

## Armuro Parcours

Generated by Doxygen 1.9.5



<b>1 Module Index</b>	<b>1</b>
1.1 Modules	1
<b>2 Data Structure Index</b>	<b>3</b>
2.1 Data Structures	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Module Documentation</b>	<b>7</b>
4.1 Armuro Hardware	7
4.1.1 Detailed Description	8
4.1.2 Function Documentation	8
4.1.2.1 angleToDistance()	8
4.1.2.2 checkSwitchesPressed()	8
4.1.2.3 didReadSensors()	9
4.1.2.4 didReadWheelEncoder()	9
4.1.2.5 distanceToAngle()	9
4.1.2.6 getAngleForWheel()	10
4.1.2.7 getAngleForWheels()	10
4.1.2.8 getLineSensorReadings()	10
4.1.2.9 getRawLineSensorReadings()	11
4.1.2.10 map()	11
4.1.2.11 mapLineSensorReadingToRange()	11
4.1.2.12 print()	12
4.1.2.13 resetAngleMeasurement()	12
4.1.2.14 setLED()	12
4.1.2.15 setRearLED()	13
4.1.2.16 speedDifferenceForRadius()	13
4.1.2.17 stopAngleMeasurement()	13
4.1.2.18 stopMotor()	14
4.1.2.19 turnMotor()	14
4.2 Blink LED	14
4.2.1 Detailed Description	14
4.2.2 Function Documentation	15
4.2.2.1 blinkLED()	15
4.2.2.2 blinkLEDTask()	15
4.2.2.3 stopBlinkingLED()	15
4.3 Calibrate Robot	15
4.3.1 Detailed Description	16
4.3.2 Function Documentation	16
4.3.2.1 calibrate()	16
4.3.2.2 calibrateTask()	16

4.3.2.3 readBlackLineSensors()	16
4.3.2.4 readWhiteLineSensors()	17
4.4 Line Follow	17
4.4.1 Detailed Description	18
4.4.2 Typedef Documentation	18
4.4.2.1 CheckLineResult	18
4.4.2.2 FollowLineResult	18
4.4.2.3 SearchLineResult	18
4.4.2.4 SearchLineState	18
4.4.3 Enumeration Type Documentation	18
4.4.3.1 CheckLineResult	18
4.4.3.2 FollowLineResult	19
4.4.3.3 SearchLineResult	19
4.4.3.4 SearchLineState	19
4.4.4 Function Documentation	20
4.4.4.1 checkForLine()	20
4.4.4.2 followLine()	20
4.4.4.3 followLineTask()	20
4.4.4.4 searchLine()	21
4.4.4.5 searchLineTask()	21
4.5 Obstacle Avoidance	21
4.5.1 Detailed Description	22
4.5.2 Typedef Documentation	22
4.5.2.1 ObstacleAvoidanceConfig	22
4.5.2.2 ObstacleAvoidanceState	22
4.5.3 Enumeration Type Documentation	22
4.5.3.1 ObstacleAvoidanceState	22
4.5.4 Function Documentation	23
4.5.4.1 avoidObstacle()	23
4.5.4.2 avoidObstacleTask()	23
4.5.4.3 checkForObstacle()	23
4.5.4.4 configureObstacleAvoidance()	24
4.6 Parcour	25
4.6.1 Detailed Description	25
4.6.2 Typedef Documentation	26
4.6.2.1 StateMachine	26
4.6.3 Enumeration Type Documentation	26
4.6.3.1 StateMachine	26
4.6.4 Function Documentation	26
4.6.4.1 driveParcour()	26
4.6.4.2 startParcour()	27
4.6.5 Variable Documentation	27

4.6.5.1 currentState	27
4.6.5.2 nextState	27
4.6.5.3 state	27
4.7 PID Controller	27
4.7.1 Detailed Description	28
4.7.2 Typedef Documentation	28
4.7.2.1 PIDConfig	28
4.7.3 Function Documentation	28
4.7.3.1 calculatePIDOutput()	28
4.7.3.2 initPID()	29
4.8 Trajectory	29
4.8.1 Detailed Description	30
4.8.2 Typedef Documentation	30
4.8.2.1 TrajectoryStateMachine	30
4.8.3 Enumeration Type Documentation	30
4.8.3.1 TrajectoryStateMachine	30
4.8.4 Function Documentation	30
4.8.4.1 driveTrajectoryTask()	30
4.8.4.2 startTrajectory()	31
4.9 Wheels	31
4.9.1 Detailed Description	32
4.9.2 Typedef Documentation	32
4.9.2.1 TurnWheelsTaskType	32
4.9.3 Enumeration Type Documentation	32
4.9.3.1 TurnWheelsTaskType	32
4.9.4 Function Documentation	32
4.9.4.1 stopWheel()	32
4.9.4.2 turnArmuro()	33
4.9.4.3 turnArmuroInTime()	33
4.9.4.4 turnWheelByAngle()	33
4.9.4.5 turnWheelByAngleInTime()	34
4.9.4.6 turnWheelsSynchronized()	34
4.9.4.7 turnWheelsSynchronizedByAngle()	34
4.9.4.8 turnWheelsTask()	35
4.9.4.9 turnWheelWithSpeed()	35
<b>5 Data Structure Documentation</b>	<b>37</b>
5.1 ObstacleAvoidanceConfig Struct Reference	37
5.1.1 Detailed Description	37
5.1.2 Field Documentation	37
5.1.2.1 attackAngle	37
5.1.2.2 backOffDistance	38

5.1.2.3 circleRadius . . . . .	38
5.1.2.4 distanceToDrive . . . . .	38
5.2 PIDConfig Struct Reference . . . . .	38
5.2.1 Detailed Description . . . . .	38
5.2.2 Field Documentation . . . . .	39
5.2.2.1 d_gain . . . . .	39
5.2.2.2 i_gain . . . . .	39
5.2.2.3 i_relax . . . . .	39
5.2.2.4 integral . . . . .	39
5.2.2.5 max_i . . . . .	39
5.2.2.6 old_input . . . . .	39
5.2.2.7 p_gain . . . . .	39
5.3 WheelAngle Struct Reference . . . . .	40
5.3.1 Field Documentation . . . . .	40
5.3.1.1 left . . . . .	40
5.3.1.2 leftTicks . . . . .	40
5.3.1.3 right . . . . .	40
5.3.1.4 rightTicks . . . . .	40
5.4 WheelAngleListItem Struct Reference . . . . .	41
5.4.1 Field Documentation . . . . .	41
5.4.1.1 angle . . . . .	41
5.4.1.2 next . . . . .	41
5.4.1.3 prev . . . . .	41
<b>6 File Documentation</b> . . . . .	<b>43</b>
6.1 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/armuro/armuro.c File Reference . . . . .	43
6.1.1 Macro Definition Documentation . . . . .	45
6.1.1.1 LEFT_ENCODER_HIGH_THRESHOLD . . . . .	45
6.1.1.2 LEFT_ENCODER_LOW_THRESHOLD . . . . .	45
6.1.1.3 MAX_PWM . . . . .	45
6.1.1.4 RIGHT_ENCODER_HIGH_THRESHOLD . . . . .	45
6.1.1.5 RIGHT_ENCODER_LOW_THRESHOLD . . . . .	45
6.1.2 Typedef Documentation . . . . .	45
6.1.2.1 WheelAngleListItem . . . . .	46
6.1.3 Function Documentation . . . . .	46
6.1.3.1 initMotors() . . . . .	46
6.1.3.2 schmittTrigger() . . . . .	46
6.1.3.3 startAngleMeasurement() . . . . .	46
6.1.4 Variable Documentation . . . . .	46
6.1.4.1 maxLineSensorValues . . . . .	47
6.1.4.2 minLineSensorValues . . . . .	47
6.1.4.3 wheelAngleList . . . . .	47

