Modeling and Control of the BlueROV2 Heavy: Dynamics, PID, and Model Predictive Control

Markus Buchholz

Abstract

This note works through the full chain that turns a physical underwater vehicle into something you can command with a line of code: the rigid-body and hydrodynamic equations that govern how a BlueROV2 Heavy actually moves, the kinematic bookkeeping that relates body-frame velocities to a position and orientation in the world, the allocation problem that turns a desired force and moment into eight individual thruster commands, and the final, easily forgotten step of turning a thruster force into the PWM pulse the electronics expect. On top of that model we build two controllers, a bank of six PID loops and a Model Predictive Controller, and we are explicit about what each buys you and what it costs. The aim is pedagogical: every matrix is introduced with the physical effect it represents before it is written down, and every simplification is derived rather than asserted, so that a student can rebuild the whole pipeline from first principles rather than memorize it.

Contents

1 Introduction

A BlueROV2 Heavy is a deceptively rich system to control. It has six degrees of freedom and eight thrusters, so it is over-actuated; it moves through a fluid that pushes back with forces that depend on velocity in complicated, sometimes quadratic ways; and the only handle you actually have on it is a PWM pulse width sent to eight electronic speed controllers. Between "I want the vehicle to move to this waypoint" and "pulse thruster 3 for 1.62 ms" sits an entire modeling and control pipeline, and if any one link in that chain is treated as a black box, the intuition needed to tune, debug, or extend the system disappears with it. This document builds that chain link by link:

- **Dynamics** (Sections 3–5): why the vehicle has more inertia than its dry mass suggests, why turning while moving forward generates forces that look like they come from nowhere, why it resists motion, and why it rights itself when disturbed.
- **Kinematics** (Section 6): how a velocity measured in the vehicle's own frame turns into a change of position and orientation in the world.
- **Actuation** (Sections 8–9): how a desired force and moment on the vehicle's center of mass is distributed across eight thrusters, and how each of those thruster forces is converted into the PWM signal that the hardware actually consumes.
- **Control** (Sections 10–11): first a simple, decentralized PID design, then a Model Predictive Controller that reasons about the coupled, constrained system as a whole.

One idea reappears throughout and is worth stating up front, because it explains why so many of the matrices below turn out simpler than they first look: the BlueROV2 Heavy is, by design, very close to symmetric fore-aft and port-starboard. Every general 6×6 matrix we write down is derived in full generality first, and then specialized using that symmetry. Seeing the general form makes clear why the simplification is legitimate rather than a shortcut, which is the difference between a formula you understand and a formula you have memorized.

2 Notation and Reference Frames

We use the two standard frames of marine vehicle dynamics: a body-fixed frame $\{b\}$, with origin at a convenient point on the vehicle (typically its geometric center) and axes pointing forward (x), to starboard (y), and down (z); and an inertial (world/NED) frame $\{n\}$. Position and orientation are expressed in the inertial frame, velocity in the body frame, following Fossen's convention [1]:

$$\boldsymbol{\eta} = \begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^T \in \mathbb{R}^6, \qquad \boldsymbol{\nu} = \begin{bmatrix} u & v & w & p & q & r \end{bmatrix}^T \in \mathbb{R}^6,$$

where (x, y, z) is the position of the origin of $\{b\}$ in $\{n\}$, (ϕ, θ, ψ) are the roll, pitch, and yaw Euler angles, (u, v, w) are the surge, sway, and heave linear velocities in $\{b\}$, and (p, q, r) are the roll, pitch, and yaw angular rates in $\{b\}$. It is worth fixing this in your head early: $\boldsymbol{\eta}$ lives in the world, $\boldsymbol{\nu}$ lives on the vehicle, and Section 6 is entirely about the bridge between the two.

Degree of Freedom	Pose / Angle	Velocity / Angular Velocity	Force / Moment
Surge	x_b	v_{xb}, u	X
Sway	y_b	v_{yb}, v	Y
Heave	z_b	v_{zb}, w	Z
Roll	ϕ	v_ϕ, p	K
Pitch	θ	v_θ, q	M
Yaw	ψ	v_ψ, r	N

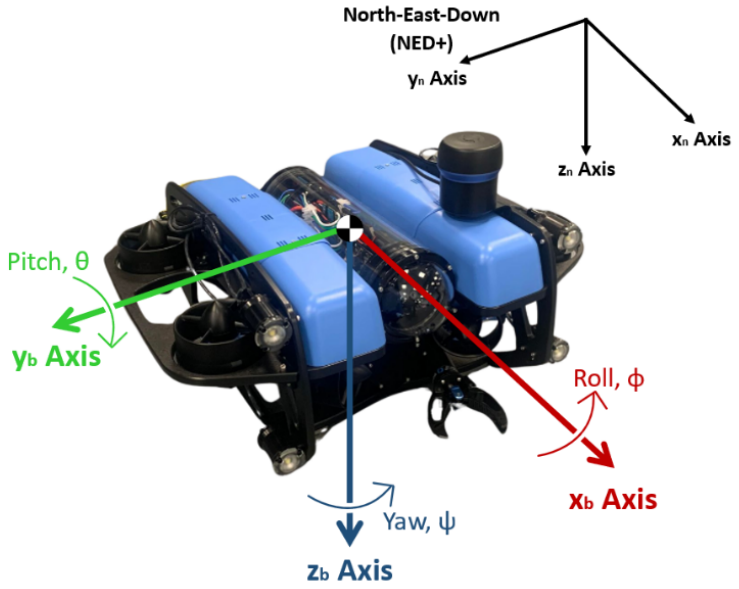

Figure 1: The BlueROV2 Heavy: four vectored thrusters in the horizontal plane and four vertical thrusters, giving full but over-actuated control authority over all six degrees of freedom.

3 Rigid-Body Dynamics

Ignore the water for a moment and treat the BlueROV2 as a rigid body coasting through empty space. Newton and Euler's equations for such a body, written compactly in body-frame coordinates, are

$$\mathbf{M}_{RB}\dot{\boldsymbol{\nu}} + \mathbf{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau}_{RB}, \tag{1}$$

and everything in this section is about what $\mathbf{M}_{RB}$ and $\mathbf{C}_{RB}$ actually are.

