Code Listing

Typeset using LaTeX.

Tele-Op:

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.OpticalDistanceSensor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.Range;

@TeleOp(name = "FinalTeleOp", group = "Competition")
//@Disabled
public class FinalTeleOp extends LinearOpMode {

    private DcMotor rightBack, leftBack, rightFront, leftFront;
    private DcMotor intake, shooter1, shooter2;

    private Servo ballFeeder;

    private OpticalDistanceSensor ods;

    //final String NORMAL = "normal", STRAIGHT = "straight";
    private final double POWER_FACTOR = 1, POSITIVE_STEP = 0.2, NEGATIVE_STEP = 0.5;
    protected static final double SHOOTER2_OFFSET = 0.07;
    private final double INTAKE_POWER = 0.7;
    private final double SHOOT = Util.SHOOT, LOAD = Util.LOAD;
    private final long MILLIS_PER_NANO = 1000000;

    //String driveMode = NORMAL;
    private long shooterStart = System.nanoTime(), shooterLoadTimer = shooterStart;
    private double targetPowerR = 1, targetPowerL = 1, currentR = 1, currentL = 1;
    private boolean shooterStatus = false; //, aHasBeenPressed = false;

    private int intakeStatus = 0;
    private boolean intakeChanged = false;

    public void runOpMode() throws InterruptedException {
        Util.colorSensors = false; Util.otherSensors = true; Util.servos = true;
        Util.init(this);

        this.rightBack = Util.rightBack;
        this.leftBack = Util.leftBack;
        this.rightFront = Util.rightFront;
        this.leftFront = Util.leftFront;

        this.shooter1 = Util.shooter1;
        this.shooter2 = Util.shooter2;

        this.intake = Util.intake;

        this.ballFeeder = Util.ballFeeder;

        this.ods = Util.ods;
```

```
    waitForStart();

    while (opModeIsActive()) {

        handleDriveMotors();

        handleIntake();

        handleShooter();



        telemetry.update();

        Thread.sleep(10);
    }
}

private void handleDriveMotors() throws InterruptedException {
    if (gamepad1.dpad_up || gamepad1.dpad_down || gamepad1.dpad_left || gamepad1.
        ↪ dpad_right) {
        DpadDrive();
    } else if (Math.abs(gamepad1.right_stick_y) > JOYSTICK_DEADZONE_LIMIT ||
               Math.abs(gamepad1.left_stick_y) > JOYSTICK_DEADZONE_LIMIT) {
        joystickDrive();
    } else if (gamepad1.a || gamepad1.y) {
        pressBeacon();
    } else {
        Util.setAllPowers(0);
        currentR = 1; currentL = 1;
    }

    telemetry.update();*/
}

final double DpadPower = 0.2;
final int DpadTime = 100;

private void DpadDrive() throws InterruptedException {
    if (gamepad1.dpad_up) {
        if (!gamepad1.dpad_right) Util.setLeftPowers(DpadPower);
        if (!gamepad1.dpad_left) Util.setRightPowers(DpadPower);
    } else if (gamepad1.dpad_down) {
        if (!gamepad1.dpad_right) Util.setLeftPowers(-DpadPower);
        if (!gamepad1.dpad_left) Util.setRightPowers(-DpadPower);
    } else if (gamepad1.dpad_right) {
        Util.setRightPowers(-DpadPower);
        Util.setLeftPowers(DpadPower);
    } else if (gamepad1.dpad_left) {
        Util.setRightPowers(DpadPower);
        Util.setLeftPowers(-DpadPower);
    }

    Thread.sleep(DpadTime);
    Util.setAllPowers(0);
```

```
    while (gamepad1.dpad_up || gamepad1.dpad_down || gamepad1.dpad_left || gamepad1.
        ↪ dpad_right) Thread.sleep(10);
}

private void joystickDrive() {
    double r = Util.getGamepadRightJoystickY(gamepad1);
    double l = Util.getGamepadLeftJoystickY(gamepad1);

    r = scaleDriveJoystick(r);
    l = scaleDriveJoystick(l);

    targetPowerR = r + 1;
    targetPowerL = l + 1;
        /*if (driveMode.equals(NORMAL)) {
            targetPowerL = l + 1;
        } else {
            targetPowerL = r + 1;
        }*/

    if (currentR < (targetPowerR - POSITIVE_STEP)) {
        currentR += POSITIVE_STEP;
    } else if (currentR < targetPowerR) {
        currentR = targetPowerR;
    }

    if (currentR > (targetPowerR + NEGATIVE_STEP)) {
        currentR -= NEGATIVE_STEP;
    } else if (currentR > targetPowerR) {
        currentR = targetPowerR;
    }

    if (currentL < (targetPowerL - POSITIVE_STEP)) {
        currentL += POSITIVE_STEP;
    } else if (currentL < targetPowerL) {
        currentL = targetPowerL;
    }