6.1.4.4 wheelAngleListEnd . . . . .	47
6.1.4.5 wheelEncoderOldValues . . . . .	47
6.1.4.6 wheelEncoderTicksCount . . . . .	47
6.2 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/armuro/armuro.h File Reference . .	48
6.2.1 Macro Definition Documentation . . . . .	49
6.2.1.1 ARMURO_LENGTH . . . . .	49
6.2.1.2 BACKWARD . . . . .	50
6.2.1.3 FORWARD . . . . .	50
6.2.1.4 HIGH . . . . .	50
6.2.1.5 LOW . . . . .	50
6.2.1.6 MAX_LEFT_LINE_SENSOR_VALUE . . . . .	50
6.2.1.7 MAX_MIDDLE_LINE_SENSOR_VALUE . . . . .	50
6.2.1.8 MAX_RIGHT_LINE_SENSOR_VALUE . . . . .	50
6.2.1.9 MIN_ANGLE . . . . .	50
6.2.1.10 MIN_LEFT_LINE_SENSOR_VALUE . . . . .	51
6.2.1.11 MIN_MIDDLE_LINE_SENSOR_VALUE . . . . .	51
6.2.1.12 MIN_RIGHT_LINE_SENSOR_VALUE . . . . .	51
6.2.1.13 TURN_CIRCUMFERENCE . . . . .	51
6.2.1.14 WHEEL_CIRCUMFERENCE . . . . .	51
6.2.1.15 WHEEL_DISTANCE . . . . .	51
6.2.2 Enumeration Type Documentation . . . . .	51
6.2.2.1 Side . . . . .	51
6.2.3 Function Documentation . . . . .	52
6.2.3.1 initMotors() . . . . .	52
6.2.3.2 startAngleMeasurement() . . . . .	52
6.2.4 Variable Documentation . . . . .	52
6.2.4.1 buffer . . . . .	52
6.2.4.2 maxLineSensorValues . . . . .	52
6.2.4.3 minLineSensorValues . . . . .	53
6.2.4.4 wheelEncoderTicksCount . . . . .	53
6.3 armuro.h . . . . .	53
6.4 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/blinkLED/blinkLED.c File Reference	54
6.4.1 Enumeration Type Documentation . . . . .	54
6.4.1.1 LEDState . . . . .	54
6.4.2 Variable Documentation . . . . .	55
6.4.2.1 ledState . . . . .	55
6.4.2.2 ledTimeInterval . . . . .	55
6.4.2.3 taskLEDTimeout . . . . .	55
6.5 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/blinkLED/blinkLED.h File Reference	55
6.6 blinkLED.h . . . . .	56
6.7 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/calibrate/calibrate.c File Reference	56
6.7.1 Enumeration Type Documentation . . . . .	56

6.7.1.1 CalibrateState . . . . .	56
6.7.2 Function Documentation . . . . .	57
6.7.2.1 startCalibrate() . . . . .	57
6.7.3 Variable Documentation . . . . .	57
6.7.3.1 calibrateState . . . . .	57
6.7.3.2 calibrateStateState . . . . .	57
6.7.3.3 nextCalibrateState . . . . .	57
6.8 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/calibrate/calibrate.h File Reference . . . . .	57
6.9 calibrate.h . . . . .	58
6.10 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/lineFollow/lineFollow.c File Reference . . . . .	58
6.10.1 Macro Definition Documentation . . . . .	59
6.10.1.1 BLACK_THRESHOLD . . . . .	59
6.10.1.2 WHITE_THRESHOLD . . . . .	59
6.10.2 Variable Documentation . . . . .	60
6.10.2.1 baseLineSpeed . . . . .	60
6.10.2.2 followLinePID . . . . .	60
6.10.2.3 lastLineValues . . . . .	60
6.10.2.4 lastState . . . . .	60
6.10.2.5 lineFollowTimeout . . . . .	60
6.10.2.6 nextSearchState . . . . .	60
6.10.2.7 searchLineState . . . . .	60
6.10.2.8 searchLineStateState . . . . .	61
6.11 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/lineFollow/lineFollow.h File Reference . . . . .	61
6.12 lineFollow.h . . . . .	62
6.13 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/obstacleAvoidance/obstacleAvoidance.c File Reference . . . . .	62
6.13.1 Macro Definition Documentation . . . . .	63
6.13.1.1 ATTACK_ANGLE . . . . .	63
6.13.1.2 OBSTACLE_RADIUS . . . . .	63
6.13.1.3 SAFETY_DISTANCE . . . . .	64
6.13.2 Variable Documentation . . . . .	64
6.13.2.1 nextObstacleAvoidanceState . . . . .	64
6.13.2.2 obstacleAvoidanceConfig . . . . .	64
6.13.2.3 obstacleAvoidanceState . . . . .	64
6.13.2.4 obstacleAvoidanceStateState . . . . .	64
6.14 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/obstacleAvoidance/obstacleAvoidance.h File Reference . . . . .	64
6.15 obstacleAvoidance.h . . . . .	65
6.16 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/parcour/parcour.c File Reference . . . . .	65
6.16.1 Function Documentation . . . . .	66
6.16.1.1 avoidingObstacle() . . . . .	66



