```java
// Copyright (c) FIRST and other WPILib contributors.
// Open Source Software; you can modify and/or share it under the terms of
// the WPILib BSD license file in the root directory of this project.

package frc.robot;

import edu.wpi.first.wpilibj.RobotBase;

/**
 * Do NOT add any static variables to this class, or any initialization at all. Unless you know what
 * you are doing, do not modify this file except to change the parameter class to the startRobot
 * call.
 */
public final class Main {
  private Main() {}

  /**
   * Main initialization function. Do not perform any initialization here.
   *
   * <p>If you change your main robot class, change the parameter type.
   */
  public static void main(String... args) {
    RobotBase.startRobot(Robot::new);
  }
}

```

```java
// Copyright (c) FIRST and other WPILib contributors.
// Open Source Software; you can modify and/or share it under the terms of
// the WPILib BSD license file in the root directory of this project.

package frc.robot;

import edu.wpi.first.wpilibj.DriverStation;
import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.CommandScheduler;

/**
 * The VM is configured to automatically run this class, and to call the functions corresponding to
 * each mode, as described in the TimedRobot documentation. If you change the name of this class or
 * the package after creating this project, you must also update the build.gradle file in the
 * project.
 */
public class Robot extends TimedRobot {
  private Command m_autonomousCommand;

  private RobotContainer m_robotContainer;

  private Boolean hasSetColor = false;

  /**
   * This function is run when the robot is first started up and should be used for any
   * initialization code.
   */
  @Override
  public void robotInit() {
    // Instantiate our RobotContainer.  This will perform all our button bindings, and put our
    // autonomous chooser on the dashboard.
    m_robotContainer = new RobotContainer();
  }

  /**
   * This function is called every 20 ms, no matter the mode. Use this for items like diagnostics
   * that you want ran during disabled, autonomous, teleoperated and test.
   *
   * <p>This runs after the mode specific periodic functions, but before LiveWindow and
   * SmartDashboard integrated updating.
   */
  @Override
```

```
44   public void robotPeriodic() {
45     // Runs the Scheduler.  This is responsible for polling buttons, adding newly-
   scheduled
46     // commands, running already-scheduled commands, removing finished or interrupted
    commands,
47     // and running subsystem periodic() methods.  This must be called from the robot'
   s periodic
48     // block in order for anything in the Command-based framework to work.
49     CommandScheduler.getInstance().run();
50   }
51
52   /** This function is called once each time the robot enters Disabled mode. */
53   @Override
54   public void disabledInit() {}
55
56   @Override
57   public void disabledPeriodic() {
58     if (DriverStation.isDSAttached() && !hasSetColor) {
59       if (DriverStation.getAlliance().equals(DriverStation.Alliance.Red)) {
60         m_robotContainer.leds.setColorRGB(250, 0, 0);
61       }
62       else {
63         m_robotContainer.leds.setColorRGB(0, 0, 250);
64       }
65       hasSetColor = true;
66     }
67   }
68
69   /** This autonomous runs the autonomous command selected by your {@link
   RobotContainer} class. */
70   @Override
71   public void autonomousInit() {
72     m_autonomousCommand = m_robotContainer.getAutonomousCommand();
73
74     // schedule the autonomous command (example)
75     if (m_autonomousCommand != null) {
76       m_autonomousCommand.schedule();
77     }
78   }
79
80   /** This function is called periodically during autonomous. */
81   @Override
82   public void autonomousPeriodic() {}
83
84   @Override
85   public void teleopInit() {
86     // This makes sure that the autonomous stops running when
87     // teleop starts running. If you want the autonomous to
88     // continue until interrupted by another command, remove
89     // this line or comment it out.
```

```
 90     if (m_autonomousCommand != null) {
 91       m_autonomousCommand.cancel();
 92     }
 93   }
 94
 95   /** This function is called periodically during operator control. */
 96   @Override
 97   public void teleopPeriodic() {}
 98
 99   @Override
100   public void testInit() {
101     // Cancels all running commands at the start of test mode.
102     CommandScheduler.getInstance().cancelAll();
103   }
104
105   /** This function is called periodically during test mode. */
106   @Override
107   public void testPeriodic() {}
108
109   /** This function is called once when the robot is first started up. */
110   @Override
111   public void simulationInit() {}
112
113   /** This function is called periodically whilst in simulation. */
114   @Override
115   public void simulationPeriodic() {}
116 }
117
```

```java
// Copyright (c) FIRST and other WPILib contributors.
// Open Source Software; you can modify and/or share it under the terms of
// the WPILib BSD license file in the root directory of this project.

package frc.robot;

import edu.wpi.first.math.geometry.Rotation3d;
import edu.wpi.first.math.geometry.Transform3d;
import edu.wpi.first.math.geometry.Translation3d;
import edu.wpi.first.math.util.Units;
import swervelib.math.Matter;

/**
 * The Constants class provides a convenient place for teams to hold robot-wide numerical or boolean
 * constants. This class should not be used for any other purpose. All constants should be declared
 * globally (i.e. public static). Do not put anything functional in this class.
 *
 * <p>It is advised to statically import this class (or one of its inner classes) wherever the
 * constants are needed, to reduce verbosity.
 */
public final class Constants {

  public static class driveConstants {
    public static final double balanceP = 0.05;
    public static final double balanceI = 0;
    public static final double balanceD = 0;

    public static final double ROBOT_MASS = 45.35924; // 32lbs * kg per pound

    // a matter var for limiting velocity
    public static final Matter CHASSIS = new Matter(new Translation3d(0, 0, Units.inchesToMeters(4)), ROBOT_MASS);

    // loop time to use
    public static final double LOOP_TIME = 0.13;
  }

  public static class cuberConstants {
    public static final int angleMotorPort = 9;
    public static final int leftShooterPort = 10;
    public static final int rightShooterPort = 11;

    public static final double angleP = 0.001;
    public static final double angleI = 0;
    public static final double angleD = 0;

    public static final double shooterP = 0.001;
```

```
47      public static final double shooterI = 0;
48      public static final double shooterD = 0;
49    }
50
51    public static class visionConstants {
52      // the height above the aim point for the top of the arc to be
53      public static final double maxHeight = 9.5;
54
55      // this enum contains 3 different levels to indicate what shelf we are aiming for
56      // as well as an offset for shooting for each shelf
57      public enum heights {
58        low (1) {
59          @Override
60          public double getHeightDiff() {
61            return 1;
62          }
63          },
64        mid (19){
65          @Override
66          public double getHeightDiff() {
67            return 19;
68          }
69          },
70        high (29){
71          @Override
72          public double getHeightDiff() {
73            return 29;
74          }
75        };
76
77        private final double heightDiff;
78
79        public double getHeightDiff() {
80          return heightDiff;
81        }
82
83        heights (double heightDiff) {
84          this.heightDiff = heightDiff;
85        }
86      }
87
88      // the type of camera used
89      public enum cameraType {
90        photonVision,
91        LimeLight
92      }
93
94      // gravity to use
95      public static final double g = 32;
96
```

```
 97     // distances from the center of the robot to the camera
 98     public static final Transform3d robotToFrontCam =
 99             new Transform3d(
100                     new Translation3d(0.222, 0.238, 0),
101                     new Rotation3d(
102                             60, 0,
103                             0));
104
105     public static final Transform3d robotToBackCam =
106             new Transform3d(
107                     new Translation3d(-0.222, 0.238, 0),
108                     new Rotation3d(
109                             -60, 0,
110                             0));
111
112     public static final Transform3d robotToLimelight =
113             new Transform3d(
114                     new Translation3d(-0.222, 0.238, 0),
115                     new Rotation3d(
116                             60, 0,
117                             0));
118   }
119
120   public static class LEDConstants {
121     public static final int PWMPort = 0;
122
123     // number of LEDs
124     public static final int length = 60;
125   }
126 }
127
```

```java
// Copyright (c) FIRST and other WPILib contributors.
// Open Source Software; you can modify and/or share it under the terms of
// the WPILib BSD license file in the root directory of this project.

package frc.robot;

import com.pathplanner.lib.PathConstraints;
import com.pathplanner.lib.auto.PIDConstants;
import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj2.command.Commands;
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.button.CommandXboxController;
import frc.robot.commands.balance;
import frc.robot.commands.drive;
import frc.robot.commands.fieldCentricDrive;
import frc.robot.subsystems.*;
import frc.robot.Constants.visionConstants.cameraType;
import frc.robot.vision.visionWrapper;

import java.util.HashMap;

/**
 * This class is where the bulk of the robot should be declared. Since Command-based
 * is a "declarative" paradigm, very little robot logic should actually be handled in
 * the {@link Robot} periodic methods (other than the scheduler calls). Instead, the
 * structure of the robot (including subsystems, commands, and trigger mappings)
 * should be declared here.
