```cpp
// drive.cpp: Source file for utilities relating to the drive
// 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/main.h"

namespace drive {
  side_t left;
  side_t right;
  double inch =
      28.647889756541160438399077407052585166202736233282160774580127 35;

  void side_t::set(int power) {
    side_t::topM.set(power);
    side_t::midM.set(power);
    side_t::lowM.set(power);
  }

  void init(void) {
    left.topM    = motors::init(2, 1, .5, .8);
    left.midM    = motors::init(3, -1, .5, .8);
    left.lowM    = motors::init(4, 1, .5, .8);
    right.topM   = motors::init(7, -1, .5, .8);
    right.midM   = motors::init(8, 1, .5, .8);
    right.lowM   = motors::init(9, -1, .5, .8);
    left.sensor  = &sensors::left;
    right.sensor = &sensors::right;
  }

  void set(int lpower, int rpower) {
    left.set(lpower);
    right.set(rpower);
  }
```

```
void tank(void) {
  int deadband = 20;
  int lj       = joystickGetAnalog(1, 3);
  int rj       = joystickGetAnalog(1, 2);
  if (abs(lj) < deadband && abs(rj) < deadband) {
    pid::enable();
    return;
  }
  lj              = (abs(lj) < deadband) ? 0 : lj;
  rj              = (abs(rj) < deadband) ? 0 : rj;
  pid::enabled[0] = (lj == 0);
  pid::enabled[1] = (rj == 0);
  if (lj != 0)
    left.set(lj);
  if (rj != 0)
    right.set(rj);
  pid::request((lj == 0) ? left.sensor->request : left.sensor->value(),
               (rj == 0) ? right.sensor->request : right.sensor->value());
}

namespace accel {
  int deadband = 20;
  int x        = 0;
  int y        = 0;
  int prevX    = 0;
  int prevY    = 0;
  void drive(void) {
    prevX             = x;
    prevY             = y;
    x                 = 0 - joystickGetAnalog(1, ACCEL_X);
    y                 = 0 - joystickGetAnalog(1, ACCEL_Y);
    int threshold     = 20;
    double multiplier = 1.1;

    if (abs(x) < threshold)
      x = 0;
    if (abs(y) < threshold)
      y = 0;

    x *= multiplier;
    y *= (multiplier * 1.25);

    int lj          = x - y;
    int rj          = x + y;
    lj              = (abs(lj) < deadband) ? 0 : lj;
    rj              = (abs(rj) < deadband) ? 0 : rj;
```

```
    pid::enabled[0] = (lj == 0);
    pid::enabled[1] = (rj == 0);
    if (lj != 0)
      left.set(lj);
    if (rj != 0)
      right.set(rj);
    pid::request((lj == 0) ? left.sensor->request : left.sensor->value(),
                 (rj == 0) ? right.sensor->request : right.sensor->value());
  }
} // namespace accel

void inches(long inches) {
  pid::enable();
  left.sensor->request += inches * inch;
  right.sensor->request += inches * inch;
  pid::wait(pid::default_precision, inches * inches * 8);
  pid::disable();
}

namespace gyro {
  void drive::task(void* none) {
    float change[2] = {0};
    while (on) {
      changer = (abs(gyro->value() - iHeading) - tolerance) * urgency;
      if (gyro->value() > iHeading + tolerance) {
        change[0] -= changer;
        change[1] += changer;
      } else if (gyro->value() < iHeading - tolerance) {
        change[0] += changer;
        change[1] -= changer;
      }
      pid::request(change[0], change[1]);
      delay(50);
    }
  }

  void drive::off(void) {
    on = false;
  }

  drive::drive(int heading, float urgency, bool absolute,
               sensors::gyro_t* gyro, unsigned int tolerance)
      : heading(heading),
        urgency(urgency),
        gyro(gyro),
        iHeading(absolute ? heading : heading + gyro->value()) {
```

```
    }
  } // namespace gyro
} // namespace drive
```