Autonomous Robot Navigation Firmware

1.0.0

Generated by Doxygen 1.11.0

Chapter 1

Autonomous Robot Navigation Firmware

This project is designed to control an autonomous robot, enabling it to navigate towards a target position. It utilizes PID control for precise movement and implements various algorithms for path planning and drive control.

1.1 Features

- **PID Controller**: Utilizes Proportional-Integral-Derivative (PID) control to manage the robot's speed and direction accurately.
- **Serial Communication**: Communicates with external devices or computers through serial communication, allowing for dynamic target position updates.
- **Odometry Calculations**: Estimates the robot's position and orientation based on encoder feedback, enabling accurate movement control.
- **Vehicle Control**: Controls the robot's motors to achieve the desired velocities and directions for navigation.
- **Circle Tracking**: Includes an algorithm for circle tracking and path planning, enabling the robot to calculate the desired turning radius and wheel speeds to get from the current position to the target position.

1.2 Components

- **vec Struct**: Represents a 2D vector with `x` and `y` components, used for positions and velocities.
- **Setup Function**: Initializes serial communication, motor, and encoder pins, and attaches interrupt routines for encoders.
- **Main Loop**: Continuously reads serial data for new target positions, calculates desired velocities using PID control, and sends commands to the motors.
- **Interrupt Service Routines (ISRs)**: Reads encoder positions to provide feedback for the control system.
- **Utility Functions**:
  - `recvWithStartEndMarkers()`: Reads incoming serial data with start and end markers.
  - `parseDataPID()`: Parses the received data into target positions, modes, and reset commands.
  - `lowPassFilter()`: Applies a low-pass filter to smooth out the control signals.
  - `boundAngle()`: Normalizes angles to the range [0, 2) for consistent calculations.

1.3 Setup

1. **Hardware Setup**: Connect motors, encoders, and any other necessary hardware to the microcontroller according to the defined pin assignments in the code.
2. **Software Setup**: Upload the provided code to the microcontroller. Ensure any external device intended to communicate with the robot is set up to send data in the correct format.

1.4 Usage

- To command the robot, send data in the format `<target_x, target_y, mode, resetEnc>` through serial communication.
  - `target_x` and `target_y` are the coordinates of the target position in centimeters.
  - `mode` can be `0` for normal operation, `1` for off, or `2` for brake.
  - `resetEnc` resets the encoder positions when changed from `0` to `1`.

1.5 Note

This project is designed for the 2024 QUT DRC. It provides a basic framework for autonomous robot navigation but may require adjustments and improvements for specific applications or more complex environments.

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Topic Documentation

5.1 Serial Communication

Functions

- void recvWithStartEndMarkers ()

  Receives a string of characters via Serial until a specific end marker is found.
  This function listens for characters coming in via Serial communication, assembling them into a string. The string assembly starts when a predefined start marker character is detected and ends when an end marker character is received. The assembled string is stored in `receivedChars` and is made available once the end marker is detected. This function uses static variables to maintain state between calls.
- void parseDataPID ()

  Parses the received data into its individual parts.
  This function uses the `strtok()` function to split the received string into its individual parts. The parts are then converted to integers and stored in the `dataRX[]` array.

Variables

- const byte numChars = 32

  number of characters in the array
- char receivedChars [numChars]

  array to store received data
- char tempChars [numChars]

  temporary array for use when parsing
- int dataRX [4] = {0,0,0,0}

  array to store parsed data
- bool newData = false

  flag to indicate if new data is available
- int prevResetEnc = 0

  previous reset encoder value

5.1.1 Detailed Description

5.1.2 Function Documentation

5.1.2.1 parseDataPID()

```
void parseDataPID ()
```

Parses the received data into its individual parts.

This function uses the `strtok()` function to split the received string into its individual parts. The parts are then converted to integers and stored in the `dataRX[]` array.

Note

This function assumes that the received string is in the format: "<int1,int2,int3,int4>"

Global Variables:

- `receivedChars[]`: The array containing the received string.
- `tempChars[]`: A temporary array used for parsing.
- `dataRX[]`: The array where the parsed data is stored.
- `newData`: A boolean flag indicating if new data is available.

Static Variables:

- `strtokIndx`: An index used by `strtok()` to keep track of the current position in the string.

See also

strtok()

5.1.2.2 recvWithStartEndMarkers()

```
void recvWithStartEndMarkers ()
```

Receives a string of characters via Serial until a specific end marker is found.

This function listens for characters coming in via Serial communication, assembling them into a string. The string assembly starts when a predefined start marker character is detected and ends when an end marker character is received. The assembled string is stored in `receivedChars` and is made available once the end marker is detected. This function uses static variables to maintain state between calls.

Note

This function relies on global variables: `newData`, a boolean indicating if new data has been fully received, and `receivedChars`, an array where the assembled string is stored. It also uses `numChars` to prevent buffer overflow, ensuring that the assembled string does not exceed the size of `receivedChars`.

The function is designed to be called repeatedly; it processes one character per call until the end marker is detected.

Warning

This function assumes that `Serial` has been initialized and is ready for reading.

Global Variables:

- `newData`: Set to true when a complete string is received, until it is reset externally.
- `receivedChars[]`: Array where the received string is stored.
- `numChars`: The maximum size of `receivedChars[]`.

Static Variables:

- `recvInProgress`: Tracks whether the reception of a string is currently in progress.
- `ndx`: The current index in `receivedChars[]` where the next character will be stored.

