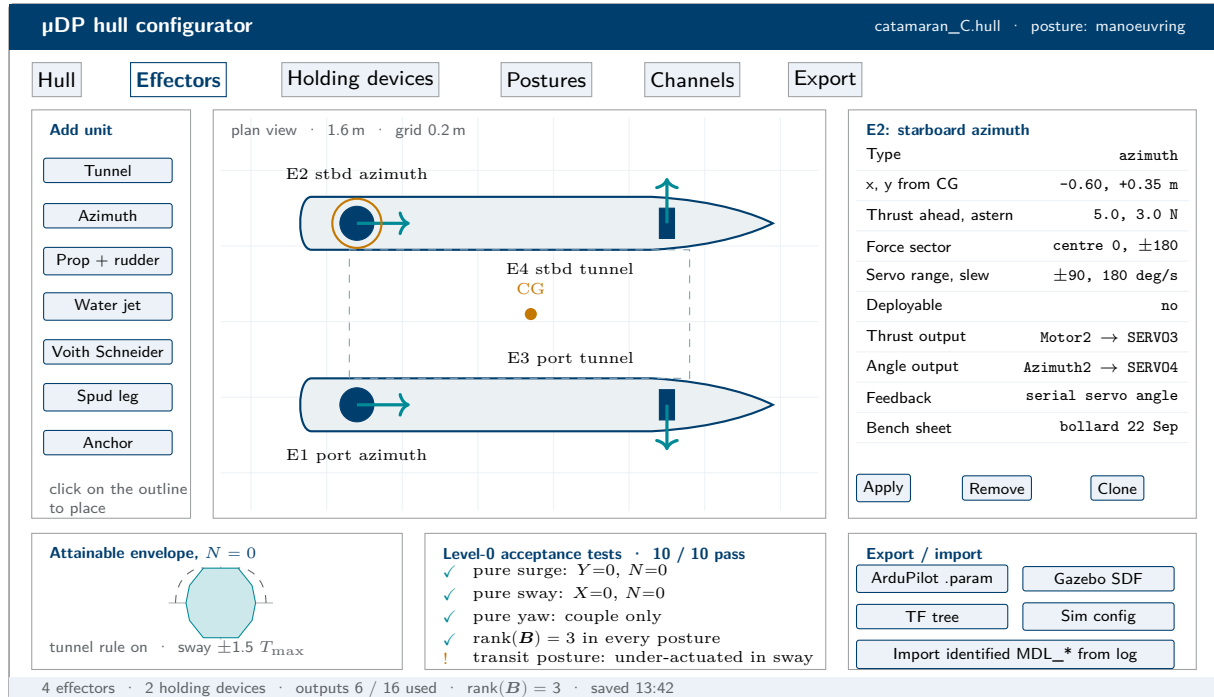


Figure 10: Concept of the configuration app: a hull is drawn or imported, thrusters are placed on the outline and wired to channels, and the envelope, the acceptance tests and the exports follow from the same description.

A hull should not be defined by entering a hundred parameters in a ground-station editor. The configuration tool takes the vessel’s dimensions and outline, then thrusters and channels one at a time. Figure 10 sketches it: plan view in the middle, properties of the selected unit on the right, live envelope and test results below, exports along the bottom.

The hull description it saves is the single source from which the parameter file, the simulation models and the transform tree are generated.

The tool opens on a plan view. The user enters length and beam, draws or imports the outline (DXF or SVG of the deck plan), and places the centre of gravity found by balancing the hull (Section 9.7).

Thrusters are added by clicking on the outline. The type is chosen, and the tool requests that type’s parameters: sector, ahead and astern thrust from the bollard-pull test, slew rate, rudder gains, retractability.

Each unit is then wired: an output channel for thrust, one for steering, one for the auxiliary function if any, and a feedback input if available. The tool knows the autopilot’s channel count and bus options and rejects conflicting assignments. Holding devices and postures are added the same way, with actuators, feedback and interlocks.

As units are added the tool draws the attainable envelope of Section 4.7 and runs the level-0 acceptance tests of Table 10 on the current table, so sign errors and inadequate layouts are visible before export. Sway on a rudder hull, the tunnel rule on a catamaran and per-posture capability are shown as the user works.

Export produces the ArduPilot parameter file, the Gazebo model, the transform tree and the Python physics configuration from one description. Import reads the identified parameters and effectiveness factors of Section 8.4 back from the vehicle logs into the hull file, keeping it current with the hull as sailed. The hull file is kept under version control with the project.

The first version is a stand-alone desktop tool for phase 1, since the table and the exports are needed before hardware is built. It later becomes a page of the Oceanomatics ground station (Section 7.3), so that a hull is configured, simulated and operated from one application.

9.7 Bench identification

Three bench measurements, about an afternoon's work, give the allocator its numbers.

1. *Bollard pull* of each effector ahead and astern against a kitchen scale, at five ESC commands, to fit T_f , T_r and the shaping exponent, and to find the ESC deadband. For a propeller and rudder pair, the same rig with the rudder hard over, ahead and astern, gives k_T and k_R .
2. *Azimuth slew rate* and hysteresis of each steerable effector with a phone camera or the serial servo's own feedback.
3. *Positions* l_x, l_y of each effector relative to the CG, the CG found by balancing the completed hull.

9.8 Wet trials

Trials follow the mode progression: Manual, Acro, Loiter with a large radius, Loiter with a tight radius, Auto on a short mission, then the composite demands left unverified in 2019. The autopilot logs every trial; the log replays in the ground station and gives the thrust, angle, position and heading traces used to refine $\mathbf{W}$, the thrust shaping and the controller gains.

