9042 Code Review

OpHelperClean

Code

```
package com.qualcomm.ftcrobotcontroller.opmodes;

import com.qualcomm.ftcrobotcontroller.BuildConfig;
import com.qualcomm.robotcore.eventloop.opmode.OpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorController;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.Range;

public class OpHelperClean extends OpMode {

    //driving motors
    DcMotor frontLeft,
            backLeft;

    DcMotor frontRight,
            backRight;

    //arm motors
    DcMotor armMotor1,
            armMotor2,
            armPivot;

    //zipline servo
    Servo zipLiner;

    //encoder targets
    private int rightTarget,
            leftTarget;

    //SERVO CONSTANTS
    private final double SERVO_MAX=1,
                         SERVO_MIN=0,
                         SERVO_NEUTRAL = 9.0/17;//Stops the continuous servo
```

```
//MOTOR RANGES
private final double MOTOR_MAX=1,
                     MOTOR_MIN=-1;

//ENCODER CONSTANTS TODO: Calibrate all of these values
private final double CIRCUMFERENCE_INCHES = 4*Math.PI,
        TICKS_PER_ROTATION = 1200/1.05,
        TICKS_PER_INCH = TICKS_PER_ROTATION/CIRCUMFERENCE_INCHES,
        TOLERANCE = 10;

public OpHelperClean(){

}

public void init(){
    //left drive
    frontLeft = hardwareMap.dcMotor.get("l1");
    backLeft = hardwareMap.dcMotor.get("l2");

    //right drive
    frontRight = hardwareMap.dcMotor.get("r1");
    backRight = hardwareMap.dcMotor.get("r2");

    //pivot motor
    armPivot = hardwareMap.dcMotor.get("arm");

    //tape measure arms
    armMotor1 = hardwareMap.dcMotor.get("tm1");
    armMotor2 = hardwareMap.dcMotor.get("tm2");

    //zipline servo
    zipLiner = hardwareMap.servo.get("zip");

    setDirection(); //ensures the proper motor directions


    resetEncoders(); //ensures that the encoders have reset
}

//sets the proper direction for the motors
public void setDirection(){
```

```
    //config drive motors
    if(frontLeft.getDirection() == DcMotor.Direction.REVERSE){
        frontLeft.setDirection(DcMotor.Direction.FORWARD);
    }
    if(backLeft.getDirection() == DcMotor.Direction.REVERSE){
        backLeft.setDirection(DcMotor.Direction.FORWARD);
    }

    if(frontRight.getDirection() == DcMotor.Direction.FORWARD){
        frontRight.setDirection(DcMotor.Direction.REVERSE);
    }

    if(backRight.getDirection() == DcMotor.Direction.FORWARD){
        backRight.setDirection(DcMotor.Direction.REVERSE);
    }


    //TODO configure arm motor direction

    //TODO config arm pivot direction
}

//moves tape measure based on direct
public void moveTapeMeasure(double power){
    armMotor2.setPower(power);
    armMotor1.setPower(power);
}

//reset all the drive encoders and return true if all encoders read 0
public boolean resetEncoders() {
    frontLeft.setChannelMode(DcMotorController.RunMode.RESET_ENCODERS);
    backLeft.setChannelMode(DcMotorController.RunMode.RESET_ENCODERS);

    frontRight.setChannelMode(DcMotorController.RunMode.RESET_ENCODERS);
    backRight.setChannelMode(DcMotorController.RunMode.RESET_ENCODERS);

    return (frontLeft.getCurrentPosition() == 0 &&
            backLeft.getCurrentPosition() == 0 &&
            frontRight.getCurrentPosition() == 0 &&
            backRight.getCurrentPosition() == 0);

}
```

```
//driving power
public void setMotorPower(double leftPower, double rightPower){
    clipValues(leftPower, ComponentType.MOTOR);
    clipValues(rightPower, ComponentType.MOTOR);

    frontLeft.setPower(leftPower);
    backLeft.setPower(leftPower);

    frontRight.setPower(rightPower);
    backRight.setPower(rightPower);
}

//sets all drive motors to encoder mode
public void setToEncoderMode(){

    frontLeft.setChannelMode(DcMotorController.RunMode.RUN_TO_POSITION);
    backLeft.setChannelMode(DcMotorController.RunMode.RUN_TO_POSITION);

    frontRight.setChannelMode(DcMotorController.RunMode.RUN_TO_POSITION);
