

MyroC Project

Generated by Doxygen 1.8.13

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	bth_inquiry_record Struct Reference	5
3.2	GLimageset Struct Reference	5
3.2.1	Detailed Description	6
3.3	image_timing_data Struct Reference	6
3.3.1	Detailed Description	6
3.4	motion_timing_data Struct Reference	7
3.4.1	Detailed Description	7
3.5	my_error_mgr Struct Reference	7
3.6	Picture Struct Reference	7
3.6.1	Detailed Description	8
3.7	Pixel Struct Reference	8
3.7.1	Detailed Description	9
3.8	robot_motion_camera_record Struct Reference	9
3.8.1	Detailed Description	9
3.9	timing_thread_info Struct Reference	9
3.9.1	Detailed Description	9

4 File Documentation	11
4.1 MyroC-image-file.c File Reference	11
4.1.1 Detailed Description	12
4.1.2 Function Documentation	13
4.1.2.1 rLoadPicture()	13
4.1.2.2 rSavePicture()	14
4.2 MyroC.h File Reference	15
4.2.1 Detailed Description	18
4.2.2 Function Documentation	18
4.2.2.1 rBackward()	18
4.2.2.2 rBeep()	19
4.2.2.3 rBeep2()	19
4.2.2.4 rConnect()	20
4.2.2.5 rDisconnect()	22
4.2.2.6 rDisplayImageProcessingErrors()	22
4.2.2.7 rDisplayPicture()	23
4.2.2.8 rFastForward()	24
4.2.2.9 rFinishProcessing()	25
4.2.2.10 rForward()	25
4.2.2.11 rGetBattery()	26
4.2.2.12 rGetBrightAll()	26
4.2.2.13 rGetBrightTxt()	27
4.2.2.14 rGetForwardness()	28
4.2.2.15 rGetInfo()	29
4.2.2.16 rGetIRAll()	29
4.2.2.17 rGetIRTxt()	30
4.2.2.18 rGetLightsAll()	32
4.2.2.19 rGetLightTxt()	32
4.2.2.20 rGetLine()	34
4.2.2.21 rGetName()	35

4.2.2.22	rGetObstacleAll()	35
4.2.2.23	rGetObstacleTxt()	36
4.2.2.24	rGetStall()	38
4.2.2.25	rHardStop()	39
4.2.2.26	rLoadPicture()	39
4.2.2.27	rMotors()	40
4.2.2.28	rSavePicture()	40
4.2.2.29	rSetBluetoothEcho()	42
4.2.2.30	rSetConnection()	42
4.2.2.31	rSetForwardness()	43
4.2.2.32	rSetForwardnessTxt()	43
4.2.2.33	rSetIRPower()	44
4.2.2.34	rSetLEDBack()	44
4.2.2.35	rSetLEDFront()	45
4.2.2.36	rSetName()	45
4.2.2.37	rSetVolume()	46
4.2.2.38	rStop()	46
4.2.2.39	rTakePicture()	46
4.2.2.40	rTurnLeft()	47
4.2.2.41	rTurnRight()	48
4.2.2.42	rTurnSpeed()	49
4.2.2.43	rWaitTimedImageDisplay()	49

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

bth_inquiry_record	5
GLimageset	5
image_timing_data	6
motion_timing_data	7
my_error_mgr	7
Picture	
Struct for a picture object	7
Pixel	
Struct for a pixel	8
robot_motion_camera_record	9
timing_thread_info	9

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

MyroC-image-file.c	
Implementation file for saving/loading jpeg images to/from a file	11
MyroC-utilities-with-getcurrentDirectory.h	??
MyroC-utilities.h	??
MyroC.h	
Header for a C-based, my-robot package for the Scribbler 2	15

Chapter 3

Data Structure Documentation

3.1 `bth_inquiry_record` Struct Reference

Data Fields

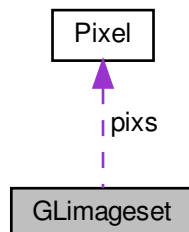
- char **addrStr** [INQUIRY_STR_MAX]
- char **serviceName** [INQUIRY_STR_MAX]

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

- MyroC-connect-windows.c

3.2 `GLimageset` Struct Reference

Collaboration diagram for `GLimageset`:



Data Fields

- char * **window_title**
- pid_t **display_pid**
- int **socket_id** [2]
- pthread_mutex_t **lock**
- GLuint **texture_objs** [1]
- int **height**
- int **width**
- int **process_status**
- int **image_num**
- Pixel * **pixs**
- HANDLE **toImgHandWr**
- HANDLE **fromImgHandRd**
- PROCESS_INFORMATION **procInfo**

3.2.1 Detailed Description

Struct for recording details of openGL process

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

- MyroC-display-linux-mac.c
- MyroC-display-windows-interface-manager.c
- MyroC-display.c

3.3 image_timing_data Struct Reference

Data Fields

- double **delay_time**
- int **GLImageset_index**
- int **image_seq_number**
- int **thread_seq_number**

3.3.1 Detailed Description

Struct for timing display of image

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

- MyroC-display-linux-mac.c
- MyroC-display-windows-interface-manager.c
- MyroC-display-windows-opengl-img-handler.c
- MyroC-display.c

3.4 motion_timing_data Struct Reference

```
#include <MyroC-utilities-with-getcurrentDirectory.h>
```

Data Fields

- double **duration_time**
- SOCKET_TYPE **robot_socket_num**
- int **motion_seq_num**

3.4.1 Detailed Description

Struct for timing motion commands of given robot

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

- MyroC-utilities-with-getcurrentDirectory.h
- MyroC-utilities.h

3.5 my_error_mgr Struct Reference

Data Fields

- struct jpeg_error_mgr **pub**
- jmp_buf **setjmp_buffer**

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

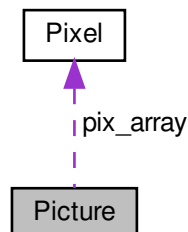
- MyroC-image-file-linux-mac.c
- [MyroC-image-file.c](#)

3.6 Picture Struct Reference

Struct for a picture object.

```
#include <MyroC.h>
```

Collaboration diagram for Picture:



Data Fields

- int [height](#)
The actual height of the image, but no more than 266.
- int [width](#)
The actual width of the image, but no more than 427.
- Pixel [pix_array](#) [266][427]
The array of pixels comprising the image.

3.6.1 Detailed Description

Struct for a picture object.

Note

images from robot cameras have varying sizes, depending on the Fluke
 pix_array is sufficiently large to accommodate any Fluke version
 images for the original Fluke are 192 (height) by 256 (width)
 low-resolution images for the Fluke 2 are 266 by 427
 high-resolution images (e.g., 800 by 1280) are not practical, due to memory constraints and thus are not available in MyroC
 user-defined images may have any size, as long as height ≤ 266 and width ≤ 427
 Following standard mathematical convention for a 2D matrix, all references to a pixel are given within an array as [row][col]

Warning

The [Picture](#) struct is defined to be sufficiently large to store several low-resolution camera images (340756 bytes each) Experimentally, an array of up to 94 (not 95) Pictures is allowed However, the display of images requires that image data be copied, so display of many images may not work If a program hangs when working with [Picture](#) variables, the issue may involve lack of space on the runtime stack. To utilize a modest number of Pictures, use "ulimit -s" command, as needed, in a terminal window For example, ulimit -s 32768 Sizes above 32768 may not be allowed in Linux or Mac OS X

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

- [MyroC.h](#)

3.7 Pixel Struct Reference

Struct for a pixel.

```
#include <MyroC.h>
```

Data Fields

- unsigned char [R](#)
The value of the red component.
- unsigned char [G](#)
The value of the green component.
- unsigned char [B](#)
The value of the blue component.