10 Roadmap

Phase 0, this document. Methodology, models, implementation outline, acceptance tests. Complete with this revision.

Phase 1, software. Implement the marine frame in the motor library on a fork of Rover (Section 6.5); host unit tests for the acceptance tests of layouts A to E; SITL with the physics model; add the sway-velocity loop; build the first version of the configuration app with its exports (Section 9.6). Deliverable: firmware branch, tests, physics model, the app, one hull description per layout.

Phase 2, hardware. Flight controller, dual RTK GNSS, ELRS receiver, ESCs, servos; bench identification; parameter file for the physical hull; Manual and Acro wet trials. Deliverable: a hull that drives as the 2019 design intended.

Phase 3, autonomy. Loiter and Auto trials, RTK base, BU0836X/Lilaas console on QGroundControl, wind sensor and current state with feed-forward (Section 8), on bench and published coefficients at a fixed confidence weight until the identifier of phase 4 takes over, failsafe drills, capability plot from measured thrusts. Deliverable: station keeping and missions.

Phase 4, push. Level-2 simulation with waves and current; companion computer with onboard vision, docking, avoidance and the ROS 2 nodes of Section 9.5; online identification, effector effectiveness and persistence (Section 8.4); constrained quadratic-programme allocator in the frame, upstream submission; deployable effectors, holding devices and postures (Section 6.6) with layouts F and G; the configuration app folded into the ground station; second and third hull types on the same stack; the marine ground station of Section 7.3, first for one hull, then with the multi-vessel, radar and chart layers. Deliverable: μDP as a propulsion-agnostic product.

11 Conclusion

The 2019 study took its requirements from experienced boat drivers and met them with the tools available. Its operator semantics, its bias and pivot-point concepts and its simulate-before-build practice are retained.

The central change is the replacement of percent mixing by force allocation through a configuration matrix built from an effector table. This removes the defects of the 2019 scheme, lets one set of sticks drive any of the hull types considered, and gives the autopilot's closed-loop modes a correct actuation model.

ArduPilot supplies modes, estimation and failsafes. The allocator is implemented in the motor library, not in a script, so that the failsafes reach the thrusters. Wind feed-forward, a current state and an identified, persistent model are added around it, with a marine ground station above. This is μDP. Phase 1 starts from the frame outline, the physics model and the acceptance tests in the appendices.

Acknowledgements

We thank Professor Thomas Bräunl of the University of Western Australia for his ongoing support of our work and for his mentoring through the PhD that runs alongside it.

We thank Uwe Bauer of Bauer-Modelle, Eibenstein, for the photographs in Figure 4 and for making equipment of a quality that lets a model behave like a ship.

References

- [1] P.-L. Constant, *Design of a control system for an OSV model*, Oceanomatics Pty Ltd, internal report, April 2019.
- [2] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*, Wiley, 2011. Chapter 12, control allocation.
- [3] T. A. Johansen and T. I. Fossen, "Control allocation: a survey," *Automatica*, vol. 49, no. 5, pp. 1087–1103, 2013.
- [4] O. J. Sørndalen, "Optimal thrust allocation for marine vessels," *Control Engineering Practice*, vol. 5, no. 9, pp. 1223–1231, 1997.
- [5] T. A. Johansen, T. I. Fossen and S. P. Berge, "Constrained nonlinear control allocation with singularity avoidance using sequential quadratic programming," *IEEE Trans. Control Systems Technology*, vol. 12, no. 1, pp. 211–216, 2004.
- [6] K.-P. Lindegaard and T. I. Fossen, "Fuel-efficient rudder and propeller control allocation for marine craft: experiments with a model ship," *IEEE Trans. Control Systems Technology*, vol. 11, no. 6, pp. 850–862, 2003.
- [7] IMCA, *Guidelines for the Design and Operation of Dynamically Positioned Vessels*, IMCA M 103, rev. 2.1, 2019.
- [8] ArduPilot Development Team, *Rover documentation; motor library; SITL JSON interface; AP_DDS ROS 2 interface; ardupilot_gazebo plugin*, <https://ardupilot.org>, https://github.com/ArduPilot/ardupilot_gazebo, accessed September 2026.
- [9] L. Ljung, *System Identification: Theory for the User*, 2nd ed., Prentice Hall, 1999.
- [10] K. J. Åström and B. Wittenmark, *Adaptive Control*, 2nd ed., Dover, 2008.
- [11] B. Bingham et al., "Toward maritime robotic simulation in Gazebo," *Proc. MTS/IEEE OCEANS*, Seattle, 2019. Virtual RobotX, <https://github.com/osrf/vrx>.
- [12] T. Foote and M. Purvis, *REP 103: Standard units of measure and coordinate conventions*, ROS Enhancement Proposal, 2010 (rev. 2014).
- [13] W. Meeussen, *REP 105: Coordinate frames for mobile platforms*, ROS Enhancement Proposal, 2010.

- [14] EdgeTX, ExpressLRS, Cockpit (Blue Robotics) and OpenHD projects, <https://edgetx.org>, <https://www.expresslrs.org>, <https://github.com/bluerobotics/cockpit>, <https://openhdfpv.org>, accessed September 2026.

