40

Racing Vehicle Control Systems using PID Controllers

Nic Melder and Simon Tomlinson

40.1 Introduction

A *control system* is defined as the entirety of the mechanical, physical, or digital machinery, including the environment in which it operates (the *plant*), and the device used to manage it (the *controller*). In a real-world control system, whenever we are trying to achieve a desired value (such as setting a temperature via a wall thermostat or a speed using a vehicle's cruise control), a controller is used to modify the externally defined target input to achieve the desired output. Characteristics of the plant can be accounted for and designed into the controller, such that the time taken to reach the desired target can also be optimized. The response of the plant at different rates of change of the desired target value can also be analyzed and controlled.

In order to design the controller, the control engineer will typically analyze the plant and represent it as a series of linear differential equations, using complex mathematical techniques to obtain the required responses under all conceivable steady state and transient conditions. Thankfully, in computer simulations or games, the effect of a badly designed controller is never catastrophic as it might be in a chemical plant, for example, and so instead of a fully specified controller, the much simpler and easier to operate Proportional

Integral Derivative (PID) controller can be used. This PID controller is "good enough" that it can be used in most situations where we are trying to control a single input target and output value system and absolute precision is not necessary. For the curious, a more complete introduction to control theory and plant analysis is available [Ogata 09] as is advanced instruction on bespoke controller design [Warwick 96].

40.2 Basic Control Theory

Any system that we are trying to control contains two parts, the plant and the controller. The plant is what is being controlled but is not necessarily a physical machine. In a home heating system the plant is the boiler, radiator, and the room environment; the controller is the thermostat. In racing games, the plant is typically the digitally simulated vehicle and track.

There are two types of control strategies: open-loop and closed-loop control as shown in Figure 40.1. The key difference between these is that closed-loop control feeds a measurement of the output of the plant back into the input to the controller. This type of system is also referred to as a feedback system. An open-loop controller does not measure the actual output as a reference and so cannot adjust for the effects of the environment. For example, consider a car driving along a flat surface with throttle held at 20%. The vehicle will accelerate slowly until it reaches a maximum speed. Now, if the car is driving up a constant incline or decline, the speed will decrease or increase to a new speed. Similarly, if a different car undertakes the same test, it will reach a different maximum speed. That is, an open-loop system is not able to compensate for any external changes or disturbances to the system.

In contrast, the closed-loop circuit takes the output value (Y) and feeds that back to the input where it is subtracted from the required target value (R). This error value (e) is then acted on by the controller. In the constant speed scenario above, a closed-loop controller will allow the vehicle to maintain the constant required speed even if there is a change in environment or plant (an incline or a different car) because the measured speed error will alter and the controller can react by adjusting the throttle. Furthermore, with a closed-loop controller, the car will accelerate more quickly to its desired speed by starting

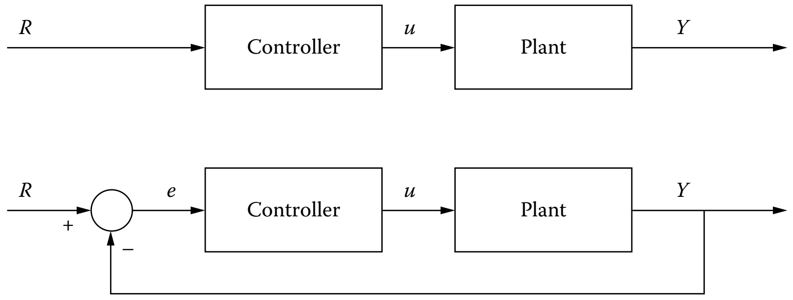

Figure 40.1

An open-loop control system (top) and a closed-loop control system (bottom). Note the output feeding back into the input for the closed-loop system.

with 100% throttle and then reducing the throttle input as it approaches the required speed, that is, the error decreases.