3.1 Rigid-body mass matrix

If the body-fixed origin coincides with the center of gravity (CG) and the body axes are principal axes of inertia, the mass matrix is simply $\mathrm{diag}(m, m, m, I_{xx}, I_{yy}, I_{zz})$: moving in any translational direction meets resistance m, rotating about any axis meets resistance I. In practice the origin of $\{b\}$ is chosen for convenience (often the geometric center of the frame) and does not exactly coincide with the CG, which sits at an offset $\mathbf{r}_g = (x_g, y_g, z_g)$. That offset couples translation and rotation: accelerating the origin forward while the CG sits above it also imparts an angular acceleration, in exactly the same way that pushing

a broom handle off-center makes it spin as well as slide. The general rigid-body mass matrix, still assuming principal axes, is

$$\mathbf{M}_{RB} = \begin{bmatrix} m & 0 & 0 & 0 & mz_g & -my_g \\ 0 & m & 0 & -mz_g & 0 & mx_g \\ 0 & 0 & m & my_g & -mx_g & 0 \\ 0 & -mz_g & my_g & I_{xx} & 0 & 0 \\ mz_g & 0 & -mx_g & 0 & I_{yy} & 0 \\ -my_g & mx_g & 0 & 0 & 0 & I_{zz} \end{bmatrix}. \tag{2}$$

For the BlueROV2 Heavy, the frame is built to be symmetric enough that the CG sits almost exactly on the body z-axis through the chosen origin, and typically also close to $z = 0$ once the origin is placed at the CG by convention. Setting $x_g = y_g = z_g = 0$ collapses (2) to the diagonal form $\mathbf{M}_{RB} = \mathrm{diag}(m, m, m, I_{xx}, I_{yy}, I_{zz})$. This is the version you should use for day-to-day simulation; (2) is what you fall back on if you ever mount enough extra payload off-center that the assumption stops holding.

3.2 Rigid-body Coriolis and centripetal forces

Even a rigid body with no external forces acting on it does not, in general, move in a straight line when described in a *rotating* frame: spin while translating and the body-frame description of your own momentum appears to twist underneath you. That apparent force is captured by $\mathbf{C}_{RB}(\boldsymbol{\nu})$, which for the mass matrix above is

$$\mathbf{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu} = \begin{bmatrix} 0 & 0 & 0 & 0 & mz_g & -my_g \\ 0 & 0 & 0 & -mz_g & 0 & mx_g \\ 0 & 0 & 0 & my_g & -mx_g & 0 \\ 0 & -mz_g & my_g & 0 & 0 & 0 \\ mz_g & 0 & -mx_g & 0 & 0 & 0 \\ -my_g & mx_g & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{3}$$

Physically this is the same phenomenon as the Coriolis effect that deflects long-range projectiles on the rotating Earth, just applied to the vehicle's own body-fixed frame rather than the planet's. With $x_g = y_g = z_g = 0$, (3) vanishes for the translational rows entirely and what remains is the ordinary gyroscopic coupling between angular rates; for a torpedo-shaped, near-axisymmetric vehicle like the BlueROV2 this term is small at the speeds the vehicle typically operates at, but it is not something you should delete from the model without first checking that assumption for your own hardware configuration.

4 Hydrodynamic Effects

Put the vehicle back in the water and three more physical effects appear, none of which exist for a body moving through vacuum: added mass, damping, and restoring forces. Each has a clean physical story.

4.1 Added mass

When the BlueROV2 accelerates, it is not just the vehicle's own mass that must be accelerated: the water immediately around the hull has to be pushed out of the way and

accelerated along with it, because the vehicle and the fluid cannot occupy the same space at the same instant. The reaction of that displaced fluid on the vehicle behaves, from the vehicle's point of view, exactly like extra inertia. This is added mass, and it is why a BlueROV2 responds more sluggishly to a given thrust command underwater than the same thrust would if the vehicle were in air. In body-frame coordinates it enters the equations of motion as another mass-like matrix,

$$\mathbf{M}_A = -\mathrm{diag}\big(X_{\dot{u}},\, Y_{\dot{v}},\, Z_{\dot{w}},\, K_{\dot{p}},\, M_{\dot{q}},\, N_{\dot{r}}\big). \tag{4}$$

The minus sign in front looks odd until you know the convention: by the SNAME notation used throughout marine hydrodynamics, a term like $X_{\dot{u}}$ is itself already negative (an added-mass *force* opposes acceleration, so the coefficient carries that sign), and the leading minus flips it back into a positive contribution to the total inertia, as physically it must be. For a slender, roughly cylindrical body of mass $\bar{m}$, length $\bar{L}$, radius $\bar{r}$, in a fluid of density ρ, strip theory gives the rough estimates

$$\begin{aligned} &X_{\dot{u}} \approx -0.1\,\bar{m}, && Y_{\dot{v}} \approx -\pi\rho\bar{r}^2\bar{L}, && Z_{\dot{w}} \approx -\pi\rho\bar{r}^2\bar{L}, \\ &K_{\dot{p}} \approx 0, && M_{\dot{q}} \approx -\tfrac{1}{12}\pi\rho\bar{r}^2\bar{L}^3, && N_{\dot{r}} \approx -\tfrac{1}{12}\pi\rho\bar{r}^2\bar{L}^3. \end{aligned} \tag{5}$$

Notice the asymmetry these formulas predict, and which matches experience with the real vehicle: added mass in surge ($X_{\dot{u}}$) is small because the vehicle is slender nose-first, while added mass in sway and heave ($Y_{\dot{v}}, Z_{\dot{w}}$) is much larger because moving sideways or vertically has to shove a much larger cross-section of water out of the way. This is precisely why a BlueROV2 accelerates crisply forward but feels noticeably heavier when you try to strafe it sideways.