A The 2019 equation set, for the record

With inputs s (surge), y (sway), n (yaw), p (pivot) in $[-1, 1]$, bias thrust t_0 , idle boolean σ , the aft-pointing constant C_0 and the dial-set semi-constants k_s, k_n (initial values $\frac{1}{2}, \frac{1}{4}$, see Section 2.2), the 2019 transmitter mixes were

$$\begin{aligned} T_P &= (|s| + y + w_{\text{aft}} |n| + t_0) \sigma, & T_S &= (|s| - y + w_{\text{aft}} |n| + t_0) \sigma, \\ \alpha_P &= (-k_s s - k_n w_{\text{aft}} n) \sigma - C_0(1 - \sigma), & \alpha_S &= (+k_s s - k_n w_{\text{aft}} n) \sigma + C_0(1 - \sigma), \\ T_{B1} = T_{B2} &= (-y - w_{\text{fwd}} n) \sigma, & w_{\text{aft}} &= \min(p + 1, 1), \quad w_{\text{fwd}} = \min(1 - p, 1), \end{aligned}$$

every output clipped to $[-1, 1]$, azimuth angles in units where ± 1 is $\pm 180^\circ$ from the outboard position, and $C_0 = \frac{1}{2}$ pointing the pods aft. This is the compact form from the original document with its printed gains replaced by symbols.

The developed form and the prose of the original contain transcription errors: swapped input channel numbers for surge and pivot, “left stick” for the yaw stick, “50 % = 0.25”, a wrong sign on the idle term of the starboard azimuth angle, a dropped +1 in the forward pivot weight, and inconsistent numbering of the weight functions. The simulator screenshots confirm that the compact form above is what we programmed. The qualitative pivot table of the original also differs from it: it gives each end half weight at pivot centre, whereas the compact form has $w_{\text{aft}} = w_{\text{fwd}} = 1$ there, a pure couple. The compact form is what the screenshots show, and it is the 2019 equation set of this paper.

Two properties of the set. Sway to starboard ($y > 0$) gives $T_P = t_0 + y$, $T_S = t_0 - y$ with both pods pointing outboard, so both push the stern to starboard while the tunnels push the bow to starboard; the induced yaw moment is $-2ay + l_B y$ in the units of Figure 3, zero only when $l_B = 2a$. And the pivot weights act on yaw only; their effect is the open-loop form of Section 4.6.

B Marine frame implementation outline

The listing below is `code/marine_frame_outline.cpp` from the accompanying folder: an outline of the frame of Section 6.5 in the style of the motor library, with the allocation loop, the steerable-effector resolution, the output stage and the pre-arm check written out and the integration points noted at the end. It is a design sketch, not compiled code; names and signatures are to be confirmed against the target branch in phase 1.

```

1 // marine_frame_outline.cpp -- outline of the microDP marine frame for
2 // ArduPilot Rover (AP_MotorsUGV). This is a design sketch in ArduPilot style,
3 // not compiled code: names, signatures and integration points are to be
4 // confirmed against the target branch. The allocator itself has no HAL
5 // dependency so that it can be built as a host unit test.
6
7 #pragma once
8 #include <AP_Param/AP_Param.h>
9 #include <AP_Math/AP_Math.h>
10 #include <SRV_Channel/SRV_Channel.h>
11
12 #define MARINE_MAX_EFF 12
13
14 // One effector: parameters (persisted) and state (runtime).
15 class AP_MarineEffector {
16 public:
17     static const struct AP_Param::GroupInfo var_info[]; // MFx_*
18     AP_Int8 type; // 0 off, 1 fixed, 2 azimuth, 3 prop+rudder, 4 steerable jet,
19                 // 5 cycloidal (as 2 with rng = 180, rate large, tr = 0)
20     AP_Float lx, ly; // m from CG, x forward, y starboard (FRD)
21     AP_Float ang; // deg, force direction (0 ahead, 90 starboard): fixed effector's
22                 // direction, or a steerable one's sector centre and park angle
23     AP_Float rng; // deg: sector half-width (180 = unrestricted)
24     AP_Float tf, tr; // N: max thrust ahead / astern (tr = 0: unidirectional)
25     AP_Float rate; // deg/s: steering slew limit
26     AP_Float kt, kr, ku; // type 3 only: rudder side force per unit thrust ahead / astern /
27                        // per u/l, at unit angle. kr ~ 0.2 kt for a Kort nozzle, more with a
28                        // Becker flap rudder, ~0 for an open rudder; measured at the bench
29     AP_Float fsec; // deg: half-width about ang of the force-direction sector that is fully effective
30     AP_Float feff; // effectiveness of a force outside that sector (wash into a skeg: 0)
31     AP_Int8 out; // n: thrust on SRV function MotorN, angle on AzimuthN
32
33     AP_Int8 depl; // 0 fixed, 1 retractable (deploy actuator on MarineAuxN)
34     AP_Float dspd; // m/s: no deployment above this speed
35     AP_Int8 fb; // feedback: 0 none, 1 serial-servo angle, 2 limit switch, 3 ESC rpm
36
37     enum class State : uint8_t { STOWED, DEPLOYING, ACTIVE, RETRACTING, FAILED };
38     State state; // columns are in B only while ACTIVE
39     float angle_rad; // current steering angle relative to ang
40     float thrust_n; // current thrust command
41     bool failed; // dropped from the allocation (GCS or aux switch)
42 };
43
44 // A holding device is not an effector: when SET it holds the vessel and the
45 // allocation stands down. Spud leg, anchor, mooring line. Both tables and the
46 // postures are authored in the configuration app and exported, not typed in.
47 class AP_MarineHold {
48 public:
49     static const struct AP_Param::GroupInfo var_info[]; // HDx_*
50     AP_Int8 type; // 1 spud leg, 2 anchor, 3 mooring line
51     AP_Int8 out; // actuator output (MarineAuxN)
52     AP_Int8 fb; // feedback: limit switch or timed
53     AP_Float t_set, t_raise; // s: timed transitions when no feedback
54     enum class State : uint8_t { RAISED, LOWERING, SET, RAISING, FAILED };
55     State state;
56 };
57
58 // Postures sequence deployable effectors and holding devices with interlocks.
59 // TRANSIT: forward azimuth stowed, tunnels faded; MANOEUVRE: all deployed;
60 // MOORED: holding devices set, thrust zero, position controller stood down;
61 // JACKED: hull clear of the water, everything off. Driven by an aux switch,
62 // the ground station or a mission command; pre-arm reports the capability of

