```
1  //@Disabled
2  /* Copyright (c) 2019 FIRST. All rights reserved.
3   *
4   * Redistribution and use in source and binary forms, with or
   without modification,
5   * are permitted (subject to the limitations in the disclaimer
    below) provided that
6   * the following conditions are met:
7   *
8   * Redistributions of source code must retain the above
   copyright notice, this list
9   * of conditions and the following disclaimer.
10  *
11  * Redistributions in binary form must reproduce the above
   copyright notice, this
12  * list of conditions and the following disclaimer in the
   documentation and/or
13  * other materials provided with the distribution.
14  *
15  * Neither the name of FIRST nor the names of its contributors
    may be used to endorse or
16  * promote products derived from this software without
   specific prior written permission.
17  *
18  * NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS
    ARE GRANTED BY THIS
19  * LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS
    AND CONTRIBUTORS
20  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
   BUT NOT LIMITED TO,
21  * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
    PARTICULAR PURPOSE
22  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
   CONTRIBUTORS BE LIABLE
23  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
   OR CONSEQUENTIAL
24  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
   SUBSTITUTE GOODS OR
25  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
   INTERRUPTION) HOWEVER
26  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT
   , STRICT LIABILITY,
```

```
27  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
    WAY OUT OF THE USE
28  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
    SUCH DAMAGE.
29  */
30
31 package org.firstinspires.ftc.teamcode;
32
33 import com.qualcomm.hardware.bosch.BNO055IMU;
34 import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
35 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
36 import com.qualcomm.robotcore.hardware.CRServo;
37 import com.qualcomm.robotcore.hardware.DcMotor;
38 import com.qualcomm.robotcore.hardware.DcMotorSimple;
39 import com.qualcomm.robotcore.hardware.Gamepad;
40 import com.qualcomm.robotcore.hardware.HardwareMap;
41 import com.qualcomm.robotcore.hardware.Servo;
42 import com.qualcomm.robotcore.util.ElapsedTime;
43
44 import org.firstinspires.ftc.robotcore.external.ClassFactory;
45 import org.firstinspires.ftc.robotcore.external.hardware.
    camera.WebcamName;
46 import org.firstinspires.ftc.robotcore.external.navigation.
    VuforiaLocalizer;
47 import org.firstinspires.ftc.robotcore.external.tfod.
    Recognition;
48 import org.firstinspires.ftc.robotcore.external.tfod.
    TFObjectDetector;
49
50 import java.util.List;
51
52 /**
53  * This 2019-2020 OpMode illustrates the basics of using the
    TensorFlow Object Detection API to
54  * determine the position of the Skystone game elements.
55  *
56  * Use Android Studio to Copy this Class, and Paste it into
    your team's code folder with a new name.
57  * Remove or comment out the @Disabled line to add this opmode
     to the Driver Station OpMode list.
58  *
59  * IMPORTANT: In order to use this OpMode, you need to obtain
```

```
59 your own Vuforia license key as
60  * is explained below.
61  */
62 @Autonomous(name = "Red Full", group = "Red")
63 public class RedFull extends LinearOpMode {
64     private static final String TFOD_MODEL_ASSET = "Skystone.
   tflite";
65     private static final String LABEL_FIRST_ELEMENT = "Stone"
   ;
66     private static final String LABEL_SECOND_ELEMENT = "
   Skystone";
67
68     /*
69      * IMPORTANT: You need to obtain your own license key to
   use Vuforia. The string below with which
70      * 'parameters.vuforiaLicenseKey' is initialized is for
   illustration only, and will not function.
71      * A Vuforia 'Development' license key, can be obtained
   free of charge from the Vuforia developer
72      * web site at https://developer.vuforia.com/license-
   manager.
73      *
74      * Vuforia license keys are always 380 characters long,
   and look as if they contain mostly
75      * random data. As an example, here is a example of a
   fragment of a valid key:
76      *      ...
   yIgIzTqZ4mWjk9wd3cZO9T1axEqzuhxoGlfOOI2dRzKS4T0hQ8kT ...
77      * Once you've obtained a license key, copy the string
   from the Vuforia web site
78      * and paste it in to your code on the next line, between
    the double quotes.
