

```

/**
 * @file auto.c
 * @brief The primary source for the autonomous operation period
 * 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/auto.h"

bool isAuto = true;

void autonLeft12();
void autonLeft22();
void autonRight12();
void autonRight22();
void autonSkills();
void autonTest();
void autonStack();

void autonNone() {}

void testMotors();

void autonRehuh();

int    selectedAuton    = 6;
Auton autons[MAX_AUTON + 1] =
{ {
    // index 0
    .name      = "left 12",
    .sensorName = "lift",
    .sensor     = &lift.sensor,
    .execute    = &autonLeft12,
}, {

```

```

        // index 1
        .name      = "left 22",
        .sensorName = "gyr2",
        .sensor     = &gyro.child,
        .execute    = &autonLeft22,
    }, {

        // index 2
        .name      = "right 12",
        .sensorName = "intake",
        .sensor     = &intake.sensor,
        .execute    = &autonRight12,
    }, {

        // index 3
        .name      = "right 22",
        .sensorName = "mgo",
        .sensor     = &mgo.sensor,
        .execute    = &autonRight22,
    }, {

        // index 4
        .name      = "skills",
        .sensorName = "snc",
        .sensor     = &sonic,
        .execute    = &autonSkills,
    }, {

        // index 5
        .name      = "none",
        .sensorName = "lift",
        .sensor     = &lift.sensor,
        .execute    = &autonNone,
    }, {

        // index 6
        .name      = "test",
        .sensorName = "lef",
        .sensor     = &drive[0].sensor,
        .execute    = &autonTest,
    }, {

        // index 7
        .name      = "test motors",
        .sensorName = "rit",
        .sensor     = &drive[1].sensor,
        .execute    = &testMotors,
    }, {

        // index 8
        .name      = "drive and autostack",
        .sensorName = "lift",
        .sensor     = &lift.sensor,

```

```

        .execute      = &autonStack,
    }, };

void liftToPosition(float pos, unsigned long until) {
    liftSettings.target = pos;
    until              += millis();

    do {
        PID(&liftSettings);
        motorUpdate(&lift);
        sensorRefresh(lift.sensor);
        delay(10);
    } while (!liftSettings.isTargetReached && millis() < until);
} /* liftToPosition */

void driveToPosition(int l, int r, unsigned long until) {
    driveSettings[0].target = l;
    driveSettings[1].target = r;
    until                  += millis();

    do {
        PID(&driveSettings[0]);
        PID(&driveSettings[1]);

        motorUpdate(&lift);
        sensorRefresh(lift.sensor);
        sensorRefresh(&gyro);

        for (int i = 0; i < 2; i++) {
            motorUpdate(&drive[i]);
            sensorRefresh(drive[i].sensor);
        }

        delay(10);
    } while ((!driveSettings[0].isTargetReached ||
        !driveSettings[1].isTargetReached) &&
        millis() < until);

    drive[0].power = 0;
    drive[1].power = 0;
    update();
} /* driveToPosition */

void driveToPositionAngle(int l, int r, int a, unsigned long until) {
    int gError;
    driveSettings[0].target = l;

```

```

driveSettings[1].target = r;
until                    += millis();

do {
    gError = (a - gyro.averageVal) * 1.5;
    PID(&driveSettings[0]);
    PID(&driveSettings[1]);
    drive[0].power += gError;
    drive[1].power -= gError;

    motorUpdate(&lift);
    sensorRefresh(lift.sensor);
    sensorRefresh(&gyro);

    for (int i = 0; i < 2; i++) {
        motorUpdate(&drive[i]);
        sensorRefresh(drive[i].sensor);
    }

    delay(10);
} while ((!driveSettings[0].isTargetReached ||
        !driveSettings[1].isTargetReached) &&
        millis() < until);

drive[0].power = 0;
drive[1].power = 0;
update();
} /* driveToPosition */

void mogoP(int p) {
    mutexGive(mogo._mutex);
    mutexGive(mogo.child->_mutex);
    mutexGive(mogo.sensor->_mutex);
    mutexGive(mogo.sensor->child->_mutex);
    int pow;

    do {
        pow = sgn(p - mogo.sensor->averageVal);
        sensorRefresh(mogo.sensor);
        mogo.power = (pow ? pow : 1) * 127;
        motorUpdate(&mogo);
        delay(10);
    } while ((isAutonomous() || !isAuto) && abs(p - mogo.sensor->averageVal) > 78);

    mogo.power = 0;
    motorUpdate(&mogo);