6.16.1.2	<a href="#">calibrateArmuro()</a>	66
6.16.1.3	<a href="#">driveTrajectory()</a>	66
6.16.1.4	<a href="#">lineFollow()</a>	66
6.16.1.5	<a href="#">overcomeGap()</a>	66
6.16.1.6	<a href="#">searchTheLine()</a>	66
6.17	<a href="#">/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/parcour/parcour.h File Reference</a>	67
6.18	<a href="#">parcour.h</a>	67
6.19	<a href="#">/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/pidController/pidController.c File Reference</a>	67
6.20	<a href="#">/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/pidController/pidController.h File Reference</a>	67
6.21	<a href="#">pidController.h</a>	68
6.22	<a href="#">/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/stateMachine/stateMachine.h File Reference</a>	68
6.22.1	<a href="#">Typedef Documentation</a>	68
6.22.1.1	<a href="#">State</a>	69
6.22.2	<a href="#">Enumeration Type Documentation</a>	69
6.22.2.1	<a href="#">State</a>	69
6.23	<a href="#">stateMachine.h</a>	69
6.24	<a href="#">/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/trajectory/trajectory.c File Reference</a>	69
6.24.1	<a href="#">Function Documentation</a>	70
6.24.1.1	<a href="#">driveFirstTrajectoryPart()</a>	70
6.24.1.2	<a href="#">driveSecondTrajectoryPart()</a>	70
6.24.1.3	<a href="#">driveThirdTrajectoryPart()</a>	70
6.24.1.4	<a href="#">turnToSecondTrajectoryPart()</a>	71
6.24.1.5	<a href="#">turnToThirdTrajectoryPart()</a>	71
6.24.2	<a href="#">Variable Documentation</a>	71
6.24.2.1	<a href="#">currentTrajectoryState</a>	71
6.24.2.2	<a href="#">nextTrajectoryState</a>	71
6.24.2.3	<a href="#">trajectoryStateState</a>	71
6.25	<a href="#">/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/trajectory/trajectory.h File Reference</a>	71
6.26	<a href="#">trajectory.h</a>	72
6.27	<a href="#">/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/wheels/wheels.c File Reference</a>	72
6.27.1	<a href="#">Macro Definition Documentation</a>	73
6.27.1.1	<a href="#">MAX_ROTATION_RATE</a>	73
6.27.2	<a href="#">Function Documentation</a>	73
6.27.2.1	<a href="#">setupSynchronizedPID()</a>	73
6.27.2.2	<a href="#">turnArmuroTask()</a>	73
6.27.2.3	<a href="#">turnWheelByAngleInTimeTask()</a>	73
6.27.2.4	<a href="#">turnWheelByAngleTask()</a>	74
6.27.2.5	<a href="#">turnWheelsSynchronizedByAngleTask()</a>	74
6.27.2.6	<a href="#">turnWheelsSynchronizedTask()</a>	74
6.27.3	<a href="#">Variable Documentation</a>	74

6.27.3.1 angleSetpoint . . . . .	74
6.27.3.2 angleTimeout . . . . .	74
6.27.3.3 currentSpeedSetpoint . . . . .	74
6.27.3.4 oldAngle . . . . .	74
6.27.3.5 rotationRateSetpoint . . . . .	75
6.27.3.6 speedSetpoint . . . . .	75
6.27.3.7 synchronizeWheelsPID . . . . .	75
6.27.3.8 synchronizeWheelsTimeout . . . . .	75
6.27.3.9 turningWheels . . . . .	75
6.27.3.10 wheelsPID . . . . .	75
6.28 /Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/wheels/wheels.h File Reference . . . . .	75
6.28.1 Function Documentation . . . . .	76
6.28.1.1 turnArmuroTask() . . . . .	76
6.28.1.2 turnWheelByAngleInTimeTask() . . . . .	76
6.28.1.3 turnWheelByAngleTask() . . . . .	77
6.28.1.4 turnWheelsSynchronizedByAngleTask() . . . . .	77
6.28.1.5 turnWheelsSynchronizedTask() . . . . .	77
6.29 wheels.h . . . . .	77
<b>Index</b>	<b>79</b>

# Chapter 1

## Module Index

### 1.1 Modules

Here is a list of all modules:

Armuro Hardware . . . . .	7
Blink LED . . . . .	14
Calibrate Robot . . . . .	15
Line Follow . . . . .	17
Obstacle Avoidance . . . . .	21
Parcour . . . . .	25
PID Controller . . . . .	27
Trajectory . . . . .	29
Wheels . . . . .	31



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">ObstacleAvoidanceConfig</a>	
The configuration for the obstacle avoidance . . . . .	37
<a href="#">PIDConfig</a>	
Configuration for the PID controller . . . . .	38
<a href="#">WheelAngle</a> . . . . .	40
<a href="#">WheelAngleListItem</a> . . . . .	41



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/armuro/ <a href="#">armuro.c</a> . . . . .	43
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/armuro/ <a href="#">armuro.h</a> . . . . .	48
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/blinkLED/ <a href="#">blinkLED.c</a> . . . . .	54
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/blinkLED/ <a href="#">blinkLED.h</a> . . . . .	55
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/calibrate/ <a href="#">calibrate.c</a> . . . . .	56
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/calibrate/ <a href="#">calibrate.h</a> . . . . .	57
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/lineFollow/ <a href="#">lineFollow.c</a> . . . . .	58
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/lineFollow/ <a href="#">lineFollow.h</a> . . . . .	61
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/obstacleAvoidance/ <a href="#">obstacleAvoidance.c</a> 62	
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/obstacleAvoidance/ <a href="#">obstacleAvoidance.h</a> 64	
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/parcour/ <a href="#">parcour.c</a> . . . . .	65
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/parcour/ <a href="#">parcour.h</a> . . . . .	67
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/pidController/ <a href="#">pidController.c</a> . . . . .	67
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/pidController/ <a href="#">pidController.h</a> . . . . .	67
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/stateMachine/ <a href="#">stateMachine.h</a> . . . . .	68
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/trajectory/ <a href="#">trajectory.c</a> . . . . .	69
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/trajectory/ <a href="#">trajectory.h</a> . . . . .	71
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/wheels/ <a href="#">wheels.c</a> . . . . .	72
/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/wheels/ <a href="#">wheels.h</a> . . . . .	75





## 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

<i>angle</i>	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

<i>side</i>	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**

<i>values</i>	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**

<i>leftValue</i>	the value of the left wheel encoder
<i>rightValue</i>	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**

<i>distance</i>	the distance for which the angle should be returned
-----------------	---

**Returns**

the angle for the distance

#### 4.1.2.6 getAngleForWheel()

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

Get the Angle (in degrees) for the wheel

##### Parameters

<i>wheel</i>	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

<i>leftAngle</i>	a pointer to a variable in which the left angle should be stored
<i>rightAngle</i>	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

<i>left</i>	a pointer to a variable in which the left sensor reading should be stored
<i>middle</i>	a pointer to a variable in which the middle sensor reading should be stored
<i>right</i>	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

<i>left</i>	a pointer to a variable in which the left sensor reading should be stored
<i>middle</i>	a pointer to a variable in which the middle sensor reading should be stored
<i>right</i>	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

<i>x</i>	the value to map
<i>in_min</i>	the minimum value of the input range
<i>in_max</i>	the maximum value of the input range
<i>out_min</i>	the minimum value of the output range
<i>out_max</i>	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**

<i>value</i>	the value to map
<i>side</i>	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**

<i>format</i>	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**

<i>wheel</i>	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

<i>led</i>	the side on which the led should be controlled
<i>state</i>	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

<i>state</i>	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

<i>radius</i>	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

<i>angle</i>	the angle struct to stop measuring for
--------------	--

#### 4.1.2.18 stopMotor()

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

Stop the motor.

##### Parameters

<i>motor</i>	the motor to stop
--------------	-------------------

#### 4.1.2.19 turnMotor()

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

Activate the motor.