    if (currentL > (targetPowerL + NEGATIVE_STEP)) {
        currentL -= NEGATIVE_STEP;
    } else if (currentL > targetPowerL) {
        currentL = targetPowerL;
    }

    rightBack.setPower((currentR - 1) * POWER_FACTOR);
    leftBack.setPower((currentL - 1) * POWER_FACTOR);
    rightFront.setPower((currentR - 1) * POWER_FACTOR);
    leftFront.setPower((currentL - 1) * POWER_FACTOR);
}

private final double JOYSTICK_DEADZONE_LIMIT = 0.1;
private final double MIN_POWER = 0.1;
private final double B = 13.2699, A = 0.0684;

private double scaleDriveJoystick(double joystickValue) {
    // if the joystick is in the deadzone I defined, return 0
    if (Math.abs(joystickValue) < JOYSTICK_DEADZONE_LIMIT) return 0.0;
    // use the formula A*B^(joystickValue)
```

```
    double power = Math.signum(joystickValue) * A * Math.pow(B, Math.abs(joystickValue));
    if (Math.abs(power) < MIN_POWER) return 0.0;
    return Range.clip(power, -1.0, 1.0);
}

private void pressBeacon() throws InterruptedException {
    Util.upDown.setPosition(Util.BEACON_DOWN);
    if (gamepad1.y) {
        if (beaconForward() == -1) return;
    }
    else if (gamepad1.a && !gamepad1.start) {
        if (beaconBackward() == -1) return;
    }
}

private int beaconForward() throws InterruptedException {
    Util.setRightPowers(0.21);
    Util.setLeftPowers(0.19);
    //if (lookForLineAndCheckJoystick(0.50) == -1) return -1;
    if (lookForLineAndCheckJoystick(0.5) == -1) return -1;
    if (sleepAndCheckJoystick(250) == -1) return -1;
    Util.setAllPowers(0);
    if (sleepAndCheckJoystick(50) == -1) return -1;
    Util.setRightPowers(-0.16);
    Util.setLeftPowers(-0.14);
    if (sleepAndCheckJoystick(1000) == -1) return -1;
    return 0;
}

private int beaconBackward() throws InterruptedException {
    Util.setRightPowers(-0.21);
    Util.setLeftPowers(-0.19);
    //if (lookForLineAndCheckJoystick(0.50) == -1) return -1;
    if (lookForLineAndCheckJoystick(0.5) == -1) return -1;
    if (sleepAndCheckJoystick(50) == -1) return -1;
    Util.setAllPowers(0);
    if (sleepAndCheckJoystick(50) == -1) return -1;
    Util.setRightPowers(0.16);
    Util.setLeftPowers(0.14);
    if (sleepAndCheckJoystick(1000) == -1) return -1;
    return 0;
}

private int lookForLineAndCheckJoystick(double lightThreshold) throws
    ↪ InterruptedException {
    while (ods.getLightDetected() < lightThreshold) {
        if (Math.abs(gamepad1.right_stick_y) > JOYSTICK_DEADZONE_LIMIT || Math.abs(
            ↪ gamepad1.left_stick_y) > JOYSTICK_DEADZONE_LIMIT) return -1;
        Thread.sleep(20);
    }
    return 0;
}

// Might not need this method in the end
private int sleepAndCheckJoystick(int sleepTimeMillis) throws InterruptedException {
    long startTime = System.nanoTime() / MILLIS_PER_NANO;
    while (((System.nanoTime() / MILLIS_PER_NANO) - startTime) < sleepTimeMillis) {
```

```
            if (Math.abs(gamepad1.right_stick_y) > JOYSTICK_DEADZONE_LIMIT || Math.abs(
                ↪ gamepad1.left_stick_y) > JOYSTICK_DEADZONE_LIMIT) return -1;
            Thread.sleep(20);
        }
        return 0;
    }

    // intake variables
    private final int INTAKE_OFF = 0, INTAKE = 1, OUTTAKE = 2;

    private void handleIntake() {
        if ((gamepad1.right_bumper && gamepad1.left_bumper) && !intakeChanged) {
            /* if the intake is off, outtake
             * if the intake is intaking, outtake
             * if the intake is outtaking, do nothing
             */
            switch (intakeStatus) {
                case INTAKE_OFF:
                case INTAKE: outtake(); break;
                case OUTTAKE: break;
            }
            intakeChanged = true;
        }
        if (gamepad1.left_bumper && !intakeChanged) {
            /* if the intake is off, do nothing
             * if the intake is intaking, turn it off
             * if the intake is outtaking, turn it off
             */
            switch (intakeStatus) {
                case INTAKE_OFF: break;
                case INTAKE:
                case OUTTAKE: intakeOff(); break;
            }
            intakeChanged = true;
        }
        if (gamepad1.right_bumper && !intakeChanged) {
            /* if the intake is off, intake
             * if the intake is intaking, do nothing
             * if the intake is outtaking, intake
             */
            switch (intakeStatus) {
                case INTAKE_OFF: intake(); break;
                case INTAKE: break;
                case OUTTAKE: intake(); break;
            }
            intakeChanged = true;
        }
        // wait until the user releases all intake-related buttons before allowing the user
            ↪ to change the intake again
        else if (!gamepad1.right_bumper && !gamepad1.left_bumper) {
            intakeChanged = false;
            if (intakeStatus == OUTTAKE) {
                intakeOff();
            }
        }
    }
```

