Library for Texas Instrument's Robotic System Learning Kit

Macros

#define **LS_NUM_SENSORS** 8

Total number of sensors on QTR line sensor.

#define **DST_NUM_SENSORS** 3

Total number of distance sensors.

#define **NUM_MOTORS** 2

Total number of motors.

#define **LP_LEFT_BTN** PUSH2

Represent the left push button on the launchpad.

#define **LP_RIGHT_BTN** PUSH1

Represent the right push button on the launchpad.

#define **TOTAL_BP_SW** 6

Total number of bump switches.

#define **LEFT_MOTOR** 0

Can be used to reference the left motor in the below functions.

#define **RIGHT_MOTOR** 1

Can be used to reference the right motor in the below functions.

#define **BOTH_MOTORS** 2

Can be used to reference both motors in the below functions.

#define **LEFT_SHRP_DIST** 0

Can be used to reference the left Sharp distance sensor.

#define **CENTER_SHRP_DIST** 1

Can be used to reference the center Sharp distance sensor.

#define **RIGHT_SHRP_DIST** 2

Can be used to reference the right Sharp distance sensor.

#define **MOTOR_DIR_FORWARD** 0

Can be used to reference setting the motor function to forward.

#define **MOTOR_DIR_BACKWARD** 1

Can be used to reference setting the motor function to backward.

#define **DARK_LINE** 0

Used to specify that the robot is running on a floor lighter than the line.

#define **LIGHT_LINE** 1

Used to specify that the robot is running on a floor darker than the line.

#define **readLineSensor** **readRawLineSensor**

Read line sensor values.

Functions

uint32_t **getEncoderLeftCnt** ()
Return number of encoder ticks from the left wheel.

uint32_t **getEncoderRightCnt** ()
Return number of encoder ticks from the right wheel.

void **resetLeftEncoderCnt** ()
Set the left encoder tick count to 0.

void **resetRightEncoderCnt** ()
Set the right encoder tick count to 0.

uint8_t **getLeftWheelDir** ()
Determines if the left wheel is going forward or backwards.

uint8_t **getRightWheelDir** ()
Determines if the right wheel is going forward or backwards.

void **setupRSLK** ()
Performs a variety of initialization needed for the RSLK.

uint16_t **readSharpDist** (uint8_t num)
Read distance sensor value.

int16_t **readSharpDistMM** (uint8_t num)
Read distance sensor value in millimeters.

float **readSharpDistIN** (uint8_t num)
Read distance sensor value in inches.

bool **isBumpSwitchPressed** (uint8_t num)
Return bump switch status.

uint8_t **getBumpSwitchPressed** ()
Return mask of bump switch states.

void **enableMotor** (uint8_t motorNum)
Enable motor (take it out of sleep)

void **disableMotor** (uint8_t motorNum)
Disable motor (puts the motor to sleep)

void **pauseMotor** (uint8_t motorNum)
Pause motor (put the motor to sleep while saving its speed)

void **resumeMotor** (uint8_t motorNum)
Resume motor (take the motor out of sleep and resumes its prior speed)

void **setMotorDirection** (uint8_t motorNum, uint8_t direction)
Set direction the motor will turn.

void **setMotorSpeed** (uint8_t motorNum, uint8_t speed)

	Set the motor speed.
void	**setupWaitBtn** (uint8_t btn) Configure pin as a wait to release button.
void	**setupLed** (uint8_t ledPin) Configure pin that is connected to an led.
void	**waitBtnPressed** (uint8_t btnPin, int8_t ledPin=0) Busy wait until user pushes and releases button.
void	**calibrateLineSensor** (uint8_t mode=**DARK_LINE**, uint32_t duration=500) Calibrates line sensor.
void	**readCalLineSensor** (uint16_t *calVal) Read calibrated line sensor values. Assumes calibration completed.
uint32_t	**getLinePosition** () Get line position.
void	**readRawLineSensor** (uint16_t *sensor) Read raw line sensor values.
void	**clearMinMax** (uint16_t *sensorMin, uint16_t *sensorMax) Provide default values for the sensor's Min and Max arrays.
void	**setSensorMinMax** (uint16_t *sensor, uint16_t *sensorMin, uint16_t *sensorMax) Update line sensor's min and max values array based on current data.
void	**readCalLineSensor** (uint16_t *sensor, uint16_t *calVal, uint16_t *sensorMin, uint16_t *sensorMax, uint8_t mode) Update sensor's min and max values array based on current data.
uint32_t	**getLinePosition** (uint16_t *calVal, uint8_t mode) Get line position.