4.2 Hydrodynamic Coriolis and centripetal forces

The added fluid mass has its own momentum, and that momentum reacts back on the vehicle whenever it rotates, in the same way the rigid-body mass produced $\mathbf{C}_{RB}$ above. Written in terms of the added-mass coefficients and the relative velocity $\boldsymbol{\nu}_r$ (Section 5 explains why it is the *relative* velocity, not $\boldsymbol{\nu}$ itself, that belongs here),

$$\mathbf{C}_A(\boldsymbol{\nu}_r) = \begin{bmatrix} 0 & 0 & 0 & 0 & -Z_{\dot{w}}w_r & Y_{\dot{v}}v_r \\ 0 & 0 & 0 & Z_{\dot{w}}w_r & 0 & -X_{\dot{u}}u_r \\ 0 & 0 & 0 & -Y_{\dot{v}}v_r & X_{\dot{u}}u_r & 0 \\ Z_{\dot{w}}w_r & -Y_{\dot{v}}v_r & X_{\dot{u}}u_r & 0 & -N_{\dot{r}}r & M_{\dot{q}}q \\ -Y_{\dot{v}}v_r & X_{\dot{u}}u_r & 0 & N_{\dot{r}}r & 0 & -K_{\dot{p}}p \\ X_{\dot{u}}u_r & 0 & -Z_{\dot{w}}w_r & -M_{\dot{q}}q & K_{\dot{p}}p & 0 \end{bmatrix}. \tag{6}$$

This term is what makes, for instance, a yawing turn at speed feel different from a stationary yaw: the water entrained by the forward motion resists the turn in a way that a simple diagonal damping term cannot capture.

4.3 Damping

Moving through a real, viscous fluid meets resistance that has no analogue in vacuum: skin friction from the fluid dragging along the hull surface, and form drag from the pressure difference between the front and back of the vehicle. The first grows roughly linearly with speed at the low Reynolds numbers relevant to slow ROV motion; the second grows

roughly with the square of speed once the flow starts to separate. A model that keeps both terms,

$$\mathbf{D}(\boldsymbol{\nu}_r) = \mathbf{D}_L + \mathbf{D}_{NL}(\boldsymbol{\nu}_r), \qquad \mathbf{D}_L = -\mathrm{diag}(X_u, Y_v, Z_w, K_p, M_q, N_r), \tag{7}$$

$$\mathbf{D}_{NL}(\boldsymbol{\nu}_r) = -\mathrm{diag}\big(X_{u|u|}|u_r|,\ Y_{v|v|}|v_r|,\ Z_{w|w|}|w_r|,\ K_{p|p|}|p|,\ M_{q|q|}|q|,\ N_{r|r|}|r|\big), \tag{8}$$

covers both regimes reasonably well across the speed range a BlueROV2 actually operates in. In full generality, drag in one direction can couple into forces in another (a cross-flow term like X_{uv}), but a hull that is symmetric port-to-starboard and roughly symmetric fore-to-aft makes almost all of those cross terms vanish, leaving the purely diagonal form above as the practical model.

4.4 Restoring forces and moments

The last hydrodynamic term has nothing to do with velocity at all: it is the net effect of gravity and buoyancy, and it is why a BlueROV2 rights itself when tilted rather than needing active control just to stay upright. Let $W = mg$ be the vehicle's weight and $B = \rho g \nabla$ its buoyancy (∇ the displaced volume), acting respectively at the center of gravity (x_g, y_g, z_g) and center of buoyancy (x_b, y_b, z_b). The general nonlinear restoring vector, with the body z-axis pointing down, is

$$\mathbf{g}(\boldsymbol{\eta}) = \begin{bmatrix} (W-B)\sin\theta \\ -(W-B)\cos\theta\sin\phi \\ -(W-B)\cos\theta\cos\phi \\ -(y_g W - y_b B)\cos\theta\cos\phi + (z_g W - z_b B)\cos\theta\sin\phi \\ (z_g W - z_b B)\sin\theta + (x_g W - x_b B)\cos\theta\cos\phi \\ -(x_g W - x_b B)\cos\theta\sin\phi - (y_g W - y_b B)\sin\theta \end{bmatrix}. \tag{9}$$

Two things are worth pausing on here. First, W and B are already forces (newtons); there is no extra factor of g anywhere else in (9) — a common transcription error is to multiply by g a second time, which breaks the units. Second, the roll and pitch restoring moments are driven by $(z_g W - z_b B)$: if the center of buoyancy sits above the center of gravity, as it does by design on the BlueROV2 (the foam floats are mounted high, the battery and electronics sit low), tilting the vehicle creates a moment that pushes it back upright, exactly like a weighted keel on a sailboat. This is passive, needs no controller, and is one of the more pleasant properties of the platform. Third, and practically important: if the vehicle is trimmed to neutral buoyancy, $W = B$, and the entire vector (9) vanishes identically. Careful ballasting is therefore not a cosmetic detail — it directly zeroes out a term that would otherwise have to be fought by the controller at every single time step.

For small roll and pitch angles, as is often the case for a slow-moving inspection vehicle, $\sin\theta \approx \theta$, $\cos\theta \approx 1$, and (9) linearizes to

$$\mathbf{g}(\boldsymbol{\eta}) \approx \begin{bmatrix} (W-B)\theta \\ -(W-B)\phi \\ -(W-B) \\ (z_b B - z_g W) \\ (x_g W - x_b B) \\ 0 \end{bmatrix}, \tag{10}$$

which is convenient for the small-angle PID analysis of Section 10, though the controller should still be commanded from the full nonlinear form (9) whenever the vehicle is expected to operate outside near-level attitudes.

5 Complete Equations of Motion

Assembling Sections 3 and 4, and writing $\boldsymbol{\tau}$ for the control wrench delivered by the thrusters and $\boldsymbol{\tau}_{ext}$ for everything else (currents modeled as a force, tether drag, collisions), the full 6-DOF equations of motion are

$$\underbrace{\mathbf{M}_{RB}\dot{\boldsymbol{\nu}} + \mathbf{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu}}_{\text{rigid body}} + \underbrace{\mathbf{M}_{A}\dot{\boldsymbol{\nu}} + \mathbf{C}_{A}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{g}(\boldsymbol{\eta})}_{\text{hydrodynamic}} = \boldsymbol{\tau} + \boldsymbol{\tau}_{ext}, \tag{11}$$

where the relative velocity

$$\boldsymbol{\nu}_r = \boldsymbol{\nu} - \boldsymbol{\nu}_c \tag{12}$$

subtracts the ambient current $\boldsymbol{\nu}_c$ (expressed in the body frame) from the vehicle's own velocity. The reason $\boldsymbol{\nu}_r$ and not $\boldsymbol{\nu}$ appears in the hydrodynamic terms is physical, not notational: the water does not know or care how fast the vehicle is moving over the seabed, only how fast the vehicle is moving *relative to the water itself*. Two matrices multiply the same acceleration $\dot{\boldsymbol{\nu}}$, so it is standard to define the total mass matrix $\mathbf{M} = \mathbf{M}_{RB} + \mathbf{M}_A$ and write (11) as an explicit acceleration,

$$\dot{\boldsymbol{\nu}} = \mathbf{M}^{-1}\Big(\boldsymbol{\tau} + \boldsymbol{\tau}_{ext} - \mathbf{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{C}_{A}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r - \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r - \mathbf{g}(\boldsymbol{\eta})\Big), \tag{13}$$

which is the form you actually integrate forward in time (Section 7).