```
    // the three following methods standardize intaking, outtaking, and neither
    private void intake() {
        this.intake.setPower(INTAKE_POWER);
        intakeStatus = INTAKE;
    }

    private void outtake() {
        this.intake.setPower(-INTAKE_POWER);
        intakeStatus = OUTTAKE;
    }

    private void intakeOff() {
        this.intake.setPower(0);
        intakeStatus = INTAKE_OFF;
    }

    private boolean SHOOTER_ON = true, SHOOTER_OFF = false;
    private int shooterSpinUp = 2000, shooterLoad = 2000, shooterFire = 400;

    private void handleShooter() throws InterruptedException {
        long time = System.nanoTime() / 1000000;

        if (gamepad1.right_trigger >= 0.5) {
            double power = calculateShooterPower();
            shooter1.setPower(power);
            shooter2.setPower(power + SHOOTER2_OFFSET);
            shooterStart = time;
            shooterStatus = SHOOTER_ON;
        }
        if (gamepad1.left_trigger >= 0.5) {
            shooter1.setPower(0);
            shooter2.setPower(0);
            shooterStatus = SHOOTER_OFF;
        }

        if (gamepad1.b && shooterStatus && (time - shooterStart) > shooterSpinUp) { // && (
            ↪ time - shooterLoadTimer) > shooterLoad) {
            ballFeeder.setPosition(this.SHOOT);
            Thread.sleep(shooterFire);
            ballFeeder.setPosition(this.LOAD);
            //shooterLoadTimer = System.nanoTime();
        }
    }

    protected static double calculateShooterPower() {
        double voltage = Util.getBatteryVoltage();
        if (voltage >= 13.6) return -0.033*voltage + 0.696;
        else return -0.04*Util.getBatteryVoltage() + 0.784;
    }
}
            /*if (driveMode.equals(NORMAL)) {
                rightBack.setPower((currentR - 1) * POWER_FACTOR);
                leftBack.setPower((currentL - 1) * POWER_FACTOR);
                if (!Util.TANK) {
                    rightFront.setPower((currentR - 1) * POWER_FACTOR);
                    leftFront.setPower((currentL - 1) * POWER_FACTOR);
                }
```

```
} else {
    rightBack.setPower((currentR - 1) * POWER_FACTOR);
    leftBack.setPower((currentR - 1) * POWER_FACTOR);
    if (!Util.TANK) {
        rightFront.setPower((currentR - 1) * POWER_FACTOR);
        leftFront.setPower((currentR - 1) * POWER_FACTOR);
    }
}

if (!aHasBeenPressed && gamepad1.a && driveMode.equals(NORMAL)) {
    driveMode = STRAIGHT;
    aHasBeenPressed = true;
} else if (!aHasBeenPressed && gamepad1.a && driveMode.equals(STRAIGHT)) {
    driveMode = NORMAL;
    aHasBeenPressed = true;
}