 */
public class RobotContainer {

  // let me enable and disable certain subsystems for testing
  public final Boolean shooterEnabled = false;
  public final Boolean angleControllerEnabled = false;
  public final Boolean driveEnabled = true;
  public final Boolean LEDsEnabled = false;


  // create the subsystem vars
  public Drive drive;
  public Shooter shooter;
  public AngleController angleController;
  public LEDs leds;


  // create the controllers
  public final CommandXboxController driveController = new CommandXboxController(0);
  public final CommandXboxController coDriveController = new CommandXboxController(1
);

```

```
50
51   // create the autonomous selector
52   private final SendableChooser<Command> chooser = new SendableChooser<>();
53
54
55   /** The container for the robot. Contains subsystems, OI devices, and commands. */
56   public RobotContainer() {
57     //  define the three cameras
58     visionWrapper frontCamera = new visionWrapper(
59             "frontCamera",
60             Constants.visionConstants.robotToFrontCam, cameraType.photonVision);
61     visionWrapper backCamera = new visionWrapper(
62             "backCamera",
63             Constants.visionConstants.robotToBackCam, cameraType.photonVision);
64     visionWrapper gpCamera = new visionWrapper("limelight",
65             Constants.visionConstants.robotToLimelight, cameraType.LimeLight);
66
67     // define and set the default command of the subsystems if they are enabled
68     if (driveEnabled) {
69       drive = new Drive(frontCamera, backCamera);
70       drive.setDefaultCommand(new drive(drive, driveController::getLeftX,
71               driveController::getLeftY, driveController::getRightX,
72               () -> SmartDashboard.getBoolean("is field oriented", true),false, true
   ));
73     }
74     if (shooterEnabled) {
75       shooter = new Shooter(frontCamera, backCamera);
76       shooter.setDefaultCommand(shooter.setShooterWithSpeed(-0.02));
77     }
78     if (angleControllerEnabled) {
79       angleController = new AngleController(frontCamera, backCamera);
80       angleController.setDefaultCommand(angleController.runWithJoysticks(
   coDriveController::getLeftX));
81     }
82     if (LEDsEnabled) {
83       leds = new LEDs();
84       leds.setDefaultCommand(leds.runOnce(leds::showCubeCounter));
85     }
86
87     // commented out till I need camera streams on the dashboard
88 //    CameraServer.startAutomaticCapture();
89
90
91     // configure the autonomous routines
92     configureAutonomous();
93
94     // Configure the button bindings
95     configureBindings();
96   }
97
```

```
 98   /**
 99    * configure the autonomous routines
100    */
101   private void configureAutonomous() {
102     // this will only run if the required subsystems are enabled
103     if (driveEnabled && shooterEnabled && angleControllerEnabled) {
104       // create a map of all events that can happen in auto
105       HashMap<String, Command> eventMap = new HashMap<>();
106       eventMap.put("shootMid", shooter.runShooterWithVision(Constants.
    visionConstants.heights.mid));
107       eventMap.put("shootHigh", shooter.runShooterWithVision(Constants.
    visionConstants.heights.high));
108       eventMap.put("shootLow", shooter.runShooterWithVision(Constants.
    visionConstants.heights.low));
109       eventMap.put("collect",
110               angleController.turnToAngle(0).
111                       andThen(shooter.runShooterSpeedForTime(-0.5, 1).
112                               andThen(angleController.turnToAngle(120))));
113       eventMap.put("eject", shooter.runShooterSpeedForTime(0.5, 0.5));
114       eventMap.put("balance", new balance(drive));
115
116       // create the auto builder
117       drive.defineAutoBuilder(
118               eventMap,
119               new PIDConstants(5.0, 0.0, 0.0),
120               new PIDConstants(0.5, 0.0, 0.0), true);
121
122       // create constraints for velocity and acceleration
123       PathConstraints constraints = new PathConstraints(6, 4);
124
125       // create the auto commands
126       // autos for the left side of the community
127       Command threePieceBalanceLeft = drive.createTrajectory(
128               "3 piece balance clean",
129               constraints);
130
131       Command threePieceLeft = drive.createTrajectory(
132               "3 piece clean",
133               constraints);
134
135       Command fourPieceLeft = drive.createTrajectory(
136               "4 piece clean", constraints);
137
138       Command fivePieceLeft = drive.createTrajectory(
139               "5 piece clean", constraints);
140
141       // create autos for the right side of the community
142
143       Command threePieceBalanceRight = drive.createTrajectory(
144               "3 piece balance bump",
```

```
145                 constraints);
146
147         Command threePieceRight = drive.createTrajectory(
148                 "3 piece bump",
149                 constraints);
150
151         Command fourPieceRight = drive.createTrajectory(
152                 "4 piece bump",
153                 constraints);
154
155         Command fivePieceRight = drive.createTrajectory(
156                 "5 piece bump",
157                 constraints);
158
159         // create autos for the middle of the community
160
161         Command leftTwoPieceCharge = drive.createTrajectory(
162                 "left 2 piece charge",
163                 constraints);
164
165         Command rightTwoPieceCharge = drive.createTrajectory(
166                 "right 2 piece charge",
167                 constraints);
168
169         Command onePieceCharge = drive.createTrajectory(
170                 "1 piece charge",
171                 constraints);
172
173         Command threePieceCharge = drive.createTrajectory(
174                 "3 piece balance mid",
175                 constraints);
176
177         // add them to the auto chooser
178         // left side autos
179         chooser.setDefaultOption("3 piece balance clean", threePieceBalanceLeft);
180         chooser.addOption("3 piece clean", threePieceLeft);
181         chooser.addOption("4 piece clean", fourPieceLeft);
182         chooser.addOption("5 piece clean", fivePieceLeft);
183         // right side autos
184         chooser.addOption("3 piece balance bump", threePieceBalanceRight);
185         chooser.addOption("3 piece bump", threePieceRight);
186         chooser.addOption("4 piece bump", fourPieceRight);
187         chooser.addOption("5 piece bump", fivePieceRight);
188         // middle autos
189         chooser.addOption("3 piece balance middle", threePieceCharge);
190         chooser.addOption("left 2 piece balance middle", leftTwoPieceCharge);
191         chooser.addOption("right 2 piece balance middle", rightTwoPieceCharge);
192         chooser.addOption("1 piece balance middle", onePieceCharge);
193     }
194     // add a do nothing auto
```

```
195     chooser.addOption("do nothing", Commands.none());
196
197     // put the chooser to the dashboard
198     SmartDashboard.putData("autos", chooser);
199   }
200
201   /**
202    * configures the button bindings
203    */
204   private void configureBindings() {
205     // all ifs are checks to see if the required subsystems are enabled
206     // basic angles (collection angle and a stowed angle)
207     if (angleControllerEnabled) {
208       coDriveController.a().onTrue(Commands.runOnce(() -> angleController.
    setAngleSetpoint(0.0)));
209       coDriveController.b().onTrue(Commands.runOnce(() -> angleController.
    setAngleSetpoint(120)));
210     }
211
212     // dynamic angles. These are to aim for high and mid from any distance within
    the community
213     if (driveEnabled && angleControllerEnabled && shooterEnabled && LEDsEnabled) {
214       // sequence locking the drivetrain, setting the LED color, get and set the
    angle, shoot,
215       // update the LED color, and up the number of cubes shot
216       coDriveController.x().onTrue(Commands.sequence(
217               Commands.runOnce(drive::lock),
218               leds.setColorRGBCommand(255, 204, 0),
219               angleController.turnToAngleVision(Constants.visionConstants.heights.
    mid),
220               shooter.runShooterWithVision(Constants.visionConstants.heights.mid),
221               leds.showColorTime(51, 204, 51, 2),
222               leds.incrementCubeCounter()));
223       coDriveController.y().onTrue(Commands.sequence(
224               Commands.runOnce(drive::lock),
225               leds.setColorRGBCommand(255, 204, 0),
226               angleController.turnToAngleVision(Constants.visionConstants.heights.
    high),
227               shooter.runShooterWithVision(Constants.visionConstants.heights.high),
228               leds.showColorTime(51, 204, 51, 2),
229               leds.incrementCubeCounter()));
230     }
231     // allow the button bindings to be created without the LEDs
232     else if (driveEnabled && angleControllerEnabled && shooterEnabled) {
233       coDriveController.x().onTrue(Commands.sequence(
234               Commands.runOnce(drive::lock),
235               angleController.turnToAngleVision(Constants.visionConstants.heights.
    mid),
236               shooter.runShooterWithVision(Constants.visionConstants.heights.mid)));
237       coDriveController.y().onTrue(Commands.sequence(
```

```
238                 Commands.runOnce(drive::lock),
239                 angleController.turnToAngleVision(Constants.visionConstants.heights.
    high),
240                 shooter.runShooterWithVision(Constants.visionConstants.heights.high
    )));
241       }
242
243       // collect and shoot. These are for running the wheels to collect or shoot
244       if (shooterEnabled && LEDsEnabled) {
245         coDriveController.leftBumper().onTrue(
246                 shooter.collect(-0.3)
247                         .andThen(() -> leds.setColorRGBCommand(0, 230, 0)));
248         coDriveController.rightBumper().whileTrue(
249                 shooter.setShooterWithSpeed(0.3).andThen(leds::incrementCubeCounter));
250       }
251       // allow the button bindings to be created without the LEDs
252       else if (shooterEnabled) {
253         coDriveController.leftBumper().onTrue(
254                 shooter.collect(-0.3));
255         coDriveController.rightBumper().whileTrue(
256                 shooter.setShooterWithSpeed(0.3));
257       }
258     }
259
260     /**
261      * @return the command to run in autonomous
262      */
263     public Command getAutonomousCommand() {
264       // return the command selected by the auto chooser
265       return chooser.getSelected();
266     }
267 }
268
```

```java
//LimelightHelpers v1.2.1 (March 1, 2023)

package frc.robot;

import edu.wpi.first.networktables.NetworkTable;
import edu.wpi.first.networktables.NetworkTableEntry;
import edu.wpi.first.networktables.NetworkTableInstance;
import edu.wpi.first.math.geometry.Pose2d;
import edu.wpi.first.math.geometry.Pose3d;
import edu.wpi.first.math.geometry.Rotation2d;
import edu.wpi.first.math.geometry.Translation3d;
import edu.wpi.first.math.util.Units;
import edu.wpi.first.math.geometry.Rotation3d;
import edu.wpi.first.math.geometry.Translation2d;

import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.CompletableFuture;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonFormat.Shape;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;


// this file is a library for the limelight. I did not make it