79      */
80     private static final String VUFORIA_KEY = "AQUWr4X/////
   AAABme38EPssRkvls9+q/
   BGPYgxKXBXELWHMdkTcCqUqHeyDpyXGWFLCTABgDXEMGe1EmsnDQxmJ7WQ069
   J3YSv+kOcfq3g2EnwZr2O3DujsIU1nT0aXgLlAtQU2r7wWAgHvR9ADO5pe/
   q7MzCyhjSTQLCgizGFLgmqfre0A9rjYcXYbYw11R3P7VRHnL3QHn3QH2oFVQf
   Mb+dIzmZkfv0cd5qWvdhjovYF8hpZ/
   HT7veIa8ZQ9CIQ0541pxplXVud80z1xWpjFGJPaoQGO+xKWZ8E+
   Zlu7z5umiaV1+
   ChGeJ9pPyIJn0LsnoIHumZoYb4di4tFygMPVmH8ChsTlGJjaPBSCRBFjxzBqs
```

```
80 XmBZY7eCa6S";
81
82     /**
83      * {@link #vuforia} is the variable we will use to store
   our instance of the Vuforia
84      * localization engine.
85      */
86     private VuforiaLocalizer vuforia;
87
88     /**
89      * {@link #tfod} is the variable we will use to store our
    instance of the TensorFlow Object
90      * Detection engine.
91      */
92     private TFObjectDetector tfod;
93
94     //Wheel Motors
95     DcMotor leftFront;
96     DcMotor rightFront;
97     DcMotor leftBack;
98     DcMotor rightBack;
99
100    //Foundation Servos
101    Servo leftFoundation;
102    Servo rightFoundation;
103
104    //Turret Motor
105    DcMotor turret;
106
107    //Lift Motor
108    DcMotor lift;
109
110    //Arm Motor
111    DcMotor arm;
112
113    //Claw Continuous Rotation Servo
114    CRServo leftClaw;
115    CRServo rightClaw;
116
117    //Gyroscope
118    BNO055IMU imu;
119
```

```
120     //Robot Classes
121     DriveTrain dT;
122     Armstrong a;
123
124     //State Machine Class
125     RedStates rS;
126
127     //Timer
128     ElapsedTime runtime;
129
130     //Stone counter
131     int count = 1;
132     //Extra distance for each stone
133     int extraTick;
134
135     //Boolean for if the skystone has been found
136     boolean skystone = false;
137
138     //The label of the object the program sees
139     String label;
140
141     @Override
142     public void runOpMode() throws InterruptedException
143     {
144         // The TFObjectDetector uses the camera frames from
    the VuforiaLocalizer, so we create that
145         // first.
146         initVuforia();
147
148         if (ClassFactory.getInstance().canC
    reateTFObjectDetector()) {
149             initTfod();
150         } else {
151             telemetry.addData("Sorry!", "This device is not
    compatible with TFOD");
152         }
153
154         /**
155          * Activate TensorFlow Object Detection before we wai
    t for the start command.
156          * Do it here so that the Camera Stream window will
    have the TensorFlow annotations visible.
```

```
157          **/
158         if (tfod != null) {
159             tfod.activate();
160         }
161
162         //Setting variables to the real life components using
     the configuration on the phone.