if (!gamepad1.a) aHasBeenPressed = false;*/

telemetry.update();*/
```

Red Autonomous (Blue Autonomous is very similar)

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.GyroSensor;
import com.qualcomm.robotcore.hardware.OpticalDistanceSensor;
import com.qualcomm.robotcore.hardware.Servo;

@Autonomous(name="RedAuto", group="Competition")
//@Disabled
public class RedAuto extends LinearOpMode {

    // motors
    DcMotor rightBack, leftBack, rightFront, leftFront;
    DcMotor shooter1, shooter2;
    DcMotor[] motors;

    // servos
    Servo ballFeeder, upDown;

    // sensors
    OpticalDistanceSensor ods;
    GyroSensor gyro;

    // autonomous constants
    final int BEACON_MOVE = 400;
    final double BEACON_POWER = 0.15;

    public void runOpMode() throws InterruptedException {
        Util.colorSensors = true; Util.otherSensors = true; Util.servos = true;
        Util.init(this);

        // drive motors
        this.rightBack = Util.rightBack; this.leftBack = Util.leftBack;
        this.rightFront = Util.rightFront; this.leftFront = Util.leftFront;

        motors = new DcMotor[4]; motors[0] = this.rightBack; motors[1] = this.leftBack;
            ↪ motors[2] = this.rightFront; motors[3] = this.leftFront;

        // shooter motors
        this.shooter1 = Util.shooter1; this.shooter2 = Util.shooter2;

        // servos
        this.ballFeeder = Util.ballFeeder;
        this.upDown = Util.upDown;

        // otherSensors
        this.ods = Util.ods;
        this.gyro = Util.gyro;
        I2C_ColorSensor.init(this);

        Util.resetEncoders(this, motors);

        waitForStart();
```

```
double shooterPower = FinalTeleOp.calculateShooterPower();
shooter1.setPower(shooterPower);
shooter2.setPower(shooterPower + FinalTeleOp.SHOOTER2_OFFSET);

Thread.sleep(500);

rightBack.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.FLOAT);
leftBack.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.FLOAT);
rightFront.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.FLOAT);
leftFront.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.FLOAT);

AutoUtil.PID_Forward(2200, 0.2, true, gyro);

Thread.sleep(200 + 500);

rightBack.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
leftBack.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
rightFront.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
leftFront.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);

shoot2();

AutoUtil.encoderTurnLeft(70, 0.2);

Thread.sleep(100);

AutoUtil.PID_Forward(4100, 0.2, true, gyro);

Thread.sleep(100);

AutoUtil.encoderTurnRight(60, 0.2);

Thread.sleep(100);

AutoUtil.PID_Forward(2250, 0.3, false, gyro);

AutoUtil.encoderSteerForward(1500, 0.3, false);

AutoUtil.encoderSteerForwardLine(0.5, 0.1, false);

AutoUtil.encoderSteerForward(240, 0.1, true);

//boolean done = false;
if (I2C_ColorSensor.beaconIsRedBlue()) {
    // first try
    Util.telemetry("beacon_status", "RED_BLUE", true);
    AutoUtil.encoderSteerForward(BEACON_MOVE, BEACON_POWER, true);
    AutoUtil.beaconDown(upDown);
    AutoUtil.encoderSteerBackward(BEACON_MOVE, BEACON_POWER, true);
    Thread.sleep(100);
    AutoUtil.encoderSteerForward(BEACON_MOVE, BEACON_POWER, true);
    AutoUtil.beaconUp(upDown);
    AutoUtil.encoderSteerBackward(2000 + BEACON_MOVE, 0.3, false);
} else if (I2C_ColorSensor.beaconIsBlueRed()) {
    Util.telemetry("beacon_status", "BLUE_RED", true);
    AutoUtil.encoderSteerBackward(BEACON_MOVE, BEACON_POWER, true);
    AutoUtil.beaconDown(upDown);
```

```
            AutoUtil.encoderSteerForward(BEACON_MOVE, BEACON_POWER, true);
            Thread.sleep(100);
            AutoUtil.encoderSteerBackward(500, BEACON_POWER, false);
            AutoUtil.encoderSteerBackward(1500, 0.3, false);
            AutoUtil.beaconUp(upDown);
        }

        AutoUtil.encoderSteerBackwardLine(0.5, 0.1, true);

        Thread.sleep(100);

        AutoUtil.encoderSteerForward(280, 0.1, true);

        if (I2C_ColorSensor.beaconIsRedBlue()) {
            AutoUtil.encoderSteerForward(BEACON_MOVE, BEACON_POWER, true);
            AutoUtil.beaconDown(upDown);
            AutoUtil.encoderSteerBackward(BEACON_MOVE, BEACON_POWER, true);
            Thread.sleep(100);
            AutoUtil.encoderSteerForward(BEACON_MOVE, BEACON_POWER, true);
            AutoUtil.beaconUp(upDown);
        } else if (I2C_ColorSensor.beaconIsBlueRed()) {
            AutoUtil.encoderSteerBackward(BEACON_MOVE, BEACON_POWER, true);
            AutoUtil.beaconDown(upDown);
            AutoUtil.encoderSteerForward(BEACON_MOVE, BEACON_POWER, true);
            Thread.sleep(100);
            AutoUtil.encoderSteerBackward(BEACON_MOVE, BEACON_POWER, true);
            AutoUtil.beaconUp(upDown);
        }

        Util.setAllPowers(0);

        while(opModeIsActive()) Thread.sleep(100);
    }

    private void shoot2() throws InterruptedException {
        ballFeeder.setPosition(Util.SHOOT);

        Thread.sleep(400);

        ballFeeder.setPosition(Util.LOAD);

        Thread.sleep(1000);

        ballFeeder.setPosition(Util.SHOOT);

        Thread.sleep(500);

        shooter1.setPower(0);
        shooter2.setPower(0);
        ballFeeder.setPosition(Util.LOAD);
    }
}
```

Util class:

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;
import com.qualcomm.robotcore.hardware.Gamepad;
import com.qualcomm.robotcore.hardware.GyroSensor;
import com.qualcomm.robotcore.hardware.OpticalDistanceSensor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.hardware.HardwareMap;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.util.RobotLog;

import org.firstinspires.ftc.robotcontroller.internal.FtcRobotControllerActivity;

public final class Util {

    protected static boolean init = false;
    //protected static boolean gyroEnabled = false;

    protected static DcMotor rightBack, leftBack, rightFront, leftFront;
    protected static DcMotor shooter1, shooter2, intake;

    protected static Servo ballFeeder, upDown;

    protected static OpticalDistanceSensor ods;

    protected static GyroSensor gyro;

    protected static boolean colorSensors = false, otherSensors = true, servos = true;

    protected final static double SEC_TO_NSEC = 1000000000, NEVEREST_37_TICS_PER_ROTATION =
        ↪ 44.4;
    protected final static double POWER_LIMIT = 1;
    protected static final double SHOOT = 0.5, LOAD = 0.95;
    protected static final double BEACON_UP = 0.6, BEACON_DOWN = 0.9; // was 0.95

    //private static LinearOpMode linearOpMode;
    protected static LinearOpMode linearOpMode;
    private static DcMotor[] /*motors,*/ motorsWithEncoders;

    private Util() throws Exception {
        throw new Exception();
    }

    public static void init(LinearOpMode opMode) throws InterruptedException {
        linearOpMode = opMode;

        DcMotor[] temp;
        DcMotor[] tempWithEncoders;

        // drive motors
        rightBack = opMode.hardwareMap.dcMotor.get("rightBack"); rightBack.setDirection(
            ↪ DcMotor.Direction.REVERSE);
        leftBack = opMode.hardwareMap.dcMotor.get("leftBack");
        rightFront = opMode.hardwareMap.dcMotor.get("rightFront"); rightFront.setDirection(
            ↪ DcMotor.Direction.REVERSE);
```

```
        leftFront = opMode.hardwareMap.dcMotor.get("leftFront");

        rightBack.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        leftBack.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        rightFront.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        leftFront.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);

        AutoUtil.r = rightFront; AutoUtil.l = leftFront;


        temp = new DcMotor[4]; temp[0] = rightBack; temp[1] = leftBack; temp[2] = rightFront;
            ↪  temp[3] = leftFront;
        tempWithEncoders = temp;

        //motors = temp;
        motorsWithEncoders = tempWithEncoders;

        // shooter motors
        shooter1 = getMotor("shooter1");
        shooter2 = getMotor("shooter2"); shooter2.setDirection(DcMotorSimple.Direction.
            ↪ REVERSE);
        shooter1.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.FLOAT);
        shooter2.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.FLOAT);

        // intake motor
        intake = getMotor("intake");

        // servos
        if (servos) {
            ballFeeder = getServo("ballFeeder"); ballFeeder.setPosition(LOAD);
            upDown = getServo("upDown"); upDown.setPosition(BEACON_UP);
        }

        // color sensors
        if (colorSensors) I2C_ColorSensor.init(opMode);

        // other sensors
        if (otherSensors) {
            ods = opMode.hardwareMap.opticalDistanceSensor.get("ods");
            gyro = opMode.hardwareMap.gyroSensor.get("gyro");
        }

        //resetEncoders();

        init = true;
    }

    public static DcMotor getMotor(HardwareMap map, String deviceName) {
        return map.dcMotor.get(deviceName);
    }

    public static DcMotor getMotor(String deviceName) {
        return linearOpMode.hardwareMap.dcMotor.get(deviceName);
    }

    public static Servo getServo(HardwareMap map, String deviceName) {
        return map.servo.get(deviceName);
```

```
}

public static Servo getServo(String deviceName) {
    return linearOpMode.hardwareMap.servo.get(deviceName);
}

public static void resetEncoders(LinearOpMode opMode, DcMotor[] motorList) throws
    ↪ InterruptedException {
    for (DcMotor motor : motorList) motor.setMode(DcMotor.RunMode.RESET_ENCODERS);
    Thread.sleep(200);
    //while (motorList[0].getMode() != DcMotorController.RunMode.RESET_ENCODERS);
    for (DcMotor motor : motorList) motor.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODERS);
    Thread.sleep(200);
}

public static void resetEncoders(LinearOpMode opMode) throws InterruptedException {
    resetEncoders(opMode, motorsWithEncoders);
}

public static void resetEncoders(DcMotor[] motorList) throws InterruptedException {
    resetEncoders(linearOpMode, motorList);
}

public static void resetEncoders() throws InterruptedException {
    resetEncoders(linearOpMode, motorsWithEncoders);
}