public class LimelightHelpers {

    public static class LimelightTarget_Retro {

        @JsonProperty("t6c_ts")
        private double[] cameraPose_TargetSpace;

        @JsonProperty("t6r_fs")
        private double[] robotPose_FieldSpace;

        @JsonProperty("t6r_ts")
        private  double[] robotPose_TargetSpace;

        @JsonProperty("t6t_cs")
        private double[] targetPose_CameraSpace;

        @JsonProperty("t6t_rs")
        private double[] targetPose_RobotSpace;

```

```
 51         public Pose3d getCameraPose_TargetSpace()
 52         {
 53             return toPose3D(cameraPose_TargetSpace);
 54         }
 55         public Pose3d getRobotPose_FieldSpace()
 56         {
 57             return toPose3D(robotPose_FieldSpace);
 58         }
 59         public Pose3d getRobotPose_TargetSpace()
 60         {
 61             return toPose3D(robotPose_TargetSpace);
 62         }
 63         public Pose3d getTargetPose_CameraSpace()
 64         {
 65             return toPose3D(targetPose_CameraSpace);
 66         }
 67         public Pose3d getTargetPose_RobotSpace()
 68         {
 69             return toPose3D(targetPose_RobotSpace);
 70         }
 71
 72         public Pose2d getCameraPose_TargetSpace2D()
 73         {
 74             return toPose2D(cameraPose_TargetSpace);
 75         }
 76         public Pose2d getRobotPose_FieldSpace2D()
 77         {
 78             return toPose2D(robotPose_FieldSpace);
 79         }
 80         public Pose2d getRobotPose_TargetSpace2D()
 81         {
 82             return toPose2D(robotPose_TargetSpace);
 83         }
 84         public Pose2d getTargetPose_CameraSpace2D()
 85         {
 86             return toPose2D(targetPose_CameraSpace);
 87         }
 88         public Pose2d getTargetPose_RobotSpace2D()
 89         {
 90             return toPose2D(targetPose_RobotSpace);
 91         }
 92
 93         @JsonProperty("ta")
 94         public double ta;
 95
 96         @JsonProperty("tx")
 97         public double tx;
 98
 99         @JsonProperty("txp")
100         public double tx_pixels;
```

```
101
102         @JsonProperty("ty")
103         public double ty;
104
105         @JsonProperty("typ")
106         public double ty_pixels;
107
108         @JsonProperty("ts")
109         public double ts;
110
111         public LimelightTarget_Retro() {
112             cameraPose_TargetSpace = new double[6];
113             robotPose_FieldSpace = new double[6];
114             robotPose_TargetSpace = new double[6];
115             targetPose_CameraSpace = new double[6];
116             targetPose_RobotSpace = new double[6];
117         }
118
119     }
120
121     public static class LimelightTarget_Fiducial {
122
123         @JsonProperty("fID")
124         public double fiducialID;
125
126         @JsonProperty("fam")
127         public String fiducialFamily;
128
129         @JsonProperty("t6c_ts")
130         private double[] cameraPose_TargetSpace;
131
132         @JsonProperty("t6r_fs")
133         private double[] robotPose_FieldSpace;
134
135         @JsonProperty("t6r_ts")
136         private double[] robotPose_TargetSpace;
137
138         @JsonProperty("t6t_cs")
139         private double[] targetPose_CameraSpace;
140
141         @JsonProperty("t6t_rs")
142         private double[] targetPose_RobotSpace;
143
144         public Pose3d getCameraPose_TargetSpace()
145         {
146             return toPose3D(cameraPose_TargetSpace);
147         }
148         public Pose3d getRobotPose_FieldSpace()
149         {
150             return toPose3D(robotPose_FieldSpace);
```

```
151         }
152         public Pose3d getRobotPose_TargetSpace()
153         {
154             return toPose3D(robotPose_TargetSpace);
155         }
156         public Pose3d getTargetPose_CameraSpace()
157         {
158             return toPose3D(targetPose_CameraSpace);
159         }
160         public Pose3d getTargetPose_RobotSpace()
161         {
162             return toPose3D(targetPose_RobotSpace);
163         }
164 
165         public Pose2d getCameraPose_TargetSpace2D()
166         {
167             return toPose2D(cameraPose_TargetSpace);
168         }
169         public Pose2d getRobotPose_FieldSpace2D()
170         {
171             return toPose2D(robotPose_FieldSpace);
172         }
173         public Pose2d getRobotPose_TargetSpace2D()
174         {
175             return toPose2D(robotPose_TargetSpace);
176         }
177         public Pose2d getTargetPose_CameraSpace2D()
178         {
179             return toPose2D(targetPose_CameraSpace);
180         }
181         public Pose2d getTargetPose_RobotSpace2D()
182         {
183             return toPose2D(targetPose_RobotSpace);
184         }
185 
186         @JsonProperty("ta")
187         public double ta;
188 
189         @JsonProperty("tx")
190         public double tx;
191 
192         @JsonProperty("txp")
193         public double tx_pixels;
194 
195         @JsonProperty("ty")
196         public double ty;
197 
198         @JsonProperty("typ")
199         public double ty_pixels;
200 
```

```
201         @JsonProperty("ts")
202         public double ts;
203 
204         public LimelightTarget_Fiducial() {
205             cameraPose_TargetSpace = new double[6];
206             robotPose_FieldSpace = new double[6];
207             robotPose_TargetSpace = new double[6];
208             targetPose_CameraSpace = new double[6];
209             targetPose_RobotSpace = new double[6];
210         }
211     }
212 
213     public static class LimelightTarget_Barcode {
214 
215     }
216 
217     public static class LimelightTarget_Classifier {
218 
219         @JsonProperty("class")
220         public String className;
221 
222         @JsonProperty("classID")
223         public double classID;
224 
225         @JsonProperty("conf")
226         public double confidence;
227 
228         @JsonProperty("zone")
229         public double zone;
230 
231         @JsonProperty("tx")
232         public double tx;
233 
234         @JsonProperty("txp")
235         public double tx_pixels;
236 
237         @JsonProperty("ty")
238         public double ty;
239 
240         @JsonProperty("typ")
241         public double ty_pixels;
242 
243         public  LimelightTarget_Classifier() {
244         }
245     }
246 
247     public static class LimelightTarget_Detector {
248 
249         @JsonProperty("class")
250         public String className;
```

```
251
252         @JsonProperty("classID")
253         public double classID;
254
255         @JsonProperty("conf")
256         public double confidence;
257
258         @JsonProperty("ta")
259         public double ta;
260
261         @JsonProperty("tx")
262         public double tx;
263
264         @JsonProperty("txp")
265         public double tx_pixels;
266
267         @JsonProperty("ty")
268         public double ty;
269
270         @JsonProperty("typ")
271         public double ty_pixels;
272
273         public LimelightTarget_Detector() {
274         }
275     }
276
277     public static class Results {
278
279         @JsonProperty("pID")
280         public double pipelineID;
281
282         @JsonProperty("tl")
283         public double latency_pipeline;
284
285         @JsonProperty("cl")
286         public double latency_capture;
287
288         public double latency_jsonParse;
289
290         @JsonProperty("ts")
291         public double timestamp_LIMELIGHT_publish;
292
293         @JsonProperty("ts_rio")
294         public double timestamp_RIOFPGA_capture;
295
296         @JsonProperty("v")
297         @JsonFormat(shape = Shape.NUMBER)
298         public boolean valid;
299
300         @JsonProperty("botpose")
```

```
301         public double[] botpose;
302
303         @JsonProperty("botpose_wpired")
304         public double[] botpose_wpired;
305
306         @JsonProperty("botpose_wpiblue")
307         public double[] botpose_wpiblue;
308
309         @JsonProperty("t6c_rs")
310         public double[] camerapose_robotspace;
311
312         public Pose3d getBotPose3d() {
313             return toPose3D(botpose);
314         }
315
316         public Pose3d getBotPose3d_wpiRed() {
317             return toPose3D(botpose_wpired);
318         }
319
320         public Pose3d getBotPose3d_wpiBlue() {
321             return toPose3D(botpose_wpiblue);
322         }
323
324         public Pose2d getBotPose2d() {
325             return toPose2D(botpose);
326         }
327
328         public Pose2d getBotPose2d_wpiRed() {
329             return toPose2D(botpose_wpired);
330         }
331
332         public Pose2d getBotPose2d_wpiBlue() {
333             return toPose2D(botpose_wpiblue);
334         }
335
336         @JsonProperty("Retro")
337         public LimelightTarget_Retro[] targets_Retro;
338
339         @JsonProperty("Fiducial")
340         public LimelightTarget_Fiducial[] targets_Fiducials;
341
342         @JsonProperty("Classifier")
343         public LimelightTarget_Classifier[] targets_Classifier;
344
345         @JsonProperty("Detector")
346         public LimelightTarget_Detector[] targets_Detector;
347
348         @JsonProperty("Barcode")
349         public LimelightTarget_Barcode[] targets_Barcode;
350
```

```
351         public Results() {
352             botpose = new double[6];
353             botpose_wpired = new double[6];
354             botpose_wpiblue = new double[6];
355             camerapose_robotspace = new double[6];
356             targets_Retro = new LimelightTarget_Retro[0];
357             targets_Fiducials = new LimelightTarget_Fiducial[0];
358             targets_Classifier = new LimelightTarget_Classifier[0];
359             targets_Detector = new LimelightTarget_Detector[0];
360             targets_Barcode = new LimelightTarget_Barcode[0];
361 
362         }
363     }
364 
365     public static class LimelightResults {
366         @JsonProperty("Results")
367         public Results targetingResults;
368 
369         public LimelightResults() {
370             targetingResults = new Results();
371         }
372     }
373 
374     private static ObjectMapper mapper;
375 
376     /**
377      * Print JSON Parse time to the console in milliseconds
378      */
379     static boolean profileJSON = false;
380 
381     static final String sanitizeName(String name) {
382         if (name == "" || name == null) {
383             return "limelight";
384         }
385         return name;
386     }
387 
388     private static Pose3d toPose3D(double[] inData){
389         if(inData.length < 6)
390         {
391             System.err.println("Bad LL 3D Pose Data!");
392             return new Pose3d();
393         }
394         return new Pose3d(
395             new Translation3d(inData[0], inData[1], inData[2]),
396             new Rotation3d(Units.degreesToRadians(inData[3]), Units.degreesToRadians
    (inData[4]),
397                     Units.degreesToRadians(inData[5])));
398     }
399 
```

```
400     private static Pose2d toPose2D(double[] inData){
401         if(inData.length < 6)
402         {
403             System.err.println("Bad LL 2D Pose Data!");
404             return new Pose2d();
405         }
406         Translation2d tran2d = new Translation2d(inData[0], inData[1]);
407         Rotation2d r2d = new Rotation2d(Units.degreesToRadians(inData[5]));
408         return new Pose2d(tran2d, r2d);
409     }
410 
411     public static NetworkTable getLimelightNTTable(String tableName) {
412         return NetworkTableInstance.getDefault().getTable(sanitizeName(tableName));