```

```

} /* mogoP */

Task mogoPT(void *p) {
    mogoP((int)p);
} /* mogoPT */

float gyroPIDC[3] = {
    5.279,
    0.0,
    1.953,
};

void gyroPID(int target, int precision) {
    int error = 0;
    int integral = 0;
    int derivative = 0;

    do {
        derivative = (target - gyro.averageVal) - error;
        error = target - gyro.averageVal;
        integral += error / 10;

        drive[0].power = -((error * gyroPIDC[0]) +
            (integral * gyroPIDC[1]) +
            (derivative * gyroPIDC[2]));
        drive[1].power = ((error * gyroPIDC[0]) +
            (integral * gyroPIDC[1]) +
            (derivative * gyroPIDC[2]));

        if (lcdReadButtons(uart1)) {
            gyroPIDC[0] += 0.005;
            delay(100);
        }
        info();
#ifdef DEBUG_MODE
        printf("%f\n", gyroPIDC[0]);
#endif
        lcdPrint(uart1, 1, "%f", ((float)(powerLevelMain())) / 1000.0);
        lcdPrint(uart1, 2, "%f", gyroPIDC[0]);
        update();
        delay(10);
    } while (true || abs(error) > precision ||
        integral > 10);

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

```

```

} /* gyroPID */

void turnTo(int angle, unsigned long until) {
    until += millis();

    gyroSettings[0].target = angle;
    gyroSettings[1].target = angle;

    do {
        PID(&gyroSettings[0]);
        PID(&gyroSettings[1]);

        update();
        delay(10);
    } while ((!gyroSettings[0].isTargetReached ||
               !gyroSettings[1].isTargetReached) &&
              millis() < until);

    driveSet(0, 0);
} /* turnTo */

void getMogo() {
    intake.power          = -25;
    liftSettings.target = ARM_QUARTER + 20;
    lift.power = -70;
    motorUpdate(&lift);
    delay(300);
    driveSet(25, 25);

    TaskHandle liftHandle = GO(liftPID, NULL);
    TaskHandle mogoHandle = GO(mogoPT, MOGO_DOWN + 10);
    delay(600);

    driveToPosition(2075, 2075, 2300);
    driveSet(15, 15);

    while (taskGetState(mogoHandle))
        delay(10);

    mogoHandle = GO(mogoPT, MOGO_UP);

    driveSet(127, 127);
    delay(260);

    driveSet(18, 18);
}

```

```

    while (mogo.sensor->averageVal > MOGO_MID) {
        sensorRefresh(mogo.sensor);
        delay(10);
    }

    if (taskGetState(liftHandle))
        taskDelete(liftHandle);
    driveSet(0, 0);
    mogo.power = 6;
    motorUpdate(&mogo);
} /* getMogo */

Task backUp(void *time) {
    unsigned long t = (unsigned long)time;

    while (millis() - t < 14250) {
        delay(10);
    }

    t = millis();

    while (isAutonomous() && millis() - t < 500) {
        driveSet(-127, -127);
        update();
        delay(10);
    }
} /* backUp */

void placeCone() {
    // Arm down
    liftToPosition(ARM_DOWN + 35, 400);

    // Drop cone
    intake.power = 127; // Open intake
    motorUpdate(&intake);
    delay(475); // Give intake time to open
    intake.power = 0; // Stop intake
    liftToPosition(ARM_QUARTER, 400);
    lift.power = 10; // Keep lift up
    motorUpdate(&lift);
#ifdef DEBUG_MODE
    print("Cone placed!\n"); // Notify computer of cone state
#endif
    delay(150);
} /* placeCone */