3.7.1 Detailed Description

Struct for a pixel.

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

- [MyroC.h](#)

3.8 robot_motion_camera_record Struct Reference

```
#include <MyroC-utilities-with-getcurrentDirectory.h>
```

Data Fields

- SOCKET_TYPE **robot_socket_num**
- int **camera_height**
- int **camera_width**
- char * **fluke_type**
- int **motion_seq_num**
- pthread_t **thread_motion_timer_id**

3.8.1 Detailed Description

Struct for recording number of motion commands by a robot

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

- MyroC-utilities-with-getcurrentDirectory.h
- MyroC-utilities.h

3.9 timing_thread_info Struct Reference

Data Fields

- int **thread_seq_number**
- pthread_t **pthread**

3.9.1 Detailed Description

Struct for acting timing-thread information

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

- MyroC-display-linux-mac.c
- MyroC-display-windows-interface-manager.c
- MyroC-display-windows-opengl-img-handler.c
- MyroC-display.c

Chapter 4

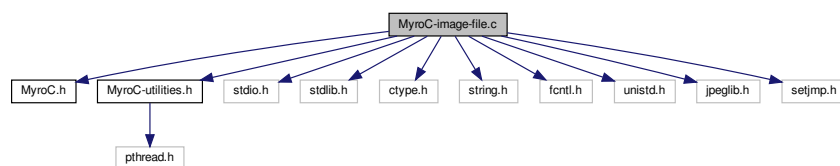
File Documentation

4.1 MyroC-image-file.c File Reference

Implementation file for saving/loading jpeg images to/from a file.

```
#include "MyroC.h"  
#include "MyroC-utilities.h"  
#include <stdio.h>  
#include <stdlib.h>  
#include <ctype.h>  
#include <string.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <jpeglib.h>  
#include <setjmp.h>
```

Include dependency graph for MyroC-image-file.c:



Data Structures

- struct [my_error_mgr](#)

Typedefs

- typedef struct [my_error_mgr](#) * [my_error_ptr](#)

Functions

- void **my_error_exit** (j_common_ptr cinfo)
- void **rSavePicture** (**Picture** *pic, char *filename)
*Save a **Picture** to a .jpeg.*
- **Picture** **rLoadPicture** (char *filename)
Load a picture from a .jpeg file.

4.1.1 Detailed Description

Implementation file for saving/loading jpeg images to/from a file.

Revision History

Version 1.0 based on a C++ package by April O'Neill, David Cowden, Dilan Ustek, Erik Opavsky, and Henry M. Walker

Developers of the C package for Linux: Creators Version 2.0 (C functions for utilities,general,sensors,movement)↔ : Spencer Liberto Dilan Ustek Jordan Yuan Henry M. Walker Contributors Version 2.2-2.3: (C functions for image processing) Anita DeWitt Jason Liu Nick Knoebber Vasilisa Bashlovkina Revision for Version 2.4: (image row/column made to match matrix notation) Henry M. Walker

Revisions for Version 3.0 Henry M. Walker

C ported to Macintosh Linux/Mac differences required for connections — otherwise same code OpenGL used to display images, replacing ImageMagick same [new] code used for both Linux and Macintosh 1 process for robot control 1 process needed for each titled window (not each image, as in 2.2-2.4) Blocking options (negative duration parameter) utilize separate thread timer MyroC implementation files organized by user function as follows:

Revisions for Version 3.1 Henry M. Walker

Picture struct and image functions revised to allow 192 by 256 images from original Fluke camera 266 x 427 low-resolution images from Fluke 2 (high-resolution (800 x 1280) too large for more than 2-4 variables on run-time stack) storage , retrieval, and display of any images up to 266 x 427

This program and all MyroC software is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License. Details available at <http://creativecommons.org/licenses/by-nc-sa/3.0/us/>↔

Revisions for Version 3.2 Henry M. Walker

Practical range of rBeep duration identified as <= 3.0014 seconds Image display and processing resolves several matters and adds functionality function rDisplayPicture completely rewritten over time OpenGL routines had encountered troubles on Linux, specifically glutHideWindow() hid images, but could not be restored, with difficulties depending on the graphics card on Mac, High Sierra generated compile warnings and restricted threads that could display images function rDisplayImageProcessingErrors added function rWaitTimedImageDisplay updated substantially

Revisions for Version 3.3 Henry M. Walker

MyroC ported to Windows 10, in addition to Linux and Mac OS X Additional refinements text parameter for rGet↔ LightTxt expanded to "Middle" as well as "Center"

This program and all MyroC software is licensed under the Creative Commons Attribution-Noncommercial-↔ Share Alike 3.0 United States License. Details may be found at <http://creativecommons.org/licenses/by-nc-sa/3.0/us/>↔

implementation overview *****

indicates functions implemented in this file

0. LOW-LEVEL UTILITY (defined in [MyroC-utilities.h](#) implemented in MyroC-utilities.c) rSend rReceive rSet↵
OpeningExchange rSetBluetoothEcho rSetEchoMode rCheckHardwareVersionSetCameraSize

1. GENERAL (mostly MyroC-general.c) 2. SENSOR (MyroC-sensors.c) rConnect (* MyroC-connect.c) a.↵
Scribbler Sensors rDisconnect (* MyroC-connect.c) rGetLightsAll rSetConnection rGetLightTxt rFinish↵
Processing rGetIRAll rSetVolume rGetIRTxt rBeep rGetLine rBeep2 rSetName b.Fluke Sensors rGet↵
Name rGetObstacleAll rSetForwardness rGetObstacleTxt rGetForwardness rGetBrightAll rSetLEDFront r↵
GetBrightTxt rSetLEDBack rSetIRPower rGetBattery rGetStall

Note:

MyroC-connect requires special
knowledge of a Mac or Linux
environment, yielding two versions:
MyroC-connect-mac.c
MyroC-connect-linux.c

1. MOVEMENT (MyroC-movement.c) 4. PICTURES (files as indicated) rTurnLeft rGetCameraSize (MyroC-
camera.c) rTurnRight rTakePicture (MyroC-camera.c) rTurnSpeed rDisplayPicture (MyroC-display.c) rForward
rWaitTimedImageDisplay(MyroC-display.c) rFastForward > rSavePicture ([MyroC-image-file.c](#)) rBackward >
rLoadPicture ([MyroC-image-file.c](#)) rMotors rStop rHardStop

4.1.2 Function Documentation

4.1.2.1 rLoadPicture()

```
Picture rLoadPicture (
    char * filename )
```

Load a picture from a .jpeg file.

Parameters

<i>filename</i>	the name of the file
-----------------	----------------------

Precondition

file must exist
file extension must be .jpeg or .jpg
file must be no larger than 266 by 427

Returns

locally-declared [Picture](#)

Postcondition

calling program should store returned [Picture](#) in a previously-declared struct

4.1.2.2 rSavePicture()

```
void rSavePicture (
    Picture * pic,
    char * filename )
```

Save a [Picture](#) to a .jpeg.

