/\* Copyright (c) 2018 FIRST. All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without modification,

\* are permitted (subject to the limitations in the disclaimer below) provided that

\* the following conditions are met:

\*

\* Redistributions of source code must retain the above copyright notice, this list

\* of conditions and the following disclaimer.

\*

\* Redistributions in binary form must reproduce the above copyright notice, this

\* list of conditions and the following disclaimer in the documentation and/or

\* other materials provided with the distribution.

\*

\* Neither the name of FIRST nor the names of its contributors may be used to endorse or

\* promote products derived from this software without specific prior written permission.

\*

\* NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED BY THIS

\* LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

\* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

\* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

\* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE

\* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL

\* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR

\* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER

\* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,

\* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;

import com.qualcomm.robotcore.hardware.DcMotor;

import com.qualcomm.robotcore.hardware.CRServo;

import com.qualcomm.robotcore.hardware.Servo;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;

import com.qualcomm.hardware.bosch.BNO055IMU;

import com.qualcomm.hardware.bosch.JustLoggingAccelerationIntegrator;

import org.firstinspires.ftc.robotcore.external.navigation.Acceleration;

import org.firstinspires.ftc.robotcore.external.navigation.Orientation;

import org.firstinspires.ftc.robotcore.external.navigation.AxesReference;

import org.firstinspires.ftc.robotcore.external.navigation.AxesOrder;

import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;

import java.util.Locale;

import java.util.List;

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.TFObjectDetector;

import org.firstinspires.ftc.robotcore.external.tfod.Recognition;

import org.firstinspires.ftc.robotcore.external.Func;

/\*\*

\* This 2018-2019 OpMode illustrates the basics of using the TensorFlow Object Detection API to

\* determine the position of the gold and silver minerals.

\*

\* 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.

\*

\* IMPORTANT: In order to use this OpMode, you need to obtain your own Vuforia license key as

\* is explained below.

\*/

@Autonomous(name = "Mineral Auto 2", group = "Concept")

public class MineralAuto2 extends LinearOpMode {

//declares all motors

private DcMotor leftFrontDrive, rightFrontDrive, leftBackDrive, rightBackDrive, rightLowerArm, leftLowerArm, upperArmMotor;

//declares all servos

CRServo leftIntakeServo, rightIntakeServo, hangingServo;

Servo mineralServo;

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";

BNO055IMU imu;

// State used for updating telemetry

Orientation angles;

Acceleration gravity;

/\*

\* 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 = "AYWGcTj/////AAABmW8QlFyADE57slQ+RlRh3jhmcaAcY34hi2fx+1MbhOj2tSubwzptbd27EjrpAy3z9w3AoCXWfzUOat4BHvYLvjo0wl78Z47YW1JhzAo5+R3M9QxzX4LyrGn99D67ZxjcnWSukyRsSeIyKW3sPMIDEXfmX3D17OS6otvKFlaHOAvZNMvKeKLG9slQuyv2RLW5YronBwsFJRy1Llo/an/yJ7/2o2XU3OtHL7u5BrEKqLC8PX/pxAfXEUm79/tjcuoP9/CdR+YBuKUZ1G++ki6ZIfPTut3HVI4620Z6MyFSs9n2A4JabHc02/YClitim6yYU0tpWkd+0RuhiN0q4EjS5RDfJZK84NDbrbwhsbpTmbOM";

/\*\*

\* {@link #vuforia} is the variable we will use to store our instance of the Vuforia

\* localization engine.

\*/

private VuforiaLocalizer vuforia;

/\*\*

\* {@link #tfod} is the variable we will use to store our instance of the Tensor Flow Object

\* Detection engine.

\*/

private TFObjectDetector tfod;

@Override

public void runOpMode() {

// The TFObjectDetector uses the camera frames from the VuforiaLocalizer, so we create that

// first.

initVuforia();

if (ClassFactory.getInstance().canCreateTFObjectDetector()) {

initTfod();

} else {

telemetry.addData("Sorry!", "This device is not compatible with TFOD");

}

// Set up the parameters with which we will use our IMU. Note that integration

// algorithm here just reports accelerations to the logcat log; it doesn't actually

// provide positional information.

