Armuro Parcours

Generated by Doxygen 1.9.5

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Module Documentation

4.1 Armuro Hardware

Manage the hardware of the Armuro Robot.

Functions

- void print (char ∗format,...)

  Print a message to the serial port.
- int32_t map (int32_t x, int32_t in_min, int32_t in_max, int32_t out_min, int32_t out_max)

  Map a value from one range to another.
- void turnMotor (Side motor, int direction, int speed)

  Activate the motor.
- void stopMotor (Side motor)

  Stop the motor.
- void setLED (Side led, int state)

  Turn the led on the side on and off.
- void setRearLED (int state)

  Set the rear LED on or off.
- void didReadSensors (uint32_t ∗values)

  Handle a new value from the sensors.
- void didReadWheelEncoder (uint32_t leftValue, uint32_t rightValue)

  Handle a new value from the wheel encoders.
- void resetAngleMeasurement (Side wheel)

  Reset the angle measurement for the wheels.
- void stopAngleMeasurement (WheelAngle ∗angle)

  Stop measuring the angle for the wheels.
- int getAngleForWheel (Side wheel)

  et the Angle (in degrees) for the wheel
- void getAngleForWheels (int ∗leftAngle, int ∗rightAngle)

  Get the Angle (in degrees) for the wheels.
- void getRawLineSensorReadings (uint32_t ∗left, uint32_t ∗middle, uint32_t ∗right)

  Get the raw readings of the line sensors.
- void getLineSensorReadings (uint32_t ∗left, uint32_t ∗middle, uint32_t ∗right)

  Get the readings of the line sensors mapped to their typical range (0 - 1023)

- uint32_t mapLineSensorReadingToRange (uint32_t value, Side side)

  Map a line sensor reading to its typical range (0 - 1023)
- int distanceToAngle (double distance)

  Translate a distance (in cm) to an angle (in degrees)
- double angleToDistance (int angle)

  Get the Distance (in cm) for an angle (in degrees)
- double speedDifferenceForRadius (double radius)

  Calculate the speed difference for the wheels to drive a circle with the given radius.
- uint8_t checkSwitchesPressed (Side ∗side)

  Check if a switch is pressed.

4.1.1 Detailed Description

Manage the hardware of the Armuro Robot.

4.1.2 Function Documentation

4.1.2.1 angleToDistance()

```
double angleToDistance (
              int angle )
```

Get the Distance (in cm) for an angle (in degrees)

Parameters

angle	the angle for which the distance should be returned

Returns

the distance for the angle

4.1.2.2 checkSwitchesPressed()

```
uint8_t checkSwitchesPressed (
              Side ∗ side )
```

Check if a switch is pressed.

Parameters

side	a pointer to store the pressed switch in

Returns

true if a switch is pressed, false otherwise

4.1.2.3 didReadSensors()

```
void didReadSensors (
             uint32_t * values )
```

Handle a new value from the sensors.

Parameters

values	an array of 6 values, containing the sensor readigns

4.1.2.4 didReadWheelEncoder()

```
void didReadWheelEncoder (
             uint32_t leftValue,
             uint32_t rightValue )
```

Handle a new value from the wheel encoders.

Parameters

leftValue	the value of the left wheel encoder
rightValue	the value of the right wheel encoder

4.1.2.5 distanceToAngle()

```
int distanceToAngle (
             double distance )
```

Translate a distance (in cm) to an angle (in degrees)

Parameters

distance	the distance for which the angle should be returned

Returns

the angle for the distance

4.1.2.6 getAngleForWheel()

```
int getAngleForWheel (
             Side wheel )
```

et the Angle (in degrees) for the wheel

Parameters

wheel	the wheel for which the angle should be returned

Returns

the angle of the wheel in degrees

4.1.2.7 getAngleForWheels()

```
void getAngleForWheels (
             int * leftAngle,
             int * rightAngle )
```

Get the Angle (in degrees) for the wheels.

Parameters

leftAngle	a pointer to a variable in which the left angle should be stored
rightAngle	a pointer to a variable in which the right angle should be stored

4.1.2.8 getLineSensorReadings()

```
void getLineSensorReadings (
             uint32_t * left,
             uint32_t * middle,
             uint32_t * right )
```

Get the readings of the line sensors mapped to their typical range (0 - 1023)

Parameters

left	a pointer to a variable in which the left sensor reading should be stored
middle	a pointer to a variable in which the middle sensor reading should be stored
right	a pointer to a variable in which the right sensor reading should be stored

4.1.2.9 getRawLineSensorReadings()

```
void getRawLineSensorReadings (
             uint32_t * left,
             uint32_t * middle,
             uint32_t * right )
```

Get the raw readings of the line sensors.

Parameters

left	a pointer to a variable in which the left sensor reading should be stored
middle	a pointer to a variable in which the middle sensor reading should be stored
right	a pointer to a variable in which the right sensor reading should be stored

4.1.2.10 map()

```
int32_t map (
             int32_t x,
             int32_t in_min,
             int32_t in_max,
             int32_t out_min,
             int32_t out_max )
```

Map a value from one range to another.

Parameters

x	the value to map
in_min	the minimum value of the input range
in_max	the maximum value of the input range
out_min	the minimum value of the output range
out_max	the maximum value of the output range

Returns

the value mapped to the output range

4.1.2.11 mapLineSensorReadingToRange()

```
uint32_t mapLineSensorReadingToRange (
             uint32_t value,
             Side side )
```

Map a line sensor reading to its typical range (0 - 1023)

Parameters

value	the value to map
side	the side on which the sensor is located

Returns

the mapped value

4.1.2.12 print()

```
void print (
             char * format,
              ...  )  [inline]
```

Print a message to the serial port.

Parameters

format	the format of the message
...	the parameters for the format

4.1.2.13 resetAngleMeasurement()

```
void resetAngleMeasurement (
             Side wheel )
```

Reset the angle measurement for the wheels.

Parameters

wheel	the wheel for which the angle measurement should be reset

4.1.2.14 setLED()

```
void setLED (
             Side led,
             int state )
```

Turn the led on the side on and off.

Parameters

led	the side on which the led should be controlled
state	the state of the LED (HIGH or LOW)

4.1.2.15 setRearLED()

```
void setRearLED (
             int state )
```

Set the rear LED on or off.

Parameters

state	the state of the LED (HIGH or LOW)

4.1.2.16 speedDifferenceForRadius()

```
double speedDifferenceForRadius (
             double radius )
```

Calculate the speed difference for the wheels to drive a circle with the given radius.

Parameters

radius	the radius (in cm) of the circle to drive (to the middle of the robot)

Returns

the factor by which the speed of the outer wheel should be multiplied to get the speed for the inner wheel

4.1.2.17 stopAngleMeasurement()

```
void stopAngleMeasurement (
             WheelAngle * angle )
```

Stop measuring the angle for the wheels.

Parameters

angle	the angle struct to stop measuring for

4.1.2.18 stopMotor()

```
void stopMotor (
            Side motor )
```

Stop the motor.

Parameters

motor	the motor to stop

4.1.2.19 turnMotor()

```
void turnMotor (
            Side motor,
            int direction,
            int speed )
```

Activate the motor.

Parameters

motor	the motor to activate
direction	the direction in which the motor should turn
speed	the speed at which the motor should turn (in %)

4.2 Blink LED

Let the LEDs blink.

Functions

- void blinkLED (Side side, uint16_t timeInterval)

  Blink the LED on the given side with the given time interval.
- void stopBlinkingLED (Side side)

  Stop blinking the LED and turn it off on the given side.
- void blinkLEDTask ()

  Run the blinking Task.

4.2.1 Detailed Description

Let the LEDs blink.

4.2.2 Function Documentation

4.2.2.1 blinkLED()

```
void blinkLED (
             Side side,
             uint16_t timeInterval )
```

Blink the LED on the given side with the given time interval.

Parameters

side	the side to blink the LED on
timeInterval	the interval at which the LED should blink

4.2.2.2 blinkLEDTask()

```
void blinkLEDTask ( )
```

Run the blinking Task.

Note

> This method should be called in the main loop of the program as frequent as possible to assure the correct timing.

4.2.2.3 stopBlinkingLED()

```
void stopBlinkingLED (
             Side side )
```

Stop blinking the LED and turn it off on the given side.

Parameters

side	the side to of the LED

4.3 Calibrate Robot

Calibrate the hardware of the armuro robot.

Functions

- void calibrate ()

  Configure the calibration task.
- State calibrateTask ()

  Run the calibration task.
- void readWhiteLineSensors ()

  Calibrate the white value for the line sensors.
- void readBlackLineSensors ()

  Calibrate the black value for the line sensors.

4.3.1 Detailed Description

Calibrate the hardware of the armuro robot.

4.3.2 Function Documentation

4.3.2.1 calibrate()

```
void calibrate ( )
```

Configure the calibration task.

4.3.2.2 calibrateTask()

```
State calibrateTask ( )
```

Run the calibration task.

Returns

RUNNING if task is still running, FINISHED if task is finished

4.3.2.3 readBlackLineSensors()

```
void readBlackLineSensors ( )
```

Calibrate the black value for the line sensors.

4.3.2.4 readWhiteLineSensors()

```
void readWhiteLineSensors ( )
```

Calibrate the white value for the line sensors.

4.4 Line Follow

Let the robot follow the line.

