```
 1
 2 package org.usfirst.frc.team4409.robot;
 3
 4 import edu.wpi.first.wpilibj.IterativeRobot;
31
32 /**
33  * The VM is configured to automatically run this class, and to call the
34  * functions corresponding to each mode, as described in the IterativeRobot
35  * documentation. If you change the name of this class or the package after
36  * creating this project, you must also update the manifest file in the resource
37  * directory.
38  */
39 public class Robot extends IterativeRobot {
40     final String defaultAuto = "Default";
41     final String lineAuto = "Line";
42     final String leftPegAuto = "Left Peg Auto";
43     final String midPegAuto = "Middle Peg Auto";
44     final String rightPegAuto = "Right Peg Auto";
45     final double wheelradius = 2; // this is the value of the wheel radius for using encoders
46
47     final double EncoderTo360 = 0.17578125;
48     boolean autoComplete = false;
49     String autoSelected;
50     SendableChooser<String> chooser;
51     RobotDrive myDrive;
52     Spark FrontRight, BackLeft;
53     TalonSRX BackRight, FrontLeft;
54     Talon Winch1, Winch2, Flywheel;
55     Victor Spinner;
56     Joystick Stick, Gamepad;
57     Timer timer;
58     DoubleSolenoid Funnel, Hatch;
59     Compressor Pump;
60     DigitalInput Rangefinder;
61     Encoder LeftEnc;
62     Thread visionThread;
63     ADXRS450_Gyro Gyro;
64     Boolean flywheelToggle;
65     Boolean flywheelDebounce;
66     Boolean winchToggle;
67     Boolean winchDebounce;
68     Boolean db180;
69     Double currentAngle;
70     Double targetAngle;
71     Boolean dbMoveZ;
72     Boolean spinDebounce;
73     Boolean spinToggle;
74     Boolean dbMoveZ2;
75     Boolean tankDrive;
76     Boolean hatchTog;
77     Boolean hatchdb;
78     Boolean turnTog;
79     Boolean turnTogdb;
80     Double lastTime;
81     Double angleToTurn;
82
83     /**
```

```
84      * This function is run when the robot is first started up and should be
85      * used for any initialization code.
86      */
87     @Override
88     public void robotInit() {
89         chooser = new SendableChooser<String>();
90         chooser.addDefault("Default Auto", defaultAuto);
91         chooser.addObject("Line Auto", lineAuto);
92         chooser.addObject("Left Peg Auto", leftPegAuto);
93         chooser.addObject("Middle Peg Auto", midPegAuto);
94         chooser.addObject("Right Peg Auto", rightPegAuto);
95         SmartDashboard.putData("Options", chooser);
96         SmartDashboard.putNumber("Gyroscope Angle", 0);
97         SmartDashboard.putNumber("Turn Angle", 0.00);
98         FrontRight = new Spark(2);
99         BackLeft = new Spark(1);
100        BackRight = new TalonSRX(4);
101        FrontLeft = new TalonSRX(3);
102        myDrive = new RobotDrive(FrontLeft, BackLeft, FrontRight, BackRight);
103        Winch1 = new Talon(5);
104        Winch2 = new Talon(6);
105        Flywheel = new Talon(7);
106        Spinner = new Victor(8);
107        Gamepad = new Joystick(0);
108        Stick = new Joystick(1);
109        timer = new Timer();
110        Pump = new Compressor(0);
111        Pump.setClosedLoopControl(true);
112        Funnel = new DoubleSolenoid(0, 1);
113        Hatch = new DoubleSolenoid(2, 3);
114        LeftEnc = new Encoder(2, 3);
115        Rangefinder = new DigitalInput(1);
116        Gyro = new ADXRS450_Gyro(SPI.Port.kOnboardCS0);
117        flywheelToggle = false;
118        flywheelDebounce = true;
119        winchToggle = false;
120        winchDebounce = true;
121        db180 = true;
122        dbMoveZ = true;
123        spinToggle = false;
124        spinDebounce = true;
125        dbMoveZ2 = true;
126        tankDrive = false;
127
128        hatchTog = false;
129        hatchdb = true;
130        turnTog = true;
131        turnTogdb = true;
132        lastTime = 0.00;
133        angleToTurn = 0.00;
134        Gyro.calibrate();
135        visionThread = new Thread(() -> {
