```python
import math
from threading import Thread
from pybricks.parameters import Button, Stop
from pybricks.robotics import DriveBase
from pybricks.tools import StopWatch, wait


# Contains many functions for the drivebase
class DriveBaseFull:
    def __init__(self, config, Lmotor, Rmotor, gyro, diameter, track, runButton=None, Llight=None, Rlight=None):
        self.drive = DriveBase(Lmotor.m, Rmotor.m, diameter, track)
        self.gyro = gyro
        self.ev3 = config.ev3
        self.runButton = runButton
        self.config = config
        self.LLmotor = Lmotor
        self.RLmotor = Rmotor
        self.Llight = Llight
        self.Rlight = Rlight

        if Llight != None and Rlight != None:
            self.readLightCal()

        self.SPEEDLIST = [self.getSpeed(dist)
                          for dist in range(0, config.SPEED_LIST_COUNT)]

    def getSpeed(self, distance):
        return round(math.sqrt(distance*2*self.config.ACCELERATION + self.config.STARTSPEED**2))

    # Gets current heading
    def getHead(self):
        return (self.gyro.angle() + 180) % 360 - 180

    # Sets current heading
    def setHead(self, angle=0):
        self.gyro.reset_angle(angle)

    def sign(self, x):
        return 1 if x >= 0 else -1

    def limit(self, input, bound):
        return max(min(input, bound[1]), bound[0])

    # Checks if gyro reading has changed within timeout seconds
    def gyroDrift(self):
        heading = self.getHead()
        while self.config.state.getState() != 3:
            self.config.ev3.screen.print(self.getHead())
            wait(100)
        self.ev3.speaker.beep(1000, 200)

    # Runs the wheel at a constant speed
    def tyreClean(self):
        self.drive.drive(200, 0)
        while self.config.state.getState() != 3:
            wait(50)
        self.drive.stop()

    """ Sensor modes
    0 - Two sensor mode
    1 - Left sensor mode
    2 - Right sensor mode

    IMPORTENT:
```

```
    Don't use negative speed or distance it won't work
    """

    def lineFollower(self, distance=None, speed=150, mode=0, kp=None, ki=0, kd=None):
        if self.config.state.getState() == 3:
            return

        self.drive.reset()

        # Use different set of default kp and kd for mode 0 and mode 1, 2
        if kp is None:
            if mode == 0:
                kp = 1
            else:
                kp = 1.2
        if kd is None:
            if mode == 0:
                kd = 5
            else:
                kd = 10

        next = False
        lastError = 0
        integral = 0
        if distance == None:
            curr_distance = abs(self.drive.distance())
            while True:
                if self.config.state.getState() == 3:
                    break
                if mode == 2:
                    if self.Llight.readLight() > 90:
                        next = True
                    elif self.Llight.readLight() < 5 and next == True:
                        break
                    error = self.Rlight.readLight() - 60
                else:
                    if self.Rlight.readLight() > 90:
                        next = True
                    elif self.Rlight.readLight() < 5 and next == True:
                        break
                    error = 60 - self.Llight.readLight()

                derivative = error - lastError
                lastError = error
                integral = (integral / 2) + error
                # Add in k values and put in variable for movement
                turnRate = (error * kp) + (integral * ki) + (derivative * kd)
                # error, turn_rate)
                ramped_speed = min(self.SPEEDLIST[abs(curr_distance)], speed)
                # Start to move robot
                if next == True:
                    self.drive.drive(50, turnRate)
                else:
                    self.drive.drive(ramped_speed, turnRate)
                curr_distance = self.drive.distance()
        else:
            curr_distance = abs(self.drive.distance())
            # Start PID line following until exit condition meet
            # Check if robot has completed the right distance
            while curr_distance <= distance:
                if self.config.state.getState() == 3:
                    break
                # Calculate error, derivative and integral
                if mode == 0:
                    error = self.Llight.readLight() - self.Rlight.readLight()
                elif mode == 1:
```

```
131                     error = 60 - self.Llight.readLight()
132                 else:
133                     error = self.Rlight.readLight() - 60
134 
135                 derivative = error - lastError
136                 lastError = error
137                 integral = (integral / 2) + error
138                 # Add in k values and put in variable for movement
139                 turnRate = (error * kp) + (integral * ki) + (derivative * kd)
140                 # error, turn_rate)
141                 ramped_speed = min(self.SPEEDLIST[curr_distance], speed)
142                 # Start to move robot
143                 self.drive.drive(ramped_speed, turnRate)
144                 curr_distance = abs(self.drive.distance())
145         self.stop()
146 
147     # Place robot before a space with maximum contrast to make sure lightsensors go
148     # through the maximum contrast
149     # Drive robot for 100mm to check for max and min light values for each
150     # colorsensor
151 
152     def lightCal(self):
153         Lmax = 0
154         Rmax = 0
155         Lmin = 100
156         Rmin = 100
157         cancel = False
158 
159         self.drive.reset()
