

Advanced Programming PID Control



Christopher Teslak
Team Ford First



Seminar Outline

- What is PID Control?
- Feedback vs. Open Loop Controllers
- Case study: Wheel angle control
- Proportional Control
- Integral Control (w/ intro to Calculus)
- Differential Control
- Do we really need all this?

What is Control?

- To adjust something so that it operates correctly.
- Real-time control is control with a deadline:
 - Read sensor inputs.
 - Decide what to do given the new info.
 - Take action by writing to outputs.
 - Wash, rinse, repeat (26.2 ms in FRC)

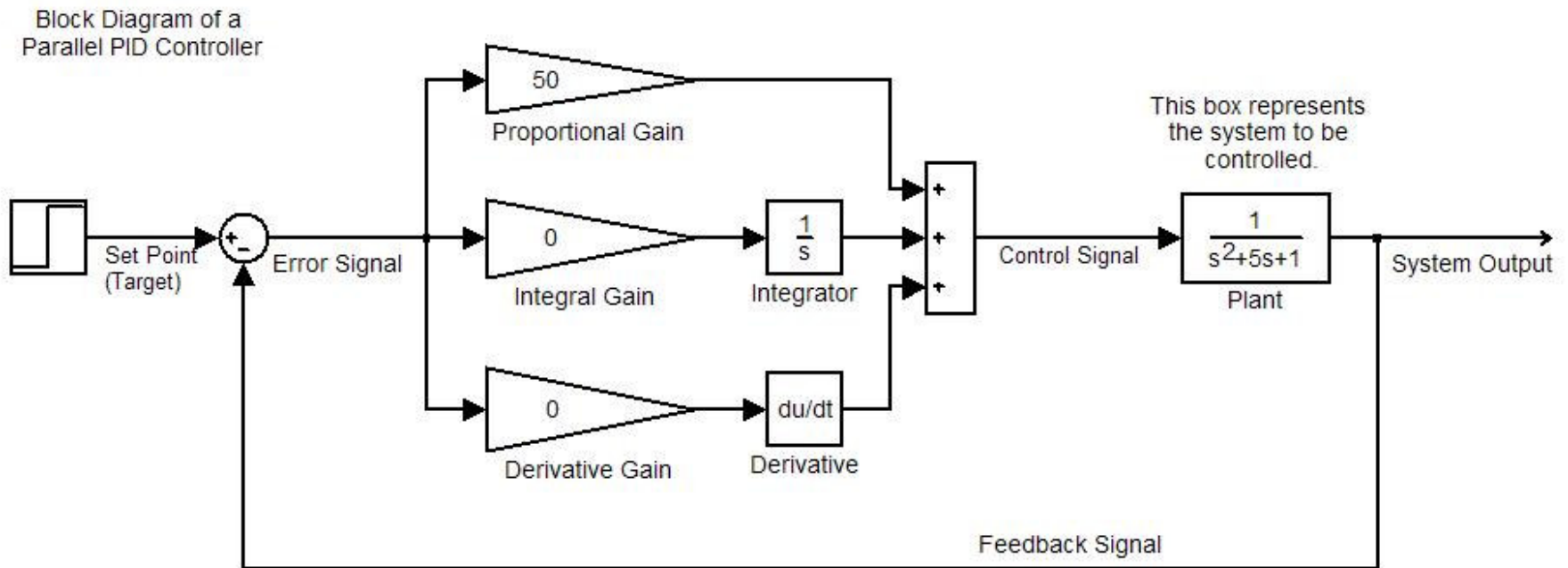
Control Requirements

- Desired system behavior
- Controlled parameter must be observable
- Set of options to adjust the system
- Knowledge of the system

Control Terminology

- **plant** = the system being controlled
- **error** = the difference between the target value and the actual value
- **reference signal** = the target or desired output, sometimes called the “set point”
- **control signal** = the command from the controller to the plant
- **output signal** = the measured value that represents the current state of the plant

Block Diagram of a PID Controller



Open Loop Control

- In open loop designs, the control action is based on good knowledge of how the system being controlled works.
- There is a map or translation in the software of expected output based on measured input.
- This approach is fast and easy to implement.

Feedback Control

- What if you don't understand the system that well? What if the open loop action yields unexpected results or expected results but at the wrong time?
- Feedback control works on the principle of a set point (target) and an error (how far off the system is from the target).
- The FB controller subtracts the measured output from the input target and applies the difference to the control signal in a variety of ways.

Benefits of Feedback Control

- Improves accuracy
- Reduces the effect of
 - changes in the plant characteristics
 - external disturbances
- Response can be made faster

Problems with Feedback Control

- A closed-loop system can be unstable (oscillations that start to grow)
- Can cause too much control activity (trade-off between fast responses and gentle control signals)
- Feedback can sometimes mask problems.

What is PID Control?

- It stands for Proportional, Integral, Differential
- The three components represent the different ways the error signal is applied to the control signal
- It is an old technique.
- Now we use computers, sensors and actuators to implement it.
- It allows for fast, accurate control with minimal overshoot.
- It can be applied to many types of control problems but not all.

