Control Derivation and Python Implementation of Controller for Underwater Vehicle

Randal W. Beard
Brigham Young University

Updated: December 22, 2016

Abstract

This is the abstract.

1 Introduction

These notes are concerned with the control design for a simplified model of the forward motion of an underwater vehicle. Let the forward position of the vehicle be denoted by $p(t)$ and the forward velocity denoted by $v(t)$. The equations of motion are given by

$$\begin{aligned}\dot{p} &= v + v_c \\ m\dot{v} &= -\mu_1 v\,|v| - \mu_2 v\,|v|^3 + \alpha T,\end{aligned}$$

where m is the mass of the vehicle, μ_1 and μ_2 are drag coefficients, α is the throttle effectiveness coefficient, T is the trust, and v_c is the velocity of the current.

For the purposes of simulation, assume that the controller believes that the parameters of the system are $m = 10\ kg$, $\mu_1 = 25\ kg/m$, $\mu_2 = 15\ kgs^2/m^3$, and $\alpha = 0.8$. The actual parameters used in each simulation will be instantiated to be $\pm 20\%$ of the assumed values. The maximum thrust is $T_{\max} = 50\ N$. For all problems assume that $\dot{p}_r(t) = \ddot{p}_r(t) = 0$. In simulation, the reference trajectory for the position will be a step function.

These notes will derive the following controllers for this system:

- PID control,
- Backstepping control,
- Feedback linearizing control,
- Sliding mode control,

- Linear quadratic regulator,
- Observer based model reference adaptive control.

For each method we define the state space equation by defining the states as

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} e \\ \dot{e} \end{pmatrix},$$

where $e = p - p_r$. Since $\dot{e} = \dot{p} - \dot{p}_r = \dot{p} = v + v_c$, and $\ddot{e} = \dot{v} + \dot{v}_c = \dot{v} = -\mu_1/mv\,|v| - \mu_2/mv\,|v|^3 + \alpha/mT$, we get the state space equations

$$\dot{x} = f(x,u) = \begin{pmatrix} v + v_c \\ -\frac{\mu_1}{m} v\,|v| - \frac{\mu_2}{m} v\,|v|^3 + \frac{\alpha}{m} T \end{pmatrix}.$$

2 PID Control Design

To derive the PID conroller, define the Lyapunov function

$$V = \frac{1}{2}\gamma(p - p_r)^2 + \frac{1}{2}mv^2.$$

Differentiation gives

$$\begin{aligned} \dot{V} &= \gamma(p - p_r)\dot{p} + mv\dot{v} \\ &= \gamma(p - p_r)(v + v_c) + v(-\mu_1 v\,|v| - \mu_2 v\,|v|^3 + \alpha T). \end{aligned}$$

Assuming that $v_c = 0$ and using the PD control law

$$T = -k_p(p - p_r) - k_d v, \tag{1}$$

gives

$$\begin{aligned} \dot{V} &= \gamma(p - p_r)v - \mu_1 v^2\,|v| = \mu_2 v^2\,|v|^3 - \alpha k_p(p - p_r)v - \alpha k_d v^2 \\ &= (\gamma - \alpha k_p)(p - p_r)v - \mu_1 v^2\,|v| - \mu_2 v^2\,|v|^3 - \alpha k_d v^2. \end{aligned}$$

Selecting $\gamma = \alpha k_p$ gives

$$\dot{V} = -\mu_1 v^2\,|v| - \mu_2 v^2\,|v|^3 - \alpha k_d v^2 \leq 0.$$

Let $\Omega_c = \{x \in \mathbb{R}^2 : V(x) = c\}$ and not that for every c, Ω_c is compact and positively invariance, and that c can be selected to guarantee that any initial condition is in Ω_c. Let $E = \{x \in \Omega_c : \dot{V} = 0\} = \{x \in \Omega_c : x_2 = v = 0\}$, and let $M \subset E$ be the largest invariant set in E. Then

$$\begin{aligned} x \in M &\implies x_2 = v \equiv 0 \\ &\implies \dot{x}_2 \equiv 0 \\ &\implies -\frac{\mu_1}{m} v\,|v| - \frac{\mu_2}{m} v\,|v|^3 + \alpha T = \alpha T \equiv 0 \\ &\implies -\alpha k_p(p - p_r) - \alpha k_d v = -\alpha k_p(p - p_r) \equiv 0 \\ &\implies p - p_r \equiv 0 \\ &\implies x_1 \equiv 0. \end{aligned}$$

Therefore $M = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}$. Since V is radially unbounded and c can be selected so that $x_0 \in \Omega_c$ for any x_0, we have that the system is globally asymptotically stable.

