```
/* ****************************************************************************************
 * Copyright (c) 2019 Edward C. Epp. All rights reserved.
 *
 * Ed C. Epp 9-2019 - Preliminary
 * Locate Gold and Push
 *
 * Locate the gold mineral using Tensorflow and push it. This is a linear version.
 *
 * Do not redistribute. This code has not been reviewed/
 *****************************************************************************************/

package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.vuforia.Vuforia;

import org.firstinspires.ftc.robotcore.external.ClassFactory;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer.CameraDirection;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.robotcore.external.tfod.TFObjectDetector;

import java.util.List;

// I don’t understand why this is necessary
import static org.firstinspires.ftc.teamcode.LocateGoldAndPushLinear.RobotState.DONE;
import static org.firstinspires.ftc.teamcode.LocateGoldAndPushLinear.RobotState.MOVE_GOLD;
import static org.firstinspires.ftc.teamcode.LocateGoldAndPushLinear.RobotState.TARGET_GOLD;
import static org.firstinspires.ftc.teamcode.LocateGoldAndPushLinear.RobotState.TEST;
import static org.firstinspires.ftc.teamcode.LocateGoldAndPushLinear.RobotState.ERROR;

@Autonomous(name = "Locate Gold And Push Linear", group = "Concept")
//@Disabled
public class LocateGoldAndPushLinear extends LinearOpMode {

    /***************************** constants *************************/
    /***************************** constants *************************/

    private static final String TFOD_MODEL_ASSET     = "RoverRuckus.tflite";
    private static final String LABEL_GOLD_MINERAL   = "Gold Mineral";
    private static final String LABEL_SILVER_MINERAL = "Silver Mineral";

    private static final int SCREEN_WIDTH        = 1280;
    private static final int SCREEN_HEIGHT       =  720;
    private static final int POINTING_TOLERANCE  =   50;

    private static final double LOW_POWER        = 0.15;
    private static final double MID_SPEED        = 0.25;
    private static final int CLICKS_TO_TARGET    =  500;

    private static final int COUNTS_PER_ROTATION = 670;

    public enum RobotState {
        TARGET_GOLD,
        MOVE_GOLD,
        DONE,
        TEST,
        ERROR,
    }

    // This Vuforia key is for exclusive use by Ed C. Epp
    private static final String VUFORIA_KEY = "-- Key Removed --";

    /***************************** member variables ******************/
    /***************************** member variables ******************/

    // Stores the apps execution state
    // private RobotState myRobotState = TEST;
    private RobotState myRobotState = TARGET_GOLD;

    // The Vuforia localization engine.
    private VuforiaLocalizer myVuforia = null;

    // The Tensor Flow Object Detection engine.
    private TFObjectDetector myTfod = null;

    // links to the physical robot driver motors
    DcMotor myLeftMotor  = null;
    DcMotor myRightMotor = null;

    /***************************** runOpMode *************************/
    /***************************** runOpMode *************************/
```

```java
    /****************************** runOpMode *************************/
    // The robot execution loop and state machine
    @Override
    public void runOpMode() {

        initRobot();

        /** Wait for the game to begin */
        telemetry.addData(">", "Press Play to start tracking");
        telemetry.update();
        waitForStart();

        // Main Linear OpMod
        while (opModeIsActive()) {

            switch (myRobotState) {
                case TARGET_GOLD:
                    targetGold();
                    break;
                case MOVE_GOLD:
                    moveToGold();
                    break;
                case DONE:
                    shutdown();
                    break;
                case TEST:
                    moveFor(200,200);
                    moveFor(-200, 200);
                    myRobotState = DONE;
                    break;
                case ERROR:
                    myRobotState = DONE;
                    break;
                default: {
                    telemetry.addData("Error", "This program should never be here");
                    myRobotState = ERROR;
                }
            }
        }
    }

    //****************************** initRobot *************************
    //****************************** initRobot *************************
    // Initialize the Vuforia Localization Engine, TensorFlow Object Detection, and motors.
    // Vuforia is required for the cameras

    private void initRobot() {

        initVuforia();

        if (myRobotState != ERROR) {
            initTfod();
        }

        if (myRobotState != ERROR) {
            initMotors();
        }
        // Tell the driver that initialization is complete.
        telemetry.addData("Status", "Initialized");
    }

    // ********** initVuforia helper
    // Configure Vuforia by creating a Parameter object, and passing it to the Vuforia engine.
    // Configure the phone to use the rear camera.

    private void initVuforia() {
        VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.Parameters();

        parameters.vuforiaLicenseKey = VUFORIA_KEY;
        parameters.cameraDirection = CameraDirection.BACK;

        //  Instantiate the Vuforia engine
        myVuforia = ClassFactory.getInstance().createVuforia(parameters);

        if (myVuforia == null){
            myRobotState = ERROR;
            telemetry.addData("ERROR", "the Vuforia engine did not initialize");
        }
    }

    // ********** initTfod helper
    // Initialize the Tensor Flow Object Detection engine.

