
ROBOTICS PROGRAMING
JAVA
VINCENT VAN GOGH

Table of Contents

AprilTag.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.apriltag.AprilTagDetection;
import org.firstinspires.ftc.vision.apriltag.AprilTagProcessor;

import java.util.List;

@Autonomous(name = "AprilTag Vision thingie", group = "LinearOpmode")
@Disabled
public class AprilTag extends LinearOpMode {

    private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

    /**
     * The variable to store our instance of the AprilTag processor.
     */
    private AprilTagProcessor aprilTag;

    /**
     * The variable to store our instance of the vision portal.
     */

    private VisionPortal visionPortal;

    @Override
    public void runOpMode() {

        initAprilTag();

        // Wait for the DS start button to be touched.
```

```
    telemetry.addData("Oki slayy", "the camera is booting up");
    telemetry.addData(":>", "Click the start to begin");
    telemetry.update();
    waitForStart();

    if (opModeIsActive()) {
        while (opModeIsActive()) {

            telemetryAprilTag();

            // Push telemetry to the Driver Station.
            telemetry.update();

            /*for (AprilTagDetection detection : currentDetections) {
                if (detection.getLabel().equals("Id 4"))
                    telemetry.addData("YASSS!! Object is finally detected", "Id 4");
                        telemetry.update();
            }*/


            // Save CPU resources; can resume streaming when needed.
            if (gamepad1.dpad_down) {
                visionPortal.stopStreaming();
            } else if (gamepad1.dpad_up) {
                visionPortal.resumeStreaming();
            }

            // Share the CPU.
            sleep(20);
        }
    }

    // Save more CPU resources when camera is no longer needed.
    visionPortal.close();

}   // end method runOpMode()

/**
 * Initialize the AprilTag processor.
 */
```

```
private void initAprilTag() {

    // Create the AprilTag processor.
    aprilTag = new AprilTagProcessor.Builder()
        //.setDrawAxes(false)
        //.setDrawCubeProjection(false)
        //.setDrawTagOutline(true)
        .setTagFamily(AprilTagProcessor.TagFamily.TAG_36h11)
        //.setTagLibrary(AprilTagGameDatabase.getCenterStageTagLibrary())
        //.setOutputUnits(DistanceUnit.INCH, AngleUnit.DEGREES)

        // == CAMERA CALIBRATION ==
        // If you do not manually specify calibration parameters, the SDK will attempt
        // to load a predefined calibration for your camera.
        .setLensIntrinsics(578.272, 578.272, 402.145, 221.506)

        // ... these parameters are fx, fy, cx, cy.

        .build();

    // Create the vision portal by using a builder.
    VisionPortal.Builder builder = new VisionPortal.Builder();

    // Set the camera (webcam vs. built-in RC phone camera).
    if (USE_WEBCAM) {
        builder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
    } else {
        builder.setCamera(BuiltinCameraDirection.BACK);
    }

    // Choose a camera resolution. Not all cameras support all resolutions.
    //builder.setCameraResolution(new Size(640, 480));

    // Enable the RC preview (LiveView).  Set "false" to omit camera monitoring.
    //builder.enableCameraMonitoring(true);

    // Set the stream format; MJPEG uses less bandwidth than default YUY2.
    //builder.setStreamFormat(VisionPortal.StreamFormat.YUY2);

    // Choose whether or not LiveView stops if no processors are enabled.
```

```
        // If set "true", monitor shows solid orange screen if no processors enabled.
        // If set "false", monitor shows camera view without annotations.
        builder.setAutoStopLiveView(false);

        // Set and enable the processor.
        builder.addProcessor(aprilTag);

        // Build the Vision Portal, using the above settings.
        visionPortal = builder.build();

        // Disable or re-enable the aprilTag processor at any time.
        //visionPortal.setProcessorEnabled(aprilTag, true);

    }   // end method initAprilTag()


    /**
     * Add telemetry about AprilTag detections.
     */
    private void telemetryAprilTag() {

        List<AprilTagDetection> currentDetections = aprilTag.getDetections();
        telemetry.addData("# AprilTags Detected", currentDetections.size());

        // Step through the list of detections and display info for each one.
        for (AprilTagDetection detection : currentDetections) {
            if (detection.metadata != null) {
                telemetry.addLine(String.format("\n==== (ID %d) %s", detection.id,
detection.metadata.name));
                telemetry.addLine(String.format("XYZ %6.1f %6.1f %6.1f  (inch)",
detection.ftcPose.x, detection.ftcPose.y, detection.ftcPose.z));
                telemetry.addLine(String.format("PRY %6.1f %6.1f %6.1f  (deg)",
detection.ftcPose.pitch, detection.ftcPose.roll, detection.ftcPose.yaw));
                telemetry.addLine(String.format("RBE %6.1f %6.1f %6.1f  (inch, deg, deg)",
detection.ftcPose.range, detection.ftcPose.bearing, detection.ftcPose.elevation));
            } else {
                telemetry.addLine(String.format("\n==== (ID %d) Unknown", detection.id));
                telemetry.addLine(String.format("Center %6.0f %6.0f   (pixels)", detection.center.x,
detection.center.y));
            }
```

```
        }   // end for() loop

        // Add "key" information to telemetry
        telemetry.addLine("\nkey:\nXYZ = X (Right), Y (Forward), Z (Up) dist.");
        telemetry.addLine("PRY = Pitch, Roll & Yaw (XYZ Rotation)");
        telemetry.addLine("RBE = Range, Bearing & Elevation");

    }   // end method telemetryAprilTag()

}   // end class
```

AprilTagDriveTank.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import java.lang.annotation.Target;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.util.Range;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.ExposureControl;
import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.GainControl;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.apriltag.AprilTagDetection;
import org.firstinspires.ftc.vision.apriltag.AprilTagProcessor;

import java.util.List;
import java.util.concurrent.TimeUnit;

@Autonomous(name="Tank Drive To AprilTag", group = "LinearOpMode")
//@Disabled
public class AprilTagDriveTank extends LinearOpMode
{
    // Adjust these numbers to suit your robot.
```

```
    final double DESIRED_DISTANCE = 10.0;//12.0; //  this is how close the camera should get to the target (inches)

    //  Set the GAIN constants to control the relationship between the measured position error, and how much power is
    //  applied to the drive motors to correct the error.
    //  Drive = Error * Gain    Make these values smaller for smoother control, or larger for a more aggressive response.
    final double SPEED_GAIN =   0.02 ;   //  Speed Control "Gain". eg: Ramp up to 50% power at a 25 inch error.   (0.50 / 25.0)
    final double TURN_GAIN  =   0.01 ;   //  Turn Control "Gain".  eg: Ramp up to 25% power at a 25 degree error. (0.25 / 25.0)

    final double MAX_AUTO_SPEED = 0.5;   //  Clip the approach speed to this max value (adjust for your robot)
    final double MAX_AUTO_TURN  = 0.25;  //  Clip the turn speed to this max value (adjust for your robot)

    private DcMotor frontLeftDrive   = null;  //  Used to control the left drive wheel
    private DcMotor frontRightDrive  = null;//  Used to control the right drive wheel
    private DcMotor backLeftDrive   = null;  //  Used to control the left drive wheel
    private DcMotor backRightDrive  = null;

    private static final boolean USE_WEBCAM = true;  // Set true to use a webcam, or false for a phone camera

    /* !!!! Register :3 : When we integrate the program we need to set the desired tag variable
    to 1, 2, or 3 depending on which path the robot takes. Path 1= 1, etc.... :) !!!!*/
    private static final int DESIRED_TAG_ID = 0;     // Choose the tag you want to approach or set to -1 for ANY tag.
    private VisionPortal visionPortal;               // Used to manage the video source.
    private AprilTagProcessor aprilTag;              // Used for managing the AprilTag detection process.
    private AprilTagDetection desiredTag = null;     // Used to hold the data for a detected AprilTag

    @Override public void runOpMode()
    {
        boolean targetFound     = false;    // Set to true when an AprilTag target is detected
        double  drive           = 0;        // Desired forward power/speed (-1 to +1) +ve is forward
```

```
        double  turn            = 0;        // Desired turning power/speed (-1 to +1) +ve is
CounterClockwise

        // Initialize the Apriltag Detection process
        initAprilTag();

        // Initialize the hardware variables. Note that the strings used here as parameters
        // to 'get' must match the names assigned during the robot configuration.
        // step (using the FTC Robot Controller app on the phone).
        frontLeftDrive  = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
        frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
        backLeftDrive  = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
        backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");

        // To drive forward, most robots need the motor on one side to be reversed because the axles
point in opposite directions.
        // When run, this OpMode should start both motors driving forward. So adjust these two
lines based on your first test drive.
        // Note: The settings here assume direct drive on left and right wheels.  Single Gear
Reduction or 90 Deg drives may require direction flips
        frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
        frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
        backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
        backRightDrive.setDirection(DcMotor.Direction.FORWARD);

        if (USE_WEBCAM)
            setManualExposure(6, 250);  // Use low exposure time to reduce motion blur

        // Wait for the driver to press Start
        telemetry.addData("Camera preview on/off", "3 dots, Camera Stream");
        telemetry.addData(">", "Touch Play to start OpMode");
        telemetry.update();
        waitForStart();

        while (opModeIsActive())
        {
            targetFound = false;
            desiredTag  = null;

            // Step through the list of detected tags and look for a matching tag
```

```
        List<AprilTagDetection> currentDetections = aprilTag.getDetections();
        for (AprilTagDetection detection : currentDetections) {
            if ((detection.metadata != null) &&
                ((DESIRED_TAG_ID < 0) || (detection.id == DESIRED_TAG_ID))  ){
                targetFound = true;
                desiredTag = detection;
                break;  // don't look any further.
            } else {
                telemetry.addData("Unknown Target", "Tag ID %d is not in TagLibrary\n",
detection.id);
            }
        }

        // Tell the driver what we see, and what to do.
        //Register :3 :
        if (targetFound) {
            telemetry.addData(">", "haiiii"/* "HOLD Left-Bumper to Drive to Target\n"*/);
            telemetry.addData("Target", "ID %d (%s)", desiredTag.id, desiredTag.metadata.name);
            telemetry.addData("Range",  "%5.1f inches", desiredTag.ftcPose.range);
            telemetry.addData("Bearing","%3.0f degrees", desiredTag.ftcPose.bearing);
        } else {
            telemetry.addData(">","Drive using joysticks to find valid target\n");
        }

        // If Left Bumper is being pressed, AND we have found the desired target, Drive to target
Automatically .
        //Register :3 :
        if (/*gamepad1.left_bumper && */targetFound) {

            // Determine heading and range error so we can use them to control the robot
automatically.
            double  rangeError   = (desiredTag.ftcPose.range - DESIRED_DISTANCE);
            double  headingError = desiredTag.ftcPose.bearing;

            // Use the speed and turn "gains" to calculate how we want the robot to move.  Clip it
to the maximum
            drive = Range.clip(rangeError * SPEED_GAIN, -MAX_AUTO_SPEED,
MAX_AUTO_SPEED);
            turn  = Range.clip(headingError * TURN_GAIN, -MAX_AUTO_TURN,
MAX_AUTO_TURN) ;
```

```
            telemetry.addData("Auto","Drive %5.2f, Turn %5.2f", drive, turn);
        } else {

            // drive using manual POV Joystick mode.
            drive = -gamepad1.left_stick_y  / 2.0;  // Reduce drive rate to 50%.
            turn  = -gamepad1.right_stick_x / 4.0;  // Reduce turn rate to 25%.
            telemetry.addData("Manual","Drive %5.2f, Turn %5.2f", drive, turn);
        }
        telemetry.update();

        // Apply desired axes motions to the drivetrain.
        moveRobot(drive, turn);
        sleep(10);
    }
}

/**
 * Move robot according to desired axes motions
 * <p>
 * Positive X is forward
 * <p>
 * Positive Yaw is counter-clockwise
 */
public void moveRobot(double x, double yaw) {
    // Calculate left and right wheel powers.
    double frontLeftPower    = x - yaw;
    double frontRightPower   = x + yaw;
    double backLeftPower    = x - yaw;
    double backRightPower   = x + yaw;

    // Normalize wheel powers to be less than 1.0
    double max = Math.max(Math.abs(frontLeftPower), Math.abs(frontRightPower));
    if (max >1.0) {
        frontLeftPower /= max;
        frontRightPower /= max;
    }

    // Send powers to the wheels.
    frontLeftDrive.setPower(frontLeftPower);
```

```
    frontRightDrive.setPower(frontRightPower);
    backLeftDrive.setPower(backLeftPower);
    backRightDrive.setPower(backRightPower);
}

/**
 * Initialize the AprilTag processor.
 */
private void initAprilTag() {
    // Create the AprilTag processor by using a builder.
    aprilTag = new AprilTagProcessor.Builder().build();

    // Create the vision portal by using a builder.
    if (USE_WEBCAM) {
        visionPortal = new VisionPortal.Builder()
                .setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"))
                .addProcessor(aprilTag)
                .build();
    } else {
        visionPortal = new VisionPortal.Builder()
                .setCamera(BuiltinCameraDirection.BACK)
                .addProcessor(aprilTag)
                .build();
    }
}

/*
 Manually set the camera gain and exposure.
 This can only be called AFTER calling initAprilTag(), and only works for Webcams;
*/
private void    setManualExposure(int exposureMS, int gain) {
    // Wait for the camera to be open, then use the controls

    if (visionPortal == null) {
        return;
    }

    // Make sure camera is streaming before we try to set the exposure controls
    if (visionPortal.getCameraState() != VisionPortal.CameraState.STREAMING) {
        telemetry.addData("Camera", "Waiting");
```

```
            telemetry.update();
            while (!isStopRequested() && (visionPortal.getCameraState() !=
VisionPortal.CameraState.STREAMING)) {
                sleep(20);
            }
            telemetry.addData("Camera", "Ready");
            telemetry.update();
        }

        // Set camera controls unless we are stopping.
        if (!isStopRequested())
        {
            ExposureControl exposureControl =
visionPortal.getCameraControl(ExposureControl.class);
            if (exposureControl.getMode() != ExposureControl.Mode.Manual) {
                exposureControl.setMode(ExposureControl.Mode.Manual);
                sleep(50);
            }
            exposureControl.setExposure((long)exposureMS, TimeUnit.MILLISECONDS);
            sleep(20);
            GainControl gainControl = visionPortal.getCameraControl(GainControl.class);
            gainControl.setGain(gain);
            sleep(20);
            telemetry.addData("Camera", "Ready");
            telemetry.update();
        }
    }
}
```

AutoCorrectDrive.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import com.qualcomm.robotcore.util.Range;
import com.qualcomm.robotcore.hardware.IMU;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;
import com.qualcomm.robotcore.util.ElapsedTime;
import org.firstinspires.ftc.vision.VisionPortal;
```

```
import com.qualcomm.robotcore.hardware.DcMotorEx;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.Blinker;
import com.qualcomm.robotcore.hardware.HardwareDevice;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.Gyroscope;
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;

@Disabled
@Autonomous(name = "Auto Correct Drive", group = "LinearOpMode")
public class AutoCorrectDrive extends LinearOpMode {
    private DcMotor backLeftMotor;
    private DcMotor backRightMotor;
    private Blinker control_Hub;
    private Blinker expansion_Hub_2;
    private DcMotor frontLeftMotor;
    private DcMotor frontRightMotor;
    private HardwareDevice webcam_1;
    private DcMotor armMotor;
    private CRServo grabberLeft;
    private CRServo grabberRight;
    //private Gyroscope imu;
    private DcMotor launcherMotor;
    private DcMotor liftBackMotor;
    private DcMotor liftFrontMotor;

    private IMU            imu         = null;      // Control/Expansion Hub IMU
    private double         headingError  = 0;

    static final double     TURN_SPEED              = 0.2;     // Max Turn speed to limit turn rate
    static final double     HEADING_THRESHOLD       = 1.0 ;

    static final double     P_TURN_GAIN             = 0.02;     // Larger is more responsive, but also
less stable
    static final double     P_DRIVE_GAIN            = 0.03;
```

```
    private double  targetHeading = 0;
    private double  driveSpeed    = 0;
    private double  turnSpeed     = 0;
    private double  frontleftSpeed     = 0;
    private double  frontrightSpeed    = 0;
    private double  backleftSpeed     = 0;
    private double  backrightSpeed    = 0;
    private int     frontleftTarget    = 0;
    private int     frontrightTarget   = 0;
    private int     backleftTarget    = 0;
    private int     backrightTarget   = 0;


    private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

        /* Declare OpMode members. */
    private DcMotor         frontLeftDrive   = null;
    private DcMotor         frontRightDrive  = null;
    private DcMotor         backLeftDrive   = null;
    private DcMotor         backRightDrive  = null;

    //private DcMotorEx armMotor;

    static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
    static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
    static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                                  (WHEEL_DIAMETER_INCHES * 3.1415);
    static final double     DRIVE_SPEED             = 0.3;
    static final double     FAST_DRIVE_SPEED            = 0.6;
    //static final double     TURN_SPEED              = 0.3;
    static final double     SLOW_DRIVE_SPEED            = 0.1;
    static final double     SLOW_TURN_SPEED             = 0.2;
    static final double     FORWARD_SPEED               = 0.4;
```

```
    /**
     * The variable to store our instance of the TensorFlow Object Detection processor.
     */
    private TfodProcessor tfod;

    private ElapsedTime     runtime = new ElapsedTime();

    /**
     * The variable to store our instance of the vision portal.
     */
    private VisionPortal visionPortal;

    @Override
    public void runOpMode() {

        initTfod();

        RevHubOrientationOnRobot.LogoFacingDirection logoDirection =
RevHubOrientationOnRobot.LogoFacingDirection.UP;
        RevHubOrientationOnRobot.UsbFacingDirection  usbDirection  =
RevHubOrientationOnRobot.UsbFacingDirection.FORWARD;
        RevHubOrientationOnRobot orientationOnRobot = new
RevHubOrientationOnRobot(logoDirection, usbDirection);

        // Now initialize the IMU with this mounting orientation
        // This sample expects the IMU to be in a REV Hub and named "imu".
        imu = hardwareMap.get(IMU.class, "imu");
        imu.initialize(new IMU.Parameters(orientationOnRobot));

        // Wait for the DS start button to be touched.
        telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
        telemetry.addData(">", "Touch Play to start OpMode");
        telemetry.update();

            // Initialize the drive system variables.
        frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
        frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
        backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
        backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
        armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
```

```
frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
imu.resetYaw();

frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

// Send telemetry message to indicate successful Encoder reset
/*telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d :%7d :%7d",
                  frontLeftDrive.getCurrentPosition(),
                  frontRightDrive.getCurrentPosition(),
                  backLeftDrive.getCurrentPosition(),
                  backRightDrive.getCurrentPosition());
telemetry.update();*/

waitForStart();

if (opModeIsActive()) {

   telemetry.addData("Going forward",  "still going forward");
   telemetry.update();
   moveForward(SLOW_DRIVE_SPEED, 0);
   sleep(10000);/*

   strafeLeft();
   sleep(900);*/
```

```
    //encoderDrive(DRIVE_SPEED, 50, 50, 50, 50, 10.0, 0);
    //sleep(5000);

    /*while (opModeIsActive()) {


        //List<Recognition> currentRecognitions = tfod.getRecognitions();
        //telemetry.addData("# Objects Detected", currentRecognitions.size());
        if (tfod != null) {
        List<Recognition> currentRecognitions = tfod.getRecognitions();
            if (currentRecognitions != null) {
                //telemetry.addData("# Objects Detected", updatedRecognitions.size());

                for (Recognition recognition : currentRecognitions) {
                    encoderDrive(SLOW_DRIVE_SPEED, -25, 25, 25, -25, 1.0);
                }
            }
        }

        telemetryTfod();

        // Push telemetry to the Driver Station.
        telemetry.update();

        // Save CPU resources; can resume streaming when needed.
        /*if (gamepad1.dpad_down) {
            visionPortal.stopStreaming();
        } else if (gamepad1.dpad_up) {
            visionPortal.resumeStreaming();
        }

        // Share the CPU.
        sleep(20);
    }*/
}

// Save more CPU resources when camera is no longer needed.
visionPortal.close();
```

```
}   // end runOpMode()

private void strafeLeft() {
   frontLeftDrive.setPower(-FORWARD_SPEED);
   frontRightDrive.setPower(FORWARD_SPEED);
   backLeftDrive.setPower(FORWARD_SPEED);
   backRightDrive.setPower(-FORWARD_SPEED);
}

private void strafeRight() {
   frontLeftDrive.setPower(FORWARD_SPEED);
   frontRightDrive.setPower(-FORWARD_SPEED);
   backLeftDrive.setPower(-FORWARD_SPEED);
   backRightDrive.setPower(FORWARD_SPEED);
}

private void moveForward(double speed, double desiredHeading) {
   frontLeftDrive.setPower(speed);
   frontRightDrive.setPower(speed);
   backLeftDrive.setPower(speed);
   backRightDrive.setPower(speed);
   targetHeading = desiredHeading;  // Save for telemetry

   // Determine the heading current error
   headingError = targetHeading - getHeading();
   if (headingError > 5) {
      correctingMove(desiredHeading);
   }
   if (headingError < -5) {
      correctingMove(desiredHeading);
   }
}

private void goForward(double speed) {
   frontLeftDrive.setPower(speed);
   frontRightDrive.setPower(speed);
   backLeftDrive.setPower(speed);
   backRightDrive.setPower(speed);
}
```

```
private void moveBackward() {
   frontLeftDrive.setPower(-FORWARD_SPEED);
   frontRightDrive.setPower(-FORWARD_SPEED);
   backLeftDrive.setPower(-FORWARD_SPEED);
   backRightDrive.setPower(-FORWARD_SPEED);
}

private void turnLeft(double speed) {
   frontLeftDrive.setPower(-speed);
   frontRightDrive.setPower(speed);
   backLeftDrive.setPower(-speed);
   backRightDrive.setPower(speed);
}

private void turnRight(double speed) {
   frontLeftDrive.setPower(speed);
   frontRightDrive.setPower(-speed);
   backLeftDrive.setPower(speed);
   backRightDrive.setPower(-speed);
}

private void correctingMove(double desiredHeading){
   targetHeading = desiredHeading;  // Save for telemetry

   // Determine the heading current error
   headingError = targetHeading - getHeading();

   while (opModeIsActive()) {
      if (headingError > 5) {
         telemetry.addData("Going too far left",  "turning right");
         telemetry.update();
         //C: stop all motion
         frontLeftDrive.setPower(0);
         frontRightDrive.setPower(0);
         backLeftDrive.setPower(0);
         backRightDrive.setPower(0);
         sleep(1000);

         while (headingError > 5) {
```

```
            telemetry.addData("Now correcting",  "stopping & turning right");
            telemetry.update();
            turnRight(SLOW_DRIVE_SPEED);
            //sleep(1000);
        }
    }
    telemetry.addData("Going straight now",  "moving forward");
    telemetry.update();
    goForward(SLOW_DRIVE_SPEED);
    sleep(1000);

    if (headingError < -5) {

        telemetry.addData("Going too far right",  "turning left");
        telemetry.update();
        //C: stop all motion
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
        sleep(1000);

        while (headingError < -5) {
            telemetry.addData("Now correcting",  "stopping & turning right");
            telemetry.update();
            turnLeft(SLOW_DRIVE_SPEED);
        }
    }
    telemetry.addData("Going straight now",  "moving forward");
    telemetry.update();
    goForward(SLOW_DRIVE_SPEED);
    sleep(1000);
  }

}

/**
 * Initialize the TensorFlow Object Detection processor.
 */
private void initTfod() {
```

```
        // Create the TensorFlow processor by using a builder.
        tfod = new TfodProcessor.Builder()

            // Use setModelAssetName() if the TF Model is built in as an asset.
            // Use setModelFileName() if you have downloaded a custom team model to the Robot
Controller.
            //.setModelAssetName(TFOD_MODEL_ASSET)
            //.setModelFileName(TFOD_MODEL_FILE)

            //.setModelLabels(LABELS)
            //.setIsModelTensorFlow2(true)
            //.setIsModelQuantized(true)
            //.setModelInputSize(300)
            //.setModelAspectRatio(16.0 / 9.0)

            .build();

        // Create the vision portal by using a builder.
        VisionPortal.Builder builder = new VisionPortal.Builder();

        // Set the camera (webcam vs. built-in RC phone camera).
        if (USE_WEBCAM) {
            builder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
        } else {
            //builder.setCamera(BuiltinCameraDirection.BACK);
        }

        // Choose a camera resolution. Not all cameras support all resolutions.
        //builder.setCameraResolution(new Size(640, 480));

        // Enable the RC preview (LiveView).  Set "false" to omit camera monitoring.
        //builder.enableCameraMonitoring(true);

        // Set the stream format; MJPEG uses less bandwidth than default YUY2.
        //builder.setStreamFormat(VisionPortal.StreamFormat.YUY2);

        // Choose whether or not LiveView stops if no processors are enabled.
        // If set "true", monitor shows solid orange screen if no processors enabled.
        // If set "false", monitor shows camera view without annotations.
```

```
        //builder.setAutoStopLiveView(false);

        // Set and enable the processor.
        builder.addProcessor(tfod);

        // Build the Vision Portal, using the above settings.
        visionPortal = builder.build();

        // Set confidence threshold for TFOD recognitions, at any time.
        //tfod.setMinResultConfidence(0.75f);

        // Disable or re-enable the TFOD processor at any time.
        //visionPortal.setProcessorEnabled(tfod, true);

    }   // end method initTfod()

    /**
     * Add telemetry about TensorFlow Object Detection (TFOD) recognitions.
     */
    /*private void telemetryTfod() {

        List<Recognition> currentRecognitions = tfod.getRecognitions();
        telemetry.addData("# Objects Detected", currentRecognitions.size());

        // Step through the list of recognitions and display info for each one.
        for (Recognition recognition : currentRecognitions) {
            double x = (recognition.getLeft() + recognition.getRight()) / 2 ;
            double y = (recognition.getTop()  + recognition.getBottom()) / 2 ;

            telemetry.addData(""," ");
            telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(),
recognition.getConfidence() * 100);
            telemetry.addData("- Position", "%.0f / %.0f", x, y);
            telemetry.addData("- Size", "%.0f x %.0f", recognition.getWidth(),
recognition.getHeight());
        }   // end for() loop

    }   // end method telemetryTfod()*/

    public double getSteeringCorrection(double desiredHeading, double proportionalGain) {
```

```
        targetHeading = desiredHeading;  // Save for telemetry

        // Determine the heading current error
        headingError = targetHeading - getHeading();

        // Normalize the error to be within +/- 180 degrees
        //C: maybe use if statements instead to make robot turn back to original heading
        //while (headingError > 180)  headingError -= 360;
        //while (headingError <= -180) headingError += 360;
        while (headingError > 5)  headingError -= 5;
        while (headingError <= -5) headingError += 5;

        // Multiply the error by the gain to determine the required steering correction/  Limit the
result to +/- 1.0
        return Range.clip(headingError * proportionalGain, -1, 1);
    }

    public double getHeading() {
        YawPitchRollAngles orientation = imu.getRobotYawPitchRollAngles();
        return orientation.getYaw(AngleUnit.DEGREES);
    }

    public void encoderDrive(double speed,
                             double frontLeftInches, double frontRightInches,
                             double backLeftInches, double backRightInches,
                             double timeoutS, double desiredHeading) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {

            //C: Ensures that whle the drive mode is active the arm does not fall down
            //R: YOU IDIOT I CANNOT BELIEVE YOU MADE THE POWER EIGHT!??!?!?!?!
GRAHHHHHHHHHHHHHHHH
            //armMotor.setPower(0.5);

            // Determine new target position, and pass to motor controller
```

```
        newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
        newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
        newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
        newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
        frontLeftDrive.setTargetPosition(newFrontLeftTarget);
        frontRightDrive.setTargetPosition(newFrontRightTarget);
        backLeftDrive.setTargetPosition(newBackLeftTarget);
        backRightDrive.setTargetPosition(newBackRightTarget);

        // Turn On RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
            (runtime.seconds() < timeoutS) &&
            (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            correctingMove(desiredHeading);

            // Display it for the driver.
            telemetry.addData("Running to",  "  %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
                                frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
```

```
                                backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
        //armMotor.setPower(0);

        // Turn off RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(250);   // optional pause after each move.
    }
  }
}
```

AutoDriveAprilTag.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.util.Range;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.ExposureControl;
import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.GainControl;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.apriltag.AprilTagDetection;
import org.firstinspires.ftc.vision.apriltag.AprilTagProcessor;
```

```
import java.util.List;
import java.util.concurrent.TimeUnit;

@Autonomous(name="Tank Drive To AprilTag", group = "Concept")
@Disabled
public class AutoDriveAprilTag extends LinearOpMode
{
   // Adjust these numbers to suit your robot.
   final double DESIRED_DISTANCE = 12.0; //  this is how close the camera should get to the target (inches)

   //  Set the GAIN constants to control the relationship between the measured position error, and how much power is
   //  applied to the drive motors to correct the error.
   //  Drive = Error * Gain    Make these values smaller for smoother control, or larger for a more aggressive response.
   final double SPEED_GAIN =   0.02 ;   //  Speed Control "Gain". eg: Ramp up to 50% power at a 25 inch error.   (0.50 / 25.0)
   final double TURN_GAIN  =   0.01 ;   //  Turn Control "Gain".  eg: Ramp up to 25% power at a 25 degree error. (0.25 / 25.0)

   final double MAX_AUTO_SPEED = 0.5;   //  Clip the approach speed to this max value (adjust for your robot)
   final double MAX_AUTO_TURN  = 0.25;  //  Clip the turn speed to this max value (adjust for your robot)

   private DcMotor leftDrive   = null;  //  Used to control the left drive wheel
   private DcMotor rightDrive  = null;  //  Used to control the right drive wheel

   private static final boolean USE_WEBCAM = true;  // Set true to use a webcam, or false for a phone camera
   private static final int DESIRED_TAG_ID = 0;     // Choose the tag you want to approach or set to -1 for ANY tag.
   private VisionPortal visionPortal;               // Used to manage the video source.
   private AprilTagProcessor aprilTag;              // Used for managing the AprilTag detection process.
   private AprilTagDetection desiredTag = null;     // Used to hold the data for a detected AprilTag
```

```
   @Override public void runOpMode()
   {
      boolean targetFound     = false;    // Set to true when an AprilTag target is detected
      double  drive           = 0;        // Desired forward power/speed (-1 to +1) +ve is forward
      double  turn            = 0;        // Desired turning power/speed (-1 to +1) +ve is
CounterClockwise

      // Initialize the Apriltag Detection process
      initAprilTag();

      // Initialize the hardware variables. Note that the strings used here as parameters
      // to 'get' must match the names assigned during the robot configuration.
      // step (using the FTC Robot Controller app on the phone).
      leftDrive  = hardwareMap.get(DcMotor.class, "left_drive");
      rightDrive = hardwareMap.get(DcMotor.class, "right_drive");

      // To drive forward, most robots need the motor on one side to be reversed because the axles
point in opposite directions.
      // When run, this OpMode should start both motors driving forward. So adjust these two
lines based on your first test drive.
      // Note: The settings here assume direct drive on left and right wheels.  Single Gear
Reduction or 90 Deg drives may require direction flips
      leftDrive.setDirection(DcMotor.Direction.REVERSE);
      rightDrive.setDirection(DcMotor.Direction.FORWARD);

      if (USE_WEBCAM)
         setManualExposure(6, 250);  // Use low exposure time to reduce motion blur

      // Wait for the driver to press Start
      telemetry.addData("Camera preview on/off", "3 dots, Camera Stream");
      telemetry.addData(">", "Touch Play to start OpMode");
      telemetry.update();
      waitForStart();

      while (opModeIsActive())
      {
         targetFound = false;
         desiredTag  = null;

         // Step through the list of detected tags and look for a matching tag
```

```
        List<AprilTagDetection> currentDetections = aprilTag.getDetections();
        for (AprilTagDetection detection : currentDetections) {
            if ((detection.metadata != null) &&
                ((DESIRED_TAG_ID < 0) || (detection.id == DESIRED_TAG_ID))  ){
                targetFound = true;
                desiredTag = detection;
                break;  // don't look any further.
            } else {
                telemetry.addData("Unknown Target", "Tag ID %d is not in TagLibrary\n",
detection.id);
            }
        }

        // Tell the driver what we see, and what to do.
        if (targetFound) {
            telemetry.addData(">","HOLD Left-Bumper to Drive to Target\n");
            telemetry.addData("Target", "ID %d (%s)", desiredTag.id, desiredTag.metadata.name);
            telemetry.addData("Range",  "%5.1f inches", desiredTag.ftcPose.range);
            telemetry.addData("Bearing","%3.0f degrees", desiredTag.ftcPose.bearing);
        } else {
            telemetry.addData(">","Drive using joysticks to find valid target\n");
        }

        // If Left Bumper is being pressed, AND we have found the desired target, Drive to target
Automatically .
        if (gamepad1.left_bumper && targetFound) {

            // Determine heading and range error so we can use them to control the robot
automatically.
            double  rangeError   = (desiredTag.ftcPose.range - DESIRED_DISTANCE);
            double  headingError = desiredTag.ftcPose.bearing;

            // Use the speed and turn "gains" to calculate how we want the robot to move.  Clip it
to the maximum
            drive = Range.clip(rangeError * SPEED_GAIN, -MAX_AUTO_SPEED,
MAX_AUTO_SPEED);
            turn  = Range.clip(headingError * TURN_GAIN, -MAX_AUTO_TURN,
MAX_AUTO_TURN) ;

            telemetry.addData("Auto","Drive %5.2f, Turn %5.2f", drive, turn);
```

```
        } else {

            // drive using manual POV Joystick mode.
            drive = -gamepad1.left_stick_y  / 2.0;  // Reduce drive rate to 50%.
            turn  = -gamepad1.right_stick_x / 4.0;  // Reduce turn rate to 25%.
            telemetry.addData("Manual","Drive %5.2f, Turn %5.2f", drive, turn);
        }
        telemetry.update();

        // Apply desired axes motions to the drivetrain.
        moveRobot(drive, turn);
        sleep(10);
    }
}

/**
 * Move robot according to desired axes motions
 * <p>
 * Positive X is forward
 * <p>
 * Positive Yaw is counter-clockwise
 */
public void moveRobot(double x, double yaw) {
    // Calculate left and right wheel powers.
    double leftPower    = x - yaw;
    double rightPower   = x + yaw;

    // Normalize wheel powers to be less than 1.0
    double max = Math.max(Math.abs(leftPower), Math.abs(rightPower));
    if (max >1.0) {
        leftPower /= max;
        rightPower /= max;
    }

    // Send powers to the wheels.
    leftDrive.setPower(leftPower);
    rightDrive.setPower(rightPower);
}

/**
```

```
     * Initialize the AprilTag processor.
     */
    private void initAprilTag() {
        // Create the AprilTag processor by using a builder.
        aprilTag = new AprilTagProcessor.Builder().build();

        // Create the vision portal by using a builder.
        if (USE_WEBCAM) {
            visionPortal = new VisionPortal.Builder()
                    .setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"))
                    .addProcessor(aprilTag)
                    .build();
        } else {
            visionPortal = new VisionPortal.Builder()
                    .setCamera(BuiltinCameraDirection.BACK)
                    .addProcessor(aprilTag)
                    .build();
        }
    }

    /*
     Manually set the camera gain and exposure.
     This can only be called AFTER calling initAprilTag(), and only works for Webcams;
    */
    private void    setManualExposure(int exposureMS, int gain) {
        // Wait for the camera to be open, then use the controls

        if (visionPortal == null) {
            return;
        }

        // Make sure camera is streaming before we try to set the exposure controls
        if (visionPortal.getCameraState() != VisionPortal.CameraState.STREAMING) {
            telemetry.addData("Camera", "Waiting");
            telemetry.update();
            while (!isStopRequested() && (visionPortal.getCameraState() !=
VisionPortal.CameraState.STREAMING)) {
                sleep(20);
            }
            telemetry.addData("Camera", "Ready");
```

```
            telemetry.update();
        }

        // Set camera controls unless we are stopping.
        if (!isStopRequested())
        {
            ExposureControl exposureControl =
visionPortal.getCameraControl(ExposureControl.class);
            if (exposureControl.getMode() != ExposureControl.Mode.Manual) {
                exposureControl.setMode(ExposureControl.Mode.Manual);
                sleep(50);
            }
            exposureControl.setExposure((long)exposureMS, TimeUnit.MILLISECONDS);
            sleep(20);
            GainControl gainControl = visionPortal.getCameraControl(GainControl.class);
            gainControl.setGain(gain);
            sleep(20);
            telemetry.addData("Camera", "Ready");
            telemetry.update();
        }
    }
}
```

AutoScorer.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.util.ElapsedTime;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;
```

```
import java.util.List;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;

@Autonomous(name = "DONT USE YET", group = "LinearOpMode")
@Disabled
public class AutoScorer extends LinearOpMode{


   //private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

   boolean USE_WEBCAM;
   TfodProcessor myTfodProcessor;
   VisionPortal myVisionPortal;


     //boolean USE_WEBCAM;
   //TfodProcessor myTfodProcessor;
   //VisionPortal myVisionPortal;

         /* Declare OpMode members. */
   private DcMotor frontLeftDrive   = null;
   private DcMotor frontRightDrive  = null;
   private DcMotor backLeftDrive   = null;
   private DcMotor backRightDrive  = null;
   private CRServo grabberLeft;
   private CRServo grabberRight;
  // private DcMotor armMotor;

   private ElapsedTime     runtime = new ElapsedTime();

   private DcMotorEx armMotor;

   static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
   static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
   static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
   static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
```

```
                                    (WHEEL_DIAMETER_INCHES * 3.1415);
    static final double     DRIVE_SPEED             = 0.4;
    static final double     FAST_DRIVE_SPEED        = 0.6;
    static final double     TURN_SPEED              = 0.3;
    static final double     SLOW_DRIVE_SPEED        = 0.1;
    static final double     SLOW_TURN_SPEED         = 0.2;
    static final double     FORWARD_SPEED           = 0.4;

    private static final String TFOD_MODEL_FILE  = "Original Modell";

    private static final String[] LABEL = {
        "Blue Warrior",
        "Red Warrior"
    };

    //static final double TURN_SPEED = 0.5;

   // private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

    /**
     * The variable to store our instance of the TensorFlow Object Detection processor.
     */
    private TfodProcessor tfod;

    /**
     * The variable to store our instance of the vision portal.
     */
    private VisionPortal visionPortal;

    @Override
    public void runOpMode() {

        USE_WEBCAM = true;
        initTfod();

