PID Control

\[ u \xrightarrow{+} e \xrightarrow{1/s} Ki \xrightarrow{Kp} \Sigma \xrightarrow{s} F(s) \xrightarrow{\frac{1}{ms^2 + bs + k}} X(s) \]
A controller is a mechanism that controls the output of a system by adjusting its control input. A feedback controller continuously measures the output, compares it to the desired output (the setpoint), and adjusts the input depending on the calculated error. A PID controller is the most common type of feedback controller. It adjusts the control variable depending on the present error (P - proportional control), the accumulated error in the past (I - integral control), and the predicted future error (D - derivative control).

A block diagram of a PID controller can be seen in the figure below.

The Plant/Process block represents the system to which we apply the input \( u \) and it returns the output \( y \). A car is a system to which we apply a steering wheel angle command as an input and it responds with a change of direction as an output. The characteristics of the car such as its weight, center of gravity position, suspension geometry, and tires will define the direction change for a given change in steering wheel angle.

If a controller is implemented, the input \( u \) is calculated as shown in the diagram by summing the proportional, integral, and derivative terms. The terms are obtained by multiplying the error, the integral of the error and the derivative of the error with P, I and D gains respectively.
What is the role of the Controller Gains (Kp, Kd, Ki)?
The order of the system

Before designing the PID controller it is useful to identify the order of the system. The system that was analyzed in the above example is a second-order system which is evident from its transfer function that has a form of \(1/(as^2 + bs + c)\). We can recognize a second-order system by the \(s^2\) in the denominator which is a consequence of a second derivative in the differential equation.

Another characteristic of the second-order system is its open-loop step response which can have oscillations about the set point. Because of the combination of the \(m, b,\) and \(k\) parameters in the example, the system was overdamped and therefore did not oscillate. However, if we change the damping parameter \(b\) to 2, the step response is underdamped:

![Open loop response, m = 2, b = 2, k = 10](image)

A second-order system equation can also be rewritten in the following form:

\[
\frac{X(s)}{u(s)} = \frac{K\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}
\]

The constant in the numerator does not change the shape of the response but acts purely as a scaling factor. The values in the denominator tell us much more about the system. \(\omega_n\) is the natural frequency of the system, which in our example is \(\sqrt{\frac{k}{m}}\). Parameter \(\xi\) is the damping ratio and is equal to \(\frac{b}{2m\omega_n}\) in our example. If we remove the damper from our system \((b = 0)\), the \(\xi\) is zero and there is no damping in the system. If \(\xi = 1\), the system is critically damped \(â€¢\) this gives the fastest response without oscillations. If \(\xi > 1\), the system is overdamped, and if \(\xi < 1\), the system is underdamped which gives an oscillatory response. Therefore the response of the moving mass in our example can be reasonably tuned by setting the spring and the damper to appropriate values, which is an example of open-loop control. In many cases the stiffness and damping in the system cannot be tuned directly and closed-loop control is needed.

**Second-order system examples**

A displacement response of a spring-mass-damper system to the force input is a typical second-order system \(â€¢\) the force on the mass is proportional to the acceleration of the mass, which is the second derivative of the displacement. Another example of a second-order system is a voltage response across the capacitor in an RLC circuit to the supply voltage input \(â€¢\) voltage across the inductor is proportional to the second derivative of the voltage across the capacitor.

A simpler system is a first-order system which has a transfer function of the following form:

\[
\frac{X(s)}{u(s)} = \frac{a}{s + a}
\]

There is only \(s^1\) in the denominator, indicating that the differential equation of a first-order system only involves a first derivative of the output. Step response of a 1st order system is shown in image 13.
Analysing the transfer function of a 1st order system, we can compute the step response (\( u(s) = \frac{1}{s} \) for a step input) in the frequency domain:

\[
X(s) = \frac{1}{s} \cdot \frac{a}{s + a} = \frac{1}{s} - \frac{1}{s + a}
\]

Taking the inverse Laplace transform of the response, the time domain response is:

\[
x(t) = 1 - e^{-at}
\]

This function is plotted in the figure above (step response of the 1st order system). In this equation, the parameter \( a \) tells us all about the system. When \( t=\frac{1}{a} \), the amplitude is \( 1 - e^{-1} \), which is approximately 63% of the steady-state amplitude and the \( t=\frac{1}{a} \) is actually called a time constant. Parameter \( a \) itself is also the derivative of the \( x(t) \) when \( t=0 \), thus it tells us the initial slope of the response.