    backRight.setChannelMode(DcMotorController.RunMode.RUN_TO_POSITION);
}

//sets all drive motors to run without encoders
public void setToWOEncoderMode()
{
    frontLeft.setChannelMode(DcMotorController.RunMode.RUN_WITHOUT_ENCODERS);
    backLeft.setChannelMode(DcMotorController.RunMode.RUN_WITHOUT_ENCODERS);

    frontRight.setChannelMode(DcMotorController.RunMode.RUN_WITHOUT_ENCODERS);
    backRight.setChannelMode(DcMotorController.RunMode.RUN_WITHOUT_ENCODERS);
}

//makes the robot move straight using the encoders
public boolean runStraight(double distance_in_inches) {
    leftTarget = (int)(distance_in_inches*TICKS_PER_INCH);
    rightTarget = leftTarget;
    setTargetValueMotor();

    setMotorPower(.4, .4);

    if(hasReached())
```

```
    {
        setMotorPower(0,0);
        return true;//done traveling
    }
    return false;
}

//sets the target position for the drive encoders
public void setTargetValueMotor(){
    frontLeft.setTargetPosition(leftTarget);
    backLeft.setTargetPosition(leftTarget);

    frontRight.setTargetPosition(rightTarget);
    backRight.setTargetPosition(rightTarget);
}

//returns true if all the motors have reached the desired postiion
public boolean hasReached()
{
    return (Math.abs(frontLeft.getCurrentPosition()-leftTarget)<=TOLERANCE &&
            Math.abs(backLeft.getCurrentPosition()-leftTarget)<=TOLERANCE &&
            Math.abs(frontRight.getCurrentPosition()-rightTarget)<=TOLERANCE &&
            Math.abs(backRight.getCurrentPosition()-rightTarget)<=TOLERANCE);
}

//TODO: Run tests to determine the relationship between degrees turned and ticks
public boolean setTargetValueTurn(double degrees){
    return false;
}

//basic debugging and feedback
public void basicTel(){
    //left drive
    telemetry.addData("frontLeftPos: ", frontLeft.getCurrentPosition());
    telemetry.addData("backLeftPos: ", backLeft.getCurrentPosition());
    telemetry.addData("LeftTarget: ", leftTarget);

    //right drive
    telemetry.addData("frontRightPos: ", frontRight.getCurrentPosition());
    telemetry.addData("backRightPos: ", backRight.getCurrentPosition());
    telemetry.addData("RightTarget: ", rightTarget);
}
```

```
enum ComponentType{          //helps with clipValues
    NONE,
    MOTOR,
    SERVO
}

//makes sure values are within the range for various components
public double clipValues(double initialValue, ComponentType type) {
    double finalval=0;
    if (type == ComponentType.MOTOR)
        finalval =  Range.clip(initialValue, MOTOR_MIN, MOTOR_MAX);
    if (type == ComponentType.SERVO)
        finalval= Range.clip(initialValue, SERVO_MIN, SERVO_MAX);
    return finalval;
}

//sets the postiion of the zipline
public boolean setZipLinePosition(double pos){//slider values
    if(pos == 1){
        zipLiner.setPosition(SERVO_MAX);
    } else if(pos == -1){
        zipLiner.setPosition(SERVO_MIN);
    } else if(pos == 0){
        zipLiner.setPosition(SERVO_NEUTRAL);
    }

    return true;
}

//moves the arm at a constant speed
//TODO: Calibrate this motor for the arm
public void setArmPivot(double power){
    armPivot.setPower(power);
}


//normal driving mode
//boolean is true when turtle drive should be enabled
public void manualDrive(boolean turtleDrive){
    setToWOEncoderMode();
```

```
        double rightPower = gamepad1.right_stick_y;
        double leftPower = gamepad1.left_stick_y;

        if(turtleDrive){
            setMotorPower(rightPower*.5, leftPower*.5);
        } else{
            setMotorPower(rightPower, leftPower);
        }
    }

    //TODO: Make a function to move drive at same speed as the tape measure (Eric's suggestion
    public void upMountain(){

    }

    public void loop(){

    }

    public void stop(){

        setMotorPower(0,0);//brake the movement of drive
        moveTapeMeasure(0);//brake the tape measure
        setArmPivot(0);//brake the arm pivot

    }

}
```