##### Parameters

<i>motor</i>	the motor to activate
<i>direction</i>	the direction in which the motor should turn
<i>speed</i>	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

<i>side</i>	the side to blink the LED on
<i>timeInterval</i>	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

<i>side</i>	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

<i>speed</i>	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**

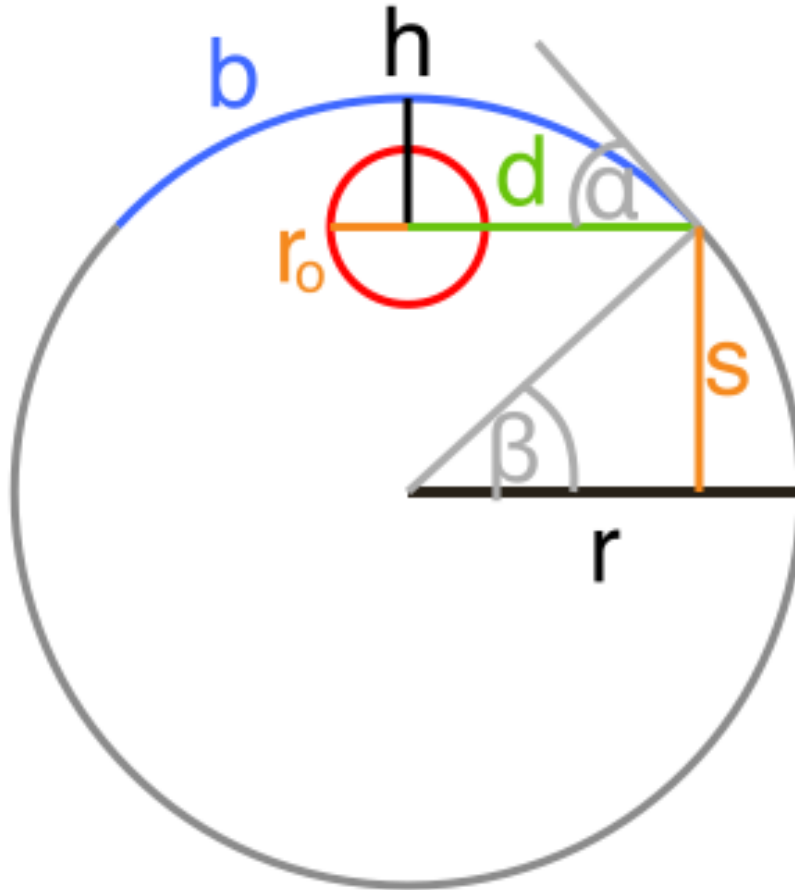
true if there is an obstacle, false otherwise

#### 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.



$$\begin{aligned}\beta &= 90 - \alpha \\ s &= \sin(\beta) \cdot r \\ h &= r - s = r_o + \delta + \frac{1}{2}w \\ r &= \frac{1}{1 - \sin(\beta)} \cdot h \\ d &= \cos(\beta) \cdot r \\ b &= r \cdot (\pi - 2 \cdot \sin(\beta))\end{aligned}$$

where  $\alpha$  is the attack angle,  
 $\delta$  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

<code>obstacleRadius</code>	the radius of the obstacle
<code>attackAngle</code>	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

<i>setpoint</i>	the desired value
<i>input</i>	the current value
<i>config</i>	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

<i>p_gain</i>	the gain for the proportional part
<i>i_gain</i>	the gain for the integral part
<i>d_gain</i>	the gain for the derivative part
<i>max_i</i>	the maximum value for the integral part
<i>i_relax</i>	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\\_PART](#) = 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

<i>wheel</i>	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

<i>angle</i>	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

<i>angle</i>	the angle to turn by (positive for left, negative for right)
<i>time</i>	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

<i>wheel</i>	the wheel to turn
<i>angle</i>	the angle to turn the wheel by (positive for forward, negative for backward)
<i>speed</i>	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

<i>wheel</i>	the wheel to turn
<i>angle</i>	the angle to turn the wheel by (positive for forward, negative for backward)
<i>time</i>	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

<i>leftSpeed</i>	the speed of the left wheel
<i>rightSpeed</i>	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

<i>leftSpeed</i>	the speed of the left wheel
------------------	-----------------------------

**Parameters**

<i>rightSpeed</i>	the speed of the right wheel
<i>rightAngle</i>	the angle the right wheel should be turned by
<i>softStart</i>	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**

<i>wheel</i>	the wheel to turn
<i>speed</i>	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

<i>side</i>	
<i>value</i>	

#### 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.](#)

```
1 #ifndef __ARMURO_H_
2 #define __ARMURO_H_
3
4 #define FORWARD 0
5 #define BACKWARD 1
6
7 #define HIGH 1
8 #define LOW 0
9
10 #define WHEEL_CIRCUMFERENCE 12.566
11 #define TURN_CIRCUMFERENCE 24.504422698
12 #define WHEEL_DISTANCE 7.8
13 #define ARMURO_LENGTH 9
14
15 #define MIN_ANGLE 15
16
17 #define MAX_LEFT_LINE_SENSOR_VALUE 3380
18 #define MAX_MIDDLE_LINE_SENSOR_VALUE 3380
19 #define MAX_RIGHT_LINE_SENSOR_VALUE 2900
20 #define MIN_LEFT_LINE_SENSOR_VALUE 1250
21 #define MIN_MIDDLE_LINE_SENSOR_VALUE 1120
22 #define MIN_RIGHT_LINE_SENSOR_VALUE 170
23
24 #include <stdint.h>
25 #include <stdarg.h>
26
27 extern uint32_t minLineSensorValues[3];
28 extern uint32_t maxLineSensorValues[3];
29
30 extern uint16_t wheelEncoderTicksCount[2];
31 extern uint32_t buffer[6];
32
33 typedef enum {
34     RIGHT = 0,
35     LEFT = 1,
36     MIDDLE = 3
37 } Side;
38
39 typedef struct {
40     int left;
41     int right;
42
43     int leftTicks;
44     int rightTicks;
45 } WheelAngle;
46
47 void print(char* format, ...);
48
49 int32_t map(int32_t x, int32_t in_min, int32_t in_max, int32_t out_min, int32_t out_max);
50
51 void initMotors();
52
53 void turnMotor(Side motor, int direction, int speed);
54
55 void stopMotor(Side motor);
56
57 void setLED(Side led, int state);
58
59 void setRearLED(int state);
60
61 void didReadSensors(uint32_t* values);
```

```

121
129 void didReadWheelEncoder(uint32_t leftValue, uint32_t rightValue);
130
137 void resetAngleMeasurement(Side wheel);
138
144 WheelAngle* startAngleMeasurement();
145
152 void stopAngleMeasurement(WheelAngle* angle);
153
161 int getAngleForWheel(Side wheel);
162
170 void getAngleForWheels(int* leftAngle, int* rightAngle);
171
180 void getRawLineSensorReadings(uint32_t* left, uint32_t* middle, uint32_t* right);
181
190 void getLineSensorReadings(uint32_t* left, uint32_t* middle, uint32_t* right);
191
200 uint32_t mapLineSensorReadingToRange(uint32_t value, Side side);
201
209 int distanceToAngle(double distance);
210
218 double angleToDistance(int angle);
219
227 double speedDifferenceForRadius(double radius);
228
236 uint8_t checkSwitchesPressed(Side* side);
237
238 #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.](#)