        // Wait for the DS start button to be touched.
        telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
        telemetry.addData(">", "Touch Play to start OpMode");
        telemetry.update();
```

```
        // Initialize the drive system variables.
frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
grabberLeft = hardwareMap.get(CRServo.class,"grabberLeft");
grabberRight = hardwareMap.get(CRServo.class, "grabberRight");

frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
grabberLeft.setDirection(DcMotor.Direction.FORWARD);
grabberRight.setDirection(DcMotor.Direction.FORWARD);

// Send telemetry message to indicate successful Encoder reset
telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d :%7d :%7d",
                  frontLeftDrive.getCurrentPosition(),
                  frontRightDrive.getCurrentPosition(),
                  backLeftDrive.getCurrentPosition(),
                  backRightDrive.getCurrentPosition());
telemetry.update();

waitForStart();
```

```
        if (opModeIsActive()) {

            initTfod();

            wholeStep();

        }
    }   // end runOpMode()

    //C: this is basically the whole program/ Robot searches for the team prop and then takes
action
    private void wholeStep() {
        //C: drive forward off the wall for 0.5 seconds for the first position

        while (opModeIsActive()) {

        }

        encoderDrive(SLOW_DRIVE_SPEED, 12, 12, 12, 12, 15.0);

        firstStep();
    }

    private void firstStep() {

        encoderDrive(SLOW_DRIVE_SPEED, 7, -7, 7, -7, 10.0);

        List<Recognition> currentRecognitions = tfod.getRecognitions();
        telemetry.addData("# Objects Detected", currentRecognitions.size());

        // Step through the list of recognitions and display info for each one.
        for (Recognition recognition : currentRecognitions) {
            if (recognition.getLabel().equals("Blue Warrior") ){
                telemetry.addData("OMG!! Object is finally detected", "Blue boy");
                telemetry.update();

                firstPath();

            } else if (recognition.getLabel().equals("Red Warrior")) {
                telemetry.addData("OMG!! Object is finally detected", "Red boy");
```

```
            telemetry.update();

            fourthPath();

        } else {

            secondStep();

        }
    }
}

private void secondStep() {

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, -5, 5, 10.0);

    List<Recognition> currentRecognitions = tfod.getRecognitions();
    telemetry.addData("# Objects Detected", currentRecognitions.size());

    // Step through the list of recognitions and
    for (Recognition recognition : currentRecognitions) {
        if (recognition.getLabel().equals("Blue Warrior") ){
            telemetry.addData("OMG!! Object is finally detected", "Blue boy");
            telemetry.update();

            secondPath();

        } else if (recognition.getLabel().equals("Red Warrior") ) {
            telemetry.addData("OMG!! Object is finally detected", "Red boy");
            telemetry.update();

            fifthPath();

        } else {

            thirdStep();

        }
    }
}
```

```
private void thirdStep() {
   encoderDrive(SLOW_DRIVE_SPEED, -6, 6, -6, 6, 10.0);

   List<Recognition> currentRecognitions = tfod.getRecognitions();
   telemetry.addData("# Objects Detected", currentRecognitions.size());

   // Step through the list of recognitions and display info for each one.
   for (Recognition recognition : currentRecognitions) {
      if (recognition.getLabel().equals("Blue Warrior") ){
         telemetry.addData("OMG!! Object is finally detected", "Blue boy");
         telemetry.update();

         thirdPath();

      } else if (recognition.getLabel().equals("Red Warrior") ){
         telemetry.addData("OMG!! Object is finally detected", "Red boy");
         telemetry.update();

         sixthPath();

      } else {
         telemetry.addData("Uh oh", "nothing here");
         telemetry.update();
      }
   }
}

public void firstPath() {

   encoderDrive(SLOW_DRIVE_SPEED, -5, 5, -5, 5, 10.0);

   encoderDrive(SLOW_DRIVE_SPEED, -6, 6, -6, 6, 10.0);

   encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

   encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 5.0);

   encoderDrive(SLOW_DRIVE_SPEED, -11, -11, -11, -11, 15.0);
```

```
    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, 15, -15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -20, 20, 20, -20, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -4, 4, 4, -4, 8.0);

        //encoderDrive(SLOW_DRIVE_SPEED, -12, 12, 12, -12, 12.0);

    encoderDrive(SLOW_DRIVE_SPEED, -7, 7, 7, -7, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 5.0);


    encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);

    armMotor.setPower(FORWARD_SPEED);
    sleep(1000);

    pixelScoOne();

    //armMotor.setPower(-FORWARD_SPEED);
    //sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, 30, -30, -30, 30, 20.0);

}

public void secondPath() {
    //C: Drives forward 13.5 seconds

    encoderDrive(SLOW_DRIVE_SPEED, -6, 6, -6, 6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 5.0);
```

```
        //encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);


    encoderDrive(SLOW_DRIVE_SPEED, -11, -11, -11, -11, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, 15, -15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -20, 20, 20, -20, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -4, 4, 4, -4, 8.0);

        //encoderDrive(SLOW_DRIVE_SPEED, -12, 12, 12, -12, 12.0);

    encoderDrive(SLOW_DRIVE_SPEED, -7, 7, 7, -7, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 5.0);


    encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);

    armMotor.setPower(FORWARD_SPEED);
    sleep(1000);
        //armMotor.setPower(0);

    pixelScoTwo();

    //armMotor.setPower(-FORWARD_SPEED);
    //sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, 20, -20, -20, 20, 20.0);

}//C: end of the second path

//C: The third path goes to the third april tag code
public void thirdPath() {
```

```
    encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 5.0);

       //encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);


    encoderDrive(SLOW_DRIVE_SPEED, -11, -11, -11, -11, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, 15, -15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -20, 20, 20, -20, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

       //encoderDrive(SLOW_DRIVE_SPEED, -4, 4, 4, -4, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, 12, -12, 12.0);

    encoderDrive(SLOW_DRIVE_SPEED, -7, 7, 7, -7, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 5.0);


    encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);

    armMotor.setPower(FORWARD_SPEED);
    sleep(1000);
       //armMotor.setPower(0);

    pixelScoThree();

    //armMotor.setPower(-FORWARD_SPEED);
    //sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, 16, -16, -16, 16, 16.0);

}//C: end of the third path
```

```
//C: the fourth scores the fourth april tag
public void fourthPath() {

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, -5, 5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -6, 6, -6, 6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 5.0);

    //encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -11, -11, -11, -11, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 20, -20, -20, 20, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, 15, -15, 15.0);

        //encoderDrive(SLOW_DRIVE_SPEED, -4, 4, 4, -4, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, 12, -12, -12, 12, 12.0);

    encoderDrive(SLOW_DRIVE_SPEED, 7, -7, -7, 7, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 5.0);


    encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);

    armMotor.setPower(FORWARD_SPEED);
    sleep(1000);

    pixelScoOne();
```

```
    encoderDrive(SLOW_DRIVE_SPEED, -16, 16, -16, 16, 16.0);

}//C: fourth path ended

//C: the fifth path scores the fifth april tag
public void fifthPath() {

    encoderDrive(SLOW_DRIVE_SPEED, -6, 6, -6, 6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 5.0);


    //encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);


    encoderDrive(SLOW_DRIVE_SPEED, -11, -11, -11, -11, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 20, -20, -20, 20, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, 15, -15, 15.0);

        //encoderDrive(SLOW_DRIVE_SPEED, -4, 4, 4, -4, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, 12, -12, -12, 12, 12.0);

    //encoderDrive(SLOW_DRIVE_SPEED, 7, -7, -7, 7, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 5.0);


    encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);

    armMotor.setPower(FORWARD_SPEED);
    sleep(1000);
```

```
    pixelScoTwo();
}//C: fifth path ended
public void sixthPath() {
    encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);
    encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 5.0);

    //encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -11, -11, -11, -11, 15.0);
    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, -15, 15, 15.0);
    encoderDrive(SLOW_DRIVE_SPEED, 20, -20, -20, 20, 20.0);
    encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);
    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, 15, -15, 15.0);
       //encoderDrive(SLOW_DRIVE_SPEED, -4, 4, 4, -4, 8.0);
    //encoderDrive(SLOW_DRIVE_SPEED, 12, -12, -12, 12, 12.0);
    //encoderDrive(SLOW_DRIVE_SPEED, 7, -7, -7, 7, 8.0);
    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);
    armMotor.setPower(FORWARD_SPEED);
    sleep(1000);
    pixelScoThree();
```

```
    }

    private void initTfod() {
        TfodProcessor.Builder myTfodProcessorBuilder;
        VisionPortal.Builder myVisionPortalBuilder;

        // First, create a TfodProcessor.Builder.
        myTfodProcessorBuilder = new TfodProcessor.Builder();
        // Set the name of the file where the model can be found.
        myTfodProcessorBuilder.setModelFileName("Original Modell");
        // Set the full ordered list of labels the model is trained to recognize.
        myTfodProcessorBuilder.setModelLabels(JavaUtil.createListWith("Blue Warrior", "Red
Warrior"));
        // Set the aspect ratio for the images used when the model was created.
        myTfodProcessorBuilder.setModelAspectRatio(16 / 9);
        // Create a TfodProcessor by calling build.
        myTfodProcessor = myTfodProcessorBuilder.build();
        // Next, create a VisionPortal.Builder and set attributes related to the camera.
        myVisionPortalBuilder = new VisionPortal.Builder();
        if (USE_WEBCAM) {
          // Use a webcam.
          myVisionPortalBuilder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
        } else {
            // Use the device's back camera.
            myVisionPortalBuilder.setCamera(BuiltinCameraDirection.BACK);
        }
        // Add myTfodProcessor to the VisionPortal.Builder.
        myVisionPortalBuilder.addProcessor(myTfodProcessor);
        // Create a VisionPortal by calling build.
        myVisionPortal = myVisionPortalBuilder.build();
    }

    private void telemetryTfod() {
    List<Recognition> myTfodRecognitions;
    Recognition myTfodRecognition;
    float x;
    float y;

    // Get a list of recognitions from TFOD.
    myTfodRecognitions = myTfodProcessor.getRecognitions();
```

```
    telemetry.addData("# Objects Detected", JavaUtil.listLength(myTfodRecognitions));
    // Iterate through list and call a function to display info for each recognized object.
    for (Recognition myTfodRecognition_item : myTfodRecognitions) {
      myTfodRecognition = myTfodRecognition_item;
      // Display info about the recognition.
      telemetry.addLine("");
      // Display label and confidence.
      // Display the label and confidence for the recognition.
      telemetry.addData("Image", myTfodRecognition.getLabel() + " (" +
JavaUtil.formatNumber(myTfodRecognition.getConfidence() * 100, 0) + " % Conf.)");
      // Display position.
      x = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;
      y = (myTfodRecognition.getTop() + myTfodRecognition.getBottom()) / 2;
      // Display the position of the center of the detection boundary for the recognition
      telemetry.addData("- Position", JavaUtil.formatNumber(x, 0) + ", " +
JavaUtil.formatNumber(y, 0));
      // Display size
      // Display the size of detection boundary for the recognition
      telemetry.addData("- Size", JavaUtil.formatNumber(myTfodRecognition.getWidth(), 0) + " x
" + JavaUtil.formatNumber(myTfodRecognition.getHeight(), 0));
    }
  }

    public void pixelScoOne() {
        grabberLeft.setPower(-1);
        sleep(1500);
        grabberRight.setPower(1);
        sleep(1000);

        armMotor.setPower(-FORWARD_SPEED);
        sleep(1000);

        grabberLeft.setPower(1);
        sleep(1500);
        grabberRight.setPower(-1);
        sleep(1000);
    }

    public void pixelScoTwo() {
        grabberLeft.setPower(-1);
```

```
    grabberRight.setPower(1);
    sleep(1000);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);

    grabberLeft.setPower(1);
    grabberRight.setPower(-1);
    sleep(1000);
}

public void pixelScoThree() {
    grabberLeft.setPower(-1);
    sleep(1000);
    grabberRight.setPower(1);
    sleep(1500);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);

    grabberLeft.setPower(1);
    sleep(1000);
    grabberRight.setPower(-1);
    sleep(1500);
}

//function for grabber opening and closing
public void  pixelOpen(){
    grabberLeft.setPower(-1);
    grabberRight.setPower(1);
    sleep(1000);
    grabberLeft.setPower(1);
    grabberRight.setPower(-1);
    sleep(1000);
}

public void  pixelClose(){
    grabberLeft.setPower(1);
    grabberRight.setPower(-1);
    sleep(1000);
```

```
        grabberLeft.setPower(-1);
        grabberRight.setPower(1);
        sleep(1000);
    }

    public void encoderDrive(double speed,
                             double frontLeftInches, double frontRightInches,
                             double backLeftInches, double backRightInches,
                             double timeouts) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {

            //C: Ensures that whle the drive mode is active the arm does not fall down
            //R: YOU IDIOT I CANNOT BELIEVE YOU MADE THE POWER EIGHT!??!?!?!?!
GRAHHHHHHHHHHHHHHHH
            //armMotor.setPower(0.5);

            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
            frontLeftDrive.setTargetPosition(newFrontLeftTarget);
            frontRightDrive.setTargetPosition(newFrontRightTarget);
            backLeftDrive.setTargetPosition(newBackLeftTarget);
            backRightDrive.setTargetPosition(newBackRightTarget);

            // Turn On RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
```

```
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
             (runtime.seconds() < timeouts) &&
             (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  "  %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
                                frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
                                backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
        //armMotor.setPower(0);

        // Turn off RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(250);   // optional pause after each move.
    }
```

```
    }

}
```

BackDropScorer.java

```
package Cam_auto_2023;

//import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;

import java.util.List;

@Autonomous(name = "Back Drop Scorer", group = "LinearOpMode")
@Disabled
public class BackDropScorer extends LinearOpMode {

    private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

        /* Declare OpMode members. */
    private DcMotor         frontLeftDrive   = null;
    private DcMotor         frontRightDrive  = null;
    private DcMotor         backLeftDrive    = null;
    private DcMotor         backRightDrive   = null;

    private DcMotorEx armMotor;
```

```
   static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
   static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
   static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
   static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                             (WHEEL_DIAMETER_INCHES * 3.1415);
   static final double     DRIVE_SPEED             = 0.4;
   static final double     FAST_DRIVE_SPEED            = 0.6;
   static final double     TURN_SPEED              = 0.3;
   static final double     SLOW_DRIVE_SPEED            = 0.2;
   static final double     SLOW_TURN_SPEED             = 0.2;
   static final double FORWARD_SPEED = 0.4;
   //static final double TURN_SPEED = 0.5;

   /**
    * The variable to store our instance of the TensorFlow Object Detection processor.
    */
   private TfodProcessor tfod;

   private ElapsedTime     runtime = new ElapsedTime();

   /**
    * The variable to store our instance of the vision portal.
    */
   private VisionPortal visionPortal;

   @Override
   public void runOpMode() {

      initTfod();

      // Wait for the DS start button to be touched.
      telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
      telemetry.addData(">", "Touch Play to start OpMode");
      telemetry.update();

            // Initialize the drive system variables.
      frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
```

```
frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");

frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

// Send telemetry message to indicate successful Encoder reset
telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d :%7d :%7d",
                  frontLeftDrive.getCurrentPosition(),
                  frontRightDrive.getCurrentPosition(),
                  backLeftDrive.getCurrentPosition(),
                  backRightDrive.getCurrentPosition());
telemetry.update();

waitForStart();

if (opModeIsActive()) {

   //C: BB strafes to the left
   //encoderDrive(DRIVE_SPEED, -25, 30, 25, -25, 10.0);

   moveBackward(SLOW_DRIVE_SPEED);
   sleep(200);
```

```
        turnRight(SLOW_DRIVE_SPEED);
        sleep(200);


        //C: BB backs up anddrops pixel
        backDropper();

        //C: BB strafes to the right
        //encoderDrive(DRIVE_SPEED, 30, -35, -30, 30, 10.0);
        strafeRight(SLOW_DRIVE_SPEED);
        sleep(900);

        //C: BB backs up
        //encoderDrive(DRIVE_SPEED, -25, -30, -25, -25, 10.0);
        moveBackward(SLOW_DRIVE_SPEED);
        sleep(900);

        /*moveForward();
        sleep(100);

        strafeLeft();
        sleep(900);*/

        //encoderDrive(SLOW_DRIVE_SPEED, -25, 25, 25, -25, 10.0);
        //sleep(5000);

        /*while (opModeIsActive()) {

        }*/
    }

    // Save more CPU resources when camera is no longer needed.
    visionPortal.close();

}   // end runOpMode()

//C: BB strafes to the left
private void strafeLeft(double speed) {
    frontLeftDrive.setPower(-FORWARD_SPEED);
```

```
    frontRightDrive.setPower(FORWARD_SPEED);
    backLeftDrive.setPower(FORWARD_SPEED);
    backRightDrive.setPower(-FORWARD_SPEED);
}

//C: BB strafes to the right
private void strafeRight(double speed) {
    frontLeftDrive.setPower(FORWARD_SPEED);
    frontRightDrive.setPower(-FORWARD_SPEED);
    backLeftDrive.setPower(-FORWARD_SPEED);
    backRightDrive.setPower(FORWARD_SPEED);
}

//C: BB moves forward
private void moveForward(double speed) {
    frontLeftDrive.setPower(FORWARD_SPEED);
    frontRightDrive.setPower(FORWARD_SPEED);
    backLeftDrive.setPower(FORWARD_SPEED);
    backRightDrive.setPower(FORWARD_SPEED);
}

//C: BB moves backwards
private void moveBackward(double speed) {
    frontLeftDrive.setPower(-FORWARD_SPEED);
    frontRightDrive.setPower(-FORWARD_SPEED);
    backLeftDrive.setPower(-FORWARD_SPEED);
    backRightDrive.setPower(-FORWARD_SPEED);
}

//C: BB turns to the left
private void turnLeft(double speed) {
    frontLeftDrive.setPower(-FORWARD_SPEED);
    frontRightDrive.setPower(FORWARD_SPEED);
    backLeftDrive.setPower(-FORWARD_SPEED);
    backRightDrive.setPower(FORWARD_SPEED);
}

//C: BB Turns to the right
private void turnRight(double speed) {
    frontLeftDrive.setPower(FORWARD_SPEED);
```

```
        frontRightDrive.setPower(-FORWARD_SPEED);
        backLeftDrive.setPower(FORWARD_SPEED);
        backRightDrive.setPower(-FORWARD_SPEED);
    }

    //C: BB moves back and uses arm to drop a pixel on back drop
    private void backDropper() {
        //encoderDrive(DRIVE_SPEED, -25, -25, -25, -25, 10.0);
        moveBackward(SLOW_DRIVE_SPEED);
        sleep(900);
        armMotor.setPower(1);
        sleep(900);
        armMotor.setPower(-1);
        sleep(900);
    }

    /**
     * Initialize the TensorFlow Object Detection processor.
     */
    private void initTfod() {

        // Create the TensorFlow processor by using a builder.
        tfod = new TfodProcessor.Builder()

            // Use setModelAssetName() if the TF Model is built in as an asset.
            // Use setModelFileName() if you have downloaded a custom team model to the Robot
Controller.
            //.setModelAssetName(TFOD_MODEL_ASSET)
            //.setModelFileName(TFOD_MODEL_FILE)

            //.setModelLabels(LABELS)
            //.setIsModelTensorFlow2(true)
            //.setIsModelQuantized(true)
            //.setModelInputSize(300)
            //.setModelAspectRatio(16.0 / 9.0)

            .build();

        // Create the vision portal by using a builder.
        VisionPortal.Builder builder = new VisionPortal.Builder();
```

```
    // Set the camera (webcam vs. built-in RC phone camera).
    if (USE_WEBCAM) {
        builder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
    } else {
        builder.setCamera(BuiltinCameraDirection.BACK);
    }

    // Choose a camera resolution. Not all cameras support all resolutions.
    //builder.setCameraResolution(new Size(640, 480));

    // Enable the RC preview (LiveView).  Set "false" to omit camera monitoring.
    //builder.enableCameraMonitoring(true);

    // Set the stream format; MJPEG uses less bandwidth than default YUY2.
    //builder.setStreamFormat(VisionPortal.StreamFormat.YUY2);

    // Choose whether or not LiveView stops if no processors are enabled.
    // If set "true", monitor shows solid orange screen if no processors enabled.
    // If set "false", monitor shows camera view without annotations.
    //builder.setAutoStopLiveView(false);

    // Set and enable the processor.
    builder.addProcessor(tfod);

    // Build the Vision Portal, using the above settings.
    visionPortal = builder.build();

    // Set confidence threshold for TFOD recognitions, at any time.
    //tfod.setMinResultConfidence(0.75f);

    // Disable or re-enable the TFOD processor at any time.
    //visionPortal.setProcessorEnabled(tfod, true);

}   // end method initTfod()

/**
 * Add telemetry about TensorFlow Object Detection (TFOD) recognitions.
 */
private void telemetryTfod() {
```

```
        List<Recognition> currentRecognitions = tfod.getRecognitions();
        telemetry.addData("# Objects Detected", currentRecognitions.size());

        // Step through the list of recognitions and display info for each one.
        for (Recognition recognition : currentRecognitions) {
            double x = (recognition.getLeft() + recognition.getRight()) / 2 ;
            double y = (recognition.getTop()  + recognition.getBottom()) / 2 ;

            telemetry.addData(""," ");
            telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(),
recognition.getConfidence() * 100);
            telemetry.addData("- Position", "%.0f / %.0f", x, y);
            telemetry.addData("- Size", "%.0f x %.0f", recognition.getWidth(),
recognition.getHeight());
        }   // end for() loop

    }   // end method telemetryTfod()

    public void encoderDrive(double speed,
                             double frontLeftInches, double frontRightInches,
                             double backLeftInches, double backRightInches,
                             double timeoutS) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {

            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
```

```
        frontLeftDrive.setTargetPosition(newFrontLeftTarget);
        frontRightDrive.setTargetPosition(newFrontRightTarget);
        backLeftDrive.setTargetPosition(newBackLeftTarget);
        backRightDrive.setTargetPosition(newBackRightTarget);

        // Turn On RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
             (runtime.seconds() < timeoutS) &&
             (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  "  %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
                                frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
                                backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
        //armMotor.setPower(0);
```

```
            // Turn off RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

            sleep(250);   // optional pause after each move.
        }
    }
}   // end class
```

BlueVisAutoClose.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.hardware.Blinker;
import com.qualcomm.robotcore.hardware.HardwareDevice;
import com.qualcomm.robotcore.hardware.Gyroscope;
import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import com.qualcomm.robotcore.hardware.IMU;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;
import java.util.List;
```

```
//@Disabled
@Autonomous(name = "Blue Vision Close Side", group = "LinearOpMode")
public class BlueVisAutoClose extends LinearOpMode{
//C: Binary variable added for the Blue Team Prop.
   boolean isBlueWarriorDetected;

   //C: Binary variable added to activate the web cam when set to true.
   boolean USE_WEBCAM;

   //C: Activating vision and setting parameters.
   TfodProcessor myTfodProcessor;
   VisionPortal myVisionPortal;

   //C: Declaring motors and servos.
   private DcMotor frontLeftDrive   = null;
   private DcMotor frontRightDrive  = null;
   private DcMotor backLeftDrive   = null;
   private DcMotor backRightDrive  = null;

   private CRServo grabberLeft;
   private CRServo grabberRight;
   private DcMotorEx armMotor;
   private CRServo pixelDrop;

   //C: OpMode runtime added.
   private ElapsedTime     runtime = new ElapsedTime();

   //C: This is the math involved in converting revolutions to inches.
   static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
   static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
   static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
   static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                             (WHEEL_DIAMETER_INCHES * 3.1415);

   //C: The speeds the robot travels at when set.
   static final double     DRIVE_SPEED             = 0.2;
```

```
static final double     FAST_DRIVE_SPEED       = 0.4;
static final double     TURN_SPEED             = 0.2;
static final double     SLOW_DRIVE_SPEED       = 0.1;
static final double     SLOW_TURN_SPEED        = 0.2;
static final double     FORWARD_SPEED          = 0.4;
static final double     HEADING_THRESHOLD      = 1.0 ;

private static final String TFOD_MODEL_FILE  = "Prop_Orbs";

private static final String[] LABEL = {
   "Blue Orbs",
   "Red Orbs"
};


//C: The variable to store the TensorFlow Object Detection processor.
private TfodProcessor tfod;

//C: The variable to store our instance of the vision portal.
private VisionPortal visionPortal;

@Override
public void runOpMode() {

   //C: Web cam variable is set to true activating it.
   USE_WEBCAM = true;

   initTfod();

   //C: Wait for the Control Hub start button to be activated.
   telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
   telemetry.addData(">", "Touch Play to start OpMode");
   telemetry.update();

   //C: Starting the drive and other system variables.
   frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
   frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
   backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
   backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
   armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
```

```
grabberLeft = hardwareMap.get(CRServo.class,"grabberLeft");
grabberRight = hardwareMap.get(CRServo.class, "grabberRight");
pixelDrop = hardwareMap.get(CRServo.class, "pixelDrop");

//C: Setting the direction of the motors to go reverse.
frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

//C: Setting the run mode to stop and reset encoders after movement.
frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

//C: Setting the run mode to move by encoders.
frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

//C: Setting the arm motor to move forward and by encoders.
armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

//C: Setting the grabber and the pixel dropper servos to move forward.
grabberLeft.setDirection(DcMotor.Direction.FORWARD);
grabberRight.setDirection(DcMotor.Direction.FORWARD);
pixelDrop.setDirection(DcMotor.Direction.FORWARD);

//C: Sends message to reveal a successful encoder reset.
telemetry.addData("Yass! Oki, Starting at",  "%7d :%7d :%7d :%7d",
    frontLeftDrive.getCurrentPosition(),
    frontRightDrive.getCurrentPosition(),
    backLeftDrive.getCurrentPosition(),
    backRightDrive.getCurrentPosition());
telemetry.update();

//C: Sounds like the name. Waiting for the opMode to begin.
```

```
waitForStart();

if (opModeIsActive()) {

   encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 15.0);

   while (opModeIsActive()) {

   List<Recognition> myTfodRecognitions;
   Recognition myTfodRecognition;

   // Get a list of recognitions from TFOD.
   myTfodRecognitions = myTfodProcessor.getRecognitions();

   for (Recognition myTfodRecognition_item : myTfodRecognitions) {
      myTfodRecognition = myTfodRecognition_item;
      float prop;

      prop = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;

      if (prop >= 330){
         telemetry.addData("OMG!! Object is finally detected", "On the right");
         telemetry.update();
         sleep(1000);

         USE_WEBCAM = false;

         thirdPath();
       }

      if (prop < 330 && prop > 210){
         telemetry.addData("OMG!! Object is finally detected", "In the center");
         telemetry.update();
         sleep(1000);

         USE_WEBCAM = false;

         secondPath();
       }
```

```
            if (prop <= 210){
                telemetry.addData("OMG!! Object is finally detected", "On the left");
                telemetry.update();
                sleep(1000);

                USE_WEBCAM = false;

                firstPath();
            }
        }

        }
    }
}

public void firstPath() {

    //C: Robot moves forward 22 inches
    //encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 12, 12, 12, 12, 8.0);

    //C: Robot strafes to the right and drops pixel and strafes back
    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 14, -14, 14, -14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, 3, 3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 22, 22, 22, 22, 20.0);
```

```
    //C: 1
    encoderDrive(SLOW_DRIVE_SPEED, -8, 8, 8, -8, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, -2, -2, -2, 10.0);

    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);
}

public void secondPath() {
    //C: Robot turns right for 7 inches and
    //encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 12, 12, 12, 12, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -14, 14, -14, 14, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 3, 3, 3, 3, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, 14, -14, 14, -14, 15.0);

    //encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);
```

```
    //encoderDrive(SLOW_DRIVE_SPEED, -2, -2, -2, -2, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, 14, -14, 14, -14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 22, 22, 22, 22, 20.0);

    //C: 2
    //encoderDrive(SLOW_DRIVE_SPEED, -8, 8, 8, -8, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);

}//C: end of the second path

//C: The third path goes to the third april tag code
public void thirdPath() {

    //C: Robot turns right for 7 inches and
    //encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 12, 12, 12, 12, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
```

```
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, 15, -15, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 14, -14, 14, -14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 22, 22, 22, 22, 20.0);

    //C: 3
    encoderDrive(SLOW_DRIVE_SPEED, 8, -8, -8, 8, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

    //encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);

}//C: end of the third path

private void initTfod() {

    TfodProcessor.Builder myTfodProcessorBuilder;
    VisionPortal.Builder myVisionPortalBuilder;

    //C: Created a TfodProcessor.Builder.
    myTfodProcessorBuilder = new TfodProcessor.Builder();

    //C: The model file name "Original Modell" is added.
    myTfodProcessorBuilder.setModelFileName("Prop_Orbs");
    //Original Modell");
```

```
        // Set the full ordered list of labels the model is trained to recognize.
        myTfodProcessorBuilder.setModelLabels(JavaUtil.createListWith("Blue Orbs", "Red
Orbs"));
        //C: The aspect ratio is set to a horizontal postion (due to the higher being higher than the
bottom number)
        myTfodProcessorBuilder.setModelAspectRatio(16 / 9);
        // Create a TfodProcessor by calling build.
        myTfodProcessor = myTfodProcessorBuilder.build();
        // Next, create a VisionPortal.Builder and set attributes related to the camera.
        myVisionPortalBuilder = new VisionPortal.Builder();

        if (USE_WEBCAM) {
          // Use a webcam.
          myVisionPortalBuilder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
        } else {
           // Use the device's back camera.
           myVisionPortalBuilder.setCamera(BuiltinCameraDirection.BACK);
        }
        // Add myTfodProcessor to the VisionPortal.Builder.
        myVisionPortalBuilder.addProcessor(myTfodProcessor);
        // Create a VisionPortal by calling build.
        myVisionPortal = myVisionPortalBuilder.build();
    }

    private void telemetryTfod() {
        List<Recognition> myTfodRecognitions;
        Recognition myTfodRecognition;
        float x;
        float y;

        // Get a list of recognitions from TFOD.
        myTfodRecognitions = myTfodProcessor.getRecognitions();
        telemetry.addData("# Objects Detected", JavaUtil.listLength(myTfodRecognitions));
        // Iterate through list and call a function to display info for each recognized object.
        for (Recognition myTfodRecognition_item : myTfodRecognitions) {
        myTfodRecognition = myTfodRecognition_item;
      // Display info about the recognition.
      telemetry.addLine("");
      // Display label and confidence.
```

```
    // Display the label and confidence for the recognition.
    telemetry.addData("Image", myTfodRecognition.getLabel() + " (" +
JavaUtil.formatNumber(myTfodRecognition.getConfidence() * 100, 0) + " % Conf.)");
    // Display position.
    x = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;
    y = (myTfodRecognition.getTop() + myTfodRecognition.getBottom()) / 2;
    // Display the position of the center of the detection boundary for the recognition
    telemetry.addData("- Position", JavaUtil.formatNumber(x, 0) + ", " +
JavaUtil.formatNumber(y, 0));
    // Display size
    // Display the size of detection boundary for the recognition
    telemetry.addData("- Size", JavaUtil.formatNumber(myTfodRecognition.getWidth(), 0) + " x
" + JavaUtil.formatNumber(myTfodRecognition.getHeight(), 0));
   }
  }

   public void pixelScoOne() {

     armMotor.setPower(FORWARD_SPEED);
     sleep(1000);
     armMotor.setPower(0);
     sleep(600);

     grabberLeft.setPower(1);
     sleep(800);

     armMotor.setPower(-FORWARD_SPEED);
     sleep(200);
     armMotor.setPower(0);

     grabberLeft.setPower(-1);
     sleep(800);

     armMotor.setPower(-FORWARD_SPEED);
     sleep(1000);
     armMotor.setPower(0);
   }

   public void pixelScoTwo() {
     grabberLeft.setPower(-1);
```

```
    grabberRight.setPower(1);
    sleep(1000);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);

    grabberLeft.setPower(1);
    grabberRight.setPower(-1);
    sleep(1000);

}

public void pixelScoThree() {
    grabberLeft.setPower(-1);
    sleep(1000);
    grabberRight.setPower(1);
    sleep(1500);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);

    grabberLeft.setPower(1);
    sleep(1000);
    grabberRight.setPower(-1);
    sleep(1500);

}

//function for grabber opening and closing
public void  pixelOpen(){
    grabberLeft.setPower(-1);
    grabberRight.setPower(1);
    sleep(1000);
    grabberLeft.setPower(1);
    grabberRight.setPower(-1);
    sleep(1000);
}

public void  pixelClose(){
    grabberLeft.setPower(1);
```

```
        grabberRight.setPower(-1);
        sleep(1000);
        grabberLeft.setPower(-1);
        grabberRight.setPower(1);
        sleep(1000);
    }

    public void encoderDrive(double speed, double frontLeftInches, double frontRightInches,
double backLeftInches, double backRightInches, double timeouts) {

        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {
            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
            frontLeftDrive.setTargetPosition(newFrontLeftTarget);
            frontRightDrive.setTargetPosition(newFrontRightTarget);
            backLeftDrive.setTargetPosition(newBackLeftTarget);
            backRightDrive.setTargetPosition(newBackRightTarget);

            // Turn On RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            // reset the timeout time and start motion.
            runtime.reset();
            frontLeftDrive.setPower(Math.abs(speed));
```

```
            frontRightDrive.setPower(Math.abs(speed));
            backLeftDrive.setPower(Math.abs(speed));
            backRightDrive.setPower(Math.abs(speed));

            while (opModeIsActive() &&
                 (runtime.seconds() < timeouts) &&
                 (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

                // Display it for the driver.
                telemetry.addData("Running to",  "  %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);

                telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
frontLeftDrive.getCurrentPosition(), frontRightDrive.getCurrentPosition(),
backLeftDrive.getCurrentPosition(), backRightDrive.getCurrentPosition());
                telemetry.update();
            }

            // Stop all motion;
            frontLeftDrive.setPower(0);
            frontRightDrive.setPower(0);
            backLeftDrive.setPower(0);
            backRightDrive.setPower(0);

            // Turn off RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

            sleep(250);   // optional pause after each move.
        }
    }
}
```

BlueVisAutoFar.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.hardware.Blinker;
import com.qualcomm.robotcore.hardware.HardwareDevice;
import com.qualcomm.robotcore.hardware.Gyroscope;
import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import com.qualcomm.robotcore.hardware.IMU;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;
import java.util.List;

//@Disabled
@Autonomous(name = "Blue Vision Far side", group = "LinearOpMode")
public class BlueVisAutoFar extends LinearOpMode{
//C: Binary variable added for the Blue Team Prop.
   boolean isBlueWarriorDetected;

   //C: Binary variable added to activate the web cam when set to true.
   boolean USE_WEBCAM;

   //C: Activating vision and setting parameters.
   TfodProcessor myTfodProcessor;
   VisionPortal myVisionPortal;

   //C: Declaring motors and servos.
   private DcMotor frontLeftDrive   = null;
```

```
    private DcMotor frontRightDrive  = null;
    private DcMotor backLeftDrive   = null;
    private DcMotor backRightDrive  = null;

    private CRServo grabberLeft;
    private CRServo grabberRight;
    private DcMotorEx armMotor;
    private CRServo pixelDrop;

    //C: OpMode runtime added.
    private ElapsedTime     runtime = new ElapsedTime();

    //C: This is the math involved in converting revolutions to inches.
    static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
    static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
    static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                                      (WHEEL_DIAMETER_INCHES * 3.1415);

    //C: The speeds the robot travels at when set.
    static final double     DRIVE_SPEED             = 0.2;
    static final double     FAST_DRIVE_SPEED        = 0.4;
    static final double     TURN_SPEED              = 0.2;
    static final double     SLOW_DRIVE_SPEED        = 0.1;
    static final double     SLOW_TURN_SPEED         = 0.2;
    static final double     FORWARD_SPEED           = 0.4;
    static final double     HEADING_THRESHOLD       = 1.0 ;

    private static final String TFOD_MODEL_FILE  = "Original Modell";

    private static final String[] LABEL = {
        "Red Warrior",
        "Blue Warrior"
    };


    //C: The variable to store the TensorFlow Object Detection processor.
```

```
private TfodProcessor tfod;

//C: The variable to store our instance of the vision portal.
private VisionPortal visionPortal;

@Override
public void runOpMode() {

    //C: Web cam variable is set to true activating it.
    USE_WEBCAM = true;

    initTfod();

    //C: Wait for the Control Hub start button to be activated.
    telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
    telemetry.addData(">", "Touch Play to start OpMode");
    telemetry.update();

    //C: Starting the drive and other system variables.
    frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
    frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
    backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
    backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
    armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
    grabberLeft = hardwareMap.get(CRServo.class,"grabberLeft");
    grabberRight = hardwareMap.get(CRServo.class, "grabberRight");
    pixelDrop = hardwareMap.get(CRServo.class, "pixelDrop");

    //C: Setting the direction of the motors to go reverse.
    frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
    backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    backRightDrive.setDirection(DcMotor.Direction.FORWARD);

    //C: Setting the run mode to stop and reset encoders after movement.
    frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
```

```
//C: Setting the run mode to move by encoders.
frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

//C: Setting the arm motor to move forward and by encoders.
armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

//C: Setting the grabber and the pixel dropper servos to move forward.
grabberLeft.setDirection(DcMotor.Direction.FORWARD);
grabberRight.setDirection(DcMotor.Direction.FORWARD);
pixelDrop.setDirection(DcMotor.Direction.FORWARD);

//C: Sends message to reveal a successful encoder reset.
telemetry.addData("Yass! Oki, Starting at",  "%7d :%7d :%7d :%7d",
    frontLeftDrive.getCurrentPosition(),
    frontRightDrive.getCurrentPosition(),
    backLeftDrive.getCurrentPosition(),
    backRightDrive.getCurrentPosition());
telemetry.update();

//C: Sounds like the name. Waiting for the opMode to begin.
waitForStart();

if (opModeIsActive()) {

    while (opModeIsActive()) {

    List<Recognition> myTfodRecognitions;
    Recognition myTfodRecognition;

    // Get a list of recognitions from TFOD.
    myTfodRecognitions = myTfodProcessor.getRecognitions();

    for (Recognition myTfodRecognition_item : myTfodRecognitions) {
        myTfodRecognition = myTfodRecognition_item;
        float prop;
```

```
            prop = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;

            if (prop >= 330){
               telemetry.addData("OMG!! Object is finally detected", "On the right");
               telemetry.update();