This section contains functions for taking and manipulating images All images are constrained with height ≤ 266 and width ≤ 427

images from robot cameras have varying sizes, depending on the Fluke images for the original Fluke are 192 (height) by 256 (width) low-resolution images for the Fluke 2 are 266 by 427 high-resolution images for the fluke 2 are 800 by 1280 (not supported)

Bluetooth communication constrains the time required for the Fluke to take a picture Typical times: original fluke: 2-4 seconds Fluke 2 (low res): 4- 6 seconds Fluke 2 (high res): 25-30 seconds

BASED ON TIMINGS AND MEMORY CONSIDERATIONS, Myro C ALLOWS ONLY LOW RESOLUTION IMAGES

the [Picture](#) struct allows direct access to [Pixel](#) data Pictures can be saved and loaded as .jpeg files

Note

Following standard mathematical convention for a 2D matrix, all references to a pixel are given within an array as [row][col]
 user-defined images may have any size, as long as height ≤ 266 and width ≤ 427
 Following standard mathematical convention for a 2D matrix, all references to a pixel are given within an array as [row][col]

Warning

The [Picture](#) struct is defined to be sufficiently large to store several low-resolution camera images (340756 bytes each) Experimentally, an array of up to 94 (not 95) Pictures is allowed However, the display of images requires that image data be copied, so display of many images may not work If a program hangs when working with [Picture](#) variables, the issue may involve lack of space on the runtime stack. To utilize a modest number of Pictures, use "ulimit -s" command, as needed, in a terminal window For example, ulimit -s 32768 Sizes above 32768 may not be allowed in Linux or Mac OS X Save a [Picture](#) to a .jpeg

Parameters

<i>pic</i>	pointer to an RGB picture struct from Scribbler 2 camera
<i>filename</i>	the name of the file

Precondition

filename ends with .jpeg or .jpg.

Postcondition

If the file does not exist, a new file will be created.

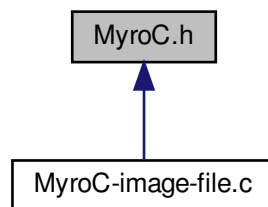
If the file exists, the file will be overwritten.

this code is a slightly modified version of `write_JPEG_file(char *, int)` from `example.c` from the `libjpeg.v8d` library directory.

4.2 MyroC.h File Reference

Header for a C-based, my-robot package for the Scribbler 2.

This graph shows which files directly or indirectly include this file:

**Data Structures**

- struct [Pixel](#)
Struct for a pixel.
- struct [Picture](#)
Struct for a picture object.

Macros

- `#define ROBOT_CONNECTION_TYPE int`

Functions

- **ROBOT_CONNECTION_TYPE rConnect** (const char *address)
connects program to Scribbler
- void **rDisconnect** ()
stop Scribbler motion and close Bluetooth
- void **rSetConnection** (ROBOT_CONNECTION_TYPE new_socket_num)
set current connection to the socket number
- void **rFinishProcessing** ()
all timed motions/image displays completed, all robots stopped, and all robot Bluetooth connections closed
- void **rSetVolume** (char highMute)
Set sound to high volume (H) or mute (M) highMute set volume of Scribbler.
- void **rBeep** (double duration, int frequency)
Beeps with the given duration and frequency.
- void **rBeep2** (double duration, int freq1, int freq2)
Generates two notes for the prescribed duration.
- void **rSetName** (const char *name)
Change name stored in the robot to the 16-byte name given.
- const char * **rGetName** ()
Get the name of the robot.
- void **rSetForwardness** (char *direction)
specifies which end of the Scribbler is considered the front
- void **rSetForwardnessTxt** (char *direction)
alternative to rSetForwardness for compatibility with earlier MyroC
- char * **rGetForwardness** ()
Gets the forwardness of the Scribbler.
- void **rSetLEDFront** (int led)
Set the front [fluke] LED on or off.
- void **rSetLEDBack** (double led)
Set the the intensity of the back fluke LED.
- double **rGetBattery** ()
Get the current voltage from the Scribbler batteries; Maximum charge from 6 batteries could be up to 6 volts; Manufacturer suggests batteries should be changed below 4.1 volts.
- int **rGetStall** (int sampleSize)
- void **rSetBluetoothEcho** (char onOff)
*Turn on and off the printing transmission and echong of Bluetooth transmissions All robot commands involve the transmission of a command over Bluetooth Scribbler commands are always 9 bytes
Fluke commands have varying lengths
The fluke echos most, but not all, of the commands
For many commands, the fluke also echos 11 bytes of sensor data.*
- void **rGetLightsAll** (int lightSensors[3], int sampleSize)
Get the average values of each of the three light sensors in an array. Values of each light sensor can somewhat (typically under 5%-10%). To even out variability, the sensor can be queried sampleSize times and an average obtained.
- int **rGetLightTxt** (const char *sensorName, int sampleSize)
Get the average values of a specified light sensor. Values of each light sensor can somewhat (typically under 5%-10%). To even out variability, the sensor can be queried sampleSize times and an average obtained.
- void **rGetIRAll** (int irSensors[2], int sampleSize)
Get an array of true/false values regarding the presence of obstacle based on the average values of each of the three IR sensors. Since readings of each light sensor can vary substantially, each sensor can be queried sampleSize times and an average obtained.
- int **rGetIRTxt** (const char *sensorName, int sampleSize)

- Use specified IR sensor to determine if obstacle is present. Since values of each light sensor can vary substantially, the sensor can be queried sampleSize times and an average obtained.*
- void **rGetLine** (int lineSensors[2], int sampleSize)
Use Scribbler 2 line sensors of Scribbler to check for a black line on a white surface under the robot. Since values of each light sensor can vary substantially, the sensor can be queried sampleSize times and an average obtained.
 - void **rSetIRPower** (int power)
Set the amount of power for the dongle's IR sensors.
 - void **rGetObstacleAll** (int obstSensors[3], int sampleSize)
Get the average values of the three obstacle sensors in an array. Since readings of each obstacle sensor can vary substantially (successive readings may differ by several hundred or more), each sensor can be queried sampleSize times and an average obtained.
 - int **rGetObstacleTxt** (const char *sensorName, int sampleSize)
Get the average values of a specified obstacle (IR) sensor. Since values of each obstacle sensor can vary substantially (successive readings may differ by several hundred or more), the sensor can be queried sampleSize times and an average obtained.
 - void **rGetBrightAll** (int brightSensors[3], int sampleSize)
Read the Fluke's virtual light sensors. Since readings of each brightness sensor can vary substantially (successive readings may differ by 5000-10000), each sensor can be queried sampleSize times and an average obtained.
 - int **rGetBrightTxt** (char *sensorName, int sampleSize)
Reads one of the Fluke's virtual light sensors. Each sensor reports a total intensity in the left, middle, or right of the Fluke's camera. Since values of each obstacle sensor can vary substantially (successive readings may differ by 5000-10000), the sensor can be queried sampleSize times and an average obtained.
 - void **rGetInfo** (char *infoBuffer)
returns information about the robot's dongle, firmware, and communication mode as a 60 character array in infoBuffer.
 - void **rTurnLeft** (double speed, double time)
turn Scribbler left for a specified time and speed
 - void **rTurnRight** (double speed, double time)
turn Scribbler right for a specified time and speed
 - void **rTurnSpeed** (char *direction, double speed, double time)
turn Scribbler in direction for a specified time and speed
 - void **rForward** (double speed, double time)
moves Scribbler forward for a specified time and speed
 - void **rFastForward** (double time)
moves Scribbler forward at the largest possible speed for a specified time
 - void **rBackward** (double speed, double time)
moves Scribbler backward for a specified time and speed
 - void **rMotors** (double leftSpeed, double rightSpeed)
move robot with given speeds for the left and right motors continues until given another motion command or disconnected (non-blocking)
 - void **rStop** ()
directs robot to stop movement
 - void **rHardStop** ()
cuts power to the motor of the robot
 - **Picture** **rTakePicture** ()
Use the camera to take a photo.
 - void **rSavePicture** (**Picture** *pic, char *filename)
*Save a **Picture** to a .jpeg.*
 - **Picture** **rLoadPicture** (char *filename)
Load a picture from a .jpeg file.
 - void **rDisplayPicture** (**Picture** *pic, double duration, const char *windowTitle)
Display a picture in a new window.
 - void **rDisplayImageProcessingErrors** ()
Turn on and off the printing of error message for image processing.
 - void **rWaitTimedImageDisplay** ()
Wait until all timed, non-blocking image window timers are complete.