See also

Serial.read()

5.1.3 Variable Documentation

5.1.3.1 dataRX

```
int dataRX[4] = {0,0,0,0}
```

array to store parsed data

5.1.3.2 newData

```
bool newData = false
```

flag to indicate if new data is available

5.1.3.3 numChars

```
const byte numChars = 32
```

number of characters in the array

5.1.3.4 prevResetEnc

```
int prevResetEnc = 0
```

previous reset encoder value

5.1.3.5 receivedChars

```
char receivedChars[numChars]
```

array to store received data

5.1.3.6 tempChars

```
char tempChars[numChars]
```

temporary array for use when parsing

5.2 Vehicle

Functions

- void drive (int data[ ])

  Controls the driving mechanism of a robot based on input commands.
  This function interprets an array of integers to control the driving mechanism of a robot, adjusting the speed and direction of its motors. The input array contains values for left and right motor PWM (Pulse Width Modulation) signals and a mode selector. The function supports three modes: normal operation, off, and brake. In normal operation, the PWM values directly control the motor speeds, allowing for forward and reverse motion. The off mode stops all motor activity, and the brake mode actively halts the motors, which can cause voltage spikes and should be used cautiously.

Variables

- const float WHEEL_RATIO = 555

  encoder ticks per metre (∼0.00188496m/t)
- const float WHEELBASE = 0.36

  distance between the two wheels in metres
- const float HALF_VEHICLE_WIDTH = WHEELBASE / 2

  half the wheelbase, in metres
- const float MAX_SPEED = 1300 / WHEEL_RATIO

  max speed in ticks per second
- const int MAX_PWM = 250

  max PWM value for the motors
- const int MIN_SPEED = 4

  minimum speed to move the robot in ticks per second (4 t/s ∼= 8mm/s)

5.2.1 Detailed Description

5.2.2 Function Documentation

5.2.2.1 drive()

```
void drive (
            int data[])
```

Controls the driving mechanism of a robot based on input commands.

This function interprets an array of integers to control the driving mechanism of a robot, adjusting the speed and direction of its motors. The input array contains values for left and right motor PWM (Pulse Width Modulation) signals and a mode selector. The function supports three modes: normal operation, off, and brake. In normal operation, the PWM values directly control the motor speeds, allowing for forward and reverse motion. The off mode stops all motor activity, and the brake mode actively halts the motors, which can cause voltage spikes and should be used cautiously.

Warning

> The brake mode can cause voltage spikes and should be used with caution to avoid damaging the motors or other components.

The function uses `digitalWriteFast` to set the direction of the motors and `analogWrite` to control the speed. It ensures that the motors are stopped if both PWM values are zero, regardless of the selected mode. This function is designed for robots with differential drive systems, where the speed and direction of each side (left and right) can be controlled independently.

Parameters

data	An array of integers where: • data[0] is the PWM value for the left motor (positive for forward, negative for reverse | range: -255 to 255), • data[1] is the PWM value for the right motor (positive for forward, negative for reverse | range: -255 to 255), • data[2] is the mode selector (0 for normal operation, 1 for off, 2 for brake).

5.2.3 Variable Documentation

5.2.3.1 HALF_VEHICLE_WIDTH

```
const float HALF_VEHICLE_WIDTH = WHEELBASE / 2
```

half the wheelbase, in metres

5.2.3.2 MAX_PWM

```
const int MAX_PWM = 250
```

max PWM value for the motors

5.2.3.3 MAX_SPEED

```
const float MAX_SPEED = 1300 / WHEEL_RATIO
```

max speed in ticks per second

5.2.3.4 MIN_SPEED

```
const int MIN_SPEED = 4
```

minimum speed to move the robot in ticks per second (4 t/s ∼= 8mm/s)

5.2.3.5 WHEEL_RATIO

```
const float WHEEL_RATIO = 555
```

encoder ticks per metre (∼0.00188496m/t)

5.2.3.6 WHEELBASE

```
const float WHEELBASE = 0.36
```

distance between the two wheels in metres

5.3 Odometry Calculations

Classes

- struct vec

  The struct `vec` represents a 2D vector with `x` and `y` components.

Functions

- void readEncoderR ()

  Interrupt service routine for reading the right encoder.
- void readEncoderL ()

  Interrupt service routine for reading the left encoder.
- float boundAngle (float angle)

  Normalizes an angle to the range [0, 2).
  This function ensures that any given angle in radians is converted to an equivalent angle within the range [0, 2). It is useful for angle comparisons or operations where the angle needs to be within a single, continuous 360-degree cycle. The normalization is done by adding or subtracting 2 from the angle until it falls within the desired range.
- void updateOdometry ()

  Updates the odometry information of the robot.
  This function calculates the new position and heading of the robot based on the encoder readings. It updates the robot's current position (currPos), heading, and the change in position (dPos) and heading (deltaHeading) since the last update. The calculations differentiate between straight movement and turning by checking the difference in encoder values for the left and right wheels. For straight movement, the position update is straightforward. For turning, it calculates the turning radius and adjusts the position and heading accordingly. The function ensures that the heading is always normalized to the range [0, 2) using the boundAngle function.
- void sendOdometry ()

  sends the odometry information to the serial monitor.