413     }
414 
415     public static NetworkTableEntry getLimelightNTTableEntry(String tableName,
    String entryName) {
416         return getLimelightNTTable(tableName).getEntry(entryName);
417     }
418 
419     public static double getLimelightNTDouble(String tableName, String entryName) {
420         return getLimelightNTTableEntry(tableName, entryName).getDouble(0.0);
421     }
422 
423     public static void setLimelightNTDouble(String tableName, String entryName,
    double val) {
424         getLimelightNTTableEntry(tableName, entryName).setDouble(val);
425     }
426 
427     public static void setLimelightNTDoubleArray(String tableName, String entryName
    , double[] val) {
428         getLimelightNTTableEntry(tableName, entryName).setDoubleArray(val);
429     }
430 
431     public static double[] getLimelightNTDoubleArray(String tableName, String
    entryName) {
432         return getLimelightNTTableEntry(tableName, entryName).getDoubleArray(new
    double[0]);
433     }
434 
435     public static String getLimelightNTString(String tableName, String entryName) {
436         return getLimelightNTTableEntry(tableName, entryName).getString("");
437     }
438 
439     public static URL getLimelightURLString(String tableName, String request) {
440         String urlString = "http://" + sanitizeName(tableName) + ".local:5807/" +
    request;
441         URL url;
442         try {
443             url = new URL(urlString);
```

```
444             return url;
445         } catch (MalformedURLException e) {
446             System.err.println("bad LL URL");
447         }
448         return null;
449     }
450     /////
451     /////
452
453     public static double getTX(String limelightName) {
454         return getLimelightNTDouble(limelightName, "tx");
455     }
456
457     public static double getTY(String limelightName) {
458         return getLimelightNTDouble(limelightName, "ty");
459     }
460
461     public static double getTA(String limelightName) {
462         return getLimelightNTDouble(limelightName, "ta");
463     }
464
465     public static double getLatency_Pipeline(String limelightName) {
466         return getLimelightNTDouble(limelightName, "tl");
467     }
468
469     public static double getLatency_Capture(String limelightName) {
470         return getLimelightNTDouble(limelightName, "cl");
471     }
472
473     public static double getCurrentPipelineIndex(String limelightName) {
474         return getLimelightNTDouble(limelightName, "getpipe");
475     }
476
477     public static String getJSONDump(String limelightName) {
478         return getLimelightNTString(limelightName, "json");
479     }
480
481     /**
482      * Switch to getBotPose
483      *
484      * @param limelightName
485      * @return
486      */
487     @Deprecated
488     public static double[] getBotpose(String limelightName) {
489         return getLimelightNTDoubleArray(limelightName, "botpose");
490     }
491
492     /**
493      * Switch to getBotPose_wpiRed
```

```
494      *
495      * @param limelightName
496      * @return
497      */
498     @Deprecated
499     public static double[] getBotpose_wpiRed(String limelightName) {
500         return getLimelightNTDoubleArray(limelightName, "botpose_wpired");
501     }
502
503     /**
504      * Switch to getBotPose_wpiBlue
505      *
506      * @param limelightName
507      * @return
508      */
509     @Deprecated
510     public static double[] getBotpose_wpiBlue(String limelightName) {
511         return getLimelightNTDoubleArray(limelightName, "botpose_wpiblue");
512     }
513
514     public static double[] getBotPose(String limelightName) {
515         return getLimelightNTDoubleArray(limelightName, "botpose");
516     }
517
518     public static double[] getBotPose_wpiRed(String limelightName) {
519         return getLimelightNTDoubleArray(limelightName, "botpose_wpired");
520     }
521
522     public static double[] getBotPose_wpiBlue(String limelightName) {
523         return getLimelightNTDoubleArray(limelightName, "botpose_wpiblue");
524     }
525
526     public static double[] getBotPose_TargetSpace(String limelightName) {
527         return getLimelightNTDoubleArray(limelightName, "botpose_targetspace");
528     }
529
530     public static double[] getCameraPose_TargetSpace(String limelightName) {
531         return getLimelightNTDoubleArray(limelightName, "camerapose_targetspace");
532     }
533
534     public static double[] getTargetPose_CameraSpace(String limelightName) {
535         return getLimelightNTDoubleArray(limelightName, "targetpose_cameraspace");
536     }
537
538     public static double[] getTargetPose_RobotSpace(String limelightName) {
539         return getLimelightNTDoubleArray(limelightName, "targetpose_robotspace");
540     }
541
542     public static double[] getTargetColor(String limelightName) {
543         return getLimelightNTDoubleArray(limelightName, "tc");
```

```
544     }
545
546     public static double getFiducialID(String limelightName) {
547         return getLimelightNTDouble(limelightName, "tid");
548     }
549
550     public static double getNeuralClassID(String limelightName) {
551         return getLimelightNTDouble(limelightName, "tclass");
552     }
553
554     /////
555     /////
556
557     public static Pose3d getBotPose3d(String limelightName) {
558         double[] poseArray = getLimelightNTDoubleArray(limelightName, "botpose");
559         return toPose3D(poseArray);
560     }
561
562     public static Pose3d getBotPose3d_wpiRed(String limelightName) {
563         double[] poseArray = getLimelightNTDoubleArray(limelightName, "
    botpose_wpired");
564         return toPose3D(poseArray);
565     }
566
567     public static Pose3d getBotPose3d_wpiBlue(String limelightName) {
568         double[] poseArray = getLimelightNTDoubleArray(limelightName, "
    botpose_wpiblue");
569         return toPose3D(poseArray);
570     }
571
572     public static Pose3d getBotPose3d_TargetSpace(String limelightName) {
573         double[] poseArray = getLimelightNTDoubleArray(limelightName, "
    botpose_targetspace");
574         return toPose3D(poseArray);
575     }
576
577     public static Pose3d getCameraPose3d_TargetSpace(String limelightName) {
578         double[] poseArray = getLimelightNTDoubleArray(limelightName, "
    camerapose_targetspace");
579         return toPose3D(poseArray);
580     }
581
582     public static Pose3d getTargetPose3d_CameraSpace(String limelightName) {
583         double[] poseArray = getLimelightNTDoubleArray(limelightName, "
    targetpose_cameraspace");
584         return toPose3D(poseArray);
585     }
586
587     public static Pose3d getTargetPose3d_RobotSpace(String limelightName) {
588         double[] poseArray = getLimelightNTDoubleArray(limelightName, "
```

```
588 targetpose_robotspace");
589         return toPose3D(poseArray);
590     }
591 
592     public static Pose3d getCameraPose3d_RobotSpace(String limelightName) {
593         double[] poseArray = getLimelightNTDoubleArray(limelightName, "
    camerapose_robotspace");
594         return toPose3D(poseArray);
595     }
596 
597     /**
598      * Gets the Pose2d for easy use with Odometry vision pose estimator
599      * (addVisionMeasurement)
600      *
601      * @param limelightName
602      * @return
603      */
604     public static Pose2d getBotPose2d_wpiBlue(String limelightName) {
605 
606         double[] result = getBotPose_wpiBlue(limelightName);
607         return toPose2D(result);
608     }
609 
610     /**
611      * Gets the Pose2d for easy use with Odometry vision pose estimator
612      * (addVisionMeasurement)
613      *
614      * @param limelightName
615      * @return
616      */
617     public static Pose2d getBotPose2d_wpiRed(String limelightName) {
618 
619         double[] result = getBotPose_wpiRed(limelightName);
620         return toPose2D(result);
621 
622     }
623 
624     /**
625      * Gets the Pose2d for easy use with Odometry vision pose estimator
626      * (addVisionMeasurement)
627      *
628      * @param limelightName
629      * @return
630      */
631     public static Pose2d getBotPose2d(String limelightName) {
632 
633         double[] result = getBotPose(limelightName);
634         return toPose2D(result);
635 
636     }
```

```java

    public static boolean getTV(String limelightName) {
        return 1.0 == getLimelightNTDouble(limelightName, "tv");
    }

    /////
    /////

    public static void setPipelineIndex(String limelightName, int pipelineIndex) {
        setLimelightNTDouble(limelightName, "pipeline", pipelineIndex);
    }

    /**
     * The LEDs will be controlled by Limelight pipeline settings, and not by robot
     * code.
     */
    public static void setLEDMode_PipelineControl(String limelightName) {
        setLimelightNTDouble(limelightName, "ledMode", 0);
    }

    public static void setLEDMode_ForceOff(String limelightName) {
        setLimelightNTDouble(limelightName, "ledMode", 1);
    }

    public static void setLEDMode_ForceBlink(String limelightName) {
        setLimelightNTDouble(limelightName, "ledMode", 2);
    }

    public static void setLEDMode_ForceOn(String limelightName) {
        setLimelightNTDouble(limelightName, "ledMode", 3);
    }

    public static void setStreamMode_Standard(String limelightName) {
        setLimelightNTDouble(limelightName, "stream", 0);
    }

    public static void setStreamMode_PiPMain(String limelightName) {
        setLimelightNTDouble(limelightName, "stream", 1);
    }

    public static void setStreamMode_PiPSecondary(String limelightName) {
        setLimelightNTDouble(limelightName, "stream", 2);
    }

    public static void setCameraMode_Processor(String limelightName) {
        setLimelightNTDouble(limelightName, "camMode", 0);
    }
    public static void setCameraMode_Driver(String limelightName) {
        setLimelightNTDouble(limelightName, "camMode", 1);
    }
```

```
687
688
689     /**
690      * Sets the crop window. The crop window in the UI must be completely open for
691      * dynamic cropping to work.
692      */
693     public static void setCropWindow(String limelightName, double cropXMin, double
    cropXMax, double cropYMin, double cropYMax) {
694         double[] entries = new double[4];
695         entries[0] = cropXMin;
696         entries[1] = cropXMax;
697         entries[2] = cropYMin;
698         entries[3] = cropYMax;
699         setLimelightNTDoubleArray(limelightName, "crop", entries);
700     }
701
702     public static void setCameraPose_RobotSpace(String limelightName, double forward
    , double side, double up, double roll, double pitch, double yaw) {
703         double[] entries = new double[6];
704         entries[0] = forward;
705         entries[1] = side;
706         entries[2] = up;
707         entries[3] = roll;
708         entries[4] = pitch;
709         entries[5] = yaw;
710         setLimelightNTDoubleArray(limelightName, "camerapose_robotspace_set",
    entries);
711     }
712
713     /////
714     /////
715
716     public static void setPythonScriptData(String limelightName, double[]
    outgoingPythonData) {
717         setLimelightNTDoubleArray(limelightName, "llrobot", outgoingPythonData);
718     }
719
720     public static double[] getPythonScriptData(String limelightName) {
721         return getLimelightNTDoubleArray(limelightName, "llpython");
722     }
723
724     /////
725     /////
726
727     /**
728      * Asynchronously take snapshot.
