Armstrong Class

```
 1 package org.firstinspires.ftc.teamcode;
 2
 3 import com.qualcomm.hardware.bosch.BNO055IMU;
 4 import com.qualcomm.hardware.bosch.
   JustLoggingAccelerationIntegrator;
 5 import com.qualcomm.robotcore.hardware.CRServo;
 6 import com.qualcomm.robotcore.hardware.DcMotor;
 7 import com.qualcomm.robotcore.hardware.Gamepad;
 8 import com.qualcomm.robotcore.hardware.HardwareMap;
 9 import com.qualcomm.robotcore.hardware.Servo;
10
11 import org.firstinspires.ftc.robotcore.external.navigation.
   Acceleration;
12 import org.firstinspires.ftc.robotcore.external.navigation.
   AngleUnit;
13 import org.firstinspires.ftc.robotcore.external.navigation.
   AxesOrder;
14 import org.firstinspires.ftc.robotcore.external.navigation.
   AxesReference;
15 import org.firstinspires.ftc.robotcore.external.navigation.
   Orientation;
16
17 /*This class is made by FTC #12535 Revolutionary Robots for
   our robot Armstrong. The variables and
18 methods used in this program are made for the upper half of
   Armstrong. This includes the turret,
19 lift, arm, and claw.*/
20
21 public class Armstrong
22 {
23
24     //Gamepad Variables
25     Gamepad gamepad1;
26     Gamepad gamepad2;
27
28     //Turret Motor
29     DcMotor turret;
30
31     //Lift Motor
32     DcMotor lift;
33
34     //Arm Motor
```

```
35      DcMotor arm;
36
37      //Claw Continuous Rotation Servos
38      CRServo leftClaw;
39      CRServo rightClaw;
40
41      //Capstone Servo
42      Servo capstone;
43
44      //Gyroscope
45      BNO055IMU imu;
46
47      //Classes for Gyroscope
48      Orientation angles;
49      Acceleration gravity;
50
51      //Rotation Speed Limiter
52      int rSpd = 1;
53
54      //Turret Encoder Tracker
55      int turTicks;
56      int turGoal;
57      int turTotal;
58
59      //Turret Angle
60      float tDeg;
61
62      //Arm Speed Limiter
63      double aSpd = 0.5;
64
65      //Lift Idle Power
66      double lIdle = 0;
67      //Arm Idle Power
68      double aIdle = 0;
69      //Claw IdlePower
70      double cIdle = 0;
71
72      public Armstrong (Gamepad g1, Gamepad g2, DcMotor t,
    DcMotor l, DcMotor a, CRServo lC, CRServo rC, Servo c,
    BNO055IMU i)
73      {
74
```

```
75          //The constructor of the class allows the class to be
     used on other programs as well as
76          //synchronizing the variables from the main program
77
78          gamepad1 = g1;
79          gamepad2 = g2;
80
81          turret = t;
82
83          lift = l;
84
85          arm = a;
86
87          leftClaw = lC;
88          rightClaw = rC;
89
90          capstone = c;
91
92          imu = i;
93
94          BNO055IMU.Parameters parameters = new BNO055IMU.
    Parameters();
95          parameters.angleUnit           = BNO055IMU.AngleUnit.
    DEGREES;
96          parameters.accelUnit           = BNO055IMU.AccelUnit.
    METERS_PERSEC_PERSEC;
97          parameters.calibrationDataFile = "
    BNO055IMUCalibration.json"; // see the calibration sample
    opmode
98          parameters.loggingEnabled      = true;
99          parameters.loggingTag          = "IMU";
100         parameters.accelerationIntegrationAlgorithm = new
    JustLoggingAccelerationIntegrator();
101         imu.initialize(parameters);
102
103         angles   = imu.getAngularOrientation(AxesReference.
    INTRINSIC, AxesOrder.ZYX, AngleUnit.DEGREES);
104
105         tDeg = angles.firstAngle;
106
107     }
108
```

```
109     void teleOpPackage ()
110     {
111
112         if(gamepad2.dpad_left)
113         {
114
115             tLeftNoStop(0.25, 950, false);
116
117         } else if (gamepad2.dpad_right)
118         {
119
120             tRightNoStop(0.25, 950, false);
121
122         } else
123         {
124
125             if (!turret.isBusy() || gamepad2.right_stick_x !=
     0)
126             {
127
128                 turret.setMode(DcMotor.RunMode.