4.2.1 Detailed Description

Header for a C-based, my-robot package for the Scribbler 2.

Revision History

Version 1.0 based on a C++ package by April O'Neill, David Cowden, Dilan Ustek, Erik Opavsky, and Henry M. Walker

Developers of the C package for Linux: Creators Version 2.0 (C functions for utilities,general,sensors,movement)↔ : Spencer Liberto Dilan Ustek Jordan Yuan Henry M. Walker Contributors Version 2.2-2.3: (C functions for image processing) Anita DeWitt Jason Liu Nick Knoebber Vasilisa Bashlovkina Revision for Version 2.4: (image row/column made to match matrix notation) Henry M. Walker

Revisions for Version 3.0 Henry M. Walker

C ported to Macintosh Linux/Mac differences required for connections — otherwise same code OpenGL used to display images, replacing ImageMagick same [new] code used for both Linux and Macintosh 1 process for robot control 1 process needed for each titled window (not each image, as in 2.2-2.4) Blocking options (negative duration parameter) utilize separate thread timer MyroC implementation files organized by user function as follows:

Revisions for Version 3.1 Henry M. Walker

Picture struct and image functions revised to allow 192 by 256 images from original Fluke camera 266 x 427 low-resolution images from Fluke 2 (high-resolution (800 x 1280) too large for more than 2-4 variables on run-time stack) storage, retrieval, and display of any images up to 266 x 427

Revisions for Version 3.2 Henry M. Walker

Practical range of rBeep duration identified as ≤ 3.0014 seconds Image display and processing resolves several matters and adds functionality function rDisplayPicture completely rewritten over time OpenGL routines had encountered troubles on Linux, specifically glutHideWindow() hid images, but could not be restored, with difficulties depending on the graphics card on Mac, High Sierra generated compile warnings and restricted threads that could display images function rDisplayImageProcessingErrors added function rWaitTimedImageDisplay updated substantially

Revisions for Version 3.3 Henry M. Walker

MyroC ported to Windows 10, in addition to Linux and Mac OS X Additional refinements text parameter for rGet↔ LightTxt expanded to "Middle" as well as "Center"

This program and all MyroC software is licensed under the Creative Commons Attribution-Noncommercial-↔ Share Alike 3.0 United States License. Details may be found at <http://creativecommons.org/licenses/by-nc-sa/3.0/us/>

4.2.2 Function Documentation

4.2.2.1 rBackward()

```
void rBackward (
    double speed,
    double time )
```

moves Scribbler backward for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move backward linear range: -1.0 specifies move forward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move backward at full speed
<i>time</i>	specifies the duration of the turn if negative: the robot starts backward (non-blocking) other processing proceeds, and the robot continues backward until given another motion command or disconnected (non-blocking) if zero: robot starts moving backward (non-blocking) other processing proceeds if positive: robot moves backward for the given duration, in seconds

moves Scribbler backward for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move backward linear range: -1.0 specifies move forward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move backward at full speed
<i>time</i>	specifies the duration of the turn if negative: robot continues to go backward until given another motion command or disconnected (non-blocking) if nonnegative: robot moves backward for the given duration, in seconds

4.2.2.2 rBeep()

```
void rBeep (
    double duration,
    int frequency )
```

Beeps with the given duration and frequency.

Parameters

<i>duration</i>	length of note in seconds
<i>frequency</i>	frequency of pitch in cycles per second (hertz)

Precondition

$0.0 < \text{duration} > 0.0 \leq 3.0$ seconds (Scribbler 2 hangs if duration > 3.0014 seconds; although reason unknown, experimentation suggests part of Scribbler 2 firmware)

4.2.2.3 rBeep2()

```
void rBeep2 (
    double duration,
    int freq1,
    int freq2 )
```

Generates two notes for the prescribed duration.

Parameters

<i>duration</i>	length of note in seconds
<i>freq1</i>	frequency of first pitch in cycles per second (hertz)
<i>freq2</i>	frequency of second pitch in cycles per second (hertz)

Precondition

0.0 < duration <= 3.0 seconds (Scribbler 2 hangs if duration > 3.0014 seconds; although reason unknown, experimentation suggests part of Scribbler 2 firmware)

4.2.2.4 rConnect()

```
ROBOT_CONNECTION_TYPE rConnect (
    const char * address )
```

connects program to Scribbler

Parameters

<i>address</i>	string, giving name of workstation port or a Scribbler Bluetooth designation
----------------	--

several string formats are possible

Linux and Mac:

a communications port, such as "/dev/rfcomm0"

a Scribbler 2 fluke serial number, such as "245787"

a full IPRE serial number, such as "IPRE245787"

a Fluke 2 serial number (hexadecimal), such as "021F"

a full Fluke 2 serial number, such as "Fluke2-021F"

Linux only:

a MAC address, such as "00:1E:19:01:0E:13"

Mac only:

any substring of a complete device file name,

as long as the resulting device is unique

some possibilities include

a complete device file name, such as

- "/dev/tty.IPRE6-365877-DevB"
- "/dev/tty.Fluke2-0958-Fluke2"
 - a fluke or fluke2 serial number
 - full path of symbolic link to a device filename string or substring in /dev

uniqueness is ensured by requiring 4 hex digits
or 6 decimal digits

Returns

if a connection is made the socket number/id of communications port is returned otherwise, the program exits, as robot processing cannot continue

Note

on Linux and Mac computers, sockets are ints, so rConnect returns an int on Windows computers, sockets have type SOCKET, so rConnect returns a SOCKET together, these are defined as a ROBOT_CONNECTI↵
ON_TYPE

Postcondition

subsequent communications will take place through this robot address, unless changed by rSetConnection

connects program to Scribbler

Parameters

<i>address</i>	string, giving name of workstation port or a Scribbler Bluetooth designation
----------------	--

several string formats are possible a communications port, such as "/dev/rfcomm0" a MAC address, such as "00:1↵
E:19:01:0E:13" a Scribbler 2 fluke serial number, such as "245787" a full IPRE serial number, such as "IPRE245787"
a Fluke 2 serial number (hexadecimal), such as "021F" a full Fluke 2 serial number, such as "Fluke2-021F"

Returns

if a connection is made the socket number of communications port is returned otherwise, the program exits, as robot processing cannot continue

Postcondition

subsequent communications will take place through this socket, unless changed by rSetConnection

Parameters

<i>address</i>	string, giving name of workstation port or a Scribbler Bluetooth designation
----------------	--

several string formats are possible a complete device file name, such as

- "/dev/rfcomm0"
- "/dev/tty.IPRE6-365877-DevB"
- "/dev/tty.Fluke2-0958-Fluke2" any substring of a complete device file name, as long as the resulting device is unique some possibilities include
- a Scribbler 2 fluke serial number, such as "245787"
- a full IPRE serial number, such as "IPRE6-245787"
- a Fluke 2 serial number (hexadecimal), such as "021F"