History of PID Controllers

- 1788: James Watt introduces mechanical feedback with proportional control by equipping a steam engine with a flyball governor.
- 1933: Taylor Instrument Co. – first pneumatic controller w/ fully tunable P control
- 1951: Swartwout Co. – first electronic controller based on vacuum tube technology
- 1959: Bailey Meter Co. – first solid-state electronic controller
- 1964: Taylor Instruments – first digital controller
- 1975: Process Systems – first microprocessor based PID controller

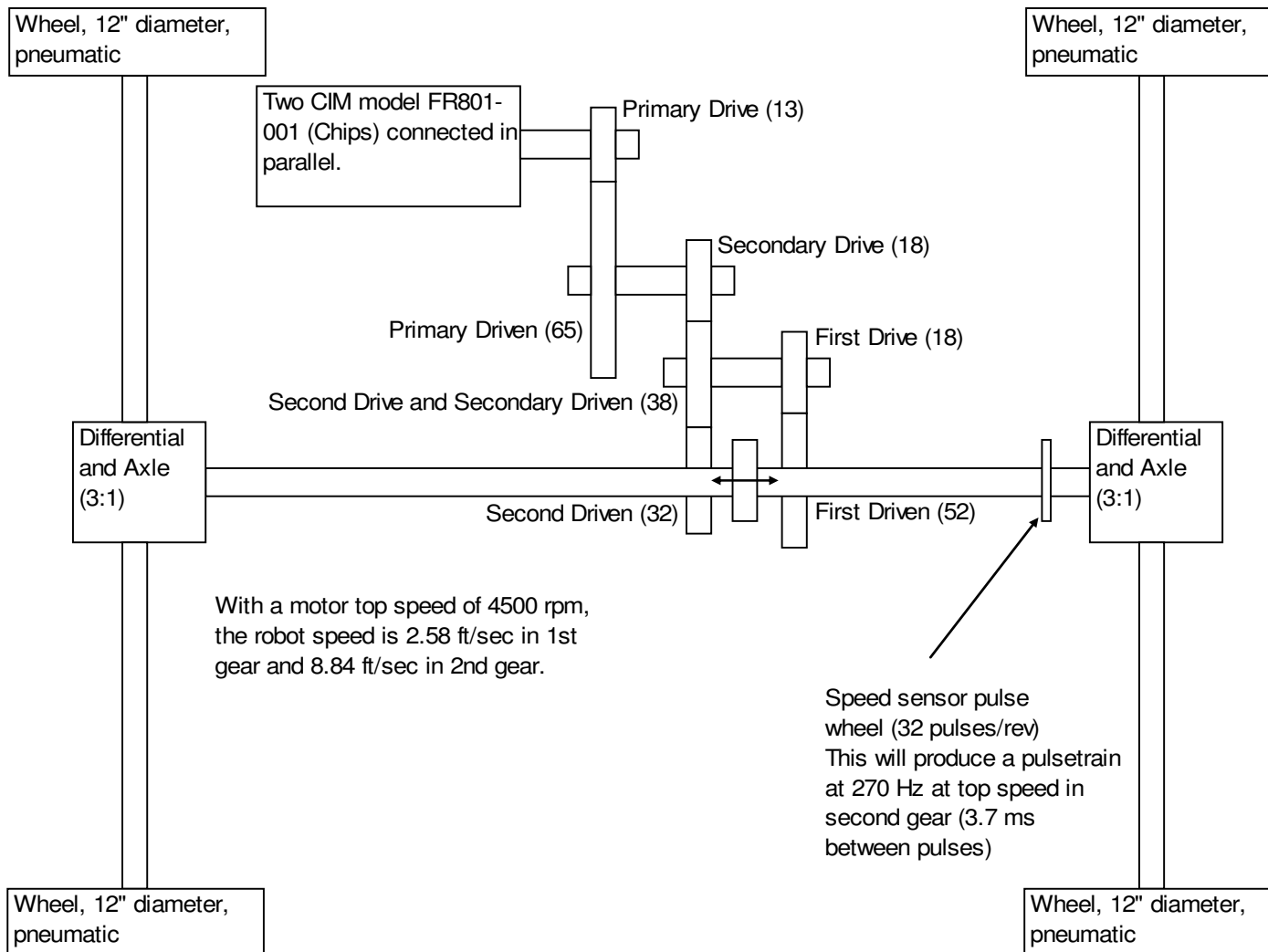
PID Controller Objectives

- To make *plant output* follow the *reference*
- to make response fast, but not oscillatory or unstable
- to reduce the effect of disturbances
- to not move actuators too aggressively