6 Kinematics: From Body Velocity to World Pose

Equation (13) tells you how the body-frame velocity changes; it says nothing yet about where the vehicle actually is in the world. That is the job of the kinematic equations,

$$\dot{\mathbf{p}} = \mathbf{R}(\boldsymbol{\eta})\,\boldsymbol{\nu}_1, \qquad \dot{\boldsymbol{\eta}}_2 = \mathbf{T}(\boldsymbol{\eta}_2)\,\boldsymbol{\nu}_2, \tag{14}$$

with $\mathbf{p} = (x, y, z)$, $\boldsymbol{\nu}_1 = (u, v, w)$, $\boldsymbol{\eta}_2 = (\phi, \theta, \psi)$, and $\boldsymbol{\nu}_2 = (p, q, r)$. Using the standard z-y-x (yaw-pitch-roll) Euler angle convention, the rotation matrix carrying a vector from the body frame to the inertial frame is

$$\mathbf{R}(\boldsymbol{\eta}) = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix}, \qquad c_\alpha = \cos\alpha,\ s_\alpha = \sin\alpha, \tag{15}$$

and the angular-rate transformation, relating body-frame angular velocity to the rate of change of the Euler angles, is

$$\mathbf{T}(\boldsymbol{\eta}_2) = \begin{bmatrix} 1 & s_\phi \tan\theta & c_\phi \tan\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi / c_\theta & c_\phi / c_\theta \end{bmatrix}. \tag{16}$$

It is worth being deliberate about $\mathbf{R}$ in (15): it must map *body-frame* velocity to *world-frame* velocity, and it is easy, when copying formulas between references, to accidentally pick up the transpose $\mathbf{R}^T$ instead (which maps the other way, world to body) — the two look superficially similar and the mistake will not throw an error, it will simply make the simulated vehicle drift in the wrong direction under yaw. (15) is the correct one for use in (14).

The full kinematic transformation is the block-diagonal combination

$$\dot{\boldsymbol{\eta}} = \mathbf{J}(\boldsymbol{\eta})\,\boldsymbol{\nu}, \qquad \mathbf{J}(\boldsymbol{\eta}) = \begin{bmatrix} \mathbf{R}(\boldsymbol{\eta}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{T}(\boldsymbol{\eta}_2) \end{bmatrix}. \tag{17}$$

Notice that $\mathbf{T}$ has $\cos\theta$ in two denominators: as $\theta \to \pm 90^\circ$ the transformation blows up. This is gimbal lock, the same physical phenomenon that locks a mechanical gimbal when two of its three rotation axes become aligned — at $\theta = \pm 90^\circ$ the roll and yaw axes coincide and a degree of freedom is lost from the Euler-angle representation, even though the vehicle itself still has all six degrees of freedom available. A BlueROV2 pitched to vertical is an edge case worth guarding against numerically (clamp θ away from $\pm 90^\circ$, or switch to a quaternion representation) rather than an argument against Euler angles for the everyday, near-level operation this document assumes.

7 State-Space Model and Numerical Integration

Collect $\boldsymbol{\nu}$ and $\boldsymbol{\eta}$ into a single state $\mathbf{x} = [\boldsymbol{\nu};\ \boldsymbol{\eta}] \in \mathbb{R}^{12}$. Equations (13) and (17) together give a single first-order ODE,

$$\dot{\mathbf{x}} = f(\mathbf{x}, \boldsymbol{\tau}) = \begin{bmatrix} \mathbf{M}^{-1}\big(\boldsymbol{\tau} + \boldsymbol{\tau}_{ext} - \mathbf{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{C}_A(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r - \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r - \mathbf{g}(\boldsymbol{\eta})\big) \\ \mathbf{J}(\boldsymbol{\eta})\,\boldsymbol{\nu} \end{bmatrix}, \tag{18}$$

which is the object both a simulator and a model-based controller need to evaluate repeatedly. Because f is nonlinear, it is integrated numerically rather than solved in closed form. The classical fourth-order Runge-Kutta method (RK4) is a good default: instead of taking a single slope estimate at the start of the step, as Euler's method does, it averages four slope estimates taken at increasingly informed points across the step,

$$\begin{aligned} \mathbf{k}_1 &= f(\mathbf{x}_k,\ \boldsymbol{\tau}_k), \\ \mathbf{k}_2 &= f\Big(\mathbf{x}_k + \tfrac{\Delta t}{2}\mathbf{k}_1,\ \boldsymbol{\tau}_k\Big), \\ \mathbf{k}_3 &= f\Big(\mathbf{x}_k + \tfrac{\Delta t}{2}\mathbf{k}_2,\ \boldsymbol{\tau}_k\Big), \\ \mathbf{k}_4 &= f\big(\mathbf{x}_k + \Delta t\,\mathbf{k}_3,\ \boldsymbol{\tau}_k\big), \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \frac{\Delta t}{6}\big(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\big). \end{aligned} \tag{19}$$

The four evaluations cancel out the leading error terms that a single-slope method leaves behind, so for the same step size Δt, RK4 tracks the true trajectory of (18) far more faithfully than Euler integration, at the cost of four function evaluations instead of one. This is the integration scheme assumed everywhere below, including inside the Model Predictive Controller of Section 11, where the prediction model *is* (18) rolled forward with (19).

8 From Body Wrench to Individual Thrusters

Every controller in this document, PID or MPC, ultimately produces a single desired quantity: the wrench $\boldsymbol{\tau}_{des} = [F_x, F_y, F_z, M_x, M_y, M_z]^T$ that should act at the vehicle's origin. The BlueROV2 Heavy has no actuator that directly produces a wrench; it has

eight thrusters, each producing a scalar force f_i along its own fixed mounting direction. Turning $\boldsymbol{\tau}_{des}$ into eight numbers $f_1, \ldots, f_8$ is the control allocation problem, and it is a genuinely different problem from anything in Sections 3–7: those sections describe the vehicle's physics, this one describes its hardware layout.

