File: C:\Users\TheAv\Desktop\Robot Code\3946X-2018-19\functions.c

```
//This file contains the main functions used in driver and programming/autonomom


//runs the left side of the drive at a given speed
void runLeftDrive(int speed) {
    motor[backLeftDrive] = speed;
    motor[frontLeftDrive] = speed;
}
//runs the right side of the drive at a given speed
void runRightDrive(int speed) {
    motor[backRightDrive] = speed;
    motor[frontRightDrive] = speed;
}

//runs the drive at a given speed
void drive(int speed) {
    runLeftDrive(speed);
    runRightDrive(speed);
}

// Gets average of both drive quads, useful when driving straight
int driveQuadAvg() {
    return SensorValue[rightDriveQuad]/2+SensorValue[leftDriveQuad]/2;
}
// Stes drive quads to specied value
void setDriveQuads(int n) {
    SensorValue[rightDriveQuad]=n;
    SensorValue[leftDriveQuad]=n;
}

//lift at a specified speed
void lift(int speed) {
    motor[topLift]=speed;
}


//contains all the abstract variables for PID
typedef struct {
    float pGain;
    float iGain;
    float dGain;
    float constant;

    float iMin;
    float iMax;
    float iState;

    float position;
    float target;

    int timeDuration;
    int lastRan;
    float prevError;
    int currSpeed;
} PIDStruct;

//http://robotsforroboticists.com/pid-control/
int getPIDSpeed(PIDStruct PIDData) {
    //Only run if specified duration has elapsed
```

```
    if(nPgmTime-PIDData.lastRan>PIDData.timeDuration) {

        float timeInterval=(nPgmTime+0.001)-PIDData.lastRan;
        PIDData.lastRan=nPgmTime;
        float error=PIDData.target-PIDData.position;

        //Run proportional control
        int pTerm=PIDData.pGain*error;

        //Run integral control

        PIDData.iState+=error;
        int iTerm=PIDData.iGain*PIDData.iState;
        //if(sgn(PIDData.iState)!=sgn(error))PIDData.iState=0;
        if(PIDData.iState>PIDData.iMax)PIDData.iState=PIDData.iMax;
        if(PIDData.iState<PIDData.iMin)PIDData.iState=PIDData.iMin;

        //Run derivative control
        int dTerm=PIDData.dGain*(error-PIDData.prevError)/(timeInterval+0.001);

        //Update variables for next run
        PIDData.prevError=error;

        PIDData.currSpeed=pTerm+iTerm+dTerm+PIDData.constant;
    }
    return PIDData.currSpeed;
}

PIDStruct rotatorPID;
int rotatorLowPos=550;
int rotatorHighPos=3730;


task rotatorPIDTask{
    while(1) {
        if(vexRT[btn8R]) {
            motor[rotator]=127;
            //wait1Msec(1100);
        } else if(vexRT[Btn8L]) {
            motor[rotator]=-127;
            //wait1Msec(1100);
        } else {
            motor[rotator]=0;
        }
    }

}

PIDStruct liftPID
task liftControl{

// Sets lift PID variables

    liftPID.target=SensorValue[rightLift];
    while(1) {
        bool runPID=True;
        // Controls height of lift from button presses
        if(vexRT[Btn8U]) {
            if(SensorValue[rightLift]<2025-200)liftPID.target=2025;
```

```
            else {
                liftPID.target=SensorValue[rightLift];
                lift(127);
                runPID=false;
            }
        } else if(vexRT[Btn7U]) {
            if(SensorValue[rightLift]<1525-200)liftPID.target=1525;
            else {
                liftPID.target=SensorValue[rightLift];
                lift(127);
                runPID=false;
            }
        } else if(vexRT[Btn6D] && liftPID.target>550) {
            liftPID.target-=1.5;
            //if(liftPID.target<640)liftPID.target=640;
        }
        if(runPID)lift(getPIDSpeed(liftPID));

// Updates lift positio in PID
        liftPID.position=SensorValue[rightLift];

    }
}


// Drive certain distace using PID
PIDStruct drivePID;
void pDrive(int distance) {
    setDriveQuads(0);
    SensorValue[leftDriveQuad]=0;
    SensorValue[rightDriveQuad]=0;

    drivePID.target=distance+driveQuadAvg();
    drivePID.position=driveQuadAvg();
    drive(getPIDSpeed(drivePID));

    int counter=0;

// Loop through drive PID
    while(counter<200) {
        if(abs(drivePID.target-drivePID.position)<80)counter++;
        else counter=0;
        drivePID.position=driveQuadAvg();
        drive(getPIDSpeed(drivePID));

        wait1Msec(1);
    }
// Motor brake
    drive(-sgn(distance)*10);
    wait1Msec(50);
    drive(0);

}

void calibrateGyro() {
    SensorType[gyro] = sensorNone;
    wait1Msec(1000);
    SensorType[gyro] = sensorGyro;
    wait1Msec(2000);
```

```
}

//scales gyro value so 900=90 degree turn
float gyroValue() {
    return  SensorValue[gyro]*1.32;
}

// Turn a specified distace using PID

PIDStruct gyroPID;
void pTurn(int degrees, bool resetValue=false, int kicker = 2000) {
    clearTimer(T2);
    if(resetValue)SensorValue[gyro]=0;

    gyroPID.target=degrees;
    gyroPID.position=gyroValue();
    drive(getPIDSpeed(gyroPID));

    int counter=0;
    // PID loop
    while(counter<100 && time1[t2] < kicker) { //loop until the robot has been i
        if(abs(gyroPID.target-gyroPID.position)<80)counter++;//add another milli
        else counter=0;
        gyroPID.position=gyroValue();
        int motorSpeed=getPidSpeed(gyroPID);
        runRightDrive(motorSpeed);
        runLeftDrive(-motorSpeed);
        wait1Msec(1);
    }
    SensorValue[gyro]=false;

}



// Run claw PID
PIDStruct clawPID;

void runClawPID(PIDStruct clawPID) {



    clawPID.position=SensorValue[clawPot];
    motor[claw]=-getPidSpeed(clawPID);
}


//LCD set up code
void waitForRelease()
{
    while(nLCDButtons != 0) {}
    wait1Msec(5);
}



// Task for rotator
task rotatorTask() {
```

```
    while(1) {
        rotatorPID.position=SensorValue[rotatorPot];
        motor[rotator]=-getPIDSpeed(rotatorPID);
    }
}
// Task for claw
bool clawIdle=false;

task clawTask() {
    while(1) {
        if(clawIdle)motor[claw]=0;
        else runClawPID(clawPID);
    }
}

bool lockDrive=false;
task driveLocker() {
    while(1) {
        if(lockDrive) {
            drivePID.position=driveQuadAvg();
            int speed=getPIDSpeed(drivePID)*1.5
                      if(abs(speed)>30)speed=30*sgn(speed);
            drive(speed);
        } else {
            setDriveQuads(0);
            drivePID.target=driveQuadAvg();
        }
    }
}
```