- a full Fluke 2 serial number, such as "Fluke2-021F" uniqueness is ensured by requiring 4 hex digits or 6 decimal digits the full path of a symbolic link to a device file name string or substring in /dev

Note: Connections via a MAC address, such as "00:1E:19:01:0E:13", is NOT available, for MyroC on Mac OS X, since AF_BLUETOOTH socket types are not defined in /usr/include/sys/socket.h Type man socket for additional information

Returns

if a connection is made the socket number of communications port is returned otherwise, the program exits, as robot processing cannot continue

Postcondition

subsequent communications will take place through this socket, unless changed by rSetConnection

4.2.2.5 rDisconnect()

```
void rDisconnect ( )
```

stop Scribbler motion and close Bluetooth

Postcondition

motion for the current robot is stopped, blocking until any non-blocking motion time has expired
 i.e., if a motion timer is set,
 this procedure blocks
 when the timer completes,
 then the motion stops
 else, procedure stops motion immediately
 Bluetooth for the current robot is closed
 motion for the current robot is stopped, blocking until any non-blocking motion time has expired
 i.e., if a motion timer is set,
 this procedure blocks, then stops motion
 else, procedure stops motion immediately
 Bluetooth for the current robot is closed

4.2.2.6 rDisplayImageProcessingErrors()

```
void rDisplayImageProcessingErrors ( )
```

Turn on and off the printing of error message for image processing.

Note

Some operating systems (e.g., Mac OS X High Sierra) display an error message whenever a new process for image display starts

Postcondition

This function toggles this printing on and off Initially, printing is off

Note

Some operating systems (e.g., Mac OS High Sierra) display an error message whenever a new process for image display starts

Postcondition

This function toggles this printing on and off Initially, printing is off

Note

Some operating systems (e.g., Mac OS High Sierra) display an error message whenever a new process for image display starts

Windows 10 does not generate these error messages, so this function has no useful purpose for Windows systems

Postcondition

This function turns this printing on Initially, printing is off

4.2.2.7 rDisplayPicture()

```
void rDisplayPicture (
    Picture * pic,
    double duration,
    const char * windowTitle )
```

Display a picture in a new window.

Parameters

<i>pic</i>	pointer to an RGB picture struct from Scribbler 2 camera
<i>duration</i>	if duration > 0, operation is blocking if duration <= 0, operation is non-blocking for duration != 0, picture displayed for abs(duration) seconds or until picture closed manually or until the program terminates if duration == 0, picture displayed until closed manually
<i>windowTitle</i>	The title of the window that appears. white spaces will be replaced with underscores.

Precondition

windowTitle is less than 100 characters.

Postcondition

image is displayed for the duration specified,
EXCEPT all display windows are closed when the main program terminates.

Warning

If images are displayed with a non-blocking option, and if the user wants images to appear for a full duration, use `rWaitTimedImageDisplay` or `rFinishProcessing` to block processing until all image timers are finished. Otherwise, program termination may close windows prematurely.

Display a picture.

Parameters

<i>pic</i>	pointer to an RGB picture struct from Scribbler 2 camera
<i>duration</i>	if duration > 0, operation is blocking if duration <= 0, operation is non-blocking for duration != 0, picture displayed for abs(duration) seconds or until picture closed manually or until the program terminates if duration == 0, picture displayed until closed manually
<i>windowTitle</i>	The title of the window that appears. white spaces will be replaced with underscores.

Precondition

windowTitle is less than 100 characters.

Postcondition

image is displayed for the duration specified,
EXCEPT all display windows are closed when the main program terminates.

Warning

If images are displayed with a non-blocking option, and if the user wants images to appear for a full duration, use `rWaitTimedImageDisplay` or `rFinishProcessing` to block processing until all image timers are finished. Otherwise, program termination may close windows prematurely.

4.2.2.8 rFastForward()

```
void rFastForward (
    double time )
```

moves Scribbler forward at the largest possible speed for a specified time

Parameters

<i>time</i>	specifies the duration of the turn if negative: the robot starts forward (non-blocking) other processing proceeds, and the robot continues forward until given another motion command or disconnected (non-blocking) if zero: robot starts moving forward (non-blocking); other processing proceeds if positive: robot moves forward for the given duration, in seconds
-------------	--

Warning

may take longer than usual to execute

moves Scribbler forward at the largest possible speed for a specified time

Parameters

<i>time</i>	specifies the duration of movement if negative: robot continues to move forward until given another motion command or disconnected (non-blocking) if nonnegative: robot moves forward for the given duration, in seconds
-------------	--

Warning

may take longer than usual to execute

4.2.2.9 rFinishProcessing()

```
void rFinishProcessing ( )
```

all timed motions/image displays completed, all robots stopped, and all robot Bluetooth connections closed

same result as rDisconnect for all robots plus rComleteImageDisplay

Postcondition

blocks until all timed robot motions are complete, and all timed image displays no longer visible upon completion, all timed images are invisible, all robot motion is halted, and all robot Bluetooth connections closed

produces same result as calling rDisconnect for all robots and also calling rComleteImageDisplay

Postcondition

procedure blocks until all timed robot motions are complete, and all timed image displays are no longer visible upon completion, all timed image displays are closed, all robot motion is halted, and all robot Bluetooth connections closed

4.2.2.10 rForward()

```
void rForward (
    double speed,
    double time )
```

moves Scribbler forward for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move forward linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forward at full speed
<i>time</i>	specifies the duration of the turn if negative: the robot starts forward (non-blocking) other processing proceeds, and the robot continues forward until given another motion command or disconnected (non-blocking) if zero: robot starts moving forward (non-blocking); other processing proceeds if positive: robot moves forward for the given duration, in seconds

moves Scribbler forward for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move forward linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forward at full speed
<i>time</i>	specifies the duration of movement if negative: robot continues to move forward until given another motion command or disconnected (non-blocking) if nonnegative: robot moves forward for the given duration, in seconds

4.2.2.11 rGetBattery()

```
double rGetBattery ( )
```

Get the current voltage from the Scribbler batteries; Maximum charge from 6 batteries could be up to 6 volts; Manufacturer suggests batteries should be changed below 4.1 volts.

Returns

percentage of battery voltage

Get the current voltage from the Scribbler batteries; Maximum charge from 6 batteries could be up to 6 volts; Manufacturer suggests batteries should be changed below 4.1 volts.

Returns

percentage of battery voltage

4.2.2.12 rGetBrightAll()

```
void rGetBrightAll (
    int brightSensors[3],
    int sampleSize )
```

Read the Fluke's virtual light sensors. Since readings of each brightness sensor can vary substantially (successive readings may differ by 5000-10000), each sensor can be queried sampleSize times and an average obtained.

Parameters

<i>brightSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for brightSensors array `sampleSize > 0`

Postcondition

`brightSensors[0]` gives average value for left sensor
`brightSensors[1]` gives average value for middle sensor
`brightSensors[2]` gives average value for right sensor
Brightness values near 0 represent bright light
Brightness values may extend to about 65535 for a dark region.

Read the Fluke's virtual light sensors. Since readings of each brightness sensor can vary substantially (successive readings may differ by 5000-10000), each sensor can be queried `sampleSize` times and an average obtained.

Parameters

<i>brightSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for brightSensors array `sampleSize > 0`

Postcondition

`brightSensors[0]` gives average value for left sensor `brightSensors[1]` gives average value for middle sensor
`brightSensors[2]` gives average value for right sensor
Brightness values near 0 represent bright light
Brightness values may extend to about 65535 for a very dark region.