Detailed Description

Macro Definition Documentation

◆ LS_NUM_SENSORS

```
#define LS_NUM_SENSORS   8
```

Total number of sensors on QTR line sensor.

Definition at line **19** of file **SimpleRSLK.h**.

◆ DST_NUM_SENSORS

```
#define DST_NUM_SENSORS   3
```

Total number of distance sensors.

Definition at line **24** of file **SimpleRSLK.h**.

◆ NUM_MOTORS

```
#define NUM_MOTORS   2
```

Total number of motors.

Definition at line **29** of file **SimpleRSLK.h**.

◆ LP_LEFT_BTN

```
#define LP_LEFT_BTN   PUSH2
```

Represent the left push button on the launchpad.

Definition at line **34** of file **SimpleRSLK.h**.

◆ LP_RIGHT_BTN

```
#define LP_RIGHT_BTN   PUSH1
```

Represent the right push button on the launchpad.

Definition at line **39** of file **SimpleRSLK.h**.

◆ TOTAL_BP_SW

#define TOTAL_BP_SW 6

Total number of bump switches.

Definition at line **44** of file **SimpleRSLK.h**.

◆ LEFT_MOTOR

#define LEFT_MOTOR 0

Can be used to reference the left motor in the below functions.

Definition at line **49** of file **SimpleRSLK.h**.

◆ RIGHT_MOTOR

#define RIGHT_MOTOR 1

Can be used to reference the right motor in the below functions.

Definition at line **54** of file **SimpleRSLK.h**.

◆ BOTH_MOTORS

#define BOTH_MOTORS 2

Can be used to reference both motors in the below functions.

Definition at line **59** of file **SimpleRSLK.h**.

◆ LEFT_SHRP_DIST

#define LEFT_SHRP_DIST 0

Can be used to reference the left Sharp distance sensor.

Definition at line **64** of file **SimpleRSLK.h**.

◆ CENTER_SHRP_DIST

```
#define CENTER_SHRP_DIST   1
```

Can be used to reference the center Sharp distance sensor.

Definition at line **69** of file **SimpleRSLK.h**.

◆ RIGHT_SHRP_DIST

```
#define RIGHT_SHRP_DIST   2
```

Can be used to reference the right Sharp distance sensor.

Definition at line **74** of file **SimpleRSLK.h**.

◆ MOTOR_DIR_FORWARD

```
#define MOTOR_DIR_FORWARD   0
```

Can be used to reference setting the motor function to forward.

Definition at line **79** of file **SimpleRSLK.h**.

◆ MOTOR_DIR_BACKWARD

```
#define MOTOR_DIR_BACKWARD   1
```

Can be used to reference setting the motor function to backward.

Definition at line **84** of file **SimpleRSLK.h**.

◆ DARK_LINE

#define DARK_LINE 0

Used to specify that the robot is running on a floor lighter than the line.

Definition at line **89** of file **SimpleRSLK.h**.

◆ LIGHT_LINE

#define LIGHT_LINE 1

Used to specify that the robot is running on a floor darker than the line.

Definition at line **94** of file **SimpleRSLK.h**.

◆ readLineSensor

#define readLineSensor **readRawLineSensor**

Read line sensor values.

Deprecated:

Method deprecated since 0.2.2. Use `readRawLineSensor` instead. Method still used internally and retained for compatibility.

Parameters

`[out]` **sensor** array that stores values read from line sensor. Must pass an array with 8 elements. Array index 0 represents the left most sensor. Array index 7 represents the right most sensor. Each index will contain a value from 0 - 2500.

- 0 max reflection (light line)

  ...
- 2500 no reflection (dark line)

Read and store sensor values in the passed in array.

Definition at line **335** of file **SimpleRSLK.h**.

Function Documentation

◆ getEncoderLeftCnt()

uint32_t getEncoderLeftCnt ( )

Return number of encoder ticks from the left wheel.

Returns

The number of encoder ticks from the left wheel.

Definition at line **56** of file **Encoder.cpp**.

◆ getEncoderRightCnt()

uint32_t getEncoderRightCnt ( )

Return number of encoder ticks from the right wheel.

Returns

The number of encoder ticks from the right wheel.

Definition at line **51** of file **Encoder.cpp**.

◆ resetLeftEncoderCnt()

void resetLeftEncoderCnt ( )

Set the left encoder tick count to 0.

Definition at line **61** of file **Encoder.cpp**.

◆ resetRightEncoderCnt()

void resetRightEncoderCnt ( )

Set the right encoder tick count to 0.

Definition at line **66** of file **Encoder.cpp**.

