INTEGER PI CONTROL WITH INTEGRATOR ANTI-WINDUP

• March 31, 2013 • Coded in C

For many applications, PI control is sufficient, and PID control is nontrivial to implement correctly. This is a PI controller written with integer arithmetic. The implementation uses a struct to hold both the gain parameters and the integrator value, so many controllers can be implemented with the same code. The error is passed as an argument to the function, so care should be taken to make sure that the error computation, which occurs outside the controller in the user's code, does not cause overflow.

Integrator wind-up is a phenomenon that can occur when the output is saturated. This controller checks the output for saturation and, if the P+I terms exceed the desired control range, the new integrator value is not saved if doing so would deepen the saturation. This helps to prevent extreme overshoot during large disturbances or setpoint steps.

This controller is implemented with integer math. To provide fractional gain resolution, a right bit shift is performed on the final result. Adjusting the shift parameter scales the net control gain by a factor of two for coarse adjustment. The gain parameters kp and ki can then be used to make fine adjustsments.

```
/**
 * @file
 * Proportional-integral (PI) control law.
 *
 * This module implements a simple position-type PI controller:
 * <pre>
 *   u = [ kp * e + ki * sum(e) ] >> shift
 * </pre>
 * <tt>shift</tt> is a right bit shift used to scale the output of the
 * controller down from the 32-bit intermediate result.
 *
 * An anti-windup provision is implemented on the PI integrator to prevent
 * deep saturation (aka integrator windup):
 * - The new control output with the latest integrator value is computed.
 * - If the control output exceeds either output limit, <i>and</i> the latest
```

```
 *   change in the integrator is in the same direction, then the new integrator
 *   value is not saved for the next call.
 * - Otherwise, the integrator is saved for the next call.
 */

#include <stdbool.h>
#include "pi_control.h"

/**
 * Proportional-integral (PI) control law.
 *
 * @param[in,out]  p    control parameter and state structure
 * @param[in]      e    error signal
 *
 * @return              control output <code>u</code>
 */
int pi_control (struct PIControl *p, int e)
{
  bool int_ok;      /* Whether or not the integrator should update */
  long new_i;       /* Proposed new integrator value */
  long u;           /* Control output */

  /* Compute new integrator and the final control output. */
  new_i = p->i + e;
  u = (p->kp * (long)e + p->ki * new_i) >> p->shift;

  /* Check for saturation.  In the event of saturation in any one direction,
     inhibit saving the integrator if doing so would deepen the saturation. */
  int_ok = true;

  /* Positive saturation? */
  if (u > p->max)
  {
    /* Clamp the output */
    u = p->max;

    /* Error is the same sign? Inhibit integration. */
    if (e > 0)
    {
      int_ok = false;
    }
  }
  /* Repeat for negative sign */
  else if (u < p->min)
  {
    u = p->min;

    if (e < 0)
    {
      int_ok = false;
    }
  }

  /* Update the integrator if allowed. */
  if (int_ok)
  {
    p->i = new_i;
  }

  return (int)u;
}

/**
 * Initializes the PI control.
 *
 * This function resets the PI integrator to zero.
 *
 * @param[in,out]  p  control parameter structure
 */
void pi_control_init (struct PIControl *p)
{
  p->i = 0L;
}
```

```
/* Header file */
#if !defined(_PI_CONTROL_H)
#define _PI_CONTROL_H

/**
 * @file
 * Proportional-integral (PI) control law header file.
 */

/** PI control data structure.  This structure contains configuration (the
    proportional and integral gain, plus a final divisor), output limits, and
    an integration accumulator (the PI controller's state variable). */
struct PIControl
{
  int kp;              /**< Proportional gain constant */
  int ki;              /**< Integral gain constant */
  unsigned char shift; /**< Right shift to divide */
  int max;             /**< Maximum value */
  int min;             /**< Minimum value */
  long i;              /**< Current integrator value */
};

/* Prototypes */
int pi_control (struct PIControl *p, int e);
void pi_control_init (struct PIControl *p);

#endif /* _PI_CONTROL_H */
```