Typedefs

- typedef enum SearchLineState SearchLineState

  The state of the line following state machine.
- typedef enum SearchLineResult SearchLineResult

  The result of the search line task.
- typedef enum CheckLineResult CheckLineResult

  The result of performing a line check.
- typedef enum FollowLineResult FollowLineResult

  The result of following the line.

Enumerations

- enum SearchLineState {
  DRIVE , TURNING_LEFT , TURNING_RIGHT , TURN_LEFT_TO_RIGHT ,
  TURN_RIGHT_TO_LEFT , TURN_RIGHT_TO_MIDDLE , TURN_LEFT_TO_MIDDLE , DONE }

  The state of the line following state machine.
- enum SearchLineResult { SEARCHING = 0 , FOUND = 1 , LOST = -1 , END_OF_LINE = 2 }

  The result of the search line task.
- enum CheckLineResult { ON_LINE = 0 , OFF_LINE = -1 , ALL_BLACK = 1 }

  The result of performing a line check.
- enum FollowLineResult { FOLLOWING = 0 , LOST_LINE = -1 , ALL_LINE = 1 }

  The result of following the line.

Functions

- void followLine (int speed)

  Follows the line until the end of the line is reached.
- FollowLineResult followLineTask ()

  Follows the line until the end of the line is reached.
- CheckLineResult checkForLine ()

  Checks if the line is lost.
- void searchLine ()

  Searches for the line.
- SearchLineResult searchLineTask ()

  Searches for the line.

4.4.1 Detailed Description

Let the robot follow the line.

4.4.2 Typedef Documentation

4.4.2.1 CheckLineResult

```
typedef enum CheckLineResult CheckLineResult
```

The result of performing a line check.

4.4.2.2 FollowLineResult

```
typedef enum FollowLineResult FollowLineResult
```

The result of following the line.

4.4.2.3 SearchLineResult

```
typedef enum SearchLineResult SearchLineResult
```

The result of the search line task.

4.4.2.4 SearchLineState

```
typedef enum SearchLineState SearchLineState
```

The state of the line following state machine.

4.4.3 Enumeration Type Documentation

4.4.3.1 CheckLineResult

```
enum CheckLineResult
```

The result of performing a line check.

Enumerator

ON_LINE	The robot is partly on the line.
OFF_LINE	The robot is not on the line.
ALL_BLACK	The robot is completely on the line.

4.4.3.2 FollowLineResult

```
enum FollowLineResult
```

The result of following the line.

Enumerator

FOLLOWING	The robot is currently following the line.
LOST_LINE	The robot lost the line.
ALL_LINE	The robot is fully on the line.

4.4.3.3 SearchLineResult

```
enum SearchLineResult
```

The result of the search line task.

Enumerator

SEARCHING	The robot is currently searching the line.
FOUND	The robot found the line.
LOST	The robot lost the line.
END_OF_LINE	The robot reached the end of the line.

4.4.3.4 SearchLineState

```
enum SearchLineState
```

The state of the line following state machine.

Enumerator

DRIVE	
TURNING_LEFT	

Enumerator

TURNING_RIGHT	
TURN_LEFT_TO_RIGHT	
TURN_RIGHT_TO_LEFT	
TURN_RIGHT_TO_MIDDLE	
TURN_LEFT_TO_MIDDLE	
DONE	

4.4.4 Function Documentation

4.4.4.1 checkForLine()

```
CheckLineResult checkForLine ( )
```

Checks if the line is lost.

Returns

0 if partly on line, -1 if line is lost, 1 if all is black

4.4.4.2 followLine()

```
void followLine (
            int speed )
```

Follows the line until the end of the line is reached.

A PID Controller is used to follow the line. The error is the difference between the left and right line sensor, which should be corrected to 0.

Parameters

speed	the speed at which the robot should follow the line (0-100)

4.4.4.3 followLineTask()

```
FollowLineResult followLineTask ( )
```

Follows the line until the end of the line is reached.

Note

> This method should be called in the main loop of the program as frequent as possible to assure the correct timing.

Returns

> 0 if currently following the line, -1 if line is lost, 1 if all is black

4.4.4.4 searchLine()

```
void searchLine ( )
```

Searches for the line.

4.4.4.5 searchLineTask()

```
SearchLineResult searchLineTask ( )
```

Searches for the line.

Turn to each side for 90 degrees and check if the line is found. This behaviour is implemented in a state machine in SearchLineState.

Note

> This method should be called in the main loop of the program as frequent as possible to assure the correct timing.

Returns

> 0 if currently searching for the line, 1 if line is found, -1 if line is lost, 2 if end of line is reached

4.5 Obstacle Avoidance

Let the robot avoid obstacles.

Data Structures

- struct ObstacleAvoidanceConfig

  The configuration for the obstacle avoidance.

Typedefs

- typedef enum ObstacleAvoidanceState ObstacleAvoidanceState

  The state machine for the obstacle avoidance.
- typedef struct ObstacleAvoidanceConfig ObstacleAvoidanceConfig

  The configuration for the obstacle avoidance.

Enumerations

- enum ObstacleAvoidanceState { BACK_OFF , TURN_FROM_OBSTACLE , DRIVE_CIRCLE , OBSTACLE_AVOIDANCE_DONE
  }

  The state machine for the obstacle avoidance.

Functions

- ObstacleAvoidanceConfig configureObstacleAvoidance (double obstacleRadius, int attackAngle)

  Configure the obstacle avoidance task with the given parameters.
- uint8_t checkForObstacle ()

  Check if there is an obstacle in front of the robot.
- void avoidObstacle ()

  Configure the obstacle avoidance task.
- State avoidObstacleTask ()

  Run the obstacle avoidance task.

4.5.1 Detailed Description

Let the robot avoid obstacles.

4.5.2 Typedef Documentation

4.5.2.1 ObstacleAvoidanceConfig

```
typedef struct ObstacleAvoidanceConfig ObstacleAvoidanceConfig
```

The configuration for the obstacle avoidance.

4.5.2.2 ObstacleAvoidanceState

```
typedef enum ObstacleAvoidanceState ObstacleAvoidanceState
```

The state machine for the obstacle avoidance.

4.5.3 Enumeration Type Documentation

4.5.3.1 ObstacleAvoidanceState

```
enum ObstacleAvoidanceState
```

The state machine for the obstacle avoidance.

Enumerator

BACK_OFF	Backing off from the obstacle.
TURN_FROM_OBSTACLE	Turning from the obstacle.
DRIVE_CIRCLE	Driving a circle around the obstacle.
OBSTACLE_AVOIDANCE_DONE	Finished driving around the obstacle.

4.5.4 Function Documentation

4.5.4.1 avoidObstacle()

```
void avoidObstacle ( )
```

Configure the obstacle avoidance task.

Back off from the obstacle, and drive a circle around it to avoid it. This behaviour is implemented in a state machine in ObstacleAvoidanceState.

4.5.4.2 avoidObstacleTask()

```
State avoidObstacleTask ( )
```

Run the obstacle avoidance task.

Note

This method should be called in the main loop of the program as frequent as possible to assure the correct timing.

Returns

RUNNING if task is still running, FINISHED if task is finished

4.5.4.3 checkForObstacle()

```
uint8_t checkForObstacle ( )
```

Check if there is an obstacle in front of the robot.

Checks all three switches in the front of the robot to detect an obstacle.

Returns

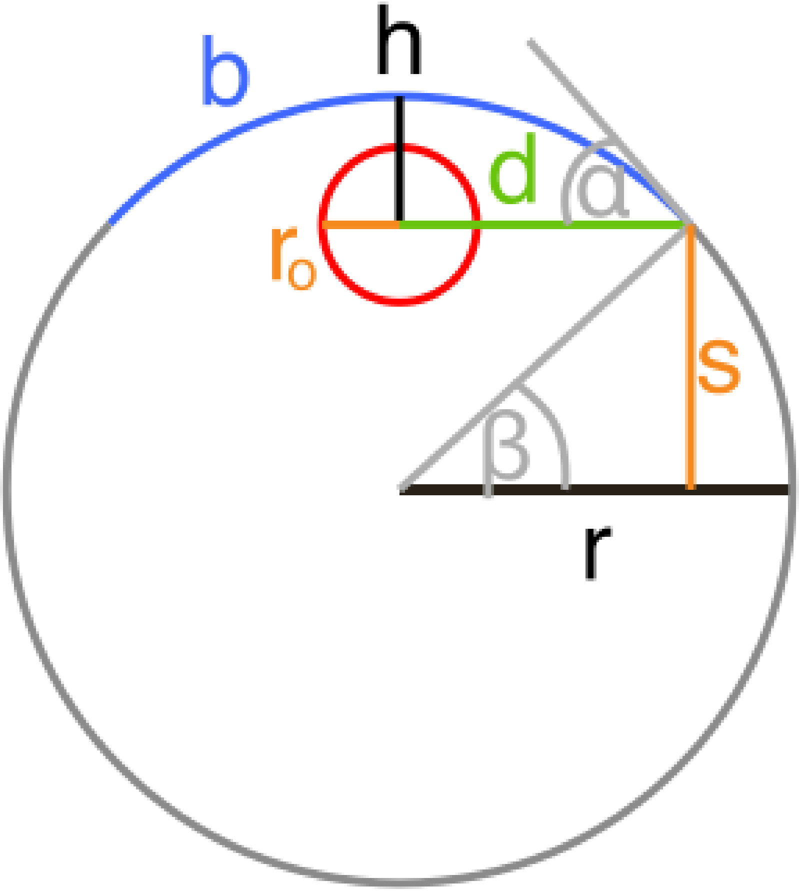
true if there is an obstacle, false otherwise

4.5.4.4 configureObstacleAvoidance()

```
ObstacleAvoidanceConfig configureObstacleAvoidance (
              double obstacleRadius,
              int attackAngle )
```

Configure the obstacle avoidance task with the given parameters.