               USE_WEBCAM = false;

               thirdPath();
             }

            if (prop < 330 && prop > 210){
               telemetry.addData("OMG!! Object is finally detected", "In the center");
               telemetry.update();

               USE_WEBCAM = false;

               secondPath();
             }

            if (prop <= 210){
               telemetry.addData("OMG!! Object is finally detected", "On the left");
               telemetry.update();

               USE_WEBCAM = false;

               firstPath();
            }
         }

         }
      }
   }

public void firstPath() {

   //C: Robot moves forward 22 inches
   encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

   encoderDrive(SLOW_DRIVE_SPEED, 6, 6, 6, 6, 8.0);
```

```
//C: Robot strafes to the right and drops pixel and strafes back
encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

pixelDrop.setPower(0.5);
sleep(3000);

pixelDrop.setPower(-0.5);
sleep(1000);

encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

encoderDrive(SLOW_DRIVE_SPEED, 14, -14, 14, -14, 10.0);

encoderDrive(SLOW_DRIVE_SPEED, -3, 3, 3, -3, 10.0);

encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);

encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);

//C: 1
encoderDrive(SLOW_DRIVE_SPEED, -8, 8, 8, -8, 10.0);

encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

encoderDrive(SLOW_DRIVE_SPEED, -2, -2, -2, -2, 10.0);

pixelScoOne();

encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);
```

```
}

public void secondPath() {
    //C: Robot turns right for 7 inches and
    encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 8, 8, 8, 8, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, 10, -10, 10, -10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, -10, 10, -10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, -2, -2, -2, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, 14, -14, 14, -14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    //C: 2
    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);
```

```
    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);

}//C: end of the second path

//C: The third path goes to the third april tag code
public void thirdPath() {

    //C: Robot turns right for 7 inches and
    encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 8, 8, 8, 8, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, 15, -15, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 14, -14, 14, -14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);
```

```
        encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);

        //C: 3
        encoderDrive(SLOW_DRIVE_SPEED, 8, -8, -8, 8, 10.0);

        encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

        encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

        encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

        pixelScoOne();

        encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

        encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);

    }//C: end of the third path

    private void initTfod() {

        TfodProcessor.Builder myTfodProcessorBuilder;
        VisionPortal.Builder myVisionPortalBuilder;

        //C: Created a TfodProcessor.Builder.
        myTfodProcessorBuilder = new TfodProcessor.Builder();

        //C: The model file name "Original Modell" is added.
        myTfodProcessorBuilder.setModelFileName("Original Modell");

        // Set the full ordered list of labels the model is trained to recognize.
        myTfodProcessorBuilder.setModelLabels(JavaUtil.createListWith("Blue Warrior", "Red
Warrior"));
        //C: The aspect ratio is set to a horizontal postion (due to the higher being higher than the
bottom number)
        myTfodProcessorBuilder.setModelAspectRatio(16 / 9);
        // Create a TfodProcessor by calling build.
        myTfodProcessor = myTfodProcessorBuilder.build();
        // Next, create a VisionPortal.Builder and set attributes related to the camera.
```

```
        myVisionPortalBuilder = new VisionPortal.Builder();

        if (USE_WEBCAM) {
          // Use a webcam.
          myVisionPortalBuilder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
        } else {
           // Use the device's back camera.
           myVisionPortalBuilder.setCamera(BuiltinCameraDirection.BACK);
        }
        // Add myTfodProcessor to the VisionPortal.Builder.
        myVisionPortalBuilder.addProcessor(myTfodProcessor);
        // Create a VisionPortal by calling build.
        myVisionPortal = myVisionPortalBuilder.build();
    }

    private void telemetryTfod() {
        List<Recognition> myTfodRecognitions;
        Recognition myTfodRecognition;
        float x;
        float y;

        // Get a list of recognitions from TFOD.
        myTfodRecognitions = myTfodProcessor.getRecognitions();
        telemetry.addData("# Objects Detected", JavaUtil.listLength(myTfodRecognitions));
        // Iterate through list and call a function to display info for each recognized object.
        for (Recognition myTfodRecognition_item : myTfodRecognitions) {
        myTfodRecognition = myTfodRecognition_item;
      // Display info about the recognition.
      telemetry.addLine("");
      // Display label and confidence.
      // Display the label and confidence for the recognition.
      telemetry.addData("Image", myTfodRecognition.getLabel() + " (" +
JavaUtil.formatNumber(myTfodRecognition.getConfidence() * 100, 0) + " % Conf.)");
      // Display position.
      x = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;
      y = (myTfodRecognition.getTop() + myTfodRecognition.getBottom()) / 2;
      // Display the position of the center of the detection boundary for the recognition
      telemetry.addData("- Position", JavaUtil.formatNumber(x, 0) + ", " +
JavaUtil.formatNumber(y, 0));
      // Display size
```

```
      // Display the size of detection boundary for the recognition
      telemetry.addData("- Size", JavaUtil.formatNumber(myTfodRecognition.getWidth(), 0) + " x
" + JavaUtil.formatNumber(myTfodRecognition.getHeight(), 0));
    }
  }

    public void pixelScoOne() {

        armMotor.setPower(FORWARD_SPEED);
        sleep(1000);
        armMotor.setPower(0);
        sleep(600);

        grabberLeft.setPower(1);
        sleep(800);

        armMotor.setPower(-FORWARD_SPEED);
        sleep(200);
        armMotor.setPower(0);

        grabberLeft.setPower(-1);
        sleep(800);

        armMotor.setPower(-FORWARD_SPEED);
        sleep(1000);
        armMotor.setPower(0);
    }

    public void pixelScoTwo() {
        grabberLeft.setPower(-1);
        grabberRight.setPower(1);
        sleep(1000);

        armMotor.setPower(-FORWARD_SPEED);
        sleep(1000);

        grabberLeft.setPower(1);
        grabberRight.setPower(-1);
        sleep(1000);
```

```
    }

    public void pixelScoThree() {
        grabberLeft.setPower(-1);
        sleep(1000);
        grabberRight.setPower(1);
        sleep(1500);

        armMotor.setPower(-FORWARD_SPEED);
        sleep(1000);

        grabberLeft.setPower(1);
        sleep(1000);
        grabberRight.setPower(-1);
        sleep(1500);

    }

    //function for grabber opening and closing
    public void  pixelOpen(){
        grabberLeft.setPower(-1);
        grabberRight.setPower(1);
        sleep(1000);
        grabberLeft.setPower(1);
        grabberRight.setPower(-1);
        sleep(1000);
    }

    public void  pixelClose(){
        grabberLeft.setPower(1);
        grabberRight.setPower(-1);
        sleep(1000);
        grabberLeft.setPower(-1);
        grabberRight.setPower(1);
        sleep(1000);
    }

    public void encoderDrive(double speed, double frontLeftInches, double frontRightInches,
double backLeftInches, double backRightInches, double timeouts) {
```

```
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {
            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
            frontLeftDrive.setTargetPosition(newFrontLeftTarget);
            frontRightDrive.setTargetPosition(newFrontRightTarget);
            backLeftDrive.setTargetPosition(newBackLeftTarget);
            backRightDrive.setTargetPosition(newBackRightTarget);

            // Turn On RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            // reset the timeout time and start motion.
            runtime.reset();
            frontLeftDrive.setPower(Math.abs(speed));
            frontRightDrive.setPower(Math.abs(speed));
            backLeftDrive.setPower(Math.abs(speed));
            backRightDrive.setPower(Math.abs(speed));

            while (opModeIsActive() &&
                 (runtime.seconds() < timeouts) &&
                 (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

                // Display it for the driver.
```

```
            telemetry.addData("Running to",  "  %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);

            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
frontLeftDrive.getCurrentPosition(), frontRightDrive.getCurrentPosition(),
backLeftDrive.getCurrentPosition(), backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);

        // Turn off RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(250);   // optional pause after each move.
    }
  }
}
```

BlueVisionAuto.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import com.qualcomm.robotcore.hardware.IMU;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
```

```
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;
import java.util.List;

//@Disabled
@Autonomous(name = "Blue Vision", group = "LinearOpMode")
public class BlueVisionAuto extends LinearOpMode{

    //C: Binary variable added for the Blue Team Prop.
    boolean isBlueWarriorDetected;

    //C: Binary variable added to activate the web cam when set to true.
    boolean USE_WEBCAM;

    //C: Activating vision and setting parameters.
    TfodProcessor myTfodProcessor;
    VisionPortal myVisionPortal;

    //C: Declaring motors and servos.
    private DcMotor frontLeftDrive   = null;
    private DcMotor frontRightDrive  = null;
    private DcMotor backLeftDrive   = null;
    private DcMotor backRightDrive  = null;

    private CRServo grabberLeft;
    private CRServo grabberRight;
    private DcMotorEx armMotor;
    private CRServo pixelDrop;

    //C: OpMode runtime added.
    private ElapsedTime     runtime = new ElapsedTime();
```

```
    //C: This is the math involved in converting revolutions to inches.
    static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
    static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
    static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                                      (WHEEL_DIAMETER_INCHES * 3.1415);

    //C: The speeds the robot travels at when set.
    static final double     DRIVE_SPEED             = 0.2;
    static final double     FAST_DRIVE_SPEED        = 0.4;
    static final double     TURN_SPEED              = 0.2;
    static final double     SLOW_DRIVE_SPEED        = 0.1;
    static final double     SLOW_TURN_SPEED         = 0.2;
    static final double     FORWARD_SPEED           = 0.4;
    static final double     HEADING_THRESHOLD       = 1.0 ;

    private static final String TFOD_MODEL_FILE  = "Original Modell";

    private static final String[] LABEL = {
        "Red Warrior",
        "Blue Warrior"
    };


    //C: The variable to store the TensorFlow Object Detection processor.
    private TfodProcessor tfod;

    //C: The variable to store our instance of the vision portal.
    private VisionPortal visionPortal;

    @Override
    public void runOpMode() {

        //C: Web cam variable is set to true activating it.
        USE_WEBCAM = true;

        //C: Set to false until the camera reocgnizes the team prop
```

```
isBlueWarriorDetected = false;

//C: Wait for the Control Hub start button to be activated.
telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
telemetry.addData(">", "Touch Play to start OpMode");
telemetry.update();

//C: Starting the drive and other system variables.
frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
grabberLeft = hardwareMap.get(CRServo.class,"grabberLeft");
grabberRight = hardwareMap.get(CRServo.class, "grabberRight");
pixelDrop = hardwareMap.get(CRServo.class, "pixelDrop");

//C: Setting the direction of the motors to go reverse.
frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

//C: Setting the run mode to stop and reset encoders after movement.
frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

//C: Setting the run mode to move by encoders.
frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

//C: Setting the arm motor to move forward and by encoders.
armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

//C: Setting the grabber and the pixel dropper servos to move forward.
```

```
grabberLeft.setDirection(DcMotor.Direction.FORWARD);
grabberRight.setDirection(DcMotor.Direction.FORWARD);
pixelDrop.setDirection(DcMotor.Direction.FORWARD);

//C: Sends message to reveal a successful encoder reset.
telemetry.addData("Yass! Oki, Starting at",  "%7d :%7d :%7d :%7d",
    frontLeftDrive.getCurrentPosition(),
    frontRightDrive.getCurrentPosition(),
    backLeftDrive.getCurrentPosition(),
    backRightDrive.getCurrentPosition());
telemetry.update();

//C: Sounds like the name. Waiting for the opMode to begin.
waitForStart();

if (opModeIsActive()) {

    //C: drives forward 10 inches and turns left for 7 then pauses
    encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

    initTfod();

    //C: Robot stops moving to identify what's in front of it.
    searchTime(2.0);

    List<Recognition> myTfodRecognitions;
    Recognition myTfodRecognition;

    // Get a list of recognitions from TFOD.
    myTfodRecognitions = myTfodProcessor.getRecognitions();

    for (Recognition myTfodRecognition_item : myTfodRecognitions) {

        myTfodRecognition = myTfodRecognition_item;

        if (isBlueWarriorDetected = true) {
            telemetry.addData("OMG!! Object is finally detected", " Blue soldier");
            telemetry.update();
```

```
    firstPath();

} else {

    //C: Turn to the right
    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, -5, 5, 10.0);

    //lookToSeeTwo();

    searchTime(2.0);

    //List<Recognition> myTfodRecognitions;
    //Recognition myTfodRecognition;

    // Get a list of recognitions from TFOD.
    //myTfodRecognitions = myTfodProcessor.getRecognitions();

    if (isBlueWarriorDetected = true) {
        telemetry.addData("OMG!! Object is finally detected", " Blue soldier");
        telemetry.update();

        secondPath();

    } else {

        encoderDrive(SLOW_DRIVE_SPEED, -6, 6, -6, 6, 10.0);

        //lookToSeeThree();

        searchTime(2.0);

        //List<Recognition> myTfodRecognitions;
        //Recognition myTfodRecognition;

        // Get a list of recognitions from TFOD.
        // myTfodRecognitions = myTfodProcessor.getRecognitions();

        if (isBlueWarriorDetected = true) {
            telemetry.addData("OMG!! Object is finally detected", "Blue soldier");
```

```
                telemetry.update();

                thirdPath();

            } else {
                telemetry.addData("Uh oh", "nothing here");
                telemetry.update();

                 secondPath();
            }
        }
    }
  }
}

public void lookToSeeTwo() {

  searchTime(2.0);

  List<Recognition> myTfodRecognitions;
  Recognition myTfodRecognition;

  // Get a list of recognitions from TFOD.
  myTfodRecognitions = myTfodProcessor.getRecognitions();

  //for (Recognition myTfodRecognition_item : myTfodRecognitions) {

    //myTfodRecognition = myTfodRecognition_item;

    if (isBlueWarriorDetected = true) {
      telemetry.addData("OMG!! Object is finally detected", " Blue soldier");
      telemetry.update();

      secondPath();

    } else {

      encoderDrive(SLOW_DRIVE_SPEED, -6, 6, -6, 6, 10.0);
```

```
            lookToSeeThree();
        }
    //}

}

public void lookToSeeThree() {

    searchTime(2.0);

    List<Recognition> myTfodRecognitions;
    Recognition myTfodRecognition;

    // Get a list of recognitions from TFOD.
    myTfodRecognitions = myTfodProcessor.getRecognitions();

    //for (Recognition myTfodRecognition_item : myTfodRecognitions) {

        //myTfodRecognition = myTfodRecognition_item;

        if (isBlueWarriorDetected = true) {
            telemetry.addData("OMG!! Object is finally detected", "Blue soldier");
            telemetry.update();

            thirdPath();

        } else {
            telemetry.addData("Uh oh", "nothing here");
            telemetry.update();

            secondPath();
        }
    //}
}

public void searchTime(double timeouts) {

    runtime.reset();

    while (opModeIsActive() && (runtime.seconds() < timeouts)) {
```

```
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
    }
}

public void firstPath() {

    //C:
    encoderDrive(SLOW_DRIVE_SPEED, 2, 2, 2, 2, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, -5, 5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 6, 6, 6, 6, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 14, -14, 14, -14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, 3, 3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 25, 25, 25, 25, 20.0);

    //C: 1
    encoderDrive(SLOW_DRIVE_SPEED, -8, 8, 8, -8, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);
```

```
    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, -2, -2, -2, 10.0);

    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);
}

public void secondPath() {
    //C: Robot turns right for 7 inches and
    encoderDrive(SLOW_DRIVE_SPEED, 2, 2, 2, 2, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, -5, 5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 8, 8, 8, 8, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, 10, -10, 10, -10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, -10, 10, -10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, -2, -2, -2, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, 14, -14, 14, -14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 25, 25, 25, 25, 20.0);

    //C: 1
```

```
    //encoderDrive(SLOW_DRIVE_SPEED, -8, 8, 8, -8, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);

}//C: end of the second path

//C: The third path goes to the third april tag code
public void thirdPath() {

    //C: Robot turns right for 7 inches and
    encoderDrive(SLOW_DRIVE_SPEED, 2, 2, 2, 2, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, -5, 5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 8, 8, 8, 8, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, 15, -15, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);
```

```
encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

encoderDrive(SLOW_DRIVE_SPEED, 14, -14, 14, -14, 10.0);

encoderDrive(SLOW_DRIVE_SPEED, 25, 25, 25, 25, 20.0);

//C: 3
encoderDrive(SLOW_DRIVE_SPEED, 8, -8, -8, 8, 10.0);

encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

pixelScoOne();

encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);


/*encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 5.0);

encoderDrive(SLOW_DRIVE_SPEED, -11, -11, -11, -11, 15.0);

encoderDrive(SLOW_DRIVE_SPEED, -15, 15, 15, -15, 15.0);

encoderDrive(SLOW_DRIVE_SPEED, -15, 15, 15, -15, 20.0);

encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);

encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

encoderDrive(SLOW_DRIVE_SPEED, -12, 12, 12, -12, 12.0);

encoderDrive(SLOW_DRIVE_SPEED, -7, 7, 7, -7, 8.0);
```

```
        encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 5.0);

        encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);

        armMotor.setPower(0.5);
        sleep(1000);

        pixelScoThree();*/

    }//C: end of the third path

    private void initTfod() {
        TfodProcessor.Builder myTfodProcessorBuilder;
        VisionPortal.Builder myVisionPortalBuilder;

        //C: Created a TfodProcessor.Builder.
        myTfodProcessorBuilder = new TfodProcessor.Builder();

        //C: The model file name "Original Modell" is added.
        myTfodProcessorBuilder.setModelFileName("Original Modell");

        // Set the full ordered list of labels the model is trained to recognize.
        myTfodProcessorBuilder.setModelLabels(JavaUtil.createListWith("Blue Warrior", "Red
Warrior"));
        //C: The aspect ratio is set to a horizontal postion (due to the higher being higher than the
bottom number)
        myTfodProcessorBuilder.setModelAspectRatio(16 / 9);
        // Create a TfodProcessor by calling build.
        myTfodProcessor = myTfodProcessorBuilder.build();
        // Next, create a VisionPortal.Builder and set attributes related to the camera.
        myVisionPortalBuilder = new VisionPortal.Builder();

        if (USE_WEBCAM) {
          // Use a webcam.
          myVisionPortalBuilder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
        } else {
           // Use the device's back camera.
           myVisionPortalBuilder.setCamera(BuiltinCameraDirection.BACK);
        }
        // Add myTfodProcessor to the VisionPortal.Builder.
```

```
        myVisionPortalBuilder.addProcessor(myTfodProcessor);
        // Create a VisionPortal by calling build.
        myVisionPortal = myVisionPortalBuilder.build();
    }

    private void telemetryTfod() {
        List<Recognition> myTfodRecognitions;
        Recognition myTfodRecognition;
        float x;
        float y;

        // Get a list of recognitions from TFOD.
        myTfodRecognitions = myTfodProcessor.getRecognitions();
        telemetry.addData("# Objects Detected", JavaUtil.listLength(myTfodRecognitions));
        // Iterate through list and call a function to display info for each recognized object.
        for (Recognition myTfodRecognition_item : myTfodRecognitions) {
        myTfodRecognition = myTfodRecognition_item;
      // Display info about the recognition.
      telemetry.addLine("");
      // Display label and confidence.
      // Display the label and confidence for the recognition.
      telemetry.addData("Image", myTfodRecognition.getLabel() + " (" +
JavaUtil.formatNumber(myTfodRecognition.getConfidence() * 100, 0) + " % Conf.)");
      // Display position.
      x = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;
      y = (myTfodRecognition.getTop() + myTfodRecognition.getBottom()) / 2;
      // Display the position of the center of the detection boundary for the recognition
      telemetry.addData("- Position", JavaUtil.formatNumber(x, 0) + ", " +
JavaUtil.formatNumber(y, 0));
      // Display size
      // Display the size of detection boundary for the recognition
      telemetry.addData("- Size", JavaUtil.formatNumber(myTfodRecognition.getWidth(), 0) + " x
" + JavaUtil.formatNumber(myTfodRecognition.getHeight(), 0));
    }
  }

    public void pixelScoOne() {

        armMotor.setPower(FORWARD_SPEED);
        sleep(1000);
```

```
    armMotor.setPower(0);
    sleep(600);

    grabberLeft.setPower(1);
    sleep(800);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(200);
    armMotor.setPower(0);

    grabberLeft.setPower(-1);
    sleep(800);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);
    armMotor.setPower(0);
}

public void pixelScoTwo() {
    grabberLeft.setPower(-1);
    grabberRight.setPower(1);
    sleep(1000);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);

    grabberLeft.setPower(1);
    grabberRight.setPower(-1);
    sleep(1000);

}

public void pixelScoThree() {
    grabberLeft.setPower(-1);
    sleep(1000);
    grabberRight.setPower(1);
    sleep(1500);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);
```

```
        grabberLeft.setPower(1);
        sleep(1000);
        grabberRight.setPower(-1);
        sleep(1500);

    }

    //function for grabber opening and closing
    public void  pixelOpen(){
        grabberLeft.setPower(-1);
        grabberRight.setPower(1);
        sleep(1000);
        grabberLeft.setPower(1);
        grabberRight.setPower(-1);
        sleep(1000);
    }

    public void  pixelClose(){
        grabberLeft.setPower(1);
        grabberRight.setPower(-1);
        sleep(1000);
        grabberLeft.setPower(-1);
        grabberRight.setPower(1);
        sleep(1000);
    }

    public void encoderDrive(double speed, double frontLeftInches, double frontRightInches,
double backLeftInches, double backRightInches, double timeouts) {

        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {
            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
```

```
        newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
        newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
        newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
        frontLeftDrive.setTargetPosition(newFrontLeftTarget);
        frontRightDrive.setTargetPosition(newFrontRightTarget);
        backLeftDrive.setTargetPosition(newBackLeftTarget);
        backRightDrive.setTargetPosition(newBackRightTarget);

        // Turn On RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
             (runtime.seconds() < timeouts) &&
             (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  "  %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);

            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
frontLeftDrive.getCurrentPosition(), frontRightDrive.getCurrentPosition(),
backLeftDrive.getCurrentPosition(), backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
```

```
            frontLeftDrive.setPower(0);
            frontRightDrive.setPower(0);
            backLeftDrive.setPower(0);
            backRightDrive.setPower(0);

            // Turn off RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

            sleep(250);   // optional pause after each move.
        }
    }
}
```

Color_Sensor.java

```
package Cam_auto_2023;

import android.app.Activity;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import android.graphics.Color;
import android.view.View;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DistanceSensor;
import com.qualcomm.robotcore.hardware.NormalizedColorSensor;
import com.qualcomm.robotcore.hardware.NormalizedRGBA;
import com.qualcomm.robotcore.hardware.SwitchableLight;
import org.firstinspires.ftc.robotcore.external.navigation.DistanceUnit;

@Autonomous(name = "Sensor: Color", group = "Sensor")
@Disabled
public class Color_Sensor extends LinearOpMode {

  /** The colorSensor field will contain a reference to our color sensor hardware object */
  NormalizedColorSensor colorSensor;
```

```
  /** The relativeLayout field is used to aid in providing interesting visual feedback
   * in this sample application; you probably *don't* need this when you use a color sensor on
your
   * robot. Note that you won't see anything change on the Driver Station, only on the Robot
Controller. */
  View relativeLayout;

  @Override public void runOpMode() {

    // Get a reference to the RelativeLayout so we can later change the background
    // color of the Robot Controller app to match the hue detected by the RGB sensor.
    int relativeLayoutId =
hardwareMap.appContext.getResources().getIdentifier("RelativeLayout", "id",
hardwareMap.appContext.getPackageName());
    relativeLayout = ((Activity) hardwareMap.appContext).findViewById(relativeLayoutId);

    try {
      runSample(); // actually execute the sample
    } finally {
      // On the way out, *guarantee* that the background is reasonable. It doesn't actually start off
      // as pure white, but it's too much work to dig out what actually was used, and this is good
      // enough to at least make the screen reasonable again.
      // Set the panel back to the default color
      relativeLayout.post(new Runnable() {
        public void run() {
          relativeLayout.setBackgroundColor(Color.WHITE);
        }
      });
      }
  }

  protected void runSample() {
    // You can give the sensor a gain value, will be multiplied by the sensor's raw value before the
    // normalized color values are calculated. Color sensors (especially the REV Color Sensor V3)
    // can give very low values (depending on the lighting conditions), which only use a small part
    // of the 0-1 range that is available for the red, green, and blue values. In brighter conditions,
    // you should use a smaller gain than in dark conditions. If your gain is too high, all of the
    // colors will report at or near 1, and you won't be able to determine what color you are
    // actually looking at. For this reason, it's better to err on the side of a lower gain
    // (but always greater than  or equal to 1).
```

```
    float gain = 2;

    // Once per loop, we will update this hsvValues array. The first element (0) will contain the
    // hue, the second element (1) will contain the saturation, and the third element (2) will
    // contain the value. See
http://web.archive.org/web/20190311170843/https://infohost.nmt.edu/tcc/help/pubs/colortheory/
web/hsv.html
    // for an explanation of HSV color.
    final float[] hsvValues = new float[3];

    // xButtonPreviouslyPressed and xButtonCurrentlyPressed keep track of the previous and
current
    // state of the X button on the gamepad
    boolean xButtonPreviouslyPressed = false;
    boolean xButtonCurrentlyPressed = false;

    // Get a reference to our sensor object. It's recommended to use NormalizedColorSensor over
    // ColorSensor, because NormalizedColorSensor consistently gives values between 0 and 1,
while
    // the values you get from ColorSensor are dependent on the specific sensor you're using.
    colorSensor = hardwareMap.get(NormalizedColorSensor.class, "sensor_color");

    // If possible, turn the light on in the beginning (it might already be on anyway,
    // we just make sure it is if we can).
    if (colorSensor instanceof SwitchableLight) {
      ((SwitchableLight)colorSensor).enableLight(true);
    }

    // Wait for the start button to be pressed.
    waitForStart();

    // Loop until we are asked to stop
    while (opModeIsActive()) {
      // Explain basic gain information via telemetry
      telemetry.addLine("Hold the A button on gamepad 1 to increase gain, or B to decrease it.\n");
      telemetry.addLine("Higher gain values mean that the sensor will report larger numbers for
Red, Green, and Blue, and Value\n");

      // Update the gain value if either of the A or B gamepad buttons is being held
      if (gamepad1.a) {
```

```
      // Only increase the gain by a small amount, since this loop will occur multiple times per second.
      gain += 0.005;
    } else if (gamepad1.b && gain > 1) { // A gain of less than 1 will make the values smaller, which is not helpful.
      gain -= 0.005;
    }

    // Show the gain value via telemetry
    telemetry.addData("Gain", gain);

    // Tell the sensor our desired gain value (normally you would do this during initialization,
    // not during the loop)
    colorSensor.setGain(gain);

    // Check the status of the X button on the gamepad
    xButtonCurrentlyPressed = gamepad1.x;

    // If the button state is different than what it was, then act
    if (xButtonCurrentlyPressed != xButtonPreviouslyPressed) {
      // If the button is (now) down, then toggle the light
      if (xButtonCurrentlyPressed) {
        if (colorSensor instanceof SwitchableLight) {
          SwitchableLight light = (SwitchableLight)colorSensor;
          light.enableLight(!light.isLightOn());
        }
      }
    }
    xButtonPreviouslyPressed = xButtonCurrentlyPressed;

    // Get the normalized colors from the sensor
    NormalizedRGBA colors = colorSensor.getNormalizedColors();

    /* Use telemetry to display feedback on the driver station. We show the red, green, and blue
     * normalized values from the sensor (in the range of 0 to 1), as well as the equivalent
     * HSV (hue, saturation and value) values. See http://web.archive.org/web/20190311170843/https://infohost.nmt.edu/tcc/help/pubs/colortheory/web/hsv.html
     * for an explanation of HSV color. */
```

```
        // Update the hsvValues array by passing it to Color.colorToHSV()
        Color.colorToHSV(colors.toColor(), hsvValues);

        telemetry.addLine()
                .addData("Red", "%.3f", colors.red)
                .addData("Green", "%.3f", colors.green)
                .addData("Blue", "%.3f", colors.blue);
        telemetry.addLine()
                .addData("Hue", "%.3f", hsvValues[0])
                .addData("Saturation", "%.3f", hsvValues[1])
                .addData("Value", "%.3f", hsvValues[2]);
        telemetry.addData("Alpha", "%.3f", colors.alpha);

        /* If this color sensor also has a distance sensor, display the measured distance.
         * Note that the reported distance is only useful at very close range, and is impacted by
         * ambient light and surface reflectivity. */
        if (colorSensor instanceof DistanceSensor) {
          telemetry.addData("Distance (cm)", "%.3f", ((DistanceSensor)
colorSensor).getDistance(DistanceUnit.CM));
        }

        telemetry.update();

        // Change the Robot Controller's background color to match the color detected by the color
sensor.
        relativeLayout.post(new Runnable() {
          public void run() {
            relativeLayout.setBackgroundColor(Color.HSVToColor(hsvValues));
          }
        });
      }
    }
}
```

DriveAuto.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.util.ElapsedTime;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
```

```
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;
import java.util.List;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;

@Disabled
@Autonomous(name = "DRIVE TESTING", group = "LinearOpMode")
public class DriveAuto extends LinearOpMode {


    boolean USE_WEBCAM;
    TfodProcessor myTfodProcessor;
    VisionPortal myVisionPortal;

        /* Declare OpMode members. */
    private DcMotor frontLeftDrive   = null;
    private DcMotor frontRightDrive  = null;
    private DcMotor backLeftDrive   = null;
    private DcMotor backRightDrive  = null;
    private CRServo grabberLeft;
    private CRServo grabberRight;
   // private DcMotor armMotor;

    private ElapsedTime     runtime = new ElapsedTime();

    private DcMotorEx armMotor;

    static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
    static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
```

```
    static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                              (WHEEL_DIAMETER_INCHES * 3.1415);
    static final double     DRIVE_SPEED             = 0.4;
    static final double     FAST_DRIVE_SPEED        = 0.6;
    static final double     TURN_SPEED              = 0.3;
    static final double     SLOW_DRIVE_SPEED        = 0.1;
    static final double     SLOW_TURN_SPEED         = 0.2;
    static final double     FORWARD_SPEED           = 0.4;

    private static final String TFOD_MODEL_FILE  = "Original Modell";

    private static final String[] LABEL = {
        "Red Warrior",
        "Blue Warrior"
    };

    //static final double TURN_SPEED = 0.5;

   // private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

    /**
     * The variable to store our instance of the TensorFlow Object Detection processor.
     */
    private TfodProcessor tfod;

    /**
     * The variable to store our instance of the vision portal.
     */
    private VisionPortal visionPortal;

    @Override
    public void runOpMode() {

        USE_WEBCAM = true;

        //initTfod();
```

```
//telemetryTfod();
//telemetry.update();

// Wait for the DS start button to be touched.
telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
telemetry.addData(">", "Touch Play to start OpMode");
telemetry.update();

        // Initialize the drive system variables.
frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
grabberLeft = hardwareMap.get(CRServo.class,"grabberLeft");
grabberRight = hardwareMap.get(CRServo.class, "grabberRight");

frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
grabberLeft.setDirection(DcMotor.Direction.FORWARD);
grabberRight.setDirection(DcMotor.Direction.FORWARD);

// Send telemetry message to indicate successful Encoder reset
telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d :%7d :%7d",
                frontLeftDrive.getCurrentPosition(),
```

```
                frontRightDrive.getCurrentPosition(),
                backLeftDrive.getCurrentPosition(),
                backRightDrive.getCurrentPosition());
telemetry.update();

waitForStart();

if (opModeIsActive()) {

   //C: Robot drives for 12 inches then turns left for 7
   encoderDrive(SLOW_DRIVE_SPEED, 12, 12, 12, 12, 15.0);

   encoderDrive(SLOW_DRIVE_SPEED, 7, -7, 7, -7, 10.0);

   //C: Robot stops moving to identify what's in front of it

   frontLeftDrive.setPower(0);
   frontRightDrive.setPower(0);
   backLeftDrive.setPower(0);
   backRightDrive.setPower(0);

   sleep(3000);

   //C: Robot turns right  5 inches to search for team prop
encoderDrive(SLOW_DRIVE_SPEED, -5, 5, -5, 5, 10.0);

frontLeftDrive.setPower(0);
frontRightDrive.setPower(0);
backLeftDrive.setPower(0);
backRightDrive.setPower(0);

sleep(3000);

   //C: Robot turns right 6 inches to search for team prop
encoderDrive(SLOW_DRIVE_SPEED, -6, 6, -6, 6, 10.0);

   frontLeftDrive.setPower(0);
   frontRightDrive.setPower(0);
   backLeftDrive.setPower(0);
   backRightDrive.setPower(0);
```

```
            sleep(3000);

            //C: Robot has successfully identified the team prop and turns 6 inches to the left to
straighten out
        encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

            //C: Robot straightens out by turning 2 inches to the right
        encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 5.0);

            //C: Robot backs up by 11 inches
        encoderDrive(SLOW_DRIVE_SPEED, -11, -11, -11, -11, 15.0);

            //C: Robot strafes to the left for 15 inches
        encoderDrive(SLOW_DRIVE_SPEED, -15, 15, 15, -15, 15.0);

            //C: Robot strafes to the left further by 20 inches
        encoderDrive(SLOW_DRIVE_SPEED, -20, 20, 20, -20, 20.0);

            //C: Robot moves forward 10 inches
        encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);

            //C: Robot turns to the right for 15 inches
        encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

            //C: Robot strafes to the left for 4 inches
        encoderDrive(SLOW_DRIVE_SPEED, -4, 4, 4, -4, 8.0);

            //C: Robot strafes left for 7 inches
        encoderDrive(SLOW_DRIVE_SPEED, -7, 7, 7, -7, 8.0);

           //C: Robot backs up for 3 inches
        encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 5.0);

            //C: Robot turns left for 2 inches to straighten out
        encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);
        }
    }

    public void encoderDrive(double speed,
```

```
                    double frontLeftInches, double frontRightInches,
                    double backLeftInches, double backRightInches,
                    double timeouts) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {
            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
            frontLeftDrive.setTargetPosition(newFrontLeftTarget);
            frontRightDrive.setTargetPosition(newFrontRightTarget);
            backLeftDrive.setTargetPosition(newBackLeftTarget);
            backRightDrive.setTargetPosition(newBackRightTarget);

            // Turn On RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            // reset the timeout time and start motion.
            runtime.reset();
            frontLeftDrive.setPower(Math.abs(speed));
            frontRightDrive.setPower(Math.abs(speed));
            backLeftDrive.setPower(Math.abs(speed));
            backRightDrive.setPower(Math.abs(speed));

            while (opModeIsActive() &&
                 (runtime.seconds() < timeouts) &&
```

```
            (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  "  %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
                                frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
                                backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
           //armMotor.setPower(0);

           // Turn off RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(250);   // optional pause after each move.
    }
}
}
```

EncodersTest2.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.hardware.DcMotor;
```

```
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;

//@Disabled
@Autonomous(name = "Running thru Auto motions", group = "LinearOpMode")
public class EncoderTest2 extends LinearOpMode {

   private ElapsedTime     runtime = new ElapsedTime();

         /* Declare OpMode members. */
   private DcMotor         frontLeftDrive   = null;
   private DcMotor         frontRightDrive  = null;
   private DcMotor         backLeftDrive   = null;
   private DcMotor         backRightDrive  = null;

   private DcMotorEx armMotor;
   private CRServo pixelDrop;
   private CRServo grabberLeft;
   private CRServo grabberRight;

   static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
   static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
   static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
   static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                             (WHEEL_DIAMETER_INCHES * 3.1415);
   static final double     DRIVE_SPEED             = 0.4;
```

```
static final double     FAST_DRIVE_SPEED       = 0.6;
static final double     TURN_SPEED             = 0.3;
static final double     SLOW_DRIVE_SPEED       = 0.1;
static final double     SLOW_TURN_SPEED        = 0.2;
static final double     FORWARD_SPEED          = 0.4;

@Override
public void runOpMode() {

    // Wait for the DS start button to be touched.
    telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
    telemetry.addData(">", "Touch Play to start OpMode");
    telemetry.update();

    // Initialize the drive system variables.
    frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
    frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
    backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
    backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
    armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
    grabberLeft = hardwareMap.get(CRServo.class,"grabberLeft");
    grabberRight = hardwareMap.get(CRServo.class, "grabberRight");
    pixelDrop = hardwareMap.get(CRServo.class, "pixelDrop");

    frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
    backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    backRightDrive.setDirection(DcMotor.Direction.FORWARD);

    frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

    frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

    armMotor.setDirection(DcMotor.Direction.FORWARD);
```

```
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
grabberLeft.setDirection(DcMotor.Direction.FORWARD);
grabberRight.setDirection(DcMotor.Direction.FORWARD);
pixelDrop.setDirection(DcMotor.Direction.FORWARD);

// Send telemetry message to indicate successful Encoder reset
telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d :%7d :%7d",
                  frontLeftDrive.getCurrentPosition(),
                  frontRightDrive.getCurrentPosition(),
                  backLeftDrive.getCurrentPosition(),
                  backRightDrive.getCurrentPosition());
telemetry.update();

waitForStart();

if (opModeIsActive()) {

   //  encoderDrive(SLOW_DRIVE_SPEED, -5, 5, -5, 5, 10.0);
   //C: Go forward 8 inches
   encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 11.0);

   //C: Robot turns right for 7 inches and go forward 6 inches
   //encoderDrive(SLOW_DRIVE_SPEED, -7, 7, -7, 7, 10.0);

   encoderDrive(SLOW_DRIVE_SPEED, 13, 13, 13, 13, 15.0);

   //encoderDrive(SLOW_DRIVE_SPEED, 11, 11, 11, 11, 11.0);

   encoderDrive(SLOW_DRIVE_SPEED, 7, -7, -7, 7, 10.0);

   pixelDrop.setPower(0.5);
   sleep(2000);

   pixelDrop.setPower(-0.5);
   sleep(1000);

   encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

   encoderDrive(SLOW_DRIVE_SPEED, -25, -25, -25, -25, 20.0);
```

```
        encoderDrive(SLOW_DRIVE_SPEED, -15, 15, 15, -15, 15.0);

        encoderDrive(SLOW_DRIVE_SPEED, -15, 15, 15, -15, 20.0);

        encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);

        encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

        encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 8.0);

        encoderDrive(SLOW_DRIVE_SPEED, -4, -4, -4, -4, 5.0);

        encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);

        armMotor.setPower(FORWARD_SPEED);
        sleep(1000);
        armMotor.setPower(0);
        sleep(600);

        grabberLeft.setPower(1);
        sleep(800);

        armMotor.setPower(-FORWARD_SPEED);
        sleep(200);
        armMotor.setPower(0);

        grabberLeft.setPower(-1);
        sleep(800);

        armMotor.setPower(-FORWARD_SPEED);
        sleep(1000);
        armMotor.setPower(0);

        while (opModeIsActive()) {

        }
    }
}

public void encoderDrive(double speed,
```

```
                    double frontLeftInches, double frontRightInches,
                    double backLeftInches, double backRightInches,
                    double timeouts) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {

            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
            frontLeftDrive.setTargetPosition(newFrontLeftTarget);
            frontRightDrive.setTargetPosition(newFrontRightTarget);
            backLeftDrive.setTargetPosition(newBackLeftTarget);
            backRightDrive.setTargetPosition(newBackRightTarget);

            // Turn On RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            // reset the timeout time and start motion.
            runtime.reset();
            frontLeftDrive.setPower(Math.abs(speed));
            frontRightDrive.setPower(Math.abs(speed));
            backLeftDrive.setPower(Math.abs(speed));
            backRightDrive.setPower(Math.abs(speed));

            while (opModeIsActive() &&
                 (runtime.seconds() < timeouts) &&
```

```
            (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  " %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
                              frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
                              backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
        //armMotor.setPower(0);

        // Turn off RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(250);   // optional pause after each move.
    }
  }
}
```