Furthermore we can define the rise time as the time it takes the amplitude to rise from 10% to 90% of the steady-state value: \( Tr=\frac{2.2}{a} \). The settling time is defined as the time when the amplitude reaches 98% of the steady-state value: \( Ts=\frac{4}{a} \). We arrive at these numbers by inserting appropriate amplitude in the \( x(t) \) equation and solve for \( t \).

A more general 1st order system has the transfer function of the following form:

\[
\frac{X(s)}{u(s)} = \frac{k}{s + a}
\]

The only difference in the step response is that the amplitude is not 1 but rather \( \frac{k}{a} \), since the time domain response is:

\[
x(t) = \frac{k}{a} (1 - e^{-at})
\]

\( s+a \) term in the denominator is also called a lag term, because it introduces lag to the system (process variable lags behind the control input).

**First-order system examples**

An exemplary first-order system is a rotational velocity response of a DC motor to the step voltage input. We mentioned that an RLC circuit has a 2nd order system characteristics. On the other hand, the LR and RC circuits are 1st order systems because the supply voltage is related to the voltage or the derivative of the voltage across the respective elements in the circuit. Another example is the emptying of the water tank with a valve at the bottom. The pressure at the valve will decrease exponentially as the water level drops.
PID control in Dewesoft X

Dewesoft X can be used as a PID controller by making use of the PID Control function in the Math section.

**WARNING:** Dewesoft X software running on Windows is not a real-time system because Windows is not a real-time operating system. This means that the delay of the controller is not always the same and can be unpredictable. For example, Windows could give priority to another application instead of Dewesoft X, which can dramatically increase controller delay.

However, a lot of applications do not require real-time control. The controller response time depends on the computer specifications, but as we will see in the example, the response time can be very short.

To access PID control in Dewesoft X, go to Math -> Add Math tab and write PID in the search box.

User inputs accessible during measurement are the inputs for three gains (\(K_p\), \(K_i\), and \(K_d\)) and two buttons to stop and reset the controller (Stop PID, Reset PID).

Clicking on the Setup tab opens a new window with essential PID settings:
Input - from the list on the left-hand side of the setup window we choose the input channel that will be controlled.

Loop type - if an Open loop is chosen, there is no feedback to the controller. In measure mode, the user can set a value to the controlling channel (analog or digital output, for example) and measure the response without controller interference. This option is useful for obtaining the step response of the system. If the Closed loop is chosen, the feedback loop is turned on and additional settings appear.

Controller type - we can choose between a PID controller and a PI controller with anti-windup (anti-windup will be described later in this tutorial). If only the P controller is needed, the PID type can be chosen and zeros filled in for I and D gains.

Wanted value - the set point. In the figure above we created a user input channel (under Channel Setup, User Inputs) called ‘Set point freq’ in order to be able to change the set point during measurement.

Output channel - the controller output. Normally an analog or digital output channel would be chosen. In the figure above we named the analog output channel as “Control A0 voltage”.

PID loop output time - the period at which the controller will update the Output channel.

Averaging type - the method of averaging the error (error = wanted value - input). It can be used if there is a lot of noise in the process value.

Maximum output levels - the minimum and maximum levels of the Output channel can be set if desired.

Integrator windup limit - (visible when Controller type is set to PI with anti-windup) the limits of the Output channel in its units. When the Output channel is outside these limits, the error will not be integrated (anti-windup will be described later in this tutorial).
Which are the PID Tuning Methods?

*Dewesoft X*’s PID Control function provides the user with some basic PID tuning methods. To access the tuning methods click on Tuner... in the PID Control setup. The four available methods will be shortly described in this section. The meaning of the variables used in this section is best described in the following figures.