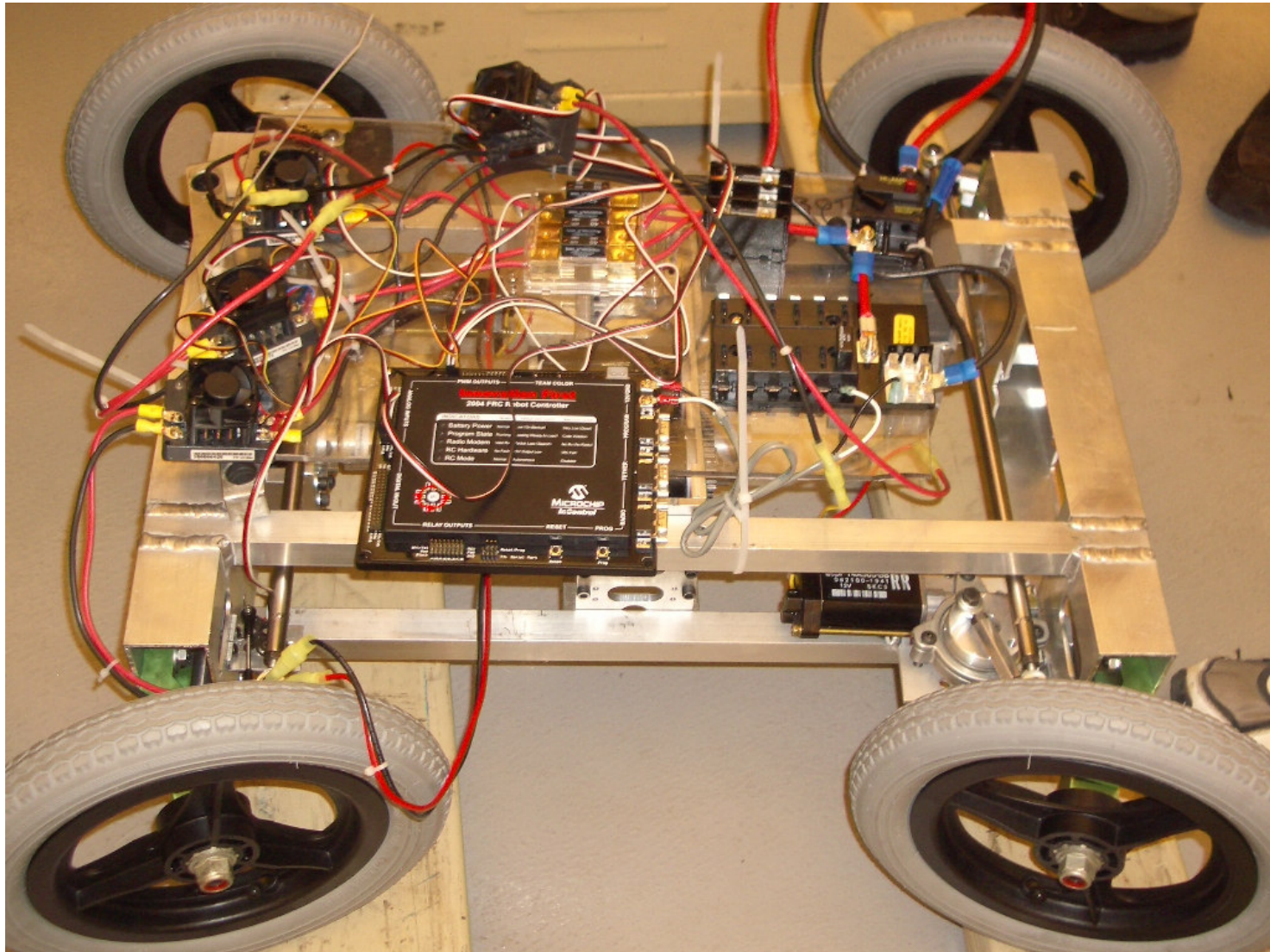
PID Motivation and Limitations

- Motivation
 - Adequate for most applications
 - Simple to get working
 - Easy digital implementation
- Limitations
 - For single-input single-output systems only
 - Difficult to tune to meet precise specifications
 - Subtle differences in implementation causes problems