GryoDrive.java

```
package Cam_auto_2023;

import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
```

```
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.IMU;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.util.Range;
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;

@Autonomous(name="Auto Gyro", group="LinearOpMode")
@Disabled
public class GryoDrive extends LinearOpMode {

    /* Declare OpMode members. */
    private DcMotor         frontleftDrive = null;
    private DcMotor         frontrightDrive  = null;
    private DcMotor         backleftDrive   = null;
    private DcMotor         backrightDrive  = null;
    private IMU             imu         = null;      // Control/Expansion Hub IMU

    private double          headingError  = 0;

    // These variable are declared here (as class members) so they can be updated in various methods,
    // but still be displayed by sendTelemetry()
    private double  targetHeading = 0;
    private double  driveSpeed    = 0;
    private double  turnSpeed     = 0;
    private double  frontleftSpeed     = 0;
    private double  frontrightSpeed    = 0;
    private double  backleftSpeed     = 0;
    private double  backrightSpeed    = 0;
    private int     frontleftTarget    = 0;
    private int     frontrightTarget   = 0;
    private int     backleftTarget    = 0;
    private int     backrightTarget   = 0;

    // Calculate the COUNTS_PER_INCH for your specific drive train.
    // Go to your motor vendor website to determine your motor's COUNTS_PER_MOTOR_REV
    // For external drive gearing, set DRIVE_GEAR_REDUCTION as needed.
```

```
    // For example, use a value of 2.0 for a 12-tooth spur gear driving a 24-tooth spur gear.
    // This is gearing DOWN for less speed and more torque.
    // For gearing UP, use a gear ratio less than 1.0. Note this will affect the direction of wheel rotation.
    static final double     COUNTS_PER_MOTOR_REV    = 28; //537.7 ;   // eg: GoBILDA 312 RPM Yellow Jacket
    static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;     // No External Gearing.
    static final double     WHEEL_DIAMETER_INCHES   = 4.0 ;     // For figuring circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV * DRIVE_GEAR_REDUCTION) /
                                                      (WHEEL_DIAMETER_INCHES * 3.1415);

    // These constants define the desired driving/control characteristics
    // They can/should be tweaked to suit the specific robot drive train.
    static final double     DRIVE_SPEED             = 0.3;     // Max driving speed for better distance accuracy.
    static final double     TURN_SPEED              = 0.2;     // Max Turn speed to limit turn rate
    static final double     HEADING_THRESHOLD       = 1.0 ;    // How close must the heading get to the target before moving to next step.
                                                               // Requiring more accuracy (a smaller number) will often make the turn take longer to get into the final position.
    // Define the Proportional control coefficient (or GAIN) for "heading control".
    // We define one value when Turning (larger errors), and the other is used when Driving straight (smaller errors).
    // Increase these numbers if the heading does not corrects strongly enough (eg: a heavy robot or using tracks)
    // Decrease these numbers if the heading does not settle on the correct value (eg: very agile robot with omni wheels)
    static final double     P_TURN_GAIN             = 0.02;     // Larger is more responsive, but also less stable
    static final double     P_DRIVE_GAIN            = 0.03;     // Larger is more responsive, but also less stable


    @Override
    public void runOpMode() {

        // Initialize the drive system variables.
        frontleftDrive  = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
        frontrightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
```

```
        backleftDrive  = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
        backrightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");

        /*frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
        frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
        backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
        backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");*/

        // To drive forward, most robots need the motor on one side to be reversed, because the axles point in opposite directions.
        // When run, this OpMode should start both motors driving forward. So adjust these two lines based on your first test drive.
        // Note: The settings here assume direct drive on left and right wheels.  Gear Reduction or 90 Deg drives may require direction flips
        frontleftDrive.setDirection(DcMotor.Direction.REVERSE);
        frontrightDrive.setDirection(DcMotor.Direction.FORWARD);
        backleftDrive.setDirection(DcMotor.Direction.REVERSE);
        backrightDrive.setDirection(DcMotor.Direction.FORWARD);

        /* The next two lines define Hub orientation.
         * The Default Orientation (shown) is when a hub is mounted horizontally with the printed logo pointing UP and the USB port pointing FORWARD.
         *
         * To Do:  EDIT these two lines to match YOUR mounting configuration.
         */
        RevHubOrientationOnRobot.LogoFacingDirection logoDirection = RevHubOrientationOnRobot.LogoFacingDirection.UP;
        RevHubOrientationOnRobot.UsbFacingDirection  usbDirection  = RevHubOrientationOnRobot.UsbFacingDirection.FORWARD;
        RevHubOrientationOnRobot orientationOnRobot = new RevHubOrientationOnRobot(logoDirection, usbDirection);

        // Now initialize the IMU with this mounting orientation
        // This sample expects the IMU to be in a REV Hub and named "imu".
        imu = hardwareMap.get(IMU.class, "imu");
        imu.initialize(new IMU.Parameters(orientationOnRobot));

        // Ensure the robot is stationary.  Reset the encoders and set the motors to BRAKE mode
        frontleftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        frontrightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
```

```
    backleftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backrightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    frontleftDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    frontrightDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    backleftDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    backrightDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);

    // Wait for the game to start (Display Gyro value while waiting)
    while (opModeInInit()) {
       telemetry.addData(">", "Robot Heading = %4.0f", getHeading());
       telemetry.update();
    }

    // Set the encoders for closed loop speed control, and reset the heading.
    frontleftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    frontrightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    backleftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    backrightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    imu.resetYaw();

    // Step through each leg of the path,
    // Notes:   Reverse movement is obtained by setting a negative distance (not speed)
    //          holdHeading() is used after turns to let the heading stabilize
    //          Add a sleep(2000) after any step to keep the telemetry data visible for review

    driveStraight(DRIVE_SPEED, 14.0, 0.0);    // Drive Forward 24"
    turnToHeading( TURN_SPEED, -45.0);               // Turn  CW to -45 Degrees
    holdHeading( TURN_SPEED, -45.0, 0.5);   // Hold -45 Deg heading for a 1/2 second

    driveStraight(DRIVE_SPEED, 14.0, -45.0);  // Drive Forward 17" at -45 degrees (12"x and
12"y)
    turnToHeading( TURN_SPEED,  45.0);               // Turn  CCW  to  45 Degrees
    holdHeading( TURN_SPEED,  45.0, 0.5);    // Hold  45 Deg heading for a 1/2 second

    driveStraight(DRIVE_SPEED, 14.0, 45.0);  // Drive Forward 17" at 45 degrees (-12"x and
12"y)
    turnToHeading( TURN_SPEED,   0.0);               // Turn  CW  to 0 Degrees
    holdHeading( TURN_SPEED,   0.0, 1.0);    // Hold  0 Deg heading for 1 second
```

```
        driveStraight(DRIVE_SPEED,-14.0, 0.0);    // Drive in Reverse 48" (should return to
approx. staring position)

        telemetry.addData("Path", "Complete");
        telemetry.update();
        sleep(1000);  // Pause to display last telemetry message.
    }

    /*
     *
============================================================================
================================
     * Driving "Helper" functions are below this line.
     * These provide the high and low level methods that handle driving straight and turning.
     *
============================================================================
================================
     */

    // **********  HIGH Level driving functions.  ********************

    /**
    *  Drive in a straight line, on a fixed compass heading (angle), based on encoder counts.
    *  Move will stop if either of these conditions occur:
    *  1) Move gets to the desired position
    *  2) Driver stops the OpMode running.
    *
    * @param maxDriveSpeed MAX Speed for forward/rev motion (range 0 to +1.0) .
    * @param distance   Distance (in inches) to move from current position.  Negative distance
means move backward.
    * @param heading      Absolute Heading Angle (in Degrees) relative to last gyro reset.
    *                0 = fwd. +ve is CCW from fwd. -ve is CW from forward.
    *                If a relative angle is required, add/subtract from the current robotHeading.
    */
    public void driveStraight(double maxDriveSpeed,
                              double distance,
                              double heading) {

        // Ensure that the OpMode is still active
        if (opModeIsActive()) {
```

```
        // Determine new target position, and pass to motor controller
        int moveCounts = (int)(distance * COUNTS_PER_INCH);
        frontleftTarget = frontleftDrive.getCurrentPosition() + moveCounts;
        frontrightTarget = frontrightDrive.getCurrentPosition() + moveCounts;
        backleftTarget = backleftDrive.getCurrentPosition() + moveCounts;
        backrightTarget = backrightDrive.getCurrentPosition() + moveCounts;

        // Set Target FIRST, then turn on RUN_TO_POSITION
        frontleftDrive.setTargetPosition(frontleftTarget);
        frontrightDrive.setTargetPosition(frontrightTarget);
        backleftDrive.setTargetPosition(backleftTarget);
        backrightDrive.setTargetPosition(backrightTarget);

        frontleftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontrightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backleftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backrightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // Set the required driving speed  (must be positive for RUN_TO_POSITION)
        // Start driving straight, and then enter the control loop
        maxDriveSpeed = Math.abs(maxDriveSpeed);
        moveRobot(maxDriveSpeed, 0);

        // keep looping while we are still active, and BOTH motors are running.
        while (opModeIsActive() &&
            (frontleftDrive.isBusy() && frontrightDrive.isBusy() && backleftDrive.isBusy()
&& backrightDrive.isBusy())) {

            // Determine required steering to keep on heading
            turnSpeed = getSteeringCorrection(heading, P_DRIVE_GAIN);

            // if driving in reverse, the motor correction also needs to be reversed
            if (distance < 0)
                turnSpeed *= -1.0;

            // Apply the turning correction to the current driving speed.
            moveRobot(driveSpeed, turnSpeed);

            // Display drive status for the driver.
```

```
            sendTelemetry(true);
        }

        // Stop all motion & Turn off RUN_TO_POSITION
        moveRobot(0, 0);
        frontleftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontrightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backleftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backrightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    }
}

/**
 *  Spin on the central axis to point in a new direction.
 *  <p>
 *  Move will stop if either of these conditions occur:
 *  <p>
 *  1) Move gets to the heading (angle)
 *  <p>
 *  2) Driver stops the OpMode running.
 *
 * @param maxTurnSpeed Desired MAX speed of turn. (range 0 to +1.0)
 * @param heading Absolute Heading Angle (in Degrees) relative to last gyro reset.
 *            0 = fwd. +ve is CCW from fwd. -ve is CW from forward.
 *            If a relative angle is required, add/subtract from current heading.
 */
public void turnToHeading(double maxTurnSpeed, double heading) {

    // Run getSteeringCorrection() once to pre-calculate the current error
    getSteeringCorrection(heading, P_DRIVE_GAIN);

    // keep looping while we are still active, and not on heading.
    while (opModeIsActive() && (Math.abs(headingError) > HEADING_THRESHOLD)) {

        // Determine required steering to keep on heading
        turnSpeed = getSteeringCorrection(heading, P_TURN_GAIN);

        // Clip the speed to the maximum permitted value.
        turnSpeed = Range.clip(turnSpeed, -maxTurnSpeed, maxTurnSpeed); //-maxTurnSpeed,
maxTurnSpeed);
```

```
            // Pivot in place by applying the turning correction
            moveRobot(0, turnSpeed);

            // Display drive status for the driver.
            sendTelemetry(false);
        }

        // Stop all motion;
        moveRobot(0, 0);//, 0, 0);
    }

    /**
     *  Obtain & hold a heading for a finite amount of time
     *  <p>
     *  Move will stop once the requested time has elapsed
     *  <p>
     *  This function is useful for giving the robot a moment to stabilize it's heading between
movements.
     *
     * @param maxTurnSpeed      Maximum differential turn speed (range 0 to +1.0)
     * @param heading    Absolute Heading Angle (in Degrees) relative to last gyro reset.
     *                   0 = fwd. +ve is CCW from fwd. -ve is CW from forward.
     *                   If a relative angle is required, add/subtract from current heading.
     * @param holdTime   Length of time (in seconds) to hold the specified heading.
     */
    public void holdHeading(double maxTurnSpeed, double heading, double holdTime) {

        ElapsedTime holdTimer = new ElapsedTime();
        holdTimer.reset();

        // keep looping while we have time remaining.
        while (opModeIsActive() && (holdTimer.time() < holdTime)) {
            // Determine required steering to keep on heading
            turnSpeed = getSteeringCorrection(heading, P_TURN_GAIN);

            // Clip the speed to the maximum permitted value.
            turnSpeed = Range.clip(turnSpeed, -maxTurnSpeed, maxTurnSpeed);

            // Pivot in place by applying the turning correction
```

```
            moveRobot(0, turnSpeed);

            // Display drive status for the driver.
            sendTelemetry(false);
        }

        // Stop all motion;
        moveRobot(0, 0);
    }

    // **********  LOW Level driving functions.  ********************

    /**
     * Use a Proportional Controller to determine how much steering correction is required.
     *
     * @param desiredHeading        The desired absolute heading (relative to last heading reset)
     * @param proportionalGain      Gain factor applied to heading error to obtain turning power.
     * @return                      Turning power needed to get to required heading.
     */
    public double getSteeringCorrection(double desiredHeading, double proportionalGain) {
        targetHeading = desiredHeading;  // Save for telemetry

        // Determine the heading current error
        headingError = targetHeading - getHeading();

        // Normalize the error to be within +/- 180 degrees
        //C: maybe use if statements instead to make robot turn back to original heading
        while (headingError > 180)  headingError -= 360;
        while (headingError <= -180) headingError += 360;

        // Multiply the error by the gain to determine the required steering correction/  Limit the
result to +/- 1.0
        return Range.clip(headingError * proportionalGain, -1, 1);
    }

    /**
     * Take separate drive (fwd/rev) and turn (right/left) requests,
     * combines them, and applies the appropriate speed commands to the left and right wheel
motors.
     * @param drive forward motor speed
```

```
     * @param turn  clockwise turning motor speed.
     */
    public void moveRobot(double drive, double turn) {
        driveSpeed = drive;     // save this value as a class member so it can be used by telemetry.
        turnSpeed  = turn;      // save this value as a class member so it can be used by telemetry.

        frontleftSpeed  = drive - turn;
        frontrightSpeed = drive + turn;
        backleftSpeed  = drive - turn;
        backrightSpeed = drive + turn;

        // Scale speeds down if either one exceeds +/- 1.0;
        double max = Math.max(Math.abs(frontleftSpeed), Math.abs(frontrightSpeed)/*,
Math.abs(backleftSpeed), Math.abs(backrightSpeed)*/);
        if (max > 1.0)
        {
            frontleftSpeed /= max;
            frontrightSpeed /= max;
            backleftSpeed /= max;
            backrightSpeed /= max;
        }

        frontleftDrive.setPower(frontleftSpeed);
        frontrightDrive.setPower(frontrightSpeed);
        backleftDrive.setPower(backleftSpeed);
        backrightDrive.setPower(backrightSpeed);
    }

    /**
     *  Display the various control parameters while driving
     *
     * @param straight  Set to true if we are driving straight, and the encoder positions should be
included in the telemetry.
     */
    private void sendTelemetry(boolean straight) {

        if (straight) {
            telemetry.addData("Motion", "Drive Straight");
            telemetry.addData("Target Pos FL:FR:BL:BR",  "%7d:%7d:%7d:%7d",
frontleftTarget,  frontrightTarget, backleftTarget,  backrightTarget);
```

```
            telemetry.addData("Actual Pos FL:FR:BL:BR",  "%7d:%7d:%7d:%7d",
frontleftDrive.getCurrentPosition(),
                    frontrightDrive.getCurrentPosition(), backleftDrive.getCurrentPosition(),
                    backrightDrive.getCurrentPosition());
        } else {
            telemetry.addData("Motion", "Turning");
        }

        telemetry.addData("Heading- Target : Current", "%5.2f : %5.0f", targetHeading,
getHeading());
        telemetry.addData("Error  : Steer Pwr",  "%5.1f : %5.1f", headingError, turnSpeed);
        telemetry.addData("Wheel Speeds FL:FR:BL:BR", "%5.2f : %5.2f: %5.2f : %5.2f",
frontleftSpeed, frontrightSpeed, backleftSpeed, backrightSpeed);
        telemetry.update();
    }

    /**
     * read the Robot heading directly from the IMU (in degrees)
     */
    public double getHeading() {
        YawPitchRollAngles orientation = imu.getRobotYawPitchRollAngles();
        return orientation.getYaw(AngleUnit.DEGREES);
    }
}
```

GryoTurn.java

```
package Cam_auto_2023;

import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import com.qualcomm.robotcore.hardware.IMU;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.IMU;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.util.Range;
```

```
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;

@Autonomous(name="Robot: Auto Drive By Gyro", group="Robot")
//@Disabled
public class GryoTurn extends LinearOpMode {

    //C: Declaring motors and servos.
    private DcMotor frontLeftDrive   = null;
    private DcMotor frontRightDrive  = null;
    private DcMotor backLeftDrive   = null;
    private DcMotor backRightDrive  = null;
    private IMU             imu         = null;      // Control/Expansion Hub IMU

    private double          headingError  = 0;

    // These variable are declared here (as class members) so they can be updated in various methods,
    // but still be displayed by sendTelemetry()
    private double  targetHeading = 0;
    private double  driveSpeed    = 0;
    private double  turnSpeed     = 0;
    private double  frontLeftSpeed     = 0;
    private double  frontRightSpeed    = 0;
    private double  backLeftSpeed     = 0;
    private double  backRightSpeed    = 0;
    private int     frontLeftTarget    = 0;
    private int     frontRightTarget   = 0;
    private int     backLeftTarget    = 0;
    private int     backRightTarget   = 0;

    // Calculate the COUNTS_PER_INCH for your specific drive train.
    // Go to your motor vendor website to determine your motor's COUNTS_PER_MOTOR_REV
    // For external drive gearing, set DRIVE_GEAR_REDUCTION as needed.
    // For example, use a value of 2.0 for a 12-tooth spur gear driving a 24-tooth spur gear.
    // This is gearing DOWN for less speed and more torque.
    // For gearing UP, use a gear ratio less than 1.0. Note this will affect the direction of wheel rotation.
    static final double     COUNTS_PER_MOTOR_REV    = 28 ; //537.7 ;   // eg: GoBILDA 312 RPM Yellow Jacket
```

```
    static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;     // No External Gearing.
    static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//4.0 ;     // For figuring
circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                                      (WHEEL_DIAMETER_INCHES * 3.1415);

    // These constants define the desired driving/control characteristics
    // They can/should be tweaked to suit the specific robot drive train.
    static final double     DRIVE_SPEED             = 0.1;     // Max driving speed for better distance
accuracy.
    static final double     TURN_SPEED              = 0.1;     // Max Turn speed to limit turn rate
    static final double     HEADING_THRESHOLD       = 1.0 ;    // How close must the heading
get to the target before moving to next step.
                                                   // Requiring more accuracy (a smaller number) will often
make the turn take longer to get into the final position.
    // Define the Proportional control coefficient (or GAIN) for "heading control".
    // We define one value when Turning (larger errors), and the other is used when Driving
straight (smaller errors).
    // Increase these numbers if the heading does not corrects strongly enough (eg: a heavy robot
or using tracks)
    // Decrease these numbers if the heading does not settle on the correct value (eg: very agile
robot with omni wheels)
    static final double     P_TURN_GAIN             = 0.02;     // Larger is more responsive, but also
less stable
    static final double     P_DRIVE_GAIN            = 0.03;     // Larger is more responsive, but also
less stable


    @Override
    public void runOpMode() {

        // Initialize the drive system variables.
        frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
        frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
        backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
        backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
        //armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
```

```
        // To drive forward, most robots need the motor on one side to be reversed, because the axles point in opposite directions.
        // When run, this OpMode should start both motors driving forward. So adjust these two lines based on your first test drive.
        // Note: The settings here assume direct drive on left and right wheels.  Gear Reduction or 90 Deg drives may require direction flips
        frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
        frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
        backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
        backRightDrive.setDirection(DcMotor.Direction.FORWARD);

        /* The next two lines define Hub orientation.
         * The Default Orientation (shown) is when a hub is mounted horizontally with the printed logo pointing UP and the USB port pointing FORWARD.
         *
         * To Do:  EDIT these two lines to match YOUR mounting configuration.
         */
        RevHubOrientationOnRobot.LogoFacingDirection logoDirection = RevHubOrientationOnRobot.LogoFacingDirection.UP;
        RevHubOrientationOnRobot.UsbFacingDirection  usbDirection  = RevHubOrientationOnRobot.UsbFacingDirection.BACKWARD; //FORWARD;
        RevHubOrientationOnRobot orientationOnRobot = new RevHubOrientationOnRobot(logoDirection, usbDirection);

        // Now initialize the IMU with this mounting orientation
        // This sample expects the IMU to be in a REV Hub and named "imu".
        imu = hardwareMap.get(IMU.class, "imu");
        imu.initialize(new IMU.Parameters(orientationOnRobot));

        // Ensure the robot is stationary.  Reset the encoders and set the motors to BRAKE mode
        frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        frontLeftDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        frontRightDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        backLeftDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        backRightDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);

        // Wait for the game to start (Display Gyro value while waiting)
```

```
        while (opModeInInit()) {
            telemetry.addData(">", "Robot Heading = %4.0f", getHeading());
            telemetry.update();
        }

        // Set the encoders for closed loop speed control, and reset the heading.
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        imu.resetYaw();

        // Step through each leg of the path,
        // Notes:   Reverse movement is obtained by setting a negative distance (not speed)
        //          holdHeading() is used after turns to let the heading stabilize
        //          Add a sleep(2000) after any step to keep the telemetry data visible for review

        //driveStraight(DRIVE_SPEED, 24.0, 0.0);    // Drive Forward 24"
        //turnToHeading( TURN_SPEED, -45.0);               // Turn  CW to -45 Degrees
        holdHeading( TURN_SPEED, -45.0, 3.0);   // Hold -45 Deg heading for a 1/2 second

        //driveStraight(DRIVE_SPEED, 17.0, -45.0);  // Drive Forward 17" at -45 degrees (12"x
and 12"y)
        //turnToHeading( TURN_SPEED,  45.0);               // Turn  CCW  to  45 Degrees
        holdHeading( TURN_SPEED,  45.0, 3.0);    // Hold  45 Deg heading for a 1/2 second

       // driveStraight(DRIVE_SPEED, 17.0, 45.0);  // Drive Forward 17" at 45 degrees (-12"x and
12"y)
        //turnToHeading( TURN_SPEED,   0.0);               // Turn  CW  to 0 Degrees
        holdHeading( TURN_SPEED,   90.0, 5.0);    // Hold  0 Deg heading for 1 second

        //driveStraight(DRIVE_SPEED,-48.0, 0.0);    // Drive in Reverse 48" (should return to
approx. staring position)

        telemetry.addData("Path", "Complete");
        telemetry.update();
        sleep(1000);  // Pause to display last telemetry message.
    }

    /*
```

```
    *
================================================================================
================================
    * Driving "Helper" functions are below this line.
    * These provide the high and low level methods that handle driving straight and turning.
    *
================================================================================
================================
    */

   // **********  HIGH Level driving functions.  ********************

   /**
   *  Drive in a straight line, on a fixed compass heading (angle), based on encoder counts.
   *  Move will stop if either of these conditions occur:
   *  1) Move gets to the desired position
   *  2) Driver stops the OpMode running.
   *
   * @param maxDriveSpeed MAX Speed for forward/rev motion (range 0 to +1.0) .
   * @param distance   Distance (in inches) to move from current position.  Negative distance
means move backward.
   * @param heading      Absolute Heading Angle (in Degrees) relative to last gyro reset.
   *                0 = fwd. +ve is CCW from fwd. -ve is CW from forward.
   *                If a relative angle is required, add/subtract from the current robotHeading.
   */
   public void driveStraight(double maxDriveSpeed,
                             double distance,
                             double heading) {

      // Ensure that the OpMode is still active
      if (opModeIsActive()) {

         // Determine new target position, and pass to motor controller
         int moveCounts = (int)(distance * COUNTS_PER_INCH);
         frontLeftTarget = frontLeftDrive.getCurrentPosition() + moveCounts;
         frontRightTarget = frontRightDrive.getCurrentPosition() + moveCounts;
         backLeftTarget = backLeftDrive.getCurrentPosition() + moveCounts;
         backRightTarget = backRightDrive.getCurrentPosition() + moveCounts;

         // Set Target FIRST, then turn on RUN_TO_POSITION
```

```
        frontLeftDrive.setTargetPosition(frontLeftTarget);
        frontRightDrive.setTargetPosition(frontRightTarget);
        backLeftDrive.setTargetPosition(backLeftTarget);
        backRightDrive.setTargetPosition(backRightTarget);

        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // Set the required driving speed  (must be positive for RUN_TO_POSITION)
        // Start driving straight, and then enter the control loop
        maxDriveSpeed = Math.abs(maxDriveSpeed);
        moveRobot(maxDriveSpeed, 0);

        // keep looping while we are still active, and BOTH motors are running.
        while (opModeIsActive() &&
            (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Determine required steering to keep on heading
            turnSpeed = getSteeringCorrection(heading, P_DRIVE_GAIN);

            // if driving in reverse, the motor correction also needs to be reversed
            if (distance < 0)
                turnSpeed *= -1.0;

            // Apply the turning correction to the current driving speed.
            moveRobot(driveSpeed, turnSpeed);

            // Display drive status for the driver.
            sendTelemetry(true);
        }

        // Stop all motion & Turn off RUN_TO_POSITION
        moveRobot(0, 0);
        frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
```

```
    }
}

/**
 *  Spin on the central axis to point in a new direction.
 *  <p>
 *  Move will stop if either of these conditions occur:
 *  <p>
 *  1) Move gets to the heading (angle)
 *  <p>
 *  2) Driver stops the OpMode running.
 *
 * @param maxTurnSpeed Desired MAX speed of turn. (range 0 to +1.0)
 * @param heading Absolute Heading Angle (in Degrees) relative to last gyro reset.
 *            0 = fwd. +ve is CCW from fwd. -ve is CW from forward.
 *            If a relative angle is required, add/subtract from current heading.
 */
public void turnToHeading(double maxTurnSpeed, double heading) {

    // Run getSteeringCorrection() once to pre-calculate the current error
    getSteeringCorrection(heading, P_DRIVE_GAIN);

    // keep looping while we are still active, and not on heading.
    while (opModeIsActive() && (Math.abs(headingError) > HEADING_THRESHOLD)) {

        // Determine required steering to keep on heading
        turnSpeed = getSteeringCorrection(heading, P_TURN_GAIN);

        // Clip the speed to the maximum permitted value.
        turnSpeed = Range.clip(turnSpeed, -maxTurnSpeed, maxTurnSpeed);

        // Pivot in place by applying the turning correction
        moveRobot(0, turnSpeed);

        // Display drive status for the driver.
        sendTelemetry(false);
    }

    // Stop all motion;
    moveRobot(0, 0);
```

```
    }

    /**
     *  Obtain & hold a heading for a finite amount of time
     *  <p>
     *  Move will stop once the requested time has elapsed
     *  <p>
     *  This function is useful for giving the robot a moment to stabilize it's heading between movements.
     *
     * @param maxTurnSpeed      Maximum differential turn speed (range 0 to +1.0)
     * @param heading    Absolute Heading Angle (in Degrees) relative to last gyro reset.
     *                   0 = fwd. +ve is CCW from fwd. -ve is CW from forward.
     *                   If a relative angle is required, add/subtract from current heading.
     * @param holdTime   Length of time (in seconds) to hold the specified heading.
     */
    public void holdHeading(double maxTurnSpeed, double heading, double holdTime) {

        ElapsedTime holdTimer = new ElapsedTime();
        holdTimer.reset();

        // keep looping while we have time remaining.
        while (opModeIsActive() && (holdTimer.time() < holdTime)) {
            // Determine required steering to keep on heading
            //turnSpeed = getSteeringCorrection(heading, P_TURN_GAIN);

            // Clip the speed to the maximum permitted value.
            turnSpeed = Range.clip(turnSpeed, -maxTurnSpeed, maxTurnSpeed);

            // Pivot in place by applying the turning correction
            moveRobot(0, turnSpeed);

            // Display drive status for the driver.
            sendTelemetry(false);
        }

        // Stop all motion;
        moveRobot(0, 0);
    }
```

```
    // **********  LOW Level driving functions.  ********************

    /**
     * Use a Proportional Controller to determine how much steering correction is required.
     *
     * @param desiredHeading        The desired absolute heading (relative to last heading reset)
     * @param proportionalGain      Gain factor applied to heading error to obtain turning power.
     * @return                      Turning power needed to get to required heading.
     */
    public double getSteeringCorrection(double desiredHeading, double proportionalGain) {
        targetHeading = desiredHeading;  // Save for telemetry

        // Determine the heading current error
        headingError = targetHeading - getHeading();

        // Normalize the error to be within +/- 180 degrees
        while (headingError > 180)  headingError -= 360;
        while (headingError <= -180) headingError += 360;

        // Multiply the error by the gain to determine the required steering correction/  Limit the
result to +/- 1.0
        return Range.clip(headingError * proportionalGain, -1, 1);
    }

    /**
     * Take separate drive (fwd/rev) and turn (right/left) requests,
     * combines them, and applies the appropriate speed commands to the left and right wheel
motors.
     * @param drive forward motor speed
     * @param turn  clockwise turning motor speed.
     */
    public void moveRobot(double drive, double turn) {
        driveSpeed = drive;     // save this value as a class member so it can be used by telemetry.
        turnSpeed  = turn;      // save this value as a class member so it can be used by telemetry.

        frontLeftSpeed  = drive - turn;
        frontRightSpeed = drive + turn;
        backLeftSpeed  = drive - turn;
        backRightSpeed = drive + turn;
```

```
        // Scale speeds down if either one exceeds +/- 1.0;
        double max = Math.max(Math.abs(frontLeftSpeed), Math.abs(frontRightSpeed)); //&&
Math.max(Math.abs(backLeftSpeed), Math.abs(backRightSpeed));
        if (max > 1.0)
        {
           frontLeftSpeed /= max;
           frontRightSpeed /= max;
           backLeftSpeed /= max;
           backRightSpeed /= max;
        }

        frontLeftDrive.setPower(frontLeftSpeed);
        frontRightDrive.setPower(frontRightSpeed);
        backLeftDrive.setPower(backLeftSpeed);
        backRightDrive.setPower(backRightSpeed);
    }

    /**
     *  Display the various control parameters while driving
     *
     * @param straight  Set to true if we are driving straight, and the encoder positions should be
included in the telemetry.
     */
    private void sendTelemetry(boolean straight) {

        if (straight) {
           telemetry.addData("Motion", "Drive Straight");
           telemetry.addData("Target Pos FL:FR:BL:BR",  "%7d:%7d:%7d:%7d",
frontLeftTarget,  frontRightTarget, backLeftTarget,  backRightTarget);
           telemetry.addData("Actual Pos FL : FR : BL : BR",  "%7d:%7d:%7d:%7d",
frontLeftDrive.getCurrentPosition(),
                   frontRightDrive.getCurrentPosition(), backLeftDrive.getCurrentPosition(),
                   backRightDrive.getCurrentPosition());
        } else {
           telemetry.addData("Motion", "Turning");
        }

        telemetry.addData("Heading- Target : Current", "%5.2f : %5.0f", targetHeading,
getHeading());
        telemetry.addData("Error  : Steer Pwr",  "%5.1f : %5.1f", headingError, turnSpeed);
```

```
        telemetry.addData("Wheel Speeds FL : FR : BL : BR", "%5.2f : %5.2f : %5.2f : %5.2f",
frontLeftSpeed, frontRightSpeed, backLeftSpeed, backRightSpeed);
        telemetry.update();
    }

    /**
     * read the Robot heading directly from the IMU (in degrees)
     */
    public double getHeading() {
        YawPitchRollAngles orientation = imu.getRobotYawPitchRollAngles();
        return orientation.getYaw(AngleUnit.DEGREES);
    }
}
```

LeftAutoPark.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;

import java.util.List;

@Autonomous(name = "Left Auto Park", group = "LinearOpMode")
@Disabled
public class LeftAutoPark extends LinearOpMode {

    private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera
```

```
        /* Declare OpMode members. */
    private DcMotor         frontLeftDrive   = null;
    private DcMotor         frontRightDrive  = null;
    private DcMotor         backLeftDrive   = null;
    private DcMotor         backRightDrive  = null;

    private DcMotorEx armMotor;

    static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
    static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
    static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                              (WHEEL_DIAMETER_INCHES * 3.1415);
    static final double     DRIVE_SPEED             = 0.4;
    static final double     FAST_DRIVE_SPEED            = 0.6;
    static final double     TURN_SPEED              = 0.3;
    static final double     SLOW_DRIVE_SPEED            = 0.1;
    static final double     SLOW_TURN_SPEED             = 0.2;
    static final double FORWARD_SPEED = 0.4;
    //static final double TURN_SPEED = 0.5;

    /**
     * The variable to store our instance of the TensorFlow Object Detection processor.
     */
    private TfodProcessor tfod;

    private ElapsedTime     runtime = new ElapsedTime();

    /**
     * The variable to store our instance of the vision portal.
     */
    private VisionPortal visionPortal;

    @Override
    public void runOpMode() {

        initTfod();
```

```
// Wait for the DS start button to be touched.
telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
telemetry.addData(">", "Touch Play to start OpMode");
telemetry.update();

        // Initialize the drive system variables.
frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");

frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

// Send telemetry message to indicate successful Encoder reset
telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d :%7d :%7d",
                  frontLeftDrive.getCurrentPosition(),
                  frontRightDrive.getCurrentPosition(),
                  backLeftDrive.getCurrentPosition(),
                  backRightDrive.getCurrentPosition());
telemetry.update();

waitForStart();
```

```
if (opModeIsActive()) {

    //moveForward();
    //sleep(100);

    //strafeLeft();
    //sleep(900);

    encoderDrive(SLOW_DRIVE_SPEED, -25, 25, 25, -25, 10.0);
    //sleep(5000);

    /*while (opModeIsActive()) {


        //List<Recognition> currentRecognitions = tfod.getRecognitions();
        //telemetry.addData("# Objects Detected", currentRecognitions.size());
        if (tfod != null) {
        List<Recognition> currentRecognitions = tfod.getRecognitions();
            if (currentRecognitions != null) {
                //telemetry.addData("# Objects Detected", updatedRecognitions.size());

                for (Recognition recognition : currentRecognitions) {
                    encoderDrive(SLOW_DRIVE_SPEED, -25, 25, 25, -25, 1.0);
                }
            }
        }

        telemetryTfod();

        // Push telemetry to the Driver Station.
        telemetry.update();

        // Save CPU resources; can resume streaming when needed.
        /*if (gamepad1.dpad_down) {
            visionPortal.stopStreaming();
        } else if (gamepad1.dpad_up) {
            visionPortal.resumeStreaming();
        }
```

```
            // Share the CPU.
            sleep(20);
        }*/
    }

    // Save more CPU resources when camera is no longer needed.
    visionPortal.close();

}   // end runOpMode()

private void strafeLeft() {
    frontLeftDrive.setPower(-FORWARD_SPEED);
    frontRightDrive.setPower(FORWARD_SPEED);
    backLeftDrive.setPower(FORWARD_SPEED);
    backRightDrive.setPower(-FORWARD_SPEED);
}

private void strafeRight() {
    frontLeftDrive.setPower(FORWARD_SPEED);
    frontRightDrive.setPower(-FORWARD_SPEED);
    backLeftDrive.setPower(-FORWARD_SPEED);
    backRightDrive.setPower(FORWARD_SPEED);
}

private void moveForward() {
    frontLeftDrive.setPower(FORWARD_SPEED);
    frontRightDrive.setPower(FORWARD_SPEED);
    backLeftDrive.setPower(FORWARD_SPEED);
    backRightDrive.setPower(FORWARD_SPEED);
}

private void moveBackward() {
    frontLeftDrive.setPower(-FORWARD_SPEED);
    frontRightDrive.setPower(-FORWARD_SPEED);
    backLeftDrive.setPower(-FORWARD_SPEED);
    backRightDrive.setPower(-FORWARD_SPEED);
}

private void turnLeft() {
    frontLeftDrive.setPower(-FORWARD_SPEED);
```

```
        frontRightDrive.setPower(FORWARD_SPEED);
        backLeftDrive.setPower(-FORWARD_SPEED);
        backRightDrive.setPower(FORWARD_SPEED);
    }

    private void turnRight() {
        frontLeftDrive.setPower(FORWARD_SPEED);
        frontRightDrive.setPower(-FORWARD_SPEED);
        backLeftDrive.setPower(FORWARD_SPEED);
        backRightDrive.setPower(-FORWARD_SPEED);
    }
    /**
     * Initialize the TensorFlow Object Detection processor.
     */
    private void initTfod() {

        // Create the TensorFlow processor by using a builder.
        tfod = new TfodProcessor.Builder()

            // Use setModelAssetName() if the TF Model is built in as an asset.
            // Use setModelFileName() if you have downloaded a custom team model to the Robot
Controller.
            //.setModelAssetName(TFOD_MODEL_ASSET)
            //.setModelFileName(TFOD_MODEL_FILE)

            //.setModelLabels(LABELS)
            //.setIsModelTensorFlow2(true)
            //.setIsModelQuantized(true)
            //.setModelInputSize(300)
            //.setModelAspectRatio(16.0 / 9.0)

            .build();

        // Create the vision portal by using a builder.
        VisionPortal.Builder builder = new VisionPortal.Builder();