136            // Get the Axis camera from CameraServer
137            // origanal Ip address of camera one is: 10.44.9.82
138            AxisCamera camera = CameraServer.getInstance().addAxisCamera("10.44.9.11");
139            // AxisCamera camera2 = CameraServer.getInstance().addAxisCamera("
140            // 10.44.9.11");
```

```
141             // Set the resolution
142             camera.setResolution(640, 480);
143             // camera2.setResolution(640, 480);
144 
145             // Get a CvSink. This will capture Mats from the camera
146             CvSink cvSink = CameraServer.getInstance().getVideo();
147             // Setup a CvSource. This will send images back to the Dashboard
148             CvSource outputStream = CameraServer.getInstance().putVideo("Rectangle", 640,
    480);
149 
150             // Mats are very memory expensive. Lets reuse this Mat.
151             Mat mat = new Mat();
152 
153             // This cannot be 'true'. The program will never exit if it is. This
154             // lets the robot stop this thread when restarting robot code or
155             // deploying.
156             while (!Thread.interrupted()) {
157                 // Tell the CvSink to grab a frame from the camera and put it
158                 // in the source mat. If there is an error notify the output.
159                 if (cvSink.grabFrame(mat) == 0) {
160                     // Send the output the error.
161                     outputStream.notifyError(cvSink.getError());
162                     // skip the rest of the current iteration
163                     continue;
164                 }
165                 // Put a rectangle on the image
166                 // Imgproc.rectangle(mat, new Point(100, 100), new Point(400,
167                 // 400), new Scalar(255, 255, 255), 5);
168                 // Give the output stream a new image to display
169                 outputStream.putFrame(mat);
170             }
171         });
172         visionThread.setDaemon(true);
173         visionThread.start();
174     }
175 
176     /**
177      * This autonomous (along with the chooser code above) shows how to select
178      * between different autonomous modes using the dashboard. The sendable
179      * chooser code works with the Java SmartDashboard. If you prefer the
180      * LabVIEW Dashboard, remove all of the chooser code and uncomment the
181      * getString line to get the auto name from the text box below the Gyro
182      *
183      * You can add additional auto modes by adding additional comparisons to the
184      * switch structure below with additional strings. If using the
185      * SendableChooser make sure to add them to the chooser code above as well.
186      */
187     @Override
188     public void autonomousInit() {
189         autoSelected = chooser.getSelected();
190         DriverStation.reportWarning("Auto selected: " + autoSelected, true);
191         autoComplete = true;
192     }
193 
194     /**
195      * This function is called periodically during autonomous
196      */
```

```
197     public void updateDash() {
198         SmartDashboard.putNumber("Gyroscope Angle", Math.floor(Gyro.getAngle() * 100) / 100);
199         tankDrive = SmartDashboard.getBoolean("Tank Drive", false);
200         SmartDashboard.putNumber("Left Encoder", LeftEnc.get());
201         SmartDashboard.putNumber("Timer", Math.floor(timer.get() * 100) / 100);
202         angleToTurn = SmartDashboard.getNumber("Turn Angle", 0.00);
203         // .autoComplete./autoComplete =
204         // SmartDashboard.getBoolean("AutoCompleteStatus", false);
205 
206     }
207 
208     public boolean turnAngle(Double angle) {
209         Gyro.reset();
210         angle *= 0.5;
211         if (angle >= 0) {
212             while (Gyro.getAngle() < angle) {
213                 if (Gamepad.getRawButton(2)) {
214                     break;
215                 }