```

```

63 // each posture (rank and envelope of B with the columns that posture keeps).
64 enum class MarinePosture : uint8_t { TRANSIT, MANOEUVRE, MOORED, JACKED };
65
66 // The frame. Owned by AP_MotorsUGV and selected with FRAME_TYPE = Marine.
67 class AP_MotorsUGV_Marine {
68 public:
69     static const struct AP_Param::GroupInfo var_info[];    // MFR_*
70
71     // Build the extended-thrust columns from the effector table.
72     bool setup();
73
74     // Refuse to arm on a degenerate table. Called from AP_MotorsUGV::pre_arm_check.
75     bool pre_arm_check(char *failmsg, uint8_t failmsg_len) const;
76
77     // Called every main-loop cycle from AP_MotorsUGV::output with the mode's
78     // demands already normalised to [-1, 1]. speed is the signed body-axis
79     // surge speed THROUGH THE WATER (negative when making sternway): the EKF
80     // surge velocity minus the body-axis component of the current estimate
81     // when one exists, the EKF velocity alone until then.
82     void output(bool armed, float steering, float throttle, float lateral,
83                float speed, float dt);
84
85     // Lets the position controller and Loiter command sway.
86     bool have_lateral_control() const { return true; }
87
88     void set_effector_failed(uint8_t idx, bool failed);
89
90 private:
91     // MFR_* parameters (the surviving semi-constants of the 2019 study)
92     AP_Float _kx, _ky, _kn;    // full-stick force X, Y (N) and moment N (N m)
93     AP_Float _bias;           // null-space bias, fraction of tf; default 0
94     AP_Float _blank;          // thrust factor while a pod is > 45 deg off target
95     AP_Float _btspd;           // water-relative speed (m/s) at which lateral fixed thrusters fade out
96     AP_Int8 _prio;             // 1 = yaw priority on saturation, 0 = uniform
97     AP_Float _expo;            // thrust-to-command shaping, 0 linear .. 1 sqrt
98     AP_Float _split;           // prop+rudder pairs: split thrust per N of sway demand
99     AP_Float _split_spd;       // ... applied below this water-relative surge speed (m/s)
100
101     AP_MarineEffector _eff[MARINE_MAX_EFF];
102     AP_MarineHold _hold[4];
103     MarinePosture _posture;
104     void update_postures(float speed);    // state machines, interlocks, deploy/hold outputs
105
106     struct Column { float bx, by, bn, w; uint8_t eff; uint8_t kind; };    // kind: T, X, Y
107     Column _cols[2 * MARINE_MAX_EFF];
108     uint8_t _ncols;
109
110     void build_columns(float speed);    // column effectiveness scheduled on
111                                         // water-relative speed
112     bool allocate(const Vector3f &tau, float *u) const;    // weighted pseudo-inverse (3x3 inverse)
113     // One-directional rule, then saturation policy (MFR_PRIO): a column whose
114     // effector cannot thrust the way the pseudo-inverse asks (tr = 0 and u < 0)
115     // is dropped and the allocation re-solved, one step per such effector;
116     // then the demand is scaled against the limit of the direction commanded.
117     bool allocate_saturation(const Vector3f &tau, float *u) const;
118     float saturation(const float *u) const;    // max |Fi| / Tlim,i, limit by direction
119     void add_bias(float *u) const;    // null-space bias: wash outboard, force
120                                         // inboard
121     void apply_split(Vector3f &tau, float speed);    // twin-screw walking: see below
122     void resolve_steerable(AP_MarineEffector &e, float fx, float fy, float dt) const;
123     void resolve_rudder(AP_MarineEffector &e, float fx, float fy, float speed, float dt) const;
124     void write_outputs(bool armed);
125     void log(const Vector3f &tau, float sat) const;    // MARF log message
126 };
127
128 // ----- output -----
129 void AP_MotorsUGV_Marine::output(bool armed, float steering, float throttle,
130                                  float lateral, float speed, float dt)
131 {
132     // demand in body FRD: X from throttle, Y from lateral, N from steering
133     Vector3f tau(throttle * _kx, lateral * _ky, steering * _kn);