Calculate the distance to back off from the obstacle, the radius of the circle to drive around the obstacle, and the distance to drive around the obstacle based on the desired angel of attack and the radius of the obstacle. The minimum distance to the obstacle is always the same, no matter the parameters.



$$\beta = 90 - \alpha$$

$$s = sin(\beta) \cdot r$$

$$h = r - s = r_o + \delta + \frac{1}{2}w$$

$$r = \frac{1}{1 - sin(\beta)} \cdot h$$

$$d = cos(\beta) \cdot r$$

$$b = r \cdot (\pi - 2 \cdot sin(\beta))$$

where α is the attack angle,
δ is the safety distance,
w is the wheel distance,
d is the distance to back off from the obstacle,
r is the radius of the circle to drive around the obstacle,
b is the distance to drive around the obstacle

Parameters

obstacleRadius	the radius of the obstacle
attackAngle	the angle to start the circle drive

Returns

ObstacleAvoidanceConfig

4.6 Parcour

Let the robot run the parcour.

Typedefs

- typedef enum StateMachine StateMachine

  The state machine of the parcour.

Enumerations

- enum StateMachine {
  DRIVE_TRAJECTORY = 0 , FOLLOW_LINE = 1 , SEARCH_LINE = 2 , OVERCOME_GAP = 3 ,
  AVOID_OBSTACLE = 4 , CALIBRATE = 5 , IDLE = -1 }

  The state machine of the parcour.

Functions

- void startParcour ()

  Configure the parcour task.
- void driveParcour ()

  Run the parcour task.

Variables

- StateMachine currentState = DRIVE_TRAJECTORY

  The current state of the parcour state machine.
- StateMachine nextState = SEARCH_LINE

  The next state of the parcour state machine.
- State state = READY

  The status of the parcour state machine's state.

4.6.1 Detailed Description

Let the robot run the parcour.

The parcour is implemented as a hirachical state machine. The robot is in a certain state and performs a certain action depending on the state. The differenet states of the robot while running the parcour are defined in the StateMachine enum.

4.6.2 Typedef Documentation

4.6.2.1 StateMachine

```
typedef enum StateMachine StateMachine
```

The state machine of the parcour.

4.6.3 Enumeration Type Documentation

4.6.3.1 StateMachine

```
enum StateMachine
```

The state machine of the parcour.

Enumerator

DRIVE_TRAJECTORY	Drive the trajectory.
FOLLOW_LINE	Follow the line.
SEARCH_LINE	Search the line.
OVERCOME_GAP	Overcome a gap in the line.
AVOID_OBSTACLE	Avoid an obstacle on the line.
CALIBRATE	Calibrate the hardware of the robot.
IDLE	Stop the robot.

4.6.4 Function Documentation

4.6.4.1 driveParcour()

```
void driveParcour ( )
```

Run the parcour task.

Note

> This method should be called in the main loop of the program as frequent as possible to assure the correct timing.

4.6.4.2 startParcour()

```
void startParcour ( )
```

Configure the parcour task.

4.6.5 Variable Documentation

4.6.5.1 currentState

```
StateMachine currentState = DRIVE_TRAJECTORY
```

The current state of the parcour state machine.

4.6.5.2 nextState

```
StateMachine nextState = SEARCH_LINE
```

The next state of the parcour state machine.

4.6.5.3 state

```
State state = READY
```

The status of the parcour state machine's state.

4.7 PID Controller

PID controller.

Data Structures

- struct PIDConfig

  Configuration for the PID controller.

Typedefs

- typedef struct PIDConfig PIDConfig

  Configuration for the PID controller.

Functions

- PIDConfig initPID (double p_gain, double i_gain, double d_gain, double max_i, double i_relax)

    Configure the PID controller.
- double calculatePIDOutput (double setpoint, double input, PIDConfig ∗config)

    Calculate the output of the PID controller.

4.7.1 Detailed Description

PID controller.

4.7.2 Typedef Documentation

4.7.2.1 PIDConfig

```
typedef struct PIDConfig PIDConfig
```

Configuration for the PID controller.

4.7.3 Function Documentation

4.7.3.1 calculatePIDOutput()

```
double calculatePIDOutput (
              double setpoint,
              double input,
              PIDConfig ∗ config )
```

Calculate the output of the PID controller.

Parameters

setpoint	the desired value
input	the current value
config	the configuration of the PID controller

Returns

the value to write in order to reach the desired value

4.7.3.2 initPID()

```
PIDConfig initPID (
             double p_gain,
             double i_gain,
             double d_gain,
             double max_i,
             double i_relax )
```

Configure the PID controller.

Parameters

p_gain	the gain for the proportional part
i_gain	the gain for the integral part
d_gain	the gain for the derivative part
max↩_i	the maximum value for the integral part
i_relax	the relaxation value for the integral part

Returns

a configuration for the PID controller

4.8 Trajectory

Drive a predefined trajectory.

Typedefs

- typedef enum TrajectoryStateMachine TrajectoryStateMachine

  The state machine of the trajectory.

Enumerations

- enum TrajectoryStateMachine {
  DRIVE_FIRST_TRAJECTORY_PART = 0 , TURN_TO_SECOND_TRAJECTORY_PART = 1 , DRIVE_SECOND_TRAJECTORY
  = 2 , TURN_TO_THIRD_TRAJECTORY_PART = 3 ,
  DRIVE_THIRD_TRAJECTORY_PART = 4 , FOLLOW_LINE = 5 , TRAJECTORY_DONE = -1 }

  The state machine of the trajectory.

Functions

- void startTrajectory ()

  Configure the trajectory task.
- State driveTrajectoryTask ()

  Run the trajectory task.

4.8.1 Detailed Description

Drive a predefined trajectory.

This module is implemented as a state machine in TrajectoryState.

4.8.2 Typedef Documentation

4.8.2.1 TrajectoryStateMachine

```
typedef enum TrajectoryStateMachine TrajectoryStateMachine
```

The state machine of the trajectory.

4.8.3 Enumeration Type Documentation

4.8.3.1 TrajectoryStateMachine

```
enum TrajectoryStateMachine
```

The state machine of the trajectory.

Enumerator

DRIVE_FIRST_TRAJECTORY_PART	
TURN_TO_SECOND_TRAJECTORY_PART	
DRIVE_SECOND_TRAJECTORY_PART	
TURN_TO_THIRD_TRAJECTORY_PART	
DRIVE_THIRD_TRAJECTORY_PART	
FOLLOW_LINE	
TRAJECTORY_DONE	

4.8.4 Function Documentation

4.8.4.1 driveTrajectoryTask()

```
State driveTrajectoryTask ( )
```

Run the trajectory task.

Note

This method should be called in the main loop of the program as frequent as possible to assure the correct timing.

Returns

RUNNING if task is still running, FINISHED if task is finished

4.8.4.2 startTrajectory()

```
void startTrajectory ( )
```

Configure the trajectory task.

4.9 Wheels

Control the wheels of the robot.

Typedefs

- typedef enum TurnWheelsTaskType TurnWheelsTaskType

  The type of tasks the wheels are currently executing.

Enumerations

- enum TurnWheelsTaskType {
  NONE = 0 , ANGLE = 1 , SPEED = 2 , SYNCHRONIZED = 3 ,
  TIMED_ANGLE = 4 , SYNCHRONIZED_ANGLE = 5 , TURN = 6 }

  The type of tasks the wheels are currently executing.

Functions

- void stopWheel (Side wheel)

  Stop the wheel.
- void turnWheelByAngle (Side wheel, int angle, int speed)

  Start to turn the wheel by a certain angle.
- void turnWheelByAngleInTime (Side wheel, int angle, int time)

  Start to turn the wheel by a certain angle in a certain time.
- void turnWheelWithSpeed (Side wheel, int speed)

  Start to turn the wheel with a certain speed.
- void turnWheelsSynchronized (int leftSpeed, int rightSpeed)

  Turn the wheels with a certain speed.
- void turnWheelsSynchronizedByAngle (int leftSpeed, int rightSpeed, int rightAngle, uint8_t softStart)

  Turn the wheels with a certain speed and a certain angle.
- void turnArmuroInTime (int angle, int time)

  Turn the armuro by a certain angle in a certain time.
- void turnArmuro (int angle)

  Turn the armuro by a certain angle with a certain speed.
- TurnWheelsTaskType ∗ turnWheelsTask ()

  Manage the turning of the wheels.

4.9.1 Detailed Description

Control the wheels of the robot.

4.9.2 Typedef Documentation

4.9.2.1 TurnWheelsTaskType

```
typedef enum TurnWheelsTaskType TurnWheelsTaskType
```

The type of tasks the wheels are currently executing.

4.9.3 Enumeration Type Documentation

4.9.3.1 TurnWheelsTaskType

```
enum TurnWheelsTaskType
```

The type of tasks the wheels are currently executing.