729      */
730     public static CompletableFuture<Boolean> takeSnapshot(String tableName, String
    snapshotName) {
731         return CompletableFuture.supplyAsync(() -> {
```

```
732             return SYNCH_TAKESNAPSHOT(tableName, snapshotName);
733         });
734     }
735 
736     private static boolean SYNCH_TAKESNAPSHOT(String tableName, String snapshotName
    ) {
737         URL url = getLimelightURLString(tableName, "capturesnapshot");
738         try {
739             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
740             connection.setRequestMethod("GET");
741             if (snapshotName != null && snapshotName != "") {
742                 connection.setRequestProperty("snapname", snapshotName);
743             }
744 
745             int responseCode = connection.getResponseCode();
746             if (responseCode == 200) {
747                 return true;
748             } else {
749                 System.err.println("Bad LL Request");
750             }
751         } catch (IOException e) {
752             System.err.println(e.getMessage());
753         }
754         return false;
755     }
756 
757     /**
758      * Parses Limelight's JSON results dump into a LimelightResults Object
759      */
760     public static LimelightResults getLatestResults(String limelightName) {
761 
762         long start = System.nanoTime();
763         LimelightHelpers.LimelightResults results = new LimelightHelpers.
    LimelightResults();
764         if (mapper == null) {
765             mapper = new ObjectMapper().configure(DeserializationFeature.
    FAIL_ON_UNKNOWN_PROPERTIES, false);
766         }
767 
768         try {
769             results = mapper.readValue(getJSONDump(limelightName), LimelightResults.
    class);
770         } catch (JsonProcessingException e) {
771             System.err.println("lljson error: " + e.getMessage());
772         }
773 
774         long end = System.nanoTime();
775         double millis = (end - start) * .000001;
776         results.targetingResults.latency_jsonParse = millis;
777         if (profileJSON) {
```

```
778             System.out.printf("lljson: %.2f\r\n", millis);
779         }
780 
781         return results;
782     }
783 }
```

```java
package frc.robot.vision;

import frc.robot.LimelightHelpers;
import org.photonvision.targeting.PhotonPipelineResult;
import frc.robot.Constants.visionConstants.cameraType;
import org.photonvision.targeting.PhotonTrackedTarget;

public class results {

    PhotonPipelineResult photonResult;
    LimelightHelpers.LimelightResults LLResult;
    cameraType type;

    /**
     * a simple class to allow having one variable that can be either a photon result or a limelight result
     * @param results the photon results
     * @param type the type of the camera
     */
    public results(PhotonPipelineResult results, cameraType type) {
        photonResult = results;
        this.type = type;
    }

    /**
     * a simple class to allow having one variable that can be either a photon result or a limelight result
     * @param results the limelight results
     * @param type the type of the camera
     */
    public results(LimelightHelpers.LimelightResults results, cameraType type) {
        LLResult = results;
        this.type = type;
    }

    /**
     * @return whether there are targets
     */
    public boolean hasTargets() {
        if (type.equals(cameraType.photonVision)) {
            return photonResult.hasTargets();
        }
        return LLResult.targetingResults.valid;
    }

    /**
     * @return the best target in view. photon only
     */
    public PhotonTrackedTarget getBestTarget() {
        return photonResult.getBestTarget();
```

```
49     }
50 }
51
```

```java
package frc.robot.vision;

import edu.wpi.first.apriltag.AprilTagFieldLayout;
import edu.wpi.first.apriltag.AprilTagFields;
import edu.wpi.first.math.geometry.Pose2d;
import edu.wpi.first.math.geometry.Pose3d;
import edu.wpi.first.math.geometry.Transform3d;
import edu.wpi.first.wpilibj.DriverStation;
import edu.wpi.first.wpilibj.RobotBase;
import frc.robot.LimelightHelpers;
import org.photonvision.EstimatedRobotPose;
import org.photonvision.PhotonCamera;
import org.photonvision.PhotonPoseEstimator;
import org.photonvision.PhotonPoseEstimator.PoseStrategy;
import frc.robot.Constants.visionConstants.cameraType;
import org.photonvision.targeting.PhotonPipelineResult;
import org.photonvision.targeting.PhotonTrackedTarget;

import java.io.IOException;
import java.util.*;


/**
 * This class creates a light wrapper for the {@link PhotonCamera} and {@link PhotonPoseEstimator}.
 * the only function it has is getEstimatedGlobalPose
 */
public class visionWrapper {

    private PhotonCamera photonCamera;

    private String cameraName;

    private PhotonPoseEstimator poseEstimator;

    private final cameraType type;

    /**
     * constructs a new vision wrapper object using the name of the photonCamera and the position on the robot
     * @param cameraName the name of the photonCamera
     * @param robotToCam a {@link Transform3d} from the center of the robot to the position of the photonCamera
     */
    public visionWrapper(String cameraName, Transform3d robotToCam, cameraType type) {

        this.type = type;

        if (type.equals(cameraType.photonVision)) {
```

```
47              // set up a photon vision camera
48              photonCamera = new PhotonCamera(cameraName);
49
50              // if it is a simulation, disable some error throwing
51              if (RobotBase.isSimulation()) {
52                  PhotonCamera.setVersionCheckEnabled(false);
53              }
54
55              try {
56                  // Attempt to load the AprilTagFieldLayout that will tell us where
   the tags are on the field.
57                  AprilTagFieldLayout fieldLayout = AprilTagFields.k2023ChargedUp.
   loadAprilTagLayoutField();
58                  // Create pose estimator
59                  poseEstimator =
60                          new PhotonPoseEstimator(
61                                  fieldLayout, PhotonPoseEstimator.PoseStrategy.
   MULTI_TAG_PNP, photonCamera, robotToCam);
62                  poseEstimator.setMultiTagFallbackStrategy(PoseStrategy.
   LOWEST_AMBIGUITY);
63              } catch (IOException e) {
64                  // The AprilTagFieldLayout failed to load. We won't be able to
   estimate poses if we don't know
65                  // where the tags are.
66                  DriverStation.reportError("Failed to load AprilTagFieldLayout", e.
   getStackTrace());
67                  poseEstimator = null;
68              }
69          }
70          else {
71              // set up a limelight camera
72              this.cameraName = cameraName;
73              LimelightHelpers.setCameraMode_Processor(cameraName);
74              LimelightHelpers.setLEDMode_ForceOff(cameraName);
75          }
76      }
77
78      /**
79       * get the type of the camera
80       * @return the cameraType of the camera
81       */
82      public cameraType getCameraType() {
83          return type;
84      }
85
86      /**
87       * get the latest results from the camera
88       * @return the latest results
89       */
90      public results getLatestResult() {
```

```
 91         if (type.equals(cameraType.photonVision)) {
 92             return new results(photonCamera.getLatestResult(), type);
 93         }
 94         else {
 95             return new results(LimelightHelpers.getLatestResults(cameraName), type);
 96         }
 97     }
 98 
 99     /**
100      * A function to estimate the global pose of the robot according to the april tags in view of the robot
101      * @param prevEstimatedRobotPose the previous robot pose
102      * @return the estimated pose of the robot according to this photonCamera
103      */
104     public Optional<EstimatedRobotPose> getEstimatedGlobalPose(Pose2d prevEstimatedRobotPose) {
105         if (poseEstimator == null) {
106             // The field layout failed to load, so we cannot estimate poses.
107             return Optional.empty();
108         }
109 
110         poseEstimator.setReferencePose(prevEstimatedRobotPose);
111         return poseEstimator.update();
112     }
113 }
114 
```

```java
package frc.robot.commands;

import edu.wpi.first.math.geometry.Translation2d;
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj2.command.CommandBase;
import frc.robot.subsystems.Drive;
import swervelib.SwerveController;
import edu.wpi.first.math.kinematics.ChassisSpeeds;

import java.util.function.BooleanSupplier;
import java.util.function.DoubleSupplier;

public class drive extends CommandBase {

    private final Drive swerve;
    private final DoubleSupplier vX;
    private final DoubleSupplier vY;
    private final DoubleSupplier omega;
    private final BooleanSupplier driveMode;
    private final boolean isOpenLoop;
    private final SwerveController controller;
    private final Timer timer = new Timer();
    private final boolean headingCorrection;
    private double angle = 0;
    private double lastTime = 0;

    /**
     * constructs a command to drive the robot in either robot-centric or field-centric mode
     * @param swerve the drive subsystem
     * @param vX the x velocity supplier
     * @param vY the y velocity supplier
     * @param omega the rotation supplier
     * @param driveMode whether it should drive in robot-centric or field-centric
     * @param isOpenLoop whether it should drive in open loop mode
     * @param headingCorrection whether it should correct the heading
     */
    public drive(Drive swerve, DoubleSupplier vX, DoubleSupplier vY, DoubleSupplier omega,
                 BooleanSupplier driveMode, boolean isOpenLoop, boolean headingCorrection) {
        this.swerve = swerve;
        this.vX = vX;
        this.vY = vY;
        this.omega = omega;
        this.driveMode = driveMode;
        this.isOpenLoop = isOpenLoop;
        this.controller = swerve.getSwerveController();
        this.headingCorrection = headingCorrection;
        if (headingCorrection) {
```

```
48              timer.start();
49          }
50
51          addRequirements(swerve);
52      }
53
54      @Override
55      public void initialize() {
56          if (headingCorrection) {
57              lastTime = timer.get();
58          }
59      }
60
61      @Override
62      public void execute() {
63          // cube teh inputs for more controllability
64          double xVelocity = Math.pow(vX.getAsDouble(), 3);
65          double yVelocity = Math.pow(vY.getAsDouble(), 3);
66          double angVelocity = Math.pow(omega.getAsDouble(), 3);
67
68          if (headingCorrection) {
69              angle += (angVelocity * (timer.get() - lastTime)) * controller.config.