        // Set the camera (webcam vs. built-in RC phone camera).
        if (USE_WEBCAM) {
            builder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
        } else {
```

```
        builder.setCamera(BuiltinCameraDirection.BACK);
    }

    // Choose a camera resolution. Not all cameras support all resolutions.
    //builder.setCameraResolution(new Size(640, 480));

    // Enable the RC preview (LiveView).  Set "false" to omit camera monitoring.
    //builder.enableCameraMonitoring(true);

    // Set the stream format; MJPEG uses less bandwidth than default YUY2.
    //builder.setStreamFormat(VisionPortal.StreamFormat.YUY2);

    // Choose whether or not LiveView stops if no processors are enabled.
    // If set "true", monitor shows solid orange screen if no processors enabled.
    // If set "false", monitor shows camera view without annotations.
    //builder.setAutoStopLiveView(false);

    // Set and enable the processor.
    builder.addProcessor(tfod);

    // Build the Vision Portal, using the above settings.
    visionPortal = builder.build();

    // Set confidence threshold for TFOD recognitions, at any time.
    //tfod.setMinResultConfidence(0.75f);

    // Disable or re-enable the TFOD processor at any time.
    //visionPortal.setProcessorEnabled(tfod, true);

}   // end method initTfod()

/**
 * Add telemetry about TensorFlow Object Detection (TFOD) recognitions.
 */
private void telemetryTfod() {

    List<Recognition> currentRecognitions = tfod.getRecognitions();
    telemetry.addData("# Objects Detected", currentRecognitions.size());

    // Step through the list of recognitions and display info for each one.
```

```
        for (Recognition recognition : currentRecognitions) {
            double x = (recognition.getLeft() + recognition.getRight()) / 2 ;
            double y = (recognition.getTop()  + recognition.getBottom()) / 2 ;

            telemetry.addData(""," ");
            telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(),
recognition.getConfidence() * 100);
            telemetry.addData("- Position", "%.0f / %.0f", x, y);
            telemetry.addData("- Size", "%.0f x %.0f", recognition.getWidth(),
recognition.getHeight());
        }   // end for() loop

    }   // end method telemetryTfod()

    public void encoderDrive(double speed,
                             double frontLeftInches, double frontRightInches,
                             double backLeftInches, double backRightInches,
                             double timeoutS) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {

            //C: Ensures that whle the drive mode is active the arm does not fall down
            //R: YOU IDIOT I CANNOT BELIEVE YOU MADE THE POWER EIGHT!??!?!?!?!
GRAHHHHHHHHHHHHHHHH
            //armMotor.setPower(0.5);

            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
```

```
        frontLeftDrive.setTargetPosition(newFrontLeftTarget);
        frontRightDrive.setTargetPosition(newFrontRightTarget);
        backLeftDrive.setTargetPosition(newBackLeftTarget);
        backRightDrive.setTargetPosition(newBackRightTarget);

        // Turn On RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
            (runtime.seconds() < timeoutS) &&
            (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  "  %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
                                frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
                                backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
        //armMotor.setPower(0);
```

```
            // Turn off RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

            sleep(250);   // optional pause after each move.
        }
    }
}   // end class
```

Original_modell_detection.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import java.util.List;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;



//@Disabled
@Autonomous(name = "Test prop vision")
public class Original_modell_detection extends LinearOpMode {

  boolean USE_WEBCAM;
  TfodProcessor myTfodProcessor;
  VisionPortal myVisionPortal;

  boolean isBlueOrbDetected;
  boolean isRedOrbDetected;

  boolean isBlueOrbRight;
```

```
boolean isBlueOrbLeft;
boolean isBlueOrbCenter;

boolean isRedOrbRight;
boolean isRedOrbLeft;
boolean isRedOrbCenter;

/**
 * This function is executed when this OpMode is selected from the Driver Station.
 */
@Override
public void runOpMode() {
  // This 2023-2024 OpMode illustrates the basics of TensorFlow Object Detection, using
  // Original Modell
  USE_WEBCAM = true;

  /*isBlueOrbDetected = false;

  isBlueOrbRight = false;
  isBlueOrbLeft = false;
  isBlueOrbCenter = false;

  isRedOrbRight = false;
  isRedOrbLeft = false;
  isRedOrbCenter = false;*/

  // Initialize TFOD before waitForStart.
  initTfod();

  /*List<Recognition> myTfodRecognitions;
     Recognition myTfodRecognition;

     myTfodRecognitions = myTfodProcessor.getRecognitions();

     //myTfodRecognition.getLabel();

     float prop;

     for (Recognition myTfodRecognition_item : myTfodRecognitions) {
       myTfodRecognition = myTfodRecognition_item;
```

```
      prop = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;

      if (prop >= 330){
        telemetry.addData("OMG!! Object is finally detected", "On the right");
        telemetry.update();

        isBlueOrbRight = true;

      }

      if (prop < 330 && prop > 210){
        telemetry.addData("OMG!! Object is finally detected", "In the center");
        telemetry.update();

        isBlueOrbCenter = true;
      }

      if (prop <= 210){
        telemetry.addData("OMG!! Object is finally detected", "On the left");
        telemetry.update();

        isBlueOrbCenter = true;

      }*/

    //}

// Wait for the match to begin.
telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
telemetry.addData(">", "Touch Play to start OpMode");
telemetry.update();
waitForStart();
if (opModeIsActive()) {
  // Put run blocks here.

  /*List<Recognition> myTfodRecognitions;
    Recognition myTfodRecognition;

     myTfodRecognitions = myTfodProcessor.getRecognitions();
```

```
  for (Recognition myTfodRecognition_item : myTfodRecognitions) {
   myTfodRecognition = myTfodRecognition_item;

   if (myTfodRecognitions.equals("Blue Warrior")){
    telemetry.addData("OMG!! Object is finally detected", "Blue boy");
    telemetry.update();
   }
  }*/

 while (opModeIsActive()) {
  // Put loop blocks here.

  List<Recognition> myTfodRecognitions;
     Recognition myTfodRecognition;

     // Get a list of recognitions from TFOD.
     myTfodRecognitions = myTfodProcessor.getRecognitions();

     for (Recognition myTfodRecognition_item : myTfodRecognitions) {

        myTfodRecognition = myTfodRecognition_item;

        float prop;

        prop = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;

        if (prop >= 330){
           telemetry.addData("OMG!! Object is finally detected", "On the right");
           telemetry.update();
           //sleep(1000);

           //USE_WEBCAM = false;

           //thirdPath();
         }

        if (prop < 330 && prop > 210){
           telemetry.addData("OMG!! Object is finally detected", "In the center");
           telemetry.update();
```

```
            //sleep(1000);

            //USE_WEBCAM = false;

            //secondPath();
          }

        if (prop <= 210){
            telemetry.addData("OMG!! Object is finally detected", "On the left");
            telemetry.update();
            //sleep(1000);

            //USE_WEBCAM = false;

            //firstPath();
        }
    }

//List<Recognition> myTfodRecognitions;
//Recognition myTfodRecognition;

//myTfodRecognitions = myTfodProcessor.getRecognitions();

//myTfodRecognition.getLabel();

//float prop;

//for (Recognition myTfodRecognition_item : myTfodRecognitions) {
  //myTfodRecognition = myTfodRecognition_item;

  //prop = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;

  /*if (isBlueOrbRight = true){
    telemetry.addData("Yess, lets go..", "right");
    telemetry.update();
  }

  if (isBlueOrbCenter = true){
    telemetry.addData("Yess, lets go..", "center");
    telemetry.update();
```

```
  }

  if (isBlueOrbLeft = true){
    telemetry.addData("Yess, lets go..", "left");
    telemetry.update();
  }*/


  /*if (isBlueWarriorDetected = true) {
    telemetry.addData("OMG!! Object is finally detected", "Blue Warrior");
    telemetry.update();
  }*/

  /*if (isBlueWarriorDetected = true) {
    telemetry.addData("OMG!! Object is finally detected", "Blue Warrior");
    telemetry.update();
  }
}*/

/*List<Recognition> myTfodRecognitions;
Recognition myTfodRecognition;

// Get a list of recognitions from TFOD.
myTfodRecognitions = myTfodProcessor.getRecognitions();

if (JavaUtil.listLength(myTfodRecognitions) == 1) {
  telemetry.addData("OMG!! Object is finally detected", "Warrior");
  telemetry.update();
}*/

// Display info about the recognition.
//telemetry.addLine("");

//telemetryTfod();

// Push telemetry to the Driver Station.
//telemetry.update();
//}
//List<Recognition> currentRecognitions = tfod.getRecognitions();
//telemetry.addData("# Objects Detected", currentRecognitions.size());
```

```
      if (gamepad1.dpad_down) {
        // Temporarily stop the streaming session.
        myVisionPortal.stopStreaming();
      } else if (gamepad1.dpad_up) {
        // Resume the streaming session if previously stopped.
        myVisionPortal.resumeStreaming();
      }
      // Share the CPU.
      //sleep(20);
    }
  }
}
/*
 * Initialize TensorFlow Object Detection.
 */
private void initTfod() {
  TfodProcessor.Builder myTfodProcessorBuilder;
  VisionPortal.Builder myVisionPortalBuilder;

  // First, create a TfodProcessor.Builder.
  myTfodProcessorBuilder = new TfodProcessor.Builder();
  // Set the name of the file where the model can be found.
  myTfodProcessorBuilder.setModelFileName("Prop_Orbs");
  // Set the full ordered list of labels the model is trained to recognize.
  myTfodProcessorBuilder.setModelLabels(JavaUtil.createListWith("Blue Orb", "Red Orb"));
  // Set the aspect ratio for the images used when the model was created.
  myTfodProcessorBuilder.setModelAspectRatio(19 / 9);
  // Create a TfodProcessor by calling build.
  myTfodProcessor = myTfodProcessorBuilder.build();
  // Next, create a VisionPortal.Builder and set attributes related to the camera.
  myVisionPortalBuilder = new VisionPortal.Builder();
  if (USE_WEBCAM) {
    // Use a webcam.
    myVisionPortalBuilder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
  } else {
    // Use the device's back camera.
    myVisionPortalBuilder.setCamera(BuiltinCameraDirection.BACK);
  }
```

```
    // Add myTfodProcessor to the VisionPortal.Builder.
    myVisionPortalBuilder.addProcessor(myTfodProcessor);
    // Create a VisionPortal by calling build.
    myVisionPortal = myVisionPortalBuilder.build();
  }

  /**
   * Display info (using telemetry) for a detected object
   */
  private void telemetryTfod() {
    List<Recognition> myTfodRecognitions;
    Recognition myTfodRecognition;
    float x;
    float y;

    // Get a list of recognitions from TFOD.
    myTfodRecognitions = myTfodProcessor.getRecognitions();
    telemetry.addData("# Objects Detected", JavaUtil.listLength(myTfodRecognitions));
    // Iterate through list and call a function to display info for each recognized object.
    for (Recognition myTfodRecognition_item : myTfodRecognitions) {
      myTfodRecognition = myTfodRecognition_item;
      // Display info about the recognition.
      telemetry.addLine("");
      // Display label and confidence.
      // Display the label and confidence for the recognition.
      telemetry.addData("Image", myTfodRecognition.getLabel() + " (" +
JavaUtil.formatNumber(myTfodRecognition.getConfidence() * 100, 0) + " % Conf.)");
      // Display position.
      x = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;
      y = (myTfodRecognition.getTop() + myTfodRecognition.getBottom()) / 2;
      // Display the position of the center of the detection boundary for the recognition
      telemetry.addData("- Position", JavaUtil.formatNumber(x, 0) + ", " +
JavaUtil.formatNumber(y, 0));
      // Display size
      // Display the size of detection boundary for the recognition
      telemetry.addData("- Size", JavaUtil.formatNumber(myTfodRecognition.getWidth(), 0) + " x
" + JavaUtil.formatNumber(myTfodRecognition.getHeight(), 0));
    }
  }
}
```

OriginalModellauto.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;

import java.util.List;

@Autonomous(name = "Original modell detection", group = "LinearOpMode")
@Disabled
public class OriginalModellauto extends LinearOpMode {

    private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

    /**
     * The variable to store our instance of the TensorFlow Object Detection processor.
     */
    private TfodProcessor tfod;

    private static final String TFOD_MODEL_FILE  = "Original Modell";

    private static final String[] LABEL = {
        "Red Warrior",
        "Blue Warrior"
    };

    /**
     * The variable to store our instance of the vision portal.
     */
    private VisionPortal visionPortal;
```

```
    @Override
    public void runOpMode() {

        //initTfod();

        // Wait for the DS start button to be touched.
        telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
        telemetry.addData(">", "Touch Play to start OpMode");
        telemetry.update();
        waitForStart();

        if (opModeIsActive()) {
            while (opModeIsActive()) {

                //telemetryTfod();

                List<Recognition> currentRecognitions = tfod.getRecognitions();
                telemetry.addData("# Objects Detected", currentRecognitions.size());

                for (Recognition recognition : currentRecognitions) {
                    if (recognition.getLabel().equals("Blue Warrior") ){
                        telemetry.addData("OMG!! Object is finally detected", "Blue boy");
                        telemetry.update();
                        sleep(1000);
                    }
                }

                /*if (recognition.getLabel().equals("Blue Warrior")) {
                    telemetry.addData("OMG!! Object is finally detected", "Blue boy");
                    telemetry.update();
                }

                if (recognition.getLabel().equals("Red Warrior")) {
telemetry.addData("EEK!! Object detected hehe", "Red boy");
                    telemetry.update();
                }*/

                // Push telemetry to the Driver Station.
                telemetry.update();
```

```
            // Save CPU resources; can resume streaming when needed.
            if (gamepad1.dpad_down) {
                visionPortal.stopStreaming();
            } else if (gamepad1.dpad_up) {
                visionPortal.resumeStreaming();
            }

            // Share the CPU.
            sleep(20);
        }
    }

    // Save more CPU resources when camera is no longer needed.
    visionPortal.close();

}   // end runOpMode()

/**
 * Initialize the TensorFlow Object Detection processor.
 */
private void initTfod() {

    // Create the TensorFlow processor by using a builder.
    tfod = new TfodProcessor.Builder()

        // Use setModelAssetName() if the TF Model is built in as an asset.
        // Use setModelFileName() if you have downloaded a custom team model to the Robot Controller.
        //.setModelAssetName(TFOD_MODEL_ASSET)
        .setModelFileName(TFOD_MODEL_FILE)

        .setModelLabels(LABEL)
        //.setIsModelTensorFlow2(true)
        //.setIsModelQuantized(true)
        //.setModelInputSize(300)
        //.setModelAspectRatio(16.0 / 9.0)

        .build();

    // Create the vision portal by using a builder.
```

```
VisionPortal.Builder builder = new VisionPortal.Builder();

// Set the camera (webcam vs. built-in RC phone camera).
if (USE_WEBCAM) {
   builder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
} else {
   builder.setCamera(BuiltinCameraDirection.BACK);
}

// Choose a camera resolution. Not all cameras support all resolutions.
//builder.setCameraResolution(new Size(640, 480));

// Enable the RC preview (LiveView).  Set "false" to omit camera monitoring.
//builder.enableCameraMonitoring(true);

// Set the stream format; MJPEG uses less bandwidth than default YUY2.
//builder.setStreamFormat(VisionPortal.StreamFormat.YUY2);

// Choose whether or not LiveView stops if no processors are enabled.
// If set "true", monitor shows solid orange screen if no processors enabled.
// If set "false", monitor shows camera view without annotations.
//builder.setAutoStopLiveView(false);

// Set and enable the processor.
builder.addProcessor(tfod);

// Build the Vision Portal, using the above settings.
visionPortal = builder.build();

// Set confidence threshold for TFOD recognitions, at any time.
//tfod.setMinResultConfidence(0.75f);

// Disable or re-enable the TFOD processor at any time.
//visionPortal.setProcessorEnabled(tfod, true);

/*List<Recognition> updatedRecognitions = tfod.getUpdatedRecognitions();
   if (updatedRecognitions != null) {
   //telemetry.addData("# Objects Detected", updatedRecognitions.size());
      for (Recognition recognition : updatedRecognitions) {
         if (recognition.getLabel().equals("Blue Warrior")) {
```

```
                telemetry.addData("OMG!! Object is finally detected", "Blue boy");
                telemetry.update();
            }

            if (recognition.getLabel().equals("Red Warrior")) {
                telemetry.addData("EEK!! Object detected hehe", "Red boy");
                telemetry.update();
            }
        }
    }
    }*/
}   // end method initTfod()

/**
 * Add telemetry about TensorFlow Object Detection (TFOD) recognitions.
 */
private void telemetryTfod() {

    List<Recognition> currentRecognitions = tfod.getRecognitions();
    telemetry.addData("# Objects Detected", currentRecognitions.size());

    // Step through the list of recognitions and display info for each one.
    for (Recognition recognition : currentRecognitions) {
        if (recognition.getLabel().equals("Blue Warrior") ){
            telemetry.addData("OMG!! Object is finally detected", "Blue boy");
            telemetry.update();
        }

        double x = (recognition.getLeft() + recognition.getRight()) / 2 ;
        double y = (recognition.getTop()  + recognition.getBottom()) / 2 ;

        telemetry.addData(""," ");
        telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(),
recognition.getConfidence() * 100);
        telemetry.addData("- Position", "%.0f / %.0f", x, y);
        telemetry.addData("- Size", "%.0f x %.0f", recognition.getWidth(),
recognition.getHeight());
    }   // end for() loop

}   // end method telemetryTfod()
```

```
}   // end class
```

RedVisAutoClose.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.hardware.Blinker;
import com.qualcomm.robotcore.hardware.HardwareDevice;
import com.qualcomm.robotcore.hardware.Gyroscope;
import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import com.qualcomm.robotcore.hardware.IMU;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;
import java.util.List;

//@Disabled
@Autonomous(name = "Red Vision Close side", group = "LinearOpMode")
public class RedVisAutoClose extends LinearOpMode{
//C: Binary variable added for the Blue Team Prop.
   boolean isBlueWarriorDetected;

   //C: Binary variable added to activate the web cam when set to true.
   boolean USE_WEBCAM;

   //C: Activating vision and setting parameters.
```

```
    TfodProcessor myTfodProcessor;
    VisionPortal myVisionPortal;

    //C: Declaring motors and servos.
    private DcMotor frontLeftDrive   = null;
    private DcMotor frontRightDrive  = null;
    private DcMotor backLeftDrive   = null;
    private DcMotor backRightDrive  = null;

    private CRServo grabberLeft;
    private CRServo grabberRight;
    private DcMotorEx armMotor;
    private CRServo pixelDrop;

    //C: OpMode runtime added.
    private ElapsedTime     runtime = new ElapsedTime();

    //C: This is the math involved in converting revolutions to inches.
    static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
    static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
    static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                                      (WHEEL_DIAMETER_INCHES * 3.1415);

    //C: The speeds the robot travels at when set.
    static final double     DRIVE_SPEED             = 0.2;
    static final double     FAST_DRIVE_SPEED        = 0.4;
    static final double     TURN_SPEED              = 0.2;
    static final double     SLOW_DRIVE_SPEED        = 0.1;
    static final double     SLOW_TURN_SPEED         = 0.2;
    static final double     FORWARD_SPEED           = 0.4;
    static final double     HEADING_THRESHOLD       = 1.0 ;

    private static final String TFOD_MODEL_FILE  = "Prop_Orbs";

    private static final String[] LABEL = {
        "Blue Orbs",
```

```
    "Red Orbs"
};


//C: The variable to store the TensorFlow Object Detection processor.
private TfodProcessor tfod;

//C: The variable to store our instance of the vision portal.
private VisionPortal visionPortal;

@Override
public void runOpMode() {

    //C: Web cam variable is set to true activating it.
    USE_WEBCAM = true;

    initTfod();

    //C: Wait for the Control Hub start button to be activated.
    telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
    telemetry.addData(">", "Touch Play to start OpMode");
    telemetry.update();

    //C: Starting the drive and other system variables.
    frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
    frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
    backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
    backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
    armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
    grabberLeft = hardwareMap.get(CRServo.class,"grabberLeft");
    grabberRight = hardwareMap.get(CRServo.class, "grabberRight");
    pixelDrop = hardwareMap.get(CRServo.class, "pixelDrop");

    //C: Setting the direction of the motors to go reverse.
    frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
    backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    backRightDrive.setDirection(DcMotor.Direction.FORWARD);

    //C: Setting the run mode to stop and reset encoders after movement.
```

```
frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

//C: Setting the run mode to move by encoders.
frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

//C: Setting the arm motor to move forward and by encoders.
armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

//C: Setting the grabber and the pixel dropper servos to move forward.
grabberLeft.setDirection(DcMotor.Direction.FORWARD);
grabberRight.setDirection(DcMotor.Direction.FORWARD);
pixelDrop.setDirection(DcMotor.Direction.FORWARD);

//C: Sends message to reveal a successful encoder reset.
telemetry.addData("Yass! Oki, Starting at",  "%7d :%7d :%7d :%7d",
    frontLeftDrive.getCurrentPosition(),
    frontRightDrive.getCurrentPosition(),
    backLeftDrive.getCurrentPosition(),
    backRightDrive.getCurrentPosition());
telemetry.update();

//C: Sounds like the name. Waiting for the opMode to begin.
waitForStart();

if (opModeIsActive()) {

    encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 15.0);

    while (opModeIsActive()) {

    List<Recognition> myTfodRecognitions;
    Recognition myTfodRecognition;
```

```
        // Get a list of recognitions from TFOD.
        myTfodRecognitions = myTfodProcessor.getRecognitions();

        for (Recognition myTfodRecognition_item : myTfodRecognitions) {
           myTfodRecognition = myTfodRecognition_item;
           float prop;

           prop = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;

           if (prop >= 330){
              telemetry.addData("OMG!! Object is finally detected", "On the right");
              telemetry.update();

              USE_WEBCAM = false;

              sixthPath();
            }

           if (prop < 330 && prop > 210){
              telemetry.addData("OMG!! Object is finally detected", "In the center");
              telemetry.update();

              USE_WEBCAM = false;

              fifthPath();
            }

           if (prop <= 210){
              telemetry.addData("OMG!! Object is finally detected", "On the left");
              telemetry.update();

              USE_WEBCAM = false;

              fourthPath();
           }
        }

        }
     }
}
```

```
public void fourthPath() {

    //C: Robot moves forward 22 inches
    //encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 12, 12, 12, 12, 8.0);

    //C: Robot strafes to the right and drops pixel and strafes back
    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -14, 14, -14, 14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, 3, 3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 25, 25, 25, 25, 20.0);

    //C: 4
    encoderDrive(SLOW_DRIVE_SPEED, -8, 8, 8, -8, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, -2, -2, -2, 10.0);

    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);
```

```
    encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);
}

public void fifthPath() {

    //C: Robot turns right for 7 inches and
    //encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 12, 12, 12, 12, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -10, 10, -10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, 10, -10, 10, -10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, -2, -2, -2, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, -14, 14, -14, 14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 25, 25, 25, 25, 20.0);

    //C: 5
    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);
```

```
    encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);

}//C: end of the second path

//C: The third path goes to the third april tag code
public void sixthPath() {

    //C: Robot turns right for 7 inches and
    //encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 12, 12, 12, 12, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, 15, -15, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -14, 14, -14, 14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 25, 25, 25, 25, 20.0);

    //C: 6
    encoderDrive(SLOW_DRIVE_SPEED, 8, -8, -8, 8, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);
```

```
        encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

        pixelScoOne();

        encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

        encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);

    }//C: end of the third path

    private void initTfod() {

        TfodProcessor.Builder myTfodProcessorBuilder;
        VisionPortal.Builder myVisionPortalBuilder;

        //C: Created a TfodProcessor.Builder.
        myTfodProcessorBuilder = new TfodProcessor.Builder();

        //C: The model file name "Original Modell" is added.
        myTfodProcessorBuilder.setModelFileName("Prop_Orbs"); //Original Modell");

        // Set the full ordered list of labels the model is trained to recognize.
        myTfodProcessorBuilder.setModelLabels(JavaUtil.createListWith("Blue Orbs", "Red
Orbs"));//"Blue Warrior", "Red Warrior"));
        //C: The aspect ratio is set to a horizontal postion (due to the higher being higher than the
bottom number)
        myTfodProcessorBuilder.setModelAspectRatio(16 / 9);
        // Create a TfodProcessor by calling build.
        myTfodProcessor = myTfodProcessorBuilder.build();
        // Next, create a VisionPortal.Builder and set attributes related to the camera.
        myVisionPortalBuilder = new VisionPortal.Builder();

        if (USE_WEBCAM) {
          // Use a webcam.
          myVisionPortalBuilder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
        } else {
            // Use the device's back camera.
            myVisionPortalBuilder.setCamera(BuiltinCameraDirection.BACK);
        }
        // Add myTfodProcessor to the VisionPortal.Builder.
```

```
        myVisionPortalBuilder.addProcessor(myTfodProcessor);
        // Create a VisionPortal by calling build.
        myVisionPortal = myVisionPortalBuilder.build();
    }

    private void telemetryTfod() {
        List<Recognition> myTfodRecognitions;
        Recognition myTfodRecognition;
        float x;
        float y;

        // Get a list of recognitions from TFOD.
        myTfodRecognitions = myTfodProcessor.getRecognitions();
        telemetry.addData("# Objects Detected", JavaUtil.listLength(myTfodRecognitions));
        // Iterate through list and call a function to display info for each recognized object.
        for (Recognition myTfodRecognition_item : myTfodRecognitions) {
        myTfodRecognition = myTfodRecognition_item;
      // Display info about the recognition.
      telemetry.addLine("");
      // Display label and confidence.
      // Display the label and confidence for the recognition.
      telemetry.addData("Image", myTfodRecognition.getLabel() + " (" +
JavaUtil.formatNumber(myTfodRecognition.getConfidence() * 100, 0) + " % Conf.)");
      // Display position.
      x = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;
      y = (myTfodRecognition.getTop() + myTfodRecognition.getBottom()) / 2;
      // Display the position of the center of the detection boundary for the recognition
      telemetry.addData("- Position", JavaUtil.formatNumber(x, 0) + ", " +
JavaUtil.formatNumber(y, 0));
      // Display size
      // Display the size of detection boundary for the recognition
      telemetry.addData("- Size", JavaUtil.formatNumber(myTfodRecognition.getWidth(), 0) + " x
" + JavaUtil.formatNumber(myTfodRecognition.getHeight(), 0));
    }
  }

    public void pixelScoOne() {

        armMotor.setPower(FORWARD_SPEED);
        sleep(1000);
```

```
    armMotor.setPower(0);
    sleep(600);

    grabberLeft.setPower(1);
    sleep(800);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(200);
    armMotor.setPower(0);

    grabberLeft.setPower(-1);
    sleep(800);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);
    armMotor.setPower(0);
}

public void pixelScoTwo() {
    grabberLeft.setPower(-1);
    grabberRight.setPower(1);
    sleep(1000);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);

    grabberLeft.setPower(1);
    grabberRight.setPower(-1);
    sleep(1000);

}

public void pixelScoThree() {
    grabberLeft.setPower(-1);
    sleep(1000);
    grabberRight.setPower(1);
    sleep(1500);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);
```

```
        grabberLeft.setPower(1);
        sleep(1000);
        grabberRight.setPower(-1);
        sleep(1500);

    }

    //function for grabber opening and closing
    public void  pixelOpen(){
        grabberLeft.setPower(-1);
        grabberRight.setPower(1);
        sleep(1000);
        grabberLeft.setPower(1);
        grabberRight.setPower(-1);
        sleep(1000);
    }

    public void  pixelClose(){
        grabberLeft.setPower(1);
        grabberRight.setPower(-1);
        sleep(1000);
        grabberLeft.setPower(-1);
        grabberRight.setPower(1);
        sleep(1000);
    }

    public void encoderDrive(double speed, double frontLeftInches, double frontRightInches,
double backLeftInches, double backRightInches, double timeouts) {

        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {
            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
```

```
        newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
        newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
        newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
        frontLeftDrive.setTargetPosition(newFrontLeftTarget);
        frontRightDrive.setTargetPosition(newFrontRightTarget);
        backLeftDrive.setTargetPosition(newBackLeftTarget);
        backRightDrive.setTargetPosition(newBackRightTarget);

        // Turn On RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
            (runtime.seconds() < timeouts) &&
            (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  "  %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);

            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
frontLeftDrive.getCurrentPosition(), frontRightDrive.getCurrentPosition(),
backLeftDrive.getCurrentPosition(), backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
```

```
            frontLeftDrive.setPower(0);
            frontRightDrive.setPower(0);
            backLeftDrive.setPower(0);
            backRightDrive.setPower(0);

            // Turn off RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

            sleep(250);   // optional pause after each move.
        }
    }
```

RedVisAutoFar.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.hardware.Blinker;
import com.qualcomm.robotcore.hardware.HardwareDevice;
import com.qualcomm.robotcore.hardware.Gyroscope;
import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import com.qualcomm.robotcore.hardware.IMU;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;
```

```
import java.util.List;

//@Disabled
@Autonomous(name = "Red Vision Far side", group = "LinearOpMode")
public class RedVisAutoFar extends LinearOpMode{
//C: Binary variable added for the Blue Team Prop.
   boolean isBlueWarriorDetected;

   //C: Binary variable added to activate the web cam when set to true.
   boolean USE_WEBCAM;

   //C: Activating vision and setting parameters.
   TfodProcessor myTfodProcessor;
   VisionPortal myVisionPortal;

   //C: Declaring motors and servos.
   private DcMotor frontLeftDrive   = null;
   private DcMotor frontRightDrive  = null;
   private DcMotor backLeftDrive   = null;
   private DcMotor backRightDrive  = null;

   private CRServo grabberLeft;
   private CRServo grabberRight;
   private DcMotorEx armMotor;
   private CRServo pixelDrop;

   //C: OpMode runtime added.
   private ElapsedTime     runtime = new ElapsedTime();

   //C: This is the math involved in converting revolutions to inches.
   static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
   static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
   static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
   static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                                 (WHEEL_DIAMETER_INCHES * 3.1415);

   //C: The speeds the robot travels at when set.
```

```
static final double     DRIVE_SPEED             = 0.2;
static final double     FAST_DRIVE_SPEED        = 0.4;
static final double     TURN_SPEED              = 0.2;
static final double     SLOW_DRIVE_SPEED        = 0.1;
static final double     SLOW_TURN_SPEED         = 0.2;
static final double     FORWARD_SPEED           = 0.4;
static final double     HEADING_THRESHOLD       = 1.0 ;

private static final String TFOD_MODEL_FILE  = "Original Modell";

private static final String[] LABEL = {
   "Red Warrior",
   "Blue Warrior"
};


//C: The variable to store the TensorFlow Object Detection processor.
private TfodProcessor tfod;

//C: The variable to store our instance of the vision portal.
private VisionPortal visionPortal;

@Override
public void runOpMode() {

   //C: Web cam variable is set to true activating it.
   USE_WEBCAM = true;

   initTfod();

   //C: Wait for the Control Hub start button to be activated.
   telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
   telemetry.addData(">", "Touch Play to start OpMode");
   telemetry.update();

   //C: Starting the drive and other system variables.
   frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
   frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
   backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
   backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
```

```
armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
grabberLeft = hardwareMap.get(CRServo.class,"grabberLeft");
grabberRight = hardwareMap.get(CRServo.class, "grabberRight");
pixelDrop = hardwareMap.get(CRServo.class, "pixelDrop");

//C: Setting the direction of the motors to go reverse.
frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

//C: Setting the run mode to stop and reset encoders after movement.
frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

//C: Setting the run mode to move by encoders.
frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

//C: Setting the arm motor to move forward and by encoders.
armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

//C: Setting the grabber and the pixel dropper servos to move forward.
grabberLeft.setDirection(DcMotor.Direction.FORWARD);
grabberRight.setDirection(DcMotor.Direction.FORWARD);
pixelDrop.setDirection(DcMotor.Direction.FORWARD);

//C: Sends message to reveal a successful encoder reset.
telemetry.addData("Yass! Oki, Starting at",  "%7d :%7d :%7d :%7d",
    frontLeftDrive.getCurrentPosition(),
    frontRightDrive.getCurrentPosition(),
    backLeftDrive.getCurrentPosition(),
    backRightDrive.getCurrentPosition());
telemetry.update();
```

```
//C: Sounds like the name. Waiting for the opMode to begin.
waitForStart();

if (opModeIsActive()) {

   while (opModeIsActive()) {

   List<Recognition> myTfodRecognitions;
   Recognition myTfodRecognition;

   // Get a list of recognitions from TFOD.
   myTfodRecognitions = myTfodProcessor.getRecognitions();

   for (Recognition myTfodRecognition_item : myTfodRecognitions) {
      myTfodRecognition = myTfodRecognition_item;
      float prop;

      prop = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;

      if (prop >= 330){
         telemetry.addData("OMG!! Object is finally detected", "On the right");
         telemetry.update();

         USE_WEBCAM = false;

         sixthPath();
       }

      if (prop < 330 && prop > 210){
         telemetry.addData("OMG!! Object is finally detected", "In the center");
         telemetry.update();

         USE_WEBCAM = false;

         fifthPath();
       }

      if (prop <= 210){
         telemetry.addData("OMG!! Object is finally detected", "On the left");
         telemetry.update();
```

```
                USE_WEBCAM = false;

                fourthPath();
            }
        }

        }
    }
}

public void fourthPath() {

    //C: Robot moves forward 22 inches
    encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 6, 6, 6, 6, 8.0);

    //C: Robot strafes to the right and drops pixel and strafes back
    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -14, 14, -14, 14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, 3, 3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);
```

```
    encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    //C: 4
    encoderDrive(SLOW_DRIVE_SPEED, -8, 8, 8, -8, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, -2, -2, -2, 10.0);

    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);
}

public void fifthPath() {

    //C: Robot turns right for 7 inches and
    encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 8, 8, 8, 8, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, 10, -10, 10, -10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, -10, 10, -10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);
```

```
    encoderDrive(SLOW_DRIVE_SPEED, -2, -2, -2, -2, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, -14, 14, -14, 14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    //C: 5
    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);

}//C: end of the second path

//C: The third path goes to the third april tag code
public void sixthPath() {

    //C: Robot turns right for 7 inches and
    encoderDrive(SLOW_DRIVE_SPEED, 16, 16, 16, 16, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 8, 8, 8, 8, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, 15, -15, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 4, -4, -4, 4, 10.0);

    pixelDrop.setPower(0.5);
```

```
    sleep(3000);

    pixelDrop.setPower(-0.5);
    sleep(1000);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, 5, -5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -14, 14, -14, 14, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    encoderDrive(DRIVE_SPEED, 59, 59, 59, 59, 20.0);

    //C: 6
    encoderDrive(SLOW_DRIVE_SPEED, 8, -8, -8, 8, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -15, 15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 10.0);

    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, 5, -5, -5, 5, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 5.0);

}//C: end of the third path

private void initTfod() {
```

```
        TfodProcessor.Builder myTfodProcessorBuilder;
        VisionPortal.Builder myVisionPortalBuilder;

        //C: Created a TfodProcessor.Builder.
        myTfodProcessorBuilder = new TfodProcessor.Builder();

        //C: The model file name "Original Modell" is added.
        myTfodProcessorBuilder.setModelFileName("Original Modell");

        // Set the full ordered list of labels the model is trained to recognize.
        myTfodProcessorBuilder.setModelLabels(JavaUtil.createListWith("Blue Warrior", "Red
Warrior"));
        //C: The aspect ratio is set to a horizontal postion (due to the higher being higher than the
bottom number)
        myTfodProcessorBuilder.setModelAspectRatio(16 / 9);
        // Create a TfodProcessor by calling build.
        myTfodProcessor = myTfodProcessorBuilder.build();
        // Next, create a VisionPortal.Builder and set attributes related to the camera.
        myVisionPortalBuilder = new VisionPortal.Builder();

        if (USE_WEBCAM) {
          // Use a webcam.
          myVisionPortalBuilder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
        } else {
            // Use the device's back camera.
            myVisionPortalBuilder.setCamera(BuiltinCameraDirection.BACK);
        }
        // Add myTfodProcessor to the VisionPortal.Builder.
        myVisionPortalBuilder.addProcessor(myTfodProcessor);
        // Create a VisionPortal by calling build.
        myVisionPortal = myVisionPortalBuilder.build();
    }

    private void telemetryTfod() {
        List<Recognition> myTfodRecognitions;
        Recognition myTfodRecognition;
        float x;
        float y;

        // Get a list of recognitions from TFOD.
```

```
        myTfodRecognitions = myTfodProcessor.getRecognitions();
        telemetry.addData("# Objects Detected", JavaUtil.listLength(myTfodRecognitions));
        // Iterate through list and call a function to display info for each recognized object.
        for (Recognition myTfodRecognition_item : myTfodRecognitions) {
        myTfodRecognition = myTfodRecognition_item;
      // Display info about the recognition.
      telemetry.addLine("");
      // Display label and confidence.
      // Display the label and confidence for the recognition.
      telemetry.addData("Image", myTfodRecognition.getLabel() + " (" +
JavaUtil.formatNumber(myTfodRecognition.getConfidence() * 100, 0) + " % Conf.)");
      // Display position.
      x = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;
      y = (myTfodRecognition.getTop() + myTfodRecognition.getBottom()) / 2;
      // Display the position of the center of the detection boundary for the recognition
      telemetry.addData("- Position", JavaUtil.formatNumber(x, 0) + ", " +
JavaUtil.formatNumber(y, 0));
      // Display size
      // Display the size of detection boundary for the recognition
      telemetry.addData("- Size", JavaUtil.formatNumber(myTfodRecognition.getWidth(), 0) + " x
" + JavaUtil.formatNumber(myTfodRecognition.getHeight(), 0));
    }
  }

    public void pixelScoOne() {

        armMotor.setPower(FORWARD_SPEED);
        sleep(1000);
        armMotor.setPower(0);
        sleep(600);

        grabberLeft.setPower(1);
        sleep(800);

        armMotor.setPower(-FORWARD_SPEED);
        sleep(200);
        armMotor.setPower(0);

        grabberLeft.setPower(-1);
        sleep(800);
```

```
    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);
    armMotor.setPower(0);
}

public void pixelScoTwo() {
    grabberLeft.setPower(-1);
    grabberRight.setPower(1);
    sleep(1000);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);

    grabberLeft.setPower(1);
    grabberRight.setPower(-1);
    sleep(1000);

}

public void pixelScoThree() {
    grabberLeft.setPower(-1);
    sleep(1000);
    grabberRight.setPower(1);
    sleep(1500);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);

    grabberLeft.setPower(1);
    sleep(1000);
    grabberRight.setPower(-1);
    sleep(1500);