8.1 Thruster configuration matrix

The BlueROV2 Heavy carries four thrusters in the horizontal plane, each canted at 45° to the vehicle's centerline, and four vertical thrusters. Each thruster contributes to the body wrench in proportion to its force f_i, its direction cosine, and its moment arm about the origin; stacking these contributions column by column gives the configuration matrix $\mathbf{B} \in \mathbb{R}^{6\times 8}$ such that

$$\boldsymbol{\tau} = \mathbf{B}\,\mathbf{f}, \qquad \mathbf{f} = [f_1, \ldots, f_8]^T. \tag{20}$$

For the standard BlueROV2 Heavy thruster layout,

$$\mathbf{B} = \begin{bmatrix} 0.7071 & 0.7071 & -0.7071 & -0.7071 & 0 & 0 & 0 & 0 \\ -0.7071 & 0.7071 & -0.7071 & 0.7071 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0.2180 & 0.2180 & -0.2180 & -0.2180 \\ 0 & 0 & 0 & 0 & 0.1200 & -0.1200 & 0.1200 & -0.1200 \\ -0.1888 & 0.1888 & 0.1888 & -0.1888 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{21}$$

Reading the matrix column by column tells the layout story directly: the first four columns are the vectored horizontal thrusters, and the factor $0.7071 = \cos 45^\circ = \sin 45^\circ$ is exactly why each one contributes equally to surge and sway; the ± 0.1888 m entries in the yaw row are those same thrusters' moment arm about the vertical axis, which is how four thrusters with no dedicated yaw actuator still give the vehicle full yaw authority. The last four columns are the vertical thrusters, contributing directly to heave (unit entries) and, through their 0.2180 m and 0.1200 m offsets from the origin, to roll and pitch.

8.2 Allocating a desired wrench to thruster forces

Equation (20) runs the wrong way for control: given a matrix, computing $\boldsymbol{\tau}$ from $\mathbf{f}$ is direct, but a controller has $\boldsymbol{\tau}_{des}$ and needs $\mathbf{f}$. Since $\mathbf{B}$ is 6×8 and full row rank, the system is underdetermined — there are infinitely many combinations of eight thruster forces that produce the same six-dimensional wrench, precisely because the vehicle is over-actuated. Among all solutions, the Moore-Penrose pseudo-inverse selects the one of minimum norm,

$$\mathbf{f} = \mathbf{B}^{+}\boldsymbol{\tau}_{des} = \mathbf{B}^T(\mathbf{B}\mathbf{B}^T)^{-1}\boldsymbol{\tau}_{des}. \tag{22}$$

Minimum norm is not an arbitrary tie-break: since thruster power draw grows with the square of thrust, the minimum-norm allocation is also, approximately, the minimum-power one, so (22) is doing something physically sensible, not just mathematically convenient.

9 From Thruster Force to PWM Command

Equation (22) still leaves you one step short of an actual command, and this is the step most often skipped in write-ups but not in real deployments. A BlueROV2's T200

thrusters are not commanded in newtons; they are commanded with a PWM pulse of width between 1100 μs (full reverse) and 1900 μs (full forward), centered on 1500 μs (zero thrust), the same convention used throughout radio-control hardware and, on the BlueROV2, sent to the Pixhawk-based flight controller running ArduSub. The relationship between pulse width and delivered thrust is neither linear nor symmetric: the propeller geometry produces more forward thrust than reverse thrust for the same pulse offset, and the whole curve scales with battery voltage, since a sagging battery under load delivers less thrust for the same command.

Blue Robotics actually publishes three different performance curves for the T200 as functions of PWM: thrust, draw current, and electrical power, each given as a family of curves parameterized by supply voltage (10 V through 20 V); the full set is available on the thruster's product page.[1] It matters which one is used here, because only one of them is the right object to invert. The controller's allocation stage, Eq. (22), produces a desired force f_i in newtons for each thruster, so the quantity that has to be inverted to recover a PWM value is specifically the *thrust curve*, $F_i(\text{PWM})$, evaluated at (or interpolated to) the battery's present voltage. The current and power curves describe something different: how many amps and watts that same commanded thrust costs to produce. They play no role in the force-to-PWM mapping itself, but they are not irrelevant, they are what you consult to size the battery and wiring for the vehicle's expected duty cycle, to set a current limit on the ESCs so a hard maneuver cannot brown out the electronics, and to sanity-check that a given mission profile's average power draw is within the pack's endurance. Conflating the two is a common mistake: feeding a current or power value into what should be a thrust inversion produces a PWM command with the right order of magnitude but the wrong physical meaning, since current and power both grow roughly with $|F_i|^{3/2}$ near the origin rather than linearly, so the resulting command is systematically off except by coincidence near one operating point.

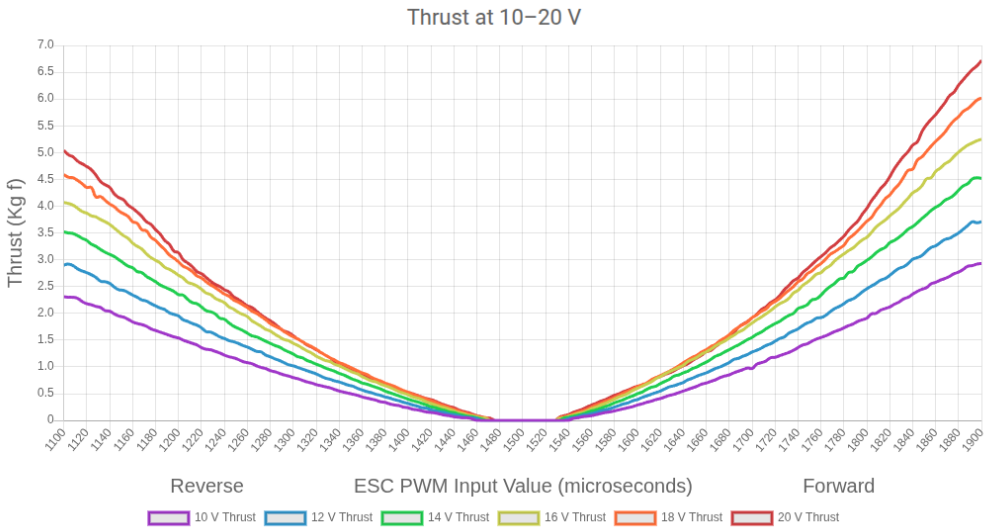

Figure 2: Manufacturer thrust-vs-PWM calibration curves for the T200, shown across supply voltages. This is the curve that Eq. (23) fits and that must be inverted (by lookup) to turn a desired thruster force f_i into a PWM_i command; the current and power curves published alongside it answer a different, electrical-budget question and are not part of this mapping. Source: Blue Robotics T200 product page.