4.2.2.13 rGetBrightTxt()

```
int rGetBrightTxt (
    char * sensorName,
    int sampleSize )
```

Reads one of the Fluke's virtual light sensors. Each sensor reports a total intensity in the left, middle, or right of the Fluke's camera. Since values of each obstacle sensor can vary substantially (successive readings may differ by 5000-10000), the sensor can be queried `sampleSize` times and an average obtained.

Parameters

<i>sensorName</i>	name of the bright sensor
-------------------	---------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive)
 designations "center" and "middle" are alternatives for the same bright sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

$sampleSize > 0$

Returns

reading from the specified bright sensor, averaged over sampleSize number of data samples
 Brightness values near 0 represent bright light
 Brightness values may extend to about 65535 for a very dark region.

Parameters

<i>sensorName</i>	name of the bright sensor
-------------------	---------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive) designations "center" and "middle" are alternatives for the same bright sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

$sampleSize > 0$

Returns

reading from the specified bright sensor, averaged over sampleSize number of data samples Brightness values near 0 represent bright light Brightness values may extend to about 65535 for a very dark region.

4.2.2.14 rGetForwardness()

```
char* rGetForwardness ( )
```

Gets the forwardness of the Scribbler.

Returns

either "fluke-forward" or "scribbler-forward"

Gets the forwardness of the Scribbler

Returns

either "fluke-forward" or "scribbler-forward"

4.2.2.15 rGetInfo()

```
void rGetInfo (
    char * infoBuffer )
```

returns information about the robot's dongle, firmware, and communication mode as a 60 character array in info↵Buffer.

Parameters

<i>infoBuffer</i>	a pre-defined, 60-character array
-------------------	-----------------------------------

Postcondition

infoBuffer contains relevant robot information

returns information about the robot's dongle, firmware, and communication mode as a 60 character array in info↵Buffer.

Parameters

<i>infoBuffer</i>	a pre-defined, 60-character array
-------------------	-----------------------------------

Postcondition

infoBuffer contains relevant robot information

4.2.2.16 rGetIRAll()

```
void rGetIRAll (
    int irSensors[2],
    int sampleSize )
```

Get an array of true/false values regarding the presence of obstacle based on the average values of each of the three IR sensors. Since readings of each light sensor can vary substantially, each sensor can be queried sample↵Size times and an average obtained.

Parameters

<i>irSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for irSensors array $sampleSize > 0$

Postcondition

irSensors[0] checks obstacle for left sensor
 irSensors[1] checks obstacle for right sensor
 for each irSensors array value
 return 0 indicates no obstacle detected
 return 1 indicates obstacle detected

Get an array of true/false values regarding the presence of obstacle based on the average values of each of the three IR sensors. Since readings of each light sensor can vary substantially, each sensor can be queried $sampleSize$ times and an average obtained.

Parameters

<i>irSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for irSensors array $sampleSize > 0$

Postcondition

irSensors[0] checks obstacle for left sensor irSensors[1] checks obstacle for right sensor
 for each irSensors array value return 0 indicates no obstacle detected return 1 indicates obstacle detected

4.2.2.17 rGetIRTxt()

```
int rGetIRTxt (
    const char * sensorName,
    int sampleSize )
```

Use specified IR sensor to determine if obstacle is present. Since values of each light sensor can vary substantially, the sensor can be queried $sampleSize$ times and an average obtained.

Parameters

<i>sensorName</i>	name of the light sensor
-------------------	--------------------------

Precondition

sensorName is "left" or "right" (not case sensitive)

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

sampleSize > 0

Returns

true/false (0/1) determination of obstacle, based on IR sensorName sensor, averaged over sampleSize number of data samples

Postcondition

return 0 indicates no obstacle detected
return 1 indicates obstacle detected

Parameters

<i>sensorName</i>	name of the light sensor
-------------------	--------------------------

Precondition

sensorName is "left" or "right" (not case sensitive)

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

sampleSize > 0

Returns

true/false (0/1) determination of obstacle, based on IR sensorName sensor, averaged over sampleSize number of data samples

Postcondition

return 0 indicates no obstacle detected return 1 indicates obstacle detected

4.2.2.18 rGetLightsAll()

```
void rGetLightsAll (
    int  lightSensors[3],
    int  sampleSize )
```

Get the average values of each of the three light sensors in an array. Values of each light sensor can somewhat (typically under 5%-10%). To even out variability, the sensor can be queried sampleSize times and an average obtained.

Parameters

<i>lightSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for lightSensors array sampleSize > 0

Postcondition

lightSensors[0] gives average value for left sensor
 lightSensors[1] gives average value for middle sensor
 lightSensors[2] gives average value for right sensor
 Intensity values near 0 represent bright light
 Intensities may extend to about 65000 for a dark region.

Parameters

<i>lightSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for lightSensors array sampleSize > 0

Postcondition

lightSensors[0] gives average value for left sensor lightSensors[1] gives average value for middle sensor light↔
 Sensors[2] gives average value for right sensor Intensity values near 0 represent bright light Intensities may
 extend to about 65000 for a very dark region.

4.2.2.19 rGetLightTxt()

```
int rGetLightTxt (
    const char * sensorName,
    int  sampleSize )
```

Get the average values of a specified light sensor. Values of each light sensor can somewhat (typically under 5%-10%). To even out variability, the sensor can be queried sampleSize times and an average obtained.

Parameters

<i>sensorName</i>	name of the light sensor
-------------------	--------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive)
designations "center" and "middle" are alternatives for the same light sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

sampleSize > 0

Returns

reading from the specified light sensor, averaged over sampleSize number of data samples
if sensorName invalid, returns -1.0

Get the average values of a specified light sensor. Values of each light sensor can somewhat (typically under 5%-10%). To even out variability, the sensor can be queried sampleSize times and an average obtained.

Parameters

<i>sensorName</i>	name of the light sensor
-------------------	--------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive) designations "center" and "middle" are alternatives for the same light sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

sampleSize > 0

Returns

reading from the specified light sensor, averaged over sampleSize number of data samples if sensorName invalid, returns -1.0

4.2.2.20 rGetLine()

```
void rGetLine (
    int lineSensors[2],
    int sampleSize )
```

Use Scribbler 2 line sensors of Scribbler to check for a black line on a white surface under the robot. Since values of each light sensor can vary substantially, the sensor can be queried sampleSize times and an average obtained.

Warning

results of these sensors may be flakey!

Parameters

<i>lineSensors</i>	array to store line values detected
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for lineSensors array sampleSize > 0

Postcondition

lineSensors[0] checks left sensor for line
lineSensors[1] checks right sensor for line
for each irSensors array value
return 0 indicates line is identified
return 1 indicates line is not identified

Use Scribbler 2 line sensors of Scribbler to check for a black line on a white surface under the robot. Since values of each light sensor can vary substantially, the sensor can be queried sampleSize times and an average obtained.

Warning

results of these sensors may be flakey!

Parameters

<i>lineSensors</i>	array to store line values detected
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for lineSensors array sampleSize > 0

Postcondition

lineSensors[0] checks left sensor for line lineSensors[1] checks right sensor for line (left/right based on Scribbler-forward orientation)
for each line sensors array value return 0 indicates line is identified (light color under Scribbler) return 1 indicates line is not identified (dark color)

4.2.2.21 rGetName()

```
const char* rGetName ( )
```

Get the name of the robot.

Returns

information about the name of the robot

Postcondition

the returned name is a newly-allocated 17-byte string

4.2.2.22 rGetObstacleAll()

```
void rGetObstacleAll (
    int  obstSensors[3],
    int  sampleSize )
```

Get the average values of the three obstacle sensors in an array. Since readings of each obstacle sensor can vary substantially (successive readings may differ by several hundred or more), each sensor can be queried sampleSize times and an average obtained.