160         self.drive.drive(50, 0)
161         while self.drive.distance() < 100:
162             Lmax = max(self.Llight.sensor.reflection(), Lmax)
163             Rmax = max(self.Rlight.sensor.reflection(), Rmax)
164             Lmin = min(self.Llight.sensor.reflection(), Lmin)
165             Rmin = min(self.Rlight.sensor.reflection(), Rmin)
166             if self.config.state.getState() == 3:
167                 cancel = True
168                 break
169 
170         if not cancel:
171             with open(self.config.LIGHTCAL_CONF, "w") as f:
172                 f.write(str(Lmax) + ",")
173                 f.write(str(Lmin) + ",")
174                 f.write(str(Rmax) + ",")
175                 f.write(str(Rmin) + ",")
176             self.Llight.setCalValues(Lmin, Lmax)
177             self.Rlight.setCalValues(Rmin, Rmax)
178             print("Llight %2i:%2i Rlight %2i:%2i" % (Lmin, Lmax, Rmin, Rmax))
179         self.drive.stop()
180 
181     # Read light calibration values from file and set it through the \
182     # light_sensor class
183     # function setCalValues()
184 
185     def readLightCal(self):
186         try:
187             with open(self.config.LIGHTCAL_CONF, "r") as f:
188                 lines = f.readlines()
189                 lines = lines[0].split(",")
190                 light_Lmax = int(lines[0])
191                 light_Lmin = int(lines[1])
192                 light_Rmax = int(lines[2])
193                 light_Rmin = int(lines[3])
194                 self.Llight.setCalValues(light_Lmin, light_Lmax)
195                 self.Rlight.setCalValues(light_Rmin, light_Rmax)
196         except:
```

```
197             print("Lightcal file does not exist")
198
199     # Movement Functions
200
201     # Calculate how much to turn using the heading wanted to turn to
202
203     def turnAngle(self, heading):
204         return (heading - self.getHead() + 180) % 360 - 180
205
206     # Calculates speed using a precalculated lookup table
207
208     def rampSpeed(self, distance, curr_distance, speedLimit):
209         if curr_distance > distance / 2:
210             delta_distance = round(abs(distance - curr_distance))
211         else:
212             delta_distance = round(abs(curr_distance))
213         speed = self.SPEEDLIST[min(
214             delta_distance, self.config.SPEED_LIST_COUNT-1)]
215         return self.sign(speedLimit) * min(speed, abs(speedLimit))
216
217     # angle: angle to turn in degrees
218
219     def turnSpeed(self, angle):
220         turn_speed = angle / 180 * (self.config.TURN_SPEED_MAX - self.config.
        TURN_SPEED_MIN) +\
221             self.sign(angle) * self.config.TURN_SPEED_MIN
222         return turn_speed
223
224     def stop(self):
225         self.drive.drive(0, 0)
226         wait(100)
227         self.drive.stop()
228
229     # When heading is set to None, moves with current heading
230
231     def moveDist(self, distance, speed=600, heading=None, turn=True, up=True, down=True,
        timeout=None):
232         if self.config.state.getState() == 3:
233             return
234
235         posDistance = abs(distance)
236         if speed < 0:
237             print("Error Negative speed", speed)
238             return
239
240         if heading == None:
241             heading = self.getHead()
242         elif turn and abs(self.turnAngle(heading)) > 5:
243             self.turnTo(heading)
244
245         rampSpeed_max = self.rampSpeed(posDistance, posDistance/2, speed)
246         if timeout == None:
247             # * 2000 to double time and convert to milliseconds
248             timeout = (posDistance / rampSpeed_max) * 2 * 1000 + 500
249         # logData = []
250
251         self.drive.reset()
252         timer = StopWatch()
253         while self.config.state.getState() != 3 and timer.time() < timeout:
254             # print(runState.getStopFlag(), runButton.pressed())
255             curr_distance = abs(self.drive.distance())
256             if curr_distance >= posDistance:
257                 break
258             if up == False and curr_distance < posDistance/2:
259                 drive_speed = speed
260             elif down == False and curr_distance > posDistance/2:
```

```
261                 drive_speed = speed
262             else:
263                 drive_speed = self.rampSpeed(posDistance, curr_distance, speed)
264 
265             self.drive.drive(drive_speed*self.sign(distance),
266                              self.turnAngle(heading))
267             # print("Speed, drive_speed, distance: ", speed, drive_speed, \
268             #         curr_distance)
269             # logData.append([drive_speed, curr_distance])
270         # print("MoveDist timeout=", timeout, "ms")
271         self.stop()
272         # print("current distance, max speed:", robot.distance(), max_speed)
273         # with open("moveDist.csv", "w") as f:
274         #   f.write("Drive Speed, Distance\n")
275         #   for line in logData:
276         #     f.write("{}, {}".format(line[0], line[1]))
277         #     f.write("\n")
278 
279     # Moves robot along the radius of a circle
280     def moveArc(self, radius, heading, speed=100, timeout=10000):
281         if self.config.state.getState() == 3:
282             return
283 
284         turn_rate = (360 * speed) / (math.pi * 2 * radius)
285         tolerance = int(2 * abs(speed) / 100)
286 
287         # st_heading = self.getHead()
288         runTime = StopWatch()
289         self.drive.drive(speed, turn_rate)
290         while self.turnAngle(heading) not in range(-tolerance, tolerance) and runTime.