```

```

133     if (!armed) {
134         tau.zero();
135     }
136     apply_split(tau, speed);    // propeller pairs at low speed: create rudder authority first
137     build_columns(speed);      // column weights use the split thrust for the rudder gains
138
139     float u[2 * MARINE_MAX_EFF];
140     if (!allocate_saturating(tau, u)) {
141         // singular table: cannot happen past pre-arm, but never trust it
142         for (auto &e : _eff) { e.thrust_n = 0; }
143         write_outputs(false);
144         return;
145     }
146     if (armed) {
147         add_bias(u);
148     }
149
150     for (uint8_t i = 0; i < _ncols; i++) {
151         AP_MarineEffector &e = _eff[_cols[i].eff];
152         if (_cols[i].kind == 'T') {
153             e.thrust_n = constrain_float(u[i], -e.tr, e.tf);    // no-op after the one-directional rule
154         } else if (_cols[i].kind == 'X') {
155             // X and Y columns of one effector are adjacent
156             if (e.type == 3) {
157                 resolve_rudder(e, u[i], u[i + 1], speed, dt);
158             } else {
159                 resolve_steerable(e, u[i], u[i + 1], dt);
160             }
161             i++;
162         }
163     }
164     write_outputs(armed);
165     log(tau, saturation(u));
166 }
167
168 // Flip for reversible units, sector clamp, slew limit, blanking during swing.
169 // A cycloidal unit (type 5) comes through here with rng = 180, a large rate
170 // and tr = 0, so that the flip, sector, slew and blanking steps are no-ops.
171 void AP_MotorsUGV_Marine::resolve_steerable(AP_MarineEffector &e, float fx, float fy, float dt) const
172 {
173     const float centre = radians(e.ang);
174     const float rng     = radians(e.rng);
175     float T = norm(fx, fy);
176     float a = wrap_PI(atan2f(fy, fx) - centre);
177
178     // force-direction sector, checked BEFORE the flip: a force whose wash
179     // would strike a skeg or the neighbouring pod is lost whichever way the
180     // propeller turns. (A catamaran tunnel is handled differently: it is a
181     // fixed thruster with tr = 0 and ang set to the force direction it has
182     // when washing outboard, i.e. inboard, toward the other hull.)
183     const float fsec = radians(e.fsec);
184     if (fabsf(a) > fsec) {
185         if (e.feff <= 0.0f) {
186             const float clipped = (a > 0) ? fsec : -fsec;
187             T *= MAX(0.0f, cosf(a - clipped));
188             a = clipped;
189         } else {
190             T *= e.feff;
191         }
192     }
193
194     if (e.tr > 0) {    // reversible: least travel
195         const float alt = wrap_PI(a + M_PI);
196         if (fabsf(wrap_PI(alt - e.angle_rad)) < fabsf(wrap_PI(a - e.angle_rad))) {
197             a = alt; T = -T;
198         }
199     }
200     if (fabsf(a) > rng) {    // sector: keep useful component
201         const float clipped = (a > 0) ? rng : -rng;
202         T *= MAX(0.0f, cosf(a - clipped));
203         a = clipped;
204     }

```

```

205     const float step = radians(e.rate) * dt;           // slew
206     e.angle_rad += constrain_float(wrap_PI(a - e.angle_rad), -step, step);
207     if (fabsf(wrap_PI(a - e.angle_rad)) > radians(45)) { // blanking
208         T *= _blank;
209     }
210     e.thrust_n = constrain_float(T, -e.tr, e.tf);
211 }
212
213 // Twin-screw walking. A linear allocator cannot make sway from a pair of
214 // propellers and rudders at rest: both rudder gains are small or zero until a
215 // propeller pushes ahead. When |Y_d| exceeds what the fixed lateral thrusters can give
216 // and |speed| < _split_spd, command a split thrust T_s ahead on one propeller
217 // and the astern thrust that cancels it on the other (T_s = _split * |Y_d|,
218 // capped at tf). The ahead rudder then has gain kt*T_s, the astern one only
219 // the reduced kr*T_s of the nozzle inflow (reversed sign), and the couple
220 // 2*b*T_s of the split is subtracted
221 // from tau as a known term before the linear allocation. The side that goes
222 // ahead is chosen so that this couple helps the yaw balance rather than
223 // fighting it. The linear step then puts the stern side force on the ahead
224 // rudder and the forward side force and yaw balance on the bow thruster.
225 void AP_MotorsUGV_Marine::apply_split(Vector3f &tau, float speed)
226 {
227     // ... find the propeller pair (two type-3 effectors), decide the side,
228     //      set their pre-commanded thrusts, and subtract the split couple.
229 }
230
231 // Propeller + rudder: the force vector does not rotate when the propeller
232 // reverses. Fx is the propeller thrust; Fy follows the flow over the rudder,
233 // Fy = -(kt*max(T,0) + kr*min(T,0) + ku*u/|u|) * delta. The lateral gain is
234 // recomputed every cycle, so astern with sternway the rudder command inverts by
235 // itself. Astern at rest an open rudder has next to no authority; a rudder
236 // behind a Kort nozzle keeps roughly a fifth of the ahead value (kr), with the
237 // sign of sternway, and a Becker flap rudder keeps more.
238 // build_columns() gives the pair's Y column the weight |gy| / gy_ref, which
239 // drops to zero below a threshold and moves sway to the tunnels.
240 void AP_MotorsUGV_Marine::resolve_rudder(AP_MarineEffector &e, float fx, float fy, float speed, float dt)
241 {
242     const
243     const float T = constrain_float(fx, -e.tr, e.tf);
244     const float gy = -(e.kt * MAX(T, 0.0f) + e.kr * MIN(T, 0.0f) + e.ku * speed * fabsf(speed));
245     float delta = 0.0f;
246     if (fabsf(gy) > 1e-3f) {
247         delta = constrain_float(fy / gy, -radians(e.rng), radians(e.rng));
248     }
249     const float step = radians(e.rate) * dt;           // rudder slew
250     e.angle_rad += constrain_float(delta - e.angle_rad, -step, step);
251     e.thrust_n = T;
252 }
253
254 // Thrust to MotorN, angle to AzimuthN. Disarmed: zero thrust, park at centre.
255 // SERVOx_MIN/MAX/TRIM/REVERSED and MOT_SLEWRATE apply downstream as usual.
256 void AP_MotorsUGV_Marine::write_outputs(bool armed)
257 {
258     for (uint8_t i = 0; i < MARINE_MAX_EFF; i++) {
259         AP_MarineEffector &e = _eff[i];
260         if (e.type == 0 || e.out < 1) continue;
261         if (!armed) { e.thrust_n = 0; e.angle_rad = 0; }
262         const float lim = (e.thrust_n < 0) ? e.tr : e.tf;
263         float x = (lim > 0) ? MIN(fabsf(e.thrust_n) / lim, 1.0f) : 0.0f;
264         x = (1 - _expo) * x + _expo * sqrtf(x);           // shaping
265         if (e.thrust_n < 0) x = -x;
266         SRV_Channels::set_output_scaled(SRV_Channel::Aux_servo_function_t(SRV_Channel::k_motor1 + e.out -
267             1), x * 100.0f);
268         if (e.type >= 2) {
269             const float ratio = constrain_float(e.angle_rad / MAX(radians(e.rng), 0.01f), -1.0f, 1.0f);
270             SRV_Channels::set_output_scaled(SRV_Channel::Aux_servo_function_t(SRV_Channel::k_azimuth1 +
271                 e.out - 1), ratio * 4500.0f);
272         }
273     }
274 }
275
276 // Pre-arm: at least one effector, every enabled effector bound to an output,