Parameters

<i>obstSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for obstSensors array; sampleSize > 0

Postcondition

obstSensors[0] gives average value for left sensor
obstSensors[1] gives average value for middle sensor
obstSensors[2] gives average value for right sensor
Returned values are between 0 and 6400
Obstacle values near 0 represent no obstacle seen
Obstacle values may approach 6400 as obstacle gets close.

Warning

As battery degrades, sensor readings degrade, yielding systematically lower numbers.

Parameters

<i>obstSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for *obstSensors* array *sampleSize* > 0

Postcondition

obstSensors[0] gives average value for left sensor *obstSensors*[1] gives average value for middle sensor *obstSensors*[2] gives average value for right sensor Returned values are between 0 and 6400 Obstacle values near 0 represent no obstacle seen Obstacle values may approach 6400 as obstacle gets close.

Warning

As battery degrades, sensor readings degrade, yielding systematically lower numbers.

4.2.2.23 rGetObstacleTxt()

```
int rGetObstacleTxt (
    const char * sensorName,
    int sampleSize )
```

Get the average values of a specified obstacle (IR) sensor. Since values of each obstacle sensor can vary substantially (successive readings may differ by several hundred or more), the sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>sensorName</i>	name of the obstacle sensor
-------------------	-----------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive)
designations "center" and "middle" are alternatives for the same light sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

space already allocated for vals array; sampleSize > 0

Returns

reading from the specified obstacle sensor, averaged over sampleSize number of data samples
Returned values are between 0 and 6400
Obstacle values near 0 represent no obstacle seen
Obstacle values may approach 6400 as obstacle gets close.

Warning

As battery degrades, sensor values degrade, yielding systematically lower numbers.

Parameters

<i>sensorName</i>	name of the obstacle sensor
-------------------	-----------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive) designations "center" and "middle" are alternatives for the same light sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

space already allocated for vals array sampleSize > 0

Returns

reading from the specified obstacle sensor, averaged over sampleSize number of data samples
Returned values are between 0 and 6400
Obstacle values near 0 represent no obstacle seen
Obstacle values may approach 6400 as obstacle gets close.

Warning

As battery degrades, sensor values degrade, yielding systematically lower numbers.

4.2.2.24 rGetStall()

```
int rGetStall (
    int sampleSize )
```

Determine if robot has stalled

MyroC Reference Manual (http://wiki.roboteducation.org/Myro_Reference_Manual) states "Every time you issue a move command, the stall sensor resets, and it needs to wait a short time to see whether the motors are stalled. This means that the sensor won't give accurate results if you test it too soon after the robot starts to move."

In practice, it may take 0.5-1.0 seconds for rGetStall to sense the robot is stalled

Parameters

<i>sampleSize</i>	how many readings are taken for each sensor
-------------------	---

Precondition

$sampleSize > 0$

Returns

whether or not robot current has stalled

Postcondition

Returns 1 if the robot has stalled
Returns 0 otherwise.

Determine if robot has stalled

MyroC Reference Manual (http://wiki.roboteducation.org/Myro_Reference_Manual) states Every time you issue a move command, the stall sensor resets, and it needs to wait a short time to see whether the motors are stalled. This means that the sensor won't give accurate results if you test it too soon after the robot starts to move.

In practice, it may take 0.5-1.0 seconds for rGetStall to sense the robot is stalled

Parameters

<i>sampleSize</i>	how many readings are taken for each sensor
-------------------	---

Precondition

$sampleSize > 0$

Returns

whether or not robot current has stalled

Postcondition

Returns 1 if the robot has stalled Returns 0 otherwise.

4.2.2.25 rHardStop()

```
void rHardStop ( )
```

cuts power to the motor of the robot

cuts power to the motor of the robot

4.2.2.26 rLoadPicture()

```
Picture rLoadPicture (
    char * filename )
```

Load a picture from a .jpeg file.

Parameters

<i>filename</i>	the name of the file
-----------------	----------------------

Precondition

file must exist

file extension must be .jpeg or .jpg

file must be no larger than 266 by 427

Returns

Picture

Parameters

<i>filename</i>	the name of the file
-----------------	----------------------

Precondition

file must exist

file extension must be .jpeg or .jpg

file must be no larger than 266 by 427

Returns

locally-declared Picture

Postcondition

calling program should store returned [Picture](#) in a previously-declared struct

4.2.2.27 rMotors()

```
void rMotors (
    double leftSpeed,
    double rightSpeed )
```

move robot with given speeds for the left and right motors continues until given another motion command or disconnected (non-blocking)

Parameters

<i>leftSpeed</i>	the rate at which the left wheel should turn linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forwardward at full speed
<i>rightSpeed</i>	the rate at which the right wheel should turn linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forward at full speed

move robot with given speeds for the left and right motors continues until given another motion command or disconnected (non-blocking)

Parameters

<i>leftSpeed</i>	the rate at which the left wheel should turn linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forwardward at full speed
<i>rightSpeed</i>	the rate at which the right wheel should turn linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forward at full speed

4.2.2.28 rSavePicture()

```
void rSavePicture (
    Picture * pic,
    char * filename )
```

Save a [Picture](#) to a .jpeg.

Parameters

<i>pic</i>	pointer to an RGB picture struct from Scribbler 2 camera
<i>filename</i>	the name of the file

Precondition

filename ends with .jpeg or .jpg.

Postcondition

If the file does not exist, a new file will be created.

If the file exists, the file will be overwritten.

This section contains functions for taking and manipulating images All images are constrained with height ≤ 266 and width ≤ 427

images from robot cameras have varying sizes, depending on the Fluke images for the original Fluke are 192 (height) by 256 (width) low-resolution images for the Fluke 2 are 266 by 427 high-resolution images for the fluke 2 are 800 by 1280 (not supported)

Bluetooth communication constrains the time required for the Fluke to take a picture Typical times: original fluke: 2-4 seconds Fluke 2 (low res): 4-6 seconds Fluke 2 (high res): 25-30 seconds

BASED ON TIMINGS AND MEMORY CONSIDERATIONS, Myro C ALLOWS ONLY LOW RESOLUTION IMAGES

the [Picture](#) struct allows direct access to [Pixel](#) data Pictures can be saved and loaded as .jpeg files

Note

Following standard mathematical convention for a 2D matrix, all references to a pixel are given within an array as [row][col]

user-defined images may have any size, as long as height ≤ 266 and width ≤ 427

Following standard mathematical convention for a 2D matrix, all references to a pixel are given within an array as [row][col]

Warning

The [Picture](#) struct is defined to be sufficiently large to store several low-resolution camera images (340756 bytes each) Experimentally, an array of up to 94 (not 95) Pictures is allowed However, the display of images requires that image data be copied, so display of many images may not work If a program hangs when working with [Picture](#) variables, the issue may involve lack of space on the runtime stack. To utilize a modest number of Pictures, use "ulimit -s" command, as needed, in a terminal window For example, ulimit -s 32768 Sizes above 32768 may not be allowed in Linux or Mac OS X Save a [Picture](#) to a .jpeg

Parameters

<i>pic</i>	pointer to an RGB picture struct from Scribbler 2 camera
<i>filename</i>	the name of the file

Precondition

filename ends with .jpeg or .jpg.

Postcondition

If the file does not exist, a new file will be created.

If the file exists, the file will be overwritten.

this code is a slightly modified version of `write_JPEG_file(char *, int)` from `example.c` from the `libjpeg.v8d` library directory.