From this example it would seem that open-loop control systems have no practical use. However, if there is a direct relationship between the input and output value, then open-loop control is sufficient. Turning the wheels on a vehicle (for steering) can be done using open-loop control as the mechanics in a vehicle's steering system are well defined so that an applied angle at the steering wheel results in a specific angle on the wheels.

While closed-loop control has many advantages over open-loop control, it can suffer from undesirable effects due to a badly designed controller. These include *overshoot*, where the actual value exceeds the target value, *oscillations* or *ringing* where the actual value oscillates around the target value, and *positive feedback* where an increase in error actually pushes up the plant input resulting in a continual runaway build-up around the feedback loop. Figure 40.2 shows the response of a plant to a step of the input target and the different types of responses that can occur.

There are four major characteristics of the closed-loop controller in response to a step change in the required target value, R:

1. *Rise time:* The time for the plant output to rise to 90% of R.
2. *Overshoot:* The peak value the plant output rises above R, if any.
3. *Settling time:* The time it takes for the system to settle to its steady state.
4. *Steady-state error:* The difference between R and the output Y once the system has settled to a constant state.

Even for the simple PID controller, these effects must be addressed when designing the solution.

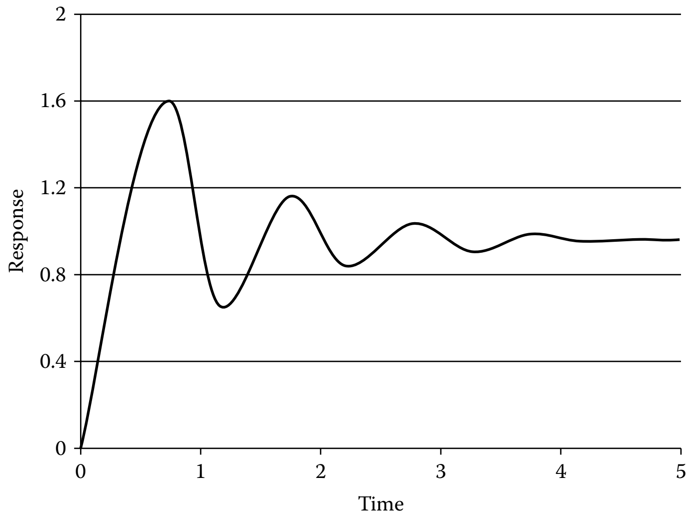

Figure 40.2

A typical response to a step input. This is not the ideal desired response but shows an overshoot, diminishing oscillations, and settling to a constant state.

Table 40.1 A summary of how the three parts of a PID controller affect various aspects of the output

Response	Rise time	Overshoot	Settling time	Steady-state error
K_p	Decrease	Increase	Minor	Decrease
K_i	Decrease	Increase	Increase	Removes
K_d	Minor	Decrease	Decrease	Minor

40.3 Introducing the PID Controller

The closed-loop PID controller is comprised of three parts. The first term (K_p) is proportional to the error (e) between the required target value (R) and the plant output (Y). The second term (K_i) is proportional to the integral of the error, and the last term (K_d) is proportional to the derivative of the error. The output is the sum of these three components:

$$e(t) = Y(t) - R(t) \tag{40.1}$$

$$u(t) = K_p \cdot e(t) + K_i \cdot \int e(t)dt + K_d \cdot \frac{d}{dt}e(t) \tag{40.2}$$

Calculating the error, integral error, and differential of the error is straightforward; it is the selection of *gain* for K_p, K_i, and K_d that give the controller its characteristics. Note that the error can be either positive or negative, that is the measured speed could be above or below the required speed. Similarly K_p, K_i, and K_d can be negative. The effect of increasing the different parameters on the four main system characteristics is summarized in Table 40.1.

40.3.1 Proportional Control

Proportional control simply multiplies the error by a constant gain (K_p). If K_p is too large, this can cause the system to start to oscillate due to too much input being applied to the plant. This causes the output to overshoot, which could then require a large correction in the opposite direction. Positive feedback can also occur where the average error continuously increases, resulting in an unstable and uncontrollable system.