            time() < timeout:
291             if self.config.state.getState() == 3:
292                 break
293         self.stop()
294         # wait(1000)
295         # print(tolerance, st_heading, "->", self.getHead(), ":", self.getHead() -
            st_heading)
296 
297     # Turns the robot to a given heading
298     def turnTo(self, heading, tolerance=2, timeout=4000):
299         if self.config.state.getState() == 3:
300             return
301 
302         angle = self.turnAngle(heading)
303         runTime = StopWatch()
304         while angle not in range(-tolerance, tolerance) and runTime.time() < timeout:
305             if self.config.state.getState() == 3:
306                 break
307             self.drive.drive(0, self.turnSpeed(angle))
308             angle = self.turnAngle(heading)
309         self.stop()
310         # print(heading, self.getHead(), range(-tolerance, tolerance))
311 
312     # Turns quickly first, then uses turnTo for more accuracy
313     def spinTo(self, heading, tolerance=2, timeout=4000):
314         if self.config.state.getState() == 3:
315             return
316 
317         angle = self.turnAngle(heading)
318         if angle not in range(-30, 30):
319             self.drive.turn(angle - self.sign(angle) * 10)
320         self.turnTo(heading, tolerance=tolerance, timeout=timeout)
321         self.stop()
322 
323     """ Under Development
324     Colour reading not very accurate.
```

```
325         """
326
327         def moveColour(self, sensor, colorlist, heading=None):
328             if self.config.state.getState() == 3:
329                 return
330
331             if heading == None:
332                 heading = self.getHead()
333             self.drive.drive(50, self.turnAngle(heading))
334             for colour in colorlist:
335                 while sensor.color() != colour:
336                     self.drive.drive(50, self.turnAngle(heading))
337             self.stop()
338
339         # Moves forward until given lightsensor value is with the limits
340         def moveLight(self, sensor, limits, heading=None, timeout=10000):
341             if self.config.state.getState() == 3:
342                 return
343
344             timer = StopWatch()
345             if heading == None:
346                 heading = self.getHead()
347             else:
348                 if self.turnAngle(heading) not in range(-5, 5):
349                     self.turnTo(heading)
350             while sensor.readLight() not in range(int(limits[0]), int(limits[1])) and timer.
                time() < timeout:
351                 if self.config.state.getState() == 3:
352                     break
353                 self.drive.drive(80, self.turnAngle(heading))
354             self.stop()
355
356         # Moves until wheels have stalled
357         def moveStall(self, duty=30, heading=None, speed=-200, timeout=10000):
358             if self.config.state.getState() == 3:
359                 return
360
361             if heading != None:
362                 self.turnTo(heading)
363             l_angle = None
364             r_angle = None
365             timer = StopWatch()
366             L = Thread(target=self.LLmotor.run_until_stalled,
367                        args=(speed, Stop.COAST, duty)).start()
368             R = Thread(target=self.RLmotor.run_until_stalled,
369                        args=(speed, Stop.COAST, duty)).start()
370             wait(50)
371             while (self.LLmotor.control.done() == False or self.RLmotor.control.done() ==
                False) and timer.time() < timeout:
372                 if self.config.state.getState() == 3:
373                     self.stop()
374                     return
375             self.stop()
376
377         # 1 for the black line in front, -1 for the white line in front
378         def lineReset(self, direction=1, timeout=10000):
379             timer = StopWatch()
380             while True:
381                 if self.Llight.readLight() in range(45, 55) and self.Rlight.readLight() in
                    range(45, 55):
382                     break
383                 if timer.time() > timeout:
384                     break
385                 if self.config.state.getState() == 3:
386                     break
387
```

```
388                 self.LLmotor.run((self.Llight.readLight()-50)*1.5*direction)
389                 self.RLmotor.run((self.Rlight.readLight()-50)*1.5*direction)
390             self.stop()
391
```