BNO055IMU.Parameters parameters = new BNO055IMU.Parameters();

parameters.angleUnit = BNO055IMU.AngleUnit.DEGREES;

parameters.accelUnit = BNO055IMU.AccelUnit.METERS\_PERSEC\_PERSEC;

parameters.calibrationDataFile = "BNO055IMUCalibration.json"; // see the calibration sample opmode

parameters.loggingEnabled = true;

parameters.loggingTag = "IMU";

parameters.accelerationIntegrationAlgorithm = new JustLoggingAccelerationIntegrator();

// Retrieve and initialize the IMU. We expect the IMU to be attached to an I2C port

// on a Core Device Interface Module, configured to be a sensor of type "AdaFruit IMU",

// and named "imu".

imu = hardwareMap.get(BNO055IMU.class, "imu");

imu.initialize(parameters);

// Set up our telemetry dashboard

composeTelemetry();

//initializes motors

leftFrontDrive = hardwareMap.get(DcMotor.class, "left\_front\_drive");

rightFrontDrive = hardwareMap.get(DcMotor.class, "right\_front\_drive");

leftBackDrive = hardwareMap.get(DcMotor.class, "left\_back\_drive");

rightBackDrive = hardwareMap.get(DcMotor.class,"right\_back\_drive");

rightLowerArm = hardwareMap.get(DcMotor.class,"right\_lower\_arm");

leftLowerArm = hardwareMap.get(DcMotor.class, "left\_lower\_arm");

upperArmMotor = hardwareMap.get(DcMotor.class,"upper\_arm\_drive");

//iniializes Servos

rightIntakeServo = hardwareMap.get(CRServo.class,"right\_intake\_servo"); // <-- Coninuous servo

leftIntakeServo = hardwareMap.get(CRServo.class,"left\_intake\_servo"); // <-- Coninuous servo

mineralServo = hardwareMap.get(Servo.class, "mineral\_servo");

//sets direction of motors

leftFrontDrive.setDirection(DcMotor.Direction.FORWARD);

rightFrontDrive.setDirection(DcMotor.Direction.FORWARD);

leftBackDrive.setDirection(DcMotor.Direction.FORWARD);

rightBackDrive.setDirection(DcMotor.Direction.FORWARD);

rightLowerArm.setDirection(DcMotor.Direction.REVERSE);

leftLowerArm.setDirection(DcMotor.Direction.FORWARD);

upperArmMotor.setDirection(DcMotor.Direction.FORWARD);

//set initial speeds of motor

leftFrontDrive.setPower(0.0);

leftBackDrive.setPower(0.0);

rightFrontDrive.setPower(0.0);

rightBackDrive.setPower(0.0);

rightLowerArm.setPower(0.0);

leftLowerArm.setPower(0.0);

upperArmMotor.setPower(0.0);

//set inital speeds of all Continuous Rotation servos

leftIntakeServo.setPower(0.02); // 0.02 is not moving for continuous servo

rightIntakeServo.setPower(0.02);

hangingServo.setPower(0.02);

//initial position of servo

mineralServo.setPosition(0.0);

/\*\* Wait for the game to begin \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

telemetry.addData(">", "Press Play to start tracking");

telemetry.update();

waitForStart();

if (opModeIsActive()) {

/\*\* Activate Tensor Flow Object Detection. \*/

if (tfod != null) {

tfod.activate();

}

while (opModeIsActive()) {

//using gryo values, move robot until it can see all 3 minerals

if (tfod != null) {

// getUpdatedRecognitions() will return null if no new information is available since

// the last time that call was made.

List<Recognition> updatedRecognitions = tfod.getUpdatedRecognitions();

if (updatedRecognitions != null) {

telemetry.addData("# Object Detected", updatedRecognitions.size());

if (updatedRecognitions.size() == 3) {

int goldMineralX = -1;

int silverMineral1X = -1;

int silverMineral2X = -1;

for (Recognition recognition : updatedRecognitions) {

if (recognition.getLabel().equals(LABEL\_GOLD\_MINERAL)) {

goldMineralX = (int) recognition.getLeft();

} else if (silverMineral1X == -1) {

silverMineral1X = (int) recognition.getLeft();