Robot Drivetrain Design



Robot Top View

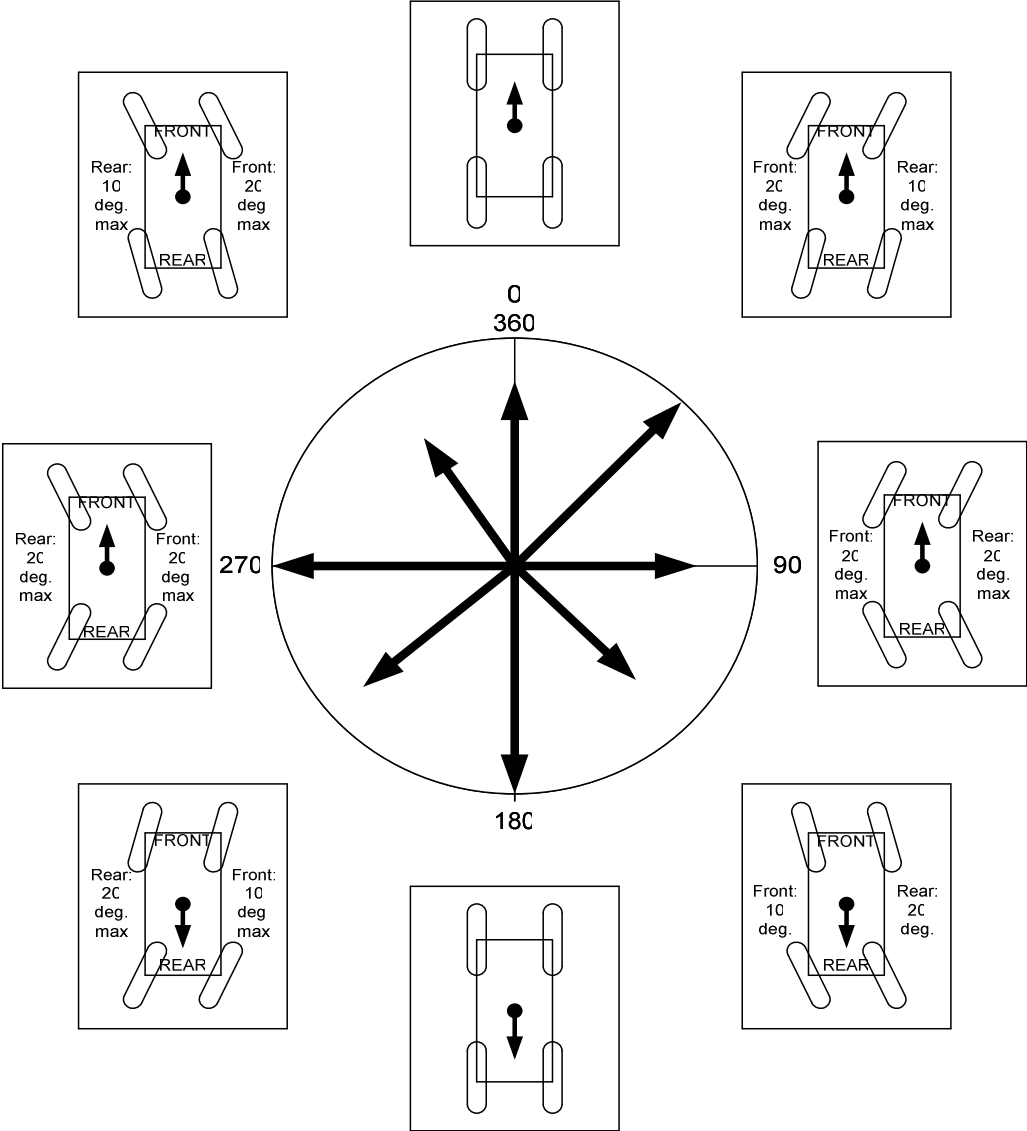


November 13, 2004

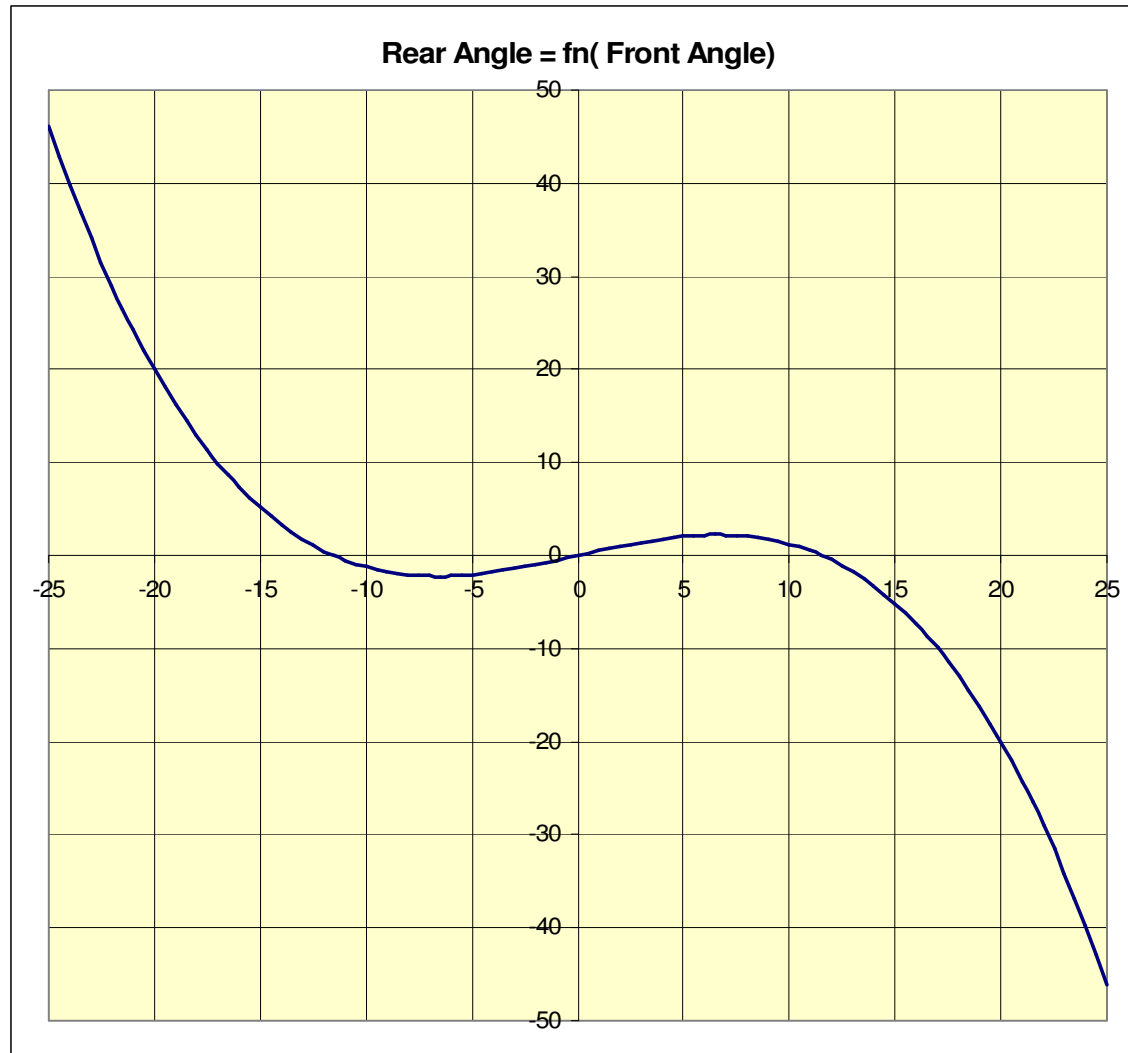
PID Control by Chris Teslak
Team Ford FIRST