```

```

274 // limits and sectors sane, configuration matrix of full row rank.
275 bool AP_MotorsUGV_Marine::pre_arm_check(char *failmsg, uint8_t failmsg_len) const
276 {
277     uint8_t n = 0;
278     for (uint8_t i = 0; i < MARINE_MAX_EFF; i++) {
279         const AP_MarineEffector &e = _eff[i];
280         if (e.type == 0) continue;
281         n++;
282         if (!SRV_Channels::function_assigned(SRV_Channel::Aux_servo_function_t(SRV_Channel::k_motor1 +
283             e.out - 1))) {
284             hal.util->snprintf(failmsg, failmsg_len, "Marine: MF%u thrust output not assigned", i + 1);
285             return false;
286         }
287         if (e.type >= 2 &&
288             !SRV_Channels::function_assigned(SRV_Channel::Aux_servo_function_t(SRV_Channel::k_azimuth1 + e.out -
289             1))) {
290             hal.util->snprintf(failmsg, failmsg_len, "Marine: MF%u angle output not assigned", i + 1);
291             return false;
292         }
293         if (e.tf <= 0 || e.rng <= 0 || e.rate <= 0) {
294             hal.util->snprintf(failmsg, failmsg_len, "Marine: MF%u limits invalid", i + 1);
295             return false;
296         }
297     }
298     if (n == 0) { hal.util->snprintf(failmsg, failmsg_len, "Marine: no effectors"); return false; }
299     float u[2 * MARINE_MAX_EFF];
300     if (!allocate(Vector3f(1, 1, 1), u)) { // B W^-1 B' singular
301         hal.util->snprintf(failmsg, failmsg_len, "Marine: configuration matrix singular");
302         return false;
303     }
304     return true;
305 }
306
307 // Integration point in AP_MotorsUGV::output():
308 //
309 // if (_frame_type == FRAME_TYPE_MARINE) {
310 //     _marine.output(armed, _steering / 4500.0f, _throttle / 100.0f,
311 //         _lateral / 100.0f, water_speed_x, dt);
312 // }
313 //
314 // where water_speed_x is the body-axis surge velocity from the EKF minus the
315 // body-axis component of the current estimate (see the paper, Section 8.3),
316 // or the EKF velocity alone until that estimate exists;
317 //
318 // with FRAME_TYPE_MARINE added to the frame-type enum, have_lateral_control()
319 // forwarded to the marine object, pre_arm_check() chained, and the new output
320 // functions k_azimuth1..k_azimuth6 added to SRV_Channel. Pivot point and thrust
321 // bias are read through two new auxiliary RC input functions and folded into
322 // tau before allocation (open loop) or into the mode's velocity demand
323 // (closed loop), as described in the paper.

