A Technical Guide to Calculating and Understanding Linear and Angular Velocities for Rover Navigation

The code aims to accurately measure and control the speed of wheels using a speedometer sensor and PID (Proportional-Integral-Derivative) controllers. It is specifically designed for implementation on an Arduino microcontroller. As shown in Figure 1

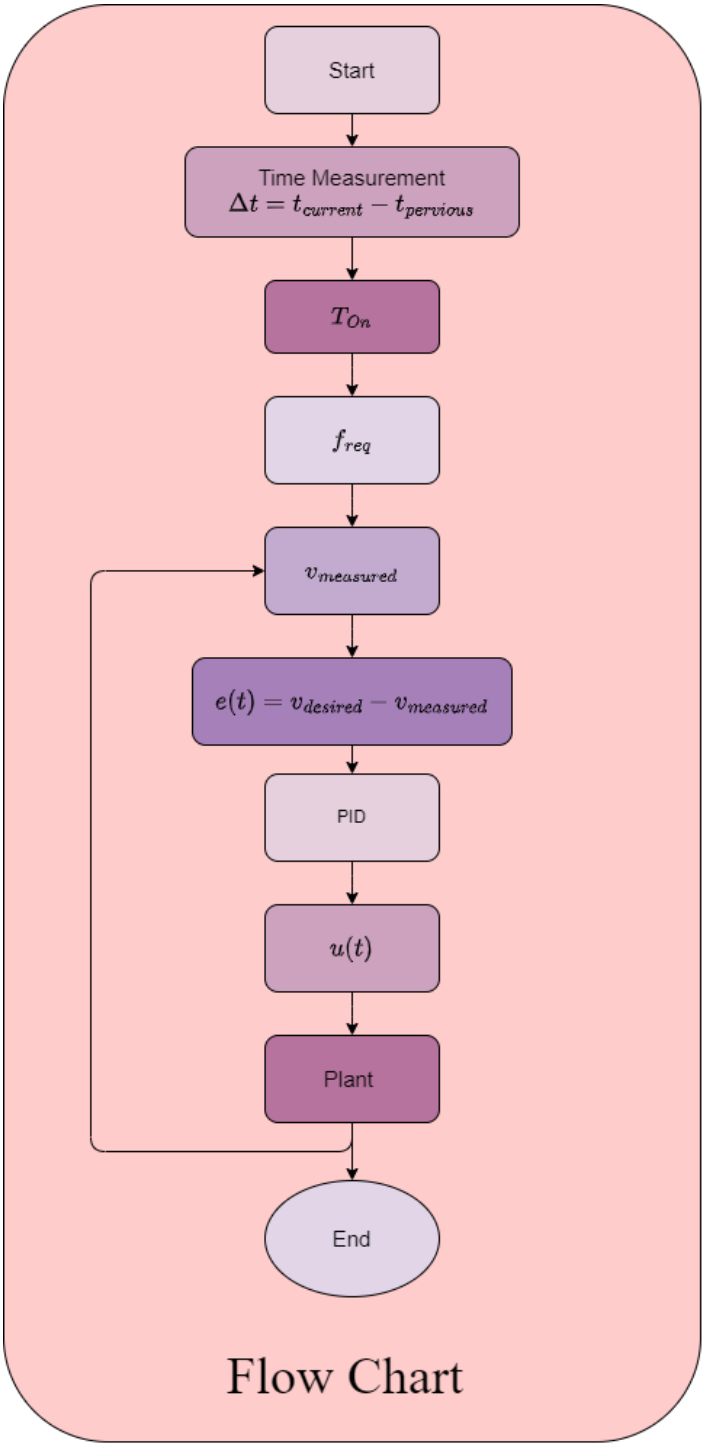

Figure 1-Flowchart

The code begins by defining various parameters and configuring necessary variables. It establishes the pin (speedometer) to which the speedometer sensor is connected and specifies important characteristics such as the speedometer frequency, maximum speed, and wheel radius. These values can be customized based on the specific hardware setup.In the setup() function, the code sets the speedometer pin as an input, preparing it to receive speedometer sensor readings.The core functionality is implemented in the loop() function, which performs the following steps repeatedly:

- The code measures the current time using the micros() function, which returns the time in microseconds. This allows the calculation of the time elapsed since the previous measurement.
- The time difference (deltaT) is calculated in seconds by subtracting the previous time value from the current time value. This provides the duration between consecutive measurements.
- The pulseIn() function is employed to determine the duration of the speedometer pulses. By measuring the high pulse duration on the speedometer pin, the code obtains the time during which the speedometer sensor output indicates wheel movement.
- Using the speedometer frequency and the pulse duration, the code calculates the number of pulses (noOfPulses) that occurred during the measurement period. This quantity represents the angular movement of the wheel.
- Based on the number of pulses and the maximum speed, the code computes the angular velocity measured (angularVelocityMeasured) in radians per second. This provides an indication of the rotational speed of the wheel.
- By multiplying the angular velocity measured by the wheel radius, the code determines the wheel's current linear velocity (currentVelocity) in meters per second. This represents the speed of the wheel.
- The code employs a PID control approach to regulate the wheel speed. It begins by calculating the error in velocity (errorVelocity) as the difference between the target velocity and the current velocity. This error represents the deviation from the desired speed. Figure 2 explains the PID controller structure