    maxAngularVelocity;
70
71              ChassisSpeeds correctedChassisSpeeds = controller.getTargetSpeeds(
    xVelocity, yVelocity, angle,
72                      swerve.getHeading().getRadians());
73
74              swerve.drive(
75                      SwerveController.getTranslation2d(correctedChassisSpeeds),
76                      correctedChassisSpeeds.omegaRadiansPerSecond,
77                      driveMode.getAsBoolean(),
78                      isOpenLoop);
79
80              lastTime = timer.get();
81          }
82          else {
83              swerve.drive(new Translation2d(
84                              xVelocity * controller.config.maxSpeed,
85                              yVelocity * controller.config.maxSpeed),
86                      angVelocity * controller.config.maxAngularVelocity,
87                      driveMode.getAsBoolean(), isOpenLoop);
88          }
89      }
90
91      @Override
92      public boolean isFinished() {
93          return false;
94      }
95  }
```

balance.java

```java
package frc.robot.commands;

import edu.wpi.first.math.geometry.Translation2d;
import edu.wpi.first.wpilibj2.command.CommandBase;
import frc.robot.subsystems.Drive;


public class balance extends CommandBase {
    private final Drive drive;

    double gyroReading, speed;
    boolean useRoll = false;

    public balance(Drive swerveSubsystem) {
        this.drive = swerveSubsystem;

        // this command requires the drive subsystem
        addRequirements(this.drive);
    }

    @Override
    public void initialize() {
        // determine if it is using roll or pitch
        if (drive.getRoll().getDegrees() > drive.getPitch().getDegrees()) {
            useRoll = true;
        }
    }

    @Override
    public void execute() {

        // take the input
        if (useRoll) {
            gyroReading = drive.getRoll().getDegrees();
        }
        else {
            gyroReading = drive.getPitch().getDegrees();
        }

        // calculate the speed with the pid controller
        speed = drive.calculate(gyroReading);

        // limit the speed
        if (Math.abs(speed) > 0.6) {
            speed = Math.copySign(0.6, speed);
        }

        // run the drivetrain
        if (useRoll) {
            drive.drive(new Translation2d(speed, 0), 0, false, false);
```

```
51          }
52          else {
53              drive.drive(new Translation2d(0, speed), 0, false, false);
54          }
55      }
56
57      @Override
58      public boolean isFinished() {
59          // the command is never done
60          return false;
61      }
62  }
63
```

```java
package frc.robot.commands;

import edu.wpi.first.math.geometry.Rotation2d;
import edu.wpi.first.math.geometry.Translation2d;
import edu.wpi.first.math.kinematics.ChassisSpeeds;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj2.command.CommandBase;
import frc.robot.Constants;
import frc.robot.subsystems.Drive;
import swervelib.SwerveController;
import swervelib.math.SwerveMath;

import java.util.List;
import java.util.function.BooleanSupplier;
import java.util.function.DoubleSupplier;

public class fieldCentricDrive extends CommandBase {
    private final Drive drive;
    private final DoubleSupplier vX, vY, heading;

    private final BooleanSupplier isSlowMode;

    private final Boolean isOpenLoop;

    /**
     * a command to drive the robot with joysticks in relation to the field
     * @param drive the drive subsystem
     * @param vX the x velocity supplier
     * @param vY the y velocity supplier
     * @param heading the rotation supplier
     * @param isSlowMode the slow mode button supplier
     * @param isOpenLoop weather or not it is open loop controlled
     */
    public fieldCentricDrive(Drive drive, DoubleSupplier vX, DoubleSupplier vY,
                             DoubleSupplier heading, BooleanSupplier isSlowMode,
  boolean isOpenLoop) {
        this.drive = drive;
        this.vX = vX;
        this.vY =vY;
        this.heading = heading;
        this.isSlowMode = isSlowMode;
        this.isOpenLoop = isOpenLoop;

        // this command requires the drive subsystem
        addRequirements(this.drive);
    }

    @Override
    public void execute() {
        // get the target speeds given the inputs and whether the robot is in slow
```

```
49 mode
50         ChassisSpeeds desiredSpeeds = drive.getTargetSpeeds(
51                 vX.getAsDouble()*(1),
52                 vY.getAsDouble()*(-1),
53                 new Rotation2d(heading.getAsDouble() * Math.PI));
54
55         // set the rotation speed to the heading input
56 //        desiredSpeeds.omegaRadiansPerSecond = heading.getAsDouble() * Math.PI;
57
58         // Limit velocity to prevent tippy
59         Translation2d translation = SwerveController.getTranslation2d(desiredSpeeds);
60         translation = SwerveMath.limitVelocity(translation, drive.getFieldVelocity
   (), drive.getPose(),
61                 Constants.driveConstants.LOOP_TIME, Constants.driveConstants.
   ROBOT_MASS,
62                 List.of(Constants.driveConstants.CHASSIS),
63                 drive.getSwerveDriveConfiguration());
64
65         // dashboard debugging values
66         SmartDashboard.putNumber("LimitedTranslation", translation.getX());
67         SmartDashboard.putString("Translation", translation.toString());
68
69         // Make the robot move
70         drive.drive(translation, desiredSpeeds.omegaRadiansPerSecond, true,
   isOpenLoop);
71     }
72 }
73
```

```java
package frc.robot.subsystems;


import edu.wpi.first.wpilibj.AddressableLED;
import edu.wpi.first.wpilibj.AddressableLEDBuffer;
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.CommandBase;
import edu.wpi.first.wpilibj2.command.SubsystemBase;
import frc.robot.Constants.LEDConstants;

/**
 * this class allows you to control the leds on the robot in multiple different ways
 */
public class LEDs extends SubsystemBase {

    private final AddressableLED led;

    private final AddressableLEDBuffer ledBuffer;

    private int cubes = 0;

    /**
     * constructs the leds class
     */
    public LEDs() {
        // create an addressable led object and a buffer for it
        led = new AddressableLED(LEDConstants.PWMPort);
        ledBuffer = new AddressableLEDBuffer(LEDConstants.length);

        // set the length of the strip and give it data
        led.setLength(LEDConstants.length);
        led.setData(ledBuffer);
        led.start();
    }

    /**
     * set the color of the entire strip to a rgb color
     */
    public void setColorRGB(int r, int g, int b) {
        // loop through the buffer and set each led to the desired color
        for (int i = 0; i < ledBuffer.getLength(); i++) {
            ledBuffer.setRGB(i, r, g, b);
        }

        led.setData(ledBuffer);
    }

    /**
     * creates a command to set the entire strip to a rgb color
     * @return the generated command
```

```
 51      */
 52     public Command setColorRGBCommand(int r, int g, int b) {
 53         return this.runOnce(() -> setColorRGB(r, g, b));
 54     }
 55 
 56     /**
 57      * set the color of each led individually
 58      */
 59     public void setIndividualColors(int[] r, int[] g, int[] b) {
 60         for (int i = 0; i < ledBuffer.getLength(); i++) {
 61             ledBuffer.setRGB(i, r[i], g[i], b[i]);
 62         }
 63 
 64         led.setData(ledBuffer);
 65     }
 66 
 67     /**
 68      * creates a command to increase the counter of cubes shot by 1
 69      * @return the generated command
 70      */
 71     public Command incrementCubeCounter() {
 72         return this.runOnce(() -> cubes += 1);
 73     }
 74 
 75     /**
 76      * updates the leds to show the number of cubes shot
 77      */
 78     public void showCubeCounter() {
 79         // loop through all the leds
 80         for (int i = 0; i < ledBuffer.getLength(); i++) {
 81             if (i <= cubes*4) {
 82                 ledBuffer.setRGB(i, 153, 51, 255);
 83             }
 84             else {
 85                 ledBuffer.setRGB(i, 0, 0, 0);
 86             }
 87         }
 88 
 89         led.setData(ledBuffer);
 90     }
 91 
 92     /**
 93      * create a command to show a rgb color for a specified time
 94      * @param time the amount of time
 95      * @return the generated command
 96      */
 97     public CommandBase showColorTime(int r, int g, int b, double time) {
 98         return runOnce(() -> setColorRGB(r, g, b))
 99                 .until(() -> false) // to not end automatically
100                 .withTimeout(time); // to end
```

```
101     }
102 }
103
104
```

```java
package frc.robot.subsystems;

import com.pathplanner.lib.PathConstraints;
import com.pathplanner.lib.PathPlanner;
import com.pathplanner.lib.PathPlannerTrajectory;
import com.pathplanner.lib.auto.PIDConstants;
import edu.wpi.first.math.controller.PIDController;
import edu.wpi.first.math.geometry.Pose2d;
import edu.wpi.first.math.geometry.Rotation2d;
import edu.wpi.first.math.geometry.Translation2d;
import edu.wpi.first.math.kinematics.ChassisSpeeds;
import edu.wpi.first.wpilibj.Filesystem;
import edu.wpi.first.wpilibj.smartdashboard.Field2d;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.SubsystemBase;
import frc.robot.Constants.driveConstants;
import frc.robot.vision.visionWrapper;
import org.photonvision.EstimatedRobotPose;
import swervelib.SwerveController;
import swervelib.SwerveDrive;
import swervelib.math.SwerveKinematics2;
import swervelib.parser.SwerveDriveConfiguration;
import swervelib.parser.SwerveParser;

import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.Map;
import java.util.Optional;

import com.pathplanner.lib.auto.SwerveAutoBuilder;
import swervelib.telemetry.SwerveDriveTelemetry;

/**
 * This class sets up everything needed for the drivetrain.
 * This includes, the drivetrain from the library, the autonomous builder,
 * the pose estimator, and the debugging values for the dashboard
 */
public class Drive extends SubsystemBase {

    // define the directory that contains the config files for the swerve drive
    File swerveJsonDir = new File(Filesystem.getDeployDirectory(),"swerve");
    SwerveDrive drive;

    private SwerveAutoBuilder autoBuilder = null;

    // PID controller for balancing
    private final PIDController balanceController;

```

```
51      private final visionWrapper frontCamera, backCamera;
52
53      // 2d field to put to the dashboard
54      private final Field2d field = new Field2d();
55
56      /**
57       * constructs a new drivetrain.