163         leftFront = hardwareMap.dcMotor.get("leftFront");
164         rightFront = hardwareMap.dcMotor.get("rightFront");
165         leftBack = hardwareMap.dcMotor.get("leftBack");
166         rightBack = hardwareMap.dcMotor.get("rightBack");
167
168         leftFoundation = hardwareMap.servo.get("lef
    tFoundation");
169         rightFoundation = hardwareMap.servo.get("rig
    htFoundation");
170
171         turret = hardwareMap.dcMotor.get("turret");
172
173         lift = hardwareMap.dcMotor.get("lift");
174
175         arm = hardwareMap.dcMotor.get("arm");
176
177         leftClaw = hardwareMap.crservo.get("leftClaw");
178         rightClaw = hardwareMap.crservo.get("rightClaw");
179
180         imu = hardwareMap.get(BNO055IMU.class, "imu");
181
182         //Setting up the classes to run using the variables
    above
183         dT = new DriveTrain(gamepad1, gamepad2 ,leftFront, ri
    ghtFront, leftBack, rightBack, leftFoundation, righ
    tFoundation);
184         a = new Armstrong(gamepad1, gamepad2,  turret, lift,
    arm, leftClaw, rightClaw, imu);
185         runtime = new ElapsedTime();
186
187         //Set the motors to stop and stay still instead of
    removing all power and coasting
188         leftFront.setZeroPowerBehavior(DcMotor.Zero
    PowerBehavior.BRAKE);
189         leftFront.setDirection(DcMotorSimple.Direction.REVE
```

```
189 RSE);
190
191         rightFront.setZeroPowerBehavior(DcMotor.Zero
    PowerBehavior.BRAKE);
192
193         leftBack.setZeroPowerBehavior(DcMotor.Zero
    PowerBehavior.BRAKE);
194         leftBack.setDirection(DcMotorSimple.Direction.REVERSE
    );
195
196         rightBack.setZeroPowerBehavior(DcMotor.Zero
    PowerBehavior.BRAKE);
197
198         turret.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior
    .BRAKE);
199
200         lift.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.
    BRAKE);
201
202         arm.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.
    BRAKE);
203
204         //Stops Robot
205         dT.kill();
206         dT.release();
207         a.kill();
208
209         //Setting up Red States
210         rS = new RedStates(leftFront, rightFront, leftBack,
    rightBack, leftFoundation, rightFoundation, turret, lift, arm
    , leftClaw, rightClaw, dT, a, runtime);
211
212         /** Wait for the game to begin */
213         telemetry.addData(">", "Press Play to start op mode
    ");
214         telemetry.update();
215
216         waitForStart();
217
218         //Resets Timer
219         runtime.reset();
220
```

```
221         //If the play button is pressed
222         if (opModeIsActive())
223         {
224
225             //While the state machine is still running
226             while (rS.getState() != "finished")
227             {
228
229                 //Run the state machine method
230                 rS.runFull();
231
232                 //Telemetry that displays the current state
233                 telemetry.addData("State>", rS.getState());
234
235                 //Display Telemetry
236                 telemetry.update();
237
238             }
239
240             //Loops the program
241             while (opModeIsActive())
242             {
243
244                 if (tfod != null)
245                 {
246
247                     // getUpdatedRecognitions() will return
    null if no new information is available since
248                     // the last time that call was made.
249
250                     List<Recognition> updatedRecognitions =
    tfod.getUpdatedRecognitions();
251
252                     if (updatedRecognitions != null)
253                     {
254
255                       telemetry.addData("# Object Detected",
    updatedRecognitions.size());
256
257                       // step through the list of
    recognitions and display boundary info.
258
```

```
259                         int i = 0;
260
261                         for (Recognition recognition :
    updatedRecognitions)
262                         {
263
264                           telemetry.addData(String.format("labe
    l (%d)", i), recognition.getLabel());
265                           //Sets the label to the name of the
    recognition
266                           label = recognition.getLabel();
267
268                           telemetry.addData(String.format("  le
    ft,top (%d)", i), "%.03f , %.03f",
269                                   recognition.getLeft(), r
    ecognition.getTop());
270
271                           telemetry.addData(String.format("  ri
    ght,bottom (%d)", i), "%.03f , %.03f",
272                                   recognition.getRight(),
    recognition.getBottom());
273
274                           telemetry.update();
275                         }
276
277                         //If label equals Skystone or if two
    stones have been detected
278                         if(label == "Skystone" || count == 3)
279                         {
280
281                             //Telemetry for detecting the
    Skystone
282                             telemetry.addData("Guess What", "
    Skystone Baby");
283
284                             //Displays telemetry
285                             telemetry.update();
286
287                             //Sets the Skystone flag to true
288                             skystone = true;
289
290                             //Stops robot