16

Mobility Design



Relationship of Wheels

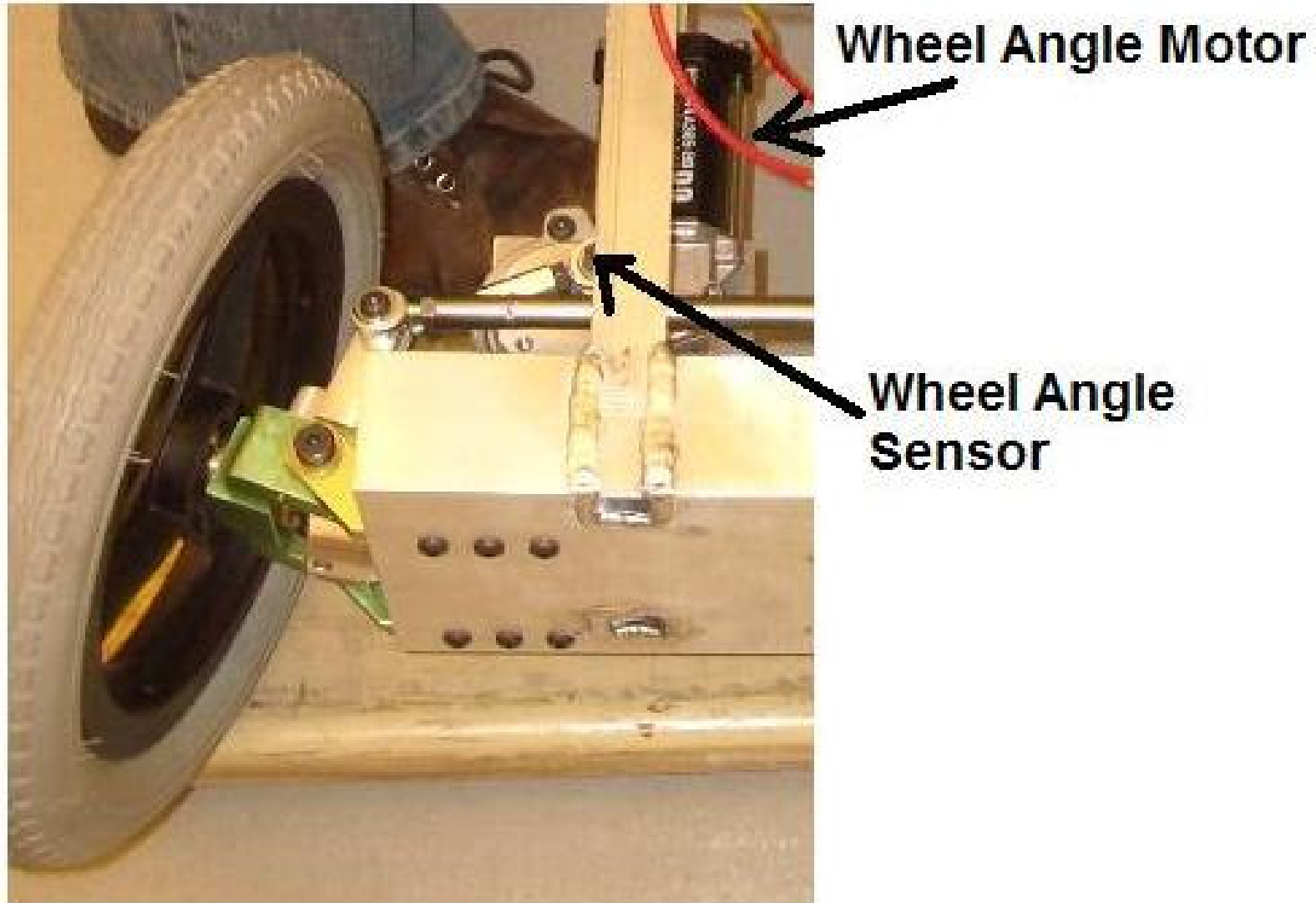


November 13, 2004

PID Control by Chris Teslak
Team Ford FIRST

18

Robot Wheel Angle Control



November 13, 2004

PID Control by Chris Teslak
Team Ford FIRST

19

Proportional Control

Error = Reference Input – Plant Output

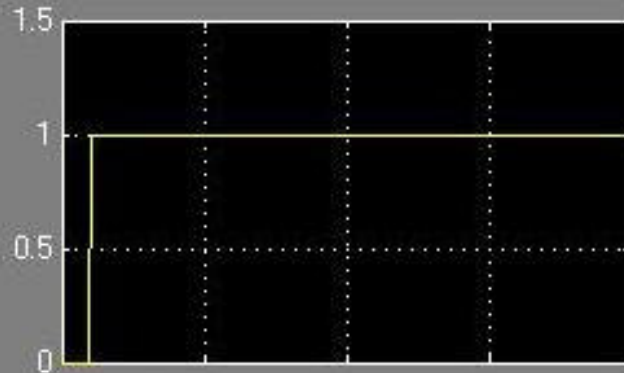
Control Signal = Constant * Error

If the output of the plant matches what you wanted it to be, the error would be zero and so would the control signal.

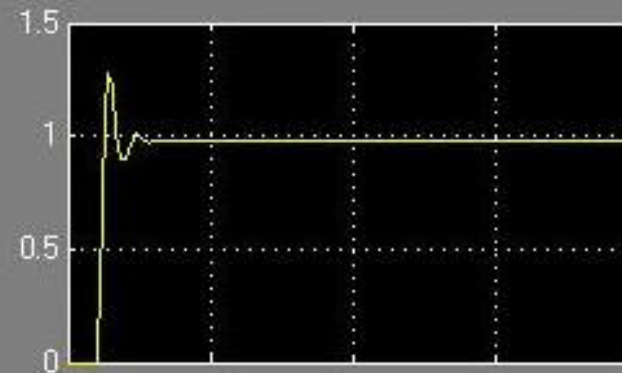
When the control signal is zero, that tells the plant not to change anything.

The constant that multiplies the error is called the proportional gain. Tweaking this number will adjust your controller's sensitivity to the error.

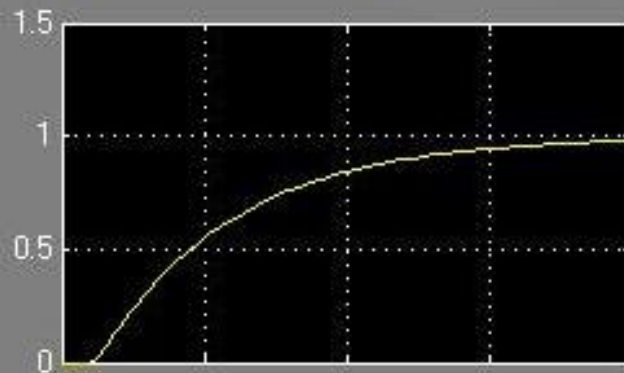
Proportional Control



**Target
(reference signal)**



**Output Signal with
Proportional Feedback**



**Output Signal with
no feedback**

Integral Control

- The purpose of integral control is to eliminate the steady-state or constant error.

Error = Reference Input – Plant Output

Control Signal = Constant * Integral of the Error

- The question is: What is the integral of the error?

What's an Integral?

- The term “integral” is well defined in Calculus.
- In terms of how a digital computer will be programmed to implement one, think of an integral as a sum, like a running total, over a period of time.
- In practical terms, in each loop of the control program, the small constant error is accumulating until it reaches a point where it makes a big correction to the control signal and clears the sum starting to accumulate again from zero.

What Does This Mean to PID?

- The integral of the error is multiplied by a constant, called an integral gain, to become the integral term of the PID controller.
- This is then added to the proportional term and now you have a PI controller.
- The hard part is getting the gain just right. If the control is too aggressive, the PI controller can cause closed-loop instability which may lead to hunting.