For a vehicle driving around a circular track (the error is the distance from the center of the vehicle to the center of the road), if K_p is too large, a small error will result in the vehicle turning too much, then overcorrecting and ultimately weaving across the line uncontrollably with larger and larger amplitude. When K_p is small, the vehicle will slowly turn in the correct direction to reduce the error. However, the smaller the error gets, the smaller the processed input to the vehicle will become. In this example, Y will never reach R so the error will never become minimized to 0. It can be seen that our vehicle will never reach the racing line as the steering corrections become minuscule. As the steering corrections are insufficient, the car will settle a fixed distance away from the racing line. This is known as the steady-state error.

40.3.2 Integral Control

In order to counter the steady-state error caused by the proportional term, an integral component can be added to the controller. This integrates the error over time, which is then added back into the input. Therefore, as the steady-state error persists, the integral term builds up, pushing the input further in that direction, thus reducing the steady-state error.

One effect of using an integral component is that oscillations can be introduced into the output. Once the steady-state error has been reduced to zero, the integral error will still be nonzero and contribute to the controller output. It is this residual error (or memory) that causes the overshoot. Furthermore, as the integral and proportional term try to correct for the overshoot, oscillation around the target value can develop. However, unlike positive feedback instability, this oscillation will diminish over time.

40.3.3 Derivative Control

Derivative control generates an input to the plant based upon the rate of change of the error. Because of this, it can be used to either dampen or exaggerate the effects of the proportional term. If the error is changing slowly, the derivative component will have little effect, but if the error suddenly increases, a positive derivative output will also increase giving the system a "kick start." This effect can be used to reduce the rise time of the controller response. Conversely, if K_i is negative, the derivative component can be used to hold back sudden changes in the input, that is, it dampens sudden changes so that the system can focus on long-term changes in R.

40.4 Implementing the PID Controller

When implementing the PID controller, there are a number of additional factors that must be accounted for. This section discusses some of these more fine grain design points.

Limit and reset the integral error. Because the integral error builds over time, if the system has a constant error (such as waiting for the start lights to change) the integral error will be constantly increasing. To counter this, it is desirable to be able to reset the integral error based upon external events (such as the lights changing). Similarly, capping the integral error may also be desirable to limit the effects of the *error memory*.

Maximum history of the error (time constants). In a real-world system, the integral and derivative terms are governed not only by a constant multiplier (gain) but also by what is known as a *time-constant*. For the integral terms, this affects how long in the past the integral is based upon. This effectively limits the amount of "history" stored in the integral, which would otherwise contain information going back as far as the initialization of the controller. This is best implemented by calculating the integral error as a rolling average, that is, on each frame the current integral is reduced by (1–T%) and T% of the current error is added back. This also has the effect of normalizing the integral error.

The choice of the time constant affects the length of the history. A small value will look back in time a significant number of frames while a large value will look only a few frames. Larger histories make the plant output smoother, while shorter histories make the plant output more responsive to changes in R. Using longer time constants does come with a cost, though; it takes longer for the integral error to build up after a step change resulting in a greater lag before the controller responds.

Similarly, the derivative error can be sampled at varying time constants; current and last frame, current and second to last frame, and so on. Using a longer time constant has the effect of making the derivative term less spiky. If you want to use a continuously variable time constant in the derivative, simply estimate interframe error values using interpolation. Optimizing time constants is a useful secondary tool in shaping the controller characteristics.

Filtering input data. If the input data (R) is noisy, the derivative term can fluctuate in an undesirable manner. Smoothing the error data with a low-pass filter (i.e., a rolling average of the last few frames) can help to eliminate this. However, this is effectively adding another integrator. If a low-pass filter is placed either in the input stage or in the feedback loop, this can result in a further time lag in response to changes in R. Alternatively, a separate low pass filter can be added at the input of each individual PID term so that the filtering can be adjusted to suit each component.

