package org.firstinspires.ftc.teamcode;

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

import com.qualcomm.robotcore.hardware.CRServo;

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

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

import com.qualcomm.robotcore.hardware.DistanceSensor;

import com.qualcomm.robotcore.hardware.DcMotorController;

import com.qualcomm.robotcore.hardware.Servo;

import com.qualcomm.robotcore.hardware.DcMotor;

import com.qualcomm.hardware.bosch.BNO055IMU;

import java.util.List;

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

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

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

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

import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer.CameraDirection;

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

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

import com.qualcomm.robotcore.util.ElapsedTime;

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;

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

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

import java.util.Locale;

/\*\*

\* This file illustrates the concept of driving a path based on time.

\* It uses the common Pushbot hardware class to define the drive on the robot.

\* The code is structured as a LinearOpMode

\*

\* The code assumes that you do NOT have encoders on the wheels,

\* otherwise you would use: PushbotAutoDriveByEncoder;

\*

\* The desired path in this example is:

\* - Drive forward for 3 seconds

\* - Spin right for 1.3 seconds

\* - Drive Backwards for 1 Second

\* - Stop and close the claw.

\*

\* The code is written in a simple form with no optimizations.

\* However, there are several ways that this type of sequence could be streamlined,

\*

\* Use Android Studios 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="Autonomus Bun - crater", group="Autonomus")

public class AutoCrater extends LinearOpMode {

private DcMotor leftDrive = null;

private DcMotor rightDrive = null;

private DcMotor liftMotor = null;

public DcMotor armMotor1 = null;

public DcMotor armMotor2 = null;

private Servo marker = null;

BNO055IMU imu;

private ElapsedTime runtime = new ElapsedTime();

Orientation lastAngles = new Orientation();

double globalAngle, power = .50, correction;

private DistanceSensor distSensor = null;

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 int deltaTicksLeft = 0;

private int deltaTicksRight = 0;

static final double FORWARD\_SPEED = 0.6;

static final double TURN\_SPEED = 0.35;

private static final String VUFORIA\_KEY = "Aa7fyjT/////AAABmdBjW+SzNUcbmHd4VMolvgMIV2UPiQzKUg0Rffbd5DYjaftc6UOM+kYuUPQPuClJoONQ6LmkDNuBPTWTXN+831hmi9LrJyd3/HX2ksqY5yGVw4FjXLToDchnS99xmaq0FHNsodd9njZBf97OD6NyTT7fprjxNY3hgFYlXEBoy5+0cv+GoOr+M/L72i3CodKDMwLEsCTmRuglWQK8MshnqgLYfLihioRO8JJ6afeMlQg9PUjCxnxS8+mqkpRcqLBBBvTwZNnYfeFr1oOqIe9kx1hBzrVtsRQLK5+frx82htoRQTlZ8RsF04ZXSu0geNcP+xDlxf9DVL3G71pC/+gMIWncO9PGjkbegzeqXDTPxFH0";

private VuforiaLocalizer vuforia;

private TFObjectDetector tfod;

private boolean goldMoved = false;

private int goldPosition=0;

@Override

public void runOpMode() {

/\*

\* Initialize the drive system variables.

\* The init() method of the hardware class does all the work here

\*/

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

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

liftMotor = hardwareMap.get(DcMotor.class, "lift\_motor");

armMotor1 = hardwareMap.get(DcMotor.class, "arm\_motor1");

armMotor2 = hardwareMap.get(DcMotor.class, "arm\_motor2");

distSensor = hardwareMap.get(DistanceSensor.class, "dist\_sensor");

leftDrive.setMode(DcMotor.RunMode.RESET\_ENCODERS); //Resets encoders

rightDrive.setMode(DcMotor.RunMode.RESET\_ENCODERS);

liftMotor.setMode(DcMotor.RunMode.RESET\_ENCODERS);

marker = hardwareMap.servo.get("duciuc");

while(leftDrive.getCurrentPosition() != 0 || rightDrive.getCurrentPosition() != 0) { //Ensures encoders are zero

leftDrive.setMode(DcMotor.RunMode.RESET\_ENCODERS);

rightDrive.setMode(DcMotor.RunMode.RESET\_ENCODERS);

liftMotor.setMode(DcMotor.RunMode.RESET\_ENCODERS);

sleep(50);