Variables

- long int encoderPosition [2] = {0, 0}

  Stores last encoder positions L and R.
- double leftDelta = 0

  change in left encoder ticks
- double rightDelta = 0

  change in right encoder ticks
- double newX = 0

  new x position
- double newY = 0

  new y position
- double x = 0

  current x position
- double y = 0

  current y position
- double dx = 0

  change in x position
- double dy = 0

  change in y position
- float dist = 0

  average distance travelled
- long prevLeft = 0

  previous left encoder position
- long prevRight = 0

  previous right encoder position
- unsigned long prevTime = 0

  previous time since last odometry update
- unsigned long currentTime = 0

  current time
- float heading = 0

  current heading in radians
- float newHeading = 0

  new heading in radians
- float deltaHeading = 0

  change in heading
- vec currPos = {0,0}

  current position of the robot
- vec newPos = {0,0}

  new position of the robot
- vec dPos = {0,0}

  change in position of the robot
- vec targPos = {0,0}

  target position of the robot
- vec targVel = {0,0}

  target velocity of the robot, where the x component is the left wheel and the y component is the right wheel
- vec headingVec = {1,0}

  heading vector of the robot
- double current_L = 0

  current left side velocity IN TICKS PER SECOND => 1m/s = 531 ticks/s
- double current_R = 0

current right side velocity IN TICKS PER SECOND => 1m/s = 531 ticks/s

- double prev_Vel_L = 0

  previous left side velocity
- double prev_Vel_R = 0

  previous right side velocity
- double vec::x
- double vec::y

5.3.1 Detailed Description

5.3.2 Function Documentation

5.3.2.1 boundAngle()

```
float boundAngle (
            float angle)
```

Normalizes an angle to the range [0, 2).

This function ensures that any given angle in radians is converted to an equivalent angle within the range [0, 2). It is useful for angle comparisons or operations where the angle needs to be within a single, continuous 360-degree cycle. The normalization is done by adding or subtracting 2 from the angle until it falls within the desired range.

Parameters

angle	The input angle in radians to be normalized.

Returns

The normalized angle in radians, guaranteed to be between 0 (inclusive) and 2 (exclusive).

5.3.2.2 readEncoderL()

```
void readEncoderL ()
```

Interrupt service routine for reading the left encoder.

5.3.2.3 readEncoderR()

```
void readEncoderR ()
```

Interrupt service routine for reading the right encoder.

5.3.2.4 sendOdometry()

```
void sendOdometry ()
```

sends the odometry information to the serial monitor.

5.3.2.5 updateOdometry()

```
void updateOdometry ()
```

Updates the odometry information of the robot.

This function calculates the new position and heading of the robot based on the encoder readings. It updates the robot's current position (currPos), heading, and the change in position (dPos) and heading (deltaHeading) since the last update. The calculations differentiate between straight movement and turning by checking the difference in encoder values for the left and right wheels. For straight movement, the position update is straightforward. For turning, it calculates the turning radius and adjusts the position and heading accordingly. The function ensures that the heading is always normalized to the range [0, 2) using the boundAngle function.

Note

This function uses the `WHEEL_RATIO` constant to convert encoder ticks to metres and the `WHEELBASE` constant to determine the distance between the two wheels. It also uses the `headingVec` vector to represent the heading

See also

boundAngle()

5.3.3 Variable Documentation

5.3.3.1 current_L

```
double current_L = 0
```

current left side velocity IN TICKS PER SECOND => 1m/s = 531 ticks/s

5.3.3.2 current_R

```
double current_R = 0
```

current right side velocity IN TICKS PER SECOND => 1m/s = 531 ticks/s

5.3.3.3 currentTime

```
unsigned long currentTime = 0
```

current time

5.3.3.4 currPos

```
vec currPos = {0,0}
```

current position of the robot

5.3.3.5 deltaHeading

```
float deltaHeading = 0
```

change in heading

5.3.3.6 dist

```
float dist = 0
```

average distance travelled

5.3.3.7 dPos

```
vec dPos = {0,0}
```

change in position of the robot

5.3.3.8 dx

```
double dx = 0
```

change in x position

5.3.3.9 dy

```
double dy = 0
```

change in y position

5.3.3.10 encoderPosition

```
long int encoderPosition[2] = {0, 0}
```

Stores last encoder positions L and R.

5.3.3.11 heading

```
float heading = 0
```

current heading in radians

5.3.3.12 headingVec

```
vec headingVec = {1,0}
```

heading vector of the robot

5.3.3.13 leftDelta

```
double leftDelta = 0
```

change in left encoder ticks

5.3.3.14 newHeading

```
float newHeading = 0
```

new heading in radians

5.3.3.15 newPos

```
vec newPos = {0,0}
```

new position of the robot

5.3.3.16 newX

```
double newX = 0
```

new x position

5.3.3.17 newY

```
double newY = 0
```

new y position

5.3.3.18 prev_Vel_L

```
double prev_Vel_L = 0
```

previous left side velocity

5.3.3.19 prev_Vel_R

```
double prev_Vel_R = 0
```

previous right side velocity

5.3.3.20 prevLeft

```
long prevLeft = 0
```

previous left encoder position

5.3.3.21 prevRight

```
long prevRight = 0
```

previous right encoder position

5.3.3.22 prevTime

```
unsigned long prevTime = 0
```

previous time since last odometry update

5.3.3.23 rightDelta

```
double rightDelta = 0
```

change in right encoder ticks

5.3.3.24 targPos

```
vec targPos = {0,0}
```

target position of the robot

5.3.3.25 targVel

```
vec targVel = {0,0}
```

target velocity of the robot, where the x component is the left wheel and the y component is the right wheel

5.3.3.26 x [1/2]

```
double vec::x
```

The property `x` in the `vec` struct represents the x-coordinate of a 2D vector.

5.3.3.27 x [2/2]

```
double x = 0
```

current x position

5.3.3.28 y [1/2]

```
double vec::y
```

The `y` property in the `vec` struct represents the y-coordinate of a 2D vector.

5.3.3.29 y [2/2]

```
double y = 0
```

current y position

5.4 PID Controller

Files

- file lut.h

  Defines the lookup table for feedforward of PID.
  This header file contains the definition of a lookup table (LUT) used for the feedforward component of a PID controller. The LUT provides predefined output values for given input conditions, enhancing the PID controller's response by compensating for known system behaviors in advance.