Differential Term

- Can fix a hunting controller
- Derivative terms are many times not used because they're not needed or they are more trouble to calibrate than they're worth.
- It does allow the controller to be very fast. In fact in the 1940's the derivative term was called, "pre-act".
- On the other hand, the D term is problematic when the measurement of the output signal is noisy.

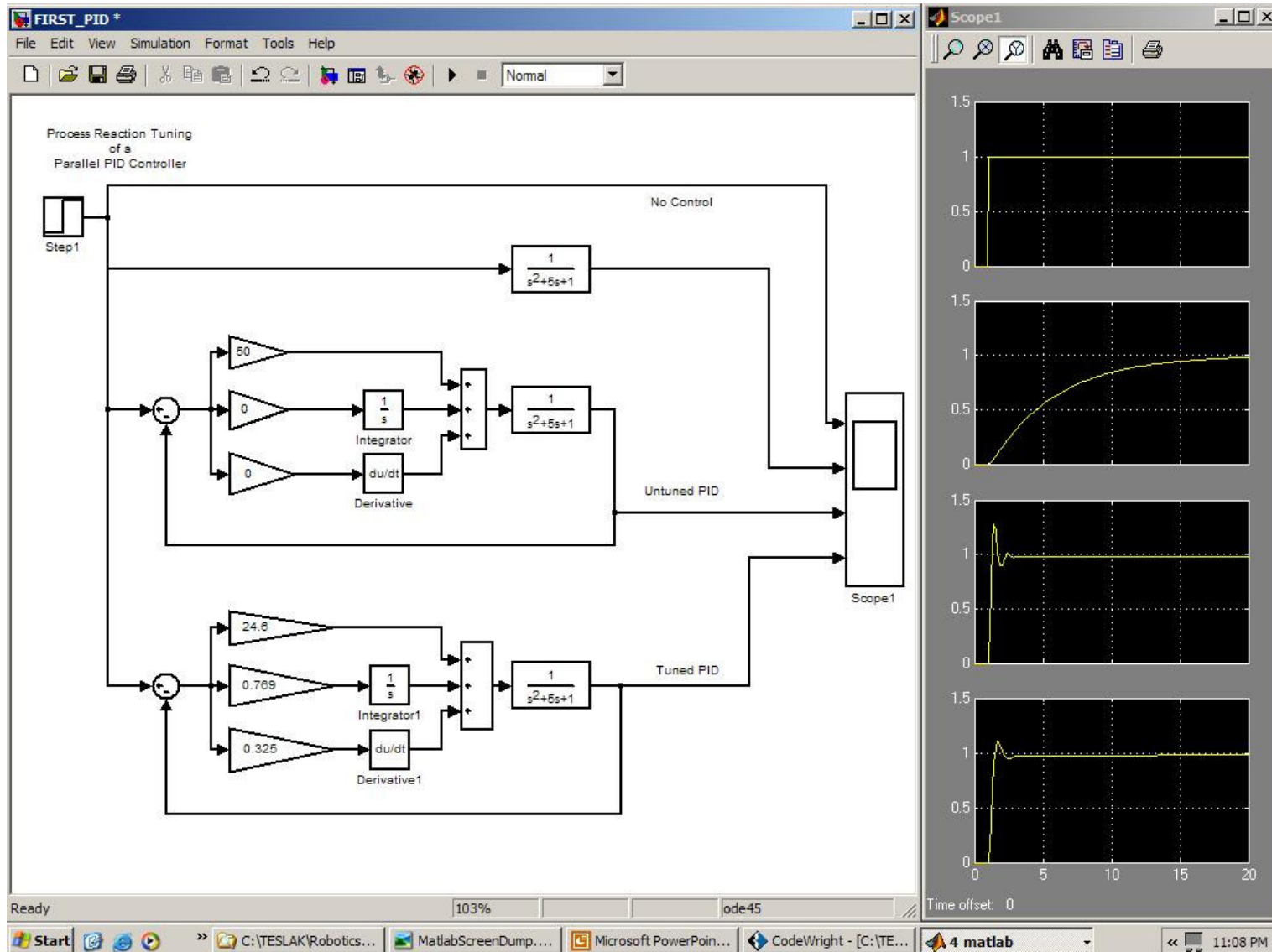
What's a Derivative?

- This is another term that is first introduced formally in Calculus.
- However, we can think of it as the rate of change of something, in this case, the error.
- The derivative term is the constant, called derivative gain, multiplied by the rate of change of error.

