```c
/**
 * @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"

double inch =
  (1 / (M_PI * (DRIVE_WHEEL_DIAMETER / 360) * (1 / DRIVE_ENCODER_RATIO)));

// Sensors
Sensor gyro, *sonic, armLimit[2], line[3];

// Motors and servos
Motor claw, arm, mogo, drive[2];

// PID settings
PIDSettings armSettings = {
        DEFAULT_PID_SETTINGS,
        .kP     = -.7f,
        .kI     = -.22f,
        .kD     = -.08f,
        .root   = &arm,
        .target = 10,
```

```
};

PIDSettings clawSettings = {
        DEFAULT_PID_SETTINGS,
        .kP        = .22f,
        .kI        = .0f,
        .kD        = 2.3f,
        .root      = &claw,
        .tolerance = 35,
        .precision = 175,
};

#define _DRIVE_SETTINGS_(index) \
  DEFAULT_PID_SETTINGS,         \
  .kP        = .170f,           \
  .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(arm.sensor->_mutex);

	  mutexGive(claw._mutex);
	  mutexGive(arm._mutex);
	  mutexGive(arm.child->_mutex);
	  mutexGive(mogo.child->_mutex);

	for (int i = 0; i < 2; i++) {
		mutexGive(drive[i]._mutex);

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

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

	// Reset PID times
	armSettings._time      = millis();
	driveSettings[0]._time = millis();
	driveSettings[1]._time = millis();
} /* reset */

void update() {
	motorUpdate(&claw);
	motorUpdate(&mogo);
	motorUpdate(&arm);

	sensorRefresh(arm.sensor);
	sensorRefresh(claw.sensor);
	sensorRefresh(mogo.sensor);

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

	for (size_t i = 0; i < 2; i++) {
```

```
			motorUpdate(&drive[i]);
			sensorRefresh(drive[i].sensor);
			sensorRefresh(&armLimit[i]);
			sensorRefresh(&line[i]);
		}
} /* 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, " BLUE "%d, " CYAN \
		  RED "%d, " GREEN "%d, " YELLOW "%d, " BLUE "%d, " CYAN \
		  "%d, " RED "%d, " GREEN "%d, " YELLOW "%d, %d, %d" BLUE " // %u mv"
		  RESET "%s",
		  drive[0].sensor->value,
		  drive[1].sensor->value,
		  arm.sensor->value,
		  claw.sensor->value,
		  drive[0].sensor->velocity,
		  drive[1].sensor->velocity,
		  arm.sensor->velocity,
		  claw.sensor->velocity,
		  mogo.sensor->averageVal,
		  gyro.averageVal,
		  sonic->value,
		  line[0].value,
		  line[1].value,
		  line[2].value,
		  powerLevelMain(),
		  en);
		lcdPrint(uart1, 2, "%u mV", powerLevelMain());
		time = millis();
	}
} /* info */

bool takeDrive(unsigned long blockTime) {
	blockTime /= 2;
```

```c
	if (!mutexTake(drive[0]._mutex, blockTime)) {
		return false;
	} else if (!mutexTake(drive[1]._mutex, blockTime)) {
		mutexGive(drive[0]._mutex);
		return false;
	}
	return true;
} /* takeDrive */

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

void driveSet(int l, int r) {
	if (!takeDrive(10)) {
		return;
	}

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

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

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 */
```