◆ getLeftWheelDir()

uint8_t getLeftWheelDir ( )

Determines if the left wheel is going forward or backwards.

Returns

1 for forward or 0 for backwards

Definition at line **71** of file **Encoder.cpp**.

◆ getRightWheelDir()

uint8_t getRightWheelDir ( )

Determines if the right wheel is going forward or backwards.

Returns

1 for forward or 0 for backwards

Definition at line **75** of file **Encoder.cpp**.

◆ setupRSLK()

void setupRSLK ( )

Performs a variety of initialization needed for the RSLK.

This function must be called before calling any other functions listed on this page.

Definition at line **15** of file **SimpleRSLK.cpp**.

◆ readSharpDist()

uint16_t readSharpDist (uint8_t num)

Read distance sensor value.

Parameters

[in] **num** of the distance sensor to read. Valid values are 0 - 2. Representing the 3 RSLK's sensors that can be mounted on the RSLK (on top of the bump switch assembly).

- 0 (LEFT_SHRP_DIST) for the left sensor.
- 1 (CENTER_SHRP_DIST) for the center sensor.
- 2 (RIGHT_SHRP_DIST) for the right sensor.

Returns

A value from 0 - 4065.

- 0 represents object right infront of sensor
  ....
- 4065 represents no object detected

Definition at line **45** of file **SimpleRSLK.cpp**.

◆ readSharpDistMM()

int16_t readSharpDistMM (uint8_t num)

Read distance sensor value in millimeters.

This function returns a value representing the distance an object is from the sensor in millimeters. Range is 100 to 800 mm.

Parameters

[in] **num** of the distance sensor to read. Valid values are 0 - 2. Representing the 3 RSLK's sensors that can be mounted on the RSLK (on top of the bump switch assembly).

- 0 (LEFT_SHRP_DIST) for the left sensor.
- 1 (CENTER_SHRP_DIST) for the center sensor.
- 2 (RIGHT_SHRP_DIST) for the right sensor.

Returns

distance in millimeters (-1 if no object detected).

Definition at line **53** of file **SimpleRSLK.cpp**.

◆ readSharpDistIN()

float readSharpDistIN (uint8_t num)

Read distance sensor value in inches.

This function returns a value representing the distance an object is from the sensor in inches. Range is ~4 to 31 inches.

Parameters

[in] **num** of the distance sensor to read. Valid values are 0 - 2. Representing the 3 RSLK's sensors that can be mounted on the RSLK (on top of the bump switch assembly).

- 0 (LEFT_SHRP_DIST) for the left sensor.
- 1 (CENTER_SHRP_DIST) for the center sensor.
- 2 (RIGHT_SHRP_DIST) for the right sensor.

Returns

distance in inches (-1 if no object detected).

Definition at line **61** of file **SimpleRSLK.cpp**.

◆ isBumpSwitchPressed()

bool isBumpSwitchPressed (uint8_t num)

Return bump switch status.

Parameters

[in] **num** bump switch number. Valid values are 0 - 5 representing the RSLK's 6 bump switches.

- 0 for left most switch.

  ...
- 5 for right most switch.

Returns

- true if switch is pressed
- false if switch isn't pressed.

Definition at line **69** of file **SimpleRSLK.cpp**.

◆ getBumpSwitchPressed()

uint8_t getBumpSwitchPressed ( )

Return mask of bump switch states.

Returns mask representing state of the RSLK's 6 bump switches.

Returns

mask of bump switch states (0 if switch not pressed, 1 if switch pressed)

- bit 0 represents left most bump switch (BP_SW_PIN_0)

  ...
- bit 5 represents right most bump switch (BP_SW_PIN_5)

Definition at line **80** of file **SimpleRSLK.cpp**.

◆ enableMotor()

void enableMotor (uint8_t motorNum)

Enable motor (take it out of sleep)

Takes the motor out of sleep. The motor will not move unless you also call setMotorSpeed.

Parameters

[in] **motorNum** that designates the the motor. Valid values are 0 - 2.

- 0 for left motor
- 1 for right motor
- 2 for both motors

Definition at line **91** of file **SimpleRSLK.cpp**.

◆ disableMotor()

void disableMotor (uint8_t motorNum)

Disable motor (puts the motor to sleep)

Disabling the motor sets its speed to 0 and puts it to sleep.

Parameters

[in] **motorNum** that designates the the motor. Valid values are 0 - 2.

- 0 for left motor
- 1 for right motor
- 2 for both motors

Definition at line **102** of file **SimpleRSLK.cpp**.