If $v_c \neq 0$, the integral control is needed to guarantee asymptotic stability. In that case, the PID controller is given by

$$T_{PID} = -k_p(p - p_r) - k_d v - k_i \int (p - p_r). \tag{2}$$

A Python function implementing the PID controller is shown below.

```
class controller:
    def __init__(self, integrator0=0.0):
        #-- memory variables for integrator (PID and LQR) --
        self.integrator = integrator0  # initial condition for integrator
        self.error_d1 = 0.0  # delay of input to integrator

        #-- memory variables for dirty derivative --
        self.y_d1 = 0.0  # delay of y for differentiator
        self.ydot = 0.0  # output of differentiator
        # gains for dirty derivative
        self.diff_gain1 = (2.0*P.sigma-P.Ts)/(2.0*P.sigma+P.Ts)
        self.diff_gain2 = 2.0/(2.0*P.sigma+P.Ts)

        #-- states for adaptive controller --
        self.xref = np.matrix(np.zeros((P.Gam.shape[0],1)))
        self.thetahat = np.matrix(np.zeros((P.Gam.shape[0],1)))

    def PID(self, x, r):
        # read in states
        y = x[0]  # position
        v = x[1]  # velocity
        # compute the error
        error = y-r
        # integrate the error
        if np.abs(v) < 0.5:  # crude anti-windup
            self.integrator = self.integrator \
                              + (P.Ts/2.0)*(error+self.error_d1)
        self.error_d1 = error
        # PID controller before saturation
        u_unsat = - P.pid_kp*error \
                  - P.pid_ki*self.integrator \
                  - P.pid_kd*v
        u = self.sat(u_unsat, P.Tmax)
        if P.pid_ki != 0.0:
            self.integrator = self.integrator + (P.Ts/P.pid_ki)*(u-u_unsat)
```

```
        return u
```

3 Backstepping Control Design

The backstepping control is derived by defining the pre-Lyapunov function

$$V_1 = \frac{1}{2}e^2.$$

Differentiation gives

$$\begin{aligned}\dot{V}_1 &= e\dot{e} \\ &= e(v + v_c + \zeta - \zeta) \\ &= e(\zeta + v_c) + e(v - \zeta).\end{aligned}$$

Choosing $\zeta = -v_c - k_1 e$ gives

$$\dot{V}_1 = -k_1 e^2 + e(v - \zeta).$$

Now define the Lyapunov function

$$V_2 = V_1 + \frac{1}{2}(v - \zeta)^2,$$

and differentiate to get

$$\begin{aligned}\dot{V}_2 &= \dot{V}_1 + (v - \zeta)(\dot{v} - \dot{\zeta}) \\ &= -k_1 e^2 + (v - \zeta)(e - \frac{\mu_1}{m} v\,|v| - \frac{\mu_2}{m} v\,|v|^3 + \frac{\alpha}{m} T - \dot{\zeta}).\end{aligned}$$

Selecting the thrust as

$$T = \frac{m}{\alpha}(-e + \frac{\mu_1}{m} v\,|v| + \frac{\mu_2}{m} v\,|v|^3 + \dot{\zeta} - k_2(v - \zeta)),$$

gives

$$\dot{V}_2 = -k_1 e^2 - k_2(v - \zeta)^2.$$

Therefore, both $e \to 0$ and $v \to \zeta$. But since $\zeta = -v_c - k_1 e$ we also have that $\zeta \to -v_c$ which implies that $v + v_c \to 0$. In other words, the velocity of the vehicle will exactly counter the velocity of the current. Therefore the system is globally asymptotically stable.

