

```

/**
 * @file robot.c
 * @brief More general things related to the robot
 * Copyright (C) 2017 Ethan Wells
 *
 * This program is free software: you can redistribute it and/or modify it
 * under the terms of the GNU 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 General Public License for more
 * details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program. If not, see <https://www.gnu.org/licenses/>
 */

#include "../include/robot.h"

#define RED      "\x1b[31m"
#define GREEN    "\x1b[32m"
#define YELLOW   "\x1b[33m"
#define BLUE     "\x1b[34m"
#define MAGENTA  "\x1b[35m"
#define CYAN     "\x1b[36m"
#define RESET    "\x1b[0m"

// Sensors and motors
Sensor gyro, line[3];
Motor drive[2], lift, intake[2];

// PID settings
#define _INTAKE_SETTINGS_(index) \
    DEFAULT_PID_SETTINGS, \
    .kP      = -.7f, \
    .kI      = -.22f, \
    .kD      = -.08f, \
    .root    = &intake[index], \

PIDSettings intakeSettings[2] = {
    { _INTAKE_SETTINGS_(0) },
    { _INTAKE_SETTINGS_(1) },
};

```

```

#define _DRIVE_SETTINGS_(index) \
    DEFAULT_PID_SETTINGS, \
    .kP      = .151f, \
    .kI      = .043f, \
    .kD      = .253f, \
    .tolerance = 200, \
    .precision = 275, \
    .root     = &drive[index]

PIDSettings driveSettings[2] = {
    { _DRIVE_SETTINGS_(0) },
    { _DRIVE_SETTINGS_(1) },
};

#define _GYRO_SETTINGS_(index, m) \
    DEFAULT_PID_SETTINGS, \
    .kP      = m * 2.8625f, \
    .kI      = m * 0.5877f, \
    .kD      = m * 2.3363f, \
    .tolerance = 2, \
    .precision = 425, \
    .root     = &drive[index], \
    .sensor   = &gyro

PIDSettings gyroSettings[2] = {
    { _GYRO_SETTINGS_(0, 1) },
    { _GYRO_SETTINGS_(1, -1) },
};

void altRefresh(Sensor *s) {
    mutexTake(s->_mutex, 5);
    s->value = analogReadCalibrated(s->port);
    mutexGive(s->_mutex);
} /* altRefresh */

void init();

void reset() {
    /* free mutexes
    mutexGive(gyro._mutex);
    mutexGive(gyro.child->_mutex);

    mutexGive(lift._mutex);
    mutexGive(lift.child->_mutex);

    for (int i = 0; i < 2; i++) {

```

```

        mutexGive(drive[i]._mutex);
        mutexGive(intake[i]._mutex);

        mutexGive(drive[i].sensor->_mutex);
    }
    */

    // Reset sensors
    sensorReset(&gyro);
    sensorReset(drive[0].sensor);
    sensorReset(drive[1].sensor);

    // Reset PID times
    for (int i = 0; i < 2; i++) {
        intakeSettings[i]._time = millis();
        driveSettings[i]._time = millis();
        gyroSettings[i]._time = millis();
    }
} /* reset */

void update() {
    motorUpdate(&lift);

    sensorRefresh(&gyro);
    sensorRefresh(&line[2]);

    for (size_t i = 0; i < 2; i++) {
        motorUpdate(&drive[i]);
        motorUpdate(&intake[i]);
        sensorRefresh(&line[i]);
        sensorRefresh(drive[i].sensor);
        intake[i].sensor->value = encoderGet(intake[i].sensor->_pros);
    }
} /* update */

void info() {
    #ifndef DEBUG_MODE
        return;
    #endif

    static unsigned long time = 0;
    char *en = isEnabled() ? "\n" : "\r";

    if (millis() - time >= 20) {
        printf(
            RESET "\r"

```

\

```

        RED "%d, " GREEN "%d, " YELLOW "%d, " CYAN "%d, " \
        RED "%d, " GREEN "%d, " YELLOW "%d, " CYAN "%d, " \
        "%d, " RED "%d, " YELLOW "%d, %d, %d" BLUE " // %u mV"
        RESET "%s",
        drive[0].sensor->value,
        drive[1].sensor->value,
        intake[0].sensor->value,
        intake[1].sensor->value,
        drive[0].sensor->velocity,
        drive[1].sensor->velocity,
        intake[0].sensor->velocity,
        intake[1].sensor->velocity,
        0,
        gyro.averageVal,
        line[0].value,
        line[1].value,
        line[2].value,
        powerLevelMain(),
        en);
    lcdPrint(uart1, 2, "%u mV", powerLevelMain());
    time = millis();
}
} /* info */

bool takeTwo(unsigned long blockTime, Mutex mutex1, Mutex mutex2) {
    blockTime /= 2;

    if (!mutexTake(mutex1, blockTime)) {
        return false;
    } else if (!mutexTake(mutex2, blockTime)) {
        mutexGive(mutex2);
        return false;
    }
    return true;
} /* takeDrive */

void giveDrive() {
    mutexGive(drive[0]._mutex);
    mutexGive(drive[1]._mutex);
} /* giveDrive */

void driveSet(int l, int r) {
    if (!takeTwo(10, drive[0]._mutex, drive[1]._mutex)) {
        return;
    }
}

```

```

        drive[0].power = 1;
        drive[1].power = r;

        for (int i = 0; i < 2; i++) {
            mutexGive(drive[i]._mutex);
            motorUpdate(&drive[i]);
        }
    } /* driveSet */

void intakeSet(int p) {
    if (!takeTwo(10, intake[0]._mutex, intake[1]._mutex)) {
        return;
    }

    if (p) {
        intake[0].power = p;
        intake[1].power = p;
    } else {
        for (int i = 0; i < 2; i++) {
            intakeSettings[i].target = intake[i].sensor->value;
            PID(&intakeSettings[i]);
        }
    }

    for (int i = 0; i < 2; i++) {
        mutexGive(intake[i]._mutex);
        motorUpdate(&intake[i]);
    }
}

bool initialized = false;

void initialize() {
    // Call the init function to perform actions in init.c
    if (!initialized) {
        init();
    }
    reset();

    // Wait for initialization to end
    while (!isAutonomous() && !isEnabled()) {
        delay(15);
    }
} /* initialize */

#define stallVel 10 / 100

```

```

bool waitForDriveStall(unsigned long blockTime) {
    unsigned long stop = millis() + blockTime;

    int sV[2] = { drive[0].sensor->value, drive[1].sensor->value };
    int dV[2] = { 100, 100 };
    int p[2] = { drive[0].power, drive[1].power };

    unsigned long sT[2] = { millis(), millis() };
    unsigned long dT[2] = { 1, 1 };

    do {
        delay(10);

        for (int i = 0; i < 2; i++) {
            sensorRefresh(drive[i].sensor);
            dV[i] = abs(drive[i].sensor->value - sV[i]);
            dT[i] = millis() - sT[i];

            if (dV[i] / dT[i] > stallVel) {
                sV[i] = drive[i].sensor->value;
                sT[i] = millis();
            } else {
                p[i] = 0;
            }
        }

        driveSet(p[0], p[1]);

        if (millis() > stop) {
            return false;
        }
    } while (p[0] != 0 || p[1] != 0);

    return true;
} /* waitForDriveStall */

void resetDrive() {
    sensorReset(drive[0].sensor);
    sensorReset(drive[1].sensor);
} /* resetDrive */

```