◆ pauseMotor()

void pauseMotor (uint8_t motorNum)

Pause motor (put the motor to sleep while saving its speed)

Puts the motor to sleep while also preserving the previously set motor speed.

Parameters

[in] **motorNum** that designates the the motor. Valid values are 0-2.

- 0 for left motor
- 1 for right motor
- 2 for both motors

Definition at line **113** of file **SimpleRSLK.cpp**.

◆ resumeMotor()

void resumeMotor (uint8_t motorNum)

Resume motor (take the motor out of sleep and resumes its prior speed)

Take the motor out of sleep and sets its speed to its prior value.

Parameters

[in] **motorNum** that designates the the motor. Valid values are 0-2.

- 0 for left motor
- 1 for right motor
- 2 for both motors

Definition at line **124** of file **SimpleRSLK.cpp**.

◆ setMotorDirection()

void setMotorDirection (uint8_t motorNum,
uint8_t direction
)

Set direction the motor will turn.

Specifies the motor's direction. Can control an indivdual motor or both motors.

Parameters

[in] **motorNum** that designates the the motor. Valid values are 0-2.

- 0 for left motor
- 1 for right motor
- 2 for both motors

[in] **direction** that specifies the motor's direction

- 0 for forward
- 1 for for backward

Definition at line **135** of file **SimpleRSLK.cpp**.

◆ setMotorSpeed()

void setMotorSpeed (uint8_t motorNum,
uint8_t speed
)

Set the motor speed.

Sets the speed of the motor. A value of 0 means no movement. 100 will set the motor to its fastest speed.

Parameters

[in] **motorNum** that designates the the motor. Valid values are 0-2.

- 0 for left motor
- 1 for right motor
- 2 for both motors

[in] **speed** that specifies the motor speed. Valid values are 0 - 100.

- 0 for 0% of motor speed.
  ...
- 100 for 100% of motor speed.

Definition at line **154** of file **SimpleRSLK.cpp**.

◆ setupWaitBtn()

void setupWaitBtn (uint8_t btn)

Configure pin as a wait to release button.

Configure pin to be used as a wait until pushed and released button. Useful if you want to halt the robot's operation until the uses pushes and then releases the button.

Parameters

[in] **btn** the Launchpad pin number you want to use.

Definition at line **266** of file **SimpleRSLK.cpp**.

◆ setupLed()

void setupLed (uint8_t ledPin)

Configure pin that is connected to an led.

Configure pin to be used for as an led.

Parameters

[in] **ledPin** the Launchpad pin number you want to use.

Definition at line **271** of file **SimpleRSLK.cpp**.

◆ waitBtnPressed()

void waitBtnPressed (uint8_t btnPin,
int8_t ledPin = 0
)

Busy wait until user pushes and releases button.

Prevent additional code from executing until use has pushed and released specified button.

Parameters

[in] **btnPin** the Launchpad pin number you want to use.

[in] **ledPin** represents the pin to toggle high and low while waiting for btn to be pressed.

Definition at line **276** of file **SimpleRSLK.cpp**.

◆ calibrateLineSensor()

```
void calibrateLineSensor ( uint8_t   mode = DARK_LINE,
                           uint32_t  duration = 500
                         )
```

Calibrates line sensor.

Calibrates line sensor by identifying min/max sensor range of non-line surface. Sensors *SHOULD NOT* be positioned over line while calibrating (i.e. calibrate to background surface color).

Parameters

`[in]` **mode** determines if the line is dark or light (default is DARK_LINE)

- 0 (DARK_LINE) is used when the line is darker than the floor
- 1 (LIGHT_LINE) is used when the line is lighter than the floor.

`[in]` **duration** duration for calibration in milliseconds (default is 500)

Definition at line **189** of file **SimpleRSLK.cpp**.

◆ readCalLineSensor() [1/2]

void readCalLineSensor (uint16_t * calVal)

Read calibrated line sensor values. Assumes calibration completed.

Takes the current line sensor values and sets calVal to the calibrated values. Assumes **calibrateLineSensor()** has already been called (only necessary to calibrate once).

Parameters

`[out]` **calVal** is an array that will be filled with the calibrated values based on the sensor.
Elements will be filled with values of 0 - 1000

- 0 means no line detected
  ...
- 1000 means line is detected right under sensor.

Note

Calibration:

- When the line is dark then calibration subtracts sensorMax values from the sensor value read.
- When the line is light then calibration subtracts sensorMin values from the sensor value read. Then the value is subtracted from 1000 to provide a consistent scale.

Definition at line **201** of file **SimpleRSLK.cpp**.

◆ getLinePosition() `[1/2]`

uint32_t getLinePosition ( )

Get line position.