}

//function for grabber opening and closing
public void  pixelOpen(){
    grabberLeft.setPower(-1);
    grabberRight.setPower(1);
```

```
        sleep(1000);
        grabberLeft.setPower(1);
        grabberRight.setPower(-1);
        sleep(1000);
    }

    public void  pixelClose(){
        grabberLeft.setPower(1);
        grabberRight.setPower(-1);
        sleep(1000);
        grabberLeft.setPower(-1);
        grabberRight.setPower(1);
        sleep(1000);
    }

    public void encoderDrive(double speed, double frontLeftInches, double frontRightInches,
double backLeftInches, double backRightInches, double timeouts) {

        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {
            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
            frontLeftDrive.setTargetPosition(newFrontLeftTarget);
            frontRightDrive.setTargetPosition(newFrontRightTarget);
            backLeftDrive.setTargetPosition(newBackLeftTarget);
            backRightDrive.setTargetPosition(newBackRightTarget);

            // Turn On RUN_TO_POSITION
```

```
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
             (runtime.seconds() < timeouts) &&
             (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  "  %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);

            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
frontLeftDrive.getCurrentPosition(), frontRightDrive.getCurrentPosition(),
backLeftDrive.getCurrentPosition(), backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);

        // Turn off RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(250);   // optional pause after each move.
```

```
        }
    }
}
```

RedVisionAuto.java

```
package Centerstage2023;

import com.qualcomm.robotcore.util.ElapsedTime;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;
import java.util.List;

@Disabled
@Autonomous(name = "Red Vision Auto", group = "LinearOpMode")

public class RedVisionAuto extends LinearOpMode{

    //private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

    boolean USE_WEBCAM;
    TfodProcessor myTfodProcessor;
    VisionPortal myVisionPortal;

        /* Declare OpMode members. */
    private DcMotor frontLeftDrive   = null;
    private DcMotor frontRightDrive  = null;
    private DcMotor backLeftDrive   = null;
```

```
    private DcMotor backRightDrive  = null;
    private CRServo grabberLeft;
    private CRServo grabberRight;
   // private DcMotor armMotor;

    private ElapsedTime     runtime = new ElapsedTime();

    private DcMotorEx armMotor;

    static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
    static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
    static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                              (WHEEL_DIAMETER_INCHES * 3.1415);
    static final double     DRIVE_SPEED             = 0.4;
    static final double     FAST_DRIVE_SPEED        = 0.6;
    static final double     TURN_SPEED              = 0.3;
    static final double     SLOW_DRIVE_SPEED        = 0.1;
    static final double     SLOW_TURN_SPEED         = 0.2;
    static final double     FORWARD_SPEED           = 0.4;

    private static final String TFOD_MODEL_FILE  = "Original Modell";

    private static final String[] LABEL = {
        "Red Warrior",
        "Blue Warrior"
    };

    //static final double TURN_SPEED = 0.5;

   // private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

    /**
     * The variable to store our instance of the TensorFlow Object Detection processor.
     */
    private TfodProcessor tfod;
```

```
/**
 * The variable to store our instance of the vision portal.
 */
private VisionPortal visionPortal;

@Override
public void runOpMode() {

    USE_WEBCAM = true;

    initTfod();
    telemetryTfod();
    telemetry.update();

    // Wait for the DS start button to be touched.
    telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
    telemetry.addData(">", "Touch Play to start OpMode");
    telemetry.update();

        // Initialize the drive system variables.
    frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
    frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
    backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
    backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
    armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
    grabberLeft = hardwareMap.get(CRServo.class,"grabberLeft");
    grabberRight = hardwareMap.get(CRServo.class, "grabberRight");

    frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
    backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    backRightDrive.setDirection(DcMotor.Direction.FORWARD);

    frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

    frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
```

```
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
grabberLeft.setDirection(DcMotor.Direction.FORWARD);
grabberRight.setDirection(DcMotor.Direction.FORWARD);

// Send telemetry message to indicate successful Encoder reset
telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d :%7d :%7d",
                  frontLeftDrive.getCurrentPosition(),
                  frontRightDrive.getCurrentPosition(),
                  backLeftDrive.getCurrentPosition(),
                  backRightDrive.getCurrentPosition());
telemetry.update();

waitForStart();

if (opModeIsActive()) {

   encoderDrive(SLOW_DRIVE_SPEED, 12, 12, 12, 12, 15.0);

   encoderDrive(SLOW_DRIVE_SPEED, 7, -7, 7, -7, 10.0);

   searchTime(4.0);

   List<Recognition> myTfodRecognitions;
   Recognition myTfodRecognition;

   // Get a list of recognitions from TFOD.
   myTfodRecognitions = myTfodProcessor.getRecognitions();

   if (JavaUtil.listLength(myTfodRecognitions) == 1) {
      telemetry.addData("OMG!! Object is finally detected", "Red soldier");
      telemetry.update();

      fourthPath();

   } else {
```

```
            encoderDrive(SLOW_DRIVE_SPEED, -5, 5, -5, 5, 10.0);

            lookToSeeTwo();

        }
    }
}


public void lookToSeeTwo() {

    searchTime(4.0);

    List<Recognition> myTfodRecognitions;
    Recognition myTfodRecognition;

    // Get a list of recognitions from TFOD.
    myTfodRecognitions = myTfodProcessor.getRecognitions();

    if (JavaUtil.listLength(myTfodRecognitions) == 1) {
        telemetry.addData("OMG!! Object is finally detected", "Red soldier");
        telemetry.update();

        fifthPath();

    } else {

        encoderDrive(SLOW_DRIVE_SPEED, -6, 6, -6, 6, 10.0);

        lookToSeeThree();

    }
}

public void lookToSeeThree() {

    searchTime(4.0);

    List<Recognition> myTfodRecognitions;
```

```
    Recognition myTfodRecognition;

    // Get a list of recognitions from TFOD.
    myTfodRecognitions = myTfodProcessor.getRecognitions();

    if (JavaUtil.listLength(myTfodRecognitions) == 1) {
        telemetry.addData("OMG!! Object is finally detected", "Red soldier");
        telemetry.update();

        sixthPath();

    } else {
        telemetry.addData("Uh oh", "nothing here");
        telemetry.update();

        fifthPath();
    }
}

public void searchTime(double timeouts) {

    if (opModeIsActive() && (runtime.seconds() < timeouts)) {

        initTfod();

        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
    }

    initTfod();
}

public void fourthPath() {

    encoderDrive(SLOW_DRIVE_SPEED, -5, 5, -5, 5, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -6, 6, -6, 6, 10.0);
```

```
    encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, -11, -11, -11, -11, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 20, -20, -20, 20, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, 15, -15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 12, -12, -12, 12, 12.0);

    encoderDrive(SLOW_DRIVE_SPEED, 7, -7, -7, 7, 8.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);

    armMotor.setPower(FORWARD_SPEED);
    sleep(1000);

    pixelScoOne();

    encoderDrive(SLOW_DRIVE_SPEED, -16, 16, -16, 16, 16.0);
}//C: fourth path ended

//C: the fifth path scores the fifth april tag
public void fifthPath() {

    encoderDrive(SLOW_DRIVE_SPEED, -6, 6, -6, 6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, -11, -11, -11, -11, 15.0);
```

```
    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 20, -20, -20, 20, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, 15, -15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 12, -12, -12, 12, 12.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);

    armMotor.setPower(FORWARD_SPEED);
    sleep(1000);

    pixelScoTwo();
}//C: fifth path ended

public void sixthPath() {

    encoderDrive(SLOW_DRIVE_SPEED, 6, -6, 6, -6, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, -11, -11, -11, -11, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, -15, 15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, 20, -20, -20, 20, 20.0);

    encoderDrive(SLOW_DRIVE_SPEED, 10, 10, 10, 10, 10.0);

    encoderDrive(SLOW_DRIVE_SPEED, 15, -15, 15, -15, 15.0);

    encoderDrive(SLOW_DRIVE_SPEED, -3, -3, -3, -3, 5.0);

    encoderDrive(SLOW_DRIVE_SPEED, 2, -2, 2, -2, 8.0);
```

```
        armMotor.setPower(FORWARD_SPEED);
        sleep(1000);

        pixelScoThree();
    }

    private void initTfod() {
        TfodProcessor.Builder myTfodProcessorBuilder;
        VisionPortal.Builder myVisionPortalBuilder;

        // First, create a TfodProcessor.Builder.
        myTfodProcessorBuilder = new TfodProcessor.Builder();
        // Set the name of the file where the model can be found.
        myTfodProcessorBuilder.setModelFileName("Original Modell");
        // Set the full ordered list of labels the model is trained to recognize.
        myTfodProcessorBuilder.setModelLabels(JavaUtil.createListWith("Blue Warrior", "Red
Warrior"));
        // Set the aspect ratio for the images used when the model was created.
        myTfodProcessorBuilder.setModelAspectRatio(16 / 9);
        // Create a TfodProcessor by calling build.
        myTfodProcessor = myTfodProcessorBuilder.build();
        // Next, create a VisionPortal.Builder and set attributes related to the camera.
        myVisionPortalBuilder = new VisionPortal.Builder();
        if (USE_WEBCAM) {
          // Use a webcam.
          myVisionPortalBuilder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
        } else {
            // Use the device's back camera.
            myVisionPortalBuilder.setCamera(BuiltinCameraDirection.BACK);
        }
        // Add myTfodProcessor to the VisionPortal.Builder.
        myVisionPortalBuilder.addProcessor(myTfodProcessor);
        // Create a VisionPortal by calling build.
        myVisionPortal = myVisionPortalBuilder.build();
    }

    private void telemetryTfod() {
        List<Recognition> myTfodRecognitions;
        Recognition myTfodRecognition;
```

```
        float x;
        float y;

        // Get a list of recognitions from TFOD.
        myTfodRecognitions = myTfodProcessor.getRecognitions();
        telemetry.addData("# Objects Detected", JavaUtil.listLength(myTfodRecognitions));
        // Iterate through list and call a function to display info for each recognized object.
        for (Recognition myTfodRecognition_item : myTfodRecognitions) {
          myTfodRecognition = myTfodRecognition_item;
          // Display info about the recognition.
          telemetry.addLine("");
          // Display label and confidence.
          // Display the label and confidence for the recognition.
          telemetry.addData("Image", myTfodRecognition.getLabel() + " (" +
JavaUtil.formatNumber(myTfodRecognition.getConfidence() * 100, 0) + " % Conf.)");
          // Display position.
          x = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;
          y = (myTfodRecognition.getTop() + myTfodRecognition.getBottom()) / 2;
          // Display the position of the center of the detection boundary for the recognition
          telemetry.addData("- Position", JavaUtil.formatNumber(x, 0) + ", " +
JavaUtil.formatNumber(y, 0));
          // Display size
          // Display the size of detection boundary for the recognition
          telemetry.addData("- Size", JavaUtil.formatNumber(myTfodRecognition.getWidth(), 0) +
" x " + JavaUtil.formatNumber(myTfodRecognition.getHeight(), 0));
        }
      }

      public void pixelScoOne() {
        grabberLeft.setPower(-1);
        sleep(1500);
        grabberRight.setPower(1);
        sleep(1000);

        armMotor.setPower(-FORWARD_SPEED);
        sleep(1000);

        grabberLeft.setPower(1);
        sleep(1500);
        grabberRight.setPower(-1);
```

```
    sleep(1000);

}

public void pixelScoTwo() {
    grabberLeft.setPower(-1);
    grabberRight.setPower(1);
    sleep(1000);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);

    grabberLeft.setPower(1);
    grabberRight.setPower(-1);
    sleep(1000);

}

public void pixelScoThree() {
    grabberLeft.setPower(-1);
    sleep(1000);
    grabberRight.setPower(1);
    sleep(1500);

    armMotor.setPower(-FORWARD_SPEED);
    sleep(1000);

    grabberLeft.setPower(1);
    //sleep(1000);
    grabberRight.setPower(-1);
    sleep(1500);

}

//function for grabber opening and closing
public void  pixelOpen(){
    grabberLeft.setPower(-1);
    grabberRight.setPower(1);
    sleep(1000);
```

```
        grabberLeft.setPower(1);
        grabberRight.setPower(-1);
        sleep(1000);
    }

    public void  pixelClose(){
        grabberLeft.setPower(1);
        grabberRight.setPower(-1);
        sleep(1000);

        grabberLeft.setPower(-1);
        grabberRight.setPower(1);
        sleep(1000);
    }

    public void encoderDrive(double speed,
                             double frontLeftInches, double frontRightInches,
                             double backLeftInches, double backRightInches,
                             double timeouts) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {
            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
            frontLeftDrive.setTargetPosition(newFrontLeftTarget);
            frontRightDrive.setTargetPosition(newFrontRightTarget);
            backLeftDrive.setTargetPosition(newBackLeftTarget);
            backRightDrive.setTargetPosition(newBackRightTarget);
```

```
        // Turn On RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
            (runtime.seconds() < timeouts) &&
            (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  "  %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
                                frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
                                backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
           //armMotor.setPower(0);

           // Turn off RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
```

```
        backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(250);   // optional pause after each move.
    }
}
}
```

RightAutoPark.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;

import java.util.List;

@Autonomous(name = "Right Auto Park", group = "LinearOpMode")
@Disabled
public class RightAutoPark extends LinearOpMode {

    private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

        /* Declare OpMode members. */
    private DcMotor         frontLeftDrive   = null;
    private DcMotor         frontRightDrive  = null;
    private DcMotor         backLeftDrive    = null;
    private DcMotor         backRightDrive  = null;

    private DcMotorEx armMotor;
```

```
   static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
   static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
   static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
   static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                                 (WHEEL_DIAMETER_INCHES * 3.1415);
   static final double     DRIVE_SPEED             = 0.4;
   static final double     FAST_DRIVE_SPEED            = 0.6;
   static final double     TURN_SPEED              = 0.3;
   static final double     SLOW_DRIVE_SPEED            = 0.2;
   static final double     SLOW_TURN_SPEED             = 0.2;

   /**
    * The variable to store our instance of the TensorFlow Object Detection processor.
    */
   private TfodProcessor tfod;

   private ElapsedTime     runtime = new ElapsedTime();

   /**
    * The variable to store our instance of the vision portal.
    */
   private VisionPortal visionPortal;

   @Override
   public void runOpMode() {

      initTfod();

      // Wait for the DS start button to be touched.
      telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
      telemetry.addData(">", "Touch Play to start OpMode");
      telemetry.update();

            // Initialize the drive system variables.
      frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
      frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
```

```
backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");

frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

// Send telemetry message to indicate successful Encoder reset
telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d: %7d :%7d",
                  frontLeftDrive.getCurrentPosition(),
                  frontRightDrive.getCurrentPosition(),
                  backLeftDrive.getCurrentPosition(),
                  backRightDrive.getCurrentPosition());
telemetry.update();

waitForStart();

if (opModeIsActive()) {

   encoderDrive(SLOW_DRIVE_SPEED, 25, -25, -25, 25, 5.0);

   while (opModeIsActive()) {

      //encoderDrive(SLOW_DRIVE_SPEED, 25, -25, -25, 25, 1.0);
```

```
            //List<Recognition> currentRecognitions = tfod.getRecognitions();
            //telemetry.addData("# Objects Detected", currentRecognitions.size());
            /*if (tfod != null) {
            List<Recognition> currentRecognitions = tfod.getRecognitions();
               if (currentRecognitions != null) {
                  //telemetry.addData("# Objects Detected", updatedRecognitions.size());

                  for (Recognition recognition : currentRecognitions) {
                     encoderDrive(SLOW_DRIVE_SPEED, 25, -25, -25, 25, 1.0);
                  }
               }
            }*/

            //telemetryTfod();

            // Push telemetry to the Driver Station.
            //telemetry.update();

            // Save CPU resources; can resume streaming when needed.
            /*if (gamepad1.dpad_down) {
               visionPortal.stopStreaming();
            } else if (gamepad1.dpad_up) {
               visionPortal.resumeStreaming();
            }

            // Share the CPU.
            sleep(20);*/
         }
      }

      // Save more CPU resources when camera is no longer needed.
      visionPortal.close();

   }   // end runOpMode()


   /**
    * Initialize the TensorFlow Object Detection processor.
    */
   private void initTfod() {
```

```
        // Create the TensorFlow processor by using a builder.
        tfod = new TfodProcessor.Builder()

            // Use setModelAssetName() if the TF Model is built in as an asset.
            // Use setModelFileName() if you have downloaded a custom team model to the Robot
Controller.
            //.setModelAssetName(TFOD_MODEL_ASSET)
            //.setModelFileName(TFOD_MODEL_FILE)

            //.setModelLabels(LABELS)
            //.setIsModelTensorFlow2(true)
            //.setIsModelQuantized(true)
            //.setModelInputSize(300)
            //.setModelAspectRatio(16.0 / 9.0)

            .build();

        // Create the vision portal by using a builder.
        VisionPortal.Builder builder = new VisionPortal.Builder();

        // Set the camera (webcam vs. built-in RC phone camera).
        if (USE_WEBCAM) {
            builder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
        } else {
            builder.setCamera(BuiltinCameraDirection.BACK);
        }

        // Choose a camera resolution. Not all cameras support all resolutions.
        //builder.setCameraResolution(new Size(640, 480));

        // Enable the RC preview (LiveView).  Set "false" to omit camera monitoring.
        //builder.enableCameraMonitoring(true);

        // Set the stream format; MJPEG uses less bandwidth than default YUY2.
        //builder.setStreamFormat(VisionPortal.StreamFormat.YUY2);

        // Choose whether or not LiveView stops if no processors are enabled.
        // If set "true", monitor shows solid orange screen if no processors enabled.
        // If set "false", monitor shows camera view without annotations.
```

```
        //builder.setAutoStopLiveView(false);

        // Set and enable the processor.
        builder.addProcessor(tfod);

        // Build the Vision Portal, using the above settings.
        visionPortal = builder.build();

        // Set confidence threshold for TFOD recognitions, at any time.
        //tfod.setMinResultConfidence(0.75f);

        // Disable or re-enable the TFOD processor at any time.
        //visionPortal.setProcessorEnabled(tfod, true);

    }   // end method initTfod()

    /**
     * Add telemetry about TensorFlow Object Detection (TFOD) recognitions.
     */
    private void telemetryTfod() {

        List<Recognition> currentRecognitions = tfod.getRecognitions();
        telemetry.addData("# Objects Detected", currentRecognitions.size());

        // Step through the list of recognitions and display info for each one.
        for (Recognition recognition : currentRecognitions) {
            double x = (recognition.getLeft() + recognition.getRight()) / 2 ;
            double y = (recognition.getTop()  + recognition.getBottom()) / 2 ;

            telemetry.addData(""," ");
            telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(),
recognition.getConfidence() * 100);
            telemetry.addData("- Position", "%.0f / %.0f", x, y);
            telemetry.addData("- Size", "%.0f x %.0f", recognition.getWidth(),
recognition.getHeight());
        }   // end for() loop

    }   // end method telemetryTfod()

    public void encoderDrive(double speed,
```

```
                    double frontLeftInches, double frontRightInches,
                    double backLeftInches, double backRightInches,
                    double timeoutS) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {

            //C: Ensures that whle the drive mode is active the arm does not fall down
            //R: YOU IDIOT I CANNOT BELIEVE YOU MADE THE POWER EIGHT!??!?!?!?!
GRAHHHHHHHHHHHHHHHH
            //armMotor.setPower(0.5);

            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
            frontLeftDrive.setTargetPosition(newFrontLeftTarget);
            frontRightDrive.setTargetPosition(newFrontRightTarget);
            backLeftDrive.setTargetPosition(newBackLeftTarget);
            backRightDrive.setTargetPosition(newBackRightTarget);

            // Turn On RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            // reset the timeout time and start motion.
            runtime.reset();
            frontLeftDrive.setPower(Math.abs(speed));
            frontRightDrive.setPower(Math.abs(speed));
```

```
            backLeftDrive.setPower(Math.abs(speed));
            backRightDrive.setPower(Math.abs(speed));

            while (opModeIsActive() &&
                   (runtime.seconds() < timeoutS) &&
                   (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

                // Display it for the driver.
                telemetry.addData("Running to",  " %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
                telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
                                  frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
                                  backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
                telemetry.update();
            }

            // Stop all motion;
            frontLeftDrive.setPower(0);
            frontRightDrive.setPower(0);
            backLeftDrive.setPower(0);
            backRightDrive.setPower(0);
            //armMotor.setPower(0);

            // Turn off RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

            //sleep(250);   // optional pause after each move.
        }
    }
}   // end class
```

TfodAutoDriveTest.java

```
package Cam_auto_2023;
```

```
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.util.ElapsedTime;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;

import java.util.List;

/*
 * This OpMode illustrates the basics of TensorFlow Object Detection,
 * including Java Builder structures for specifying Vision parameters.
 *
 * Use Android Studio to Copy this Class, and Paste it into your team's code folder with a new name.
 * Remove or comment out the @Disabled line to add this OpMode to the Driver Station OpMode list.
 */
@Autonomous(name = "Testing the moving", group = "LinearOpMode")
@Disabled
public class TfodAutoDriveTest extends LinearOpMode {

    private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone camera

        /* Declare OpMode members. */
    private DcMotor         frontLeftDrive   = null;
    private DcMotor         frontRightDrive  = null;
    private DcMotor         backLeftDrive   = null;
    private DcMotor         backRightDrive  = null;

    private DcMotorEx armMotor;
```

```
   static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
   static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
   static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
   static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                             (WHEEL_DIAMETER_INCHES * 3.1415);
   static final double     DRIVE_SPEED             = 0.4;
   static final double     FAST_DRIVE_SPEED            = 0.6;
   static final double     TURN_SPEED              = 0.3;
   static final double     SLOW_DRIVE_SPEED            = 0.1;
   static final double     SLOW_TURN_SPEED             = 0.2;

   /**
    * The variable to store our instance of the TensorFlow Object Detection processor.
    */
   private TfodProcessor tfod;

   private ElapsedTime     runtime = new ElapsedTime();

   /**
    * The variable to store our instance of the vision portal.
    */
   private VisionPortal visionPortal;

   @Override
   public void runOpMode() {

      initTfod();

      // Wait for the DS start button to be touched.
      telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
      telemetry.addData(">", "Touch Play to start OpMode");
      telemetry.update();

            // Initialize the drive system variables.
      frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
      frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
      backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
```

```
backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");

frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

// Send telemetry message to indicate successful Encoder reset
telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d",
                  frontLeftDrive.getCurrentPosition(),
                  frontRightDrive.getCurrentPosition(),
                  backLeftDrive.getCurrentPosition(),
                  backRightDrive.getCurrentPosition());
telemetry.update();

waitForStart();

if (opModeIsActive()) {

encoderDrive(SLOW_DRIVE_SPEED, 20, 20, 20, 20, 8.0);

encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 8.0);

encoderDrive(SLOW_DRIVE_SPEED, -25, -25, -25, -25, 8.0);

encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 8.0);
```

```
encoderDrive(SLOW_DRIVE_SPEED, -6, -6, -6, -6, 8.0);

encoderDrive(SLOW_DRIVE_SPEED, -1, 1, -1, 1, 8.0);

encoderDrive(SLOW_DRIVE_SPEED, -8, 8, 8, -8, 8.0);

encoderDrive(SLOW_DRIVE_SPEED, -2, 2, -2, 2, 8.0);




   while (opModeIsActive()) {

/*encoderDrive(SLOW_DRIVE_SPEED, 20, 20, 20, 20, 8.0);

encoderDrive(SLOW_DRIVE_SPEED, -12, 12, -12, 12, 8.0);

encoderDrive(SLOW_DRIVE_SPEED, -22, -22, -22, -22, 8.0);*/

//encoderDrive(SLOW_DRIVE_SPEED, 15, 15, 15, 15, 8.0);

      /*
      //List<Recognition> currentRecognitions = tfod.getRecognitions();
      //telemetry.addData("# Objects Detected", currentRecognitions.size());
      if (tfod != null) {
      List<Recognition> currentRecognitions = tfod.getRecognitions();
         if (currentRecognitions != null) {
            //telemetry.addData("# Objects Detected", updatedRecognitions.size());

            for (Recognition recognition : currentRecognitions) {
               encoderDrive(SLOW_DRIVE_SPEED, 5, 5, 5, 5, 1.0);
            }
         }
      }*/

      telemetryTfod();
```

```
            // Push telemetry to the Driver Station.
            telemetry.update();

            // Save CPU resources; can resume streaming when needed.
            if (gamepad1.dpad_down) {
                visionPortal.stopStreaming();
            } else if (gamepad1.dpad_up) {
                visionPortal.resumeStreaming();
            }

            // Share the CPU.
            sleep(20);
        }
    }

    // Save more CPU resources when camera is no longer needed.
    visionPortal.close();

}   // end runOpMode()


/**
 * Initialize the TensorFlow Object Detection processor.
 */
private void initTfod() {

    // Create the TensorFlow processor by using a builder.
    tfod = new TfodProcessor.Builder()

        // Use setModelAssetName() if the TF Model is built in as an asset.
        // Use setModelFileName() if you have downloaded a custom team model to the Robot Controller.
        //.setModelAssetName(TFOD_MODEL_ASSET)
        //.setModelFileName(TFOD_MODEL_FILE)

        //.setModelLabels(LABELS)
        //.setIsModelTensorFlow2(true)
        //.setIsModelQuantized(true)
        //.setModelInputSize(300)
        //.setModelAspectRatio(16.0 / 9.0)
```

```
        .build();

    // Create the vision portal by using a builder.
    VisionPortal.Builder builder = new VisionPortal.Builder();

    // Set the camera (webcam vs. built-in RC phone camera).
    if (USE_WEBCAM) {
        builder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
    } else {
        builder.setCamera(BuiltinCameraDirection.BACK);
    }

    // Choose a camera resolution. Not all cameras support all resolutions.
    //builder.setCameraResolution(new Size(640, 480));

    // Enable the RC preview (LiveView).  Set "false" to omit camera monitoring.
    //builder.enableCameraMonitoring(true);

    // Set the stream format; MJPEG uses less bandwidth than default YUY2.
    //builder.setStreamFormat(VisionPortal.StreamFormat.YUY2);

    // Choose whether or not LiveView stops if no processors are enabled.
    // If set "true", monitor shows solid orange screen if no processors enabled.
    // If set "false", monitor shows camera view without annotations.
    //builder.setAutoStopLiveView(false);

    // Set and enable the processor.
    builder.addProcessor(tfod);

    // Build the Vision Portal, using the above settings.
    visionPortal = builder.build();

    // Set confidence threshold for TFOD recognitions, at any time.
    //tfod.setMinResultConfidence(0.75f);

    // Disable or re-enable the TFOD processor at any time.
    //visionPortal.setProcessorEnabled(tfod, true);

}   // end method initTfod()
```

```
    /**
     * Add telemetry about TensorFlow Object Detection (TFOD) recognitions.
     */
    private void telemetryTfod() {

        List<Recognition> currentRecognitions = tfod.getRecognitions();
        telemetry.addData("# Objects Detected", currentRecognitions.size());

        // Step through the list of recognitions and display info for each one.
        for (Recognition recognition : currentRecognitions) {
            double x = (recognition.getLeft() + recognition.getRight()) / 2 ;
            double y = (recognition.getTop()  + recognition.getBottom()) / 2 ;

            telemetry.addData(""," ");
            telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(),
recognition.getConfidence() * 100);
            telemetry.addData("- Position", "%.0f / %.0f", x, y);
            telemetry.addData("- Size", "%.0f x %.0f", recognition.getWidth(),
recognition.getHeight());
        }   // end for() loop

    }   // end method telemetryTfod()

    public void encoderDrive(double speed,
                             double frontLeftInches, double frontRightInches,
                             double backLeftInches, double backRightInches,
                             double timeoutS) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {

            //C: Ensures that whle the drive mode is active the arm does not fall down
            //R: YOU IDIOT I CANNOT BELIEVE YOU MADE THE POWER EIGHT!??!?!?!?!
GRAHHHHHHHHHHHHHHHH
            //armMotor.setPower(0.5);
```

```
        // Determine new target position, and pass to motor controller
        newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
        newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
        newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
        newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
        frontLeftDrive.setTargetPosition(newFrontLeftTarget);
        frontRightDrive.setTargetPosition(newFrontRightTarget);
        backLeftDrive.setTargetPosition(newBackLeftTarget);
        backRightDrive.setTargetPosition(newBackRightTarget);

        // Turn On RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
             (runtime.seconds() < timeoutS) &&
             (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  " %7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d",
                                frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
```

```
                                backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
        //armMotor.setPower(0);

        // Turn off RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(550);   // optional pause after each move.
      }
   }
}   // end class
```

TfodAutoDriveTest3.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
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.DcMotorEx;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;
import com.qualcomm.robotcore.util.ElapsedTime;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;

import java.util.List;
@Autonomous(name = "Test the motors", group = "LinearOpMode")
```

```
@Disabled
public class TfodAutoDriveTest3 extends LinearOpMode {

   private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

         /* Declare OpMode members. */
   private DcMotor         frontLeftDrive   = null;
   private DcMotor         frontRightDrive  = null;
   private DcMotor         backLeftDrive   = null;
   private DcMotor         backRightDrive  = null;

   private DcMotorEx armMotor;

   static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
   static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
   static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
   static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                             (WHEEL_DIAMETER_INCHES * 3.1415);
   static final double     DRIVE_SPEED             = 0.4;
   static final double     FAST_DRIVE_SPEED            = 0.6;
   static final double     TURN_SPEED              = 0.3;
   static final double     SLOW_DRIVE_SPEED            = 0.2;
   static final double     SLOW_TURN_SPEED             = 0.2;

   /**
    * The variable to store our instance of the TensorFlow Object Detection processor.
    */
   private TfodProcessor tfod;

   private ElapsedTime     runtime = new ElapsedTime();

   /**
    * The variable to store our instance of the vision portal.
    */
   private VisionPortal visionPortal;
```

```
@Override
public void runOpMode() {

    initTfod();

    // Wait for the DS start button to be touched.
    telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
    telemetry.addData(">", "Touch Play to start OpMode");
    telemetry.update();

        // Initialize the drive system variables.
    frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
    frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
    backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
    backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
    armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");

    frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
    backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    backRightDrive.setDirection(DcMotor.Direction.FORWARD);

    frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

    frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

    armMotor.setDirection(DcMotor.Direction.FORWARD);
    armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

    // Send telemetry message to indicate successful Encoder reset
    telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d :%7d :%7d",
                      frontLeftDrive.getCurrentPosition(),
                      frontRightDrive.getCurrentPosition(),
                      backLeftDrive.getCurrentPosition(),
```

```
                backRightDrive.getCurrentPosition());
telemetry.update();

waitForStart();

if (opModeIsActive()) {
   while (opModeIsActive()) {

      encoderDrive(SLOW_DRIVE_SPEED, 1000, 1000, 1000, 1000, 1.0);

      //List<Recognition> currentRecognitions = tfod.getRecognitions();
      //telemetry.addData("# Objects Detected", currentRecognitions.size());
      /*if (tfod != null) {
      List<Recognition> currentRecognitions = tfod.getRecognitions();
         if (currentRecognitions != null) {
            //telemetry.addData("# Objects Detected", updatedRecognitions.size());

            for (Recognition recognition : currentRecognitions) {
               encoderDrive(SLOW_DRIVE_SPEED, 5, 5, 5, 5, 1.0);
            }
         }
      }*/

      telemetryTfod();

      // Push telemetry to the Driver Station.
      telemetry.update();

      // Save CPU resources; can resume streaming when needed.
      /*if (gamepad1.dpad_down) {
         visionPortal.stopStreaming();
      } else if (gamepad1.dpad_up) {
         visionPortal.resumeStreaming();
      }

      // Share the CPU.
      sleep(20);
   }*/
}
```

```
        // Save more CPU resources when camera is no longer needed.
        visionPortal.close();

    }   // end runOpMode()
    }


    /**
     * Initialize the TensorFlow Object Detection processor.
     */
    private void initTfod() {

        // Create the TensorFlow processor by using a builder.
        tfod = new TfodProcessor.Builder()

            // Use setModelAssetName() if the TF Model is built in as an asset.
            // Use setModelFileName() if you have downloaded a custom team model to the Robot
Controller.
            //.setModelAssetName(TFOD_MODEL_ASSET)
            //.setModelFileName(TFOD_MODEL_FILE)

            //.setModelLabels(LABELS)
            //.setIsModelTensorFlow2(true)
            //.setIsModelQuantized(true)
            //.setModelInputSize(300)
            //.setModelAspectRatio(16.0 / 9.0)

            .build();

        // Create the vision portal by using a builder.
        VisionPortal.Builder builder = new VisionPortal.Builder();

        // Set the camera (webcam vs. built-in RC phone camera).
        if (USE_WEBCAM) {
            builder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
        } else {
            //builder.setCamera(BuiltinCameraDirection.BACK);
        }

        // Choose a camera resolution. Not all cameras support all resolutions.
```

```
    //builder.setCameraResolution(new Size(640, 480));

    // Enable the RC preview (LiveView).  Set "false" to omit camera monitoring.
    //builder.enableCameraMonitoring(true);

    // Set the stream format; MJPEG uses less bandwidth than default YUY2.
    //builder.setStreamFormat(VisionPortal.StreamFormat.YUY2);

    // Choose whether or not LiveView stops if no processors are enabled.
    // If set "true", monitor shows solid orange screen if no processors enabled.
    // If set "false", monitor shows camera view without annotations.
    //builder.setAutoStopLiveView(false);

    // Set and enable the processor.
    builder.addProcessor(tfod);

    // Build the Vision Portal, using the above settings.
    visionPortal = builder.build();

    // Set confidence threshold for TFOD recognitions, at any time.
    //tfod.setMinResultConfidence(0.75f);

    // Disable or re-enable the TFOD processor at any time.
    //visionPortal.setProcessorEnabled(tfod, true);

}   // end method initTfod()

/**
 * Add telemetry about TensorFlow Object Detection (TFOD) recognitions.
 */
private void telemetryTfod() {

    List<Recognition> currentRecognitions = tfod.getRecognitions();
    telemetry.addData("# Objects Detected", currentRecognitions.size());

    // Step through the list of recognitions and display info for each one.
    for (Recognition recognition : currentRecognitions) {
        double x = (recognition.getLeft() + recognition.getRight()) / 2 ;
        double y = (recognition.getTop()  + recognition.getBottom()) / 2 ;
```

```
            telemetry.addData("","  ");
            telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(),
recognition.getConfidence() * 100);
            telemetry.addData("- Position", "%.0f / %.0f", x, y);
            telemetry.addData("- Size", "%.0f x %.0f", recognition.getWidth(),
recognition.getHeight());
        }   // end for() loop

    }   // end method telemetryTfod()

    public void encoderDrive(double speed,
                             double frontLeftInches, double frontRightInches,
                             double backLeftInches, double backRightInches,
                             double timeoutS) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {

            //C: Ensures that whle the drive mode is active the arm does not fall down
            //R: YOU IDIOT I CANNOT BELIEVE YOU MADE THE POWER EIGHT!??!?!?!?!
GRAHHHHHHHHHHHHHHHH
            //armMotor.setPower(0.5);

            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
            frontLeftDrive.setTargetPosition(newFrontLeftTarget);
            frontRightDrive.setTargetPosition(newFrontRightTarget);
            backLeftDrive.setTargetPosition(newBackLeftTarget);
            backRightDrive.setTargetPosition(newBackRightTarget);
```

```
        // Turn On RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
            (runtime.seconds() < timeoutS) &&
            (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  " %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
                                frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
                                backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
        //armMotor.setPower(0);

        // Turn off RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
```

```
            backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

            //sleep(250);   // optional pause after each move.
        }
    }
}   // end class
```

TfodEasy.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;

import java.util.List;

@Autonomous(name = "Concept: TensorFlow Object Detection Easy", group =
"LinearOpMode")
@Disabled
public class TfodEasy extends LinearOpMode {

    private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

    /**
     * The variable to store our instance of the TensorFlow Object Detection processor.
     */
    private TfodProcessor tfod;

    /**
     * The variable to store our instance of the vision portal.
     */
```

```
private VisionPortal visionPortal;


@Override
public void runOpMode() {

   initTfod();

   // Wait for the DS start button to be touched.
   telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
   telemetry.addData(">", "Touch Play to start OpMode");
   telemetry.update();
   waitForStart();

   if (opModeIsActive()) {
      while (opModeIsActive()) {

         //telemetryTfod();

         //initTfod();

         // Push telemetry to the Driver Station.
         telemetry.update();
         //.setModelAspectRatio(16.0 / 9.0);
         //tfod.setZoom(1.5, 16.0/9.0);
         List<Recognition> currentRecognitions = tfod.getRecognitions();
         //telemetry.addData("# Objects Detected", currentRecognitions.size());

         for (Recognition recognition : currentRecognitions) {
            double A = (recognition.getLeft());
            double B = (recognition.getRight());
            double C = (recognition.getTop());

            if (A > B){
               telemetry.addData("It's to the left buddy", "hehe");
               telemetry.update();
            }

            if (B > A){
               telemetry.addData("It's to the middle buddy", "hehe");
```

```
            telemetry.update();
        }

        if (B > C){
            telemetry.addData("It's to the Middle buddy", "hehe");
            telemetry.update();
        }

        if (C > B){
            telemetry.addData("It's to the right buddy", "hehe");
            telemetry.update();
        }

        if (C > A){
            telemetry.addData("It's to the Right buddy", "hehe");
            telemetry.update();
        }

        if (A > C){
            telemetry.addData("It's to the left buddy", "hehe");
            telemetry.update();
        }
    }

    // Save CPU resources; can resume streaming when needed.
    /*if (gamepad1.dpad_down) {
        visionPortal.stopStreaming();
    } else if (gamepad1.dpad_up) {
        visionPortal.resumeStreaming();
    }

    // Share the CPU.
    sleep(20);*/
  }
}

// Save more CPU resources when camera is no longer needed.
visionPortal.close();

}   // end runOpMode()
```

```
private void foundLeft() {
    telemetry.update();
    telemetry.addData("Its to the left", "We gott'him");

}

/**
 * Initialize the TensorFlow Object Detection processor.
 */
private void initTfod() {

    // Create the TensorFlow processor the easy way.
    tfod = TfodProcessor.easyCreateWithDefaults();

    // Create the vision portal the easy way.
    if (USE_WEBCAM) {
        visionPortal = VisionPortal.easyCreateWithDefaults(
            hardwareMap.get(WebcamName.class, "Webcam 1"), tfod);
    } else {
        visionPortal = VisionPortal.easyCreateWithDefaults(
            BuiltinCameraDirection.BACK, tfod);
    }



    //tfod2 = new TfodProcessor.Builder()

    //.setModelAspectRatio(16.0 / 12.0)

    //.build();

}   // end method initTfod()

/**
 * Add telemetry about TensorFlow Object Detection (TFOD) recognitions.
 */
private void telemetryTfod() {

    List<Recognition> currentRecognitions = tfod.getRecognitions();
```

```
        telemetry.addData("# Objects Detected", currentRecognitions.size());

        // Step through the list of recognitions and display info for each one.
        for (Recognition recognition : currentRecognitions) {
            double x = (recognition.getLeft() + recognition.getRight()) / 2 ;
            double y = (recognition.getTop()  + recognition.getBottom()) / 2 ;

            telemetry.addData(""," ");
            telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(),
recognition.getConfidence() * 100);
            telemetry.addData("- Position", "%.0f / %.0f", x, y);
            telemetry.addData("- Size", "%.0f x %.0f", recognition.getWidth(),
recognition.getHeight());
        }   // end for() loop

    }   // end method telemetryTfod()

}   // end class
```