216                 myDrive.tankDrive(-0.75, 0.75);
217             }
218             myDrive.tankDrive(0.1, -0.1);
219         } else {
220             while (Gyro.getAngle() > angle) {
221                 if (Gamepad.getRawButton(2)) {
222                     break;
223                 }
224                 myDrive.tankDrive(0.75, -0.75);
225             }
226             myDrive.tankDrive(-0.1, 0.1);
227         }
228         return true;
229     }
230 
231     public boolean moveZ(long dur, Boolean forward) {
232         // System.out.println("run");
233         Timer time = new Timer();
234         time.start();
235         while (time.get() < dur) {
236             if (forward) {
237                 myDrive.tankDrive(0.75, 0.75);
238             } else {
239                 myDrive.tankDrive(-0.75, -0.75);
240             }
241         }
242         myDrive.arcadeDrive(0, 0);
243         return true;
244     }
245 
246     public void driveForward(double seconds) {
247         timer.reset();
248         myDrive.tankDrive(0, 0);
249         while (timer.get() < seconds) {
250             DriverStation.reportWarning(Double.toString(timer.get()), true);
251             myDrive.tankDrive(-0.75, -0.75);
252         }
253         myDrive.tankDrive(0, 0);
```

```
254     }
255
256     public void driveBackwards(double seconds) {
257         timer.reset();
258         myDrive.tankDrive(0, 0);
259         while (timer.get() < seconds) {
260             DriverStation.reportWarning(Double.toString(timer.get()), true);
261             myDrive.tankDrive(0.75, 0.75);
262         }
263         myDrive.tankDrive(0, 0);
264     }
265
266     @Override
267     public void autonomousPeriodic() {
268         switch (autoSelected) {
269         case lineAuto:
270             updateDash();
271             if (timer.get() >= 0 && timer.get() < 1) {
272                 myDrive.tankDrive(-0.5, -0.5);
273             } else if (timer.get() >= 7 && timer.get() < 8) {
274                 myDrive.tankDrive(0, 0);
275             } else if (timer.get() >= 15) {
276                 break;
277             }
278         case leftPegAuto:
279             updateDash();
280
281             break;
282         case rightPegAuto:
283             updateDash();
284
285             break;
286         case midPegAuto:
287             updateDash();
288
289             break;
290         case defaultAuto:
291         default:
292             updateDash();
293             EncoderDrive(93.25, false);
294             break;
295         }
296     }
297
298     /**
299      * This function is called periodically during operator control
300      */
301     @Override
302     public void teleopPeriodic() {
303         timer.stop();
304         updateDash();
305
306         // debuging things(we really should make a dashboard)to
307         if (Gamepad.getRawButton(11) && turnTogdb) {
308             turnTogdb = false;
309             turnTog = turnTog == true ? false : true;
310         }
```

```
311         if (Gamepad.getRawButton(11) == false) {
312             turnTogdb = true;
313         }
314         // DriverStation.reportWarning(Double.toString(LeftEnc.get() *
315         // 0.17578125),true);//that fancy number worked last year
316         if (Gamepad.getRawButton(3) == true && db180 == true && turnTog) {
317             db180 = false;
318             db180 = turnAngle(angleToTurn);
319         }
320         if (Gamepad.getRawButton(5) == true && db180 == true && turnTog) {
321             db180 = false;
322             db180 = turnAngle(90.00);
323         }
324         if (Gamepad.getRawButton(4) == true && db180 == true && turnTog) {
325             db180 = false;
326             db180 = turnAngle(-90.00);
327         }
328         if (Gamepad.getRawButton(1) == true) {
329             if (tankDrive) {
330                 myDrive.tankDrive(Gamepad.getRawAxis(1) * 0.8, Stick.getRawAxis(1) * 0.8);
331             } else {