```

C SITL physics model

code/vessel_sim_json.py: a minimal 3-DOF surface-vessel model for the ArduPilot SITL JSON interface. Mass, added mass and damping values are placeholders for a 1.6m hull and should be replaced by estimates from the actual model (a spin-down and a drift test give the damping; added mass can be taken from strip theory or simply as 10% in surge and 80% in sway of the displacement).

```

1  #!/usr/bin/env python3
2  """Minimal 3-DOF surface-vessel physics for ArduPilot SITL (JSON backend), microDP.
3
4  Run SITL with:  sim_vehicle.py -v Rover --model JSON -f rover --console --map
5  then start this script. SITL sends servo PWM on UDP 9002; we reply with the
6  vehicle state. The marine frame runs unchanged inside SITL, so the whole
7  chain (sticks -> modes -> allocator -> thrusters -> hull) is exercised.
8
9  Model (Fossen, 3-DOF manoeuvring):   $M \dot{u} + C(u)u + D(u)u = \tau$ 
10 State  $u = [u, v, r]$  (surge, sway, yaw rate),  $\eta = [x, y, \psi]$  in NED.
11 Thrusters are described with the same effector table as the frame's MFX_*
12 parameters (layout A of the paper). Angles are force directions:  $\text{ang} = 0$  is
13 wash aft, push ahead, the park position.
14 """
15 import json
16 import math
17 import socket
18 import struct
19 import time
20
21 import numpy as np
22
23 # ----- vessel -----
24 M = np.diag([18.0, 29.0, 3.2])          # mass + added mass [kg, kg, kg m^2]:
25                                         # 16 kg displacement, +10 % surge, +80 % sway
26 D_LIN = np.diag([1.5, 6.0, 1.2])       # linear damping
27 D_QUAD = np.diag([4.0, 25.0, 3.0])     # quadratic damping
28
29 # effector table: type, lx, ly, ang_deg, rng_deg, tf, tr, rate_deg_s,
30 # pwm slot for thrust (0-based servo channel), pwm slot for angle (or None)
31 EFFECTORS = [
32     dict(type=2, lx=-0.65, ly=-0.15, ang=0, rng=180, tf=5.0, tr=3.0,
33           rate=180, ch_t=0, ch_a=1),
34     dict(type=2, lx=-0.65, ly=+0.15, ang=0, rng=180, tf=5.0, tr=3.0,
35           rate=180, ch_t=2, ch_a=3),
36     dict(type=1, lx=+0.60, ly=0.00, ang=90, rng=0, tf=2.0, tr=2.0,
37           rate=0, ch_t=4, ch_a=None),
38 ]
39 for e in EFFECTORS:
40     e["angle"] = 0.0 # current steering angle relative to ang [rad]
41
42
43 # SERVOx_MIN/MAX/TRIM assumed 1000/2000/1500 for thrust and angle alike.
44 def pwm_to_thrust(pwm, tf, tr, expo=0.5):
45     x = max(-1.0, min(1.0, (pwm - 1500) / 500.0))
46     # invert the (1-expo)*x + expo*sqrt(x) shaping used by the allocator
47     mag = abs(x)
48     if expo > 0:
49         # solve (1-e) m + e sqrt(m) = mag for m by fixed point
50         m = mag
51         for _ in range(8):
52             if expo < 1:
53                 m = max(0.0, (mag - expo * math.sqrt(m)) / (1 - expo))
54             else:
55                 m = mag * mag
56         else:
57             m = mag
58     return (tf if x >= 0 else tr) * m * (1 if x >= 0 else -1)
59
60
61 def pwm_to_angle(pwm, rng_deg):
62     x = max(-1.0, min(1.0, (pwm - 1500) / 500.0))

```

```

63     return x * math.radians(max(rng_deg, 1))
64
65
66 def thruster_forces(pwm, dt):
67     tau = np.zeros(3)
68     for e in EFFECTORS:
69         T = pwm_to_thrust(pwm[e["ch_t"]], e["tf"], e["tr"])
70         centre = math.radians(e["ang"])
71         if e["type"] == 1:
72             a = centre
73         else:
74             target = pwm_to_angle(pwm[e["ch_a"]], e["rng"])
75             step = math.radians(e["rate"]) * dt
76             e["angle"] += max(-step, min(step, target - e["angle"]))
77             a = centre + e["angle"]
78         fx, fy = T * math.cos(a), T * math.sin(a)
79         tau += [fx, fy, e["lx"] * fy - e["ly"] * fx]
80     return tau
81
82
83 def step(eta, nu, tau, dt):
84     u, v, r = nu
85     m_u, m_v, _ = np.diag(M)
86     C = np.array([[0, 0, -m_v * v], [0, 0, m_u * u], [m_v * v, -m_u * u, 0]])
87     D = D_LIN + D_QUAD * np.abs(nu)
88     nu_dot = np.linalg.solve(M, tau - C @ nu - D @ nu)
89     nu = nu + nu_dot * dt
90     psi = eta[2]
91     c, s = math.cos(psi), math.sin(psi)
92     R = np.array([[c, -s, 0], [s, c, 0], [0, 0, 1]])
93     eta = eta + R @ nu * dt
94     return eta, nu, nu_dot
95
96
97 def main():
98     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
99     sock.bind(("", 9002))
100     eta, nu = np.zeros(3), np.zeros(3)
101     t = 0.0
102     while True:
103         data, addr = sock.recvfrom(1024)
104         magic, frame_rate, frame_count = struct.unpack("<HHI", data[:8])
105         pwm = struct.unpack("<16H", data[8:40])
106         if magic != 18458:
107             continue
108         dt = 1.0 / max(frame_rate, 1)
109         tau = thruster_forces(pwm, dt)
110         eta, nu, nu_dot = step(eta, nu, tau, dt)
111         t += dt
112         psi = eta[2]
113         vel_ned = [nu[0] * math.cos(psi) - nu[1] * math.sin(psi),
114                   nu[0] * math.sin(psi) + nu[1] * math.cos(psi), 0.0]
115         state = {
116             "timestamp": t,
117             "imu": {
118                 "gyro": [0.0, 0.0, float(nu[2])],
119                 "accel_body": [float(nu_dot[0] - nu[2] * nu[1]),
120                               float(nu_dot[1] + nu[2] * nu[0]), -9.80665],
121             },
122             "position": [float(eta[0]), float(eta[1]), 0.0],
123             "attitude": [0.0, 0.0, float(psi)],
124             "velocity": vel_ned,
125         }
126         sock.sendto(("\\n" + json.dumps(state) + "\\n").encode(), addr)
127
128
129 if __name__ == "__main__":
130     main()

```

D Parameter reference

Table 14: Parameters of the marine frame (working names).