TfodNon.java

```
package Cam_auto_2023;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import org.firstinspires.ftc.robotcore.external.tfod.TFObjectDetector;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;

//import org.firstinspires.ftc.robotcore.external.tfod.TFObjectDetector;


import java.util.List;

//@Disabled
```

```
/*@Autonomous(name = "Concept: TensorFlow Object Detection Easy", group =
"LinearOpMode")
@Disabled
/*public class TfodNon extends LinearOpMode {

   private static final boolean USE_WEBCAM = true;  // true for webcam, false for phone
camera

   private TFObjectDetector tfod;

   /**
    * The variable to store our instance of the TensorFlow Object Detection processor.
    */
   //private TfodProcessor tfod;

   /*private static final String TFOD_MODEL_FILE  = "Original Modell";

   //C: The provided trackable elements in the new season PowerPlay
   private static final String[] LABELS = {
      "Red Warrior",
      "Blue Warrior"
   };

   /**
    * The variable to store our instance of the vision portal.
    */
   /*private VisionPortal visionPortal;

   @Override
   public void runOpMode() {

      initTfod();

      // Wait for the DS start button to be touched.
      telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
      telemetry.addData(">", "Touch Play to start OpMode");
      telemetry.update();
      waitForStart();

      if (opModeIsActive()) {
```

```
        while (opModeIsActive()) {

            telemetryTfod();

            // Push telemetry to the Driver Station.
            telemetry.update();

            if (tfod != null) {
            List<Recognition> updatedRecognitions = tfod.getUpdatedRecognitions();
               if (updatedRecognitions != null) {
                  //telemetry.addData("# Objects Detected", updatedRecognitions.size());

                  for (Recognition recognition : updatedRecognitions) {

                     //C: This is to move the robot into the 1st slot
                     if (recognition.getLabel().equals("Blue Warrior")) {
                        telemetry.addData("OMG!! Object is finally detected", "the blue warriorss");
                        telemetry.update();
                     }
                  }
               }

            // Save CPU resources; can resume streaming when needed.
            if (gamepad1.dpad_down) {
               visionPortal.stopStreaming();
            } else if (gamepad1.dpad_up) {
               visionPortal.resumeStreaming();
            }

            // Share the CPU.
            sleep(20);
         }
      }

      // Save more CPU resources when camera is no longer needed.
      visionPortal.close();

   }   // end runOpMode()
```

```
    /**
     * Initialize the TensorFlow Object Detection processor.
     */
    /*private void initTfod() {

        // Create the TensorFlow processor the easy way.
        tfod = TfodProcessor.easyCreateWithDefaults();

        // Create the vision portal the easy way.
        if (USE_WEBCAM) {
            visionPortal = VisionPortal.easyCreateWithDefaults(
                hardwareMap.get(WebcamName.class, "Webcam 1"), tfod);
        } else {
            visionPortal = VisionPortal.easyCreateWithDefaults(
                BuiltinCameraDirection.BACK, tfod);
        }
        //tfod.loadModelFromFile(TFOD_MODEL_FILE, LABELS);

    }   // end method initTfod()

    /**
     * Add telemetry about TensorFlow Object Detection (TFOD) recognitions.
     */
    /*private void telemetryTfod() {

        List<Recognition> currentRecognitions = tfod.getRecognitions();
        telemetry.addData("# Objects Detected", currentRecognitions.size());

        // Step through the list of recognitions and display info for each one.
        for (Recognition recognition : currentRecognitions) {
            double x = (recognition.getLeft() + recognition.getRight()) / 2 ;
            double y = (recognition.getTop()  + recognition.getBottom()) / 2 ;

            telemetry.addData(""," ");
            telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(),
recognition.getConfidence() * 100);
            telemetry.addData("- Position", "%.0f / %.0f", x, y);
            telemetry.addData("- Size", "%.0f x %.0f", recognition.getWidth(),
recognition.getHeight());
        }   // end for() loop
```

```
    }   // end method telemetryTfod()

}*/   // end class
```

The_Ultimate_Autonomous.java

```
/*package PowerPlay2022;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.ElapsedTime;

import java.util.List;
import org.firstinspires.ftc.robotcore.external.ClassFactory;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;
import org.firstinspires.ftc.robotcore.external.tfod.TFObjectDetector;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;

//@Disabled
@Autonomous(name= "The Ultimate Plan", group="Robot")
public class The_Ultimate_Autonomous extends LinearOpMode {

       /* Declare OpMode members. */
    /*private DcMotor         frontLeftDrive   = null;
    private DcMotor         frontRightDrive  = null;
    private DcMotor         backLeftDrive   = null;
    private DcMotor         backRightDrive  = null;

    private DcMotorEx armMotor;

    //Specify the source for the Tensor Flow Model
    //private static final String TFOD_MODEL_ASSET
    private static final String TFOD_MODEL_FILE  = "model_20221119_152414.tflite";
```

```
    //C: The provided trackable elements in the new season PowerPlay
    private static final String[] LABELS = {
        "Alpha",
        "Beta",
        "Charlie"
    };

    //My own added Vuforia key for season and years 2022-2023
    private static final String VUFORIA_KEY =

"AdYA5yv/////AAABmcMqplZzQEtonj1I0BpWQfZ5OSvJVqXae/05ULvqsqBbUrs6N36Lb6ws
m9l9By5irHspecbUSKok+XZ0Bf3hqsUtb0ky0ROPv6zrzwQFBuYwDCOLEgUPAa5fd1rqfivHI
zsJax8H8FeD77FHei8p1rjstdHJTt8flekQ/TiI1MHu8HpyAlgabjOOYJsRlAt5uoj/KAUKOOeSF
DtSpAMJNTwQHIVnuPRfyite5XTnsxUf8Odr7o6GilSG3XzghQRO4NtvNeqgPgImM2ldctN1A
q8fBcGGMc3tGxTRamsaysjmyfYyoTv4zzm7i3n8gdCvZDXuTOEMySV5qU7te52NAsFamfg
U9+TMyywhCNxys9N4";

    private VuforiaLocalizer vuforia;
    private TFObjectDetector tfod;

    private ElapsedTime     runtime = new ElapsedTime();

    static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
    static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;//1.0 ;     // No External Gearing.
    static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;//3.2 ;     // For figuring
circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                              (WHEEL_DIAMETER_INCHES * 3.1415);
    static final double     DRIVE_SPEED             = 0.4;
    static final double     FAST_DRIVE_SPEED            = 0.6;
    static final double     TURN_SPEED              = 0.3;
    static final double     SLOW_DRIVE_SPEED            = 0.2;
    static final double     SLOW_TURN_SPEED             = 0.2;

    @Override
    public void runOpMode() {

        initVuforia();
```

```
initTfod();

if (tfod != null) {
   tfod.activate();
   tfod.setZoom(1.1, 16.0/9.0);
}

// Initialize the drive system variables.
frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");

frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

// Send telemetry message to indicate successful Encoder reset
telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d",
                  frontLeftDrive.getCurrentPosition(),
                  frontRightDrive.getCurrentPosition(),
                  backLeftDrive.getCurrentPosition(),
                  backRightDrive.getCurrentPosition());
telemetry.update();
```

```
        waitForStart();

        if (opModeIsActive()) {
            //Bryon the next time you change my code without telling... will be the end of you
            telemetry.addData("Toll the great bell once!", "Pull the Lever forward to engage the
Piston and Pump");
            telemetry.update();
            //C: run blocks

             encoderDrive(SLOW_DRIVE_SPEED, 2, 2, 2, 2, 1.0);
             while (opModeIsActive()) {
               if (tfod != null) {
               List<Recognition> updatedRecognitions = tfod.getUpdatedRecognitions();
                  if (updatedRecognitions != null) {
                     //telemetry.addData("# Objects Detected", updatedRecognitions.size());

                     for (Recognition recognition : updatedRecognitions) {

                        //C: This is to move the robot into the 1st slot
                        if (recognition.getLabel().equals("Alpha")) {
                           telemetry.addData("OMG!! Object is finally detected", "Alpha");
                           telemetry.update();

                           //encoderDrive(SLOW_DRIVE_SPEED, 54, 54, 54, 54, 1.0);
                           encoderDrive(SLOW_DRIVE_SPEED, 15, 15, 15, 15, 1.0);

                           //C: Backward drive
                           //encoderDrive(DRIVE_SPEED, -10, -10, -10, -10, 1.0);
                           //C: Strafe to the left
                           encoderDrive(SLOW_TURN_SPEED, -20, 20, 20, -20, 1.0);

                           //encoderDrive(SLOW_TURN_SPEED, 2, -2, 2, -2, 1.0);

                           break;
                           //C: Move forward in to slot 1
                           //encoderDrive(DRIVE_SPEED, 90, 90, 90, 90, 1.0);
                           //encoderDrive(DRIVE_SPEED, 0, 6, 0, 6, 1.0);
                            //encoderDrive(DRIVE_SPEED, 5, 15, 5, 15, 1.0);
                        }
```

```
                    //C: This is to move the robot into the 2nd slot
                    if (recognition.getLabel().equals("Beta")) {
                        telemetry.addData("Oh? Object is finally detected", "Beta");
                        telemetry.update();
                        //C: Moves forward into slot 1
                        //encoderDrive(DRIVE_SPEED, 86, 86, 86, 86, 1.0);
                        //encoderDrive(SLOW_DRIVE_SPEED, 54, 54, 54, 54, 1.0);
                        encoderDrive(SLOW_DRIVE_SPEED, 14, 14, 14, 14, 1.0);
                        //encoderDrive(SLOW_DRIVE_SPEED, 1, -1, -1, 1, 1.0);
                       // break;
                    }

                    //C: This is to move the robot into the 3rd slot
                    if (recognition.getLabel().equals("Charlie")) {
                        telemetry.addData("YASSS!! Object is finally detected", "Charlie");
                        telemetry.update();

                        //encoderDrive(DRIVE_SPEED, -20, -20, -20, -20, 1.0);

                        //encoderDrive(SLOW_DRIVE_SPEED, 54, 54, 54, 54, 1.0);
                        encoderDrive(SLOW_DRIVE_SPEED, 15, 15, 15, 15, 1.0);
                        //encoderDrive(SLOW_DRIVE_SPEED, 1, 1, 1, 1, 1.0);
                        encoderDrive(SLOW_TURN_SPEED, 21, -21, -21, 21, 4.0);
                        //encoderDrive(SLOW_TURN_SPEED, 1, -1, -1, 1, 3.0);
                        //                          //encoderDrive(SLOW_TURN_SPEED, -2, 2, -2, 2,
1.0);

                       // break;

                        //C: strafes to the right
                        //encoderDrive(SLOW_TURN_SPEED,   145, -145, -145, 145, 1.0);
                        //C: Moves forward into slot 3
                        //encoderDrive(DRIVE_SPEED, 90, 90, 90, 90, 1.0);
                        //encoderDrive(DRIVE_SPEED, 15, 5, 15, 5, 1.0);
                        //encoderDrive(DRIVE_SPEED, 6, 0, 6, 0, 1.0);

                    }
                }
            }
        }
```

```
            }
        telemetry.addData("Path", "Complete");
        telemetry.update();
        sleep(4000);  // pause to display final telemetry message.
        telemetry.addData("Robot has slayed","per usually");
        telemetry.update();
        }
    }

    private void initVuforia() {
        VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.Parameters();

        parameters.vuforiaLicenseKey = VUFORIA_KEY;
        parameters.cameraName = hardwareMap.get(WebcamName.class, "Webcam 1");

        //  Instantiate the Vuforia engine
        vuforia = ClassFactory.getInstance().createVuforia(parameters);
    }

    private void initTfod() {
        int tfodMonitorViewId = hardwareMap.appContext.getResources().getIdentifier(
            "tfodMonitorViewId", "id", hardwareMap.appContext.getPackageName());
        TFObjectDetector.Parameters tfodParameters = new
TFObjectDetector.Parameters(tfodMonitorViewId);
        tfodParameters.minResultConfidence = 0.8f;
        tfodParameters.isModelTensorFlow2 = true;
        tfodParameters.inputSize = 300;
        tfod = ClassFactory.getInstance().createTFObjectDetector(tfodParameters, vuforia);

        tfod.loadModelFromFile(TFOD_MODEL_FILE, LABELS);
    }

    public void encoderDrive(double speed,
                             double frontLeftInches, double frontRightInches,
                             double backLeftInches, double backRightInches,
                             double timeoutS) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;
```

```
        // Ensure that the opmode is still active
        if (opModeIsActive()) {

            //C: Ensures that whle the drive mode is active the arm does not fall down
            //R: YOU IDIOT I CANNOT BELIEVE YOU MADE THE POWER EIGHT!??!?!?!?!
GRAHHHHHHHHHHHHHHHHH
            armMotor.setPower(0.5);

            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
            frontLeftDrive.setTargetPosition(newFrontLeftTarget);
            frontRightDrive.setTargetPosition(newFrontRightTarget);
            backLeftDrive.setTargetPosition(newBackLeftTarget);
            backRightDrive.setTargetPosition(newBackRightTarget);

            // Turn On RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
            backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

            // reset the timeout time and start motion.
            runtime.reset();
            frontLeftDrive.setPower(Math.abs(speed));
            frontRightDrive.setPower(Math.abs(speed));
            backLeftDrive.setPower(Math.abs(speed));
            backRightDrive.setPower(Math.abs(speed));

            while (opModeIsActive() &&
                   (runtime.seconds() < timeoutS) &&
                   (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {
```

```
            // Display it for the driver.
            telemetry.addData("Running to",  " %7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d",
                              frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
                              backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
        //armMotor.setPower(0);

        // Turn off RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        sleep(250);   // optional pause after each move.
    }
  }
}*/
```

Right_Auto_Finale.java

```
/*package PowerPlay2022;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.Servo;
```

```
import com.qualcomm.robotcore.util.ElapsedTime;

import java.util.List;
import org.firstinspires.ftc.robotcore.external.ClassFactory;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;
import org.firstinspires.ftc.robotcore.external.tfod.TFObjectDetector;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;

@Disabled
@Autonomous(name= "Right Finale Auto", group="Robot")
public class Right_Auto_Finale extends LinearOpMode {

        /* Declare OpMode members. */
   /*private DcMotor        frontLeftDrive   = null;
   private DcMotor        frontRightDrive  = null;
   private DcMotor        backLeftDrive   = null;
   private DcMotor        backRightDrive  = null;

   private DcMotorEx armMotor;
   private CRServo grabberLeft;
   private CRServo grabberRight;


   //Specify the source for the Tensor Flow Model
   //private static final String TFOD_MODEL_ASSET
   private static final String TFOD_MODEL_FILE  = "model_20221119_152414.tflite";

   //C: The provided trackable elements in the new season PowerPlay
   private static final String[] LABELS = {
      "Alpha",
      "Beta",
      "Charlie"
   };

   //My own added Vuforia key for season and years 2022-2023
   private static final String VUFORIA_KEY =

"AdYA5yv/////AAABmcMqplZzQEtonj1I0BpWQfZ5OSvJVqXae/05ULvqsqBbUrs6N36Lb6ws
m9l9By5irHspecbUSKok+XZ0Bf3hqsUtb0ky0ROPv6zrzwQFBuYwDCOLEgUPAa5fd1rqfivHI
```

```
zsJax8H8FeD77FHei8p1rjstdHJTt8flekQ/TiI1MHu8HpyAlgabjOOYJsRlAt5uoj/KAUKOOeSF
DtSpAMJNTwQHIVnuPRfyite5XTnsxUf8Odr7o6GilSG3XzghQRO4NtvNeqgPgImM2ldctN1A
q8fBcGGMc3tGxTRamsaysjmyfYyoTv4zzm7i3n8gdCvZDXuTOEMySV5qU7te52NAsFamfg
U9+TMyywhCNxys9N4";

   private VuforiaLocalizer vuforia;
   private TFObjectDetector tfod;

   private ElapsedTime     runtime = new ElapsedTime();

   static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
   static final double     DRIVE_GEAR_REDUCTION    = 1.0 ;     // No External Gearing.
   static final double     WHEEL_DIAMETER_INCHES   = 4.0 ;     // For figuring circumference
   static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                             (WHEEL_DIAMETER_INCHES * 3.1415);
   static final double     DRIVE_SPEED             = 0.4;
   static final double     TURN_SPEED              = 0.3;
   static final double     SLOW_DRIVE_SPEED        = 0.2;
   static final double     SLOW_TURN_SPEED         = 0.2;
   static final double     VERY_SLOW_DRIVE_SPEED   = 0.1;
   static final double     VERY_SLOW_TURN_SPEED    = 0.1;


   @Override
   public void runOpMode() {

      initVuforia();
      initTfod();

      if (tfod != null) {
         tfod.activate();
         tfod.setZoom(1.1, 16.0/9.0);
      }

      // Initialize the drive system variables.
      frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
      frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
      backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
```

```
        backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
        armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
        grabberLeft = hardwareMap.get(CRServo.class, "grabberLeft");
        grabberRight = hardwareMap.get(CRServo.class, "grabberRight");

        frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
        frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
        backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
        backRightDrive.setDirection(DcMotor.Direction.FORWARD);

        frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

        armMotor.setDirection(DcMotor.Direction.FORWARD);
        armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        grabberLeft.setDirection(DcMotor.Direction.FORWARD);
        grabberRight.setDirection(DcMotor.Direction.FORWARD);

        // Send telemetry message to indicate successful Encoder reset
        telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d",
                          frontLeftDrive.getCurrentPosition(),
                          frontRightDrive.getCurrentPosition(),
                          backLeftDrive.getCurrentPosition(),
                          backRightDrive.getCurrentPosition());
        telemetry.update();

        waitForStart();

        if (opModeIsActive()) {
            //Bryon the next time you change my code without telling... will be the end of you
            telemetry.addData("Toll the great bell once!", "Pull the Lever forward to engage the
Piston and Pump");
            telemetry.update();
```

```
        //C: run blocks
coneHolder();
        sleep(800);
        //C: Drive forward for a short distance
         //encoderDrive(DRIVE_SPEED, 24, 24, 24, 24, 1.0);
         //add here: Turns to the left slightly and lowers cone
         while (opModeIsActive()) {
           if (tfod != null) {
           List<Recognition> updatedRecognitions = tfod.getUpdatedRecognitions();
              if (updatedRecognitions != null) {
                 //telemetry.addData("# Objects Detected", updatedRecognitions.size());

                 for (Recognition recognition : updatedRecognitions) {

                    //C: This is to move the robot into the 1st slot
                    if (recognition.getLabel().equals("Alpha")) {
                       telemetry.addData("OMG!! Object is finally detected", "Alpha");
                       telemetry.update();

           //C: this first part applies to all other directions as well, the cone scorer part
           encoderDrive(SLOW_DRIVE_SPEED, 824, 824, 824, 824, 1.0);
                  encoderDrive(SLOW_TURN_SPEED, -90, 90, -90, 90, 1.25);
           lettingGo();
                    encoderDrive(DRIVE_SPEED, 10, 10, 10, 10, 1.0);
           //turning to the right
                 encoderDrive(TURN_SPEED, 45, -45, 45, -45, 1.0);

           //C: This part is specifically for this direction
                 //strafe to the left
                 encoderDrive(TURN_SPEED, -140, 140, 130, -130, 1.0);
                 armMotor.setPower(0.5);

                  }

                    //C: This is to move the robot into the 2nd slot
                    if (recognition.getLabel().equals("Beta")) {
                       telemetry.addData("Oh? Object is finally detected", "Beta");
                       telemetry.update();

          encoderDrive(SLOW_DRIVE_SPEED, 824, 824, 824, 824, 1.0);
```

```
                encoderDrive(SLOW_TURN_SPEED, -90, 90, -90, 90, 1.25);
            lettingGo();
                  encoderDrive(DRIVE_SPEED, 10, 10, 10, 10, 1.0);
            //turning to the right
                 encoderDrive(TURN_SPEED, 45, -45, 45, -45, 1.0);

            encoderDrive(DRIVE_SPEED, 20, 20, 20, 20, 1.0);
                     }

                     //C: This is to move the robot into the 3rd slot
                     if (recognition.getLabel().equals("Charlie")) {
                        telemetry.addData("YASSS!! Object is finally detected", "Charlie");
                        telemetry.update();

                        encoderDrive(SLOW_DRIVE_SPEED, 824, 824, 824, 824, 1.0);
                   encoderDrive(SLOW_TURN_SPEED, -90, 90, -90, 90, 1.25);
            lettingGo();
                  encoderDrive(DRIVE_SPEED, 10, 10, 10, 10, 1.0);
            //turning to the left
                 encoderDrive(TURN_SPEED, 45, -45, 45, -45, 1.0);

            //C: Strafe to the right
            encoderDrive(TURN_SPEED, 140, -140, -130, 130, 1.0);

                     }
                     /*double col = (recognition.getLeft() + recognition.getRight()) / 2 ;
                     double row = (recognition.getTop()  + recognition.getBottom()) / 2 ;
                     double width  = Math.abs(recognition.getRight() - recognition.getLeft()) ;
                     double height = Math.abs(recognition.getTop()  - recognition.getBottom()) ;

                     telemetry.addData("","  ");
                     telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(),
recognition.getConfidence() * 100 );
                     telemetry.addData("- Position (Row/Col)","%.0f / %.0f", row, col);
                     telemetry.addData("- Size (Width/Height)","%.0f / %.0f", width, height);*/
   /*                }
                }
            }
          }
       telemetry.addData("Path", "Complete");
```

```
    telemetry.update();
    sleep(4000);  // pause to display final telemetry message.
    telemetry.addData("Robot has slayed","per usually");
    telemetry.update();
    }
}

public void coneHolder() {
    grabberLeft.setPower(-2);
    grabberRight.setPower(2);
    armMotor.setPower(0.5);
}

public void lettingGo() {
    armMotor.setPower(0);
    grabberLeft.setPower(-2);
    grabberRight.setPower(2);
    sleep(3000);

    armMotor.setPower(0.5);
    grabberLeft.setPower(1);
    grabberRight.setPower(-1);
    sleep(3000);
}

public void backItUp() {
    encoderDrive(SLOW_DRIVE_SPEED, -19.8, -19.8, -19.8, -19.8, 1.5);
    encoderDrive(SLOW_DRIVE_SPEED, -3.5, -3.5, -3.5, -3.5, 1.5);
}

private void initVuforia() {
    VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.Parameters();

    parameters.vuforiaLicenseKey = VUFORIA_KEY;
    parameters.cameraName = hardwareMap.get(WebcamName.class, "Webcam 1");

    //  Instantiate the Vuforia engine
    vuforia = ClassFactory.getInstance().createVuforia(parameters);
}
```

```
    private void initTfod() {
        int tfodMonitorViewId = hardwareMap.appContext.getResources().getIdentifier(
            "tfodMonitorViewId", "id", hardwareMap.appContext.getPackageName());
        TFObjectDetector.Parameters tfodParameters = new
TFObjectDetector.Parameters(tfodMonitorViewId);
        tfodParameters.minResultConfidence = 0.8f;
        tfodParameters.isModelTensorFlow2 = true;
        tfodParameters.inputSize = 300;
        tfod = ClassFactory.getInstance().createTFObjectDetector(tfodParameters, vuforia);

        tfod.loadModelFromFile(TFOD_MODEL_FILE, LABELS);
    }

    public void encoderDrive(double speed,
                             double frontLeftInches, double frontRightInches,
                             double backLeftInches, double backRightInches,
                             double timeoutS) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {

            //C: Ensures that whle the drive mode is active the arm does not fall down
            //R: YOU IDIOT I CANNOT BELIEVE YOU MADE THE POWER EIGHT!??!?!?!?!
GRAHHHHHHHHHHHHHHHH
            armMotor.setPower(0.5);

            // Determine new target position, and pass to motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
            newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
            frontLeftDrive.setTargetPosition(newFrontLeftTarget);
```

```
        frontRightDrive.setTargetPosition(newFrontRightTarget);
        backLeftDrive.setTargetPosition(newBackLeftTarget);
        backRightDrive.setTargetPosition(newBackRightTarget);

        // Turn On RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
            (runtime.seconds() < timeoutS) &&
            (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  " %7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d",
                              frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
                              backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
        backRightDrive.setPower(0);
        //armMotor.setPower(0);
```

```
            // Turn off RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

            //sleep(250);   // optional pause after each move.
        }
    }
}*/
```

Left_Auto_FInale.java

```
/*package PowerPlay2022;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.ElapsedTime;

import java.util.List;
import org.firstinspires.ftc.robotcore.external.ClassFactory;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;
import org.firstinspires.ftc.robotcore.external.tfod.TFObjectDetector;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;

import com.qualcomm.hardware.bosch.BNO055IMU;
import com.qualcomm.robotcore.util.Range;
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.navigation.AxesOrder;
import org.firstinspires.ftc.robotcore.external.navigation.AxesReference;
import org.firstinspires.ftc.robotcore.external.navigation.Orientation;

//@Disabled
@Autonomous(name= "Left Finale Auto", group="Robot")
```

```
public class Left_Auto_FInale extends LinearOpMode {

        /* Declare OpMode members. */
    /*private DcMotor         frontLeftDrive   = null;
    private DcMotor         frontRightDrive  = null;
    private DcMotor         backLeftDrive   = null;
    private DcMotor         backRightDrive  = null;

    private DcMotorEx armMotor;
    private CRServo grabberLeft;
    private CRServo grabberRight;

    private BNO055IMU       imu         = null;      // Control/Expansion Hub IMU
    private double          robotHeading  = 0;
    private double          headingOffset = 0;
    private double          headingError  = 0;
    private double  targetHeading = 0;
    private double  driveSpeed    = 0;
    private double  turnSpeed     = 0;
    private double  frontLeftSpeed     = 0;
    private double  frontRightSpeed    = 0;
    private double  backLeftSpeed     = 0;
    private double  backRightSpeed    = 0;
    private int     frontLeftTarget    = 0;
    private int     frontRightTarget   = 0;
    private int     backLeftTarget    = 0;
    private int     backRightTarget   = 0;


    //Specify the source for the Tensor Flow Model
    //private static final String TFOD_MODEL_ASSET
    private static final String TFOD_MODEL_FILE  = "model_20221119_152414.tflite";

    //C: The provided trackable elements in the new season PowerPlay
    private static final String[] LABELS = {
       "Alpha",
       "Beta",
       "Charlie"
    };
```

```
    //My own added Vuforia key for season and years 2022-2023
    private static final String VUFORIA_KEY =

"AdYA5yv/////AAABmcMqplZzQEtonj1I0BpWQfZ5OSvJVqXae/05ULvqsqBbUrs6N36Lb6ws
m9l9By5irHspecbUSKok+XZ0Bf3hqsUtb0ky0ROPv6zrzwQFBuYwDCOLEgUPAa5fd1rqfivHI
zsJax8H8FeD77FHei8p1rjstdHJTt8flekQ/TiI1MHu8HpyAlgabjOOYJsRlAt5uoj/KAUKOOeSF
DtSpAMJNTwQHIVnuPRfyite5XTnsxUf8Odr7o6GilSG3XzghQRO4NtvNeqgPgImM2ldctN1A
q8fBcGGMc3tGxTRamsaysjmyfYyoTv4zzm7i3n8gdCvZDXuTOEMySV5qU7te52NAsFamfg
U9+TMyywhCNxys9N4";

    private VuforiaLocalizer vuforia;
    private TFObjectDetector tfod;

    private ElapsedTime     runtime = new ElapsedTime();

    static final double     COUNTS_PER_MOTOR_REV    = 28; //1440 ;    // eg: TETRIX Motor
Encoder
    static final double     DRIVE_GEAR_REDUCTION    = 5.0 ;     // No External Gearing.
    static final double     WHEEL_DIAMETER_INCHES   = 2.953 ;     // For figuring
circumference
    static final double     COUNTS_PER_INCH         = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
                              (WHEEL_DIAMETER_INCHES * 3.1415);
    static final double     HEADING_THRESHOLD       = 1.0 ;    // How close must the heading
get to the target before moving to next step.

    static final double     DRIVE_SPEED             = 0.4;
    static final double     TURN_SPEED              = 0.3;
    static final double     SLOW_DRIVE_SPEED        = 0.2;
    static final double     SLOW_TURN_SPEED         = 0.2;
    static final double     VERY_SLOW_DRIVE_SPEED       = 0.01;
    static final double     VERY_SLOW_TURN_SPEED        = 0.1;

    static final double     P_TURN_GAIN             = 0.02;     // Larger is more responsive, but also
less stable
    static final double     P_DRIVE_GAIN            = 0.03;     // Larger is more responsive, but also
less stable

    @Override
    public void runOpMode() {
```

```
initVuforia();
initTfod();

if (tfod != null) {
   tfod.activate();
   tfod.setZoom(1.1, 16.0/9.0);
}

// Initialize the drive system variables.
frontLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
frontRightDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
backLeftDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
backRightDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
armMotor = hardwareMap.get(DcMotorEx.class, "armMotor");
grabberLeft = hardwareMap.get(CRServo.class, "grabberLeft");
grabberRight = hardwareMap.get(CRServo.class, "grabberRight");

frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
backRightDrive.setDirection(DcMotor.Direction.FORWARD);

frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

frontLeftDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
frontRightDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
backLeftDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
backRightDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);

frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

armMotor.setDirection(DcMotor.Direction.FORWARD);
armMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
```

```
        grabberLeft.setDirection(DcMotor.Direction.FORWARD);
        grabberRight.setDirection(DcMotor.Direction.FORWARD);

            // define initialization values for IMU, and then initialize it.
        BNO055IMU.Parameters parameters = new BNO055IMU.Parameters();
        parameters.angleUnit           = BNO055IMU.AngleUnit.DEGREES;
        imu = hardwareMap.get(BNO055IMU.class, "imu");
        imu.initialize(parameters);

        while (opModeInInit()) {
           telemetry.addData(">", "Robot Heading = %4.0f", getRawHeading());
           telemetry.update();
        }

        // Send telemetry message to indicate successful Encoder reset
        telemetry.addData("Yasss!! Oki, Starting at",  "%7d :%7d",
                          frontLeftDrive.getCurrentPosition(),
                          frontRightDrive.getCurrentPosition(),
                          backLeftDrive.getCurrentPosition(),
                          backRightDrive.getCurrentPosition());
        telemetry.update();

        waitForStart();

        if (opModeIsActive()) {
           //Bryon the next time you change my code without telling... will be the end of you
           encoderDrive(SLOW_DRIVE_SPEED, 100, 100, 100, 100, 5.0);
           telemetry.addData("Toll the great bell once!", "Pull the Lever forward to engage the
Piston and Pump");
           telemetry.update();
           //C: run blocks
           //C: grabs on to the cone
           coneHolder();
           sleep(800);

            while (opModeIsActive()) {
              if (tfod != null) {
              List<Recognition> updatedRecognitions = tfod.getUpdatedRecognitions();
                 if (updatedRecognitions != null) {
                    //telemetry.addData("# Objects Detected", updatedRecognitions.size());
```

```
            //C: Enables Bambi to recognize to search through the list of recognitions
            for (Recognition recognition : updatedRecognitions) {

                //C: Movement for the 1st slot
                if (recognition.getLabel().equals("Alpha")) {
                    telemetry.addData("OMG!! Object is finally detected", "Alpha");
                    telemetry.update();

                    //C: The function for scoring the cone
                    scorerCombo();

                    //C: This part is specifically for this direction
                    //C: strafe to the left for ten inches for 2 second interval
                    encoderDrive(VERY_SLOW_TURN_SPEED, -10, 10, 10, -10, 2.0);
                 }

                //C: Movement for 2nd slot
                if (recognition.getLabel().equals("Beta")) {
                    telemetry.addData("Oh? Object is finally detected", "Beta");
                    telemetry.update();

                    //C: The function for scoring the cone
                    scorerCombo();
                }

                //C: This is to move the robot into the 3rd slot
                if (recognition.getLabel().equals("Charlie")) {
                    telemetry.addData("YASSS!! Object is finally detected", "Charlie");
                    telemetry.update();

                    //C: The function for scoring the cone
                    scorerCombo();

                    //C: strafe to the right for ten inches for 2 second interval
                    encoderDrive(SLOW_TURN_SPEED, 10, -10, -10, 10, 2.0);
                }
            }
        }
    }
```

```
        }
    telemetry.addData("Path", "Complete");
    telemetry.update();
    sleep(3000);  // pause to display final telemetry message.
    telemetry.addData("Robot has slayed","per usually");
    telemetry.update();
    }
}

public void scorerCombo() {
     //C: this first part applies to all other directions as well, the cone scorer part
    encoderDrive(DRIVE_SPEED, 16, 16, 16, 16, 2.0);

    //C: Turns to the left
    turnToHeading( TURN_SPEED, -45.0 );   // turn -45 Deg heading

    backItUp();

    lettingGo();

    encoderDrive(VERY_SLOW_DRIVE_SPEED, 1, 1, 1, 1, 1.0);
    //turning to the right
    turnToHeading( TURN_SPEED, 45.0);   // Hold 45 Deg heading for a 1/2 second
    //encoderDrive(VERY_SLOW_TURN_SPEED, 1, -1, 1, -1, 1.0);
}

public void coneHolder() {
    grabberLeft.setPower(-1);
    grabberRight.setPower(1);
    armMotor.setPower(0.5);
}

public void lettingGo() {
    armMotor.setPower(-0.3);
    grabberLeft.setPower(-1);
    grabberRight.setPower(1);
    sleep(3000);

    armMotor.setPower(0.5);
    grabberLeft.setPower(1);
```

```
        grabberRight.setPower(-1);
        sleep(3000);
    }

    public void backItUp() {
        encoderDrive(SLOW_DRIVE_SPEED, -1, -1, -1, -1, 1.0);
    }

    private void initVuforia() {
        VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.Parameters();

        parameters.vuforiaLicenseKey = VUFORIA_KEY;
        parameters.cameraName = hardwareMap.get(WebcamName.class, "Webcam 1");

        //  Instantiate the Vuforia engine
        vuforia = ClassFactory.getInstance().createVuforia(parameters);
    }

    private void initTfod() {
        int tfodMonitorViewId = hardwareMap.appContext.getResources().getIdentifier(
            "tfodMonitorViewId", "id", hardwareMap.appContext.getPackageName());
        TFObjectDetector.Parameters tfodParameters = new
TFObjectDetector.Parameters(tfodMonitorViewId);
        tfodParameters.minResultConfidence = 0.8f;
        tfodParameters.isModelTensorFlow2 = true;
        tfodParameters.inputSize = 300;
        tfod = ClassFactory.getInstance().createTFObjectDetector(tfodParameters, vuforia);

        tfod.loadModelFromFile(TFOD_MODEL_FILE, LABELS);
    }

        public void driveStraight(double maxDriveSpeed,
                                  double distance,
                                  double heading) {

        // Ensure that the opmode is still active
        if (opModeIsActive()) {

            // Determine new target position, and pass to motor controller
            int moveCounts = (int)(distance * COUNTS_PER_INCH);
```

```
        int frontLeftTarget = frontLeftDrive.getCurrentPosition() + moveCounts;
        int frontRightTarget = frontRightDrive.getCurrentPosition() + moveCounts;
        int backLeftTarget = backLeftDrive.getCurrentPosition() + moveCounts;
        int backRightTarget = backRightDrive.getCurrentPosition() + moveCounts;

        // Set Target FIRST, then turn on RUN_TO_POSITION
        frontLeftDrive.setTargetPosition(frontLeftTarget);
        frontRightDrive.setTargetPosition(frontRightTarget);
        backLeftDrive.setTargetPosition(frontLeftTarget);
        backRightDrive.setTargetPosition(backRightTarget);

        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // Set the required driving speed  (must be positive for RUN_TO_POSITION)
        // Start driving straight, and then enter the control loop
        maxDriveSpeed = Math.abs(maxDriveSpeed);
        moveRobot(maxDriveSpeed, 0);

        // keep looping while we are still active, and BOTH motors are running.
        while (opModeIsActive() &&
            frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy()) {

            // Determine required steering to keep on heading
            turnSpeed = getSteeringCorrection(heading, P_DRIVE_GAIN);

            // if driving in reverse, the motor correction also needs to be reversed
            if (distance < 0)
                turnSpeed *= -1.0;

            // Apply the turning correction to the current driving speed.
            moveRobot(driveSpeed, turnSpeed);

            // Display drive status for the driver.
            sendTelemetry(true);
        }
```

```
        // Stop all motion & Turn off RUN_TO_POSITION
        moveRobot(0, 0);
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    }
}

   public void turnToHeading(double maxTurnSpeed, double heading) {

    // Run getSteeringCorrection() once to pre-calculate the current error
    getSteeringCorrection(heading, P_DRIVE_GAIN);

    // keep looping while we are still active, and not on heading.
    while (opModeIsActive() && (Math.abs(headingError) > HEADING_THRESHOLD)) {

        // Determine required steering to keep on heading
        turnSpeed = getSteeringCorrection(heading, P_TURN_GAIN);

        // Clip the speed to the maximum permitted value.
        turnSpeed = Range.clip(turnSpeed, -maxTurnSpeed, maxTurnSpeed);

        // Pivot in place by applying the turning correction
        moveRobot(0, turnSpeed);

        // Display drive status for the driver.
        sendTelemetry(false);
    }

    // Stop all motion;
    moveRobot(0, 0);
}

    public void holdHeading(double maxTurnSpeed, double heading, double holdTime) {

    ElapsedTime holdTimer = new ElapsedTime();
    holdTimer.reset();

    // keep looping while we have time remaining.
```

```
        while (opModeIsActive() && (holdTimer.time() < holdTime)) {
            // Determine required steering to keep on heading
            turnSpeed = getSteeringCorrection(heading, P_TURN_GAIN);

            // Clip the speed to the maximum permitted value.
            turnSpeed = Range.clip(turnSpeed, -maxTurnSpeed, maxTurnSpeed);

            // Pivot in place by applying the turning correction
            moveRobot(0, turnSpeed);

            // Display drive status for the driver.
            sendTelemetry(false);
        }

        // Stop all motion;
        moveRobot(0, 0);
    }

    public double getSteeringCorrection(double desiredHeading, double proportionalGain) {
        targetHeading = desiredHeading;  // Save for telemetry

        // Get the robot heading by applying an offset to the IMU heading
        robotHeading = getRawHeading() - headingOffset;

        // Determine the heading current error
        headingError = targetHeading - robotHeading;

        // Normalize the error to be within +/- 180 degrees
        while (headingError > 180)  headingError -= 360;
        while (headingError <= -180) headingError += 360;

        // Multiply the error by the gain to determine the required steering correction/  Limit the
result to +/- 1.0
        return Range.clip(headingError * proportionalGain, -1, 1);
    }

        public void moveRobot(double drive, double turn) {
        driveSpeed = drive;     // save this value as a class member so it can be used by telemetry.
        turnSpeed  = turn;      // save this value as a class member so it can be used by telemetry.
```

```
        frontLeftSpeed  = drive - turn;
        frontRightSpeed = drive + turn;
        backLeftSpeed  = drive - turn;
        backRightSpeed = drive + turn;

        // Scale speeds down if either one exceeds +/- 1.0;
        double max = Math.max(Math.abs(frontLeftSpeed), Math.max(Math.abs(frontRightSpeed),
Math.max(Math.abs(backLeftSpeed), Math.abs(backRightSpeed))));
        if (max > 1.0)
        {
           frontLeftSpeed /= max;
           frontRightSpeed /= max;
           backLeftSpeed /= max;
           backRightSpeed /= max;
        }

        frontLeftDrive.setPower(frontLeftSpeed);
        frontRightDrive.setPower(frontRightSpeed);
        backLeftDrive.setPower(backLeftSpeed);
        backRightDrive.setPower(backRightSpeed);
    }

        private void sendTelemetry(boolean straight) {

        if (straight) {
           telemetry.addData("Motion", "Drive Straight");
           telemetry.addData("Target Pos L:R",  "%7d:%7d",      frontLeftTarget,  frontRightTarget,
backLeftTarget,  backRightTarget);
           telemetry.addData("Actual Pos L:R",  "%7d:%7d",
frontLeftDrive.getCurrentPosition(),
                   frontRightDrive.getCurrentPosition(), backLeftDrive.getCurrentPosition(),
                   backRightDrive.getCurrentPosition());
        } else {
           telemetry.addData("Motion", "Turning");
        }

        telemetry.addData("Angle Target:Current", "%5.2f:%5.0f", targetHeading, robotHeading);
        telemetry.addData("Error:Steer",  "%5.1f:%5.1f", headingError, turnSpeed);
        telemetry.addData("Wheel Speeds L:R.", "%5.2f : %5.2f",  frontLeftSpeed,
frontRightSpeed, backLeftSpeed, backRightSpeed);
```

```
        telemetry.update();
    }

    //read the raw (un-offset Gyro heading) directly from the IMU
    public double getRawHeading() {
        Orientation angles   = imu.getAngularOrientation(AxesReference.INTRINSIC,
AxesOrder.ZYX, AngleUnit.DEGREES);
        return angles.firstAngle;
    }

    // Reset the "offset" heading back to zero
    public void resetHeading() {
        // Save a new heading offset equal to the current raw heading.
        headingOffset = getRawHeading();
        robotHeading = 0;
    }

    public void encoderDrive(double speed,
                             double frontLeftInches, double frontRightInches,
                             double backLeftInches, double backRightInches,
                             double timeouts) {
        int newFrontLeftTarget;
        int newFrontRightTarget;
        int newBackLeftTarget;
        int newBackRightTarget;

         // Ensure that the opmode is still active
        if (opModeIsActive()) {

            //C: Ensures that whle the drive mode is active the arm does not fall down
            //armMotor.setPower(0.5);

            //C: Determines the new target position, and passes onto the motor controller
            newFrontLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(frontLeftInches *
COUNTS_PER_INCH);
            newFrontRightTarget = frontRightDrive.getCurrentPosition() + (int)(frontRightInches *
COUNTS_PER_INCH);
            newBackLeftTarget = backLeftDrive.getCurrentPosition() + (int)(backLeftInches *
COUNTS_PER_INCH);
```

```
        newBackRightTarget = backRightDrive.getCurrentPosition() + (int)(backRightInches *
COUNTS_PER_INCH);
        frontLeftDrive.setTargetPosition(newFrontLeftTarget);
        frontRightDrive.setTargetPosition(newFrontRightTarget);
        backLeftDrive.setTargetPosition(newBackLeftTarget);
        backRightDrive.setTargetPosition(newBackRightTarget);

        //C: Turns on method RUN_TO_POSITION
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));

        while (opModeIsActive() &&
            (runtime.seconds() < timeouts) && //C: Reuben's idea
            (frontLeftDrive.isBusy() && frontRightDrive.isBusy() && backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  " %7d :%7d :%7d :%7d", newFrontLeftTarget,
newFrontRightTarget, newBackLeftTarget,  newBackRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d :%7d :%7d",
                                frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition(),
                                backLeftDrive.getCurrentPosition(),
backRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        frontRightDrive.setPower(0);
        backLeftDrive.setPower(0);
```

```
            backRightDrive.setPower(0);
            //armMotor.setPower(0);

            // Turn off RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

            sleep(250);   // optional pause after each move.
        }
    }
}*/
```