58       * @param frontCamera the camera facing the front
59       * @param backCamera the camera facing the back
60       */
61      public Drive(visionWrapper frontCamera, visionWrapper backCamera) {
62
63          this.frontCamera = frontCamera;
64          this.backCamera = backCamera;
65
66          // set the balance controller's P, I, D, tolerance, and setpoint
67          balanceController  = new PIDController(
68                  driveConstants.balanceP,
69                  driveConstants.balanceI,
70                  driveConstants.balanceD);
71          balanceController.setTolerance(0.3, 1);
72          balanceController.setSetpoint(0);
73
74          // put the 2d field to the dashboard
75          SmartDashboard.putData("Field", field);
76
77          // set the swerve telemetry's verbosity
78          SwerveDriveTelemetry.verbosity = SwerveDriveTelemetry.TelemetryVerbosity.
    HIGH;
79
80          // create the drivetrain from the config files
81          try {
82              drive = new SwerveParser(swerveJsonDir).createSwerveDrive();
83          } catch (IOException e) {
84              throw new RuntimeException(e);
85          }
86      }
87
88
89      @Override
90      public void periodic() {
91          // update the robot pose
92          updateOdometry();
93
94          // give the 2d field the updated pose
95          field.setRobotPose(drive.swerveDrivePoseEstimator.getEstimatedPosition());
96
97          // update the numerical pose for the dashboard. Don't need it right now
98  //      SmartDashboard.putNumberArray("2d pos", new double[]{
99  //              drive.swerveDrivePoseEstimator.getEstimatedPosition().getX(),
```

```
100 //              drive.swerveDrivePoseEstimator.getEstimatedPosition().getY(),
101 //              drive.swerveDrivePoseEstimator.getEstimatedPosition().getRotation().
    getRadians()});
102     }
103 
104     public void updateOdometry() {
105         // use the encoders to estimate pose.
106         drive.updateOdometry();
107 
108         // get the estimated poses from the cameras.
109         Optional<EstimatedRobotPose> frontResult = frontCamera.
    getEstimatedGlobalPose(drive.getPose());
110         Optional<EstimatedRobotPose> backResult = backCamera.getEstimatedGlobalPose(
    drive.getPose());
111 
112         // if the results exist (if there are april tags in view) give them to the
    pose estimator.
113         if (frontResult.isPresent()) {
114             EstimatedRobotPose camPose = frontResult.get();
115             drive.swerveDrivePoseEstimator.addVisionMeasurement(
116                     camPose.estimatedPose.toPose2d(), camPose.timestampSeconds);
117         }
118         if (backResult.isPresent()) {
119             EstimatedRobotPose camPose = backResult.get();
120             drive.swerveDrivePoseEstimator.addVisionMeasurement(
121                     camPose.estimatedPose.toPose2d(), camPose.timestampSeconds);
122         }
123     }
124 
125 
126     /**
127      * A function to supply the drivetrain with movement commands
128      * @param translation the wanted translation.
129      * @param rotation the wanted rotation.
130      * @param fieldRelative whether the robot is driving relative to the field.
131      * @param isOpenLoop whether to drive in open loop mode
132      */
133     public void drive(Translation2d translation, double rotation, boolean
    fieldRelative, boolean isOpenLoop) {
134         drive.drive(translation, rotation, fieldRelative, isOpenLoop);
135     }
136 
137     /**
138      * a way to get the swerve drive's kinematics
139      * @return the swerve drive's kinematics.
140      */
141     public SwerveKinematics2 getKinematics() {
142         return drive.kinematics;
143     }
144 
```

```java

    /**
     * get the estimated pose of the robot
     * @return the pose of the robot
     */
    public Pose2d getPose() {
        return drive.getPose();
    }


    /**
     * resets the odometry to the given pose.
     * @param initialHolonomicPose the initial pose to reset to
     */
    public void resetOdometry(Pose2d initialHolonomicPose) {
        drive.resetOdometry(initialHolonomicPose);
    }

    /**
     * @return the heading of the robot from the gyro
     */
    public Rotation2d getHeading() {
        return drive.getYaw();
    }

    /**
     * @return the pitch of the robot from the gyro
     */
    public Rotation2d getPitch() {
        return drive.getPitch();
    }

    /**
     * @return the roll of the robot from the gyro
     */
    public Rotation2d getRoll() {
        return drive.getRoll();
    }


    /**
     * calculate the desired velocity of the robot when balancing.
     * @param measurement the current angle of the robot
     * @return the desired velocity
     */
    public double calculate(double measurement) {
        return balanceController.calculate(measurement);
    }

```

```
195     /**
196      * get the target speeds for the robot based off of 4 axis.
197      * @param xInput the x-axis input of the first joystick
198      * @param yInput the y-axis input of the first joystick
199      * @param headingX the x-axis input of the second joystick
200      * @param headingY the y-axis input of the second joystick
201      * @return the target speeds
202      */
203     public ChassisSpeeds getTargetSpeeds(double xInput, double yInput, double
    headingX, double headingY) {
204         // cube the translation inputs for more controllability
205         xInput = Math.pow(xInput, 3);
206         yInput = Math.pow(yInput, 3);
207         return drive.swerveController.getTargetSpeeds(xInput, yInput, headingX,
    headingY, getHeading().getRadians());
208     }
209 
210 
211     /**
212      * get the target speeds for the robot based off of 3 axis
213      * @param xInput the 1st input axis
214      * @param yInput the 2nd input axis
215      * @param angle the 3rd input axis
216      * @return the target speeds
217      */
218     public ChassisSpeeds getTargetSpeeds(double xInput, double yInput, Rotation2d
    angle) {
219         // cube the translation inputs for more controllability
220         xInput = Math.pow(xInput, 3);
221         yInput = Math.pow(yInput, 3);
222         return drive.swerveController.getTargetSpeeds(xInput, yInput, angle.
    getRadians(), getHeading().getRadians());
223     }
224 
225 
226     /**
227      * @return get the field relative velocity
228      */
229     public ChassisSpeeds getFieldVelocity() {
230         return drive.getFieldVelocity();
231     }
232 
233 
234     /**
235      * @return the current swerve drive configuration
236      */
237     public SwerveDriveConfiguration getSwerveDriveConfiguration() {
238         return drive.swerveDriveConfiguration;
239     }
240 
```

```
241     /**
242      * @return get the swerve heading controller
243      */
244     public SwerveController getSwerveController() {
245         return drive.swerveController;
246     }
247 
248 
249     /**
250      * make the swerve drive's wheels go in an x pattern to force the robot to stay
    in position
251      */
252     public void lock() {
253         drive.lockPose();
254     }
255 
256     /**
257      * define the autonomous builder.
258      * @param eventMap the list of all events that occur in any auto
259      * @param translationPID the pid constants that should be used for translation
260      * @param rotationPID the pid constants that should be used for rotation
261      * @param useAllianceColor whether to use the alliance color in creating an auto
262      */
263     public void defineAutoBuilder(Map<String, Command> eventMap,
264                                   PIDConstants translationPID, PIDConstants
    rotationPID, boolean useAllianceColor) {
265         if (autoBuilder == null) {
266             autoBuilder = new SwerveAutoBuilder(
267                     drive::getPose,
268                     drive::resetOdometry,
269                     translationPID,
270                     rotationPID,
271                     drive::setChassisSpeeds,
272                     eventMap,
273                     useAllianceColor,
274                     this);
275         }
276     }
277 
278     /**
279      * create an auto from the given path and constrains
280      * @param path the path of the auto
281      * @param constraints the velocity and acceleration constraints
282      * @return the auto
283      */
284     public Command createTrajectory(String path, PathConstraints constraints) {
285         List<PathPlannerTrajectory> pathGroup = PathPlanner.loadPathGroup(path,
    constraints);
286 
287         return autoBuilder.fullAuto(pathGroup);
```

```
288     }
289 }
290
```

```java
package frc.robot.subsystems;


import com.revrobotics.CANSparkMaxLowLevel.MotorType;
import com.revrobotics.RelativeEncoder;
import com.revrobotics.SparkMaxPIDController;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.Commands;
import edu.wpi.first.wpilibj2.command.SubsystemBase;
import com.revrobotics.CANSparkMax;
import com.revrobotics.CANSparkMax.IdleMode;
import frc.robot.Constants;
import frc.robot.Constants.cuberConstants;
import frc.robot.vision.results;
import frc.robot.vision.visionWrapper;
import org.photonvision.targeting.PhotonPipelineResult;
import org.photonvision.targeting.PhotonTrackedTarget;

/**
 * this class allows you to control the shooter either manually or with vision.
 */
public class Shooter extends SubsystemBase {

    private final CANSparkMax leftShooter;
    private final CANSparkMax rightShooter;

    private final RelativeEncoder leftShooterEncoder;
    private final RelativeEncoder rightShooterEncoder;

    private final SparkMaxPIDController leftShooterController;
    private final SparkMaxPIDController rightShooterController;

    private final visionWrapper frontCamera, backCamera;

    /**
     * constructs a new shooter that has access to the given cameras
     * @param frontCamera the camera in front
     * @param backCamera the camera in back
     */
    public Shooter(visionWrapper frontCamera, visionWrapper backCamera) {
        leftShooter = new CANSparkMax(cuberConstants.leftShooterPort, MotorType.
kBrushless);
        rightShooter = new CANSparkMax(cuberConstants.rightShooterPort, MotorType.
kBrushless);

        // motor configuration
        configureMotors();

        // set the encoders to be the motor's encoders
```

```
49          leftShooterEncoder = leftShooter.getEncoder();
50          rightShooterEncoder = rightShooter.getEncoder();
51
52          // set up the pid controllers
53          leftShooterController = leftShooter.getPIDController();
54          leftShooterController.setP(cuberConstants.shooterP);
55          leftShooterController.setI(cuberConstants.shooterI);
56          leftShooterController.setD(cuberConstants.shooterD);
57          rightShooterController = rightShooter.getPIDController();
58          rightShooterController.setP(cuberConstants.shooterP);
59          rightShooterController.setI(cuberConstants.shooterI);
60          rightShooterController.setD(cuberConstants.shooterD);
61
62          this.frontCamera = frontCamera;
63          this.backCamera = backCamera;
64      }
65
66      /**
67       * configure the motors.