332                 myDrive.arcadeDrive(Gamepad.getRawAxis(1) * 0.8, Gamepad.getRawAxis(0) * 0.8);
333             }
334         } else {
335             if (tankDrive) {
336                 myDrive.tankDrive(Gamepad.getRawAxis(1), Stick.getRawAxis(1));
337             } else {
338                 myDrive.arcadeDrive(Gamepad.getRawAxis(1), Gamepad.getRawAxis(0));
339             }
340         }
341         // DriverStation.reportWarning(Double.toString(Gyro.getAngle()),true);
342 
343         // only drive winch forward
344         if (Stick.getRawButton(2) == true && winchDebounce == true) {
345             winchToggle = winchToggle == true ? false : true;
346             winchDebounce = false;
347         }
348         if (winchToggle == true) {
349             Winch1.set((-Stick.getRawAxis(2) + 3) / 4);
350             Winch2.set((-Stick.getRawAxis(2) + 3) / 4);
351         } else {
352             Winch1.set(0);
353             Winch2.set(0);
354         }
355         if (Stick.getRawButton(2) == false) {
356             winchDebounce = true;
357         }
358         if (Stick.getRawButton(5) == true && spinDebounce == true) {
359             spinToggle = spinToggle == true ? false : true;
360             spinDebounce = false;
361         }
362         if (spinToggle == true) {
363             Spinner.set(1.0);
364         } else {
365             Spinner.set(0.0);
366         }
367         if (Stick.getRawButton(5) == false) {
```

```
368             spinDebounce = true;
369         }
370 
371         /*
372          * if (Stick.getRawAxis(1) <= 0) { Winch1.set(Stick.getRawAxis(1));
373          * Winch2.set(Stick.getRawAxis(1)); } else { Winch1.set(0.0);
374          * Winch2.set(0.0); }
375          */
376         if (Stick.getRawButton(1) == true && hatchdb == true) {
377             hatchTog = hatchTog == true ? false : true;
378             hatchdb = false;
379         }
380         if (hatchTog == true) {
381             Hatch.set(DoubleSolenoid.Value.kReverse);
382         } else {
383             Hatch.set(DoubleSolenoid.Value.kForward);
384         }
385         if (Stick.getRawButton(1) == false) {
386             hatchdb = true;
387         }
388         /*
389          * if (Stick.getRawButton(1) == true) {
390          * Hatch.set(DoubleSolenoid.Value.kReverse); } else {
391          * Hatch.set(DoubleSolenoid.Value.kForward); }
392          */
393         if (Stick.getRawButton(3) == true) {
394             Funnel.set(DoubleSolenoid.Value.kReverse);
395         } else {
396             Funnel.set(DoubleSolenoid.Value.kForward);
397         }
398         if (Stick.getRawButton(4) == true && flywheelDebounce == true) {
399             flywheelToggle = flywheelToggle == true ? false : true;
400             flywheelDebounce = false;
401         }
402         if (flywheelToggle == true) {
403             Flywheel.set((-Stick.getRawAxis(2) + 3) / 4);
404 
405         } else {
406             Flywheel.set(0);
407         }
408         if (Stick.getRawButton(4) == false) {
409             flywheelDebounce = true;
410         }
411 
412         // SmartDashboard.putNumber("Ultrasonic",Rangefinder.getRangeInches());
413     }
414 
415 
416     public void EncoderDrive(double Distance, boolean Foward) {
417         if (autoComplete == false) {
418 
419             LeftEnc.reset();
420 
421             // figure out the distance to drive
422             double OneRotation = Math.PI * 2 * wheelradius;
423             double NumberRotations = Distance / OneRotation;
424 
```

```
425             SmartDashboard.putNumber("rotationstarget", NumberRotations);
426             while ((LeftEnc.get() * EncoderTo360) < NumberRotations) {
427                 myDrive.tankDrive(-0.5, -0.52);
428                 updateDash();
429             }
430             myDrive.tankDrive(0.0, 0.0);
431             autoComplete = true;
432 
433         }
434     }
435 
436 }
437 
```