    private void initTfod() {

        if (ClassFactory.getInstance().canCreateTFObjectDetector()) {
            int tfodMonitorViewId = hardwareMap.appContext.getResources().getIdentifier(
                    "tfodMonitorViewId", "id", hardwareMap.appContext.getPackageName());
            TFObjectDetector.Parameters tfodParameters =
                   new TFObjectDetector.Parameters(tfodMonitorViewId);
            myTfod = ClassFactory.getInstance().createTFObjectDetector(tfodParameters, myVuforia);
            myTfod.loadModelFromAsset(TFOD_MODEL_ASSET, LABEL_GOLD_MINERAL, LABEL_SILVER_MINERAL);

            if (myTfod != null) {
                myTfod.activate();
            } else {
                telemetry.addData("ERROR", "TensorFlow lite did not activate");
                myRobotState = ERROR;
            }
        }

        else {
            telemetry.addData("ERROR", "This device is not compatible with TFOD");
            myRobotState = ERROR;
        }
    }

    // ********** initMotors helper
    //Initialize the drive motors.

    private void initMotors () {

        // Set up drive motors
        myLeftMotor = hardwareMap.dcMotor.get("mLeftMotor");
        myRightMotor = hardwareMap.dcMotor.get("mRightMotor");
        //myRightMotor.setDirection(DcMotor.Direction.REVERSE);

        // reset encoder count
        myLeftMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        myRightMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    }

    //****************************** targetGold ************************
    //****************************** targetGold  ***********************
    // Turn the robot to face the gold mineral
    private void targetGold () {
        //telemetry.addData("State: ", "Target Gold");
        //telemetry.update();
        Recognition goldPiece = null;

        // Return without changing state if there is no new information.
        List<Recognition> updatedRecognitions = myTfod.getUpdatedRecognitions();
        if (updatedRecognitions != null) {
            // Find the first gold piece if there is one
            telemetry.addData("State: ", "Target Gold");
            //telemetry.update();
            for (Recognition recognition : updatedRecognitions) {
                if (recognition.getLabel().equals(LABEL_GOLD_MINERAL)) {
                    goldPiece = recognition;
                    break;
                }
            }

            // we found one
            if (goldPiece != null) {
                int goldMineralLeftX = (int) goldPiece.getLeft();
                int goldMineralRightX = (int) goldPiece.getLeft();
                int goldMineralCenterX = (goldMineralLeftX + goldMineralRightX) / 2;
                int error = goldMineralCenterX - SCREEN_WIDTH / 2;

                if (Math.abs(error) < POINTING_TOLERANCE) {
                    myRobotState = MOVE_GOLD;
                } else {
                    telemetry.addData("Action: ", "Turn " + error);
                    telemetry.update();
                    int turn_clicks = error / 8;
                    moveFor(turn_clicks, -turn_clicks);
                }
            } else {
                telemetry.addData("Status: ", "No gold found");
                telemetry.update();
            }
        } else {
            idle();
        }
    }

    /****************************** moveToGold ************************/
    /****************************** moveToGold ************************/
    // Move forward CLICKS_TO_TARGET
```

```
    private void moveToGold(){
        telemetry.addData("State: ", "Moving to Gold");
        telemetry.update();

        moveFor(CLICKS_TO_TARGET, CLICKS_TO_TARGET);

        myRobotState = DONE;
    }

    // ********** moveFor helper
    // Turn each motor for a given number of counts
    //    leftCount:  the number of counts and direction to turn the left wheel
    //    rightCount:                                              right wheel
    // if the leftCount is less than the right count the robot will turn left
    // if the rightCount is less than the left count the robot will turn right

    private void moveFor (int leftCount, int rightCount){
        myLeftMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        myRightMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        myLeftMotor.setTargetPosition(leftCount);
        myRightMotor.setTargetPosition(rightCount);
        myLeftMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        myRightMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        myLeftMotor.setPower(MID_SPEED);
        myRightMotor.setPower(MID_SPEED);
        while (opModeIsActive() && (myLeftMotor.isBusy() || myRightMotor.isBusy())) {
            idle();
        }
    }


    /****************************** oneRotation ***********************/
    /****************************** oneRotation ************************/
    // one rotation clockwise
    private void oneRotation(){
        telemetry.addData("State: ", "Test oneRotation");
        telemetry.update();

        // reset encoder count kept by motors.
        myLeftMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        myRightMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        // set motors to run to target encoder position and stop with brakes on.
        myLeftMotor.setTargetPosition(COUNTS_PER_ROTATION);
        myRightMotor.setTargetPosition(-COUNTS_PER_ROTATION);
        myLeftMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        myRightMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        myLeftMotor.setPower(MID_SPEED);
        myRightMotor.setPower(MID_SPEED);

        while (opModeIsActive() && (myLeftMotor.isBusy() || myRightMotor.isBusy())) {
            idle();
        }

        myRobotState = DONE;
    }

    /****************************** shutdown *************************/
    /****************************** shutdown *************************/
    // Turn the motor power off and shutdown the TensorFlow Object Detection Engine
    public void shutdown()
    {
        telemetry.addData("State: ", "Done");

        myLeftMotor.setPower(0.0);
        myRightMotor.setPower(0.0);

        if (myTfod != null)
        {
            myTfod.shutdown();
        }

        telemetry.update();
    }
}
```