Name	Default	Meaning
MFR_KX, MFR_KY, MFR_KN	6, 3, 2	Full-stick surge force, sway force (N) and yaw moment (N m).
MFR_BIAS	0	Null-space bias thrust as fraction of T_f , azimuth pairs only; off by default (Section 4.2).
MFR_BLANK	0.3	Thrust factor applied while a steerable effector is more than 45° from its target.
MFR_BTSPD	0.6	Water-relative speed (m/s) at which lateral fixed thrusters are fully faded out; 0 disables.
MFR_PRIO	1	Saturation policy: 1 yaw priority, 0 uniform scaling.
MFR_EXPO	0.5	Thrust-to-command shaping, 0 linear, 1 square root.
MFR_SPLT, MFR_SPSPD	0, 0.3	Propeller + rudder pairs only: split thrust per unit sway demand (fraction of T_f per N), and the water-relative speed (m/s) below which the split rule applies.
MFx_TYPE	0	0 off, 1 fixed, 2 azimuth, 3 propeller + rudder, 4 steerable jet, 5 cycloidal (resolved as an azimuth with an unrestricted sector and no slew limit).
MFx_LX, MFx_LY	0	Position relative to the CG (m), x forward, y starboard.
MFx_ANG	0	Fixed: force direction (deg, 0 ahead, 90 starboard). Steerable: sector centre and parking angle, in the same force convention (0 = wash aft, push ahead).
MFx_RNG	180	Half-width of the allowed sector (deg).
MFx_TF, MFx_TR	1, 1	Maximum thrust ahead and astern (N); TR=0 marks a unidirectional unit, such as a catamaran tunnel, entered with the force direction it has when washing outboard, toward the other hull.
MFx_RATE	180	Maximum steering slew (deg/s).
MFx_FSEC, MFx_FEFF	180, 1	Force-direction sector: half-width (deg) about ANG in which the force is fully effective, and effectiveness outside it (0–1), for a pod whose wash would strike a skeg or a neighbouring pod.
MFx_KT, MFx_KR, MFx_KU	0, 0, 0	Propeller + rudder only: rudder side force per unit thrust ahead, per unit thrust astern and per unit $u u $, at unit deflection; k_R is about $0.2 k_T$ for a Kort nozzle, more with a Becker flap rudder, near zero for an open rudder, measured at the bench. Sign convention: rudder to starboard gives force to port ahead; a positive k_R reverses that astern, as sternway does.
MFx_OUT	0	Output index n : thrust on Motor n , steering angle on Azimuth n , auxiliary (bucket, deploy) on MarineAux n .
MFx_DEPL, MFx_DSPD	0, 0	Deployable unit (0 fixed, 1 retractable) and the speed (m/s) above which it may not deploy.
MFx_FB	0	Feedback source: 0 none, 1 serial-servo angle, 2 limit switch, 3 ESC RPM telemetry.
HDx_*	—	Holding devices (spud leg, anchor, mooring): type, actuator output, feedback input, set and raise times.

E Indicative hardware

Table 15: Indicative hardware for phase 2, layout A. No prices; availability changes quickly.

Item	Notes
Flight controller	STM32H7-class ArduPilot board with at least 8 PWM outputs, 3 UARTs, CAN optional (Pixhawk 6C/6X, Cube Orange+, Matek H743 class).
GNSS	Two RTK receivers (u-blox F9P class) with helical or patch antennas ≥ 0.3 m apart on the centreline; RTK base or NTRIP.
Radio	EdgeTX-class handset with internal ELRS 2.4 GHz; ELRS receiver with CRSF to the autopilot. DP console per Section 5.4 from phase 3.
Motors	Low-Kv brushless (100–400 rpm/V) with 2:1 to 4:1 gear or toothed-belt reduction per unit (Section 6.7); brushed motors with gearbox acceptable.
ESCs	Bidirectional ESCs (brushless in 3D mode, or brushed) rated for the motors with margin; one per thruster.
Azimuth drive	Geared 180° servo with flip logic, or serial-bus multi-turn servo with feedback.
Wind sensor (phase 3)	Small ultrasonic anemometer, or vane potentiometer plus cup/propeller anemometer with pulse output, at the masthead or on a pole of its own, clear of any shadowing by superstructure, mast or antennas; read by the wind-vane library.
Water-speed log (optional)	Paddle-wheel or electromagnetic log on NMEA 0183, or the water-track of a small Doppler log; a cross-check of the current estimate in steady transit only, since thruster wash reaches it when manoeuvring.
DP console (phase 3)	Lilaas three-axis joystick and azimuth lever, heading wheel, switches, on a Leo Bodnar BU0836X USB interface into QGroundControl; later the Oceanomatics marine ground station (Section 7.3).
Companion (phase 4)	Raspberry Pi 5 or Jetson Orin Nano, camera for onboard vision, optional 2-D lidar; or an integrated Pixhawk + Raspberry Pi bundle (Holybro class, no affiliation) if it can be sealed at acceptable cost (Section 7.4).
Power	3S–4S LiPo, dedicated servo BEC, current sensor to the autopilot, main fuse.