```

```

Task placeConeT(void *none) {
    placeCone();
} /* placeCone */

TaskHandle dropMogo20(TaskHandle mogoHandle) {
    int p[2] = { drive[0].sensor->value, drive[1].sensor->value };

    // Start the mogo intake down
    if (!mogoHandle && taskGetState(mogoHandle))
        mogoHandle = GO(mogoPT, MOGO_MID + 125);
    driveToPosition(p[0] + 250, p[1] + 250, 600);
    driveSet(30, 30);

    // Wait until the mogo intake is up
    while (taskGetState(mogoHandle)) {
        delay(10);
    }

    // Wait a bit for the mobile goal to settle
    mogo.power = 127;
    motorUpdate(&mogo);
    driveSet(-64, -64);
    delay(315);
    mogo.power = 30;

    driveToPosition(p[0] - 400, p[1] - 400, 3000);
    return GO(mogoPT, MOGO_UP);
} /* dropMogo */

void autonomous() {
    unsigned long startTime = millis();
    isAuto = true;
    reset();
    sensorReset(drive[0].sensor);
    sensorReset(drive[1].sensor);
    sensorReset(lift.sensor);
    sensorReset(mogo.sensor);
    sensorReset(&gyro);

    selectedAuton = clipNum(selectedAuton, MAX_AUTON, 0);
    if (autons[selectedAuton].execute != NULL)
        autons[selectedAuton].execute();

    liftSettings.target = lift.sensor->averageVal; // Set target to current pos

#ifdef DEBUG_MODE

```



```

        printf("\n\n\rFinished autonomous in %ldms\n\n", millis() - startTime);
    #endif
    while (isAutonomous()) {
        PID(&liftSettings);
        update();
        delay(10);
    }
} /* autonomous */

void testMotors() {
    while (isAutonomous()) {
        for (int i = 1; i <= 10; i++) {
            motorSet(i, 127);
            delay(250);
            motorStopAll();
        }
    }
} /* testMotors */

Task driveToPositionAngleT(void *triple) {
    Triple *t = (Triple *)triple;

    driveSettings[0].target = t->a;
    driveSettings[1].target = t->b;

    do {
        PID(&driveSettings[0]);
        PID(&driveSettings[1]);
        drive[0].power += (t->c - gyro.averageVal) * 2.3;
        drive[1].power -= (t->c - gyro.averageVal) * 2.3;

        motorUpdate(&lift);
        sensorRefresh(lift.sensor);
        sensorRefresh(&gyro);

        for (int i = 0; i < 2; i++) {
            motorUpdate(&drive[i]);
            sensorRefresh(drive[i].sensor);
        }

        delay(10);
    } while ((!driveSettings[0].isTargetReached ||
        !driveSettings[1].isTargetReached));

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

```

```

        delete(triple);
        taskDelete(NULL);
    } /* driveToPositionAngleT */

Task liftToPositionT(void *pos) {
    int p = (int)pos;

    liftToPosition(p, p * 1.34);
    taskDelete(NULL);
} /* liftToPositionT */

Task intakeToPositionT(void *pos) {
    manipSettings.target = (int)pos;

    do {
        PID(&manipSettings);
        motorUpdate(&intake);
        sensorRefresh(intake.sensor);
        delay(10);
    } while (!manipSettings.isTargetReached);

    taskDelete(NULL);
} /* intakeToPositionT */

Task die(void *none) {
    taskDelete(NULL);
}

void moveTo(int leftV, int rightV, int liftV, int mogoV, int intakeV, int gyroV) {
    TaskHandle driveHandle, liftHandle, mogoHandle, intakeHandle;
    update();

    if ((abs(leftV - drive[0].sensor->value) > driveSettings[0].tolerance) ||
        (abs(rightV - drive[1].sensor->value) > driveSettings[1].tolerance) ||
        (abs(gyroV - gyro.averageVal) > gyroSettings[0].tolerance)) {
        Triple *t = new(Triple);
        t->a      = leftV;
        t->b      = rightV;
        t->c      = gyroV;
        driveHandle = GO(driveToPositionAngleT, t);
    }

    if (abs(liftV - lift.sensor->value) > liftSettings.tolerance)
        liftHandle = GO(liftToPositionT, liftV);

    if (abs(mogoV - mogo.sensor->averageVal) > 90)

```

```

        mogoHandle = GO(mogoPT, mogoV);

    if (abs(intakeV - intake.sensor->value) > manipSettings.tolerance)
        intakeHandle = GO(intakeToPositionT, intakeV);

    if (driveHandle)
        while (taskGetState(driveHandle))
            delay(10);

    if (liftHandle)
        while (taskGetState(liftHandle))
            delay(10);

    if (mogoHandle)
        while (taskGetState(mogoHandle))
            delay(10);

    if (intakeHandle)
        while (taskGetState(intakeHandle))
            delay(10);
} /* moveTo */

void liftPID(void *none) {
    while (isEnabled()) {
        PID(&liftSettings);
        delay(10);
    }
}

```