The backstepping controller is given by

$$\begin{aligned}\zeta &= -v_c - k_1 e \\ \dot{\xi} &= -k_1(v + v_c) \\ T_{bs} &= \frac{m}{\alpha}(-e + \frac{\mu_1}{m} v\,|v| + \frac{\mu_2}{m} v\,|v|^3 + \dot{\zeta} - k_2(v - \zeta)).\end{aligned}$$

A Python function implementing the backstepping controller is shown below.

```
    def backstepping(self, x, r):
        # read in full states
        y = x[0]  # position
        v = x[1]  # velocity
        # compute the error
        error = y-r
        # backstepping controller
        zeta = -P.vc - P.backstepping_k1*error
        zetadot = -P.backstepping_k1*(v+P.vc)
        T = P.m/P.alpha*( -error + P.mu1/P.m*v*np.abs(v) \
                          + P.mu2/P.m*v*np.abs(v)**3 \
                          + zetadot \
                          - P.backstepping_k2*(v-zeta) \
                          )
        return self.sat(T, P.Tmax)
```

4 Feedback Linearizing Control Design

The equations of motion can be written as

$$\begin{aligned}\dot{x}_1 &= x_2\\ \dot{x}_2 &= -\frac{\mu_1}{m} v\left|v\right| - \frac{\mu_2}{m} v\left|v\right|^3 + \frac{\alpha}{m} T.\end{aligned}$$

The system is therefore feedback linearized by selecting

$$T = \frac{m}{\alpha}\left(\frac{\mu_1}{m} v\left|v\right| + \frac{\mu_2}{m} v\left|v\right|^3 + \nu\right)$$

to get

$$\begin{aligned}\dot{x}_1 &= x_2\\ \dot{x}_2 &= \nu.\end{aligned}$$

Selecting $\nu = -k_1 x_1 - k_2 x_2$ gives

$$\ddot{x}_1 + k_2 \dot{x}_1 + k_1 x_1 = 0.$$

Therefore, selecting the desired natural frequency of the closed loop system as ω_n and the desired damping ratio as ζ, and selecting $k_1 = \omega_n^2$ and $k_2 = 2\zeta\omega_n$ gives the behavior of the closed loop system as

$$\ddot{x}_1 + 2\zeta\omega_n \dot{x}_1 + \omega_n^2 x_1 = 0.$$

The feedback linearizing controller is therefore given by

$$\begin{aligned}\nu &= -k_1 x_1 - k_2 x_2\\ T_{fl} &= \frac{m}{\alpha}\left(\frac{\mu_1}{m} v\left|v\right| + \frac{\mu_2}{m} v\left|v\right|^3 + \nu\right),\end{aligned}$$

where $x_1 = p - p_r$ and $x_2 = v + v_c$.

A Python function implementing the feedback linearizing controller is shown below.

```
    def feedback_linearization(self, x, r):
        # read in full states
        y = x[0]  # position
        v = x[1]  # velocity
        # feedback linearizing control
        x1 = y-r
        x2 = v + P.vc
        nu = -P.flc_k1*x1 - P.flc_k2*x2
        T = P.m/P.alpha*( P.mu1/P.m*v*np.abs(v) \
                        + P.mu2/P.m*v*np.abs(v)**3 \
                        + nu \
                        )
        return self.sat(T, P.Tmax)
```

5 Sliding Mode Control Design

Define the sliding surface as

$$s \triangleq \dot{e} + k_s e = v + v_c + k_s e.$$

On the sliding surface $s = 0$ we have that $\dot{e} = -k_s e$ which implies that $e \to 0$. The task is therefore to stabilize the sliding surface. Toward that end, let

$$V = \frac{1}{2} s^2.$$

Differentiating gives

$$\begin{aligned}
\dot{V} &= s\dot{s} \\
&= s(\ddot{e} + k_s \dot{e}) \\
&= s\left(-\frac{\mu_1}{m} v\,|v| - \frac{\mu_2}{m} v\,|v|^3 + \frac{\alpha}{m} + k_s(v + v_c)\right) \\
&= \frac{\alpha}{m}\left[sT + \frac{1}{\alpha} s\left(-\mu_1 v\,|v| - \mu_2 v\,|v|^3 + k_s m(v + v_c)\right)\right] \\
&\leq \frac{\alpha}{m}\left[sT + \frac{1}{\alpha_{\min}}\,|s|\left(\mu_1 v^2 + \mu_2 v^4 + k_s m_{\max}(|v| + v_{c,\max})\right)\right].
\end{aligned}$$