The process gain, $K$, is calculated as the change in process variable over the change in control variable:

$$ K = \frac{\Delta PV}{\Delta CO} $$

The controller gains calculated with the tuning methods will be based on the non-interactive (or ideal) definition of a PID controller in time domain:

$$ u(t) = K_e \left( e(t) + \frac{1}{T_i} \int_0^t e(t) dt + T_d \frac{de}{dt} \right) $$

Because *Dewesoft X* uses a parallel PID algorithm the following equations apply to convert from the gains $K_e$, $T_i$ and $T_d$ to $K_P$, $K_I$ and $K_D$:

$$ K_P = K_e $$

$$ K_I = \frac{K_e}{T_i} $$

$$ K_D = K_e \cdot T_d $$

The difference between the non-interactive and parallel PID algorithms is shown below.

Non-interactive PID algorithm:
Parallel PID algorithm:

<table>
<thead>
<tr>
<th>Method</th>
<th>Type of process</th>
<th>Identification</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ziegler-Nichols</td>
<td>Lag dominated</td>
<td>Closed loop</td>
<td>1/4 decay ratio</td>
</tr>
<tr>
<td>Cohen-Coon</td>
<td>Lag &amp; dead time</td>
<td>Open loop</td>
<td>1/4 decay ratio</td>
</tr>
<tr>
<td>Lambda</td>
<td>Lag dominated</td>
<td>Open loop</td>
<td>Adjustable time constant</td>
</tr>
<tr>
<td>Dead time</td>
<td>Dead time dominated</td>
<td>Open loop</td>
<td></td>
</tr>
</tbody>
</table>

Ziegler-Nichols method

This method is based on the frequency response of the system. The $K_p$, $K_d$, and $K_i$ are calculated from the $K_u$, the ultimate gain and $T_u$, the period of oscillation at $K_u$. The ultimate gain is the proportional gain of the controller (when integral and derivative gains are 0) at which the system becomes unstable. Thus we have to analyze the system by slowly increasing the proportional gain until it oscillates with a constant amplitude. When the $K_u$ is obtained, the controller gains are calculated according to the following table:

<table>
<thead>
<tr>
<th>Controller $K_u$</th>
<th>$T_u$</th>
<th>$T_i$</th>
<th>$T_d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PI</td>
<td>0.4</td>
<td>0.8</td>
<td>1.4</td>
</tr>
<tr>
<td>PID</td>
<td>0.6</td>
<td>0.5</td>
<td>0.125</td>
</tr>
</tbody>
</table>

PID controller designed by the Ziegler-Nichols method causes the system to overshoot the set point and oscillate at a decay rate of $A^{1/4}$ (amplitude is attenuated by a factor of 4 during one oscillation). If such oscillations are not appropriate for the controlled system (many processes even do not tolerate overshoot), further tuning is necessary. Another problem of the Ziegler-Nichols method is that we first have to find the ultimate gain, which can sometimes be impossible if the system is too stable for our actuator. If we are measuring the response of the real system it can even be dangerous to make it unstable.
Cohen-Coon method

This method does not require the parameters $T_u$ and $K_u$ which means we don’t have to tune the system to the edge of stability in order to calculate the optimal gains. A simple step response of the system is enough to measure dead time $T_0$, process gain $G_p$ and time constant $\tau$.

The method is more appropriate for systems with large dead time in comparison to the Ziegler-Nichols method. The result of the Cohen Coon method is again a $\frac{1}{4}$ decay rate of the oscillation, similar to the Ziegler-Nichols methods. The gains are calculated according to the following table.

<table>
<thead>
<tr>
<th>Controller</th>
<th>Controller gain</th>
<th>Integral time</th>
<th>Derivative time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>$K_C = \frac{1.03}{G_p} \left( \frac{\tau}{T_d} + 0.34 \right)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PI</td>
<td>$K_C = \frac{0.9}{G_p} \left( \frac{\tau}{T_d} + 0.092 \right)$</td>
<td>$T_i = 3.33 \cdot T_d \frac{\tau + 0.092 \cdot T_d}{\tau + 2.22 \cdot T_d}$</td>
<td></td>
</tr>
<tr>
<td>PD</td>
<td>$K_C = \frac{1.24}{G_p} \left( \frac{\tau}{T_d} + 0.129 \right)$</td>
<td>$T_i = 0.27 \cdot T_d \frac{\tau - 0.324 \cdot T_d}{\tau + 0.129 \cdot T_d}$</td>
<td></td>
</tr>
<tr>
<td>PID</td>
<td>$K_C = \frac{1.35}{G_p} \left( \frac{\tau}{T_d} + 0.185 \right)$</td>
<td>$T_i = 2.5 \cdot T_d \frac{\tau + 0.185 \cdot T_d}{\tau + 0.611 \cdot T_d}$</td>
<td>$T_i = 0.37 \cdot T_d \frac{\tau}{\tau + 0.185 \cdot T_d}$</td>
</tr>
</tbody>
</table>