Enumerator

NONE	Wheel is stopped.
ANGLE	Wheel is turning by a certain angle.
SPEED	Wheel is turning by a certain speed.
SYNCHRONIZED	Wheels are turning with a certain speed relative to each other.
TIMED_ANGLE	Wheel is turning by a certain angle in a certain time.
SYNCHRONIZED_ANGLE	Wheels are turning with a certain speed relative to each other and a certain angle.
TURN	Robot is turning in place.

4.9.4 Function Documentation

4.9.4.1 stopWheel()

```
void stopWheel (
                Side wheel )
```

Stop the wheel.

Parameters

wheel	the wheel to stop

4.9.4.2 turnArmuro()

```
void turnArmuro (
             int angle )
```

Turn the armuro by a certain angle with a certain speed.

Parameters

angle	the angle to turn by (positive for left, negative for right)

4.9.4.3 turnArmuroInTime()

```
void turnArmuroInTime (
             int angle,
             int time )
```

Turn the armuro by a certain angle in a certain time.

Parameters

angle	the angle to turn by (positive for left, negative for right)
time	the time in which the armuro should complete the turn (in ms)

4.9.4.4 turnWheelByAngle()

```
void turnWheelByAngle (
             Side wheel,
             int angle,
             int speed )
```

Start to turn the wheel by a certain angle.

Parameters

wheel	the wheel to turn
angle	the angle to turn the wheel by (positive for forward, negative for backward)
speed	the speed at which the wheel should turn (0-100)

4.9.4.5 turnWheelByAngleInTime()

```
void turnWheelByAngleInTime (
            Side wheel,
            int angle,
            int time )
```

Start to turn the wheel by a certain angle in a certain time.

Parameters

wheel	the wheel to turn
angle	the angle to turn the wheel by (positive for forward, negative for backward)
time	the time in which the wheel should complete the turn (in ms)

4.9.4.6 turnWheelsSynchronized()

```
void turnWheelsSynchronized (
            int leftSpeed,
            int rightSpeed )
```

Turn the wheels with a certain speed.

This function initiates the turning of the wheels with a certain speed while regulating the speed difference between the wheels.

Parameters

leftSpeed	the speed of the left wheel
rightSpeed	the speed of the right wheel

4.9.4.7 turnWheelsSynchronizedByAngle()

```
void turnWheelsSynchronizedByAngle (
            int leftSpeed,
            int rightSpeed,
            int rightAngle,
            uint8_t softStart )
```

Turn the wheels with a certain speed and a certain angle.

Parameters

leftSpeed	the speed of the left wheel

Parameters

rightSpeed	the speed of the right wheel
rightAngle	the angle the right wheel should be turned by
softStart	whether the wheels should be started slowly

4.9.4.8 turnWheelsTask()

```
TurnWheelsTaskType * turnWheelsTask ( )
```

Manage the turning of the wheels.

This function should be called in a loop

Note

> This method should be called in the main loop of the program as frequent as possible to assure the correct timing.

Returns

> TurnWheelsTaskType* the current state of the wheels

4.9.4.9 turnWheelWithSpeed()

```
void turnWheelWithSpeed (
             Side wheel,
             int speed )
```

Start to turn the wheel with a certain speed.

Parameters

wheel	the wheel to turn
speed	the speed at which the wheel should turn (0-100)

Chapter 5

Data Structure Documentation

5.1 ObstacleAvoidanceConfig Struct Reference

The configuration for the obstacle avoidance.

Data Fields

- double circleRadius

  The radius of the circle to drive around the obstacle.
- double backOffDistance

  The distance to back off from the obstacle.
- double attackAngle

  The angle to start the circle drive.
- double distanceToDrive

  The distance to drive around the obstacle.

5.1.1 Detailed Description

The configuration for the obstacle avoidance.

5.1.2 Field Documentation

5.1.2.1 attackAngle

```
double attackAngle
```

The angle to start the circle drive.

5.1.2.2 backOffDistance

```
double backOffDistance
```

The distance to back off from the obstacle.

5.1.2.3 circleRadius

```
double circleRadius
```

The radius of the circle to drive around the obstacle.

5.1.2.4 distanceToDrive

```
double distanceToDrive
```

The distance to drive around the obstacle.

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

- /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/obstacleAvoidance/obstacleAvoidance.c

5.2 PIDConfig Struct Reference

Configuration for the PID controller.

```
#include <pidController.h>
```

Data Fields

- double p_gain
- double i_gain
- double d_gain
- double max_i
- double i_relax
- double old_input
- double integral

5.2.1 Detailed Description

Configuration for the PID controller.

5.2.2 Field Documentation

5.2.2.1 d_gain

```
double d_gain
```

5.2.2.2 i_gain

```
double i_gain
```

5.2.2.3 i_relax

```
double i_relax
```

5.2.2.4 integral

```
double integral
```

5.2.2.5 max_i

```
double max_i
```

5.2.2.6 old_input

```
double old_input
```

5.2.2.7 p_gain

```
double p_gain
```

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

- /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/pidController/pidController.h

5.3 WheelAngle Struct Reference

```
#include <armuro.h>
```

Data Fields

- int left
- int right
- int leftTicks
- int rightTicks

5.3.1 Field Documentation

5.3.1.1 left

```
int left
```

5.3.1.2 leftTicks

```
int leftTicks
```

5.3.1.3 right

```
int right
```

5.3.1.4 rightTicks

```
int rightTicks
```

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

- /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/armuro/armuro.h

5.4 WheelAngleListItem Struct Reference

Data Fields

- WheelAngle angle
- struct WheelAngleListItem ∗ next
- struct WheelAngleListItem ∗ prev

5.4.1 Field Documentation

5.4.1.1 angle

```
WheelAngle angle
```

5.4.1.2 next

```
struct WheelAngleListItem* next
```

5.4.1.3 prev

```
struct WheelAngleListItem* prev
```

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

- /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/armuro/armuro.c

Chapter 6

File Documentation

6.1 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/armuro/armuro.c File Reference

```
#include "armuro.h"
#include "gpio.h"
#include "tim.h"
#include "usart.h"
#include "string.h"
#include "stdio.h"
#include "stdlib.h"
```

Data Structures

- struct WheelAngleListItem

Macros

- #define RIGHT_ENCODER_HIGH_THRESHOLD 2000
- #define RIGHT_ENCODER_LOW_THRESHOLD 1000
- #define LEFT_ENCODER_HIGH_THRESHOLD 3300
- #define LEFT_ENCODER_LOW_THRESHOLD 3000
- #define MAX_PWM (1 << 16) - 1

Typedefs

- typedef struct WheelAngleListItem WheelAngleListItem

Functions

- void print (char ∗format,...)

  Print a message to the serial port.
- int32_t map (int32_t x, int32_t in_min, int32_t in_max, int32_t out_min, int32_t out_max)

  Map a value from one range to another.
- void initMotors ()

  Initialize the motors.
- void turnMotor (Side motor, int direction, int speed)

  Activate the motor.
- void stopMotor (Side motor)

  Stop the motor.
- void setLED (Side led, int state)

  Turn the led on the side on and off.
- void setRearLED (int state)

  Set the rear LED on or off.
- void didReadSensors (uint32_t ∗values)

  Handle a new value from the sensors.
- int schmittTrigger (Side side, u_int32_t value)

  Check if the value is above the high threshold or below the low threshold.
- void didReadWheelEncoder (uint32_t leftValue, uint32_t rightValue)

  Handle a new value from the wheel encoders.
- void resetAngleMeasurement (Side wheel)

  Reset the angle measurement for the wheels.
- WheelAngle ∗ startAngleMeasurement ()

  Start measuring the angle for the wheels.
- void stopAngleMeasurement (WheelAngle ∗angle)

  Stop measuring the angle for the wheels.
- int getAngleForWheel (Side wheel)

  et the Angle (in degrees) for the wheel
- void getAngleForWheels (int ∗leftAngle, int ∗rightAngle)

  Get the Angle (in degrees) for the wheels.
- void getRawLineSensorReadings (uint32_t ∗left, uint32_t ∗middle, uint32_t ∗right)

  Get the raw readings of the line sensors.
- void getLineSensorReadings (uint32_t ∗left, uint32_t ∗middle, uint32_t ∗right)

  Get the readings of the line sensors mapped to their typical range (0 - 1023)
- uint32_t mapLineSensorReadingToRange (uint32_t value, Side side)

  Map a line sensor reading to its typical range (0 - 1023)
- int distanceToAngle (double distance)

  Translate a distance (in cm) to an angle (in degrees)
- double angleToDistance (int angle)

  Get the Distance (in cm) for an angle (in degrees)
- double speedDifferenceForRadius (double radius)

  Calculate the speed difference for the wheels to drive a circle with the given radius.
- uint8_t checkSwitchesPressed (Side ∗side)

  Check if a switch is pressed.

Variables

- uint32_t minLineSensorValues [3]
- uint32_t maxLineSensorValues [3]
- uint16_t wheelEncoderTicksCount [2]
- int wheelEncoderOldValues [2]
- WheelAngleListItem ∗ wheelAngleList = NULL
- WheelAngleListItem ∗ wheelAngleListEnd = NULL

6.1.1 Macro Definition Documentation

6.1.1.1 LEFT_ENCODER_HIGH_THRESHOLD

```
#define LEFT_ENCODER_HIGH_THRESHOLD 3300
```

6.1.1.2 LEFT_ENCODER_LOW_THRESHOLD

```
#define LEFT_ENCODER_LOW_THRESHOLD 3000
```

6.1.1.3 MAX_PWM

```
#define MAX_PWM (1 << 16) - 1
```

6.1.1.4 RIGHT_ENCODER_HIGH_THRESHOLD

```
#define RIGHT_ENCODER_HIGH_THRESHOLD 2000
```

6.1.1.5 RIGHT_ENCODER_LOW_THRESHOLD

```
#define RIGHT_ENCODER_LOW_THRESHOLD 1000
```

6.1.2 Typedef Documentation

6.1.2.1 WheelAngleListItem

```
typedef struct WheelAngleListItem WheelAngleListItem
```

6.1.3 Function Documentation

6.1.3.1 initMotors()

```
void initMotors ( )
```

Initialize the motors.