Design/Function

The OpMode Helper program is made to simplify other programs, like autonomous and teleop by creating functions to be used in those programs when that class extends it. It works by initializing and defining a collection of functions in the init function of the OpMode Helper

The program starts out by declaring the 4 drive motors, the tape measure arm pivoting motor, the 2 tape measure extending motors, the zipline trigger hitter rack and pinion continuous servo. It also defines the final numbers for the maximum and minimum servo speed, the stop value, the tolerance for encoders, and the mathematical calculations for the encoder counts to inches. Then, in the init, the declared motors and servos are found in the hardware map. Then, the

setDirection function is called, which basically states for each of the motors, that if they are in the wrong direction, then they are set to the direction which they should have been initialized to. After that the function resetEncoders is called, which resets the encoders for the 4 drive base motors and returns true if all encoder values are set to 0 for debugging purposes. Both of these functions are defined after the init, just like moveTapeMeasure, which sets both tape measure extending motors to an inputted power. There is an enum of 3 types of ComponentType, which were NONE, SERVO, and MOTOR. The clipValues function, used to cut off the range of powers for motors and servos, uses this to determine whether to clip the input value at either the MOTOR_MAX and MOTOR_MIN or SERVO_MAX, and SERVO_MIN. It then outputs the possibly clipped input value. The clipValues function is used in another function called setMotorPower, which with an input of a power for both left and right sides, clips the values and then sets the 4 drive motors to these clipped speeds. There is another function which just sets all the the drive motor encoders to the channel mode of RUN_TO_POSITION and another one which sets them all to RUN_WITHOUT_ENCODERS, called setToEncoderMode and setToWOEncoderMode, respectively, for when we need to switch between autonomous, which uses the drive motor encoders primarily, to teleop, which doesn't use encoders for the drive motors at this time. A function called runStraight takes an input of inches, and converts it to encoder counts using math and sets this value as both the leftTarget and rightTarget variables. Then it uses another function called setTargetValueMotor, which sets these target values to the position for the encoders to run to. After this, runStraight uses setMotorPower, described above, to set all the drive motors to a speed of 0.4. Then it uses a boolean called hasReached, which returns true if all 4 encoders return a value within the tolerance of 10. If hasReached is true, then runStraight stops the motors. Otherwise, it keeps going till it is true. Once runStraight has finished moving it outputs true, so it is useful for debugging and knowing when the robot has gone a certain distance. There is a telemetry function called basicTel for easily inserting basic telemetry for the drive motor positions and the targets they are aiming to reach. The setZiplinePosition function makes the values used to control continuous servos more intuitve because -1 now goes backwards, instead of anything below 9/17 going backwards. 1 now goes forward, instead of anything above 9/17, and 0 stops the servo, instead of 9/17. manualDrive sets the motor to run without encoders and sets the power to the gamepad values. However, it also has a turtleDrive option, if true is inputted into the manualDrive instead of false. This turtleDrive option goes at 70% power. The stop function stops all motors and servos

Future Improvements/Current Flaws

A flaws and thing left to improve is that we need to measure out how many encoder counts it takes to turn a specific angle so that we can turn accurately in the future. We can also make a function to set power to the arm pivot motor, and in this function also clip it, to restrict any mechanical stress or even parts breaking since they couldn't go as far as they were pushed.