Lambda method

The Lambda tuning method is again based on the knowledge of the system that we obtain by measuring its step response. In contrast to the Ziegler-Nichols and Cohen-Coon methods, the Lambda method gives the user a chance to choose the time constant of the controlled system response and calculate the gains based on the desired speed of the response. The method also returns a more stable system than the previous two methods.

For tuning the PI controller, the two equations below are used. $K$, $\tau$ and $t_0$ are the open loop step response characteristics. $\tau_{cl}$ is the desired closed loop time constant.

$$K_P = \frac{\tau}{K(t_0 + \tau_{cl})}$$

$$K_I = \frac{K_P}{\tau}$$

In Dewesoft X’s Lambda tuning, the fast, medium and slow loop speed corresponds to the following values of $\tau_{cl}$:

<table>
<thead>
<tr>
<th>Loop speed</th>
<th>$\tau_{cl}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast</td>
<td>$\tau$</td>
</tr>
<tr>
<td>Medium</td>
<td>2 $\tau$</td>
</tr>
<tr>
<td>Slow</td>
<td>3 $\tau$</td>
</tr>
</tbody>
</table>

Dead time

Processes with large dead time compared to the time constant are difficult to control. The Cohen-Coon method can handle the processes with $T_0$ up to $2\tau$. 
but when the dead time is larger than $2\tau$, the method returns poor results. The problem with dead time is that the initial response of the system is not intuitive: a larger controller gain does not return faster response. But if the gain is set very high, the system will be prone to large overshoots when the dead time has passed.

A typical example of a dead time dominant process is a heated water tank with a long pipe between the tank and the outlet. If we are measuring the water temperature at the outlet and controlling the heater power, it is inevitable that it will take a certain time before we see any effect of the heating at the outlet. Although the water gets hot very quickly, it is the long pipe that causes the dead time. The second problem is that when the hot water has arrived at the outlet and we start to decrease the controller gain, there is nothing we can do about the hot water already in the pipe â€” if the gain was high in the initial phase, the water in the temperature of the water could be way higher than the set point.

The tuning method used in Dewesoft X for dead time dominant processes uses the equations below to determine the controller gains. The derivative gain is omitted.

$$K_P = \frac{0.36}{K \cdot SM}$$
$$K_I = \frac{3}{T_0} \cdot K_P$$

$SM$ is the stability margin. A value of 1 should give a response with overshoot and $\frac{\pi}{4}$ amplitude decay ratio, which we already learned is not very stable. A higher value of $SM$ makes the system more stable, but consequently the response is slower. In Dewesoft X’s PID tuner, you can choose between the following three settings for the $SM$:

<table>
<thead>
<tr>
<th>Tuner setting</th>
<th>SM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low stability, fast loop</td>
<td>2</td>
</tr>
<tr>
<td>Medium stability, medium loop</td>
<td>3</td>
</tr>
<tr>
<td>High stability, slow loop</td>
<td>4</td>
</tr>
</tbody>
</table>
Controlling the propeller rotational frequency in Dewesoft X

A two-blade propeller was mounted to a DC motor. The DC motor was driven by a PWM circuit, which was controlled by the analog signal out of the analog out channel of the Dewesoft Sirius. Rotational encoder was connected to the back end of the motor shaft and used as a feedback sensor. The sensor was connected to the counter channel of the Dewesoft Sirius. The figure below schematically shows the setup.

The PWM generator we used is only capable of generating unipolar voltage, which means we cannot brake the propeller. It is not ideal, but the propeller has a tendency to decrease its speed quite fast so it is not really problematic. Therefore when the control voltage from the output channel is negative, the voltage at the motor is zero.