6.1.3.2 schmittTrigger()

```
int schmittTrigger (
            Side side,
            u_int32_t value )
```

Check if the value is above the high threshold or below the low threshold.

Parameters

side	
value	

Returns

1 for HIGH, 0 for LOW, -1 for invalid

6.1.3.3 startAngleMeasurement()

```
WheelAngle * startAngleMeasurement ( )
```

Start measuring the angle for the wheels.

Returns

a pointer to a struct keeping the angle of the wheels

6.1.4 Variable Documentation

6.1.4.1 maxLineSensorValues

```
uint32_t maxLineSensorValues[3]
```

Initial value:

```
= {
    MAX_LEFT_LINE_SENSOR_VALUE,
    MAX_MIDDLE_LINE_SENSOR_VALUE,
    MAX_RIGHT_LINE_SENSOR_VALUE
}
```

6.1.4.2 minLineSensorValues

```
uint32_t minLineSensorValues[3]
```

Initial value:

```
= {
    MIN_LEFT_LINE_SENSOR_VALUE,
    MIN_MIDDLE_LINE_SENSOR_VALUE,
    MIN_RIGHT_LINE_SENSOR_VALUE
}
```

6.1.4.3 wheelAngleList

```
WheelAngleListItem* wheelAngleList = NULL
```

6.1.4.4 wheelAngleListEnd

```
WheelAngleListItem* wheelAngleListEnd = NULL
```

6.1.4.5 wheelEncoderOldValues

```
int wheelEncoderOldValues[2]
```

6.1.4.6 wheelEncoderTicksCount

```
uint16_t wheelEncoderTicksCount[2]
```

6.2 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/armuro/armuro.h File Reference

```
#include <stdint.h>
#include <stdarg.h>
```

Data Structures

- struct WheelAngle

Macros

- #define FORWARD 0
- #define BACKWARD 1
- #define HIGH 1
- #define LOW 0
- #define WHEEL_CIRCUMFERENCE 12.566
- #define TURN_CIRCUMFERENCE 24.504422698
- #define WHEEL_DISTANCE 7.8
- #define ARMURO_LENGTH 9
- #define MIN_ANGLE 15
- #define MAX_LEFT_LINE_SENSOR_VALUE 3380
- #define MAX_MIDDLE_LINE_SENSOR_VALUE 3380
- #define MAX_RIGHT_LINE_SENSOR_VALUE 2900
- #define MIN_LEFT_LINE_SENSOR_VALUE 1250
- #define MIN_MIDDLE_LINE_SENSOR_VALUE 1120
- #define MIN_RIGHT_LINE_SENSOR_VALUE 170

Enumerations

- enum Side { RIGHT = 0 , LEFT = 1 , MIDDLE = 3 }

Functions

- void print (char ∗format,...)

  Print a message to the serial port.
- int32_t map (int32_t x, int32_t in_min, int32_t in_max, int32_t out_min, int32_t out_max)

  Map a value from one range to another.
- void initMotors ()

  Initialize the motors.
- void turnMotor (Side motor, int direction, int speed)

  Activate the motor.
- void stopMotor (Side motor)

  Stop the motor.
- void setLED (Side led, int state)

  Turn the led on the side on and off.
- void setRearLED (int state)

  Set the rear LED on or off.

- void didReadSensors (uint32_t ∗values)

  Handle a new value from the sensors.
- void didReadWheelEncoder (uint32_t leftValue, uint32_t rightValue)

  Handle a new value from the wheel encoders.
- void resetAngleMeasurement (Side wheel)

  Reset the angle measurement for the wheels.
- WheelAngle ∗ startAngleMeasurement ()

  Start measuring the angle for the wheels.
- void stopAngleMeasurement (WheelAngle ∗angle)

  Stop measuring the angle for the wheels.
- int getAngleForWheel (Side wheel)

  et the Angle (in degrees) for the wheel
- void getAngleForWheels (int ∗leftAngle, int ∗rightAngle)

  Get the Angle (in degrees) for the wheels.
- void getRawLineSensorReadings (uint32_t ∗left, uint32_t ∗middle, uint32_t ∗right)

  Get the raw readings of the line sensors.
- void getLineSensorReadings (uint32_t ∗left, uint32_t ∗middle, uint32_t ∗right)

  Get the readings of the line sensors mapped to their typical range (0 - 1023)
- uint32_t mapLineSensorReadingToRange (uint32_t value, Side side)

  Map a line sensor reading to its typical range (0 - 1023)
- int distanceToAngle (double distance)

  Translate a distance (in cm) to an angle (in degrees)
- double angleToDistance (int angle)

  Get the Distance (in cm) for an angle (in degrees)
- double speedDifferenceForRadius (double radius)

  Calculate the speed difference for the wheels to drive a circle with the given radius.
- uint8_t checkSwitchesPressed (Side ∗side)

  Check if a switch is pressed.

Variables

- uint32_t minLineSensorValues [3]
- uint32_t maxLineSensorValues [3]
- uint16_t wheelEncoderTicksCount [2]
- uint32_t buffer [6]

6.2.1 Macro Definition Documentation

6.2.1.1 ARMURO_LENGTH

```
#define ARMURO_LENGTH 9
```

6.2.1.2 BACKWARD

```
#define BACKWARD 1
```

6.2.1.3 FORWARD

```
#define FORWARD 0
```

6.2.1.4 HIGH

```
#define HIGH 1
```

6.2.1.5 LOW

```
#define LOW 0
```

6.2.1.6 MAX_LEFT_LINE_SENSOR_VALUE

```
#define MAX_LEFT_LINE_SENSOR_VALUE 3380
```

6.2.1.7 MAX_MIDDLE_LINE_SENSOR_VALUE

```
#define MAX_MIDDLE_LINE_SENSOR_VALUE 3380
```

6.2.1.8 MAX_RIGHT_LINE_SENSOR_VALUE

```
#define MAX_RIGHT_LINE_SENSOR_VALUE 2900
```

6.2.1.9 MIN_ANGLE

```
#define MIN_ANGLE 15
```

6.2.1.10 MIN_LEFT_LINE_SENSOR_VALUE

```
#define MIN_LEFT_LINE_SENSOR_VALUE 1250
```

6.2.1.11 MIN_MIDDLE_LINE_SENSOR_VALUE

```
#define MIN_MIDDLE_LINE_SENSOR_VALUE 1120
```

6.2.1.12 MIN_RIGHT_LINE_SENSOR_VALUE

```
#define MIN_RIGHT_LINE_SENSOR_VALUE 170
```

6.2.1.13 TURN_CIRCUMFERENCE

```
#define TURN_CIRCUMFERENCE 24.504422698
```

6.2.1.14 WHEEL_CIRCUMFERENCE

```
#define WHEEL_CIRCUMFERENCE 12.566
```

6.2.1.15 WHEEL_DISTANCE

```
#define WHEEL_DISTANCE 7.8
```

6.2.2 Enumeration Type Documentation

6.2.2.1 Side

```
enum Side
```

Enumerator

RIGHT	
LEFT	
MIDDLE	

6.2.3 Function Documentation

6.2.3.1 initMotors()

```
void initMotors ( )
```

Initialize the motors.

6.2.3.2 startAngleMeasurement()

```
WheelAngle * startAngleMeasurement ( )
```

Start measuring the angle for the wheels.

Returns

a pointer to a struct keeping the angle of the wheels

6.2.4 Variable Documentation

6.2.4.1 buffer

```
uint32_t buffer[6] [extern]
```

6.2.4.2 maxLineSensorValues

```
uint32_t maxLineSensorValues[3] [extern]
```

6.2.4.3 minLineSensorValues

```
uint32_t minLineSensorValues[3]  [extern]
```

6.2.4.4 wheelEncoderTicksCount

```
uint16_t wheelEncoderTicksCount[2]  [extern]
```