```
1 #ifndef _BLINK_LED_H_
2 #define _BLINK_LED_H_
3
4 #include "armuro.h"
5
18 void blinkLED(Side side, uint16_t timeInterval);
19
26 void stopBlinkingLED(Side side);
27
34 void blinkLEDTask();
35
36 #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.](#)

```
1 #ifndef _CALIBRATE_H_
2 #define _CALIBRATE_H_
3
4 #include "armuro.h"
5 #include "stateMachine.h"
6
16 void calibrate();
23 State calibrateTask();
24
29 void readWhiteLineSensors();
34 void readBlackLineSensors();
35
36 #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.](#)

```

1  #ifndef LINE_FOLLOW_H
2  #define LINE_FOLLOW_H
3
13 typedef enum SearchLineResult {
15     SEARCHING = 0,
17     FOUND = 1,
19     LOST = -1,
21     END_OF_LINE = 2
22 } SearchLineResult;
23
28 typedef enum CheckLineResult {
30     ON_LINE = 0,
32     OFF_LINE = -1,
34     ALL_BLACK = 1
35 } CheckLineResult;
36
41 typedef enum FollowLineResult {
43     FOLLOWING = 0,
45     LOST_LINE = -1,
47     ALL_LINE = 1
48 } FollowLineResult;
49
57 void followLine(int speed);
58
66 FollowLineResult followLineTask();
67
74 CheckLineResult checkForLine();
75
81 void searchLine();
82
92 SearchLineResult searchLineTask();
93
94 #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.](#)

```
1 #ifndef _OBSTACLE_AVOIDANCE_H_
2 #define _OBSTACLE_AVOIDANCE_H_
3
4 #include "stateMachine.h"
5 #include "armuro.h"
6
19 uint8_t checkForObstacle();
20
27 void avoidObstacle();
28
36 State avoidObstacleTask();
37
38 #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.](#)

```
1 #ifndef __PARCOUR_H_
2 #define __PARCOUR_H_
3
16 void startParcour();
17
24 void driveParcour();
25
26 #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.](#)

```

1 #ifndef _PID_CONTROLLER_H_
2 #define _PID_CONTROLLER_H_
3
4 #include <stdint.h>
5
15 typedef struct PIDConfig {
16     double p_gain;
17     double i_gain;
18     double d_gain;
19     double max_i;
20     double i_relax;
21     double old_input;
22     double integral;
23 } PIDConfig;
24
36 PIDConfig initPID(double p_gain, double i_gain, double d_gain, double max_i, double i_relax);
37
47 double calculatePIDOutput(double setpoint, double input, PIDConfig* config);
48
49 #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.](#)

```
1 #ifndef __STATE_MACHINE_H_
2 #define __STATE_MACHINE_H_
3
4 typedef enum State {
5     READY = 0,
6     RUNNING = 1,
7     FINISHED = 2
8 } State;
9
10 #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_PART` = 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.](#)

```
1 #ifndef _TRAJECTORY_H_
2 #define _TRAJECTORY_H_
3
4 #include "stateMachine.h"
5
16 void startTrajectory();
17
25 State driveTrajectoryTask();
26
27 #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.](#)

