# PID Control Design Made Easy

By Murad Abu-Khalaf, Rong Chen, and Arkadiy Turevskiy

Tuning a PID controller appears easy, requiring you to find just three values: proportional, integral, and derivative gains. In fact, safely and systematically finding the set of gains that ensures the best performance of your control system is a complex task. Traditionally, PID controllers are tuned either manually or using rule-based methods. Manual methods are time-consuming, and if used on the hardware, can cause damage. Rule-based methods do not support unstable plants, high-order plants, or plants with little or no time delay. PID control also involves design and implementation challenges, such as discrete-time implementation and fixed-point scaling.

Using a four-bar linkage system as an example, this article describes a method that simplifies and improves the design and implementation of PID controllers. This method is based on the PID Controller blocks in Simulink® and the PID tuning algorithm in Simulink Control Design™.

## The Four-Bar Linkage System: Control Design Goals

Four-bar linkage (Figure 1) is used in a wide range of applications, including car suspensions, robot actuators, and aircraft landing gears.

The control system consists of two elements: feedforward control and feedback PID control. Feedforward control inverts plant dynamics—it handles the major motion of the mechanism by taking into account the nonlinear behavior. Feedback PID control keeps positioning errors small in the face of modeling uncertainties and external disturbances. This article focuses on the design of feedback PID control.

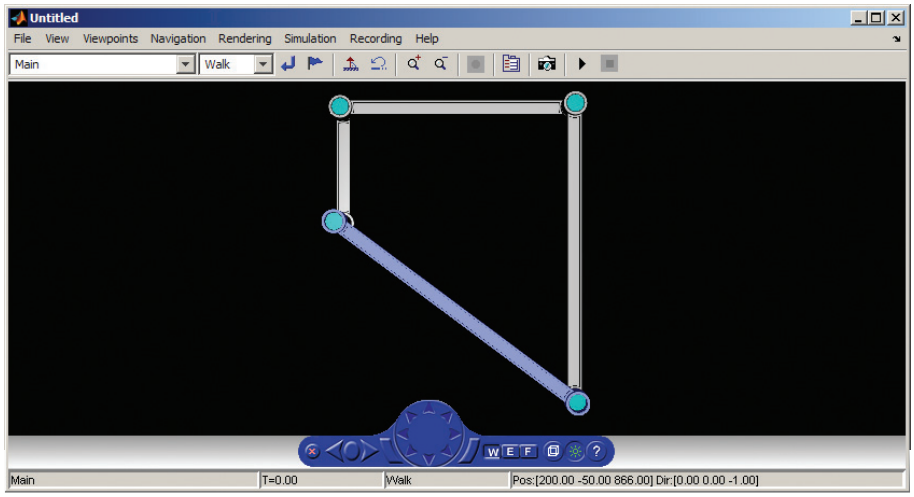


Figure 1. *Four-bar linkage mechanism, with the stationary lower link shown in blue.*

The PID controller takes the error signal between the desired and actual rotation an-