Classes

- struct motor

  Structure to represent motor parameters for PID feedforward.
  This structure is used within the lookup table to map specific motor speed (in ticks per second) to a corresponding PWM signal value and operational mode. It is a key component in implementing feedforward control in PID controllers for motors.

Functions

- float lowPassFilter (float input, float prev, float alpha)

  Applies a low-pass filter to the input signal.
  This function implements a simple low-pass filter, which is useful for smoothing out high-frequency noise in a signal. The filter is applied to the current input value based on the previous filtered value and a smoothing factor (alpha).
- float clampSpeed (float target, float input)

  Clamps the input speed based on the target speed direction and limits.
  This function adjusts the input speed to ensure it is in the correct direction relative to the target speed and within the allowable range of speed values. If the target speed is positive, the input speed is clamped to a minimum of 0 and a maximum of MAX_PWM. For a negative target speed, the input speed is clamped to a minimum of -MAX_PWM and a maximum of 0. This ensures the input speed does not exceed the maximum allowed speed (MAX_PWM) and is in the correct direction (positive or negative) as the target speed.
- motor lookupFF (float desiredSpeed)

  Finds the motor configuration closest to the desired speed.
  This function searches through a lookup table (lut) of motor configurations to find the one whose speed is closest to a desired speed. It iterates through the lut, calculating the absolute difference between the desired speed and each motor speed in the lut. The motor configuration with the smallest difference is considered the closest match and is returned.
- void PID (float desired_L, float desired_R)

  Implements a PID controller for velocity control.
  This function calculates the PID control output based on the current and desired velocities for the left and right wheels. It computes the error between the desired and current velocities, updates the integral term, and calculates the derivative term. The PID control output is then determined using the proportional, integral, and derivative gains (Kp, Ki, Kd) along with a bias term and feedforward lookup table. The output is clamped to ensure it does not exceed the maximum speed and is set to 0 if the desired speed is below a minimum threshold.
- void printVel ()

  prints the current and desired velocities for the left and right wheels.
- void printPID ()

  prints the current and desired velocities for the left and right wheels along with PID values.

Variables

- const double Kp = 0.9

  Proportional constant.
- const double Ki = 0.8

  Integral constant.
- const double Kd = 0

  Derivative constant | Keep this 0.
- const double bias = 0

  Bias constant | Keep this 0.
- const long int PID_TIME = 20000

  s | PID Controller update time
- const float VELOCITY_ALPHA = 0.1

  alpha value for low pass filter for measured velocity
- const float SETPOINT_ALPHA = 0.1

  alpha value for low pass filter for target setpoint velocity
- const float DEST_THRESHOLD = 0.15

  threshold for destination reached | in metres
- float dt

  time difference for PID update
- double e_prev_L = 0

  Previous left speed error.
- double e_prev_R = 0

  Previous right speed error.
- double integral_L = 0

  Integral term for left wheel.
- double integral_R = 0

  Integral term for right wheel.
- float output_L = 0

  Output for left wheel.
- float output_R = 0

  Output for right wheel.
- long long int prev_L = 0

  Previous left encoder position.
- long long int prev_R = 0

  Previous right encoder position.
- unsigned long prevTimePID = 0

  Previous time for PID update.
- unsigned long currTimePID = 0

  Current time for PID update.
- int driveData [3] = {0,0,0}

  Array to store drive data for the motors.
- motor lut [ ]

  Lookup table array for PID feedforward control.
  This array of `motor` structures defines the lookup table for the feedforward component of the PID controller. Each entry maps a specific motor speed to a PWM value and operational mode, allowing for immediate adjustment based on the current speed requirement.

5.4.1 Detailed Description

5.4.2 Function Documentation

5.4.2.1 clampSpeed()

```
float clampSpeed (
            float target,
            float input)
```

Clamps the input speed based on the target speed direction and limits.

This function adjusts the input speed to ensure it is in the correct direction relative to the target speed and within the allowable range of speed values. If the target speed is positive, the input speed is clamped to a minimum of 0 and a maximum of MAX_PWM. For a negative target speed, the input speed is clamped to a minimum of -MAX_PWM and a maximum of 0. This ensures the input speed does not exceed the maximum allowed speed (MAX_PWM) and is in the correct direction (positive or negative) as the target speed.

Parameters

target	The target speed, which determines the allowed direction of the input speed.
input	The current input speed to be clamped based on the target speed and MAX_PWM.

Returns

The clamped speed, adjusted to be within the allowable range and direction.

5.4.2.2 lookupFF()

```
motor lookupFF (
            float desiredSpeed)
```

Finds the motor configuration closest to the desired speed.

This function searches through a lookup table (lut) of motor configurations to find the one whose speed is closest to a desired speed. It iterates through the lut, calculating the absolute difference between the desired speed and each motor speed in the lut. The motor configuration with the smallest difference is considered the closest match and is returned.

Note

The LUT array must be defined and contain motor structures with speed values.

This function assumes that the LUT is sorted in ascending order of speed values.

Parameters

desiredSpeed	The target speed to match.

Returns

The motor configuration from the lut that is closest to the desired speed.

5.4.2.3 lowPassFilter()

```
float lowPassFilter (
            float input,
            float prev,
            float alpha)
```

Applies a low-pass filter to the input signal.

This function implements a simple low-pass filter, which is useful for smoothing out high-frequency noise in a signal. The filter is applied to the current input value based on the previous filtered value and a smoothing factor (alpha).

Parameters

input	The current raw input value to be filtered.
prev	The previous filtered value.
alpha	The smoothing factor used in the filter, between 0 and 1. A higher alpha value gives more weight to the current input, while a lower alpha value gives more weight to the previous values, resulting in stronger smoothing.