```
1 #ifndef _WHEELS_H_
2 #define _WHEELS_H_
3
4 #include <armuro.h>
5
15 typedef enum TurnWheelsTaskType {
17     NONE = 0,
19     ANGLE = 1,
21     SPEED = 2,
23     SYNCHRONIZED = 3,
25     TIMED_ANGLE = 4,
27     SYNCHRONIZED_ANGLE = 5,
29     TURN = 6
30 } TurnWheelsTaskType;
31
38 void stopWheel(Side wheel);
39
48 void turnWheelByAngle(Side wheel, int angle, int speed);
49
58 void turnWheelByAngleInTime(Side wheel, int angle, int time);
59
67 void turnWheelWithSpeed(Side wheel, int speed);
68
78 void turnWheelsSynchronized(int leftSpeed, int rightSpeed);
79
89 void turnWheelsSynchronizedByAngle(int leftSpeed, int rightSpeed, int rightAngle, uint8_t softStart);
90
98 void turnArmuroInTime(int angle, int time);
99
106 void turnArmuro(int angle);
107
116 TurnWheelsTaskType* turnWheelsTask();
117
118 void turnWheelByAngleTask(Side wheel);
119
120 void turnWheelsSynchronizedTask();
121
122 void turnWheelsSynchronizedByAngleTask();
123
124 void turnWheelByAngleInTimeTask(Side wheel);
125
126 void turnArmuroTask(Side wheel);
127
128 #endif
```





# Index

/Users/dennis/CloudStation/Studium/Mobile Roboter/Software/lib/armuro.c, 74  
43  
angleToDistance  
armuro.h, 74  
armuro Hardware, 8  
Armuro Hardware, 7  
angleToDistance, 74  
checkSwitchesPressed, 8  
didReadWheelEncoder, 9  
didReadWheelEncoder, 9  
discreteAngle, 10  
getAngleForWheel, 10  
getAngleForWheel, 10  
getLineSensorReadings, 10  
getLineSensorReadings, 11  
map, 11  
mapFromSensorReadingToRange, 11  
print, 12  
resetAngleMeasurement, 13  
setLED, 12  
setLED, 12  
speedDifferenceForRadius, 13  
stopAngleMeasurement, 13  
stopMotor, 14  
upMotor, 14  
armuro.c  
armuro.h  
initMotors, 46  
pidController.c, 46  
LEFT\_ENCODER\_HIGH\_THRESHOLD, 45  
LEFT\_ENCODER\_LOW\_THRESHOLD, 45  
MAX\_PWM, 45  
minLineSensorValues, 46  
minLineSensorValues, 47  
RIGHT\_ENCODER\_HIGH\_THRESHOLD, 45  
RIGHT\_ENCODER\_LOW\_THRESHOLD, 45  
startAngleMeasurement, 46  
wheelAngleList, 47  
wheelAngleListEnd, 47  
wheelAngleListEnd, 45  
wheelEncoderOldValues, 47  
wheelEncoderTicksCount, 47  
armuro.h  
ARMURO\_LENGTH, 49  
BACKWARD, 49  
buffer, 52  
FORWARD, 50  
HIGH, 50  
initMotors, 52  
LEFT, 52  
LOW, 50  
MAX\_LEFT\_LINE\_SENSOR\_VALUE, 50  
ALL\_BLACK  
Line Follow, 19  
ALL\_LINE  
Line Follow, 19  
ANGLE  
Wheels, 32  
angle  
WheelAngleListItem, 41  
angleSetpoint  
wheels.c, 74  
angleTimeout

- MAX\_MIDDLE\_LINE\_SENSOR\_VALUE, 50
- MAX\_RIGHT\_LINE\_SENSOR\_VALUE, 50
- maxLineSensorValues, 52
- MIDDLE, 52
- MIN\_ANGLE, 50
- MIN\_LEFT\_LINE\_SENSOR\_VALUE, 50
- MIN\_MIDDLE\_LINE\_SENSOR\_VALUE, 51
- MIN\_RIGHT\_LINE\_SENSOR\_VALUE, 51
- minLineSensorValues, 52
- RIGHT, 52
- Side, 51
- startAngleMeasurement, 52
- TURN\_CIRCUMFERENCE, 51
- WHEEL\_CIRCUMFERENCE, 51
- WHEEL\_DISTANCE, 51
- wheelEncoderTicksCount, 53
- ARMURO\_LENGTH
  - armuro.h, 49
- ATTACK\_ANGLE
  - obstacleAvoidance.c, 63
- attackAngle
  - ObstacleAvoidanceConfig, 37
- AVOID\_OBSTACLE
  - Parcour, 26
- avoidingObstacle
  - parcour.c, 66
- avoidObstacle
  - Obstacle Avoidance, 23
- avoidObstacleTask
  - Obstacle Avoidance, 23
- BACK\_OFF
  - Obstacle Avoidance, 23
- backOffDistance
  - ObstacleAvoidanceConfig, 37
- BACKWARD
  - armuro.h, 49
- baseLineSpeed
  - lineFollow.c, 60
- BLACK\_THRESHOLD
  - lineFollow.c, 59
- Blink LED, 14
  - blinkLED, 15
  - blinkLEDTask, 15
  - stopBlinkingLED, 15
- blinkLED
  - Blink LED, 15
- blinkLED.c
  - LEDState, 54
  - ledState, 55
  - ledTimeInterval, 55
  - OFF, 55
  - ON, 55
  - taskLEDTimeout, 55
- blinkLEDTask
  - Blink LED, 15
- buffer
  - armuro.h, 52
- calculatePIDOutput
  - PID Controller, 28
- CALIBRATE
  - Parcour, 26
- calibrate
  - Calibrate Robot, 16
- Calibrate Robot, 15
  - calibrate, 16
  - calibrateTask, 16
  - readBlackLineSensors, 16
  - readWhiteLineSensors, 16
- calibrate.c
  - CALIBRATE\_BLACK, 57
  - CALIBRATE\_DONE, 57
  - CALIBRATE\_START, 57
  - CALIBRATE\_WHITE, 57
  - CalibrateState, 56
  - calibrateState, 57
  - calibrateStateState, 57
  - nextCalibrateState, 57
  - startCalibrate, 57
- CALIBRATE\_BLACK
  - calibrate.c, 57
- CALIBRATE\_DONE
  - calibrate.c, 57
- CALIBRATE\_START
  - calibrate.c, 57
- CALIBRATE\_WHITE
  - calibrate.c, 57
- calibrateArmuro
  - parcour.c, 66
- CalibrateState
  - calibrate.c, 56
- calibrateState
  - calibrate.c, 57
- calibrateStateState
  - calibrate.c, 57
- calibrateTask
  - Calibrate Robot, 16
- checkForLine
  - Line Follow, 20
- checkForObstacle
  - Obstacle Avoidance, 23
- CheckLineResult
  - Line Follow, 18
- checkSwitchesPressed
  - Armuro Hardware, 8
- circleRadius
  - ObstacleAvoidanceConfig, 38
- configureObstacleAvoidance
  - Obstacle Avoidance, 23
- currentSpeedSetpoint
  - wheels.c, 74
- currentState
  - Parcour, 27
- currentTrajectoryState
  - trajectory.c, 71
- d\_gain

- PIDConfig, 39
- didReadSensors
  - Armuro Hardware, 9
- didReadWheelEncoder
  - Armuro Hardware, 9
- distanceToAngle
  - Armuro Hardware, 9
- distanceToDrive
  - ObstacleAvoidanceConfig, 38
- DONE
  - Line Follow, 20
- DRIVE
  - Line Follow, 19
- DRIVE\_CIRCLE
  - Obstacle Avoidance, 23
- DRIVE\_FIRST\_TRAJECTORY\_PART
  - Trajectory, 30
- DRIVE\_SECOND\_TRAJECTORY\_PART
  - Trajectory, 30
- DRIVE\_THIRD\_TRAJECTORY\_PART
  - Trajectory, 30
- DRIVE\_TRAJECTORY
  - Parcour, 26
- driveFirstTrajectoryPart
  - trajectory.c, 70
- driveParcour
  - Parcour, 26
- driveSecondTrajectoryPart
  - trajectory.c, 70
- driveThirdTrajectoryPart
  - trajectory.c, 70
- driveTrajectory
  - parcour.c, 66
- driveTrajectoryTask