Defining

$$\rho(x) \triangleq \frac{1}{\alpha_{\min}}\left(\mu_1 v^2 + \mu_2 v^4 + k_s m_{\max}(|v| + v_{c,\max})\right)$$

gives

$$\dot{V} \leq \frac{\alpha}{m}\left[sT + \rho(x)\,|s|\right].$$

Selecting the torque as

$$T = -(\rho(x) + \beta_0)\text{sign}(s)$$

gives

$$\dot{V} \leq -\frac{\alpha}{m}\beta_0 |s| .$$

Since the sign$(\cdot)$ function results in chattering around the sliding mode, in practice we implement

$$T = -(\rho(x) + \beta_0)\text{sat}(s/\epsilon).$$

Let $W = \sqrt{2V} = |s|$. For $|s| \geq \epsilon$ we have that

$$\dot{W} = \dot{V}/W = \frac{-\frac{\alpha}{m}\beta_0 |s|}{|s|} = -\frac{\alpha}{m}\beta_0$$

which implies that

$$W(t) \leq W(t_0) - \frac{\alpha}{m}\beta_0(t - t_0),$$

or in other words

$$|s(t)| \leq |s(t_0)| - \frac{\alpha}{m}\beta_0(t - t_0).$$

Therefore the system enters an ϵ boundary around the sliding surface in finite time.

After $|s(t)| < \epsilon$, letting $V = \frac{1}{2}e^2$ gives

$$\begin{aligned}
\dot{V} &= e\dot{e} \\
&= e(s - k_s e) \\
&= -k_s e^2 + es \\
&\leq -k_s e^2 + |e|\,|s| \\
&\leq -k_s e^2 + \epsilon\,|e| \\
&= -k_s e^2 + \epsilon\,|e| + k_s\theta e^2 - k_s\theta e^2 \\
&= -(1 - \theta)k_s e^2 - k_s\theta e^2 + \epsilon\,|e| ,
\end{aligned}$$

for $0 < \theta < 1$. Therefore $\dot{V} < 0$ when

$$\begin{aligned}
&- k_s\theta e^2 + \epsilon\,|e| < 0 \\
\implies &|e| > \frac{\epsilon}{k_s\theta}.
\end{aligned}$$

Wherefore when $|e| > \frac{\epsilon}{k_s\theta}$, $\dot{V} < 0$. By the ultimate boundedness lemma, $|e|$ will eventually become less than $\frac{\epsilon}{k_s\theta}$ in finite time. Letting $\theta \to 1$ implies that the ultimate bound for $|e|$ is ϵ/k_s.

The sliding mode controller is therefore given by

$$\begin{aligned}
s &= v + v_c + k_s e \\
\rho(x) &= \frac{1}{\alpha_{\min}}\left(\mu_1 v^2 + \mu_2 v^4 + k_s m_{\max}(|v| + v_{c,\max})\right) \\
T_{sm} &= -(\rho(x) + \beta_0)\text{sat}(s/\epsilon).
\end{aligned}$$

A Python function implementing the feedback linearizing controller is shown below.

```
    def sliding_mode(self, x, r):
        # read in full states
        y = x[0]  # position
        v = x[1]  # velocity
        # compute the error
        error = y-r
        # sliding mode controller
        s = v + P.vc + P.sm_ks*error;
        rho = (1/P.sm_alpha_min)*(P.sm_mu1_max*v**2 \
              + P.sm_mu2_max*v**4 \
              + P.sm_ks*P.sm_m_max*(P.sm_vc_max + abs(v)))
        T = -(rho+P.sm_beta0)*self.sat(s/P.sm_epsilon, 1.0)
        return T
```

6 Linear Quadratic Regulator Control Design

A Python function implementing the feedback linearizing controller is shown below.

```
    def lqr(self, x, r):
        # read in full states
        y = x[0]  # position
        v = x[1]  # velocity
        # compute the error
        error = y-r
        # integrate the error
        self.integrator = self.integrator + (P.Ts/2.0)*(error+self.error_d1)
        self.error_d1 = error
        # lqr control
        augmented_state = np.matrix([
                                    [self.integrator],
                                    [y],
                                    [v]
                                    ])
        T_unsat = - float(P.K_lqr*augmented_state)
        T = self.sat(T_unsat, P.Tmax)
        if P.K_lqr[0][0] != 0.0:
            self.integrator = self.integrator + (P.Ts/P.K_lqr[0][0])*(T-T_unsat)
        return T, augmented_state
```

7 Observer Based Model Reference Adaptive Control Design

8 Design Methodology

8.1 Step 1. Develop Augmented Model of the System.

Given the nonlinear system

$$\begin{aligned}\dot{x}_p &= f(x_p, u_p)\\ y_p &= h(x_p),\end{aligned}$$

where x_p is the state of the plant, and where u_p is the input to the plant.