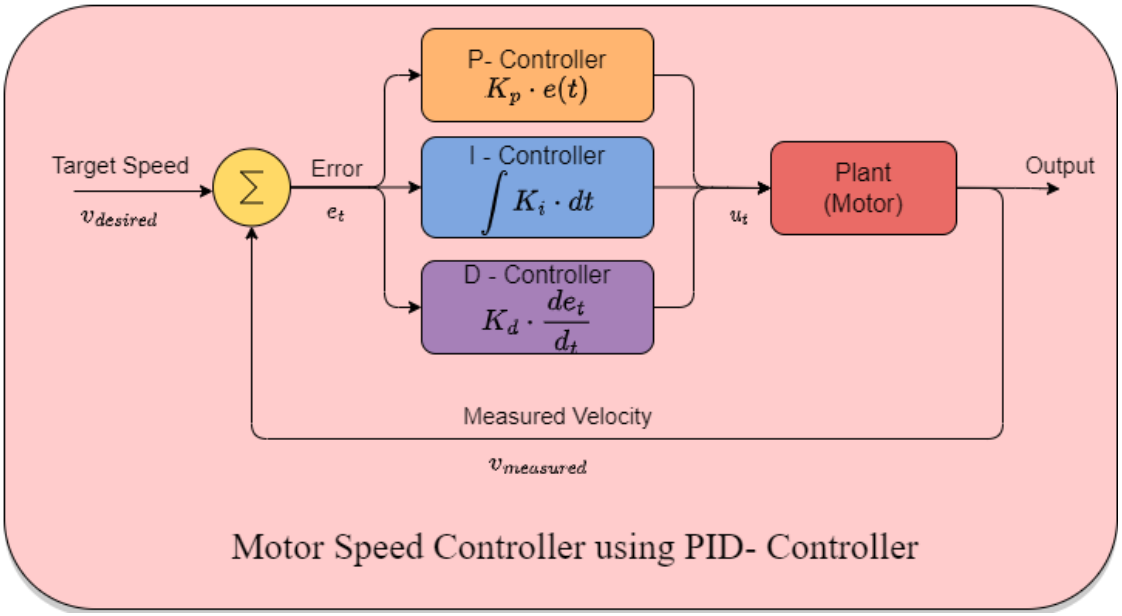

Figure 2-PID

- The integral error (errorI) is updated by adding the product of the error velocity and the time difference (deltaT). This accumulation accounts for the cumulative deviation over time.
- The derivative error (errorD) is computed by dividing the difference between the target and current velocities by deltaT. This term indicates the rate of change of the error and helps prevent overshooting or undershooting.
- The plant input (plantInput) is determined by passing the error values (errorVelocity, errorI, errorD) and the controller gains (kp, ki, kd) to the PIDController() function. This function applies the PID equation to calculate an appropriate control input for adjusting the speed.
- The previous time value is updated with the current time value, preparing it for the next iteration and ensuring accurate time measurements.
- Finally, the code includes a PIDController() function that receives the error values and controller gains as inputs. Within this function, the plant input is computed by applying the PID equation, which involves multiplying the error terms by their respective gains and summing the results.

Overall, this code provides a comprehensive framework for measuring wheel speed using a speedometer sensor and controlling the speed through the use of PID controllers. Continuously monitoring and adjusting the speed based on the target velocity enables precise control and regulation of the wheel's rotational motion.

