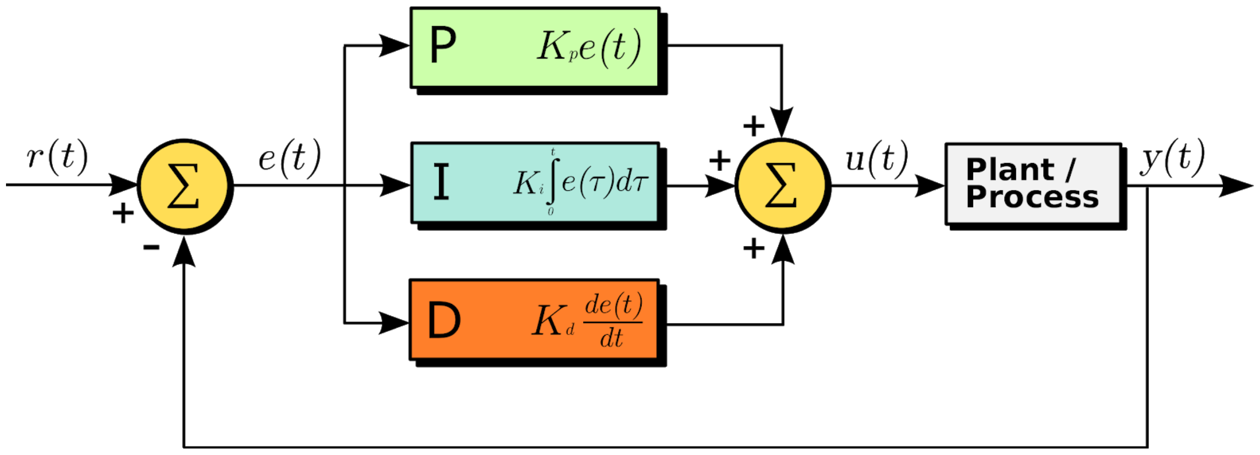
Library used: AutoPID (Ryan Downing, 2017, documentation, github)



From Wikimedia Commons

Functions used:

```
AutoPID::AutoPID(double *input, double *setpoint, double *output, double outputMin,
double outputMax,
                double Kp, double Ki, double Kd) {
  _input = input;
  _setpoint = setpoint;
  _output = output;
  _outputMin = outputMin;
  _outputMax = outputMax;
  setGains(Kp, Ki, Kd);
  _timeStep = 1000;
}
```

Constructor for the AutoPID object. Input is the value sent into the PID loop to compare to the setpoint value, equivalent to y(t). Setpoint is the value used to indicate the required condition of the controlled variable, equivalent to r(t). Output is the calculated sum of the proportional, integral and derivative parts of the virtual PID controller, equivalent to u(t). OutputMin is the minimum acceptable output value for the PID. OutputMax is the maximum acceptable one. Kp is the proportional component constant. Ki is the integral component constant. Kd is the derivative component constant. Constants are shown in the equations for each part of the PID controller. Assignment statements set the given values for each variable to near-identical internal variables for use in internal object calculations. The last assignment sets a default value for *_timeStep*, an internal variable indicating the time period after which to calculate PID output.

```
void AutoPID::setGains(double Kp, double Ki, double Kd) {
  _Kp = Kp;
  _Ki = Ki;
  _Kd = Kd;
}
```

Assigns the defined values of each PID constant to internal variables. Can be called outside the constructor.

```
void AutoPID::setTimeStep(unsigned long timeStep){
  _timeStep = timeStep;
}
```

Called in *setup()* in the Arduino sketch. Sets the time interval variable for the PID loop.

```
void AutoPID::run() {
  if (_stopped) {
    _stopped = false;
    reset();
  }
  //if bang thresholds are defined and we're outside of them, use bang-bang control
  if (_bangOn && ((*_setpoint - *_input) > _bangOn)) {
    *_output = _outputMax;
    _lastStep = millis();
  } else if (_bangOff && ((*_input - *_setpoint) > _bangOff)) {
    *_output = _outputMin;
    _lastStep = millis();
  } else {                                    //otherwise use PID control
    unsigned long _dT = millis() - _lastStep;   //calculate time since last update
    if (_dT >= _timeStep) {                   //if long enough, do PID calculations
      _lastStep = millis();
      double _error = *_setpoint - *_input;
      _integral += (_error + _previousError) / 2 * _dT / 1000.0;   //Riemann sum integral
      //_integral = constrain(_integral, _outputMin/_Ki, _outputMax/_Ki);
      double _dError = (_error - _previousError) / _dT / 1000.0;   //derivative
      _previousError = _error;
      double PID = (_Kp * _error) + (_Ki * _integral) + (_Kd * _dError);
      //*_output = _outputMin + (constrain(PID, 0, 1) * (_outputMax - _outputMin));
      *_output = constrain(PID, _outputMin, _outputMax);
    }
  }
}
```

This function is called with each iteration of *loop()* in the arduino sketch. The first section checks to see if the PID object has been stopped by an unused stop function, and terminates *run()* if so.

Reset(), also unused, resets PID calculations to their zero state. Bang thresholds, a feature of this PID library, are maximum and minimum error values at which the PID sets its output to *OutputMax* or *OutputMin*. They are unused. The second *if* statement checks if the bang thresholds have been met and sets output accordingly. The *else* statement contains standard PID operating code. *unsigned long _dT = millis() - _lastStep* is used to check a condition that terminates operation of the PID if not enough time has passed between PID calculations. This function does its own calculation of the error term e(t) = r(t) - y(t) through assignment of the difference of those variables. The integral and derivative parts of the PID, having to operate with a discrete-time dataset rather than a function, make a crude riemann-sum calculation of each part according to the time interval defined and the error value recorded by assignment in the line *_previousError = _error*, which is used to make a comparison of error change over time. *_integral += (_error + _previousError) / 2 * _dT / 1000.0* integrates change over time by taking the mean average of error change in one time step and multiplying that by the time step's fraction of one second, which is in accordance with the understanding of an integral as the area of subdivisions of a function of time over the passage of time. *double _dError = (_error - _previousError) / _dT / 1000.0* performs the opposite of the integral operation, as the derivative is an opposite process to the integral. Deriving the value of the integration in this function would return the mean average of error change, meaning it undoes the integration. *double PID = (_Kp * _error) + (_Ki * _integral) + (_Kd * _dError)* applies the PID constants to the calculated values for each component of the loop and sums the results to get the PID output, assigning it to an intermediate variable for the last operation. **_output = constrain(PID, _outputMin, _outputMax)* assigns the calculated output to the output pointer using *constrain(x, a, b)*, which is an arduino function that keeps the value *x* from going below *a* or above *b*, in this case the defined maximum and minimum output values for the PID loop.