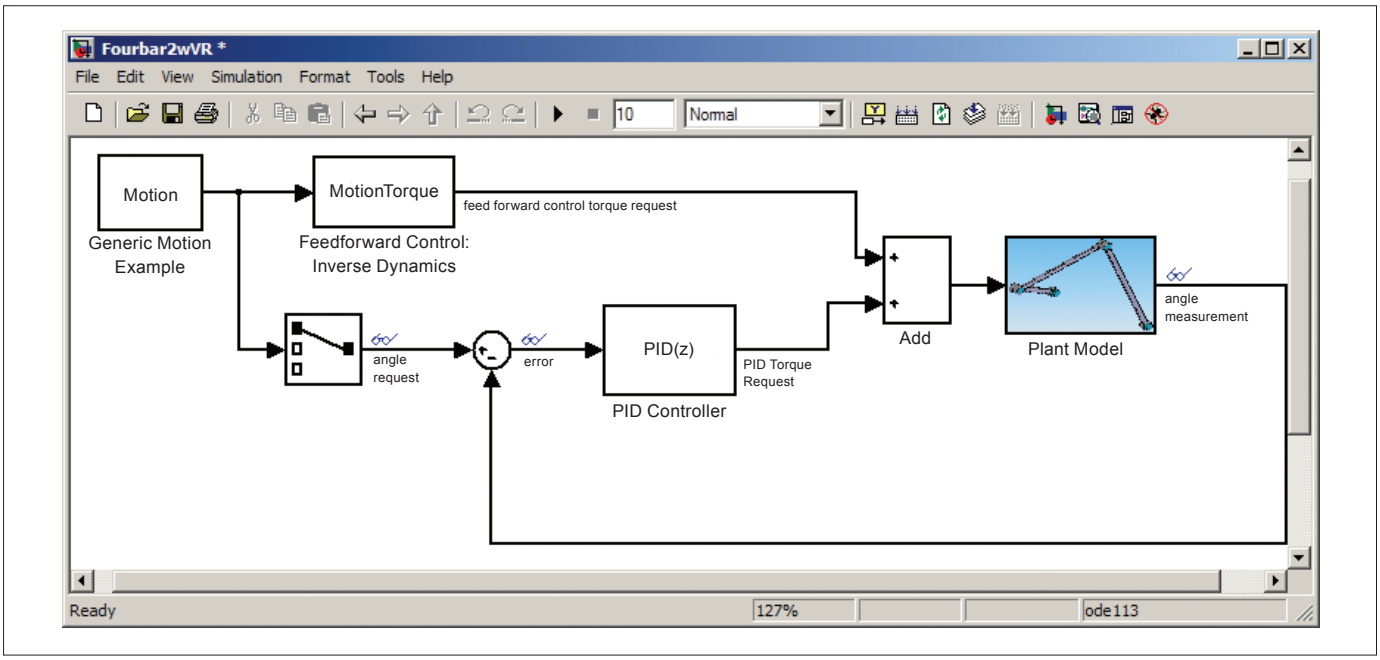


FIGURE 2. *Four-bar linkage system controller architecture.*

gle of one of the links and creates a torque request (Figure 2). This request is added to the torque request from the feedforward controller, and the sum signal is used to drive a DC motor that actuates rotation of the joint connecting the links. The controller must stabilize the operation of the plant. It must also provide fast response time and little overshoot. Because the controller will be implemented on a fixed-point processor with 16 bits, it needs to be in discrete-time form, and the gains and calculated signals must be scaled accordingly.

## Configuring the Closed-Loop System and Tuning the Controller

The plant model consists of a four-bar linkage mechanism modeled in SimMechanics™ and a DC motor modeled in SimElectronics®. To create the controller architecture shown in Figure 2, we simply add a discrete-time PID Controller block from the Simulink Discrete library. With the closed-loop system configured, we are ready to tune the controller.

To do that, we open the PID Controller block dialog box, specify controller sampling time, and press "Tune" to open the