Returns

The filtered value.

5.4.2.4 PID()

```
void PID (
            float desired_L,
            float desired_R)
```

Implements a PID controller for velocity control.

This function calculates the PID control output based on the current and desired velocities for the left and right wheels. It computes the error between the desired and current velocities, updates the integral term, and calculates the derivative term. The PID control output is then determined using the proportional, integral, and derivative gains (Kp, Ki, Kd) along with a bias term and feedforward lookup table. The output is clamped to ensure it does not exceed the maximum speed and is set to 0 if the desired speed is below a minimum threshold.

Note

This function requires the `lookupFF()` function to find the feedforward values for the desired speeds.

The PID constants (Kp, Ki, Kd) and bias term can be adjusted to tune the controller's performance.

The function uses a low-pass filter to smooth the measured velocities and the desired setpoints.

Parameters

desired_L	The desired velocity for the left wheel.
desired_R	The desired velocity for the right wheel.

See also

lookupFF()

lowPassFilter()

5.4.2.5 printPID()

```
void printPID ()
```

prints the current and desired velocities for the left and right wheels along with PID values.

5.4.2.6 printVel()

```
void printVel ()
```

prints the current and desired velocities for the left and right wheels.

5.4.3 Variable Documentation

5.4.3.1 bias

```
const double bias = 0
```

Bias constant | Keep this 0.

5.4.3.2 currTimePID

```
unsigned long currTimePID = 0
```

Current time for PID update.

5.4.3.3 DEST_THRESHOLD

```
const float DEST_THRESHOLD = 0.15
```

threshold for destination reached | in metres

5.4.3.4 driveData

```
int driveData[3] = {0,0,0}
```

Array to store drive data for the motors.

5.4.3.5 dt

```
float dt
```

time difference for PID update

5.4.3.6 e_prev_L

```
double e_prev_L = 0
```

Previous left speed error.

5.4.3.7 e_prev_R

```
double e_prev_R = 0
```

Previous right speed error.

5.4.3.8 integral_L

```
double integral_L = 0
```

Integral term for left wheel.

5.4.3.9 integral_R

```
double integral_R = 0
```

Integral term for right wheel.

5.4.3.10 Kd

```
const double Kd = 0
```

Derivative constant | Keep this 0.

5.4.3.11 Ki

```
const double Ki = 0.8
```

Integral constant.

5.4.3.12 Kp

```
const double Kp = 0.9
```

Proportional constant.

5.4.3.13 lut

```
motor lut[]
```

Lookup table array for PID feedforward control.

This array of `motor` structures defines the lookup table for the feedforward component of the PID controller. Each entry maps a specific motor speed to a PWM value and operational mode, allowing for immediate adjustment based on the current speed requirement.

5.4.3.14 output_L

```
float output_L = 0
```

Output for left wheel.

5.4.3.15 output_R

```
float output_R = 0
```

Output for right wheel.

5.4.3.16 PID_TIME

```
const long int PID_TIME = 20000
```

s | PID Controller update time

5.4.3.17 prev_L

```
long long int prev_L = 0
```

Previous left encoder position.

5.4.3.18 prev_R

```
long long int prev_R = 0
```

Previous right encoder position.

5.4.3.19 prevTimePID

```
unsigned long prevTimePID = 0
```

Previous time for PID update.

5.4.3.20 SETPOINT_ALPHA

```
const float SETPOINT_ALPHA = 0.1
```

alpha value for low pass filter for target setpoint velocity

5.4.3.21 VELOCITY_ALPHA

```
const float VELOCITY_ALPHA = 0.1
```

alpha value for low pass filter for measured velocity

5.5 Circle Tracking

Functions

- vec circle_track (vec s, vec t, vec h, float vel)

  Calculates motor speeds for tracking a circular path towards a target point.
  This function computes the speeds for the left and right motors of a vehicle to make it follow a circular path towards a target point. It uses the current position (s), target position (t), and heading vector (h) to determine the direction and magnitude of the required turn. The function calculates the distance to the target and uses it along with the desired velocity (vel) to compute the appropriate motor speeds. If the target is within a small tolerance, it stops the motors. Otherwise, it determines whether to turn left or right based on the cross product of the heading and the direction to the target. It then calculates the turning radius and adjusts the motor speeds accordingly to achieve the desired circular path. The function ensures that the vehicle turns in the most efficient manner to reach the target point.
- void sendVecs ()

  sends the current position, target position, and target velocity to the serial monitor.

Variables

- long int dtCircle

  time difference for circle tracking
- long int prevCircleTime = 0

  previous time for circle tracking
- const long int CIRCLE_TIME = 500000

  s | Circle tracking update time
- const int CIRCLE_LOOPS = CIRCLE_TIME/PID_TIME

  number of loops for circle tracking
- float AVERAGE_TARGET_VELOCITY = 0.5

  m/s | average target velocity for circle tracking
- long int loop_no = 0

  loop number for circle tracking
- int des_L = 0

  calculated desired left speed
- int des_R = 0

  calculated desired right speed

5.5.1 Detailed Description

5.5.2 Function Documentation

5.5.2.1 circle_track()

```
vec circle_track (
            vec s,
            vec t,
            vec h,
            float vel)
```

Calculates motor speeds for tracking a circular path towards a target point.

This function computes the speeds for the left and right motors of a vehicle to make it follow a circular path towards a target point. It uses the current position (s), target position (t), and heading vector (h) to determine the direction and magnitude of the required turn. The function calculates the distance to the target and uses it along with the desired velocity (vel) to compute the appropriate motor speeds. If the target is within a small tolerance, it stops the motors. Otherwise, it determines whether to turn left or right based on the cross product of the heading and the direction to the target. It then calculates the turning radius and adjusts the motor speeds accordingly to achieve the desired circular path. The function ensures that the vehicle turns in the most efficient manner to reach the target point.