} else {

silverMineral2X = (int) recognition.getLeft();

}

}

if (goldMineralX != -1 && silverMineral1X != -1 && silverMineral2X != -1) {

//code goes here when the minerals are located and detected

/\*

locate where the mineral is on the feild

turn or orient robot towards the mineral

lower mineral manipulataor (the push servo)

drive forward enough to push mineral out of the way

\*/

if (goldMineralX < silverMineral1X && goldMineralX < silverMineral2X) {

//gold mineral is on the left

telemetry.addData("Gold Mineral Position", "Left");

} else if (goldMineralX > silverMineral1X && goldMineralX > silverMineral2X) {

//gold mineral is on the right

telemetry.addData("Gold Mineral Position", "Right");

} else {

//gold mineral is in the center

telemetry.addData("Gold Mineral Position", "Center");

}

}

}

telemetry.update();

}

}

}

}

if (tfod != null) {

tfod.shutdown();

}

}

/\*\*

\* Initialize the Vuforia localization engine.

\*/

private void initVuforia() {

/\*

\* Configure Vuforia by creating a Parameter object, and passing it to the Vuforia engine.

\*/

VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.Parameters();

parameters.vuforiaLicenseKey = VUFORIA\_KEY;

parameters.cameraDirection = CameraDirection.BACK;

// Instantiate the Vuforia engine

vuforia = ClassFactory.getInstance().createVuforia(parameters);

// Loading trackables is not necessary for the Tensor Flow Object Detection engine.

}

/\*\*

\* Initialize the Tensor Flow Object Detection engine.

\*/

private void initTfod() {

int tfodMonitorViewId = hardwareMap.appContext.getResources().getIdentifier(

"tfodMonitorViewId", "id", hardwareMap.appContext.getPackageName());

TFObjectDetector.Parameters tfodParameters = new TFObjectDetector.Parameters(tfodMonitorViewId);

tfod = ClassFactory.getInstance().createTFObjectDetector(tfodParameters, vuforia);

tfod.loadModelFromAsset(TFOD\_MODEL\_ASSET, LABEL\_GOLD\_MINERAL, LABEL\_SILVER\_MINERAL);

}

void composeTelemetry() {

// At the beginning of each telemetry update, grab a bunch of data

// from the IMU that we will then display in separate lines.

telemetry.addAction(new Runnable() { @Override public void run()

{

// Acquiring the angles is relatively expensive; we don't want

// to do that in each of the three items that need that info, as that's

// three times the necessary expense.

angles = imu.getAngularOrientation(AxesReference.INTRINSIC, AxesOrder.ZYX, AngleUnit.DEGREES);

gravity = imu.getGravity();

}

});

telemetry.addLine()

.addData("status", new Func<String>() {

@Override public String value() {

return imu.getSystemStatus().toShortString();

}

})

.addData("calib", new Func<String>() {

@Override public String value() {

return imu.getCalibrationStatus().toString();

}

});

telemetry.addLine()

.addData("heading", new Func<String>() {

@Override public String value() {

return formatAngle(angles.angleUnit, angles.firstAngle);

}

})

.addData("roll", new Func<String>() {

@Override public String value() {

return formatAngle(angles.angleUnit, angles.secondAngle);

}

})

.addData("pitch", new Func<String>() {

@Override public String value() {

return formatAngle(angles.angleUnit, angles.thirdAngle);

}

});

telemetry.addLine()

.addData("grvty", new Func<String>() {

@Override public String value() {

return gravity.toString();

}

})

.addData("mag", new Func<String>() {

@Override public String value() {

return String.format(Locale.getDefault(), "%.3f",

Math.sqrt(gravity.xAccel\*gravity.xAccel

+ gravity.yAccel\*gravity.yAccel

+ gravity.zAccel\*gravity.zAccel));

}

});

}

//----------------------------------------------------------------------------------------------

// Formatting

//----------------------------------------------------------------------------------------------

String formatAngle(AngleUnit angleUnit, double angle) {

return formatDegrees(AngleUnit.DEGREES.fromUnit(angleUnit, angle));

}

String formatDegrees(double degrees){

return String.format(Locale.getDefault(), "%.1f", AngleUnit.DEGREES.normalize(degrees));

}

}