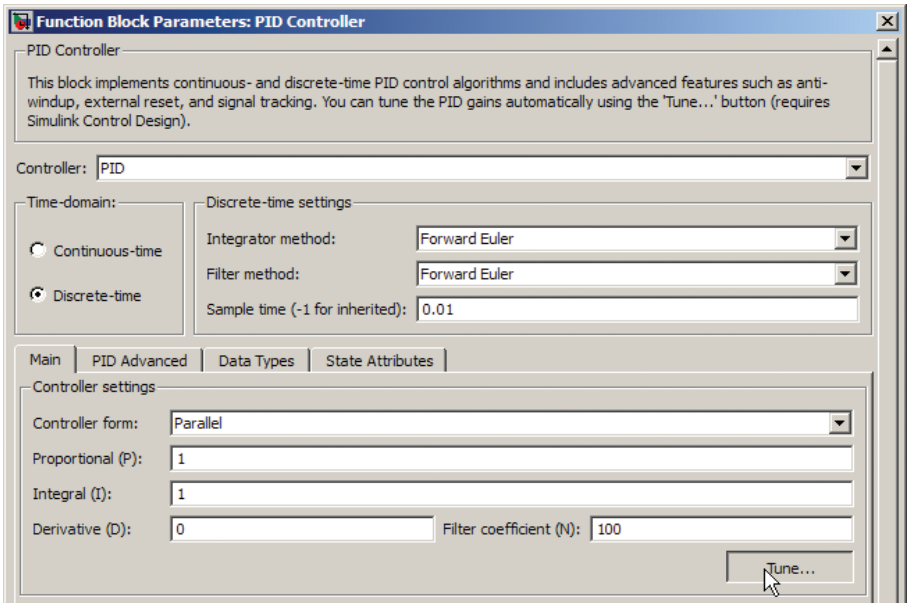


FIGURE 3. *PID Tuner, opened from the block dialog box.*

PID Tuner (Figure 3). Simulink Control Design linearizes the plant at the current operating point and derives the linear time invariant (LTI) plant model seen by the PID Controller block in this feedback control loop. Computational delay associated with sampling is automatically taken into account. Using an automatic tuning method, Simulink Control Design then generates the initial gains of the PID controller. This tuning method imposes no limits on plant order or time delay, and it works in both continuous and discrete time domains.

Figure 4 shows the setpoint tracking response of the closed-loop system with this initial PID design. If the controller performance is satisfactory, we press "Apply" to update the values of P, I, D, and N gains in the PID Controller block dialog box. We can then test the performance of our design by simulating the nonlinear model and looking at the results (Figure 5). We can also tune our design interactively using a simple slider to make the controller faster or slower (Figure 4).

## Preparing for Implementation

To prepare the controller for implementation on a 16-bit microprocessor, we scale it for the fixed-point arithmetic supported by the chip.

Using the "Data Types" tab in the block dialog box, we apply the settings required for fixed-point design (Figure 6). We can obtain these settings automatically using the Fixed-Point Tool in Simulink. We then run the simulation using fixed-point settings to verify that the fixed-point design results closely match the results we obtained when the controller gains and signals were implemented as double-precision values.

## Generating Production Code

With the PID controller prepared for implementation, the final step is to use Real-Time Workshop Embedded Coder™ to generate C code (Figure 7). To test this code, we replace the PID Controller block with the generated

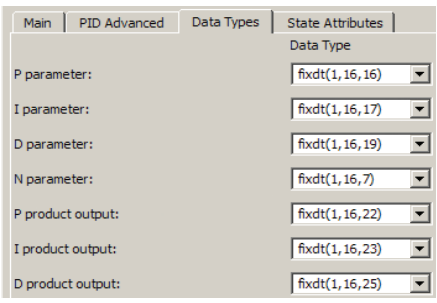


FIGURE 6. *Fixed-point settings for implementing the PID controller on a processor with 16-bit fixed-point architecture.*