Note

This function uses the `HALF_VEHICLE_WIDTH` constant to determine the turning radius based on the vehicle's dimensions.

Parameters

s	The current position of the vehicle as a vector.
t	The target position as a vector.
h	The current heading of the vehicle as a vector.
vel	The desired velocity towards the target.

Returns

A vector representing the speeds for the left (x component) and right (y component) motors.

5.5.2.2 sendVecs()

```
void sendVecs ()
```

sends the current position, target position, and target velocity to the serial monitor.

5.5.3 Variable Documentation

5.5.3.1 AVERAGE_TARGET_VELOCITY

```
float AVERAGE_TARGET_VELOCITY = 0.5
```

m/s | average target velocity for circle tracking

5.5.3.2 CIRCLE_LOOPS

```
const int CIRCLE_LOOPS = CIRCLE_TIME/PID_TIME
```

number of loops for circle tracking

5.5.3.3 CIRCLE_TIME

```
const long int CIRCLE_TIME = 500000
```

s | Circle tracking update time

5.5.3.4 des_L

```
int des_L = 0
```

calculated desired left speed

5.5.3.5 des_R

```
int des_R = 0
```

calculated desired right speed

5.5.3.6 dtCircle

```
long int dtCircle
```

time difference for circle tracking

5.5.3.7 loop_no

```
long int loop_no = 0
```

loop number for circle tracking

5.5.3.8 prevCircleTime

```
long int prevCircleTime = 0
```

previous time for circle tracking

Chapter 6

Class Documentation

6.1 motor Struct Reference

Structure to represent motor parameters for PID feedforward.

This structure is used within the lookup table to map specific motor speed (in ticks per second) to a corresponding PWM signal value and operational mode. It is a key component in implementing feedforward control in PID controllers for motors.

`#include <lut.h>`

Public Attributes

- int speed
- int pwm
- int mode

6.1.1 Detailed Description

Structure to represent motor parameters for PID feedforward.

This structure is used within the lookup table to map specific motor speed (in ticks per second) to a corresponding PWM signal value and operational mode. It is a key component in implementing feedforward control in PID controllers for motors.

Parameters

speed	Motor speed in ticks per second.
pwm	PWM signal value (0-255) for speed control.
mode	Operational mode of the motor (0 = normal, 1 = coast).

6.1.2 Member Data Documentation

6.1.2.1 mode

`int motor::mode`

6.1.2.2 pwm

```
int motor::pwm
```

6.1.2.3 speed

```
int motor::speed
```

The documentation for this struct was generated from the following file:

- lut.h

6.2 vec Struct Reference

The struct `vec` represents a 2D vector with `x` and `y` components.

Public Attributes

- double x
- double y

6.2.1 Detailed Description

The struct `vec` represents a 2D vector with `x` and `y` components.

The documentation for this struct was generated from the following file:

- controller.cpp

Chapter 7

File Documentation

7.1 controller.cpp File Reference

```
#include <math.h>
#include <digitalWriteFast.h>
#include "lut.h"
```

Classes

- struct vec

  The struct `vec` represents a 2D vector with `x` and `y` components.

Functions

- void setup ()

  Main setup function.
- void loop ()

  Main loop function.
- void recvWithStartEndMarkers ()

  Receives a string of characters via Serial until a specific end marker is found.
  This function listens for characters coming in via Serial communication, assembling them into a string. The string assembly starts when a predefined start marker character is detected and ends when an end marker character is received. The assembled string is stored in `receivedChars` and is made available once the end marker is detected. This function uses static variables to maintain state between calls.
- void parseDataPID ()

  Parses the received data into its individual parts.
  This function uses the `strtok()` function to split the received string into its individual parts. The parts are then converted to integers and stored in the `dataRX[]` array.
- void sendEncoder ()

  sends raw encoder data to the serial monitor
- void readEncoderR ()

  Interrupt service routine for reading the right encoder.
- void readEncoderL ()

  Interrupt service routine for reading the left encoder.
- float boundAngle (float angle)

Normalizes an angle to the range [0, 2).
This function ensures that any given angle in radians is converted to an equivalent angle within the range [0, 2). It is useful for angle comparisons or operations where the angle needs to be within a single, continuous 360-degree cycle. The normalization is done by adding or subtracting 2 from the angle until it falls within the desired range.

- float lowPassFilter (float input, float prev, float alpha)

  Applies a low-pass filter to the input signal.
  This function implements a simple low-pass filter, which is useful for smoothing out high-frequency noise in a signal. The filter is applied to the current input value based on the previous filtered value and a smoothing factor (alpha).

- float clampSpeed (float target, float input)

  Clamps the input speed based on the target speed direction and limits.
  This function adjusts the input speed to ensure it is in the correct direction relative to the target speed and within the allowable range of speed values. If the target speed is positive, the input speed is clamped to a minimum of 0 and a maximum of MAX_PWM. For a negative target speed, the input speed is clamped to a minimum of -MAX_PWM and a maximum of 0. This ensures the input speed does not exceed the maximum allowed speed (MAX_PWM) and is in the correct direction (positive or negative) as the target speed.

- motor lookupFF (float desiredSpeed)

  Finds the motor configuration closest to the desired speed.
  This function searches through a lookup table (lut) of motor configurations to find the one whose speed is closest to a desired speed. It iterates through the lut, calculating the absolute difference between the desired speed and each motor speed in the lut. The motor configuration with the smallest difference is considered the closest match and is returned.