Error = Reference Input – Plant Output

Control Signal = Constant * Derivative of the Error

Now Look at What We've Done

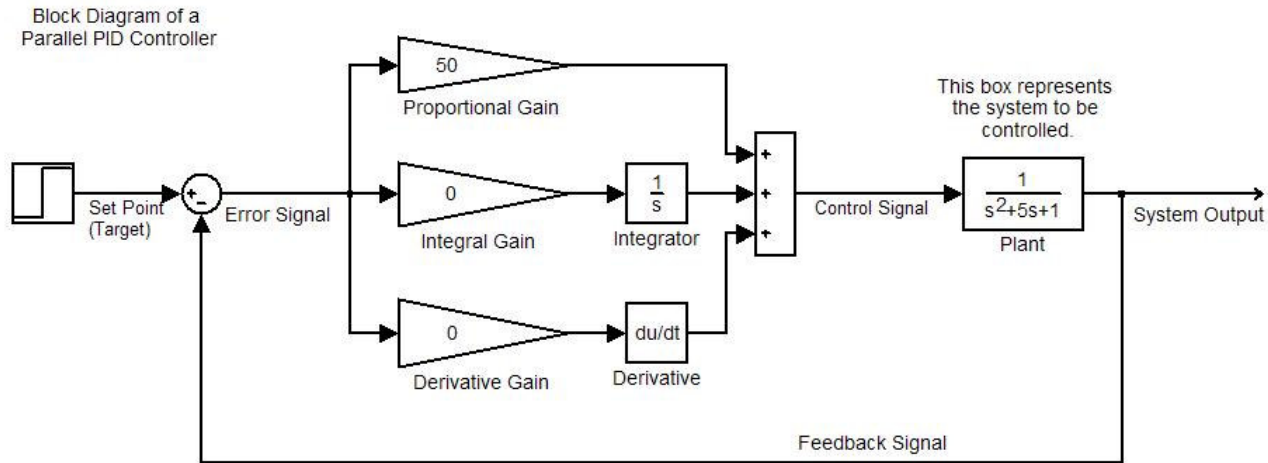


November 13, 2004

PID Control by Chris Teslak
Team Ford FIRST

27

Application of a PID Controller



- Notice that the control signal going to the plant is the sum of the three control terms.
- Also, notice that the control signal is zero when we don't want to change the wheel position. If we want to turn the wheels left or right, the control signal would go positive or negative.
- The motor that physically moves the wheel needs electrical current to do that. Somehow, we need to get a normalized controller output translated into something the motor can understand.

Robot Wheel Configuration

In software, the desired PWM value is commanded using a number between 0 and 254.

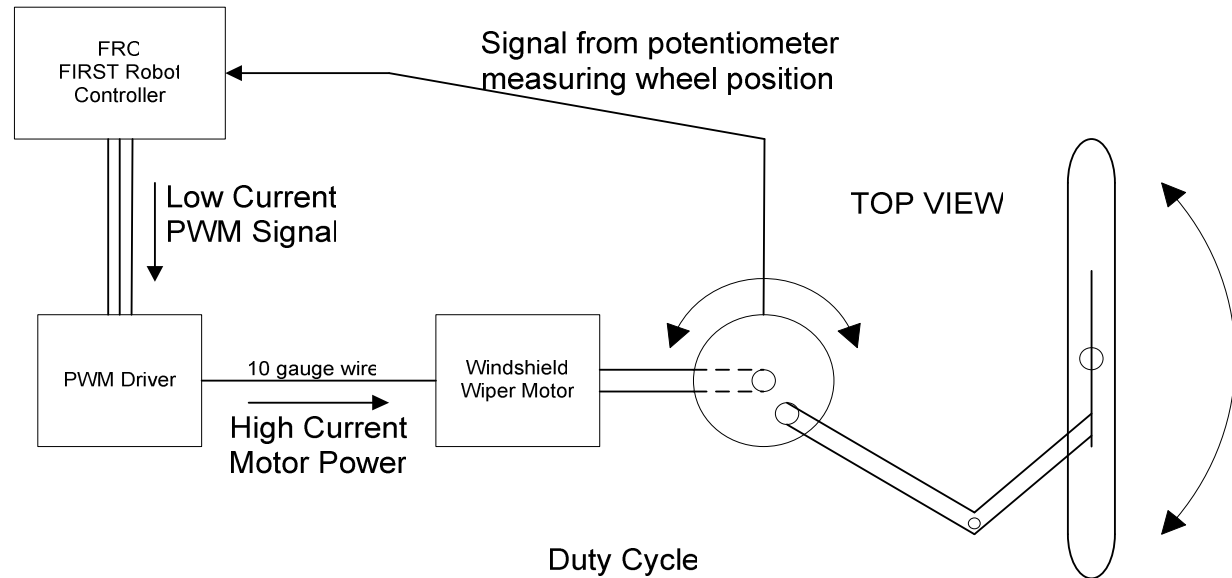
0 0%
 127 50%
 254 100%

PWM is:

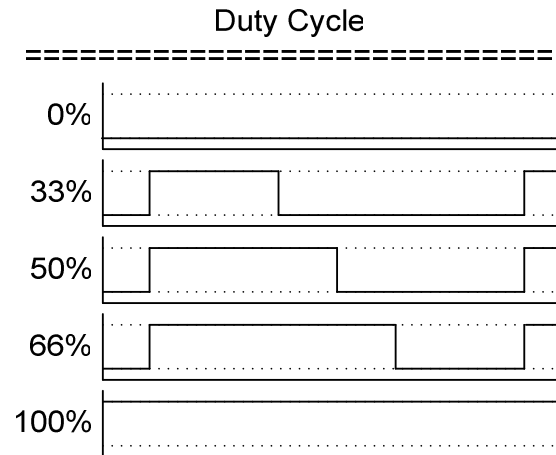
= Pulse Width Modulation

= a fixed frequency square wave pulse train that varies the percent "on" time in proportion to the desired motor current.

= a method of communicating to the PWM Driver how much electrical current to send to the motor.



D.C.	Motor Speed
0%	Full Left
33%	Partial Left
50%	Stop
66%	Partial Right
100%	Full Right



```

/*****\
*
* Robotic Steering Control
*
* RSC.C
*
* by Chris Teslak
*
* November 16, 2003
*
\*****/

int control_steering()
{

    /* MANIFEST CONSTANTS */
    #define WHL_POS_CMD_MIN (-1.0)
    #define WHL_POS_CMD_MAX (1.0)
    #define ERROR_NEAR_ZERO (0.1) /* this equates to 5% of full range of error */

    /* VARIABLE DECLARATIONS */
    auto float joystick_position_normalized = 0; /* -1 = full left, +1 = full right */
    auto float wheel_position_normalized = 0; /* -1 = full left, +1 = full right */
    auto float wheel_position_error_last = 0; /* always will be in the range -2 to +2 */
    auto float delta_time = 0; /* time in seconds since last execution */

    /* STATIC DECLARATIONS */
    static float wheel_position_error = 0; /* don't lose the value upon exiting this function */

    /* CONSTANT DECLARATIONS */
    const float gain_p = 1; /* proportional gain */
    const float gain_i = 0; /* integral gain */
    const float gain_d = 0; /* derivative gain */
    const float mul_joy_pos_norm = 1; /* joystick position multiplier for setpoint calc */
    const float add_joy_pos_norm = 0; /* joystick position adder for setpoint calc */
    const float pid_i_min = -1000;
    const float pid_i_max = 1000;

