```cpp
// pid.cpp: Source file for pid and all of it's assets
// Copyright (C) 2017 Ethan Wells
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.

#include "../include/pid.hpp"

int sgn(float __x);

namespace pid {
  float Kp              = 0.8;
  float Ki              = 0.04;
  float Kd              = 0.35;
  unsigned int deadband = 10;

  bool enabled[2]                = {true, true};
  unsigned int default_precision = 30;
  TaskHandle pidHandle;

  void controller(void* none) {
    float current[2];
    float error[2];
    float lastError[2] = {0, 0};
    float integral[2]  = {0, 0};
    float derivative[2];
    float power[2];

    sensors::left.reset();
    sensors::right.reset();
    sensors::quad_t* sides[2] = {&sensors::left, &sensors::right};

    while (true) {
      printf("| %ld | %ld |\n", sensors::left.value(), sensors::right.value());
      for (size_t i = 0; i < 2; i++) {
        if (enabled[i]) {
```

```
        current[i] = sides[i]->value();
        error[i]   = sides[i]->request - current[i];
        if ((unsigned int)abs((int)error[i]) <= deadband) {
          continue;
        }
        integral[i] = (Ki != 0 && abs((int)error[i]) < INTEGRAL_LIMIT)
                        ? (integral[i] + error[i])
                        : 0;
        derivative[i] = error[i] - lastError[i];
        lastError[i]  = error[i];
        power[i] =
            (Kp * error[i]) + (Ki * integral[i]) + (Kd * derivative[i]);
        power[i] = (power[i] <= DRIVE_MIN)
                       ? DRIVE_MIN
                       : ((power[i] >= DRIVE_MAX) ? DRIVE_MAX : power[i]);
        (i == 0) ? drive::left.set(power[i]) : drive::right.set(power[i]);
      }
    }
    delay(25);
  }
  free(none);
}

void enable(void) {
  enabled[0] = true;
  enabled[1] = true;
}

void disable(void) {
  enabled[0] = false;
  enabled[1] = false;
}

void init(void) {
  pidHandle = taskCreate(controller, TASK_DEFAULT_STACK_SIZE, NULL,
                         TASK_PRIORITY_DEFAULT);
}

void stop(void) { taskSuspend(pidHandle); }

void go(void) { taskResume(pidHandle); }

void request(long l, long r) {
  sensors::left.request  = l;
  sensors::right.request = r;
}
```

```
  void wait(unsigned long precision, unsigned long blockTime) {
    if (blockTime > 0) {
      auto start = millis();
      while ((sensors::left.value() > sensors::left.request + precision ||
              sensors::left.value() < sensors::left.request - precision ||
              sensors::right.value() > sensors::right.request + precision ||
              sensors::right.value() < sensors::right.request - precision) &&
             millis() - start <= blockTime) {
        delay(50);
      }
    } else {
      while ((sensors::left.value() > sensors::left.request + precision ||
              sensors::left.value() < sensors::left.request - precision ||
              sensors::right.value() > sensors::right.request + precision ||
              sensors::right.value() < sensors::right.request - precision)) {
        delay(50);
      }
    }
  }
}

int sgn(float __x) {
  if (__x > 0)
    return 1;
  if (__x < 0)
    return -1;
  return 0;
} // namespace pid
```