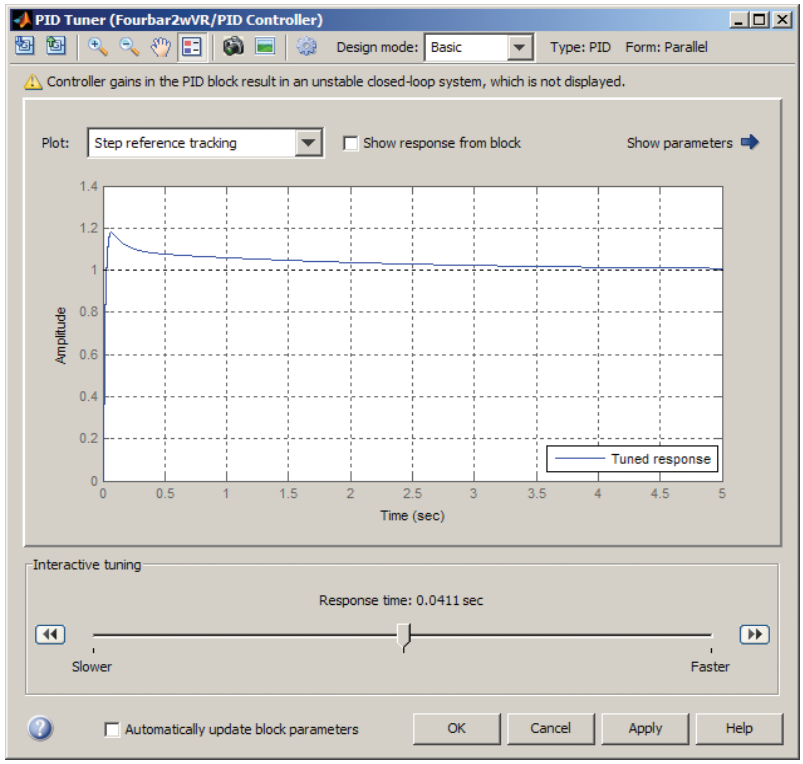


FIGURE 4. *Initial design generated by the PID Tuner.*

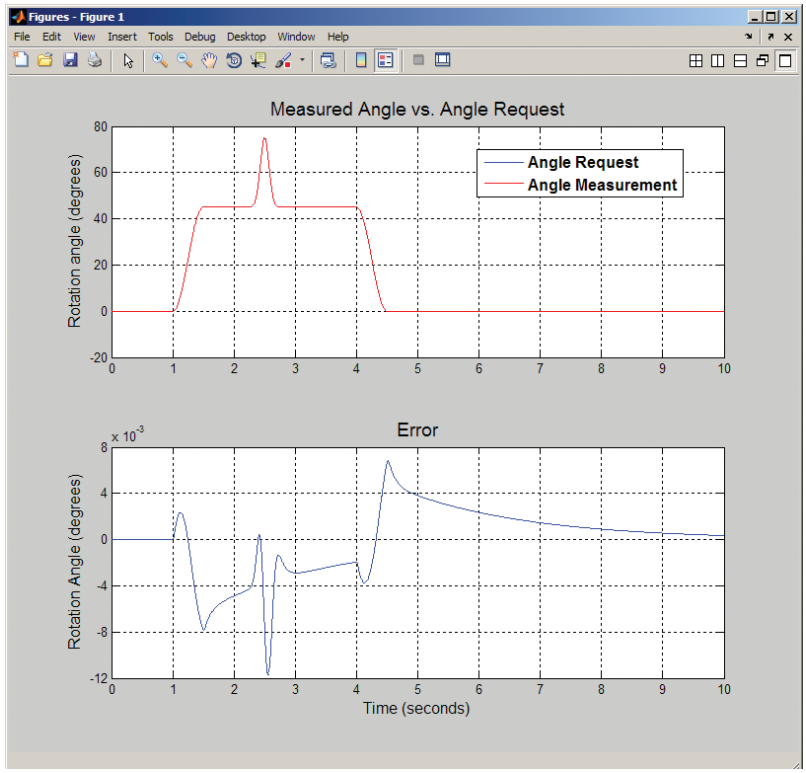


FIGURE 5. *Simulation results for the four-bar linkage model.*

```c
#include "fixed.h"
#include "fixed_private.h"

int16_T error;
int16_T torque_request;
D_Work DWork;
void fixed_step(void)
{
  int16_T FilterCoefficient_m;
  FilterCoefficient_m = (int16_T)((int32_T)(((int16_T)(5403L * (int32_T)error >>
    13U) - DWork.Filter_DSTATE) << 4U) * 17893L >> 14);
  torque_request = (((int16_T)(12475L * (int32_T)error >> 14U) >> 1) +
                    (DWork.Integrator_DSTATE >> 2)) + (FilterCoefficient_m >> 1);
  DWork.Integrator_DSTATE = (int16_T)((4643L * (int32_T)error >> 13U) * 5243L >>
    19U) + DWork.Integrator_DSTATE;
  DWork.Filter_DSTATE = (int16_T)(5243L * (int32_T)FilterCoefficient_m >> 16U) +
    DWork.Filter_DSTATE;
}

void fixed_initialize(void)
{
  torque_request = 0;
  (void) memset((void *)&DWork, 0,
                sizeof(D_Work));
  error = 0;
}
```