68       */
69      private void configureMotors() {
70          rightShooter.follow(leftShooter);
71          leftShooter.setIdleMode(IdleMode.kBrake);
72          rightShooter.setIdleMode(IdleMode.kBrake);
73          leftShooter.setInverted(false);
74          rightShooter.setInverted(false);
75          leftShooter.setSmartCurrentLimit(50);
76          rightShooter.setSmartCurrentLimit(50);
77      }
78
79      @Override
80      public void periodic() {
81          // dashboard debugging values
82          SmartDashboard.putNumberArray("SmartDashboard/shooter shooter speeds", new
    double[]{
83                  leftShooterEncoder.getVelocity(),
84                  rightShooterEncoder.getVelocity()});
85          SmartDashboard.putNumber("SmartDashboard/shooter shooter current",
86                  (leftShooter.getOutputCurrent()+rightShooter.getOutputCurrent())/2);
87      }
88
89      // ACTIONS
90
91      public void stopShooter() {
92          leftShooter.stopMotor();
93      }
94
95      public void setShooterSpeedSetpoint(double setpoint) {
96          leftShooterController.setReference(setpoint, CANSparkMax.ControlType.
    kVelocity);
```

```
 97         rightShooterController.setReference(setpoint, CANSparkMax.ControlType.
    kVelocity);
 98     }
 99 
100 
101     /**
102      * @param speed the wanted speed of the motor
103      */
104     public void set(double speed) {
105         leftShooter.set(speed);
106     }
107 
108     // COMMANDS
109 
110     public Command stopShooterCommand() {
111         return this.runOnce(this::stopShooter);
112     }
113 
114 
115     public Command setShooterWithSpeed(double speed) {
116         return this.run(() -> set(speed));
117     }
118 
119     /**
120      * create a command to run the shooter at a speed for a given time
121      * @param speed the speed to run
122      * @param time the time to run that speed for
123      * @return the generated command
124      */
125     public Command runShooterSpeedForTime(double speed, double time) {
126         return this.runOnce(() -> setShooterSpeedSetpoint(speed)).deadlineWith(
    Commands.waitSeconds(time));
127     }
128 
129     /**
130      * a command to run the motors until they detect that they have collected a cube
131      * @param speed the speed to run at
132      * @return the generated command
133      */
134     public Command collect(double speed) {
135         return this.run(() -> set(speed)).until(() -> leftShooter.getOutputCurrent
    ()>30&&
136                 rightShooter.getOutputCurrent()>30).andThen(this::stopShooter);
137     }
138 
139     /**
140      * create a command to run the shooter at a certain speed given the angle to aim
     for
141      * @param level the shelf level to aim for
142      * @return the generated command
```

```
143      */
144     public Command runShooterWithVision(Constants.visionConstants.heights level) {
145         // get the camera results
146         results frontResults = frontCamera.getLatestResult();
147         results backResults = backCamera.getLatestResult();
148 
149         PhotonTrackedTarget frontBestTarget;
150         PhotonTrackedTarget backBestTarget;
151 
152         double angle;
153 
154         double distance = 0;
155 
156         // check if either of the cameras have targets.
157         // if they do get their best targets. defaults to the front camera
158         if (frontResults.hasTargets()) {
159             frontBestTarget = frontResults.getBestTarget();
160             distance = frontBestTarget.getBestCameraToTarget().getX();
161         }
162         else if (backResults.hasTargets()) {
163             backBestTarget = backResults.getBestTarget();
164             distance = backBestTarget.getBestCameraToTarget().getX();
165         }
166 
167         // calculate the speed to run at
168         angle = Math.atan(
169                 (2/distance) *
170                         (level.getHeightDiff() + Constants.visionConstants.maxHeight
     +
171                                 Math.sqrt(Math.pow(Constants.visionConstants.
    maxHeight, 2) +
172                                         level.getHeightDiff() *
173                                                 Constants.visionConstants.maxHeight
    )))+(Math.PI/2);
174 
175         // create and return the command
176         return runShooterSpeedForTime((Math.sqrt(2* Constants.visionConstants.g*
177                 (level.getHeightDiff() +
178                         Constants.visionConstants.maxHeight)))/Math.sin(angle), 1);
179     }
180 }
181 
182 
```

```java
package frc.robot.subsystems;


import com.revrobotics.*;
import com.revrobotics.CANSparkMaxLowLevel.MotorType;
import edu.wpi.first.math.system.plant.DCMotor;
import edu.wpi.first.wpilibj.RobotBase;
import edu.wpi.first.wpilibj.smartdashboard.Mechanism2d;
import edu.wpi.first.wpilibj.smartdashboard.MechanismLigament2d;
import edu.wpi.first.wpilibj.smartdashboard.MechanismRoot2d;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj2.command.Command;
import edu.wpi.first.wpilibj2.command.SubsystemBase;
import frc.robot.Constants;
import frc.robot.Constants.visionConstants;
import frc.robot.Constants.cuberConstants;
import frc.robot.vision.results;
import frc.robot.vision.visionWrapper;
import org.photonvision.targeting.PhotonTrackedTarget;

import java.util.function.DoubleSupplier;

/**
 * this class allows you to control the angle motor either manually or with vision
 */
public class AngleController extends SubsystemBase {
    private final CANSparkMax angleMotor;

    private final AbsoluteEncoder angleEncoder;

    private final SparkMaxPIDController angleController;

    private final visionWrapper frontCamera, backCamera;

    // testing out simulating the claw
    private final Mechanism2d mech = new Mechanism2d(10, 11);
    private final MechanismRoot2d root = mech.getRoot("root", 1, 0);
    private final MechanismLigament2d claw = root.append(new MechanismLigament2d("
claw", 9.8, 0));

    /**
     * constructs a new angle controller and gives it two cameras.
     * @param frontCamera the camera in front
     * @param backCamera the camera in back
     */
    public AngleController(visionWrapper frontCamera, visionWrapper backCamera) {
        angleMotor = new CANSparkMax(cuberConstants.angleMotorPort, MotorType.
kBrushless);

        // configure the motor
```

```
49          angleMotor.setIdleMode(CANSparkMax.IdleMode.kBrake);
50          angleMotor.setInverted(false);
51          angleMotor.setSmartCurrentLimit(50);
52          // to enable when I have the correct number of rotations
53  //        angleMotor.setSoftLimit(CANSparkMax.SoftLimitDirection.kForward, 0);
54  //        angleMotor.setSoftLimit(CANSparkMax.SoftLimitDirection.kReverse, 0.3388F);
55
56          // set the encoder to be the connected absolute encoder
57          angleEncoder = angleMotor.getAbsoluteEncoder(SparkMaxAbsoluteEncoder.Type.
    kDutyCycle);
58
59          // set up the pid controller
60          angleController = angleMotor.getPIDController();
61          angleController.setP(cuberConstants.angleP);
62          angleController.setI(cuberConstants.angleI);
63          angleController.setD(cuberConstants.angleD);
64          angleController.setFeedbackDevice(angleEncoder);
65
66          // define the zero offset TODO: get the correct offset
67          angleEncoder.setZeroOffset(0);
68
69
70          this.frontCamera = frontCamera;
71          this.backCamera = backCamera;
72
73          // put the mechanism to the dashboard
74          SmartDashboard.putData("claw", mech);
75
76          // if it's a simulation, add a spark max to it.
77          if (RobotBase.isSimulation()) {
78              REVPhysicsSim.getInstance().addSparkMax(angleMotor, DCMotor.getNEO(1));
79          }
80      }
81
82      @Override
83      public void periodic() {}
84
85      @Override
86      public void simulationPeriodic() {
87          // update the simulation and put the speed of the angle motor
88          claw.setAngle(angleEncoder.getPosition()*360);
89          REVPhysicsSim.getInstance().run();
90          SmartDashboard.putNumber("angle speed", angleMotor.getEncoder().getVelocity
    ());
91      }
92
93      // ACTIONS
94
95      public void stopAngleMotor() {
96          angleMotor.stopMotor();
```

```
 97     }
 98
 99     /**
100      * set the setpoint angle
101      * @param setpoint the wanted setpoint
102      */
103     public void setAngleSetpoint(double setpoint) {
104         angleController.setReference(setpoint, CANSparkMax.ControlType.kPosition);
105     }
106
107     /**
108      * set the angle of the shooter according to the wanted shelf to shoot too
109      * @param level the shelf level
110      */
111     public void setTargetAngleVision(visionConstants.heights level) {
112         // get the latest results
113         results frontResults = frontCamera.getLatestResult();
114         results backResults = backCamera.getLatestResult();
115
116         PhotonTrackedTarget frontBestTarget;
117         PhotonTrackedTarget backBestTarget;
118
119         double distance = 0;
120
121         // if there are results get the distance from them. The front camera is
    prioritized
122         if (frontResults.hasTargets()) {
123             frontBestTarget = frontResults.getBestTarget();
124             distance = frontBestTarget.getBestCameraToTarget().getX();
125         }
126         else if (backResults.hasTargets()) {
127             backBestTarget = backResults.getBestTarget();
128             distance = backBestTarget.getBestCameraToTarget().getX();
129         }
130
131         // calculate the angle and set it as the setpoint
132         setAngleSetpoint((Math.atan(
133                 (2/distance) *
134                         (level.getHeightDiff() + Constants.visionConstants.maxHeight
     +
135                                 Math.sqrt(Math.pow(Constants.visionConstants.
    maxHeight, 2) +
136                                         level.getHeightDiff() *
137                                                 Constants.visionConstants.maxHeight
    )))+(Math.PI/2))/2*Math.PI);
138     }
139
140     /**
141      * set the speed of the angle motor
142      * @param speed the desired speed
```

```
143      */
144     public void setAngleMotor(double speed) {
145         angleMotor.set(speed);
146     }
147 
148     // GETTERS
149 
150     public double getAngle() {
151         return angleEncoder.getPosition();
152     }
153 
154     // COMMANDS
155 
156     /**
157      * create a command to stop the angle motor.
158      * @return the generated command
159      */
160     public Command stopAngleMotorCommand() {
161         return this.runOnce(this::stopAngleMotor);
162     }
163 
164     /**
165      * create a command to run the angle motor with a speed
166      * @param speed the desired speed
167      * @return the generated command
168      */
169     public Command setAngleWithSpeed(double speed) {
170         return this.run(() -> setAngleMotor(speed));
171     }
172 
173     /**
174      * create a command to turn to a wanted angle
175      * @param angle the desired angle
176      * @return the generated command
177      */
178     public Command turnToAngle(double angle) {
179         return this.
180                 runOnce(() -> setAngleSetpoint(angle));
181     }
182 
183     /**
184      * create a command to aim at a wanted cube shelf
185      * @param level the shelf level
186      * @return the generated command
187      */
188     public Command turnToAngleVision(visionConstants.heights level) {
189         return this.
190                 runOnce(() -> setTargetAngleVision(level));
191     }
192 
```

```
193     /**
194      * create a command to run the shooter with joysticks
195      * @param speed the speed supplier
196      * @return the generated command
197      */
198     public Command runWithJoysticks(DoubleSupplier speed) {
199         return this.run(() -> this.setAngleMotor(speed.getAsDouble()));
200     }
201 }
202
203
```