public static double getBatteryVoltage() {
    return linearOpMode.hardwareMap.voltageSensor.iterator().next().getVoltage();
}

public static double getGamepadRightJoystickY(Gamepad gamepad) {
    double joystick;
    joystick = gamepad.right_stick_y;
    if (joystick != 0) return -joystick;
    return joystick;
}

public static double getGamepadLeftJoystickY(Gamepad gamepad) {
    double joystick;
    joystick = gamepad.left_stick_y;
    if (joystick != 0) return -joystick;
    return joystick;
}

public static void setRightPowers(double p) {
    rightBack.setPower(p);
    rightFront.setPower(p);
}

public static void setLeftPowers(double p) {
    leftBack.setPower(p);
    leftFront.setPower(p);
}

public static void setFrontPowers(double p) {
    rightFront.setPower(p);
```

```
        leftFront.setPower(p);
    }

    public static void setBackPowers(double p) {
        rightBack.setPower(p);
        leftBack.setPower(p);
    }

    public static void setAllPowers(double p) {
        rightBack.setPower(p);
        leftBack.setPower(p);
        rightFront.setPower(p);
        leftFront.setPower(p);
    }

    public static void setMotorsPowers(DcMotor[] motors, double p) {
        for (DcMotor motor : motors) {
            motor.setPower(p);
        }
    }

    public static void log(String message) {
        if (!FtcRobotControllerActivity.LOG) return;
        RobotLog.i(message);
    }

    public static void telemetry(String key, String data) {
        Util.linearOpMode.telemetry.update();
    }

    public static void telemetry(String key, int data) {
        Util.linearOpMode.telemetry.update();
    }

    public static void telemetry(String key, double data) {
        Util.linearOpMode.telemetry.update();
    }

    public static void telemetry(String key, String data, boolean update) {
        if (update) Util.linearOpMode.telemetry.update();
    }

    public static void telemetry(String key, int data, boolean update) {
        if (update) Util.linearOpMode.telemetry.update();
    }