Provides a numerical value indicating where the robot is detecting the line. Assumes **calibrateLineSensor()** has already been called (only necessary to calibrate once).

Returns

value between 0 - 8000.

- 0 no line detected
  ...
- 1000 line is directly on the left most sensor
  ...
- 4500 line directly over two middle sensors.
  ...
- 8000 line is under right most line sensor

Definition at line **227** of file **SimpleRSLK.cpp**.

◆ readRawLineSensor()

void readRawLineSensor (uint16_t * sensor)

Read raw line sensor values.

Read and store raw line sensor values in the passed in array.

Parameters

`[out]` **sensor** array that stores values read from line sensor. Must pass an array with 8 elements. Array index 0 represents the left most sensor. Array index 7 represents the right most sensor. Each index will contain a value from 0 - 2500.

- 0 max reflection (light line)
  ...
- 2500 no reflection (dark line)

Definition at line **176** of file **SimpleRSLK.cpp**.

◆ clearMinMax()

void clearMinMax (uint16_t * sensorMin,
uint16_t * sensorMax
)

Provide default values for the sensor's Min and Max arrays.

Deprecated:
Method deprecated since 0.2.2. Method still used internally and retained for compatibility.

Parameters

[out] **sensorMin** stores sensor's min values. Must pass an array with 8 elements. All elements will by default be given a large value.

[out] **sensorMax** stores sensor's max values. Must pass an array with 8 elements. All elements will by default be given a value of 0.

Initializes arrays to be used to store line sensor's min and max values.

Definition at line **181** of file **SimpleRSLK.cpp**.

◆ setSensorMinMax()

void setSensorMinMax (uint16_t * sensor,
uint16_t * sensorMin,
uint16_t * sensorMax
)

Update line sensor's min and max values array based on current data.

Deprecated:
Method deprecated since 0.2.2. Method still used internally and retained for compatibility.

Parameters

[in] **sensor** is an array filled with line sensor values previously filled by readLineSensor.

[out] **sensorMin** stores sensor's min values.

[out] **sensorMax** stores sensor's max values.

Take the current line sensor values and update min and max values. This function along with the min and max arrays are useful when performing calibration.

Definition at line **254** of file **SimpleRSLK.cpp**.

◆ readCalLineSensor() [2/2]

```
void readCalLineSensor ( uint16_t *  sensor,
                         uint16_t *  calVal,
                         uint16_t *  sensorMin,
                         uint16_t *  sensorMax,
                         uint8_t     mode
                       )
```

Update sensor's min and max values array based on current data.

Deprecated:

Method deprecated since 0.2.2. Use **`readCalLineSensor(uint16_t*)`** instead. Method still used internally and retained for compatibility.

Parameters

[out]	**sensor**	is an array to be filled with line sensor values.
[out]	**calVal**	is an array that will be filled with the calibrated values based on the sensor, sensorMin and sensorMax array. Elements will be filled with values of 0 - 1000 • 0 means no line detected ... • 1000 means line is detected right under sensor.
[in]	**sensorMin**	stores sensor's min values.
[in]	**sensorMax**	stores sensor's max values.
[in]	**mode**	determines if the line is dark or light. • 0 is used when the line is darker than the floor • 1 is used when the line is lighter than the floor.

Takes the current line sensor values and sets calVal to the calibrated values. Uses sensorMin and sensorMax array along with mode to calibrate value.

Calibration:

- When the line is dark then calibration subtracts sensorMax values from the sensor value read.
- When the line is light then calibration subtracts sensorMin values from the sensor value read. Then the value is subtracted from 1000 to provide a consistent scale.

Definition at line **208** of file **SimpleRSLK.cpp**.

◆ getLinePosition() [2/2]

```
uint32_t getLinePosition ( uint16_t *  calVal,
                           uint8_t     mode
                         )
```

Get line position.

Deprecated:

Method deprecated since 0.2.2. Use **`getLinePosition()`** instead. Method still used internally and retained for compatibility.

Parameters

[in] **calVal** is an array that is filled with the line sensor calibrated values.

[in] **mode** determines if the line is dark or light.

- 0 is used when the line is darker than the floor
- 1 is used when the line is lighter than the floor.

Returns

value between 0 - 8000.

- 0 no line detected

  ...
- 1000 line is directly on the left most sensor

  ...
- 4500 line directly over two middle sensors.

  ...
- 8000 line is under right most line sensor

Using calibrated line sensor value this function provides a numerical value indicating where the robot is detecting the line. This function can be overridden.

Definition at line **233** of file **SimpleRSLK.cpp**.