Varying the K-coefficients. It may be the case that in different situations, different K values work better. For example, at low speed the vehicle may require much larger K values to get it to start to move, compared to when it is running at high speed. In this case, it would be useful to linearly vary the K values as a function of speed. Be careful with this, though, because if the coupling is too great, a hidden positive feedback loop can be set up, resulting in instability. Similarly, the values used when the vehicle is on tarmac may be unsuitable when driving on dirt. For this case, we would want to use a state-based system where the K values are determined by the surface the vehicle is on.

Record metrics for tuning and debugging. When tuning the K values it is desirable to use a fixed, easily repeatable scenario and record metrics every frame to see the effect that the parameter changes have made. For a steering controller, this would involve recording the vehicle position perpendicular to the racing line. Furthermore, there may be situations where the controller does not behave as expected and being able to record and visualize the controller internals can greatly aid in debugging instabilities.

Determining success. When tuning the controller it is important to be able to define what a good controller is. This "success" value could be time to hit target speed, time taken to do a lap around the track, minimum distance traveled around the track, or smoothness of controls, depending on the game requirements. Properly identifying the correct metric is important and, once such a metric has been found, it can be used as a basis for automated learning.

Using an error offset. Instead of trying to minimize the error, by having a nonzero target error it is possible to give some variety to the systems being controlled. For example, a steering controller would normally try to minimize the distance of the vehicle to the racing line, but by using an error offset, this would cause the vehicle to drive a short distance to the left or right of the racing line. Similarly, if different vehicles use a different error offset when setting a desired speed, there will be some variation in the final speed, which may be appropriate in a traffic simulation. Such effects can also be achieved by offsetting the target value (R) directly, but in some cases, it is easier to work within the controller error domain.

40.5 Designing and Tuning the PID Controller

When designing a PID controller, it is important to properly associate parameters from the tactical AI layers with the input target value R, parameters in the vehicle physics simulation with the output, and also recognizing the significance of the closed-loop error value.

If using a controller designed to manage speed, the input is our desired speed, the output is a value fed into the pedal system, and the error is the speed difference. In this case, the scaled controller output might vary from –1.0 (need to slow down) to +1.0 (need to accelerate). This value will need to be further distributed into actual brake and accelerator pedal inputs.

When used in a steering controller, the input may be the vehicle's required position, the scaled output is fed into the vehicle's steering wheel, and the error is the perpendicular distance of the vehicle to the racing line.

Understanding the input, output, and error allows the specific characteristics of the controller to be defined. For example, steering should be reasonably responsive, but smooth, for normal driving. Additionally, we should certainly avoid over-steering and zigzagging, so we should aim for an overshoot free characteristic. Speed control can also be smooth, but in collision avoidance situations, we might switch in a set of PID values tuned for fast response and accept some oscillations.

Tuning PID controllers is normally done through trial and error, but it is possible to analytically calculate what the gain values should be. However, this requires a good knowledge of the mechanics of the system and is beyond the scope of this article. It is important to only vary one parameter at a time so that the effects on the metric graph in the training scenario are not obscured.

Once the desired characteristics are determined, the following sequence is recommended for tuning the controllers:

1. Set all gains (K) to 0 and increase K_p until the system behaves in a desirable manner with no overshoot or oscillation. Tuning this value first is advisable, as it will generally have the biggest effect on the output.
2. Increase K_i to eliminate the steady-state error.
3. Adjust K_d to reduce any overshoot or reduce the settling time as required.

In practice, it is often found that K_d and K_i are approximately half of K_p. As a general rule, if the system is unable to reach its desired value, increase K_i. If it oscillates, reduce K_p and K_i. If it is too slow to change, then increase K_d. Remember that the properties of the plant also play a part, so if there is a lot of time lag in the plant, the controller may need to work harder to compensate.