The first step do a change of variables and linearize the system so that it takes the form

$$\dot{\tilde{x}}_p = A_p\tilde{x}_p + B_p\Lambda\left(\tilde{u}_p + \Theta_d^\top \Phi_d(\tilde{x}_p)\right), \tag{3}$$

$$\tilde{y}_p = C_p\tilde{x}_p, \tag{4}$$

where Λ is diagonal with positive elements, but is otherwise unknown, and where $\Phi_d(x_p)$ is a vector of known functions, and where Θ_d are unknown constants.

Perhaps the most straight forward way to do this is to linearize the system about the equilibrium point (x_e, u_e), where $f(x_e, u_e) = 0$. Letting $\tilde{x}_p = x_p - x_e$, $\tilde{u}_p = u_p - u_e$ results in the linearized equation

$$\begin{aligned}\dot{\tilde{x}}_p &= A_p\tilde{x}_p + B_p\tilde{u}_p,\\ \tilde{y}_p &= C_p\tilde{x}_p,\end{aligned}$$

where

$$\begin{aligned}A_p &= \frac{\partial f}{\partial x}(x_e)\\ B_p &= \frac{\partial f}{\partial u}(x_e)\\ C_p &= \frac{\partial h}{\partial x}(x_e).\end{aligned}$$

The methodology assumes that the errors in the linearization process can be adequately represented by equations (3). If possible, a change of variables should be applied so that the nonlinearities appear in the input channel as in Equation (3).

The control objective is for the linearized output $\tilde{y}_p$ to follow the commanded output y_c. Accordingly we add the integrator

$$x_I = \int_{-\infty}^{t} \left(\tilde{y}_p(\tau) - y_{\text{cmd}}(\tau)\right) d\tau = \int_{-\infty}^{t} \left(C_p\tilde{x}_p(\tau) - y_c(\tau)\right) d\tau,$$

and augment the state space equations as

$$\begin{pmatrix} \dot{x}_I \\ \dot{\tilde{x}}_p \end{pmatrix} = \begin{pmatrix} 0 & C_p \\ 0 & A_p \end{pmatrix} \begin{pmatrix} x_I \\ \tilde{x}_p \end{pmatrix} + \begin{pmatrix} 0 \\ B_p \end{pmatrix} \Lambda \left(\tilde{u}_p + \Theta_d^\top \Phi_d(\tilde{x}_p) \right) + \begin{pmatrix} -I \\ 0 \end{pmatrix} y_c. \tag{5}$$

Defining

$$\begin{aligned} x &= \begin{pmatrix} x_I \\ \tilde{x}_p \end{pmatrix} \\ A &= \begin{pmatrix} 0 & C_p \\ 0 & A_p \end{pmatrix} \\ B &= \begin{pmatrix} 0 \\ B_p \end{pmatrix} \\ B_{\text{ref}} &= \begin{pmatrix} -I \\ 0 \end{pmatrix}, \end{aligned}$$

gives

$$\dot{x} = Ax + B\Lambda \left(\tilde{u}_p + \Theta_d^\top \Phi_d(\tilde{x}_p) \right) + B_{\text{ref}} y_c, \tag{6}$$

where A, B, and B_c are known matrices.

8.2 Step 2. Design LQR Controller for Linear Augmented System

The second step in the design methodology outlined in [**?**] is to design an LQR controller for the linear augmented system derived from Equation (6) by removing the commanded input and the unknown quantities, i.e.,

$$\dot{x} = Ax + Bu.$$

The LQR controller is selected to minimize the cost functional

$$J = \int_0^\infty x^\top Q_{\text{ref}} x + u^\top R_{\text{ref}} u \, dt,$$

where Q_{ref} and R_{ref} are design parameters. Denote the resulting controller as

$$u_{lqr} = -K_{lqr} x.$$

8.3 Step 3. Design the Observer-Like Reference Model

The third step in the design methodology outlined in [**?**] is to construct the "observer-like" reference model

$$\dot{x}_{\text{ref}} = A_{\text{ref}} x_{\text{ref}} + B_{\text{ref}} y_c + L_\nu (x - x_{\text{ref}}), \tag{7}$$