}

rightDrive.setDirection(DcMotor.Direction.REVERSE);

leftDrive.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

rightDrive.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

liftMotor.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

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

parameters.mode = BNO055IMU.SensorMode.IMU;

parameters.angleUnit = BNO055IMU.AngleUnit.DEGREES;

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

parameters.loggingEnabled = false;

// 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);

initVuforia();

telemetry.addData("Mode", "calibrating...");

telemetry.update();

// make sure the imu gyro is calibrated before continuing.

while (!isStopRequested() &&opModeIsActive()&& !imu.isGyroCalibrated())

{

sleep(50);

idle();

}

telemetry.addData("Mode", "waiting for start");

telemetry.addData("imu calib status", imu.getCalibrationStatus().toString());

telemetry.update();

// wait for start button.

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

initTfod();

} else {

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

}

telemetry.addData("Status", "Ready to run");

telemetry.update();

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

telemetry.update();

waitForStart();

//deploy\_mk();

//detect the mineral position

if (tfod != null)

tfod.activate();

sleep(500);

detect\_mineral();

telemetry.addData("Gold Position: ",goldPosition);

if (tfod != null) {

tfod.shutdown();

}

telemetry.update();

unlatch();

if(goldPosition==0){

rotate(20,TURN\_SPEED);

runForward(750);

runBackward(250);

rotate(50,TURN\_SPEED);

runForward(1050);

rotate(42,TURN\_SPEED);

runForward(1000);

rotate(173,TURN\_SPEED);

deploy\_mk();

runForward(1500);

}

if(goldPosition==1){

runForward(650);

runBackward(250);

rotate(80,TURN\_SPEED);

runForward(1200);

rotate(38,TURN\_SPEED);

runForward(1000);

rotate(175,TURN\_SPEED);

deploy\_mk();

runForward(1550);

}

if(goldPosition==2){

rotate(-20,TURN\_SPEED);

runForward(700);

runBackward(200);

rotate(110,TURN\_SPEED);

runForward(1300);

rotate(35,TURN\_SPEED);

runForward(1100);

rotate(173,TURN\_SPEED);

deploy\_mk();

runForward(1550);

}

/\*armMotor.setTargetPosition(1440);

armMotor.setPower(0.6);

while(armMotor.isBusy() && opModeIsActive()){

};

armMotor.setPower(0);\*/

}

private void detect\_mineral(){

boolean detected = false;

runtime.reset();

while(opModeIsActive()&& !detected && runtime.seconds()< 6){

sleep(500);

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

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

telemetry.update();

if (updatedRecognitions.size()>=2) {

detected = true;

int goldMineralX = -1;

int silverMineralX = -1;

for (Recognition recognition : updatedRecognitions) {

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

goldMineralX=(int) recognition.getLeft();

}

else{

silverMineralX=(int) recognition.getLeft();

}

}

if(goldMineralX == -1) goldPosition=0;

else{

if(goldMineralX<silverMineralX)goldPosition=1;

else goldPosition=2;

}

telemetry.update();

}

}

return;

}

private void unlatch(){

//desprindere

//while(liftMotor.isBusy() && opModeIsActive()){}

liftMotor.setTargetPosition(-23004);

liftMotor.setPower(-1);

while(liftMotor.isBusy() && opModeIsActive()){};

liftMotor.setPower(0);

rotate(-70,power);

}

private void deploy\_mk(){

marker.setPosition(0);

sleep(1000);

}

private void resetAngle()

{

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

globalAngle = 0;

}

private double getAngle()