```

```
291                             dT.kill();
292
293                             //Exits while loop
294                             break;
295
296                         //If the label detects Stone and the timer has been going for one second
297                         } else if (label == "Stone" && runtime.milliseconds() >= 1000)
298                         {
299
300                             //Telemetry for not seeing a Skystone
301                             telemetry.addData("Guess What", "Nothing!");
302
303                             //Displays Telemetry
304                             telemetry.update();
305
306                             //Adds one to the count number
307                             count++;
308
309                             //Adds to the extra tick distance
310                             extraTick = extraTick + 380;
311
312                             //Moves left to next stone
313                             dT.left(0.25, 380);
314
315                             //Resets timer
316                             runtime.reset();
317
318                         //If the robot times out
319                         } else if (runtime.milliseconds() >= 1500)
320                         {
321
322                             //Rotate arm up to create movement to help detection
323                             a.rUp(0.25, 50);
324
325                             //Resets timer
326                             runtime.reset();
```

```
327
328                         }
329
330                         //Displays Telemetry
331                         telemetry.update();
332
333                     }
334                 }
335             }
336         }
337
338         //If the Skystone has been detected
339         if (skystone == true)
340         {
341
342             //Moves robot forward to the skystone
343             dT.forward(0.25, 350);
344
345             //Grabs Skystone
346             leftClaw.setPower(-1);
347             rightClaw.setPower(1);
348
349             //Continuous clamp on the skystone
350             a.setCIdle(1);
351
352             //Resets Timer
353             runtime.reset();
354
355             //Waits a quarter second to clamp
356             while (runtime.milliseconds() < 250);
357
358             //Rotates arm up to a level position
359             a.rUp(0.5, 300);
360
361             //Sets idle power
362             arm.setPower(0.15);
363
364             //Moves robot back to clear the bridge structure
365             dT.backwards(0.3, 350);
366
367             //Moves robot right to the foundation plate
368             dT.right(0.25, 3120 + extraTick);
```

```
369
370             //Moves forward to the foundation
371             dT.forward(0.3, 250);
372
373             //Releases the skystone
374             a.unclamp(1, 250);
375
376             //Backs away from the foundation plate
377             dT.backwards(0.25, 75);
378
379             //Turns 180 degrees
380             dT.tRight(0.3, 1750);
381
382             //Backs into the foundation plate
383             dT.backwards(0.3, 275);
384
385             //Grabs foundation plate
386             dT.grab();
387
388             //Rotates arm out of the way
389             a.tRight(1, 950);
390
391             //Pulls foundation plate to building zone
392             dT.forward(0.5, 1400);
393
394             //Lets go of the foundation plate
395             dT.release();
396
397             //Backwards to avoid wall
398             dT.backwards(0.25, 50);
399
400             //Moves robot right
401             dT.right(0.5, 800);
402
403             //Moves robot behind alliance member
404             dT.backwards(0.3, 950);
405
406             //Moves right to park
407             dT.right(0.3, 800);
408
409         }
410
```

```
411         if (tfod != null) {
412             tfod.shutdown();
413         }
414     }
415
416     /**
417      * Initialize the Vuforia localization engine.
418      */
419     private void initVuforia() {
420         /*
421          * Configure Vuforia by creating a Parameter object,
    and passing it to the Vuforia engine.
422          */
423         VuforiaLocalizer.Parameters parameters = new Vuf
    oriaLocalizer.Parameters();
424
425         parameters.vuforiaLicenseKey = VUFORIA_KEY;
426         parameters.cameraName = hardwareMap.get(WebcamNa
    me.class, "Webcam 1");
427
428         //  Instantiate the Vuforia engine
429         vuforia = ClassFactory.getInstance().createVuforia(
    parameters);
430
431         // Loading trackables is not necessary for the Te
    nsorFlow Object Detection engine.
432     }
433
434     /**
435      * Initialize the TensorFlow Object Detection engine.
436      */
437     private void initTfod() {
438         int tfodMonitorViewId = hardwareMap.appContext.
    getResources().getIdentifier(
439             "tfodMonitorViewId", "id", hardwareMap.appContext
    .getPackageName());
440         TFObjectDetector.Parameters tfodParameters = new TFO
    bjectDetector.Parameters(tfodMonitorViewId);
441         tfodParameters.minimumConfidence = 0.8;
442         tfod = ClassFactory.getInstance().createTFO
    bjectDetector(tfodParameters, vuforia);
443         tfod.loadModelFromAsset(TFOD_MODEL_ASSET, LABEL_F
```

```
443 IRST_ELEMENT, LABEL_SECOND_ELEMENT);
444     }
445 }
```