40.6 Adaptive Control

In a system that is constantly changing (such as the weight of a vehicle as it burns off fuel or when traveling over different surface types), the initial K values chosen for our PID controller may cease to be appropriate and the controller must adapt. State switching or simple linear variation of the controller parameters has already been outlined, but more intricate methods are also available.

One adaptive control method is to use *gain scheduling*. This is where the controller parameters are directly adjusted by an external factor, such as the vehicle's speed or the surface grip, through a simple linear relationship or a more complex polynomial equation. Gain scheduling can work well when the system is largely dependent upon only one value, but can become particularly difficult to tune when the values are dependent upon multiple factors.

Model reference adaptive control (MRAC) is a form of adaptive control where a model of the system characteristics is predetermined (e.g. given an error of 20, we want to reduce the error at a rate of 5 per second). While the system is running, a comparison between the measured response and the reference model response is made, and the difference is used to adjust the controller parameters. Although MRAC is a more complex form of adaptive control, it is much easier to understand and to tune than other, simpler methods. More information on how the described adaptive control schemes work and can be implemented can be found elsewhere [Forrester 06].

40.7 Predictive Control

Predictive control looks ahead into the future to try to respond to changes in R ahead of time and hence compensate for time lags in the plant. A well known technique used in industrial controllers is *model predictive control* [Alba 02], but this type of system is probably overkill in games.

In practice, we can use a simpler form of predictive control where we generate the input based upon what we are expecting to happen, by using knowledge of the problem. Using a steering system as an example, we can change the input from perpendicular line position to use a *runner* object that leads the vehicle [Tomlinson and Melder 13]. In this case, the input is now the angle between the forward direction of the vehicle and the line between the vehicle and the runner, and the controller error is the difference in angles. This is predictive because the runner will begin a turn before the vehicle needs to. Note that the lead distance of the runner is part of the "plant" and needs to be considered when tuning. Similarly, the speed controller can use a predicted speed by looking down the track.

40.8 Other Controller Applications in Racing

There are many more areas that the PID controller can be applied to beyond the speed and steering management discussed so far. Here are a few further applications.

Types of steering controller. When the driving simulation is high grip, requires smooth driving (e.g., Formula 1 or IndyCar), and the vehicles respond quickly, the controller will be tuned for smoothness (K_p is generally low with a small or even negative K_d). In contrast to this, for off-road racing on low grip surfaces where sliding and drifting is expected, the K values are much larger. This is to compensate for the lag between the input and the effect on the vehicle due to the lower grip of the surfaces.

Speed controller—split channels. There are actually two methods of speed control: throttle only and exact tracking. Using the throttle only technique, if the vehicle is going too fast, then engine braking and tire friction is used to slow the vehicle down. Exact tracking also uses the brake to slow down. A real-world racing driver will use engine braking to deal with small reductions in speed and only use braking for larger reductions. Also, because braking force is usually distributed to all tires and acceleration is only through one axle, the characteristics of the PID for acceleration and braking can be different. So, for more realistic speed control, two subchannels of PID controllers should be implemented: an acceleration channel that responds only to positive errors whose output is the throttle and a braking channel responding only to negative errors. A dead band may also be required so that for small negative errors both the throttle and brake are off and engine braking is used.

Stopping at a point. This can be implemented by using an error equal to the distance to the point minus the vehicle's rolling distance. In order to stop at a point, it is necessary to know the vehicle's braking and rolling distance over a range of speeds. The rolling distance is the stopping distance under engine braking and friction. By comparing the distance to the target and the rolling distance of the vehicle at that speed, the difference can be used to control the brake. When both the braking distance and rolling distance are shorter than the actual distance, it is necessary to apply the throttle. Using this method, the author has been able to make the AI drive a distance of 10 m and stop within an average of 2 mm of its target in less than 5 s. This controller was used for the staggered start mode in Codemasters' *DiRT2* and *DiRT3*.

Drifting. Drifting is where the car is deliberately slid sideways along the track so that the direction of movement is different from the orientation of the vehicle; the difference between the two is the drift angle. The controller is set up where the error is the difference between the desired and measured drift angles.