{

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

double deltaAngle = angles.firstAngle - lastAngles.firstAngle;

if (deltaAngle < -180)

deltaAngle += 360;

else if (deltaAngle > 180)

deltaAngle -= 360;

globalAngle += deltaAngle;

lastAngles = angles;

return globalAngle;

}

private void rotate(int degrees, double power)

{

leftDrive.setMode(DcMotor.RunMode.RUN\_WITHOUT\_ENCODER);

rightDrive.setMode(DcMotor.RunMode.RUN\_WITHOUT\_ENCODER);

double leftPower, rightPower;

resetAngle();

if (degrees < 0)

{

leftPower = -power;

rightPower = power;

}

else if (degrees > 0)

{

leftPower = power;

rightPower = -power;

}

else return;

leftDrive.setPower(leftPower);

rightDrive.setPower(rightPower);

if (degrees < 0)

{

while (opModeIsActive() && getAngle() == 0) {}

while (opModeIsActive() && getAngle() > degrees) {}

}

else

while (opModeIsActive() && getAngle() < degrees) {}

rightDrive.setPower(0);

leftDrive.setPower(0);

sleep(500);

resetAngle();

leftDrive.setMode(DcMotor.RunMode.RESET\_ENCODERS);

rightDrive.setMode(DcMotor.RunMode.RESET\_ENCODERS);

while(leftDrive.getCurrentPosition() != 0 || rightDrive.getCurrentPosition() != 0) { //Ensures encoders are zero

leftDrive.setMode(DcMotor.RunMode.RESET\_ENCODERS);

rightDrive.setMode(DcMotor.RunMode.RESET\_ENCODERS);

sleep(50);

}

leftDrive.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

rightDrive.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

deltaTicksLeft=0;

deltaTicksRight=0;

}

private double checkDirection()

{

double correction, angle, gain = .10;

angle = getAngle();

if (angle == 0)

correction = 0;

else

correction = -angle;

correction = correction \* gain;

return correction;

}

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.

}

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);

}

private void runForward(double distance){

int ticks = (int)(distance/0.280357);

deltaTicksLeft+=ticks;

deltaTicksRight+=ticks;

leftDrive.setTargetPosition(deltaTicksLeft);

rightDrive.setTargetPosition(deltaTicksRight);

leftDrive.setPower(FORWARD\_SPEED);

rightDrive.setPower(FORWARD\_SPEED);

while((leftDrive.isBusy()||rightDrive.isBusy()) && opModeIsActive()) {

}

leftDrive.setPower(0);

rightDrive.setPower(0);

}

private void runBackward(double distance){

int ticks = (int)(distance/0.280357);

deltaTicksLeft -= ticks;

deltaTicksRight -= ticks;

leftDrive.setTargetPosition(deltaTicksLeft);

rightDrive.setTargetPosition(deltaTicksRight);

leftDrive.setPower(-FORWARD\_SPEED);

rightDrive.setPower(-FORWARD\_SPEED);

while((leftDrive.isBusy()||rightDrive.isBusy()) && opModeIsActive()) {

}

leftDrive.setPower(0);

rightDrive.setPower(0);

}

private void drive\_until\_cm(double distance){

leftDrive.setMode(DcMotor.RunMode.RUN\_USING\_ENCODERS);

rightDrive.setMode(DcMotor.RunMode.RUN\_USING\_ENCODERS);

leftDrive.setPower(0.8);

rightDrive.setPower(0.8);

while(distSensor.getDistance(DistanceUnit.CM)>distance && opModeIsActive()){

if(distSensor.getDistance(DistanceUnit.CM)-distance<60){

leftDrive.setPower(0.35);

rightDrive.setPower(0.35);

}

}

leftDrive.setPower(0);

rightDrive.setPower(0);

leftDrive.setMode(DcMotor.RunMode.RESET\_ENCODERS);

rightDrive.setMode(DcMotor.RunMode.RESET\_ENCODERS);

leftDrive.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

rightDrive.setMode(DcMotor.RunMode.RUN\_TO\_POSITION);

deltaTicksLeft=0;

deltaTicksRight=0;

}

}