6.3 armuro.h

Go to the documentation of this file.

```
#ifndef _ARMURO_H_
#define _ARMURO_H_

#define FORWARD 0
#define BACKWARD 1

#define HIGH 1
#define LOW 0

#define WHEEL_CIRCUMFERENCE 12.566
#define TURN_CIRCUMFERENCE 24.504422698
#define WHEEL_DISTANCE 7.8
#define ARMURO_LENGTH 9

#define MIN_ANGLE 15

#define MAX_LEFT_LINE_SENSOR_VALUE 3380
#define MAX_MIDDLE_LINE_SENSOR_VALUE 3380
#define MAX_RIGHT_LINE_SENSOR_VALUE 2900
#define MIN_LEFT_LINE_SENSOR_VALUE 1250
#define MIN_MIDDLE_LINE_SENSOR_VALUE 1120
#define MIN_RIGHT_LINE_SENSOR_VALUE 170

#include <stdint.h>
#include <stdarg.h>

extern uint32_t minLineSensorValues[3];
extern uint32_t maxLineSensorValues[3];

extern uint16_t wheelEncoderTicksCount[2];
extern uint32_t buffer[6];

typedef enum {
    RIGHT = 0,
    LEFT = 1,
    MIDDLE = 3
} Side;

typedef struct {
    int left;
    int right;

    int leftTicks;
    int rightTicks;
} WheelAngle;

void print(char* format, ...);

int32_t map(int32_t x, int32_t in_min, int32_t in_max, int32_t out_min, int32_t out_max);

void initMotors();

void turnMotor(Side motor, int direction, int speed);

void stopMotor(Side motor);

void setLED(Side led, int state);

void setRearLED(int state);

void didReadSensors(uint32_t* values);

void didReadWheelEncoder(uint32_t leftValue, uint32_t rightValue);

void resetAngleMeasurement(Side wheel);

WheelAngle* startAngleMeasurement();

void stopAngleMeasurement(WheelAngle* angle);

int getAngleForWheel(Side wheel);

void getAngleForWheels(int* leftAngle, int* rightAngle);

void getRawLineSensorReadings(uint32_t* left, uint32_t* middle, uint32_t* right);

void getLineSensorReadings(uint32_t* left, uint32_t* middle, uint32_t* right);

uint32_t mapLineSensorReadingToRange(uint32_t value, Side side);

int distanceToAngle(double distance);

double angleToDistance(int angle);

double speedDifferenceForRadius(double radius);

uint8_t checkSwitchesPressed(Side* side);

#endif
```

6.4 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/blinkLED/blinkLED.c File Reference

```
#include "blinkLED.h"
#include "tim.h"
#include "gpio.h"
```

Enumerations

- enum LEDState { ON , OFF }

Functions

- void blinkLED (Side side, uint16_t timeInterval)

  Blink the LED on the given side with the given time interval.
- void stopBlinkingLED (Side side)

  Stop blinking the LED and turn it off on the given side.
- void blinkLEDTask ()

  Run the blinking Task.

Variables

- uint16_t ledTimeInterval [ ] = {0, 0}
- uint32_t taskLEDTimeout [ ] = {-1, -1}
- enum LEDState ledState [ ] = {OFF, OFF}

6.4.1 Enumeration Type Documentation

6.4.1.1 LEDState

```
enum LEDState
```

Enumerator

ON	
OFF	

6.4.2 Variable Documentation

6.4.2.1 ledState

```
enum LEDState ledState[] = {OFF, OFF}
```

6.4.2.2 ledTimeInterval

```
uint16_t ledTimeInterval[] = {0, 0}
```

6.4.2.3 taskLEDTimeout

```
uint32_t taskLEDTimeout[] = {-1, -1}
```

6.5 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/blinkLED/blinkLED.h File Reference

```
#include "armuro.h"
```

Functions

- void blinkLED (Side side, uint16_t timeInterval)

  Blink the LED on the given side with the given time interval.
- void stopBlinkingLED (Side side)

  Stop blinking the LED and turn it off on the given side.
- void blinkLEDTask ()

  Run the blinking Task.

6.6 blinkLED.h

Go to the documentation of this file.

```
#ifndef _BLINK_LED_H_
#define _BLINK_LED_H_

#include "armuro.h"

void blinkLED(Side side, uint16_t timeInterval);

void stopBlinkingLED(Side side);

void blinkLEDTask();

#endif
```

6.7 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/calibrate/calibrate.c File Reference

```
#include "calibrate.h"
```

Enumerations

- enum CalibrateState { CALIBRATE_START , CALIBRATE_WHITE , CALIBRATE_BLACK , CALIBRATE_DONE }

Functions

- void startCalibrate ()
- void calibrate ()

  Configure the calibration task.
- State calibrateTask ()

  Run the calibration task.
- void readWhiteLineSensors ()

  Calibrate the white value for the line sensors.
- void readBlackLineSensors ()

  Calibrate the black value for the line sensors.

Variables

- CalibrateState calibrateState = CALIBRATE_START
- CalibrateState nextCalibrateState = CALIBRATE_START
- State calibrateStateState = READY

6.7.1 Enumeration Type Documentation

6.7.1.1 CalibrateState

```
enum CalibrateState
```

Enumerator

CALIBRATE_START	
CALIBRATE_WHITE	
CALIBRATE_BLACK	
CALIBRATE_DONE	

6.7.2 Function Documentation

6.7.2.1 startCalibrate()

```
void startCalibrate ( )
```

6.7.3 Variable Documentation

6.7.3.1 calibrateState

```
CalibrateState calibrateState = CALIBRATE_START
```

6.7.3.2 calibrateStateState

```
State calibrateStateState = READY
```

6.7.3.3 nextCalibrateState

```
CalibrateState nextCalibrateState = CALIBRATE_START
```

6.8 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/calibrate/calibrate.h File Reference

```
#include "armuro.h"
#include "stateMachine.h"
```

Functions

- void calibrate ()

  Configure the calibration task.
- State calibrateTask ()

  Run the calibration task.
- void readWhiteLineSensors ()

  Calibrate the white value for the line sensors.
- void readBlackLineSensors ()

  Calibrate the black value for the line sensors.

6.9 calibrate.h

Go to the documentation of this file.

```
#ifndef _CALIBRATE_H_
#define _CALIBRATE_H_

#include "armuro.h"
#include "stateMachine.h"

void calibrate();
State calibrateTask();

void readWhiteLineSensors();
void readBlackLineSensors();

#endif
```

6.10 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/lineFollow/lineFollow.c File Reference

```
#include "lineFollow.h"
#include "armuro.h"
#include "pidController.h"
#include "tim.h"
#include "wheels.h"
#include "blinkLED.h"
#include "stdlib.h"
#include "stateMachine.h"
```

Macros

- #define BLACK_THRESHOLD 900
- #define WHITE_THRESHOLD 400

Typedefs

- typedef enum SearchLineState SearchLineState

  The state of the line following state machine.

Enumerations

- enum SearchLineState {
  DRIVE , TURNING_LEFT , TURNING_RIGHT , TURN_LEFT_TO_RIGHT ,
  TURN_RIGHT_TO_LEFT , TURN_RIGHT_TO_MIDDLE , TURN_LEFT_TO_MIDDLE , DONE }

  The state of the line following state machine.

Functions

- void followLine (int speed)

  Follows the line until the end of the line is reached.
- FollowLineResult followLineTask ()

  Follows the line until the end of the line is reached.
- CheckLineResult checkForLine ()

  Checks if the line is lost.
- void searchLine ()

  Searches for the line.
- SearchLineResult searchLineTask ()

  Searches for the line.

Variables

- PIDConfig followLinePID
- uint32_t lineFollowTimeout = 0
- int baseLineSpeed = 0
- int lastState = 0
- CheckLineResult lastLineValues [3] = {OFF_LINE, OFF_LINE, OFF_LINE}
- SearchLineState searchLineState = TURNING_LEFT
- SearchLineState nextSearchState = TURNING_LEFT
- State searchLineStateState = READY

6.10.1 Macro Definition Documentation

6.10.1.1 BLACK_THRESHOLD

```
#define BLACK_THRESHOLD 900
```

6.10.1.2 WHITE_THRESHOLD

```
#define WHITE_THRESHOLD 400
```

6.10.2 Variable Documentation

6.10.2.1 baseLineSpeed

```
int baseLineSpeed = 0
```

6.10.2.2 followLinePID

```
PIDConfig followLinePID
```

6.10.2.3 lastLineValues

```
CheckLineResult lastLineValues[3] = {OFF_LINE, OFF_LINE, OFF_LINE}
```

6.10.2.4 lastState

```
int lastState = 0
```

6.10.2.5 lineFollowTimeout

```
uint32_t lineFollowTimeout = 0
```

6.10.2.6 nextSearchState

```
SearchLineState nextSearchState = TURNING_LEFT
```

6.10.2.7 searchLineState

```
SearchLineState searchLineState = TURNING_LEFT
```

6.10.2.8 searchLineStateState

```
State searchLineStateState = READY
```

6.11 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/lineFollow/lineFollow.h File Reference

Typedefs

- typedef enum SearchLineResult SearchLineResult

  The result of the search line task.
- typedef enum CheckLineResult CheckLineResult

  The result of performing a line check.
- typedef enum FollowLineResult FollowLineResult

  The result of following the line.

Enumerations

- enum SearchLineResult { SEARCHING = 0 , FOUND = 1 , LOST = -1 , END_OF_LINE = 2 }

  The result of the search line task.
- enum CheckLineResult { ON_LINE = 0 , OFF_LINE = -1 , ALL_BLACK = 1 }

  The result of performing a line check.
- enum FollowLineResult { FOLLOWING = 0 , LOST_LINE = -1 , ALL_LINE = 1 }

  The result of following the line.

Functions

- void followLine (int speed)

  Follows the line until the end of the line is reached.
- FollowLineResult followLineTask ()

  Follows the line until the end of the line is reached.
- CheckLineResult checkForLine ()

  Checks if the line is lost.
- void searchLine ()

  Searches for the line.
- SearchLineResult searchLineTask ()

  Searches for the line.