Differential Rover Class Documentation

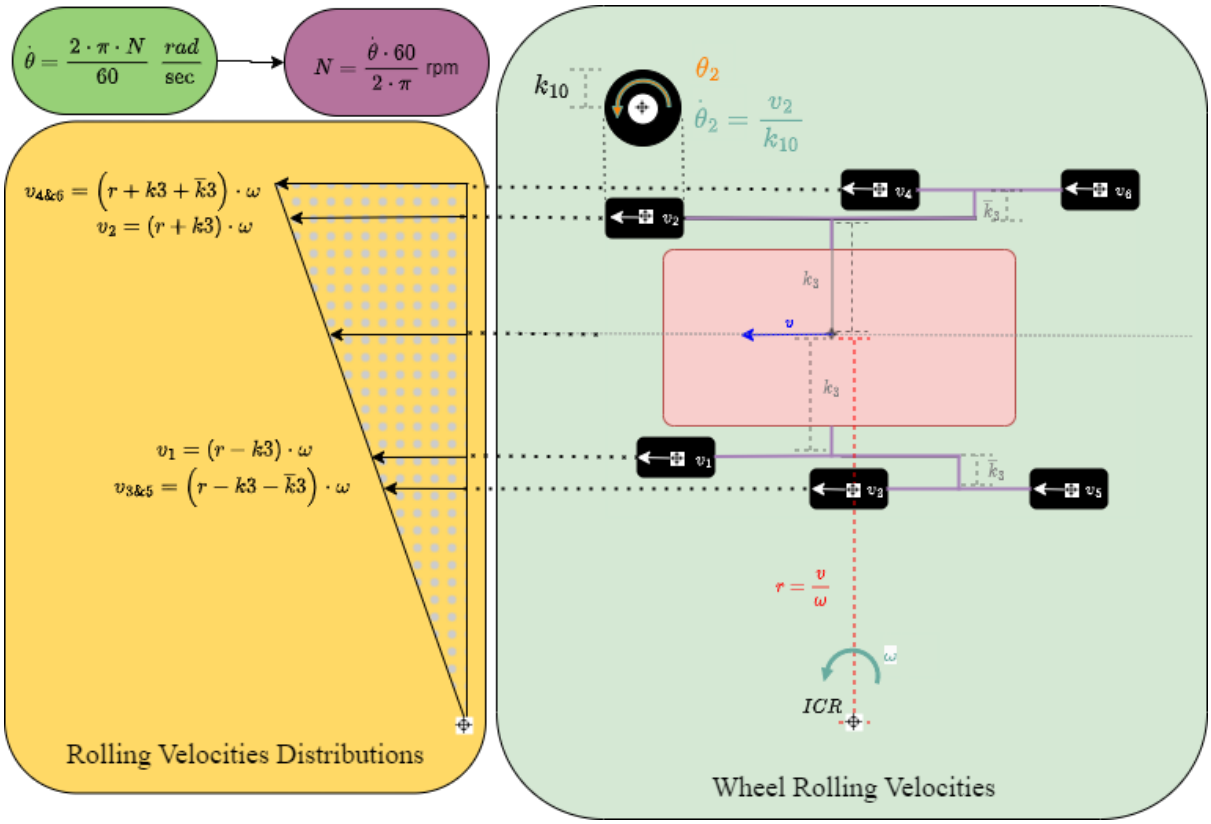

Figure 3- velocities

The provided Python code defines a class called **DiffrentialRover** that represents a differential rover and provides methods to calculate wheel rolling velocities based on the rover's input velocity and angular velocity.

Class Attributes

input (dotdict)

- **V** (float): The linear velocity of the rover in meters per second.
- **omega** (float): The angular velocity of the rover in radians per second.

dims (dotdict)

This attribute holds the geometric parameters of the rover.

- **k1** (float): Vertical offset between the rover reference point R and the differential point D in meters.
- **k2** (float): Forward offset between points R and D in meters.
- **k3** (float): Horizontal offset between the differential point D and the wheels/l1 in meters.

- **k3bar** (float): Horizontal offset between the differential point D and the wheels (additional offset) in meters.
- **k4** (float): Distance from the differential point D to the steering axis of the front wheels (l6) in meters.
- **k5** (float): Height of the differential point D from the wheel axles (l7) in meters.
- **k6** (float): Length of the link from the rocker joint to the bogie joint (l3) in meters.
- **k7f** (float): Length from the bogie joint to the forward bogie (l4) in meters.
- **k7r** (float): Length from the bogie joint to the rear bogie (l8) in meters.
- **k7** (float): Total length from the bogie joint to the forward/rear bogie (l4 + l8) in meters.
- **k8** (float): Height of the bogie joint from the wheel axles (l5) in meters.
- **k9** (float): Angle of the link between the rocker and bogie joints (l2) in radians.
- **l2** (float): Length of the link from the rocker joint to the differential point D in meters.
- **k10** (float): Wheel radius in meters.

rot_radii (dotdict)

This attribute holds the calculated rotation radii of the rover.

- **r** (float): Rotation radius of the whole vehicle.
- **r1** (float): Rotation radius of wheel 1.
- **r35** (float): Rotation radius of wheels 3 and 5.
- **r2** (float): Rotation radius of wheel 2.
- **r46** (float): Rotation radius of wheels 4 and 6.

Class Methods

calc_rotation_radii()

This method calculates the rotation radii of the rover based on the input velocity and angular velocity. It updates the **rot_radii** attribute with the calculated values.

whl_rolling_vel()

This method calculates the wheel rolling velocities of the rover based on the input velocity and angular velocity. It returns a list **theta_dot** containing the rolling velocities for each wheel.

If the angular velocity is set to 0 and the linear velocity is not 0, the method assumes the rover is moving in a straight line and calculates the rolling velocity for each wheel accordingly.

If both the angular velocity and linear velocity are set to 0, the method returns a list of zeros to indicate that the rover is not moving.

If the calculations are successful, the method returns the **theta_dot** list containing the rolling velocities for each wheel.

Example Usage

An instance of the **DiffrentialRover** class is created, and the input velocity and angular velocity are set to