```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, timeout=20000):
        timer = StopWatch()
        heading = self.getHead()
        while timer.time() < timeout and 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:
                    error = 60 - self.Llight.readLight()
                else:
                    error = self.Rlight.readLight() - 60

                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[curr_distance], speed)
                # Start to move robot
                self.drive.drive(ramped_speed, turnRate)
                curr_distance = abs(self.drive.distance())
        self.stop()

    # Place robot before a space with maximum contrast to make sure lightsensors go
    # through the maximum contrast
    # Drive robot for 100mm to check for max and min light values for each
    # colorsensor

    def lightCal(self):
        Lmax = 0
        Rmax = 0
        Lmin = 100
        Rmin = 100
        cancel = False

        self.drive.reset()
        self.drive.drive(50, 0)
        while self.drive.distance() < 100:
            Lmax = max(self.Llight.sensor.reflection(), Lmax)
            Rmax = max(self.Rlight.sensor.reflection(), Rmax)
            Lmin = min(self.Llight.sensor.reflection(), Lmin)
            Rmin = min(self.Rlight.sensor.reflection(), Rmin)
            if self.config.state.getState() == 3:
                cancel = True
                break

        if not cancel:
            with open(self.config.LIGHTCAL_CONF, "w") as f:
                f.write(str(Lmax) + ",")
                f.write(str(Lmin) + ",")
                f.write(str(Rmax) + ",")
                f.write(str(Rmin) + ",")
            self.Llight.setCalValues(Lmin, Lmax)
            self.Rlight.setCalValues(Rmin, Rmax)
            print("Llight %2i:%2i Rlight %2i:%2i" % (Lmin, Lmax, Rmin, Rmax))
        self.drive.stop()

    # Read light calibration values from file and set it through the \
    # light_sensor class
    # function setCalValues()

    def readLightCal(self):
        try:
            with open(self.config.LIGHTCAL_CONF, "r") as f:
                lines = f.readlines()
                lines = lines[0].split(",")
                light_Lmax = int(lines[0])
                light_Lmin = int(lines[1])
                light_Rmax = int(lines[2])
                light_Rmin = int(lines[3])
                self.Llight.setCalValues(light_Lmin, light_Lmax)
                self.Rlight.setCalValues(light_Rmin, light_Rmax)
        except:
            print("Lightcal file does not exist")
    # Movement Functions

    # Calculate how much to turn using the heading wanted to turn to

    def turnAngle(self, heading):
        return (heading - self.getHead() + 180) % 360 - 180

    # Calculates speed using a precalculated lookup table

    def rampSpeed(self, distance, curr_distance, speedLimit):
        if curr_distance > distance / 2:
            delta_distance = round(abs(distance - curr_distance))
        else:
            delta_distance = round(abs(curr_distance))
        speed = self.SPEEDLIST[min(
            delta_distance, self.config.SPEED_LIST_COUNT-1)]
        return self.sign(speedLimit) * min(speed, abs(speedLimit))

    # angle: angle to turn in degrees

    def turnSpeed(self, angle):
        turn_speed = angle / 180 * (self.config.TURN_SPEED_MAX - self.config.TURN_SPEED_MIN) +\
            self.sign(angle) * self.config.TURN_SPEED_MIN
        return turn_speed

    def stop(self):
        self.drive.drive(0, 0)
        wait(100)
        self.drive.stop()

    # When heading is set to None, moves with current heading

    def moveDist(self, distance, speed=400, heading=None, turn=True, up=True, down=True, timeout=None):
        if self.config.state.getState() == 3:
            return

        posDistance = abs(distance)
        if speed < 0:
            print("Error Negative speed", speed)
            return

        if heading == None:
            heading = self.getHead()
        elif turn and abs(self.turnAngle(heading)) > 5:
            self.turnTo(heading)

        rampSpeed_max = self.rampSpeed(posDistance, posDistance/2, speed)
        if timeout == None:
            # * 2000 to double time and convert to milliseconds
            timeout = (posDistance / rampSpeed_max) * 2 * 1000 + 500
        # logData = []

        self.drive.reset()
        timer = StopWatch()
        while self.config.state.getState() != 3 and timer.time() < timeout:
            # print(runState.getStopFlag(), runButton.pressed())
            curr_distance = abs(self.drive.distance())
            if curr_distance >= posDistance:
                break
            if up == False and curr_distance < posDistance/2:
                drive_speed = speed
            elif down == False and curr_distance > posDistance/2:
                drive_speed = speed
            else:
                drive_speed = self.rampSpeed(posDistance, curr_distance, speed)