[1] `https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/`

With that settled, a convenient way to work with the thrust curve is a cubic fit around the neutral point, separated by direction,

$$F_i(\delta) = \begin{cases} c_1^+ \delta + c_3^+ \delta^3, & \delta \geq 0 \\ c_1^- \delta + c_3^- \delta^3, & \delta < 0 \end{cases}, \qquad \delta = \mathrm{PWM}_i - 1500 \ [\mu s], \tag{23}$$

with $(c_1^+, c_3^+) \neq (c_1^-, c_3^-)$ fit from the manufacturer's thrust-vs-PWM data at the operating voltage, and δ saturating at $\pm 400\,\mu s$. What the controller actually needs is the inverse map, force to PWM, and (23) does not invert cleanly in closed form once both a linear and a cubic term are present. In practice this is not solved algebraically at all: the manufacturer's thrust calibration data is stored as a lookup table of (force, PWM) pairs at the vehicle's nominal operating voltage, and the desired force from (22) is converted to a command by table interpolation (linear or monotonic-cubic interpolation both work well, since the underlying curve is smooth and monotonic in each direction), which is both simpler and more robust than numerically inverting a fitted polynomial. If the vehicle's battery voltage is monitored at run time, the same lookup can be made voltage-aware by interpolating between the manufacturer's curves at the two bracketing voltages rather than committing to a single nominal one, which keeps the mapping accurate as the pack drains over a long dive. Once each f_i has been converted to a PWM_i, the eight values are written to the vehicle's motor output channels — over MAVLink as a servo/motor output message from ArduSub, or, in simulation, as the direct thruster command Stonefish or Gazebo expect — closing the loop from a mathematical wrench all the way down to the signal on the wire.

Table 1: The full command chain, restated as a single pipeline

Stage	Quantity
Controller (PID or MPC)	desired wrench $\boldsymbol{\tau}_{des} \in \mathbb{R}^6$
Allocation, Eq. (22)	desired thruster forces $\mathbf{f} \in \mathbb{R}^8$
Saturation, Section 10	clipped forces within actuator limits
Force-to-PWM, Eq. (23) (inverted via lookup)	$\mathrm{PWM}_i \in [1100, 1900]\,\mu s$
Hardware / simulator	eight ESC pulses $\rightarrow$ eight propeller thrusts

10 PID Control

The most direct way to control the vehicle is to treat each of the six degrees of freedom as an independent single-input single-output system and close a loop around each one. For a degree of freedom $i \in \{x, y, z, \phi, \theta, \psi\}$ with setpoint r_i and measured (or estimated) state $y_i(t)$, define the error

$$e_i(t) = r_i - y_i(t), \tag{24}$$

and let the controller produce

$$u_i(t) = K_{p_i}\, e_i(t) + K_{i_i} \int_0^t e_i(\tau)\, d\tau + K_{d_i}\, \frac{de_i(t)}{dt}. \tag{25}$$

Each term has a distinct physical role, and understanding it makes tuning far less of a guessing game. The proportional term is a virtual spring: it pushes back on the vehicle

with a force proportional to how far it is from where it should be, exactly like a real spring connecting the vehicle to its setpoint. The integral term exists because a spring alone cannot hold a static offset against a constant disturbance — a vehicle slightly negatively buoyant will hang below its depth setpoint forever under P control alone, since at equilibrium the spring force must exactly balance the constant bias, which requires a nonzero steady-state error. The integral term accumulates that steady error over time and keeps adding restoring force until the error is driven to zero; it is what actually rejects a constant bias like imperfect ballasting or a steady current. The derivative term looks at the rate at which the error is changing and reacts before the error itself gets there, which is what prevents the proportional term's spring from overshooting and oscillating around the setpoint — it behaves like a damper acting in parallel with the spring.

The six outputs $u_i(t)$ stack into the desired wrench $\boldsymbol{\tau}_{des} = [u_x, u_y, u_z, u_\phi, u_\theta, u_\psi]^T$, which then goes through exactly the allocation-and-PWM pipeline of Sections 8–9. Representative gains and output limits, tuned by the usual iterative process (raise K_p until the response is brisk but not oscillatory, add just enough K_d to damp the resulting overshoot, add a small K_i only if a steady-state offset persists), are given in Table 2; they are a starting point for simulation, not a substitute for retuning on the real vehicle.

Table 2: Representative PID gains and output saturation limits, per DOF

DOF	K_p	K_i	K_d
Surge (x)	0.1	0.005	1.0
Sway (y)	0.1	0.05	1.0
Heave (z)	0.5	0.0	2.0
Roll (ϕ)	1.0	0.0	0.3
Pitch (θ)	1.0	0.0	0.3
Yaw (ψ)	1.0	0.0	0.3

10.1 Where six independent loops fall short

Six SISO loops are simple to implement and simple to reason about individually, but they are blind to exactly the coupling terms Sections 3–4 spent so much effort deriving. The surge loop has no idea that a sway motion is underway and generating a Coriolis moment on yaw through $\mathbf{C}_A$; each loop reacts only after the coupling has already disturbed its own DOF, rather than anticipating it. Nor does a PID loop know that the allocation in (22) can saturate: if commanded forces exceed thruster limits, each loop keeps integrating error as though the actuator obediently delivered whatever was asked, which is the classic recipe for integrator windup. These are not flaws in the tuning; they are structural limits of treating a genuinely multivariable, constrained system as six unrelated ones, and they are exactly what motivates Section 11.

11 Model Predictive Control

Where a PID loop reacts to the error that has already appeared, a Model Predictive Controller uses the model built in Sections 3–7 to look ahead: at every control step it simulates several seconds into the future for a range of candidate control sequences, scores each one against how well it tracks the reference and how much control effort and actuator

margin it uses, and applies only the first step of the best sequence found before repeating the whole calculation at the next time step. This section builds that recipe in order: the prediction model, the cost, the constraints, the resulting optimization problem, and finally how the solution reconnects to the same allocation and PWM pipeline used by the PID controller.