where

$$A_{\text{ref}} = A - BK_{lqr},$$

and where L_ν is an observer gain. The selection of the observer gain follows design principles that are similar to loop transfer recovery for LQR. The observer gain L_ν is given by

$$L_\nu = P_\nu R_\nu^{-1}, \tag{8}$$

and P_ν is the unique positive definite solution of the Riccati equation

$$P_\nu A_{\text{ref}}^\top + A_{\text{ref}} P_\nu - P_\nu R_\nu^{-1} P_\nu + Q_\nu = 0, \tag{9}$$

where Q_ν and R_ν are selected as

$$Q_\nu = Q_0 + \left(\frac{\nu+1}{\nu}\right) I$$
$$R_\nu = \left(\frac{\nu}{\nu+1}\right) I.$$

The matrix Q_0 is a design parameter, and ν is the LTR-like tuning parameter. As $\nu \to 0$, the oscillations in the transients of the controller will be eliminated.

8.4 Step 4. Design the Adaptive Controller

The control input to the system, as given by the design methodology outlined in [**?**], is

$$\tilde{u}_p = u_{lqr} + u_{ad},$$

where u_{ad} is a model reference adaptive controller given by

$$u_{ad} = -\hat{\Theta}^\top \Phi(x),$$

where

$$\Phi(x) = \begin{pmatrix} u_{lqr} \\ \Phi_d(\tilde{x}_p) \end{pmatrix},$$

and $\hat{\Theta}$ evolves according to

$$\dot{\hat{\Theta}} = \text{Proj}(\hat{\Theta}, \Gamma \Phi(x) \mu(\|e\|) e^\top P_\nu^{-1} B), \tag{10}$$

where Γ are the adaptive gains,

$$\mu(\|e\|) = \max\left(0, \min\left(1, \frac{\|e\| - \delta e_0}{(1-\delta)e_0}\right)\right)$$

is the deadzone operator, and

$$\text{Proj}(\Theta, y) = \begin{cases} y - \frac{\Gamma \nabla f(\Theta) \nabla f(\Theta)^\top}{\|\nabla f(\Theta)\|_\Gamma^2} y f(\Theta), & \text{if } f(\Theta) > 0 \text{ and } y^\top \nabla f(\Theta) > 0 \\ y, & \text{otherwise} \end{cases}$$

is the projection operator [**?**], where the parameter vector belongs to the set

$$\Omega = \{\Theta \in \mathbb{R}^n : f(\Theta) \leq 1\}.$$

For the most common case where each element in the parameter vector satisfies

$$|\Theta_i| \leq \bar{\Theta}_i,$$

the projection operator becomes

$$\text{Proj}(\Theta, y) = \begin{pmatrix} \text{proj}(\Theta_1, y_1, \bar{\Theta}_1) \\ \vdots \\ \text{proj}(\Theta_N, y_N, \bar{\Theta}_N) \end{pmatrix}, \tag{11}$$

where

$$\text{proj}(\Theta_i, y_i, \bar{\Theta}_i) = \begin{cases} \left(1 - \frac{(1+\epsilon)\Theta_i^2 - \bar{\Theta}_i^2}{\epsilon \bar{\Theta}_i^2}\right) y, & \text{if } |\Theta_i| \geq \frac{\bar{\Theta}_i}{\sqrt{1+\epsilon^2}} \text{ and } y_i \Theta_i > 0 \\ y_i, & \text{otherwise} \end{cases}. \tag{12}$$

9 Design Examples

9.1 Underwater Vehicle

In this section we will consider a simplified model of the forward motion of an underwater vehicle. Let the forward position of the vehicle be denoted by $p(t)$ and the forward velocity denoted by $v(t)$. The equations of motion are given by

$$\begin{aligned} \dot{p} &= v + v_c \\ m\dot{v} &= -\mu_1 v\,|v| - \mu_2 v\,|v|^3 + \alpha T, \end{aligned} \tag{13}$$

where m is the mass of the vehicle, μ_1 and μ_2 are drag coefficients, α is the trottle effectiveness coefficient, T is the trust, and v_c is the velocity of the current.

We will assume that the controller does not know the true values of m, μ_1, μ_2, α, and v_c.

For the purposes of simulation, the true parameters of the system are assumed to be $m = 10\ kg$, $\mu_1 = 25\ kg/m$, $\mu_2 = 15\ kgs^2/m^3$, $\alpha = 0.8$, and $v_c = 0.5\ m/s$. The maximum thrust is $T_{\max} = 50\ N$.