We want to control the rotational frequency by applying the right voltage to the motor. The first step is to measure the response of the system to the step voltage input. We will be using the following variables in the process.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Dewesoft channel name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process variable</td>
<td>Rotational frequency of the propeller</td>
<td>Propeller rotational frequency [RPM]</td>
</tr>
<tr>
<td>Controller variable</td>
<td>Voltage at the Analog Out channel</td>
<td>Control AO voltage [-10 ... 10 V]</td>
</tr>
<tr>
<td>Controller variable feedback</td>
<td>Actual voltage at the Analog Out channel</td>
<td>AO voltage feedback [-10 ... 10 V]</td>
</tr>
</tbody>
</table>

Note that we will be measuring the actual control voltage at the analog output connector. This is because there is a delay between the software command (Control AO voltage) and the actual output channel response due to the operating system \&\#39;\&\#39; remember that Windows is not a real-time system. To minimize this delay, set the PID Loop Output Time to 0.01 s (10 ms) in the PID Control Setup. The software will thus update the Controller variable at this rate. The typical delay of the operating system will be between 20 and 30 ms.

Note: in case you are controlling a very slow process like air temperature in a room, you could set the PID loop output time to a larger value to decrease the number of small valve actuation.

The first we can do is to model the system on paper. We want to find the voltage to rotational frequency transfer function for our system. The DC motor torque can be modeled by:

![Image 2: Schematic setup of a two-blade propeller mounted on a DC motor, which is controlled by the analog signal](https://example.com/image)

![Image 21: Settings of the PID-controller type](https://example.com/image)
\[ T_P = \frac{k_T \cdot U}{R} - \frac{k_i \cdot \omega}{R} \]

\( k_T \) (motor torque constant), \( k_i \) (a product of \( k_T \) and \( k_e \), the back-emf constant) and \( R \) (motor winding resistance) are motor constants, but we are now interested in the variables that change with time (voltage \( U \), rotational frequency \( \omega \), torque \( T_P \)). We will rewrite the motor torque equation with the constants merged:

\[ T_P = C_T \cdot U - C_i \cdot \omega \]

The torque from air drag that is opposing the motor torque can be modeled by:

\[ T_D = C_Q \cdot \omega^2 \]

\( C_Q \) is a constant depending on propeller parameters like diameter, number of blades, lift and drag coefficient of the propeller airfoil, twist etc. The equation of motion for the system can be written as follow:

\[ T_P - T_D = \omega \cdot J_P \]

\( J_P \) is the mass moment of inertia of the propeller, again a constant. Combining the equations we arrive at the following expression:

\[ \dot{\omega} = \frac{1}{J_P} \left( C_T U - C_i \omega - C_Q \omega^2 \right) \]

The system is not linear since the aerodynamic drag is proportional to the square of the velocity. It is common practice in linear control theory to linearize the equations about a certain steady state condition. Then the linearized equation is valid for small steps about this condition. The optimal gains then depend on the initial condition, but that can be solved with the so called gain scheduling: a map of gains for many different conditions of operation. For example, the flight control algorithm of the F16 fighter aircraft uses gain scheduling to address hundreds of flight conditions with a linear controller. Dewesoft X PID at the moment does not support gain scheduling directly, but one could try to implement it with the help of the sequencer for some slow process control.

Coming back to our propeller model, if we linearize the system about the condition of \( \omega = 0 \) and combine the constants into new constants, we get:

\[ \dot{\omega} = K_i U - K_\omega \]

This equation can be transformed to the frequency domain to arrive at the following transfer function:

\[ \frac{\omega(s)}{U(s)} = \frac{K_i}{s + K_\omega} \]

Click for full derivation
\( f(x_0) \) is zero at our linearization point. The value in brackets is a constant (the value of the derivative at the linearization point) that we denote \( K_{\text{ff}} \). We also denote the product in front of \( U \) as a new constant \( K_U \). Keeping in mind that the function represents our system for small perturbations of the variables, we can simplify:

\[
\dot{\omega} = K_U \cdot U - K_\omega \cdot \omega
\]

From here we perform the Laplace transform:

\[
\omega(s)s = K_U U(s) - K_\omega \omega(s)
\]

Rearranging gives us the transfer function:

\[
\frac{\omega(s)}{U(s)} = \frac{K_U}{s + K_\omega}
\]

We immediately recognize the equation of a first order system. The two constants depend on the motor windings and propeller geometry, but we don't have to bother with that since the time constant and process gain of the system are easily obtained by measuring the step response, which is our next step.