    STOP_AND_RESET_ENCODER);
129                 turret.setMode(DcMotor.RunMode.
    RUN_WITHOUT_ENCODER);
130
131                 turret.setPower(-gamepad2.right_stick_x);
132
133             }
134
135         }
136
137         //Sets the turret to the value of the right stick
    including the speed limiter
138         //turret.setPower(-gamepad2.right_stick_x/rSpd);
139
140         //Turret Speed Controller
141         if (gamepad2.a)
142         {
143
144             //Full Speed
145             rSpd = 1;
146
```

```
147         } else if (gamepad2.b)
148         {
149
150             //Half Speed
151             rSpd = 2;
152
153         }
154
155         //Sets the power of the lift to the left stick y axis
156         lift.setPower(gamepad2.left_stick_y);
157
158         //Arm power controller
159         if (gamepad2.right_trigger != 0)
160         {
161
162             //Right trigger for moving the arm up
163             arm.setPower(aSpd);
164
165         } else if (gamepad2.left_trigger != 0)
166         {
167
168             //Left trigger for moving the arm down
169             arm.setPower(-aSpd);
170
171         } else
172         {
173
174             //Idle
175             arm.setPower(0.1);
176
177         }
178
179         if (gamepad2.x)
180         {
181
182             aSpd = 0.5;
183
184         } else if (gamepad2.y)
185         {
186
187             aSpd = 0.75;
188
```

```
189         }
190
191         /*Capstone Dropper
192         if (gamepad1.dpad_up && gamepad2.y)
193         {
194
195             //Drop Capstone
196             capstone.setPosition(0);
197
198         } else
199         {
200
201             //Stow Capstone
202             capstone.setPosition(1);
203
204         }*/
205
206         //Claw Power Controller
207         if (gamepad2.right_bumper)
208         {
209
210             //Clamp
211             leftClaw.setPower(-1);
212             rightClaw.setPower(1);
213
214         } else if (gamepad2.left_bumper)
215         {
216
217             //Release
218             leftClaw.setPower(1);
219             rightClaw.setPower(-1);
220
221         } else
222         {
223
224             //Idle
225             leftClaw.setPower(0);
226             rightClaw.setPower(0);
227
228         }
229
230     }
```

```
231
232     int getRSpd ()
233     {
234
235         //Returns the turret speed limiter to the main
    program for telemetry
236         return this.rSpd;
237
238     }
239
240     double getASpd ()
241     {
242
243         //Returns the arm speed limiter
244         return this.aSpd;
245
246     }
247
248     int getTurTicks ()
249     {
250
251         //Sets variable to turret encoder
252         turTicks = turret.getCurrentPosition();
253
254         //Returns the turret encoder values to the main
255         return turTicks;
256
257     }
258
259     void tRight (double spd, int tic)
260     {
261
262         //Program to turn turret right in autonomous with
    encoders
263         //The rest of the movement methods will be the same,
    except the NoStop,
264         // lift, and claw methods
265
266         //Stops and resets the encoder on the turret
267         turret.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER
    );
268
```

```
269         //Sets motor to run using the encoder
270         turret.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
271
272         //Sets motor to run to the given tick position
273         turret.setTargetPosition(-tic);
274
275         //Sets motors to run to the target position
276         turret.setMode(DcMotor.RunMode.RUN_TO_POSITION);
277
278         //Sets motor power to given speed
279         turret.setPower(spd);
280
281         //Waits until encoder hits target position
282         while (turret.isBusy());
283
284         //Stops robot
285         kill();
286
287         //Resets encoder for next method
288         turret.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER
    );
289
290         //Allows TeleOp joystick movement if needed
291         turret.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
292
293     }
294
295     void tRightNoStop (double spd, int tic, boolean reset)
296     {
297
298         //Same program as other methods as the other, but
    doesn't include the while loop
299
300         if (reset == true) {
301
302             turret.setMode(DcMotor.RunMode.