The usual way to instigate a drift is to sharply turn the vehicle and break the traction on the back wheels (either with the handbrake or by applying a huge burst of throttle power). Once the traction has been broken, the angular momentum of the car will continue to push out the rear against the friction of the tires which are sliding sideways. The drift can be controlled by applying more throttle for more angle and less throttle for less angle, to affect the balance between the friction and angular momentum. Although PID controllers can be used to manage drifting, it is very difficult to achieve, and in practice, it is easier to cheat a little and modify the power and grip on the back wheels directly! In normal circumstances, the drift management should be disabled and only enabled when specific drift situations can occur.

Grip loss prediction, recovery, and channel cross-over. In a race where the driver is at the very limit of grip, circumstances may force them to step over that limit on occasion and slide the car a little. The driver must then take positive action to avoid the slide becoming more significant. It is therefore useful to design a PID controller where the error is a direct measurement of how much the tire is sliding, that is, in normal conditions the error is zero but it becomes nonzero when grip is lost. The response to sliding is to reduce the amount of steer, brake, or acceleration that is causing it, so ideally up to three channels of grip loss monitoring are implemented to feed into the brake, accelerator, and steering inputs of the plant. However, these should work alongside the normal PID driving controllers rather than instead of. This makes it necessary to either mix the different channels before input to the plant, or better still cascade the "slide controllers" into the main driving controllers in a sort of gain scheduling scheme. When active, each of the slide controllers should reduce the K values in each channel. This technique is stepping a bit beyond a simple PID controller, but it does allow the AI to realistically test the limits of the car.

Priority mixing. Although state changes can be used to switch the controller parameters, some situations call for a more analog approach. Consider the situation where two cars are driving around a corner alongside each other. The steering can then have multiple possible required values (R), one for corner turning and one to avoid collision, both acting simultaneously. This can still be dealt with using controllers by mixing the two inputs based on a nominal priority value, P:

$$R = \frac{R_1 P_1 + R_2 P_2}{P_1 + P_2} \tag{40.3}$$

Mixing can be applied to the required values (R) before the single PID controller, but if the characteristics of the two inputs are different, it can be implemented as two separately tuned PID channels and mixed before input to the plant (u). If this latter arrangement is used, bear in mind that the feedback loop will be sampling the same steering output Y, so the two PID channels are not separate, but are in fact cross coupled. This can be used to advantage by careful tuning to balance the two channels, but it can also lead to instability.

40.9 Conclusion

The PID controller is one of the simplest, yet most versatile, tools that an AI programmer has in order to control specific aspects of a physically based system. It acts as a link between the tactical layer of the racing AI, which defines required steering and speed goals, and the vehicle simulation, which takes action toward the goals. PIDs can be used at a fairly simple level to achieve tailored responses to changes in tactical requirements or can be explored in more depth to deal with driving issues such as grip loss. The closed-loop PID also makes things easier for the programmer, in that it operates by controlling the actual actions of the vehicle on the track, and so the programmer is relieved of the onerous task of having to manually convert abstract tactical requirements into actual driving inputs.

References

[Alba 02] C. B. Alba. *Modern Predictive Control*. London: Springer, 2002.

[Forrester 06] E. Forrester. "Intelligent Steering Using Adaptive PID Controllers." In *AI Game Programming Wisdom 3*, edited by Steve Rabin. Hingham, MA: Charles River Media, 2006, pp. 205–219.

[Ogata 09] K. Ogata. *Modern Control Engineering*. New Jersey: Prentice Hall, 2009.

[Tomlinson and Melder 13] S. Tomlinson and N. Melder. "Representing and driving a race track for AI controlled vehicles." In *Game AI Pro*, edited by Steve Rabin. Boca Raton, FL: CRC Press, 2013.

[Warwick 96] K. Warwick. *An Introduction to Control Systems*. World Scientific, 1996.