  - Trajectory, 30
- END\_OF\_LINE
  - Line Follow, 19
- FINISHED
  - stateMachine.h, 69
- FOLLOW\_LINE
  - Parcour, 26
  - Trajectory, 30
- FOLLOWING
  - Line Follow, 19
- followLine
  - Line Follow, 20
- followLinePID
  - lineFollow.c, 60
- FollowLineResult
  - Line Follow, 18, 19
- followLineTask
  - Line Follow, 20
- FORWARD
  - armuro.h, 50
- FOUND
  - Line Follow, 19
- getAngleForWheel
  - Armuro Hardware, 10
- getAngleForWheels
  - Armuro Hardware, 10
- getLineSensorReadings
  - Armuro Hardware, 10
- getRawLineSensorReadings
  - Armuro Hardware, 11
- HIGH
  - armuro.h, 50
- i\_gain
  - PIDConfig, 39
- i\_relax
  - PIDConfig, 39
- IDLE
  - Parcour, 26
- initMotors
  - armuro.c, 46
  - armuro.h, 52
- initPID
  - PID Controller, 28
- integral
  - PIDConfig, 39
- lastLineValues
  - lineFollow.c, 60
- lastState
  - lineFollow.c, 60
- LEDState
  - blinkLED.c, 54
- ledState
  - blinkLED.c, 55
- ledTimeInterval
  - blinkLED.c, 55
- LEFT
  - armuro.h, 52
- left
  - WheelAngle, 40
- LEFT\_ENCODER\_HIGH\_THRESHOLD
  - armuro.c, 45
- LEFT\_ENCODER\_LOW\_THRESHOLD
  - armuro.c, 45
- leftTicks
  - WheelAngle, 40
- Line Follow, 17
  - ALL\_BLACK, 19
  - ALL\_LINE, 19
  - checkForLine, 20
  - CheckLineResult, 18
  - DONE, 20
  - DRIVE, 19
  - END\_OF\_LINE, 19
  - FOLLOWING, 19
  - followLine, 20
  - FollowLineResult, 18, 19
  - followLineTask, 20
  - FOUND, 19

- LOST, 19
- LOST\_LINE, 19
- OFF\_LINE, 19
- ON\_LINE, 19
- SEARCHING, 19
- searchLine, 21
- SearchLineResult, 18, 19
- SearchLineState, 18, 19
- searchLineTask, 21
- TURN\_LEFT\_TO\_MIDDLE, 20
- TURN\_LEFT\_TO\_RIGHT, 20
- TURN\_RIGHT\_TO\_LEFT, 20
- TURN\_RIGHT\_TO\_MIDDLE, 20
- TURNING\_LEFT, 19
- TURNING\_RIGHT, 20
- lineFollow
  - parcour.c, 66
- lineFollow.c
  - baseLineSpeed, 60
  - BLACK\_THRESHOLD, 59
  - followLinePID, 60
  - lastLineValues, 60
  - lastState, 60
  - lineFollowTimeout, 60
  - nextSearchState, 60
  - searchLineState, 60
  - searchLineStateState, 60
  - WHITE\_THRESHOLD, 59
- lineFollowTimeout
  - lineFollow.c, 60
- LOST
  - Line Follow, 19
- LOST\_LINE
  - Line Follow, 19
- LOW
  - armuro.h, 50
- map
  - Armuro Hardware, 11
- mapLineSensorReadingToRange
  - Armuro Hardware, 11
- max\_i
  - PIDConfig, 39
- MAX\_LEFT\_LINE\_SENSOR\_VALUE
  - armuro.h, 50
- MAX\_MIDDLE\_LINE\_SENSOR\_VALUE
  - armuro.h, 50
- MAX\_PWM
  - armuro.c, 45
- MAX\_RIGHT\_LINE\_SENSOR\_VALUE
  - armuro.h, 50
- MAX\_ROTATION\_RATE
  - wheels.c, 73
- maxLineSensorValues
  - armuro.c, 46
  - armuro.h, 52
- MIDDLE
  - armuro.h, 52
- MIN\_ANGLE
  - armuro.h, 50
- MIN\_LEFT\_LINE\_SENSOR\_VALUE
  - armuro.h, 50
- MIN\_MIDDLE\_LINE\_SENSOR\_VALUE
  - armuro.h, 51
- MIN\_RIGHT\_LINE\_SENSOR\_VALUE
  - armuro.h, 51
- minLineSensorValues
  - armuro.c, 47
  - armuro.h, 52
- next
  - WheelAngleListItem, 41
- nextCalibrateState
  - calibrate.c, 57
- nextObstacleAvoidanceState
  - obstacleAvoidance.c, 64
- nextSearchState
  - lineFollow.c, 60
- nextState
  - Parcour, 27
- nextTrajectoryState
  - trajectory.c, 71
- NONE
  - Wheels, 32
- Obstacle Avoidance, 21
  - avoidObstacle, 23
  - avoidObstacleTask, 23
  - BACK\_OFF, 23
  - checkForObstacle, 23
  - configureObstacleAvoidance, 23
  - DRIVE\_CIRCLE, 23
  - OBSTACLE\_AVOIDANCE\_DONE, 23
  - ObstacleAvoidanceConfig, 22
  - ObstacleAvoidanceState, 22
  - TURN\_FROM\_OBSTACLE, 23
- OBSTACLE\_AVOIDANCE\_DONE
  - Obstacle Avoidance, 23
- OBSTACLE\_RADIUS
  - obstacleAvoidance.c, 63
- obstacleAvoidance.c
  - ATTACK\_ANGLE, 63
  - nextObstacleAvoidanceState, 64
  - OBSTACLE\_RADIUS, 63
  - obstacleAvoidanceConfig, 64
  - obstacleAvoidanceState, 64
  - obstacleAvoidanceStateState, 64
  - SAFETY\_DISTANCE, 63
- ObstacleAvoidanceConfig, 37
  - attackAngle, 37
  - backOffDistance, 37
  - circleRadius, 38
  - distanceToDrive, 38
  - Obstacle Avoidance, 22
- obstacleAvoidanceConfig
  - obstacleAvoidance.c, 64
- ObstacleAvoidanceState
  - Obstacle Avoidance, 22

- obstacleAvoidanceState
  - obstacleAvoidance.c, [64](#)
- obstacleAvoidanceStateState
  - obstacleAvoidance.c, [64](#)
- OFF
  - blinkLED.c, [55](#)
- OFF\_LINE
  - Line Follow, [19](#)
- old\_input
  - PIDConfig, [39](#)
- oldAngle
  - wheels.c, [74](#)
- ON
  - blinkLED.c, [55](#)
- ON\_LINE
  - Line Follow, [19](#)
- OVERCOME\_GAP
  - Parcour, [26](#)
- overcomeGap
  - parcour.c, [66](#)
- p\_gain
  - PIDConfig, [39](#)
- Parcour, [25](#)
  - AVOID\_OBSTACLE, [26](#)
  - CALIBRATE, [26](#)
  - currentState, [27](#)
  - DRIVE\_TRAJECTORY, [26](#)
  - driveParcour, [26](#)
  - FOLLOW\_LINE, [26](#)
  - IDLE, [26](#)
  - nextState, [27](#)
  - OVERCOME\_GAP, [26](#)
  - SEARCH\_LINE, [26](#)
  - startParcour, [26](#)
  - state, [27](#)
  - StateMachine, [26](#)
- parcour.c
  - avoidingObstacle, [66](#)
  - calibrateArmuro, [66](#)
  - driveTrajectory, [66](#)
  - lineFollow, [66](#)
  - overcomeGap, [66](#)
  - searchTheLine, [66](#)
- PID Controller, [27](#)
  - calculatePIDOutput, [28](#)
  - initPID, [28](#)
  - PIDConfig, [28](#)
- PIDConfig, [38](#)
  - d\_gain, [39](#)
  - i\_gain, [39](#)
  - i\_relax, [39](#)
  - integral, [39](#)
  - max\_i, [39](#)
  - old\_input, [39](#)
  - p\_gain, [39](#)
  - PID Controller, [28](#)
- prev
  - WheelAngleListItem, [41](#)
- print
  - Armuro Hardware, [12](#)
- readBlackLineSensors
  - Calibrate Robot, [16](#)
- readWhiteLineSensors
  - Calibrate Robot, [16](#)
- READY
  - stateMachine.h, [69](#)
- resetAngleMeasurement
  - Armuro Hardware, [12](#)
- RIGHT
  - armuro.h, [52](#)
- right
  - WheelAngle, [40](#)
- RIGHT\_ENCODER\_HIGH\_THRESHOLD
