```
    def gyro_drive_straight_for_degrees(self, degrees, heading, max_speed, acceleration_distance,
     deceleration_distance):
        '''
        Drive straight for some number of degrees
        Heading indicates the angle to follow
        max_speed indicates the maximum speed for driving
        acceleration_distance and deceleration_distance is specified in number of degrees
        '''
        heading = heading * 10

        # Target is the final position, current keeps track of where robot is, and initial is
        where we started
        target_position = motor.relative_position(self.motor_port_left) - degrees
        current_position = motor.relative_position(self.motor_port_left)
        initial_position = current_position
        distance_traveled = 0
        distance = abs(degrees)
        last_angle_error = 0
        integral = 0

        while True:
            # How far off are we from where we should be going
            angle_error = self.get_gyro_angle() - heading
            # How is our error changing over time? (and correct for future predicted errors)
            derivative = angle_error - last_angle_error
            last_angle_error = angle_error
            # What is the history of our error?
            integral = integral + angle_error
            # Currently only using KP (derivative and integral coefficients are set to 0)
            correction = (self.DRIVE_KP * angle_error + self.DRIVE_KD * derivative + self.
            DRIVE_KI * integral) / 10

            # This is the acceleration curve
            if acceleration_distance > 0:
                # Calculate the acceleration speed by figuring out how far we are into the
                acceleration distance
                acceleration_speed = self.MIN_SPEED_IN_PERCENT + (distance_traveled /
                acceleration_distance) * (100 - self.MIN_SPEED_IN_PERCENT)
                if acceleration_speed > max_speed:
                    acceleration_speed = max_speed
            else:
                acceleration_speed = max_speed

            # This is the deceleration curve
            if deceleration_distance > 0:
                deceleration_speed = self.MIN_SPEED_IN_PERCENT + (distance /
                deceleration_distance) * (100 - self.MIN_SPEED_IN_PERCENT)
                if deceleration_speed > max_speed:
                    deceleration_speed = max_speed
            else:
                deceleration_speed = max_speed

            # This is the actual curve
            actual_speed = min(acceleration_speed, deceleration_speed)

            # Correct for the direction we are going
            if degrees <= 0:
                actual_speed = -1.0 * actual_speed

            speed_native = int(actual_speed * 10.5)
            left_speed = int(speed_native + correction)
            right_speed = int(speed_native - correction)

            motor_pair.move_tank(motor_pair.PAIR_1, left_speed, right_speed)
```

```
            current_position = motor.relative_position(self.motor_port_left)
            distance = abs(target_position - current_position)
            distance_traveled = abs(current_position - initial_position)

            if ((degrees > 0 and (target_position >= current_position)) or
                (degrees < 0 and (target_position <= current_position))):
                break

        motor_pair.stop(motor_pair.PAIR_1)
        self._wait_until_robot_is_stopped()

    def gyro_turn(self, relative_angle_in_degrees: float,
                          left_motor_max_speed_in_percent: int,
                          right_motor_max_speed_in_percent: int,
                          acceleration_percent: float,
                          deceleration_percent: float):
        '''
        relative_angle_in_degrees : Number of degrees that we want to turn the robot
                                    Positive number means turning clockwise
                                    Negative number means turning counter-clockwise
        left/right_motor_max_speed_in_percent : Maximum speed we will be turning the left /
        right motor
                                                Value is always positive (function will
                                                determine actual polarity)
                                                Note that we can accomplish pivot, point, or arc
                                                turns by setting
                                                max speeds appropriately
        acceleration_percent : Percentage of angular distance to accelerate over
        deceleration_percent : Percentage of angular distance to decelerate over
        '''
        current_angle = self.get_gyro_angle()
        initial_angle = current_angle
        target_angle = current_angle + relative_angle_in_degrees * 10
        # How far we have to turn
        angular_distance = abs(target_angle - current_angle)
        # How many degrees are we accelerating and decelerating for
        acceleration_distance = angular_distance * acceleration_percent / 100.0
        deceleration_distance = angular_distance * deceleration_percent / 100.0
        # Native speed
        left_max_speed_in_deg_per_sec = int(left_motor_max_speed_in_percent * 10.5)
        right_max_speed_in_deg_per_sec = int(right_motor_max_speed_in_percent * 10.5)
        if( abs(left_motor_max_speed_in_percent) > abs(right_motor_max_speed_in_percent) ):
            right_initial_speed_in_deg_per_sec = int(min(self.MIN_SPEED_IN_PERCENT, abs(
            right_motor_max_speed_in_percent)) * 10.5)
            if( right_motor_max_speed_in_percent < 0 ):
                right_initial_speed_in_deg_per_sec *= -1

            if( right_motor_max_speed_in_percent == 0 ):
                left_initial_speed_in_deg_per_sec = int(min(self.MIN_SPEED_IN_PERCENT, abs(
                left_motor_max_speed_in_percent)) * 10.5)
                if( left_motor_max_speed_in_percent < 0 ):
                    left_initial_speed_in_deg_per_sec *= -1
            else:
                left_initial_speed_in_deg_per_sec = int(10.5 * left_motor_max_speed_in_percent *