9.1.1 Step 1. Develop Augmented Model of the System.

Equation (13) can be written as

$$\begin{aligned} \begin{pmatrix} \dot{p} \\ \dot{v} \end{pmatrix} &= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} p \\ v \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \left(\frac{\alpha}{m}\right) \left(T + \begin{pmatrix} -\frac{\mu_1}{\alpha} \\ -\frac{\mu_2}{\alpha} \end{pmatrix}^\top \begin{pmatrix} v\,|v| \\ v\,|v|^3 \end{pmatrix} \right) + \begin{pmatrix} v_c \\ 0 \end{pmatrix} \\ y &= \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} p \\ v \end{pmatrix}, \end{aligned} \tag{14}$$

which, with the exception of the last term involving the current speed, is in the form of Equation (3). We will ignore the current and rely on the integrator for

steady state tracking when $v_c \neq 0$. Accordingly, we augment the system with the integrator

$$x_I(t) = \int_{-\infty}^{t} (p(\tau) - y_c(\tau))\, d\tau,$$

to get

$$\underbrace{\begin{pmatrix} \dot{x}_I \\ \dot{p} \\ \dot{v} \end{pmatrix}}_{\dot{x}} = \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_I \\ p \\ v \end{pmatrix}}_{x} + \underbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}}_{B} \underbrace{\left(\frac{\alpha}{m}\right)}_{\Lambda} \left(\underbrace{T}_{u} + \underbrace{\begin{pmatrix} -\frac{\mu_1}{\alpha} \\ -\frac{\mu_2}{\alpha} \end{pmatrix}^{\top}}_{\Theta_d^{\top}} \underbrace{\begin{pmatrix} v\,|v| \\ v\,|v|^3 \end{pmatrix}}_{\Phi_d(x)} \right) + \underbrace{\begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}}_{B_{\text{ref}}} y_c,$$

which is equivalent to Equation (6).

9.1.2 Step 2. Design LQR Controller for Linear Augmented System

The second step is to design an LQR controller for the system

$$\dot{x} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} u.$$

Using Bryson's rule [**?**] where R is selected as $1/u_{\max}$ we get

$$R = \frac{1}{T_{\max}} = \frac{1}{50} = 0.02.$$

The state weighting matrix is selected to penalize v and x_I which we hope to drive to zero. Since the steady state position p will not be zero, we set the corresponding weight to zero. After tuning we use

$$Q = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 200 \end{pmatrix}.$$

The resulting LQR gains are

$$K_{lqr} = \begin{pmatrix} 10.0000 & 45.9539 & 105.5881 \end{pmatrix}.$$

9.1.3 Step 3. Design the Observer-Like Reference Model

Given the LQR gains found in the previous section, the state coupling matrix for the reference model is given by

$$\begin{aligned} A_{\text{ref}} &= A - BK_{lqr} \\ &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 10.0000 & 45.9539 & 105.5881 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -10 & -44.82 & -100.45 \end{pmatrix}. \end{aligned}$$

Letting $\nu = 0.001$ and solving for P_ν in Equation (9) and L_ν in Equation (8) gives

$$P_\nu = \begin{pmatrix} 1 & 0.0004 & -0.005 \\ 0.0004 & 0.9997 & -0.0219 \\ -0.005 & -0.0219 & 0.9054 \end{pmatrix}$$
$$L_\nu = \begin{pmatrix} 1001 & 0.4 & -5 \\ 0.4 & 1001 & -21.9 \\ -5 & -21.9 & 906.4 \end{pmatrix}.$$

The reference model is given by Equation (7).

9.1.4 Step 4. Design the Adaptive Controller

For this problem we will use the regressor vector

$$\Phi(x) = \begin{pmatrix} u_{lqr} \\ v\,|v| \\ v\,|v|^3 \end{pmatrix}$$

that matches the dynamics given in Equation (14). The initial parameter vector is set to $\hat{\Theta}(0) = 0$. The deadzone operator will not be implemented, implying that $\mu(\|e\|) = 1$ for all e.

We will assume that the parameters are limited by $|\Theta_i| \leq 2$, which implies that the projection operator is given by Equations (11) and (12). The adaptive gains in Equation (10) are $\Gamma = 10 I_3$.

10 Conclusions

References