- void PID (float desired_L, float desired_R)

  Implements a PID controller for velocity control.
  This function calculates the PID control output based on the current and desired velocities for the left and right wheels. It computes the error between the desired and current velocities, updates the integral term, and calculates the derivative term. The PID control output is then determined using the proportional, integral, and derivative gains (Kp, Ki, Kd) along with a bias term and feedforward lookup table. The output is clamped to ensure it does not exceed the maximum speed and is set to 0 if the desired speed is below a minimum threshold.

- void printVel ()

  prints the current and desired velocities for the left and right wheels.

- void printPID ()

  prints the current and desired velocities for the left and right wheels along with PID values.

- vec circle_track (vec s, vec t, vec h, float vel)

  Calculates motor speeds for tracking a circular path towards a target point.
  This function computes the speeds for the left and right motors of a vehicle to make it follow a circular path towards a target point. It uses the current position (s), target position (t), and heading vector (h) to determine the direction and magnitude of the required turn. The function calculates the distance to the target and uses it along with the desired velocity (vel) to compute the appropriate motor speeds. If the target is within a small tolerance, it stops the motors. Otherwise, it determines whether to turn left or right based on the cross product of the heading and the direction to the target. It then calculates the turning radius and adjusts the motor speeds accordingly to achieve the desired circular path. The function ensures that the vehicle turns in the most efficient manner to reach the target point.

- void updateOdometry ()

  Updates the odometry information of the robot.
  This function calculates the new position and heading of the robot based on the encoder readings. It updates the robot's current position (currPos), heading, and the change in position (dPos) and heading (deltaHeading) since the last update. The calculations differentiate between straight movement and turning by checking the difference in encoder values for the left and right wheels. For straight movement, the position update is straightforward. For turning, it calculates the turning radius and adjusts the position and heading accordingly. The function ensures that the heading is always normalized to the range [0, 2) using the boundAngle function.

- void sendOdometry ()

  sends the odometry information to the serial monitor.

- void sendVecs ()

  sends the current position, target position, and target velocity to the serial monitor.

- void drive (int data[])

  Controls the driving mechanism of a robot based on input commands.
  This function interprets an array of integers to control the driving mechanism of a robot, adjusting the speed and direction of its motors. The input array contains values for left and right motor PWM (Pulse Width Modulation) signals and a mode selector. The function supports three modes: normal operation, off, and brake. In normal operation, the

PWM values directly control the motor speeds, allowing for forward and reverse motion. The off mode stops all motor activity, and the brake mode actively halts the motors, which can cause voltage spikes and should be used cautiously.

Variables

- const int MotorRLEN = 6
- const int MotorRREN = 7
- const int MotorRLPWM = 8
- const int MotorRRPWM = 9
- const int MotorLLEN = 10
- const int MotorLREN = 11
- const int MotorLLPWM = 12
- const int MotorLRPWM = 13
- const int encoderRPinA = 2
- const int encoderRPinB = 4
- const int encoderLPinA = 3
- const int encoderLPinB = 5
- const byte numChars = 32

  number of characters in the array
- char receivedChars [numChars]

  array to store received data
- char tempChars [numChars]

  temporary array for use when parsing
- int dataRX [4] = {0,0,0,0}

  array to store parsed data
- bool newData = false

  flag to indicate if new data is available
- int prevResetEnc = 0

  previous reset encoder value
- const float WHEEL_RATIO = 555

  encoder ticks per metre (∼0.00188496m/t)
- const float WHEELBASE = 0.36

  distance between the two wheels in metres
- const float HALF_VEHICLE_WIDTH = WHEELBASE / 2

  half the wheelbase, in metres
- const float MAX_SPEED = 1300 / WHEEL_RATIO

  max speed in ticks per second
- const int MAX_PWM = 250

  max PWM value for the motors
- const int MIN_SPEED = 4

  minimum speed to move the robot in ticks per second (4 t/s ∼= 8mm/s)
- long int encoderPosition [2] = {0, 0}

  Stores last encoder positions L and R.
- double leftDelta = 0

  change in left encoder ticks
- double rightDelta = 0

  change in right encoder ticks
- double newX = 0

  new x position
- double newY = 0

new y position

- double x = 0

  current x position

- double y = 0

  current y position

- double dx = 0

  change in x position

- double dy = 0

  change in y position

- float dist = 0

  average distance travelled

- long prevLeft = 0

  previous left encoder position

- long prevRight = 0

  previous right encoder position

- unsigned long prevTime = 0

  previous time since last odometry update

- unsigned long currentTime = 0

  current time

- float heading = 0

  current heading in radians

- float newHeading = 0

  new heading in radians

- float deltaHeading = 0

  change in heading

- vec currPos = {0,0}

  current position of the robot

- vec newPos = {0,0}

  new position of the robot

- vec dPos = {0,0}

  change in position of the robot

- vec targPos = {0,0}

  target position of the robot

- vec targVel = {0,0}

  target velocity of the robot, where the x component is the left wheel and the y component is the right wheel

- vec headingVec = {1,0}

  heading vector of the robot

- double current_L = 0

  current left side velocity IN TICKS PER SECOND => 1m/s = 531 ticks/s

- double current_R = 0

  current right side velocity IN TICKS PER SECOND => 1m/s = 531 ticks/s

- double prev_Vel_L = 0

  previous left side velocity

- double prev_Vel_R = 0

  previous right side velocity

- const double Kp = 0.9

  Proportional constant.

- const double Ki = 0.8

  Integral constant.

- const double Kd = 0

  Derivative constant | Keep this 0.