    STOP_AND_RESET_ENCODER);
303
304         }
305
306         turret.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
307
```

```
308         turret.setTargetPosition(-tic);
309
310         turret.setMode(DcMotor.RunMode.RUN_TO_POSITION);
311
312         turret.setPower(spd);
313
314         /*Awaiting
315         if(!turret.isBusy())
316         {
317
318             turret.setPower(0);
319
320         }
321          */
322
323     }
324
325     void tLeft (double spd, int tic)
326     {
327
328         turret.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER
    );
329
330         turret.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
331
332         turret.setTargetPosition(tic);
333
334         turret.setMode(DcMotor.RunMode.RUN_TO_POSITION);
335
336         turret.setPower(spd);
337
338         while (turret.isBusy());
339
340         kill();
341
342         turret.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER
    );
343
344         turret.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
345
346     }
347
```

```
348     void tLeftNoStop (double spd, int tic, boolean reset)
349     {
350
351         if (reset == true)
352         {
353
354             turret.setMode(DcMotor.RunMode.
    STOP_AND_RESET_ENCODER);
355
356         }
357
358         turret.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
359
360         turret.setTargetPosition(tic);
361
362         turret.setMode(DcMotor.RunMode.RUN_TO_POSITION);
363
364         turret.setPower(spd);
365
366     }
367
368     //Lift Method
369     void up (double spd, int time) throws
    InterruptedException
370     {
371
372         //Sets motor power to spd
373         lift.setPower(-spd);
374
375         //Waits for the given milliseconds
376         Thread.sleep(time);
377
378         //Stops Robot
379         kill();
380
381     }
382
383     void down (double spd, int time) throws
    InterruptedException
384     {
385
386         lift.setPower(spd);
```

```
387
388         Thread.sleep(time);
389
390         kill();
391
392     }
393
394     void rUp (double spd, int tic)
395     {
396
397         arm.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
398
399         arm.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
400
401         arm.setTargetPosition(tic);
402
403         arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
404
405         arm.setPower(spd);
406
407         while (arm.isBusy());
408
409         kill();
410
411         arm.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
412         arm.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
413
414         kill();
415
416     }
417
418     void rUpNoStop (double spd, int tic)
419     {
420
421         arm.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
422
423         arm.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
424
425         arm.setTargetPosition(tic);
426
427         arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
428
```

```
429         arm.setPower(spd);
430
431     }
432
433     void rDown (double spd, int tic)
434     {
435
436         arm.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
437
438         arm.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
439
440         arm.setTargetPosition(-tic);
441
442         arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
443
444         arm.setPower(spd);
445
446         while (arm.isBusy());
447
448         kill();
449
450         arm.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
451
452     }
453
454     void rDownNoStop (double spd, int tic)
455     {
456
457         arm.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
458
459         arm.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
460
461         arm.setTargetPosition(-tic);
462
463         arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
464
465         arm.setPower(spd);
466
467     }
468
469     void clamp (double spd, int time) throws
    InterruptedException
```

```
470     {
471
472         leftClaw.setPower(-spd);
473         rightClaw.setPower(spd);
474
475         Thread.sleep(time);
476
477         leftClaw.setPower(cIdle);
478         rightClaw.setPower(cIdle);
479
480     }
481
482     void unclamp (double spd, int time) throws
    InterruptedException
483     {
484
485         leftClaw.setPower(spd);
486         rightClaw.setPower(-spd);
487
488         Thread.sleep(time);
489
490         leftClaw.setPower(0);
491         rightClaw.setPower(0);
492
493     }
494
495     void setCIdle (double i)
496     {
497
498         //Sets the idle power of the claw to the given i
499         cIdle = i;
500
501     }
502
503     void kill ()
504     {
505
506         //Stops all parts of the robot
507
508         turret.setPower(0);
509
510         lift.setPower(0);
```

```
511
512             arm.setPower(0);
513
514             leftClaw.setPower(-cIdle);
515             rightClaw.setPower(cIdle);
516
517             capstone.setPosition(0.1);
518
519         }
520
521 }
522
```