    public static void telemetry(String key, double data, boolean update) {
        if (update) Util.linearOpMode.telemetry.update();
    }
}
```

AutoUtil class:

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.GyroSensor;
import com.qualcomm.robotcore.hardware.Servo;

public final class AutoUtil {

    protected static boolean init = false;
    protected static LinearOpMode linearOpMode;
    private static float stallEnabledTime;
    private static double powerFactor = Util.POWER_LIMIT;
    protected static DcMotor r, l;
    private static final double MOTOR_POWER_THRESHOLD = 0.8 * Util.POWER_LIMIT,
        ↪ TIME_THRESHOLD = 0.3 * Util.SEC_TO_NSEC;
    private static final double MIN_POWER = 0.2;

    private AutoUtil() throws Exception {
        throw new Exception();
    }

    public static void init(LinearOpMode opmode, GyroSensor gyro) throws InterruptedException
        ↪  {
        Util.init(opmode);
        StallProtection.init();
        linearOpMode = Util.linearOpMode;
        resetEncoders();
        calibrateGyro(gyro);
        init = true;
    }

    public static void encoderForward(int dist, double power, boolean stop) throws
        ↪ InterruptedException {
        int pos = (r.getCurrentPosition() + l.getCurrentPosition()) / 2;

        Util.setAllPowers(power);

        while (((r.getCurrentPosition() + l.getCurrentPosition()) / 2) < (pos + dist)) Thread
            ↪ .sleep(20);

        if (stop) Util.setAllPowers(0);
    }

    public static void encoderBackward(int dist, double power, boolean stop) throws
        ↪ InterruptedException {
        int pos = (r.getCurrentPosition() + l.getCurrentPosition()) / 2;

        Util.setAllPowers(-power);

        while (((r.getCurrentPosition() + l.getCurrentPosition()) / 2) > (pos - dist)) Thread
            ↪ .sleep(20);

        if (stop) Util.setAllPowers(0);
    }
```

```
public static void encoderSteerForward(int dist, double power, boolean stop) throws
    ↪ InterruptedException {
    int pos = (r.getCurrentPosition() + l.getCurrentPosition()) / 2;

    Util.setRightPowers(power * 1.08);
    Util.setLeftPowers(power / 1.08);

    while (((r.getCurrentPosition() + l.getCurrentPosition()) / 2) < (pos + dist)) Thread
        ↪ .sleep(20);

    if (stop) Util.setAllPowers(0);
}

public static void encoderSteerBackward(int dist, double power, boolean stop) throws
    ↪ InterruptedException {
    int pos = (r.getCurrentPosition() + l.getCurrentPosition()) / 2;

    Util.setRightPowers(-power * 1.08);
    Util.setLeftPowers(-power / 1.08);

    while (((r.getCurrentPosition() + l.getCurrentPosition()) / 2) > (pos - dist)) Thread
        ↪ .sleep(20);

    if (stop) Util.setAllPowers(0);
}

public static void encoderSteerForwardLine(double threshold, double power, boolean stop)
    ↪ throws InterruptedException {
    Util.setRightPowers(power * 1.08);
    Util.setLeftPowers(power / 1.08);

    while (Util.ods.getLightDetected() < threshold) Thread.sleep(20);

    if (stop) Util.setAllPowers(0);
}

public static void encoderSteerBackwardLine(double threshold, double power, boolean stop)
    ↪  throws InterruptedException {
    Util.setRightPowers(-power * 1.08);
    Util.setLeftPowers(-power / 1.08);

    while (Util.ods.getLightDetected() < threshold) Thread.sleep(20);

    if (stop) Util.setAllPowers(0);
}

public static void PID_Forward(double distance, double power, boolean stop, GyroSensor
    ↪ gyro) throws InterruptedException {
    resetGyroHeading(gyro);
    PID.resetDriveIntegral();
    double start = Util.rightBack.getCurrentPosition();
    Util.setAllPowers(0.1);
    Thread.sleep(30);
    Util.setAllPowers(0.15);
    Thread.sleep(75);
    while (Util.rightBack.getCurrentPosition() < (start + (distance * 0.98))) {
        PID.PIsetMotors(gyro, powerFactor * power);
```

```
        Thread.sleep(10);
    }
    if (stop) Util.setAllPowers(0);
}

public static void PID_Backward(double distance, double power, boolean stop, GyroSensor
    ↪ gyro) throws InterruptedException {
    resetGyroHeading(gyro);
    PID.resetDriveIntegral();
    double start = Util.rightBack.getCurrentPosition();
    Util.setAllPowers(-0.1);
    Thread.sleep(30);
    while (Util.rightBack.getCurrentPosition() > (start - (distance * 0.98))) {
        PID.PIsetMotors(gyro, powerFactor * -power);
        Thread.sleep(10);
    }
    if (stop) Util.setAllPowers(0);
}

final static double RAMP_UP_DELTA = 0.02, RAMP_DOWN_DELTA = 0.03;
final static int EXTRA_DEGREES = 3; // 1

public static void encoderTurnRight(double degrees, double power) throws
    ↪ InterruptedException {
    Util.resetEncoders();

    double dist = degrees / 360;
    dist = dist * 15 / 4 * 1120;

    Util.setRightPowers(-power);
    Util.setLeftPowers(power);

    while (((Math.abs(r.getCurrentPosition()) + Math.abs(l.getCurrentPosition())) / 2) <
        ↪ dist) Thread.sleep(20);

    Util.setAllPowers(0);
}

public static void encoderTurnLeft(int degrees, double power) throws InterruptedException
    ↪  { encoderTurnRight(degrees, -power); }

public static void gyroTurnRight(double degreeTarget, double targetPower, GyroSensor gyro
    ↪ ) throws InterruptedException {
    resetGyroHeading(gyro);
    double power = MIN_POWER;
    while (PID.heading(gyro) < (degreeTarget / 2)) {
        power += RAMP_UP_DELTA;
        if (power > targetPower) {
            Util.setRightPowers(-targetPower);
            Util.setLeftPowers(targetPower);
            break;
        }
        Util.setRightPowers(-power);
        Util.setLeftPowers(power);
        Thread.sleep(10);
    }
    power = targetPower;
```

```
    double rampUpDegrees = PID.heading(gyro);
    while (degreeTarget - PID.heading(gyro) > rampUpDegrees * 2) Thread.sleep(10);
    while (PID.heading(gyro) - degreeTarget > EXTRA_DEGREES) {
        power -= RAMP_DOWN_DELTA;
        if (power < MIN_POWER) {
            Util.setRightPowers(-MIN_POWER);
            Util.setLeftPowers(MIN_POWER);
        } else {
            Util.setRightPowers(-power);
            Util.setLeftPowers(power);
        }
        Thread.sleep(10);
    }
    Util.setAllPowers(0);
}

public static void gyroTurnLeft(double degreeTarget, double targetPower, GyroSensor gyro)
    ↪  throws InterruptedException {
    degreeTarget = -degreeTarget;
    resetGyroHeading(gyro);
    double power = MIN_POWER;
    while (PID.heading(gyro) > (degreeTarget / 2)) {
        power += RAMP_UP_DELTA;
        if (power > targetPower) {
            Util.setRightPowers(targetPower);
            Util.setLeftPowers(-targetPower);
            break;
        }
        Util.setRightPowers(power);
        Util.setLeftPowers(-power);
        Thread.sleep(10);
    }
    power = targetPower;
    double rampUpDegrees = PID.heading(gyro);
    while (degreeTarget - PID.heading(gyro) < rampUpDegrees) Thread.sleep(10);
    while (PID.heading(gyro) - degreeTarget > EXTRA_DEGREES) {
        power -= RAMP_DOWN_DELTA;
        if (power < MIN_POWER) {
            Util.setRightPowers(MIN_POWER);
            Util.setLeftPowers(-MIN_POWER);
        } else {
            Util.setRightPowers(power);
            Util.setLeftPowers(-power);
        }
        Thread.sleep(10);
    }
    Util.setAllPowers(0);
}

public static void resetEncoders(DcMotor[] motors) throws InterruptedException {
    Util.resetEncoders(motors);
}

public static void resetEncoders() throws InterruptedException {
    Util.resetEncoders();
}
```

```
    public static void calibrateGyro(GyroSensor gyro) throws InterruptedException {
        gyro.calibrate();
        while (gyro.isCalibrating()) Thread.sleep(50);
    }

    public static void resetGyroHeading(GyroSensor gyro) {
        gyro.resetZAxisIntegrator();
    }

    public static void beaconUp(Servo servo) throws InterruptedException {
        servo.setPosition(Util.BEACON_UP);
        Thread.sleep(100);
    }

    public static void beaconDown(Servo servo) throws InterruptedException {
        servo.setPosition(Util.BEACON_DOWN);
        Thread.sleep(100);
    }

    /*public static double gyroDrift(GyroSensor gyro) throws InterruptedException {
        resetGyro(gyro);
        Thread.sleep(15000);
        int heading = gyro.getHeading();
        double error = heading;
        if (heading > 180) error = 360 - heading;
        return error / 15000;
    }*/
}
```