```

```python

            self.drive.drive(drive_speed*self.sign(distance),
                             self.turnAngle(heading))
            # print("Speed, drive_speed, distance: ", speed, drive_speed, \
            #        curr_distance)
            # logData.append([drive_speed, curr_distance])
        # print("MoveDist timeout=", timeout, "ms")
        self.stop()
        # print("current distance, max speed:", robot.distance(), max_speed)
        # with open("moveDist.csv", "w") as f:
        #   f.write("Drive Speed, Distance\n")
        #   for line in logData:
        #     f.write("{}, {}".format(line[0], line[1]))
        #     f.write("\n")

    # Moves robot along the radius of a circle
    def moveArc(self, radius, heading, speed=100, timeout=10000):
        if self.config.state.getState() == 3:
            return

        turn_rate = (360 * speed) / (math.pi * 2 * radius)
        tolerance = int(2 * abs(speed) / 100)

        # st_heading = self.getHead()
        runTime = StopWatch()
        self.drive.drive(speed, turn_rate)
        while self.turnAngle(heading) not in range(-tolerance, tolerance) and runTime.time() < timeout:
            if self.config.state.getState() == 3:
                break
        self.stop()
        # wait(1000)
        # print(tolerance, st_heading, "->", self.getHead(), ":", self.getHead() - st_heading)

    # Turns the robot to a given heading
    def turnTo(self, heading, tolerance=2, timeout=4000):
        if self.config.state.getState() == 3:
            return

        angle = self.turnAngle(heading)
        runTime = StopWatch()
        while angle not in range(-tolerance, tolerance) and runTime.time() < timeout:
            if self.config.state.getState() == 3:
                break
            self.drive.drive(0, self.turnSpeed(angle))
            angle = self.turnAngle(heading)
        self.stop()
        # print(heading, self.getHead(), range(-tolerance, tolerance))

    # Turns quickly first, then uses turnTo for more accuracy
    def spinTo(self, heading, tolerance=2, timeout=4000):
        if self.config.state.getState() == 3:
            return

        angle = self.turnAngle(heading)
        if angle not in range(-30, 30):
            self.drive.turn(angle - self.sign(angle) * 10)
        self.turnTo(heading, tolerance=tolerance, timeout=timeout)
        self.stop()

    """ Under Development
    Colour reading not very accurate.
    """

    def moveColour(self, sensor, colorlist, heading=None):
        if self.config.state.getState() == 3:
            return

        if heading == None:
            heading = self.getHead()
        self.drive.drive(50, self.turnAngle(heading))
        for colour in colorlist:
            while sensor.color() != colour:
                self.drive.drive(50, self.turnAngle(heading))
        self.stop()

    # Moves forward until given lightsensor value is with the limits
    def moveLight(self, sensor, limits, heading=None, timeout=10000):
        if self.config.state.getState() == 3:
            return

        timer = StopWatch()
        if heading == None:
            heading = self.getHead()
        else:
            if self.turnAngle(heading) not in range(-5, 5):
                self.turnTo(heading)
        while sensor.readLight() not in range(int(limits[0]), int(limits[1])) and timer.time() < timeout:
            if self.config.state.getState() == 3:
                break
            self.drive.drive(80, self.turnAngle(heading))
        self.stop()

    # Moves until wheels have stalled
    def moveStall(self, duty=30, heading=None, speed=-200, timeout=10000):
        if self.config.state.getState() == 3:
            return

        if heading != None:
            self.turnTo(heading)
        l_angle = None
        r_angle = None
        timer = StopWatch()
        L = Thread(target=self.LLmotor.run_until_stalled,
                   args=(speed, Stop.COAST, duty)).start()
        R = Thread(target=self.RLmotor.run_until_stalled,
                   args=(speed, Stop.COAST, duty)).start()
        wait(50)
        while (self.LLmotor.control.done() == False or self.RLmotor.control.done() == False) and timer.time() < timeout:
            if self.config.state.getState() == 3:
                self.stop()
                return
        self.stop()

    # 1 for the black line in front, -1 for the white line in front
    def lineReset(self, direction=1, timeout=10000):
        timer = StopWatch()
        while True:
            if self.Llight.readLight() in range(45, 55) and self.Rlight.readLight() in range(45, 55):
                break
            if timer.time() > timeout:
                break
            if self.config.state.getState() == 3:
                break

            self.LLmotor.run((self.Llight.readLight()-50)*1.5*direction)
            self.RLmotor.run((self.Rlight.readLight()-50)*1.5*direction)
        self.stop()

```