6.12 lineFollow.h

Go to the documentation of this file.

```
#ifndef LINE_FOLLOW_H
#define LINE_FOLLOW_H

typedef enum SearchLineResult {
    SEARCHING = 0,
    FOUND = 1,
    LOST = -1,
    END_OF_LINE = 2
} SearchLineResult;

typedef enum CheckLineResult {
    ON_LINE = 0,
    OFF_LINE = -1,
    ALL_BLACK = 1
} CheckLineResult;

typedef enum FollowLineResult {
    FOLLOWING = 0,
    LOST_LINE = -1,
    ALL_LINE = 1
} FollowLineResult;

void followLine(int speed);

FollowLineResult followLineTask();

CheckLineResult checkForLine();

void searchLine();

SearchLineResult searchLineTask();

#endif
```

6.13 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/obstacleAvoidance/obstacleAvoidance.c File Reference

```
#include "obstacleAvoidance.h"
#include "armuro.h"
#include "wheels.h"
#include "lineFollow.h"
#include "math.h"
```

Data Structures

- struct ObstacleAvoidanceConfig

  The configuration for the obstacle avoidance.

Macros

- #define OBSTACLE_RADIUS 4
- #define SAFETY_DISTANCE 3
- #define ATTACK_ANGLE 60

Typedefs

- typedef enum ObstacleAvoidanceState ObstacleAvoidanceState

  The state machine for the obstacle avoidance.
- typedef struct ObstacleAvoidanceConfig ObstacleAvoidanceConfig

  The configuration for the obstacle avoidance.

Enumerations

- enum ObstacleAvoidanceState { BACK_OFF , TURN_FROM_OBSTACLE , DRIVE_CIRCLE , OBSTACLE_AVOIDANCE_DONE
  }

  The state machine for the obstacle avoidance.

Functions

- ObstacleAvoidanceConfig configureObstacleAvoidance (double obstacleRadius, int attackAngle)

  Configure the obstacle avoidance task with the given parameters.
- uint8_t checkForObstacle ()

  Check if there is an obstacle in front of the robot.
- void avoidObstacle ()

  Configure the obstacle avoidance task.
- State avoidObstacleTask ()

  Run the obstacle avoidance task.

Variables

- ObstacleAvoidanceState obstacleAvoidanceState = TURN_FROM_OBSTACLE
- ObstacleAvoidanceState nextObstacleAvoidanceState = TURN_FROM_OBSTACLE
- State obstacleAvoidanceStateState = READY
- ObstacleAvoidanceConfig obstacleAvoidanceConfig

6.13.1 Macro Definition Documentation

6.13.1.1 ATTACK_ANGLE

```
#define ATTACK_ANGLE 60
```

6.13.1.2 OBSTACLE_RADIUS

```
#define OBSTACLE_RADIUS 4
```

6.13.1.3 SAFETY_DISTANCE

```
#define SAFETY_DISTANCE 3
```

6.13.2 Variable Documentation

6.13.2.1 nextObstacleAvoidanceState

```
ObstacleAvoidanceState nextObstacleAvoidanceState = TURN_FROM_OBSTACLE
```

6.13.2.2 obstacleAvoidanceConfig

```
ObstacleAvoidanceConfig obstacleAvoidanceConfig
```

6.13.2.3 obstacleAvoidanceState

```
ObstacleAvoidanceState obstacleAvoidanceState = TURN_FROM_OBSTACLE
```

6.13.2.4 obstacleAvoidanceStateState

```
State obstacleAvoidanceStateState = READY
```

6.14 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/obstacleAvoidance/obstacleAvoidance.h File Reference

```
#include "stateMachine.h"
#include "armuro.h"
```

Functions

- uint8_t checkForObstacle ()

  Check if there is an obstacle in front of the robot.
- void avoidObstacle ()

  Configure the obstacle avoidance task.
- State avoidObstacleTask ()

  Run the obstacle avoidance task.

6.15 obstacleAvoidance.h

Go to the documentation of this file.

```
#ifndef _OBSTACLE_AVOIDANCE_H_
#define _OBSTACLE_AVOIDANCE_H_

#include "stateMachine.h"
#include "armuro.h"

uint8_t checkForObstacle();

void avoidObstacle();

State avoidObstacleTask();

#endif
```

6.16 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/parcour/parcour.c File Reference

```
#include "parcour.h"
#include "wheels.h"
#include "armuro.h"
#include "lineFollow.h"
#include "trajectory.h"
#include "stateMachine.h"
#include "blinkLED.h"
#include "calibrate.h"
#include "obstacleAvoidance.h"
```

Typedefs

- typedef enum StateMachine StateMachine

  The state machine of the parcour.

Enumerations

- enum StateMachine {
  DRIVE_TRAJECTORY = 0 , FOLLOW_LINE = 1 , SEARCH_LINE = 2 , OVERCOME_GAP = 3 ,
  AVOID_OBSTACLE = 4 , CALIBRATE = 5 , IDLE = -1 }

  The state machine of the parcour.

Functions

- void driveTrajectory ()
- void lineFollow ()
- void searchTheLine ()
- void overcomeGap ()
- void calibrateArmuro ()
- void avoidingObstacle ()
- void startParcour ()

  Configure the parcour task.
- void driveParcour ()

  Run the parcour task.

Variables

- StateMachine currentState = DRIVE_TRAJECTORY

  The current state of the parcour state machine.
- StateMachine nextState = SEARCH_LINE

  The next state of the parcour state machine.
- State state = READY

  The status of the parcour state machine's state.

6.16.1 Function Documentation

6.16.1.1 avoidingObstacle()

```
void avoidingObstacle ( )
```

6.16.1.2 calibrateArmuro()

```
void calibrateArmuro ( )
```

6.16.1.3 driveTrajectory()

```
void driveTrajectory ( )
```

6.16.1.4 lineFollow()

```
void lineFollow ( )
```

6.16.1.5 overcomeGap()

```
void overcomeGap ( )
```

6.16.1.6 searchTheLine()

```
void searchTheLine ( )
```

6.17 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/parcour/parcour.h File Reference

Functions

- void startParcour ()

  Configure the parcour task.
- void driveParcour ()

  Run the parcour task.

6.18 parcour.h

Go to the documentation of this file.

```
#ifndef _PARCOUR_H_
#define _PARCOUR_H_

void startParcour();

void driveParcour();

#endif
```

6.19 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/pidController/pidController.c File Reference

```
#include "pidController.h"
```

Functions

- PIDConfig initPID (double p, double i, double d, double max_i, double i_relaxation)

  Configure the PID controller.
- double calculatePIDOutput (double setpoint, double input, PIDConfig ∗config)

  Calculate the output of the PID controller.

6.20 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/pidController/pidController.h File Reference

```
#include <stdint.h>
```

Data Structures

- struct PIDConfig

  Configuration for the PID controller.

Typedefs

- typedef struct PIDConfig PIDConfig

  Configuration for the PID controller.

Functions

- PIDConfig initPID (double p_gain, double i_gain, double d_gain, double max_i, double i_relax)

  Configure the PID controller.
- double calculatePIDOutput (double setpoint, double input, PIDConfig ∗config)

  Calculate the output of the PID controller.

6.21 pidController.h

Go to the documentation of this file.

```
#ifndef _PID_CONTROLLER_H_
#define _PID_CONTROLLER_H_

#include <stdint.h>

typedef struct PIDConfig {
    double p_gain;
    double i_gain;
    double d_gain;
    double max_i;
    double i_relax;
    double old_input;
    double integral;
} PIDConfig;

PIDConfig initPID(double p_gain, double i_gain, double d_gain, double max_i, double i_relax);

double calculatePIDOutput(double setpoint, double input, PIDConfig* config);

#endif
```

6.22 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/stateMachine/stateMachine.h File Reference

Typedefs

- typedef enum State State

Enumerations

- enum State { READY = 0 , RUNNING = 1 , FINISHED = 2 }

6.22.1 Typedef Documentation

6.22.1.1 State

```
typedef enum State State
```

6.22.2 Enumeration Type Documentation

6.22.2.1 State

```
enum State
```

Enumerator

READY	
RUNNING	
FINISHED	

6.23 stateMachine.h

Go to the documentation of this file.

```
#ifndef _STATE_MACHINE_H_
#define _STATE_MACHINE_H_

typedef enum State {
    READY = 0,
    RUNNING = 1,
    FINISHED = 2
} State;

#endif
```

6.24 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/trajectory/trajectory.c File Reference

```
#include "trajectory.h"
#include "wheels.h"
#include "armuro.h"
#include "lineFollow.h"
```

Typedefs

- typedef enum TrajectoryStateMachine TrajectoryStateMachine

  The state machine of the trajectory.

Enumerations

- enum TrajectoryStateMachine {
  DRIVE_FIRST_TRAJECTORY_PART = 0 , TURN_TO_SECOND_TRAJECTORY_PART = 1 , DRIVE_SECOND_TRAJECTORY
  = 2 , TURN_TO_THIRD_TRAJECTORY_PART = 3 ,
  DRIVE_THIRD_TRAJECTORY_PART = 4 , FOLLOW_LINE = 5 , TRAJECTORY_DONE = -1 }

  The state machine of the trajectory.

Functions

- void driveFirstTrajectoryPart ()
- void turnToSecondTrajectoryPart ()
- void driveSecondTrajectoryPart ()
- void turnToThirdTrajectoryPart ()
- void driveThirdTrajectoryPart ()
- void startTrajectory ()

  Configure the trajectory task.
- State driveTrajectoryTask ()

  Run the trajectory task.