11.1 Prediction model

The controller predicts forward using the same nonlinear model derived above, discretized with the sample time Δt of the control loop. Using the state $\mathbf{x} = [\boldsymbol{\nu}; \boldsymbol{\eta}]$ and control $\mathbf{u} = \boldsymbol{\tau}$ from Section 7, one RK4 step of (19) defines the discrete map

$$\mathbf{x}_{k+1} = F(\mathbf{x}_k, \mathbf{u}_k), \tag{26}$$

which is evaluated repeatedly to roll a candidate control sequence $\mathbf{u}_0, \ldots, \mathbf{u}_{N-1}$ forward over a prediction horizon of N steps. Written out explicitly rather than hidden inside F, one step is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \begin{pmatrix} \mathbf{J}(\boldsymbol{\eta}_k)\, \boldsymbol{\nu}_k \\ \mathbf{M}^{-1}\big(\boldsymbol{\tau}_k - \mathbf{C}(\boldsymbol{\nu}_k)\boldsymbol{\nu}_k - \mathbf{D}(\boldsymbol{\nu}_k)\boldsymbol{\nu}_k - \mathbf{g}(\boldsymbol{\eta}_k)\big) \end{pmatrix}, \tag{27}$$

a first-order (Euler) discretization of the same equations of motion as (18). This is deliberately the cheaper cousin of (19): RK4 is the right choice for the simulator that stands in for the real vehicle, since it has to stay accurate over a long run with no correction, but inside an MPC the model is re-solved and re-linearized every single control cycle, its own error is corrected almost immediately, and (27) only has to stay accurate over one short horizon rather than an entire mission — for a prediction horizon evaluated at every step of an online optimization, one function evaluation per step instead of RK4's four is usually the better trade, and (27) is what most real-time underwater MPC implementations actually run.

11.2 Cost function and constraints

Given a reference trajectory $\mathbf{x}_k^{ref}$ over the horizon (a hold-position setpoint, a waypoint sequence, or, as in Section 10's motivating example, a trajectory that keeps the vehicle's attention on a region of interest), the controller scores a candidate sequence $\mathbf{U} = (\mathbf{u}_0, \ldots, \mathbf{u}_{N-1})$ by the quadratic cost

$$J(\mathbf{x}_0, \mathbf{U}) = \sum_{k=0}^{N-1} \Big[(\mathbf{x}_k - \mathbf{x}_k^{ref})^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_k^{ref}) + \mathbf{u}_k^T \mathbf{R}\, \mathbf{u}_k + \Delta \mathbf{u}_k^T \mathbf{R}_\Delta\, \Delta \mathbf{u}_k \Big] + (\mathbf{x}_N - \mathbf{x}_N^{ref})^T \mathbf{P} (\mathbf{x}_N - \mathbf{x}_N^{ref}), \tag{28}$$

where $\Delta \mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{k-1}$ penalizes abrupt changes in commanded wrench (which, physically, is what keeps the allocation of Section 8 from demanding a thruster reversal every sample), $\mathbf{Q} \succeq 0$ and $\mathbf{P} \succeq 0$ weight tracking accuracy, and $\mathbf{R} \succ 0$ weights control effort. Equation (28) is written with the quadratic forms spelled out; much of the MPC literature instead writes it with the weighted-norm shorthand $\|\mathbf{z}\|_{\mathbf{W}}^2 := \mathbf{z}^T \mathbf{W} \mathbf{z}$, i.e.

$$J = \sum_{k=0}^{N-1} \Big(\|\mathbf{x}_k - \mathbf{x}_k^{ref}\|_{\mathbf{Q}}^2 + \|\mathbf{u}_k\|_{\mathbf{R}}^2 \Big) + \|\mathbf{x}_N - \mathbf{x}_N^{ref}\|_{\mathbf{P}}^2, \tag{29}$$

which is exactly the same cost with the $\Delta\mathbf{u}_k$ term dropped for brevity; the terminal weight is sometimes named $\mathbf{Q}_N$ rather than $\mathbf{P}$ in other references, purely a naming choice, not a different quantity. The optimization is subject to the prediction dynamics and to the physical limits of the vehicle,

$$\mathbf{x}_{k+1} = F(\mathbf{x}_k, \mathbf{u}_k), \qquad \mathbf{x}_0 = \hat{\mathbf{x}}(t), \tag{30}$$

$$\mathbf{u}_{\min} \le \mathbf{u}_k \le \mathbf{u}_{\max}, \qquad \Delta\mathbf{u}_{\min} \le \Delta\mathbf{u}_k \le \Delta\mathbf{u}_{\max}, \tag{31}$$

where $\hat{\mathbf{x}}(t)$ is the current state estimate from the navigation filter, and $\mathbf{u}_{\min}, \mathbf{u}_{\max}$ are the wrench limits consistent with the per-thruster saturation $|f_i| \le f_{\max}$ mapped back through $\mathbf{B}$ — this is the constraint that lets the controller *see* actuator saturation coming several steps in advance, rather than discovering it only after commanding something the thrusters cannot deliver, which is the single biggest practical advantage MPC has over the PID design of Section 10.

11.3 From nonlinear program to a solvable problem

Minimizing (28) subject to (30)–(31) is, as written, a nonconvex nonlinear program, because F inherits the Coriolis and quadratic-damping nonlinearities of Section 4. Solving that exactly at, say, 20 Hz on an embedded computer is not realistic, so in practice the nonlinear model is linearized about a nominal trajectory $(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)$ — the previous solution, shifted forward by one step, serves well as that nominal trajectory —

$$\delta\mathbf{x}_{k+1} \approx \mathbf{A}_k\,\delta\mathbf{x}_k + \mathbf{B}_k\,\delta\mathbf{u}_k, \qquad \mathbf{A}_k = \left.\frac{\partial F}{\partial \mathbf{x}}\right|_{\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k}, \quad \mathbf{B}_k = \left.\frac{\partial F}{\partial \mathbf{u}}\right|_{\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k}, \tag{32}$$

which turns (28)–(31) into a linear-time-varying quadratic program that a standard QP solver returns in a few milliseconds. Re-linearizing every control cycle around the vehicle's actual, current trajectory (rather than linearizing once around hover and never updating) is what keeps this approximation valid across the full range of motions a survey or inspection mission actually needs.