TestTFOD3.java

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

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import java.util.List;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.navigation.AxesOrder;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaCurrentGame;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.robotcore.external.tfod.Tfod;
import org.firstinspires.ftc.robotcore.external.tfod.TFObjectDetector;
@Disabled
@Autonomous(name = "TestTFOD3")
public class TestTFOD3 extends LinearOpMode {

  private DcMotorEx frontLeftMotor, frontRightMotor, backRightMotor, backLeftMotor;
```

```
  //private TFObjectDetector tfod;

  private VuforiaLocalizer vuforia;

  Recognition recognition;

  public static DcMotor frontRight;
  public static DcMotor frontLeft;
  public static DcMotor backRight;
  public static DcMotor backLeft;

  private int frontRightPos;
  private int backRightPos;
  private int frontLeftPos;
  private int backLeftPos;
  static final double FORWARD_SPEED = 0.6;
  static final double TURN_SPEED = 0.5;
  /*private static final String FRONT_RIGHT_NAME = "FrontRightMotor";
  private static final String FRONT_LEFT_NAME = "FrontLeftMotor";
  private static final String BACK_RIGHT_NAME = "BackRightMotor";
  private static final String BACK_LEFT_NAME = "BackLeftMotor";
  */

  /**
   * This function is executed when this Op Mode is selected from the Driver Station.
   */
//@Override
//public void runOpMode() {

    /*frontRight = hardwareMap.get(DcMotor.class, FRONT_RIGHT_NAME);
    frontLeft = hardwareMap.get(DcMotor.class, FRONT_LEFT_NAME);
    backRight = hardwareMap.get(DcMotor.class, BACK_RIGHT_NAME);
    backLeft = hardwareMap.get(DcMotor.class, BACK_LEFT_NAME);*/


    /*frontLeftMotor = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");
    frontRightMotor = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
    backLeftMotor = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
    backRightMotor = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
```

```
    frontLeftMotor.setDirection(DcMotor.Direction.REVERSE);
    backLeftMotor.setDirection(DcMotor.Direction.REVERSE);

    frontRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    frontLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

    //frontRight.setDirection(DcMotor.Direction.REVERSE);
    //backRight.setDirection(DcMotor.Direction.REVERSE);
    frontRightPos = 0;
    frontLeftPos = 0;
    backRightPos = 0;
    backLeftPos = 0;
/*
    List<Recognition> recognitions;
    int index;

    vuforiaPOWERPLAY = new VuforiaCurrentGame();
    tfod = new Tfod();

    // Sample TFOD Op Mode
    // Initialize Vuforia.
    vuforiaPOWERPLAY.initialize(

"AdYA5yv/////AAABmcMqplZzQEtonj1I0BpWQfZ5OSvJVqXae/05ULvqsqBbUrs6N36Lb6ws
m9l9By5irHspecbUSKok+XZ0Bf3hqsUtb0ky0ROPv6zrzwQFBuYwDCOLEgUPAa5fd1rqfivHI
zsJax8H8FeD77FHei8p1rjstdHJTt8flekQ/TiI1MHu8HpyAlgabjOOYJsRlAt5uoj/KAUKOOeSF
DtSpAMJNTwQHIVnuPRfyite5XTnsxUf8Odr7o6GilSG3XzghQRO4NtvNeqgPgImM2ldctN1A
q8fBcGGMc3tGxTRamsaysjmyfYyoTv4zzm7i3n8gdCvZDXuTOEMySV5qU7te52NAsFamfg
U9+TMyywhCNxys9N4", // vuforiaLicenseKey
        hardwareMap.get(WebcamName.class, "Webcam 1"), // cameraName
        "", // webcamCalibrationFilename
        false, // useExtendedTracking
        false, // enableCameraMonitoring
        VuforiaLocalizer.Parameters.CameraMonitorFeedback.NONE, // cameraMonitorFeedback
        0, // dx
        0, // dy
        0, // dz
        AxesOrder.XZY, // axesOrder
```

```
        90, // firstAngle
        90, // secondAngle
        0, // thirdAngle
        true); // useCompetitionFieldTargetLocations
    tfod.useDefaultModel();
    // Set min confidence threshold to 0.7
    tfod.initialize(vuforiaPOWERPLAY, (float) 0.7, true, true);
    // Initialize TFOD before waitForStart.
    // Activate TFOD here so the object detection labels are visible
    // in the Camera Stream preview window on the Driver Station.
    tfod.activate();
    // Enable following block to zoom in on target.
    tfod.setZoom(1, 16 / 9);
    telemetry.addData("Gampead preview on/off", "3 dots, Camera Stream");
    telemetry.addData("->", "Press Play to start >;3");
    telemetry.update();
    // Wait for start command from Driver Station.
    */

    //waitForStart();
  //}

  //private void drive (int frontRightTarget, int frontLeftTarget, int backRightTarget, int
backLeftTarget, double speed){

    /*frontRightPos +=  frontRightTarget;
    //frontLeftPos += frontLeftTarget;
    /*backRightPos += backRightTarget;
    backLeftPos += backLeftTarget;

    frontRight.setTargetPosition(frontRightPos);
    frontLeft.setTargetPosition(frontLeftPos);
    backRight.setTargetPosition(backRightPos);
    backLeft.setTargetPosition(backLeftPos);

    frontRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    frontLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    backRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    backLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
```

```
    frontRight.setPower(speed);
    frontLeft.setPower(speed);
    backRight.setPower(speed);
    backLeft.setPower(speed);

    while (opModeIsActive() && frontRight.isBusy() && frontLeft.isBusy() &&
backRight.isBusy() && backLeft.isBusy()) {
      idle();
    }
  }
}*/
```

NewTestAuto,java

```
/*package org.firstinspires.ftc.teamcode.Autonoumous;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.ElapsedTime;
import java.util.List;
import org.firstinspires.ftc.robotcore.external.ClassFactory;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;
import org.firstinspires.ftc.robotcore.external.tfod.TFObjectDetector;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import com.qualcomm.robotcore.hardware.DcMotorEx;


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

    /* Declare OpMode members. */
    /*private DcMotor         frontLeftDrive   = null;
    private DcMotor         frontRightDrive  = null;
```

```
    private DcMotor         backLeftDrive   = null;
    private DcMotor         backRightDrive  = null;
     private static final String TFOD_MODEL_FILE  = "model_20221119_152414.tflite";


    private ElapsedTime     runtime = new ElapsedTime();

    // Calculate the COUNTS_PER_INCH for your specific drive train.
    // Go to your motor vendor website to determine your motor's COUNTS_PER_MOTOR_REV
    // For external drive gearing, set DRIVE_GEAR_REDUCTION as needed.
    // For example, use a value of 2.0 for a 12-tooth spur gear driving a 24-tooth spur gear.
    // This is gearing DOWN for less speed and more torque.
    // For gearing UP, use a gear ratio less than 1.0. Note this will affect the direction of wheel
rotation.
    static final double COUNTS_PER_MOTOR_REV= 28 ;    // eg: TETRIX Motor Encoder
    static final double DRIVE_GEAR_REDUCTION    = 5 ;     // No External Gearing.
    static final double WHEEL_DIAMETER_INCHES   = 3.0;     // For figuring circumference
    static final double COUNTS_PER_INCH = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) / (WHEEL_DIAMETER_INCHES *3.1416);
    static final double     DRIVE_SPEED             = 0.4;
    static final double     TURN_SPEED              = 0.3;
    /*private static final String[] LABELS = {
       "Alpha",
       "Beta",
       "Charlie"
    };
    private static final String VUFORIA_KEY =

"AdYA5yv/////AAABmcMqplZzQEtonj1I0BpWQfZ5OSvJVqXae/05ULvqsqBbUrs6N36Lb6ws
m9l9By5irHspecbUSKok+XZ0Bf3hqsUtb0ky0ROPv6zrzwQFBuYwDCOLEgUPAa5fd1rqfivHI
zsJax8H8FeD77FHei8p1rjstdHJTt8flekQ/TiI1MHu8HpyAlgabjOOYJsRlAt5uoj/KAUKOOeSF
DtSpAMJNTwQHIVnuPRfyite5XTnsxUf8Odr7o6GilSG3XzghQRO4NtvNeqgPgImM2ldctN1A
q8fBcGGMc3tGxTRamsaysjmyfYyoTv4zzm7i3n8gdCvZDXuTOEMySV5qU7te52NAsFamfg
U9+TMyywhCNxys9N4";
    private VuforiaLocalizer vuforia;
    private TFObjectDetector tfod;*/



    /*@Override
```

```
public void runOpMode() {

    // Initialize the drive system variables.
    //camera is on the back side of the robot
    //in autonomous direction is reversed because of the camera
    frontLeftDrive = hardwareMap.get(DcMotorEx.class, "BackRightMotor");
    frontRightDrive = hardwareMap.get(DcMotorEx.class, "BackLeftMotor");
    backLeftDrive = hardwareMap.get(DcMotorEx.class, "FrontRightMotor");
    backRightDrive = hardwareMap.get(DcMotorEx.class, "FrontLeftMotor");


    //initVuforia();
    //initTfod();

    /*if (tfod != null) {
        tfod.activate();
        tfod.setZoom(1.1, 16.0/9.0);
    }*/

    /*frontLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    frontRightDrive.setDirection(DcMotor.Direction.FORWARD);
    backLeftDrive.setDirection(DcMotor.Direction.REVERSE);
    backRightDrive.setDirection(DcMotor.Direction.FORWARD);

    frontLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    frontRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backLeftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    backRightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

    frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    backRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    frontLeftDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    frontRightDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    backLeftDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    backRightDrive.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);

    // Send telemetry message to indicate successful Encoder reset
    telemetry.addData("Starting at",  "%7d :%7d",
```

```
                frontLeftDrive.getCurrentPosition(),
                frontRightDrive.getCurrentPosition(),
                backLeftDrive.getCurrentPosition(),
                backRightDrive.getCurrentPosition());
        telemetry.update();
        waitForStart();




        // Step through each leg of the path,
        encoderDrive(DRIVE_SPEED,  5,  5, 5.0);
        // S1: Forward 47 Inches with 5 Sec timeout
        //encoderDrive(TURN_SPEED,   12, -12, 4.0);  // S2: Turn Right 12 Inches with 4 Sec
timeout
        //encoderDrive(DRIVE_SPEED, -24, -24, 4.0);  // S3: Reverse 24 Inches with 4 Sec timeout

        telemetry.addData("Path", "Complete");
        telemetry.update();
        sleep(1000);
        // pause to display final telemetry message.
    }

    /*
     *  Method to perform a relative move, based on encoder counts.
     *  Encoders are not reset as the move is based on the current position.
     *  Move will stop if any of three conditions occur:
     *  1) Move gets to the desired position
     *  2) Move runs out of time
     *  3) Driver stops the opmode running.
     */
    /*public void encoderDrive(double speed,
                             double leftInches, double rightInches,
                             double timeoutS) {
        int newLeftTarget;
        int newRightTarget;

        // Ensure that the opmode is still active
        if (opModeIsActive()) {
```

```
        // Determine new target position, and pass to motor controller
        newLeftTarget = frontLeftDrive.getCurrentPosition() + (int)(leftInches *
COUNTS_PER_INCH);
        newRightTarget = frontRightDrive.getCurrentPosition() + (int)(rightInches *
COUNTS_PER_INCH);
        frontLeftDrive.setTargetPosition(newLeftTarget);
        backLeftDrive.setTargetPosition(newLeftTarget);
        frontRightDrive.setTargetPosition(newRightTarget);
        backRightDrive.setTargetPosition(newRightTarget);

        // Turn On RUN_TO_POSITION
        backLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        backRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontLeftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        frontRightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        frontLeftDrive.setPower(Math.abs(speed));
        frontRightDrive.setPower(Math.abs(speed));
        backLeftDrive.setPower(Math.abs(speed));
        backRightDrive.setPower(Math.abs(speed));


        while (opModeIsActive() &&
             (runtime.seconds() < timeoutS) &&
             (frontLeftDrive.isBusy() && frontRightDrive.isBusy()&&backLeftDrive.isBusy()
&& backRightDrive.isBusy())) {

            // Display it for the driver.
            telemetry.addData("Running to",  " %7d :%7d", newLeftTarget,  newRightTarget);
            telemetry.addData("Currently at",  " at %7d :%7d",
                                frontLeftDrive.getCurrentPosition(),
frontRightDrive.getCurrentPosition());
            telemetry.update();
        }

        // Stop all motion;
        frontLeftDrive.setPower(0);
        backLeftDrive.setPower(0);
```

```
            frontRightDrive.setPower(0);
            backRightDrive.setPower(0);

            // Turn off RUN_TO_POSITION
            frontLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            backLeftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
            frontRightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

            sleep(250);   // optional pause after each move.
        }
    }
}*/
```

Untouched_Original_modell.java

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

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import java.util.List;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.vision.VisionPortal;
import org.firstinspires.ftc.vision.tfod.TfodProcessor;

@Autonomous(name = "Untouched version")
//@Disabled
public class Unotuched_Original_modell extends LinearOpMode {

  boolean USE_WEBCAM;
  TfodProcessor myTfodProcessor;
  VisionPortal myVisionPortal;

  /**
   * This function is executed when this OpMode is selected from the Driver Station.
   */
```

```
@Override
public void runOpMode() {
  // This 2023-2024 OpMode illustrates the basics of TensorFlow Object Detection, using
  // Original Modell
  USE_WEBCAM = true;
  // Initialize TFOD before waitForStart.
  initTfod();
  // Wait for the match to begin.
  telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
  telemetry.addData(">", "Touch Play to start OpMode");
  telemetry.update();
  waitForStart();
  if (opModeIsActive()) {
    // Put run blocks here.
    while (opModeIsActive()) {
      // Put loop blocks here.
      telemetryTfod();
      // Push telemetry to the Driver Station.
      telemetry.update();
      if (gamepad1.dpad_down) {
        // Temporarily stop the streaming session.
        myVisionPortal.stopStreaming();
      } else if (gamepad1.dpad_up) {
        // Resume the streaming session if previously stopped.
        myVisionPortal.resumeStreaming();
      }
      // Share the CPU.
      sleep(20);
    }
  }
}

/**
 * Initialize TensorFlow Object Detection.
 */
private void initTfod() {
  TfodProcessor.Builder myTfodProcessorBuilder;
  VisionPortal.Builder myVisionPortalBuilder;

  // First, create a TfodProcessor.Builder.
```

```
    myTfodProcessorBuilder = new TfodProcessor.Builder();
    // Set the name of the file where the model can be found.
    myTfodProcessorBuilder.setModelFileName("Original Modell");
    // Set the full ordered list of labels the model is trained to recognize.
    myTfodProcessorBuilder.setModelLabels(JavaUtil.createListWith("Blue Warrior", "Red
Warrior"));
    // Set the aspect ratio for the images used when the model was created.
    myTfodProcessorBuilder.setModelAspectRatio(16 / 9);
    // Create a TfodProcessor by calling build.
    myTfodProcessor = myTfodProcessorBuilder.build();
    // Next, create a VisionPortal.Builder and set attributes related to the camera.
    myVisionPortalBuilder = new VisionPortal.Builder();
    if (USE_WEBCAM) {
      // Use a webcam.
      myVisionPortalBuilder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
    } else {
      // Use the device's back camera.
      myVisionPortalBuilder.setCamera(BuiltinCameraDirection.BACK);
    }
    // Add myTfodProcessor to the VisionPortal.Builder.
    myVisionPortalBuilder.addProcessor(myTfodProcessor);
    // Create a VisionPortal by calling build.
    myVisionPortal = myVisionPortalBuilder.build();
  }

  /**
   * Display info (using telemetry) for a detected object
   */
  private void telemetryTfod() {
    List<Recognition> myTfodRecognitions;
    Recognition myTfodRecognition;
    float x;
    float y;

    // Get a list of recognitions from TFOD.
    myTfodRecognitions = myTfodProcessor.getRecognitions();
    telemetry.addData("# Objects Detected", JavaUtil.listLength(myTfodRecognitions));
    // Iterate through list and call a function to display info for each recognized object.
    for (Recognition myTfodRecognition_item : myTfodRecognitions) {
      myTfodRecognition = myTfodRecognition_item;
```

```
      // Display info about the recognition.
      telemetry.addLine("");
      // Display label and confidence.
      // Display the label and confidence for the recognition.
      telemetry.addData("Image", myTfodRecognition.getLabel() + " (" +
JavaUtil.formatNumber(myTfodRecognition.getConfidence() * 100, 0) + " % Conf.)");
      // Display position.
      x = (myTfodRecognition.getLeft() + myTfodRecognition.getRight()) / 2;
      y = (myTfodRecognition.getTop() + myTfodRecognition.getBottom()) / 2;
      // Display the position of the center of the detection boundary for the recognition
      telemetry.addData("- Position", JavaUtil.formatNumber(x, 0) + ", " +
JavaUtil.formatNumber(y, 0));
      // Display size
      // Display the size of detection boundary for the recognition
      telemetry.addData("- Size", JavaUtil.formatNumber(myTfodRecognition.getWidth(), 0) + " x
" + JavaUtil.formatNumber(myTfodRecognition.getHeight(), 0));
    }
  }
}
```

Test_VuNav.java

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

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.navigation.AxesOrder;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaBase;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaCurrentGame;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;

@Autonomous(name = "Test_VuNav (Blocks to Java)")
@Disabled
public class Test_VuNav extends LinearOpMode {

  private VuforiaCurrentGame vuforiaPOWERPLAY;
```

```
VuforiaBase.TrackingResults vuforiaResults;

/**
 * This function is executed when this Op Mode is selected from the Driver Station.
 */
/*@Override
public void runOpMode() {
  vuforiaPOWERPLAY = new VuforiaCurrentGame();

  // Initialize Vuforia
  telemetry.addData("Status", "Initializing Vuforia. Please wait...");
  telemetry.update();
  initVuforia();
  // Activate here for camera preview.
  vuforiaPOWERPLAY.activate();
  telemetry.addData("DS preview on/off", "3 dots, Camera Stream");
  telemetry.addData(">>", "Vuforia initialized, press start to continue...");
  telemetry.update();
  waitForStart();
  if (opModeIsActive()) {
    // Put run blocks here.
    while (opModeIsActive()) {
      // Are the targets visible?
      // (Note we only process first visible target).
      if (isTargetVisible("Red Audience Wall")) {
        processTarget();
      } else if (isTargetVisible("Red Rear Wall")) {
        processTarget();
      } else if (isTargetVisible("Blue Audience Wall")) {
        processTarget();
      } else if (isTargetVisible("Blue Rear Wall")) {
        processTarget();
      } else {
        telemetry.addData("No Targets Detected", "Targets are not visible.");
      }
      telemetry.update();
    }
  }
  // Don't forget to deactivate Vuforia.
  vuforiaPOWERPLAY.deactivate();
```

```
  vuforiaPOWERPLAY.close();
}

/**
 * Describe this function...
 */
/*private void initVuforia() {
  // Initialize using external web camera.
  vuforiaPOWERPLAY.initialize(
      "", // vuforiaLicenseKey
      hardwareMap.get(WebcamName.class, "Webcam 1"), // cameraName
      "", // webcamCalibrationFilename
      false, // useExtendedTracking
      true, // enableCameraMonitoring
      VuforiaLocalizer.Parameters.CameraMonitorFeedback.AXES, // cameraMonitorFeedback
      0, // dx
      0, // dy
      0, // dz
      AxesOrder.XZY, // axesOrder
      90, // firstAngle
      90, // secondAngle
      0, // thirdAngle
      true); // useCompetitionFieldTargetLocations
}

/**
 * Check to see if the target is visible.
 */
/*private boolean isTargetVisible(String trackableName) {
  boolean isVisible;

  // Get vuforia results for target.
  vuforiaResults = vuforiaPOWERPLAY.track(trackableName);
  // Is this target visible?
  if (vuforiaResults.isVisible) {
    isVisible = true;
  } else {
    isVisible = false;
  }
```

```
    return isVisible;
  }

  /**
   * This function displays location on the field and rotation about the Z
   * axis on the field. It uses results from the isTargetVisible function.
   */
  /*private void processTarget() {
    // Display the target name.
    telemetry.addData("Target Detected", vuforiaResults.name + " is visible.");
    telemetry.addData("X (in)",
Double.parseDouble(JavaUtil.formatNumber(displayValue(vuforiaResults.x, "IN"), 2)));
    telemetry.addData("Y (in)",
Double.parseDouble(JavaUtil.formatNumber(displayValue(vuforiaResults.y, "IN"), 2)));
    telemetry.addData("Z (in)",
Double.parseDouble(JavaUtil.formatNumber(displayValue(vuforiaResults.z, "IN"), 2)));
    telemetry.addData("Rotation about Z (deg)",
Double.parseDouble(JavaUtil.formatNumber(vuforiaResults.zAngle, 2)));
  }

  /**
   * By default, distances are returned in millimeters by Vuforia.
   * Convert to other distance units (CM, M, IN, and FT).
   */
  /*private double displayValue(float originalValue, String units) {
    double convertedValue;

    // Vuforia returns distances in mm.
    if (units.equals("CM")) {
      convertedValue = originalValue / 10;
    } else if (units.equals("M")) {
      convertedValue = originalValue / 1000;
    } else if (units.equals("IN")) {
      convertedValue = originalValue / 25.4;
    } else if (units.equals("FT")) {
      convertedValue = (originalValue / 25.4) / 12;
    } else {
      convertedValue = originalValue;
    }
    return convertedValue;
```

```
  }
}
*/
```

ConceptVuforiaDriveToTargetWebcammy.java

```
/*package org.firstinspires.ftc.robotcontroller.external.samples;

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

import org.firstinspires.ftc.robotcore.external.ClassFactory;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.matrices.OpenGLMatrix;
import org.firstinspires.ftc.robotcore.external.matrices.VectorF;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaTrackable;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaTrackableDefaultListener;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaTrackables;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import java.util.List;
import org.firstinspires.ftc.robotcore.external.JavaUtil;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.navigation.AxesOrder;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaCurrentGame;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.robotcore.external.tfod.TfodCurrentGame;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaCurrentGame;
import org.firstinspires.ftc.robotcore.external.tfod.TfodCurrentGame;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;
```

```
import org.firstinspires.ftc.robotcore.external.matrices.OpenGLMatrix;
import com.qualcomm.robotcore.hardware.DcMotor;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.ClassFactory;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaTrackables;

 /*
@Disabled
@TeleOp(name="ConceptVuforiaDriveToTargetWebcammy", group = "LinearOpMode")

public class ConceptVuforiaDriveToTargetWebcammy extends LinearOpMode{

       // Adjust these numbers to suit your robot.
   final double DESIRED_DISTANCE = 8.0; //  this is how close the camera should get to the target (inches)
                           //  The GAIN constants set the relationship between the measured position error,
                           //  and how much power is applied to the drive motors.  Drive = Error * Gain
                           //  Make these values smaller for smoother control.
   final double SPEED_GAIN =   0.02 ;   //  Speed Control "Gain". eg: Ramp up to 50% power at a 25 inch error.   (0.50 / 25.0)
   final double TURN_GAIN  =   0.01 ;   //  Turn Control "Gain".  eg: Ramp up to 25% power at a 25 degree error. (0.25 / 25.0)

   final double MM_PER_INCH = 25.40 ;   //  Metric conversion


   private VuforiaCurrentGame vuforiaFreightFrenzy;
   private TfodCurrentGame tfodFreightFrenzy;
   private static final String VUFORIA_KEY =

"ATQoIuv/////AAABmbSW8So4CEI/nuZAx4qqGxdlFkqpPHIqk1nuRc1lN/QDv+kM+e7l75a3i
LgdggfAP92T6kowUufFfi4JeqQaMyV+jrs71U9xyjWCeFzO41OQQSChmE1FCy6BgALxUSW
piEEY7KhSviBPnyjGIRVpqFw9YWGNHNrTEs0I/qn7RhqrIOJB6NgZfnBI+3r+/lZNuSXbladxf
FV40HW1tEa/r4leqME94prGZds4VTEIetYQsCS9gd9l2Oxctq112duojLt/KvdrpWWaZd69/GvN
nWu38pnVuind4XbHvRG82R6MKB2R5IBQLOFI+1ZDK5n4boP3NLnBTbbeMvaskQOSJxfX
eLNuqIYIic6InQ6AQsol";
```

```
    VuforiaLocalizer vuforia    = null;
    OpenGLMatrix targetPose     = null;
    String targetName           = "";

    private DcMotor leftMotor   = null;
    private DcMotor rightMotor  = null;


    Recognition recognition;

    @Override
    public void runOpMode() {
        List<Recognition> reognitions;
        int index;

        vuforiaFreightFrenzy = new VuforiaCurrentGame();
        tfodFreightFrenzy = new TfodCurrentGame();


    // Adjust these numbers to suit your robot.
    final double DESIRED_DISTANCE = 8.0; //  this is how close the camera should get to the target (inches)
                                //  The GAIN constants set the relationship between the measured position error,
                                //  and how much power is applied to the drive motors.  Drive = Error * Gain
                                //  Make these values smaller for smoother control.
    final double SPEED_GAIN =   0.02 ;   //  Speed Control "Gain". eg: Ramp up to 50% power at a 25 inch error.   (0.50 / 25.0)
    final double TURN_GAIN  =   0.01 ;   //  Turn Control "Gain".  eg: Ramp up to 25% power at a 25 degree error. (0.25 / 25.0)

    final double MM_PER_INCH = 25.40 ;   //  Metric conversion

    /*
     * IMPORTANT: You need to obtain your own license key to use Vuforia. The string below with which
     * 'parameters.vuforiaLicenseKey' is initialized is for illustration only, and will not function.
     * A Vuforia 'Development' license key, can be obtained free of charge from the Vuforia developer
```

```
     * web site at https://developer.vuforia.com/license-manager.
     *
     * Vuforia license keys are always 380 characters long, and look as if they contain mostly
     * random data. As an example, here is a example of a fragment of a valid key:
     *      ... yIgIzTqZ4mWjk9wd3cZO9T1axEqzuhxoGlfOOI2dRzKS4T0hQ8kT ...
     * Once you've obtained a license key, copy the string from the Vuforia web site
     * and paste it in to your code on the next line, between the double quotes.
     */
    /*private static final String VUFORIA_KEY =

"ATQoIuv/////AAABmbSW8So4CEI/nuZAx4qqGxdlFkqpPHIqk1nuRc1lN/QDv+kM+e7l75a3i
LgdggfAP92T6kowUufFfi4JeqQaMyV+jrs71U9xyjWCeFzO41OQQSChmE1FCy6BgALxUSW
piEEY7KhSviBPnyjGIRVpqFw9YWGNHNrTEs0I/qn7RhqrIOJB6NgZfnBI+3r+/lZNuSXbladxf
FV40HW1tEa/r4leqME94prGZds4VTEIetYQsCS9gd9l2Oxctq112duojLt/KvdrpWWaZd69/GvN
nWu38pnVuind4XbHvRG82R6MKB2R5IBQLOFI+1ZDK5n4boP3NLnBTbbeMvaskQOSJxfX
eLNuqIYIic6InQ6AQsol";*/

    /*VuforiaLocalizer vuforia    = null;
    OpenGLMatrix targetPose     = null;
    String targetName           = "";*/

    //{
        /*
         * Configure Vuforia by creating a Parameter object, and passing it to the Vuforia engine.
         * To get an on-phone camera preview, use the code below.
         * If no camera preview is desired, use the parameter-less constructor instead (commented
out below).
         */
         /*
        int cameraMonitorViewId =
hardwareMap.appContext.getResources().getIdentifier("cameraMonitorViewId", "id",
hardwareMap.appContext.getPackageName());
        VuforiaLocalizer.Parameters parameters = new
VuforiaLocalizer.Parameters(cameraMonitorViewId);
        //VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.Parameters();

        parameters.vuforiaLicenseKey = VUFORIA_KEY;

        // Turn off Extended tracking.  Set this true if you want Vuforia to track beyond the target.
        parameters.useExtendedTracking = false;
```

```
        // Connect to the camera we are to use.  This name must match what is set up in Robot
Configuration
        parameters.cameraName = hardwareMap.get(WebcamName.class, "Webcam 1");
        this.vuforia = ClassFactory.getInstance().createVuforia(parameters);

        // Load the trackable objects from the Assets file, and give them meaningful names
        VuforiaTrackables targetsFreightFrenzy =
this.vuforia.loadTrackablesFromAsset("FreightFrenzy");
        targetsFreightFrenzy.get(0).setName("Blue Storage");
        targetsFreightFrenzy.get(1).setName("Blue Alliance Wall");
        targetsFreightFrenzy.get(2).setName("Red Storage");
        targetsFreightFrenzy.get(3).setName("Red Alliance Wall");
    }
}



        // Start tracking targets in the background
        /*targetsFreightFrenzy.activate();
        //TFODvuforia

        // Initialize the hardware variables. Note that the strings used here as parameters
        // to 'get' must correspond to the names assigned during the robot configuration
        // step (using the FTC Robot Controller app on the phone).
        leftMotor  = hardwareMap.get(DcMotor.class, "leftMotor");
        rightMotor = hardwareMap.get(DcMotor.class, "rightMotor");

        // To drive forward, most robots need the motor on one side to be reversed, because the
axles point in opposite directions.
        // Pushing the left stick forward MUST make robot go forward. So adjust these two lines
based on your first test drive.
        leftMotor.setDirection(DcMotor.Direction.FORWARD);
        rightMotor.setDirection(DcMotor.Direction.REVERSE);

        telemetry.addData(">", "Press Play to start");
        telemetry.update();

        waitForStart();

//this is in the wrong spot its not correct fix it later
```

```
        boolean targetFound     = false;    // Set to true when a target is detected by Vuforia
        double  targetRange     = 8;        // Distance from camera to target in Inches was changed by
CAMMY from 0 to 8
        double  targetBearing   = 0;        // Robot Heading, relative to target.  Positive degrees means
target is to the right.
        double  drive           = 0;        // Desired forward power (-1 to +1)
        double  turn            = 0;        // Desired turning power (-1 to +1)

        while (opModeIsActive())
        {
           // Look for first visible target, and save its pose.
           targetFound = false;
           for (VuforiaTrackable trackable : targetsFreightFrenzy)
           {
              if (((VuforiaTrackableDefaultListener) trackable.getListener()).isVisible())
              {
                 targetPose =
((VuforiaTrackableDefaultListener)trackable.getListener()).getVuforiaCameraFromTarget();

                 // if we have a target, process the "pose" to determine the position of the target
relative to the robot.
                 if (targetPose != null)
                 {
                    targetFound = true;
                    targetName  = trackable.getName();
                    VectorF trans = targetPose.getTranslation();

                    // Extract the X & Y components of the offset of the target relative to the robot
                    double targetX = trans.get(0) / MM_PER_INCH; // Image X axis
                    double targetY = trans.get(2) / MM_PER_INCH; // Image Z axis

                    // target range is based on distance from robot position to origin (right triangle).
                    targetRange = Math.hypot(targetX, targetY);

                    // target bearing is based on angle formed between the X axis and the target range
line
                    targetBearing = Math.toDegrees(Math.asin(targetX / targetRange));

                    break;  // jump out of target tracking loop if we find a target.
                 }
```

```
            }
        }

        // Tell the driver what we see, and what to do.
        if (targetFound) {
            telemetry.addData(">","HOLD Left-Bumper to Drive to Target\n");
            telemetry.addData("Target", " %s", targetName);
            telemetry.addData("Range",  "%5.1f inches", targetRange);
            telemetry.addData("Bearing","%3.0f degrees", targetBearing);
       /* } else {
            telemetry.addData(">","Drive using joystick to find target\n");
        }*/

        // Drive to target Automatically if Left Bumper is being pressed, AND we have found a
target.
        //if (/*gamepad1.left_bumper &&*/ targetFound) {

            // Determine heading and range error so we can use them to control the robot
automatically.
/*              double  rangeError   = (targetRange - DESIRED_DISTANCE);
            double  headingError = targetBearing;

            // Use the speed and turn "gains" to calculate how we want the robot to move.
            drive = rangeError * SPEED_GAIN;
            turn  = headingError * TURN_GAIN ;

            telemetry.addData("Auto","Drive %5.2f, Turn %5.2f", drive, turn);
        /*} else {

            // drive using manual POV Joystick mode.
            drive = -gamepad1.left_stick_y  / 2.0;  // Reduce drive rate to 50%.
            turn  =  gamepad1.right_stick_x / 4.0;  // Reduce turn rate to 25%.
            telemetry.addData("Manual","Drive %5.2f, Turn %5.2f", drive, turn);
        }*///"
      /*   telemetry.update();

        // Calculate left and right wheel powers and send to them to the motors.
        double leftPower    = Range.clip(drive + turn, -1.0, 1.0) ;
        double rightPower   = Range.clip(drive - turn, -1.0, 1.0) ;
        leftMotor.setPower(leftPower);
```

```
            rightMotor.setPower(rightPower);

            sleep(10);
                  //}
               }
            }
         }
      }
   }*/
```