Variables

- TrajectoryStateMachine currentTrajectoryState = FOLLOW_LINE
- TrajectoryStateMachine nextTrajectoryState = DRIVE_FIRST_TRAJECTORY_PART
- State trajectoryStateState = READY

6.24.1 Function Documentation

6.24.1.1 driveFirstTrajectoryPart()

```
void driveFirstTrajectoryPart ( )
```

6.24.1.2 driveSecondTrajectoryPart()

```
void driveSecondTrajectoryPart ( )
```

6.24.1.3 driveThirdTrajectoryPart()

```
void driveThirdTrajectoryPart ( )
```

6.24.1.4 turnToSecondTrajectoryPart()

```
void turnToSecondTrajectoryPart ( )
```

6.24.1.5 turnToThirdTrajectoryPart()

```
void turnToThirdTrajectoryPart ( )
```

6.24.2 Variable Documentation

6.24.2.1 currentTrajectoryState

```
TrajectoryStateMachine currentTrajectoryState = FOLLOW_LINE
```

6.24.2.2 nextTrajectoryState

```
TrajectoryStateMachine nextTrajectoryState = DRIVE_FIRST_TRAJECTORY_PART
```

6.24.2.3 trajectoryStateState

```
State trajectoryStateState = READY
```

6.25 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/trajectory/trajectory.h File Reference

```
#include "stateMachine.h"
```

Functions

- void startTrajectory ()

  Configure the trajectory task.
- State driveTrajectoryTask ()

  Run the trajectory task.

6.26 trajectory.h

Go to the documentation of this file.

```
#ifndef _TRAJECTORY_H_
#define _TRAJECTORY_H_

#include "stateMachine.h"

void startTrajectory();

State driveTrajectoryTask();

#endif
```

6.27 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/wheels/wheels.c File Reference

```
#include "wheels.h"
#include "usart.h"
#include "pidController.h"
#include <stdlib.h>
```

Macros

- #define MAX_ROTATION_RATE 900

Functions

- void stopWheel (Side wheel)

  Stop the wheel.
- void turnWheelByAngle (Side wheel, int angle, int speed)

  Start to turn the wheel by a certain angle.
- void turnWheelByAngleInTime (Side wheel, int angle, int time)

  Start to turn the wheel by a certain angle in a certain time.
- void turnWheelWithSpeed (Side wheel, int speed)

  Start to turn the wheel with a certain speed.
- void setupSynchronizedPID ()
- void turnWheelsSynchronized (int leftSpeed, int rightSpeed)

  Turn the wheels with a certain speed.
- void turnWheelsSynchronizedByAngle (int leftSpeed, int rightSpeed, int rightAngle, uint8_t softStart)

  Turn the wheels with a certain speed and a certain angle.
- void turnArmuroInTime (int angle, int time)

  Turn the armuro by a certain angle in a certain time.
- void turnArmuro (int angle)

  Turn the armuro by a certain angle with a certain speed.
- TurnWheelsTaskType ∗ turnWheelsTask ()

  Manage the turning of the wheels.
- void turnWheelByAngleTask (Side wheel)
- void turnWheelsSynchronizedTask ()
- void turnWheelsSynchronizedByAngleTask ()
- void turnWheelByAngleInTimeTask (Side wheel)
- void turnArmuroTask (Side wheel)

Variables

- TurnWheelsTaskType turningWheels [ ] = {NONE, NONE}
- int angleSetpoint [ ] = {0, 0}
- int speedSetpoint [ ] = {0, 0}
- int currentSpeedSetpoint [ ] = {0, 0}
- int rotationRateSetpoint [ ] = {0, 0}
- int oldAngle [ ] = {0, 0}
- uint32_t angleTimeout [ ] = {0, 0}
- PIDConfig wheelsPID [2]
- PIDConfig synchronizeWheelsPID
- uint32_t synchronizeWheelsTimeout = 0

6.27.1 Macro Definition Documentation

6.27.1.1 MAX_ROTATION_RATE

```
#define MAX_ROTATION_RATE 900
```

6.27.2 Function Documentation

6.27.2.1 setupSynchronizedPID()

```
void setupSynchronizedPID ( )
```

6.27.2.2 turnArmuroTask()

```
void turnArmuroTask (
            Side wheel )
```

6.27.2.3 turnWheelByAngleInTimeTask()

```
void turnWheelByAngleInTimeTask (
            Side wheel )
```

6.27.2.4 turnWheelByAngleTask()

```
void turnWheelByAngleTask (
            Side wheel )
```

6.27.2.5 turnWheelsSynchronizedByAngleTask()

```
void turnWheelsSynchronizedByAngleTask ( )
```

6.27.2.6 turnWheelsSynchronizedTask()

```
void turnWheelsSynchronizedTask ( )
```

6.27.3 Variable Documentation

6.27.3.1 angleSetpoint

```
int angleSetpoint[] = {0, 0}
```

6.27.3.2 angleTimeout

```
uint32_t angleTimeout[] = {0, 0}
```

6.27.3.3 currentSpeedSetpoint

```
int currentSpeedSetpoint[] = {0, 0}
```

6.27.3.4 oldAngle

```
int oldAngle[] = {0, 0}
```

6.27.3.5 rotationRateSetpoint

```
int rotationRateSetpoint[] = {0, 0}
```

6.27.3.6 speedSetpoint

```
int speedSetpoint[] = {0, 0}
```

6.27.3.7 synchronizeWheelsPID

```
PIDConfig synchronizeWheelsPID
```

6.27.3.8 synchronizeWheelsTimeout

```
uint32_t synchronizeWheelsTimeout = 0
```

6.27.3.9 turningWheels

```
TurnWheelsTaskType turningWheels[] = {NONE, NONE}
```

6.27.3.10 wheelsPID

```
PIDConfig wheelsPID[2]
```

6.28 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/wheels/wheels.h File Reference

```
#include <armuro.h>
```

Typedefs

- typedef enum TurnWheelsTaskType TurnWheelsTaskType

  The type of tasks the wheels are currently executing.

Enumerations

- enum TurnWheelsTaskType {
  NONE = 0 , ANGLE = 1 , SPEED = 2 , SYNCHRONIZED = 3 ,
  TIMED_ANGLE = 4 , SYNCHRONIZED_ANGLE = 5 , TURN = 6 }

  The type of tasks the wheels are currently executing.

Functions

- void stopWheel (Side wheel)

  Stop the wheel.
- void turnWheelByAngle (Side wheel, int angle, int speed)

  Start to turn the wheel by a certain angle.
- void turnWheelByAngleInTime (Side wheel, int angle, int time)

  Start to turn the wheel by a certain angle in a certain time.
- void turnWheelWithSpeed (Side wheel, int speed)

  Start to turn the wheel with a certain speed.
- void turnWheelsSynchronized (int leftSpeed, int rightSpeed)

  Turn the wheels with a certain speed.
- void turnWheelsSynchronizedByAngle (int leftSpeed, int rightSpeed, int rightAngle, uint8_t softStart)

  Turn the wheels with a certain speed and a certain angle.
- void turnArmuroInTime (int angle, int time)

  Turn the armuro by a certain angle in a certain time.
- void turnArmuro (int angle)

  Turn the armuro by a certain angle with a certain speed.
- TurnWheelsTaskType ∗ turnWheelsTask ()

  Manage the turning of the wheels.
- void turnWheelByAngleTask (Side wheel)
- void turnWheelsSynchronizedTask ()
- void turnWheelsSynchronizedByAngleTask ()
- void turnWheelByAngleInTimeTask (Side wheel)
- void turnArmuroTask (Side wheel)

6.28.1 Function Documentation

6.28.1.1 turnArmuroTask()

```
void turnArmuroTask (
             Side wheel )
```

6.28.1.2 turnWheelByAngleInTimeTask()

```
void turnWheelByAngleInTimeTask (
             Side wheel )
```

6.28.1.3 turnWheelByAngleTask()

```
void turnWheelByAngleTask (
            Side wheel )
```

6.28.1.4 turnWheelsSynchronizedByAngleTask()

```
void turnWheelsSynchronizedByAngleTask ( )
```

6.28.1.5 turnWheelsSynchronizedTask()

```
void turnWheelsSynchronizedTask ( )
```

6.29 wheels.h

Go to the documentation of this file.

```
#ifndef _WHEELS_H_
#define _WHEELS_H_

#include <armuro.h>

typedef enum TurnWheelsTaskType {
    NONE = 0,
    ANGLE = 1,
    SPEED = 2,
    SYNCHRONIZED = 3,
    TIMED_ANGLE = 4,
    SYNCHRONIZED_ANGLE = 5,
    TURN = 6
} TurnWheelsTaskType;

void stopWheel(Side wheel);

void turnWheelByAngle(Side wheel, int angle, int speed);

void turnWheelByAngleInTime(Side wheel, int angle, int time);

void turnWheelWithSpeed(Side wheel, int speed);

void turnWheelsSynchronized(int leftSpeed, int rightSpeed);

void turnWheelsSynchronizedByAngle(int leftSpeed, int rightSpeed, int rightAngle, uint8_t softStart);

void turnArmuroInTime(int angle, int time);

void turnArmuro(int angle);

TurnWheelsTaskType* turnWheelsTask();

void turnWheelByAngleTask(Side wheel);

void turnWheelsSynchronizedTask();

void turnWheelsSynchronizedByAngleTask();

void turnWheelByAngleInTimeTask(Side wheel);

void turnArmuroTask(Side wheel);

#endif
```

Index