In the PID Control setup, we choose the Open Loop option to disable the feedback loop and the controller.

![Image 22: Open loop type Control system](image)

Our initial condition will be at the control voltage value of 1 V when the propeller rotates at about 670 RPM. This is due to the fact that the encoder we are using is not very accurate at low RPM (it has a resolution of 1024 points per rotation). We will introduce a 1.5 V step in control voltage (from 1 V to 2.5 V) and measure the open loop response.
The result is a typical first-order system response. As seen on the figure above the time constant of the system is about 415 ms. The first cursor is placed at the step of the controller output voltage (purple line). The second cursor is placed at the point where the frequency reaches 63% of the final value. Dewesoft X displays the time between the cursors at the bottom right corner of the screen.

The dead time of this process is very low, about 0.6 ms, which can also be measured from the step response. Such a low dead time is negligible for our purpose so we will assume the process has zero dead time. We will artificially introduce a longer dead time later in this tutorial to see its effect on the system.

From this knowledge of the system we select the Ziegler-Nichols and Lambda method for the tuning of the controller. The Cohen Coon method is appropriate for the systems with the larger time constant while the Dead Time method is appropriate for the systems with large dead time â€” both of which are not true for our system.
Propeller control: Ziegler-Nichols tuning

For the Ziegler-Nichols tuning method we need to find the ultimate gain. This is the proportional gain at which the process value will start to oscillate if the integral and derivative gain are set to 0. In the PID setup, we only set the P gain to a value from which we will then increase it to find the ultimate gain.

$K_P = 0.07$

$K_P = 0.08$
In the two figures above you can see the response of the system for $K_P = 0.07$ and $K_P = 0.08$. At $K_P = 0.07$, the controller just manages to damp the oscillations, while this is not the case anymore at $K_P = 0.08$. Therefore the latter is the ultimate gain $K_u$. The period of oscillation, also known as the ultimate time $T_u$, is found to be 0.13 s.

There are a few interesting phenomena to notice in these two figures that also apply to the rest of the tutorial.

- The AO voltage feedback lags behind the Control AO voltage. The Control AO voltage is the voltage calculated by the PID controller and it is updated every 10 ms (this is the period we set in the PID Control Loop Output Time option in the setup). The AO voltage feedback is the actual voltage at the analog output channel of the SIRIUS instrument and its delay is changing between about 15 ms to 25 ms. Remember again that Windows is not a real-time system!
- The voltage at the output channel is of course also not following the calculated Control AO voltage out of the channel range of +/- 10 V (we could limit the calculated Control AO in the PID setup to match the range of the output channel).
- The propeller frequency is decreasing slower than it is increasing. This is because the actual voltage at the propeller motor is never negative as the PWM generator only generates voltage in one direction. This means we have the following saturation limits:
  - calculated control voltage: unlimited
  - output voltage of the AO channel: +/- 10 V
  - motor voltage: 0-10 V

We can see from the above figures that the ultimate gain is approximately 0.08, since the controller just manages to damp the oscillations at the $K_P$ of 0.07. Using the tuner, Dewesoft X calculates the controller gains for us:

<table>
<thead>
<tr>
<th>Controller</th>
<th>$K_P$</th>
<th>$K_I$</th>
<th>$K_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PI</td>
<td>0.036</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>PID</td>
<td>0.048</td>
<td>0.8</td>
<td>0.0007</td>
</tr>
</tbody>
</table>

The system responses of all controllers are pictured below:

**P-controller**
Before discussing the results we will first tune the controller with the Lambda method.
Propeller control: Lambda tuning

To use the Lambda tuning method we need to further analyze the step response of the system to calculate the process gain from the values of $dCO$ and $dPV$. Note from the step response plot that the figures are:

$dCO = 1.5 \text{ V}$
$dPV = 640 \text{ RPM}$

We already calculated the time constant (0.415 s) and dead time (0 s). With this data, the Lambda tuner in Dewesoft X calculates the following gains for the PI controller:

<table>
<thead>
<tr>
<th>P-gain</th>
<th>I-gain</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0022</td>
<td>0.0052</td>
<td>Fast</td>
</tr>
<tr>
<td>0.0007</td>
<td>0.017</td>
<td>Slow</td>
</tr>
</tbody>
</table>

The response of the system for both options is plotted in the next two figures.