- const double bias = 0

  Bias constant | Keep this 0.
- const long int PID_TIME = 20000

  s | PID Controller update time
- const float VELOCITY_ALPHA = 0.1

  alpha value for low pass filter for measured velocity
- const float SETPOINT_ALPHA = 0.1

  alpha value for low pass filter for target setpoint velocity
- const float DEST_THRESHOLD = 0.15

  threshold for destination reached | in metres
- float dt

  time difference for PID update
- double e_prev_L = 0

  Previous left speed error.
- double e_prev_R = 0

  Previous right speed error.
- double integral_L = 0

  Integral term for left wheel.
- double integral_R = 0

  Integral term for right wheel.
- float output_L = 0

  Output for left wheel.
- float output_R = 0

  Output for right wheel.
- long long int prev_L = 0

  Previous left encoder position.
- long long int prev_R = 0

  Previous right encoder position.
- unsigned long prevTimePID = 0

  Previous time for PID update.
- unsigned long currTimePID = 0

  Current time for PID update.
- int driveData [3] = {0,0,0}

  Array to store drive data for the motors.
- long int dtCircle

  time difference for circle tracking
- long int prevCircleTime = 0

  previous time for circle tracking
- const long int CIRCLE_TIME = 500000

  s | Circle tracking update time
- const int CIRCLE_LOOPS = CIRCLE_TIME/PID_TIME

  number of loops for circle tracking
- float AVERAGE_TARGET_VELOCITY = 0.5

  m/s | average target velocity for circle tracking
- long int loop_no = 0

  loop number for circle tracking
- int des_L = 0

  calculated desired left speed
- int des_R = 0

  calculated desired right speed

7.1.1 Function Documentation

7.1.1.1 loop()

```
void loop ()
```

Main loop function.

7.1.1.2 sendEncoder()

```
void sendEncoder ()
```

sends raw encoder data to the serial monitor

7.1.1.3 setup()

```
void setup ()
```

Main setup function.

7.1.2 Variable Documentation

7.1.2.1 encoderLPinA

```
const int encoderLPinA = 3
```

7.1.2.2 encoderLPinB

```
const int encoderLPinB = 5
```

7.1.2.3 encoderRPinA

```
const int encoderRPinA = 2
```

7.1.2.4 encoderRPinB

```
const int encoderRPinB = 4
```

7.1.2.5 MotorLLEN

```
const int MotorLLEN = 10
```

7.1.2.6 MotorLLPWM

```
const int MotorLLPWM = 12
```

7.1.2.7 MotorLREN

```
const int MotorLREN = 11
```

7.1.2.8 MotorLRPWM

```
const int MotorLRPWM = 13
```

7.1.2.9 MotorRLEN

```
const int MotorRLEN = 6
```

7.1.2.10 MotorRLPWM

```
const int MotorRLPWM = 8
```

7.1.2.11 MotorRREN

```
const int MotorRREN = 7
```

7.1.2.12 MotorRRPWM

```
const int MotorRRPWM = 9
```

7.2 lut.h File Reference

Defines the lookup table for feedforward of PID.

This header file contains the definition of a lookup table (LUT) used for the feedforward component of a PID controller. The LUT provides predefined output values for given input conditions, enhancing the PID controller's response by compensating for known system behaviors in advance.

Classes

- struct motor

  Structure to represent motor parameters for PID feedforward.
  This structure is used within the lookup table to map specific motor speed (in ticks per second) to a corresponding PWM signal value and operational mode. It is a key component in implementing feedforward control in PID controllers for motors.

Variables

- motor lut [ ]

  Lookup table array for PID feedforward control.
  This array of `motor` structures defines the lookup table for the feedforward component of the PID controller. Each entry maps a specific motor speed to a PWM value and operational mode, allowing for immediate adjustment based on the current speed requirement.

7.2.1 Detailed Description

Defines the lookup table for feedforward of PID.

This header file contains the definition of a lookup table (LUT) used for the feedforward component of a PID controller. The LUT provides predefined output values for given input conditions, enhancing the PID controller's response by compensating for known system behaviors in advance.

7.3 lut.h

Go to the documentation of this file.

```
00001
00023 struct motor {
00024     int speed; // ticks per second
00025     int pwm;   // 0-255
00026     int mode;  // 0 = normal, 1 = coast
00027 };
00028
00037 motor lut[] = {
00038     {0,0,0},
00039     {60,20,0},
00040     {85,25,0},
00041     {110,30,0},
00042     {135,35,0},
00043     {160,40,0},
00044     {185,45,0},
00045     {215,50,0},
00046     {240,55,0},
00047     {265,60,0},
00048     {290,65,0},
00049     {310,70,0},
00050     {335,75,0},
00051     {360,80,0},
00052     {385,85,0},
00053     {410,90,0},
00054     {435,95,0},
00055     {455,100,0},
00056     {480,105,0},
00057     {505,110,0},
00058     {530,115,0},
00059     {540,120,0},
00060     {565,125,0},
00061     {610,130,0},
00062     {635,135,0},
00063     {650,140,0},
00064     {675,145,0},
00065     {700,150,0},
00066     {725,155,0},
00067     {770,160,0},
00068     {795,165,0},
00069     {820,170,0},
00070     {845,175,0},
00071     {850,180,0},
00072     {875,185,0},
00073     {900,190,0},
00074     {915,195,0},
00075     {920,200,0},
00076     {935,205,0},
00077     {1000,210,0},
00078     {1025,215,0},
00079     {1050,220,0},
00080     {1075,225,0},
00081     {1100,230,0},
```

```
00082     {1125,235,0},
00083     {1150,240,0},
00084     {1175,245,0},
00085     {1200,250,0},
00086     {1225,253,0},
00087     {1250,255,0}
00088 };
00089
00090
```

7.4 README.md File Reference