MainTeleOp

Code

```
package com.qualcomm.ftcrobotcontroller.opmodes;

public class MainTeleOp extends OpHelperClean {

    //TODO: Talk to drive team about controller prefs/who controls what
    //operator = gamepad2; driver = gamepad1

    public MainTeleOp(){

    }


    @Override
    public void loop() {
        //enable basic feedback
        basicTel();


        //move robot using joysticks
        if(gamepad1.right_bumper && gamepad1.left_bumper){
            manualDrive(true);
        } else{
            manualDrive(false);
        }


        //Handle zipliner positions
        if(gamepad2.x){
            setZipLinePosition(1);
        }

        if(gamepad2.b){
            setZipLinePosition(-1);
        }

        if(gamepad2.a){
            setZipLinePosition(0);
```

```
        }


        //handle arm pivot
        if(gamepad2.left_bumper){
            setArmPivot(-.2);
        }else if(gamepad2.right_bumper){
            setArmPivot(.2);
        } else{
            setArmPivot(0);
        }


        //handle tape measure movement
        if(gamepad2.left_trigger > 0) {
            moveTapeMeasure(.2);
        } else if(gamepad2.right_trigger > 0){
            moveTapeMeasure(-.2);
        } else{
            moveTapeMeasure(0);
        }

    }

    @Override
    public void stop() {

    }

}
```

Design/Function

MainTeleop was made to drive the robot in the driver-controlled period using the gamepads and the phones. It works by receiving gamepad input by one phone and interpreting it to do a specific task

The class starts out with using the basicTel function above to have full drive train telemetry in the loop. Then it says manualDrive has an input of false so it is at 100% speed but if both bumpers on gamepad 1, for the driver, are triggered, then turtleDrive is activated at 70% speed. Then, buttons on gamepad 2, for the operator, are assigned for making the zipline go in one direction,

go in the other direction or to make it stop. After that, the tape measure extender motors are made to go out if the left trigger on gamepad 2 is activated and come back in if the right trigger is activated. If nothing is activated, it should stop

Future Improvements/Current Flaws

A improvement could be to make turtleDrive activated by only one of the gamepad 1 bumpers for simplicity in competition. Also we need to make a button to make the arm pivot motor run so that the arm can pivot, because otherwise it is useless.

Auton

Code

```
package com.qualcomm.ftcrobotcontroller.opmodes;

public class TestEncoders extends OpHelperClean{


    //establish run states for auton
    enum RunState{
        RESET_STATE,
        FIRST_STATE,
        FIRST_RESET,
        SECOND_STATE,
        SECOND_RESET,
        THIRD_STATE,
        FOURTH_STATE,
        LAST_STATE
    }


    private RunState rs = RunState.RESET_STATE;

    public TestEncoders() {}


    @Override
    public void loop() {
```

```
        basicTel();
        setToEncoderMode();

        switch(rs) {
            case RESET_STATE:
            {
                resetEncoders();
                rs=RunState.FIRST_STATE;
                break;
            }
            case FIRST_STATE:
            {

                if(runStraight(10) ){
                    rs = RunState.FIRST_RESET;
                }
                break;
            }
            case FIRST_RESET: {

                if(resetEncoders()){//make sure that the encoder have reset
                    rs = RunState.SECOND_STATE;
                }
                break;
            }
            case SECOND_STATE:
            {
                if (runStraight(5)){
                    rs = RunState.LAST_STATE;
                }
                break;
            }

            case LAST_STATE:
            {
                stop();
            }
        }
    }
}
```

Design/Function

For the autonomous part of our run, we used encoders to create two autonomous codes: one for climbing the blue mountain side and one for the red mountain side. In both, the robot uses encoder degrees (1 encoder degree = 3 1/9 regular degrees) and motor power to go forward and park perpendicular to the mountain. Then, it pivot turns right and goes forward up the mountain until the high parking goal, we created an encoder tester in which we can test if the encoders reset every time.

We used enumeration (labelling numbers) to define eight different states. It comprises of a reset state, first state, first reset state, second state, second reset state, third state, fourth state, and a last state. If it is a reset state, it resets encoders and sets the run state to first state, and it continues changing states until it hits last state and is done detecting the distance to run. It may also start from any other state and continue down the chain from a different start point.

Future Improvements/Current Flaws

N/A