**Fast**

![Image 30: Fast system response](image)

**Slow**

![Image 30: Fast system response](image)
Note that the time scale on the plots is different than on the Ziegler-Nichols response plots because the Lambda response is much slower. The results are discussed in the following section.
Propeller control: Tuning summary

The following table shows the main characteristics of the systems tuned with different methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Time constant [s]</th>
<th>Overshoot [%]</th>
<th>Settling time [s]</th>
<th>Steady-state error [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZN P</td>
<td>0.019</td>
<td>32</td>
<td>0.24</td>
<td>-8.6</td>
</tr>
<tr>
<td>ZN PI</td>
<td>0.015</td>
<td>80</td>
<td>1.4</td>
<td>0</td>
</tr>
<tr>
<td>ZN PID</td>
<td>0.015</td>
<td>60</td>
<td>1.1</td>
<td>0</td>
</tr>
<tr>
<td>Lambda Fast</td>
<td>0.36</td>
<td>1</td>
<td>2.1</td>
<td>0</td>
</tr>
<tr>
<td>Lambda Slow</td>
<td>1</td>
<td>0</td>
<td>4.5</td>
<td>0</td>
</tr>
</tbody>
</table>

In the three cases tuned with Ziegler-Nichols methods we can clearly see the impact of each gain. The I-gain reduces the steady-state error to 0 while introducing oscillations. The D-gain allows a higher P-gain for the same stability (fewer oscillations, shorter settling time).

The response of the Ziegler-Nichols controller is much faster, but also has higher overshoot. Comparing the fast Lambda method and Ziegler-Nichols tuned PI controller, the settling time is not dramatically shorter for the Ziegler-Nichols controller, while the overshoot is 80%, which could be undesired in a specific application. If our PWM generator was able to decelerate the motor, the overshoots would have been smaller, though.
Dead time

We will now also consider a system with a large dead time. Since our motor is not very powerful, it is safe to block it for a few seconds to artificially introduce dead time. The following figure shows the response of a Ziegler-Nichols PI controller with the same gains as in the previous case when there was no dead time.

The controller gains are completely inappropriate. During the dead time the control voltage is increasing all the time, which causes large overshoot when the propeller is released. The control voltage then does not react quickly to the overshoot because of the large integral gain â€” during dead time, a large negative error accumulated, which needs to be counteracted by a positive error for a long enough period for the sum to diminish. Comparing the areas between the set point curve and the propeller frequency curve during dead time and during the full speed segment, we notice they are similar.

If we now use the Dead time method in Dewesoft X’s PID tuner, the calculated gains are as follows:

<table>
<thead>
<tr>
<th>P-gain</th>
<th>I-gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0004</td>
<td>0.0004</td>
</tr>
</tbody>
</table>

The response shown in the figure below is much more acceptable. Because of low gains, the large control voltage does not rise to very high levels during dead time. Since the Dead time method depends on the dead time, the gains would have been even lower if the dead time was larger.
Image 23: Because of low gains, the large control voltage does not rise to very high levels during dead time.
Integrator windup

There is another way to conquer large dead time. In the PID setup we can choose the PI (anti-windup) controller type. This will prevent the accumulation of the integration error when the actuator is in saturation, which happens during dead time. If this type of controller is chosen, the Ziegler-Nichols method provides very good response also in case of a large dead time (figure below). Note that the control voltage stops increasing when the controller notices that there is no response of the process variable (propeller frequency).

We have to set the windup limits of the output channel in the setup for the anti-windup to function properly. A practical value would be just below the actuator saturation level.