```

```

/* BEGIN CODE */

    /** INPUT PROCESSING **/

    /* Get the joystick position and normalize it. */
    /* Let's assume the actual position comes as a voltage */
    /* between 0 (full left) and 5 (full right) */

    joystick_position_volts = get_joystick_position();
    joystick_position_normalized = joystick_position_volts / 5 * 2 - 1;

    /* Get the actual wheel position and normalize it. */
    /* Let's assume the actual position comes as a voltage */
    /* between 0 (full left) and 5 (full right) */

    wheel_position_volts = get_wheel_position();
    wheel_position_normalized = wheel_position_volts / 5 * 2 - 1;

    /* If necessary, filter the signals as well. */

    /** CALCULATE THE SET POINT **/

    /* (1) This could be as simple as a linear relationship,
       in which case, you implement it as y=mx+b.
       (2) You could also give yourself more flexibility
       and implement it as a look-up table.
       (3) You may want to introduce a deadband in the
       set point to compensate for "return-to-zero" joystick
       problems. This is easier to implement using look-up table.
    */

    wheel_position_desired = joystick_position_normalized * mul_joy_pos_norm + add_joy_pos_norm; /* y=mx+b */

    /** CALCULATE THE ERROR AND THE TIME **/

    delta_time = get_delta_time(); /* returns the time in seconds elapsed since last execution */
    wheel_position_error_last = wheel_position_error; /* save the error value from the last loop */
    wheel_position_error = wheel_position_desired - wheel_position_normalized; /* don't use absolute value here */

```

```

/** CALCULATE THE PROPORTIONAL TERM */

    whl_pos_pid_p = wheel_position_error * gain_p;

    /** CALCULATE THE INTEGRAL TERM */
    /* (1) Need to design how to calculate the integral gain, for now it is a constant */
    /* (2) Need to design when to reset the integrator */
    /* (3) Need to design if and how much to clip the integrator */

    if ((gain_i == 0) || (fabs(wheel_position_error) < ERROR_NEAR_ZERO)) /* if the error is close enough to zero */
        whl_pos_pid_i = 0; /* reset the integrator */
    else
        whl_pos_pid_i = whl_pos_pid_i + (wheel_position_error * delta_time * gain_i);

    if (whl_pos_pid_i <= pid_i_min) /* clip the integrator */
        whl_pos_pid_i = pid_i_min;
    if (whl_pos_pid_i >= pid_i_max)
        whl_pos_pid_i = pid_i_max;

    /** CALCULATE THE DERIVATIVE TERM */
    /* (1) Need to design how to calculate the derivative gain, for now it is a constant */
    /* (2) Need to design if and how to implement a rate limiter */
    /* (3) Need to design if and how to implement a noise-reject hysteresis */

    error_rate = (wheel_position_error - wheel_position_error_last) / delta_time;
    whl_pos_pid_d = error_rate * gain_d; /* the derivative term is the rate of change of error times the gain */

    /** CALCULATE THE CONTROL PARAMETER */

    wheel_position_pid = whl_pos_pid_p + whl_pos_pid_i + whl_pos_pid_d;

    if (wheel_position_pid > /* clip the final control output */
        wheel_position_pid = WHL_POS_CMD_MAX;
    else if (wheel_position_pid < (-1))
        wheel_position_pid = WHL_POS_CMD_MIN;

```

```

/** OUTPUT PROCESSING */

/* Take the commanded wheel position from the PID */
/* controller and de-normalize it for output. */
/* Let's assume the position controller uses PWM and it operates this way: */
/* (1) 50% duty cycle maintains current wheel position */
/* (2) the more tending to 0%, the more force in the "left" direction */
/* (3) the more tending to 100%, the more force in the "right" direction */

wheel_position_dc = (wheel_position_pid + 1) / 2 * 100;

/* If necessary, filter the signals as well. */

} /* end of control_steering() */

/** ADDITIONAL SUB-FUNCTIONS */

/*-----*\
low_pass_filter()
inputs: unfiltered input value, low-pass cutoff frequency,
        pointer to the input from last time, and the output from last time
outputs: filtered version of the input
\*-----*/
float low_pass_filter( const float current_input, const float cutoff_freq, * const last_input, const float last_output)
{
    float current_output;

    current_output = ((current_input + *last_input) * (0.5F) * cutoff_frequency) + ((1.0F - cutoff_frequency) * (last_output));
    *last_input = current_input;
    return current_output;
}

```

```

/*-----*\
  get_joystick_position()
  inputs: none
  outputs: voltage from sensor
\*-----*/
float get_joystick_position()
{
  return 2.5; /* replace this with real I/O instructions */
}

/*-----*\
  get_wheel_position()
  inputs: none
  outputs: voltage from sensor
\*-----*/
float get_wheel_position()
{
  return 2.5; /* replace this with real I/O instructions */
}

/*-----*\
  get_delta_time()
  inputs: none
  outputs: time in seconds since last execution
\*-----*/
float get_delta_time()
{
  return 0.010; /* Assume for now 10 msec execution rate. Replace this later with real kernel instructions */
}

```

PID Control Summary

- For all its advantages and its many disadvantages, the full PID controller has been around since the mid-1950's and remains predominant today.
- It works well enough for most control applications (with or w/o the Derivative term),
- It is relatively easy to implement
- Its basic operating principles are easy to understand.
- For team 301, simple proportional control seemed adequate for what we used it for and for the results we got.

Advanced Programming

PID Control



Ford FIRST Robotics Presentation
by Chris Teslak, Team Ford FIRST