%% AUV Obstacle Stop Control Using Discrete PID in Simulink

This Live Script documents a simple Simulink model that uses a discrete PID controller to stop an autonomous underwater vehicle (AUV) at a desired distance from an obstacle.

The model is designed as a learning example: it shows how to build a PID from basic blocks, how to model simple distance dynamics, and how to tune the gains.

%% Model Overview

The system controls the distance between the AUV and a fixed obstacle.

Setpoint (set_dist): desired distance from the obstacle in meters.

Measured distance (meas_dist): current distance between AUV and obstacle.

Error: difference between measured distance and setpoint.

PID controller: outputs a forward velocity command.

Auv Plant: updates the distance based on commanded velocity.

Saturation: limits velocity to a realistic range (no reverse, max speed).

The overall loop is:

Compute error:
e[k] = meas_dist[k] - set_dist

PID controller computes velocity command v[k] from e[k].

Auv Plant updates distance:
d[k] = d[k-1] - v[k]*Ts

New distance d[k] is fed back as meas_dist[k] for the next step.

%% Units

All signals and parameters use these units:

Distance: meters (m)

Time: seconds (s)

Velocity: meters per second (m/s)

Sample time Ts: seconds

```
Make sure these units are consistent when setting constants, gains, and
initial conditions.

%% Top-Level Model Structure

The main Simulink model (for example: SimulinkPID_StopBot.slx) contains:

Set distance (Constant block)

Value: desired stop distance from obstacle, e.g. set_dist = 1 (meter).

Measured distance (signal from Auv Plant)

This is the feedback signal meas_dist.

Error (Sum block)

Computes:
e[k] = meas_dist[k] - set_dist

Positive error means the AUV is too far from the obstacle.

PID controller

Implemented using three parallel paths:

Proportional:
yP[k] = Kp * e[k]

Integral:
yI[k] = yI[k-1] + Ki * Ts * e[k]
implemented with a Gain and a Unit Delay.

Derivative:
yD[k] = Kd * (e[k] - e[k-1]) / Ts
implemented with a discrete derivative structure.

The three terms are summed:
v[k] = yP[k] + yI[k] + yD[k]

Saturation (velocity limiter)

Limits v[k] to:

Lower limit = 0 m/s (no reverse).

Upper limit = Vmax (max forward speed of the AUV).

The saturated velocity is sent to the Auv Plant.
```

```
Auv Plant (subsystem)

Models how distance changes with commanded velocity (see next section).

Logging / visualization

A Scope and/or To Workspace block connected to meas_dist
so distance vs time can be plotted in MATLAB.

%% Auv Plant Subsystem

The Auv Plant subsystem implements a simple 1D kinematic model of the AUV
approaching an obstacle.

Signals:

Input: commanded forward velocity v[k] (m/s).

Output: distance d[k] = meas_dist[k] (m).

Blocks:

Gain "Ts"

Gain value = sample time Ts (seconds), e.g. Ts = 0.1.

Output: v[k] * Ts, the distance traveled during one time step.

Unit Delay

Holds previous distance d[k-1].

Initial condition = starting distance from obstacle, for example 10
(meters).

Sample time can be left as -1 to inherit from the model.

Sum block

Implements the distance update:
d[k] = d[k-1] - v[k]*Ts

Inputs:

input: Unit Delay output (d[k-1])

input: v[k]*Ts

Output port
```

```
Outputs d[k] as meas_dist.

This simple plant assumes the AUV moves straight toward the obstacle and
that positive velocity reduces the distance.

%% Typical Parameter Settings

Below is a typical set of starting values that work for many tests.

In MATLAB, you can define:

matlab
set_dist = 1;      % desired distance from obstacle [m]
Ts       = 0.1;    % sample time [s]

Kp = 0.7;          % proportional gain
Ki = 0.02;         % integral gain
Kd = 0.01;         % derivative gain

Vmax = 2;          % max forward speed [m/s]
d0   = 10;         % initial distance [m]
In the model:

Set distance Constant = set_dist.

Gain Ts inside Auv Plant = Ts.

Unit Delay initial condition inside Auv Plant = d0.

P, I, D gains set to Kp, Ki, Kd.

Velocity Saturation: Lower = 0, Upper = Vmax.

Make sure the solver uses a fixed-step size consistent with Ts if you run a
purely discrete-time model.

%% How to Run the Simulation

Open the Simulink model:

matlab
open_system('SimulinkPID_StopBot.slx');  % adjust name if needed
Confirm or set parameters as described in the previous section.

Set the simulation stop time to a value like 10 or 20 seconds.

Click Run in Simulink.

If you use a To Workspace block with variable name simout and Timeseries
format, you can plot the distance in MATLAB:
```

```
matlab
plot(simout.Time, simout.Data);
xlabel('Time [s]');
ylabel('Distance to obstacle [m]');
title('AUV Distance vs Time');
grid on;
You should see the distance starting at d0 (e.g., 10 m) and decreasing
toward set_dist (e.g., 1 m), then settling.

%% Quick Plant Test Without PID

To verify that the Auv Plant subsystem is working correctly, you can bypass
the PID and use a constant velocity:

Disconnect the PID output from the Saturation.

Connect a Constant block with value 1 (1 m/s) to the Saturation input, and
then to Auv Plant.

Set:

Ts = 0.1 s

Unit Delay initial condition = 10 m

Run the simulation and plot distance.

With constant v = 1 m/s, the distance should decrease approximately
linearly from 10 m to 0 m in about 10 seconds, with slope −1 m/s.

This confirms that the plant equation d[k] = d[k−1] − v[k]∗Ts is
implemented correctly.

%% PID Tuning Guidelines

The PID gains Kp, Ki, and Kd control how the system behaves:

Proportional gain Kp

Increases speed of response.

Too low: AUV moves slowly and may stop far from the set distance.

Too high: can cause overshoot and oscillations.

Integral gain Ki

Reduces steady-state error; helps ensure that the final distance is close
to set_dist.
```

```
Too low: steady-state error disappears very slowly.

Too high: can cause integral wind-up, large overshoot, and slow
oscillations.

Derivative gain Kd

Reacts to the rate of change of error; adds damping.

Helps reduce overshoot and sharp oscillations.

Too high: makes the control noisy or sluggish.

Practical tuning procedure:

Set Ki = 0 and Kd = 0; tune Kp first.

Increase Kp until the distance response is reasonably fast but not unstable.

Add a small Ki to remove steady-state error.

Start with a value much smaller than Kp, e.g., Ki ≈ 0.01–0.05.

Add a small Kd if needed to reduce overshoot.

Start with Kd ≈ 0.01–0.05 relative to Kp.

Always change one gain at a time and re-run the simulation.

%% Interpreting the Response

Key behaviors to look for:

Rise time: how quickly distance comes down from the initial value toward
set_dist.

Overshoot: how far the distance goes past the set_dist (too close to the
obstacle) before coming back.

Steady-state error: difference between final distance and set_dist after
the system has settled.

Settling time: how long it takes until the distance stays within a small
band around set_dist.

For a good controller:

Distance decreases smoothly from d0 to near set_dist.
```

```
Overshoot is small or zero (distance does not go much below set_dist).

Steady-state error is near zero (final distance ≈ set_dist).

Settling time is acceptable for your AUV scenario.

%% Educational Extensions

This simple example can be extended in several ways:

Replace the 1D Auv Plant with a more detailed AUV dynamics model (including
mass, drag, and actuator dynamics).

Add sensor noise to the measured distance signal.

Add separate PID loops for depth or heading and test multi-variable control.

Add logic to handle multiple obstacles or different target distances over
time.
```