                right_initial_speed_in_deg_per_sec / right_max_speed_in_deg_per_sec)
        else:
            left_initial_speed_in_deg_per_sec = int(min(self.MIN_SPEED_IN_PERCENT, abs(
            left_motor_max_speed_in_percent)) * 10.5)
            if( left_motor_max_speed_in_percent < 0 ):
                left_initial_speed_in_deg_per_sec *= -1

            if( left_motor_max_speed_in_percent == 0 ):
                right_initial_speed_in_deg_per_sec = int(min(self.MIN_SPEED_IN_PERCENT, abs(
                right_motor_max_speed_in_percent)) * 10.5)
                if( right_motor_max_speed_in_percent < 0 ):
```

```
                right_initial_speed_in_deg_per_sec *= -1
        else:
            right_initial_speed_in_deg_per_sec = int(10.5 * right_motor_max_speed_in_percent
            * left_initial_speed_in_deg_per_sec / left_max_speed_in_deg_per_sec)

    left_speed_in_deg_per_sec = left_initial_speed_in_deg_per_sec
    right_speed_in_deg_per_sec = right_initial_speed_in_deg_per_sec

    if( left_motor_max_speed_in_percent > right_motor_max_speed_in_percent ):
        turning_cw = True
        # degrees are counting down so target must be less than current
        assert( target_angle < current_angle )
        acceleration_end = current_angle - acceleration_distance
        deceleration_start = target_angle + deceleration_distance
    elif( left_motor_max_speed_in_percent < right_motor_max_speed_in_percent ):
        turning_cw = False
        # degrees are counting up so target must be greater than current
        assert( target_angle > current_angle )
        acceleration_end = current_angle + acceleration_distance
        deceleration_start = target_angle - deceleration_distance
    else:
        print("Error: Invalid turning speeds")
        raise(SystemExit)

    while(True):
        current_angle = self.get_gyro_angle()
        if( turning_cw ):
            #print("CW CUR: " + str(current_angle) + " TGT " + str(target_angle))
            if( current_angle <= deceleration_start ):
                # Decelerating
                left_speed_in_deg_per_sec = int(left_max_speed_in_deg_per_sec +
                                        (left_initial_speed_in_deg_per_sec -
                                        left_max_speed_in_deg_per_sec) *
                                        abs(current_angle - deceleration_start) /
                                        deceleration_distance)
                right_speed_in_deg_per_sec = int(right_max_speed_in_deg_per_sec +
                                        (right_initial_speed_in_deg_per_sec -
                                        right_max_speed_in_deg_per_sec) *
                                        abs(current_angle - deceleration_start) /
                                        deceleration_distance)

            elif( current_angle > acceleration_end ):
                # Accelerating
                left_speed_in_deg_per_sec = int(left_initial_speed_in_deg_per_sec +
                                        (left_max_speed_in_deg_per_sec -
                                        left_initial_speed_in_deg_per_sec) *
                                        abs(current_angle - initial_angle) /
                                        acceleration_distance)
                right_speed_in_deg_per_sec = int(right_initial_speed_in_deg_per_sec +
                                        (right_max_speed_in_deg_per_sec -
                                        right_initial_speed_in_deg_per_sec) *
                                        abs(current_angle - initial_angle) /
                                        acceleration_distance)
            else:
                # MAX speed
                left_speed_in_deg_per_sec = left_max_speed_in_deg_per_sec
                right_speed_in_deg_per_sec = right_max_speed_in_deg_per_sec

            if( current_angle <= target_angle ):
                break
        else:
            #print("CCW CUR: " + str(current_angle) + " TGT " + str(target_angle))
            if( current_angle >= deceleration_start ):
                # Decelerating
                left_speed_in_deg_per_sec = int(left_max_speed_in_deg_per_sec +
```

```
                                    (left_initial_speed_in_deg_per_sec -
                                    left_max_speed_in_deg_per_sec) *
                                    abs(current_angle - deceleration_start) /
                                    deceleration_distance)
                right_speed_in_deg_per_sec = int(right_max_speed_in_deg_per_sec +
                                    (right_initial_speed_in_deg_per_sec -
                                    right_max_speed_in_deg_per_sec) *
                                    abs(current_angle - deceleration_start) /
                                    deceleration_distance)
            elif( current_angle < acceleration_end ):
                # Accelerating
                left_speed_in_deg_per_sec = int(left_initial_speed_in_deg_per_sec +
                                    (left_max_speed_in_deg_per_sec -
                                    left_initial_speed_in_deg_per_sec) *
                                    abs(current_angle - initial_angle) /
                                    acceleration_distance)
                right_speed_in_deg_per_sec = int(right_initial_speed_in_deg_per_sec +
                                    (right_max_speed_in_deg_per_sec -
                                    right_initial_speed_in_deg_per_sec) *
                                    abs(current_angle - initial_angle) /
                                    acceleration_distance)
            else:
                # MAX speed
                left_speed_in_deg_per_sec = left_max_speed_in_deg_per_sec
                right_speed_in_deg_per_sec = right_max_speed_in_deg_per_sec

            if( current_angle >= target_angle ):
                break

        motor_pair.move_tank(motor_pair.PAIR_1, left_speed_in_deg_per_sec,
        right_speed_in_deg_per_sec)

    motor_pair.stop(motor_pair.PAIR_1)
    self._wait_until_robot_is_stopped()
```