11.4 Receding horizon algorithm

Putting the pieces together, one control cycle of the MPC runs as follows.

1. Read the current state estimate $\hat{\mathbf{x}}(t)$ from the navigation solution (depth sensor, IMU, and DVL or, in the Stonefish simulator, the simulated state).
2. Update the reference $\{\mathbf{x}_k^{ref}\}_{k=0}^{N}$ over the horizon from the mission layer.
3. Linearize the model, Eq. (32), about the previous solution shifted one step forward (or about $\hat{\mathbf{x}}(t)$ on the first iteration).
4. Solve the resulting QP for $\mathbf{U}^\star = (\mathbf{u}_0^\star, \ldots, \mathbf{u}_{N-1}^\star)$, warm-started from the previous solution for speed.
5. Apply only $\boldsymbol{\tau}_{des} = \mathbf{u}_0^\star$ — discard the rest of the sequence, since it will be recomputed with better information next step. Pass $\boldsymbol{\tau}_{des}$ through the same allocation and PWM chain used by the PID controller (Sections 8–9, summarized in Table 1).

6. Advance one sample and return to step 1.

This is the receding-horizon principle: the controller commits to nothing beyond the very next instant, which is precisely what makes it robust to the fact that its own model is an approximation and its own prediction of the future will keep being wrong in small ways that the next re-solve simply corrects for.

11.5 From the optimal wrench to an actual control signal

It is worth being precise about what the solve in step 4 actually hands you, because it is easy to conflate the optimizer's internal bookkeeping with its physical output. The QP returns an entire predicted trajectory, both the control sequence $\mathbf{U}^\star$ and the state sequence $\mathbf{x}_1^\star, \ldots, \mathbf{x}_N^\star$ that sequence is expected to produce through (27). The predicted states are used only inside the controller, to evaluate the cost and to warm-start the next linearization; *none of them are ever sent to the vehicle.* The only quantity that leaves the MPC block on a given control cycle is the first control action,

$$\boldsymbol{\tau}_{des} = \mathbf{u}_0^\star \in \mathbb{R}^6, \tag{33}$$

a desired body-frame wrench, exactly the same object a PID loop produces in Section 10. This is a body wrench, not a thruster command, and it is not yet anything a thruster or an ESC can act on. It has to travel through precisely the same pipeline built in Sections 8–9, and it is worth writing that chain out explicitly one more time with $\boldsymbol{\tau}_{des}$ from (33) as the starting point:

$$\begin{aligned}
\mathbf{f}_k &= \mathbf{B}^{+}\boldsymbol{\tau}_{des}, && \text{allocate the wrench to eight thruster forces, Eq. (22),}\\
\mathbf{f}_k^{sat} &= \mathrm{clip}(\mathbf{f}_k,\, -f_{\max},\, f_{\max}), && \text{saturate to what the T200s can physically deliver,}\\
\mathrm{PWM}_{k,i} &= F_i^{-1}\big(f_{k,i}^{sat}\big),\ i = 1, \ldots, 8, && \text{invert the calibration curve, Eq. (23), by lookup,}
\end{aligned}$$

after which $\mathrm{PWM}_k \in [1100, 1900]^8\,\mu s$ is what is actually written to the eight motor outputs. Every control cycle therefore produces one predicted trajectory used internally and discarded, and one eight-dimensional PWM vector sent to hardware; a common and confusing mistake is to report the predicted next state $\mathbf{x}_1^\star$ as if it were "the MPC output" — it is a forecast, not a command, and sending it anywhere would not mean anything to the vehicle's ESCs. Table 1 already lists this chain; the point of repeating it here is that it is the same chain whether $\boldsymbol{\tau}_{des}$ came from six PID loops or from an optimizer, which is exactly what Section 12 means when it says the two controllers differ only in how $\boldsymbol{\tau}_{des}$ is decided.

11.6 Solver notes

None of the steps above require an exotic solver. A dense QP of the size typical for a 1–3 second horizon at a 10–20 Hz control rate (a few hundred decision variables) is comfortably within reach of an active-set or interior-point QP solver running on the same computer that already runs the vehicle's navigation stack; the linearize-solve-apply cycle above is essentially the real-time iteration scheme used throughout the nonlinear MPC literature, and off-the-shelf tools (CasADi with an embedded QP backend, or acados) implement it directly without requiring the modeling work above to be redone by hand — the point of this section is that once it is redone by hand once, the black-box tool stops being a black box.

12 PID versus MPC: What Each Approach Buys

Table 3 summarizes the trade-off this document has been building toward. Neither approach is strictly better: a PID bank is trivial to implement, easy to reason about loop by loop, and entirely adequate for station-keeping or slow, well-separated motions; an MPC is worth its added complexity exactly when the coupling between DOFs, the actuator limits, or the need to anticipate a moving reference start to matter more than the cost of solving an optimization problem every control cycle.

Table 3: PID versus MPC on the BlueROV2 Heavy

	PID (Section 10)	MPC (Section 11)
Model used	none (error-driven)	full nonlinear model, Eq. (18)
DOF coupling	ignored	explicit, via $\mathbf{A}_k, \mathbf{B}_k$
Actuator limits	handled after the fact	anticipated, in the constraints
Computation per step	six scalar updates	one QP solve
Tuning effort	six gain triples	$\mathbf{Q}, \mathbf{R}, \mathbf{R}_\Delta, N$

13 Conclusion

The chain built here, rigid-body dynamics, hydrodynamic corrections, kinematics, thruster allocation, and the force-to-PWM mapping, is the same chain regardless of which controller sits at the top of it: a PID bank and an MPC both end up producing a desired wrench $\boldsymbol{\tau}_{des}$, and both hand it to the identical allocation and PWM pipeline in Table 1. What changes between them is only how that wrench is decided, reactively from six independent errors, or predictively from the full coupled model. Understanding the physical origin of every matrix along the way, added mass as displaced fluid, Coriolis terms as apparent forces from a rotating frame, restoring forces as a righting keel, is what turns this from a page of equations into a design tool: it tells you, when the real vehicle does not behave the way the model predicted, which term in the pipeline to go back and question first.

References

[1] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*, Wiley, 2011.

[2] Blue Robotics, "T200 Thruster," performance and PWM calibration data (thrust, current, and power vs. PWM), `https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/`.

[3] ArduSub Documentation, ArduPilot Project, `https://www.ardusub.com/`.