FIGURE 7. *C-code implementation of the 16-bit, fixed-point PID controller. The code is generated from the PID Controller block.*

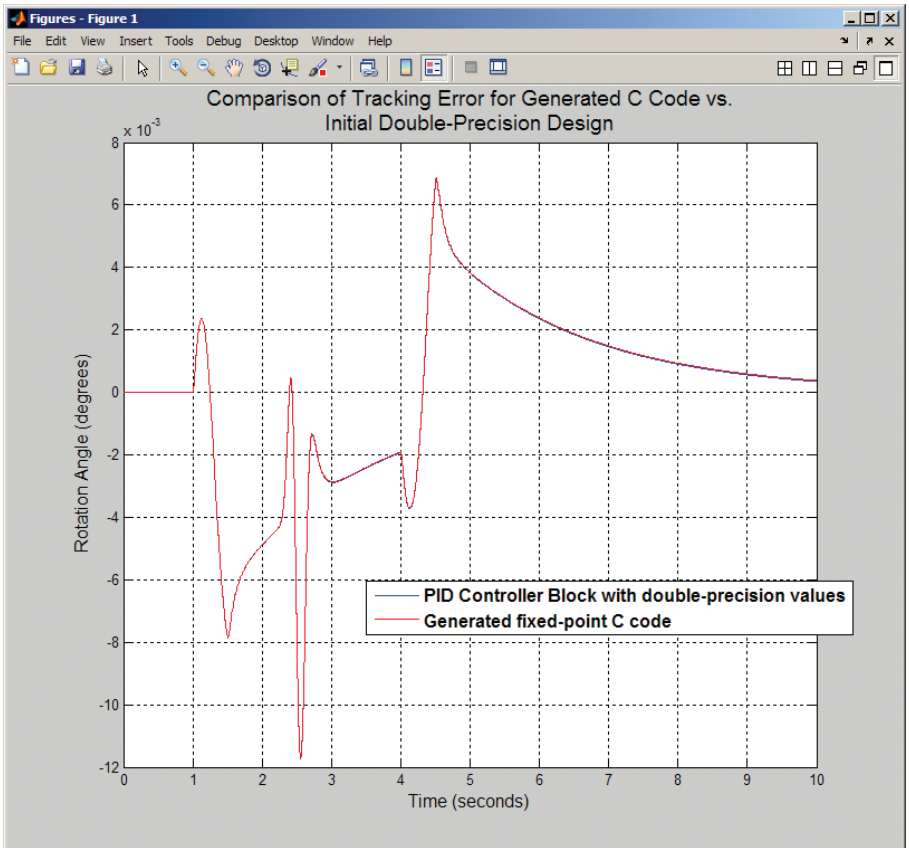


FIGURE 8. *Simulation results comparing the performance of the generated C code with the performance of the double-precision PID Controller block.*

C code and run the code in closed-loop simulation. We can do that by using Real-Time Workshop Embedded Coder to automatically create a Simulink block that invokes the generated C code.

We can now run the simulation using the C code that will run on the actual processor. Simulation shows that the generated code produces results that closely match the results obtained with our PID Controller block with double-precision values (Figure 8). We can now deploy this code to the processor and start controlling our four-bar linkage in real time. ■

## Learn More Online

**DEMO: Automated Tuning of Simulink PID Controller Block**
**www.***mathworks.com/pid-tuner*

**DEMO: Design a Simulink PID Controller (2DOF) Block for a Reactor**
*www.mathworks.com/pid-controller*