PID class:

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.hardware.GyroSensor;
import com.qualcomm.robotcore.util.Range;
import com.qualcomm.robotcore.util.RobotLog;

public final class PID {

    // drive
    private static final float Kp = 0.04f;   //proportional constant      //TODO final tune
    private static final float Ki = 0.0f;    //integral constant     //TODO tune
    private static final int offset = 0;          //value that <gyroHeading> should be
    private static double integral = 0;      //variable to hold integral value (accumulated
        ↪ error)

    //testing
    private static boolean log = false;// FtcRobotControllerActivity.LOG;

    private PID() throws Exception { throw new Exception(); }

    public static double[] P(GyroSensor gyro, double Tp) {
        //TODO factor in battery power
        int heading = heading(gyro);
        int error = heading - offset;
        double turn = Kp * error;
        double[] toReturn = {Range.clip(Tp - turn, -1, 1), Range.clip(Tp + turn, -1, 1)};

        if (!log) {
            return toReturn;
        }

        //logging
        RobotLog.i("-----------P_start-----------");
        RobotLog.i("Tp_(power)_" + Tp);
        RobotLog.i("gyro_heading:_" + gyro.getHeading());
        RobotLog.i("scaled_heading:_" + heading);
        RobotLog.i("error:_" + error);
        RobotLog.i("turn:_" + turn);
        RobotLog.i("right_power:_" + toReturn[0]);
        RobotLog.i("left_power:_" + toReturn[1]);
        RobotLog.i("-----------P_end-----------");

        return toReturn;
    }

    public static double[] PI(GyroSensor gyro, double Tp) {
        //TODO factor in battery power
        int heading = heading(gyro);
        int error = heading - offset;
        integral += error;
        double turn = Kp * error + Ki * integral;
        double[] toReturn = {Range.clip(Tp + turn, -1, 1), Range.clip(Tp - turn, -1, 1)};

        if (!log) {
            return toReturn;
        }
```

```
        //logging
        RobotLog.i("-----------P_start-----------");
        RobotLog.i("Tp_(power)_" + Tp);
        RobotLog.i("gyro_heading:_" + gyro.getHeading());
        RobotLog.i("scaled_heading:_" + heading);
        RobotLog.i("error:_" + error);
        RobotLog.i("integral:_" + integral);
        RobotLog.i("turn:_" + turn);RobotLog.i("right_power:_" + toReturn[0]);
        RobotLog.i("left_power:_" + toReturn[1]);
        RobotLog.i("-----------P_end-----------");

        return toReturn;
    }

    public static void PsetMotors(GyroSensor gyro, double Tp) {
        double[] motors = P(gyro, Tp);
        Util.setRightPowers(motors[0]);
        Util.setLeftPowers(motors[1]);
    }

    public static void PIsetMotors(GyroSensor gyro, double Tp) {
        double motors[] = PI(gyro, Tp);
        Util.setRightPowers(motors[0]);
        Util.setLeftPowers(motors[1]);
    }

    public static int heading(GyroSensor gyro) {
        int heading = gyro.getHeading();
        if (heading > 180) return heading - 360;
        return heading;
        //-179 - 180
    }

    public static void resetDriveIntegral() {
        integral = 0;
    }
}
```