  - armuro.c, [45](#)
- RIGHT\_ENCODER\_LOW\_THRESHOLD
  - armuro.c, [45](#)
- rightTicks
  - WheelAngle, [40](#)
- rotationRateSetpoint
  - wheels.c, [74](#)
- RUNNING
  - stateMachine.h, [69](#)
- SAFETY\_DISTANCE
  - obstacleAvoidance.c, [63](#)
- schmittTrigger
  - armuro.c, [46](#)
- SEARCH\_LINE
  - Parcour, [26](#)
- SEARCHING
  - Line Follow, [19](#)
- searchLine
  - Line Follow, [21](#)
- SearchLineResult
  - Line Follow, [18, 19](#)
- SearchLineState
  - Line Follow, [18, 19](#)
- searchLineState
  - lineFollow.c, [60](#)
- searchLineStateState
  - lineFollow.c, [60](#)
- searchLineTask
  - Line Follow, [21](#)
- searchTheLine
  - parcour.c, [66](#)
- setLED
  - Armuro Hardware, [12](#)
- setRearLED
  - Armuro Hardware, [13](#)
- setupSynchronizedPID
  - wheels.c, [73](#)
- Side
  - armuro.h, [51](#)
- SPEED
  - Wheels, [32](#)
- speedDifferenceForRadius

- Armuro Hardware, 13
- speedSetpoint
  - wheels.c, 75
- startAngleMeasurement
  - armuro.c, 46
  - armuro.h, 52
- startCalibrate
  - calibrate.c, 57
- startParcour
  - Parcour, 26
- startTrajectory
  - Trajectory, 31
- State
  - stateMachine.h, 68, 69
- state
  - Parcour, 27
- StateMachine
  - Parcour, 26
- stateMachine.h
  - FINISHED, 69
  - READY, 69
  - RUNNING, 69
  - State, 68, 69
- stopAngleMeasurement
  - Armuro Hardware, 13
- stopBlinkingLED
  - Blink LED, 15
- stopMotor
  - Armuro Hardware, 14
- stopWheel
  - Wheels, 32
- SYNCHRONIZED
  - Wheels, 32
- SYNCHRONIZED\_ANGLE
  - Wheels, 32
- synchronizeWheelsPID
  - wheels.c, 75
- synchronizeWheelsTimeout
  - wheels.c, 75
- taskLEDTimeout
  - blinkLED.c, 55
- TIMED\_ANGLE
  - Wheels, 32
- Trajectory, 29
  - DRIVE\_FIRST\_TRAJECTORY\_PART, 30
  - DRIVE\_SECOND\_TRAJECTORY\_PART, 30
  - DRIVE\_THIRD\_TRAJECTORY\_PART, 30
  - driveTrajectoryTask, 30
  - FOLLOW\_LINE, 30
  - startTrajectory, 31
  - TRAJECTORY\_DONE, 30
  - TrajectoryStateMachine, 30
  - TURN\_TO\_SECOND\_TRAJECTORY\_PART, 30
  - TURN\_TO\_THIRD\_TRAJECTORY\_PART, 30
- trajectory.c
  - currentTrajectoryState, 71
  - driveFirstTrajectoryPart, 70
  - driveSecondTrajectoryPart, 70
  - driveThirdTrajectoryPart, 70
  - nextTrajectoryState, 71
  - trajectoryStateState, 71
  - turnToSecondTrajectoryPart, 70
  - turnToThirdTrajectoryPart, 71
- TRAJECTORY\_DONE
  - Trajectory, 30
- TrajectoryStateMachine
  - Trajectory, 30
- trajectoryStateState
  - trajectory.c, 71
- TURN
  - Wheels, 32
- TURN\_CIRCUMFERENCE
  - armuro.h, 51
- TURN\_FROM\_OBSTACLE
  - Obstacle Avoidance, 23
- TURN\_LEFT\_TO\_MIDDLE
  - Line Follow, 20
- TURN\_LEFT\_TO\_RIGHT
  - Line Follow, 20
- TURN\_RIGHT\_TO\_LEFT
  - Line Follow, 20
- TURN\_RIGHT\_TO\_MIDDLE
  - Line Follow, 20
- TURN\_TO\_SECOND\_TRAJECTORY\_PART
  - Trajectory, 30
- TURN\_TO\_THIRD\_TRAJECTORY\_PART
  - Trajectory, 30
- turnArmuro
  - Wheels, 33
- turnArmuroInTime
  - Wheels, 33
- turnArmuroTask
  - wheels.c, 73
  - wheels.h, 76
- TURNING\_LEFT
  - Line Follow, 19
- TURNING\_RIGHT
  - Line Follow, 20
- turningWheels
  - wheels.c, 75
- turnMotor
  - Armuro Hardware, 14
- turnToSecondTrajectoryPart
  - trajectory.c, 70
- turnToThirdTrajectoryPart
  - trajectory.c, 71
- turnWheelByAngle
  - Wheels, 33
- turnWheelByAngleInTime
  - Wheels, 34
- turnWheelByAngleInTimeTask
  - wheels.c, 73
  - wheels.h, 76
- turnWheelByAngleTask
  - wheels.c, 73
  - wheels.h, 76

- turnWheelsSynchronized
  - Wheels, [34](#)
- turnWheelsSynchronizedByAngle
  - Wheels, [34](#)
- turnWheelsSynchronizedByAngleTask
  - wheels.c, [74](#)
  - wheels.h, [77](#)
- turnWheelsSynchronizedTask
  - wheels.c, [74](#)
  - wheels.h, [77](#)
- turnWheelsTask
  - Wheels, [35](#)
- TurnWheelsTaskType
  - Wheels, [32](#)
- turnWheelWithSpeed
  - Wheels, [35](#)
- WHEEL\_CIRCUMFERENCE
  - armuro.h, [51](#)
- WHEEL\_DISTANCE
  - armuro.h, [51](#)
- WheelAngle, [40](#)
  - left, [40](#)
  - leftTicks, [40](#)
  - right, [40](#)
  - rightTicks, [40](#)
- wheelAngleList
  - armuro.c, [47](#)
- wheelAngleListEnd
  - armuro.c, [47](#)
- WheelAngleListItem, [41](#)
  - angle, [41](#)
  - armuro.c, [45](#)
  - next, [41](#)
  - prev, [41](#)
- wheelEncoderOldValues
  - armuro.c, [47](#)
- wheelEncoderTicksCount
  - armuro.c, [47](#)
  - armuro.h, [53](#)
- Wheels, [31](#)
  - ANGLE, [32](#)
  - NONE, [32](#)
  - SPEED, [32](#)
  - stopWheel, [32](#)
  - SYNCHRONIZED, [32](#)
  - SYNCHRONIZED\_ANGLE, [32](#)
  - TIMED\_ANGLE, [32](#)
  - TURN, [32](#)
  - turnArmuro, [33](#)
  - turnArmuroInTime, [33](#)
  - turnWheelByAngle, [33](#)
  - turnWheelByAngleInTime, [34](#)
  - turnWheelsSynchronized, [34](#)
  - turnWheelsSynchronizedByAngle, [34](#)
  - turnWheelsTask, [35](#)
  - TurnWheelsTaskType, [32](#)
  - turnWheelWithSpeed, [35](#)
- wheels.c
  - angleSetpoint, [74](#)
  - angleTimeout, [74](#)
  - currentSpeedSetpoint, [74](#)
  - MAX\_ROTATION\_RATE, [73](#)
  - oldAngle, [74](#)
  - rotationRateSetpoint, [74](#)
  - setupSynchronizedPID, [73](#)
  - speedSetpoint, [75](#)
  - synchronizeWheelsPID, [75](#)
  - synchronizeWheelsTimeout, [75](#)
  - turnArmuroTask, [73](#)
  - turningWheels, [75](#)
  - turnWheelByAngleInTimeTask, [73](#)
  - turnWheelByAngleTask, [73](#)
  - turnWheelsSynchronizedByAngleTask, [74](#)
  - turnWheelsSynchronizedTask, [74](#)
  - wheelsPID, [75](#)
- wheels.h
  - turnArmuroTask, [76](#)
  - turnWheelByAngleInTimeTask, [76](#)
  - turnWheelByAngleTask, [76](#)
  - turnWheelsSynchronizedByAngleTask, [77](#)
  - turnWheelsSynchronizedTask, [77](#)
- wheelsPID
  - wheels.c, [75](#)
- WHITE\_THRESHOLD
  - lineFollow.c, [59](#)