4.2.2.29 `rSetBluetoothEcho()`

```
void rSetBluetoothEcho (
    char onOff )
```

Turn on and off the printing transmission and echong of Bluetooth transmissions All robot commands involve the transmission of a command over Bluetooth Scribbler commands are always 9 bytes

Fluke commands have varying lengths

The fluke echos most, but not all, of the commands

For many commands, the fluke also echos 11 bytes of sensor data.

Parameters

<i>onOff</i>	char 'y' enables printing of Bluetooth communications char 'n' disables printng other character values ignored
--------------	--

Turn on and off the printing transmission and echong of Bluetooth transmissions All robot commands involve the transmission of a command over Bluetooth Scribbler commands are always 9 bytes

Fluke commands have varying lengths

The fluke echos most, but not all, of the commands

For many commands, the fluke also echos 11 bytes of sensor data.

Parameters

<i>onOff</i>	char 'y' enables echoing char 'n' disables echoing other character values ignored
--------------	---

4.2.2.30 `rSetConnection()`

```
void rSetConnection (
    ROBOT_CONNECTION_TYPE new_socket_num )
```

set current connection to the socket number

Parameters

<i>new_socket_num</i>	the number of an open socket for communication
-----------------------	--

Precondition

`new_socket_num` has been returned by `rConnect` the designated socket has not been closed

Note

a socket has type int on Linux or Mac; type SOCKET on Windows together these are defined as ROBOT_↔
CONNECTION_TYPE

set current connection to the socket number

Parameters

<i>new_socket_num</i>	the number of an open socket for communication
-----------------------	--

Precondition

new_socket_num has been returned by rConnect the designated socket has not been closed

4.2.2.31 rSetForwardness()

```
void rSetForwardness (
    char * direction )
```

specifies which end of the Scribbler is considered the front

Parameters

<i>direction</i>	identifies front direction
------------------	----------------------------

Precondition

direction is either "fluke-forward" or "scribbler-forward" (not case sensitive)

specifies which end of the Scribbler is considered the front

Parameters

<i>direction</i>	identifies front direction
------------------	----------------------------

Precondition

direction is either "fluke-forward" or "scribbler-forward" (not case sensitive)

4.2.2.32 rSetForwardnessTxt()

```
void rSetForwardnessTxt (
    char * direction )
```

alternative to rSetForwardness for compatibility with earlier MyroC

alternative to rSetForwardness for compatibility with earlier MyroC

4.2.2.33 rSetIRPower()

```
void rSetIRPower (
    int power )
```

Set the amount of power for the dongle's IR sensors.

Parameters

<i>power</i>	the desired power level for the IR sensors
--------------	--

Precondition

power is between 0 and 255 (inclusive)
 Manufacturer notes: default value is 135
 if IR obstacle sensor is always high, try lowering IR power
 if IR obstacle sensor is always low, try raising IR power

Parameters

<i>power</i>	the desired power level for the IR sensors
--------------	--

Precondition

power is between 0 and 255 (inclusive)

4.2.2.34 rSetLEDBack()

```
void rSetLEDBack (
    double led )
```

Set the the intensity of the back fluke LED.

Parameters

<i>led</i>	intensity of the LED values between 0 and 1 provide a range of brightness from off to full intensity values bigger than 1 are treated as 1 (full brightness) values less than 0 are treated as 0 (LED off)
------------	---

Set the the intensity of the back fluke LED,

Parameters

<i>led</i>	intensity of the LED values between 0 and 1 provide a range of brightness from off to full intensity values bigger than 1 are treated as 1 (full brightness) values less than 0 are treated as 0 (LED off).
------------	---

4.2.2.35 rSetLEDFront()

```
void rSetLEDFront (
    int led )
```

Set the front [fluke] LED on or off.

Parameters

<i>led</i>	value 1 turns on LED value 0 turns off LED
------------	--

Precondition

led must be 0 or 1

Set the front [fluke] LED on or off

Parameters

<i>led</i>	value 1 turns on LED value 0 turns off LED
------------	--

Precondition

led must be 0 or 1

4.2.2.36 rSetName()

```
void rSetName (
    const char * name )
```

Change name stored in the robot to the 16-byte name given.

Parameters

<i>name</i>	specifies new name of robot if < 16 bytes given, name is filled with null characters if >= 16 bytes given, name is truncated to 15 bytes plus null Change name stored in the robot to the 16-byte name given
<i>name</i>	specifies new name of robot if < 16 bytes given, name is filled with null characters if >= 16 bytes given, name is truncated to 15 bytes plus null

4.2.2.37 rSetVolume()

```
void rSetVolume (
    char highMute )
```

Set sound to high volume (H) or mute (M) highMute set volume of Scribbler.

Precondition

highMute is 'H' to set for high volume or 'M' for mute

Set sound to high volume (H) or mute (M) highMute set volume of Scribbler.

Precondition

highMute is 'H' Oto set for high volume or 'M' for mute

4.2.2.38 rStop()

```
void rStop ( )
```

directs robot to stop movement

directs robot to stop movement

4.2.2.39 rTakePicture()

```
Picture rTakePicture ( )
```

Use the camera to take a photo.

This section contains functions for taking and manipulating images All images are constrained with height ≤ 266 and width ≤ 427

images from robot cameras have varying sizes, depending on the Fluke images for the original Fluke are 192 (height) by 256 (width) low-resolution images for the Fluke 2 are 266 by 427 high-resolution images for the fluke 2 are 800 by 1280

Bluetooth communication constrains the time required for the Fluke to take a picture Typical times: original fluke: 2-4 seconds Fluke 2 (low res): 4-6 seconds Fluke 2 (high res): 25-30 seconds

BASED ON TIMINGS AND MEMORY CONSIDERATIONS, Myro C ALLOWS ONLY LOW RESOLUTION IMAGES

the [Picture](#) struct allows direct access to [Pixel](#) data Pictures can be saved and loaded as .jpeg files

Note

Following standard mathematical convention for a 2D matrix, all references to a pixel are given within an array as [row][col]

user-defined images may have any size, as long as height ≤ 266 and width ≤ 427

Following standard mathematical convention for a 2D matrix, all references to a pixel are given within an array as [row][col]

Warning

The [Picture](#) struct is defined to be sufficiently large to store several low-resolution camera images (340756 bytes each). Experimentally, an array of up to 94 (not 95) Pictures is allowed. However, the display of images requires that image data be copied, so display of many images may not work. If a program hangs when working with [Picture](#) variables, the issue may involve lack of space on the runtime stack. To utilize a modest number of Pictures, use "ulimit -s" command, as needed, in a terminal window. For example, ulimit -s 32768. Sizes above 32768 may not be allowed in Linux or Mac OS X.

Returns

[Picture](#)

This section contains functions for taking and manipulating images. All images are constrained with height ≤ 266 and width ≤ 427 .

Images from robot cameras have varying sizes, depending on the Fluke. Images for the original Fluke are 192 (height) by 256 (width). Low-resolution images for the Fluke 2 are 266 by 427. High-resolution images for the Fluke 2 are 800 by 1280.

Bluetooth communication constrains the time required for the Fluke to take a picture. Typical times: original fluke: 2-4 seconds. Fluke 2 (low res): 4-6 seconds. Fluke 2 (high res): 25-30 seconds.

BASED ON TIMINGS AND MEMORY CONSIDERATIONS, Myro C ALLOWS ONLY LOW RESOLUTION IMAGES.

The [Picture](#) struct allows direct access to [Pixel](#) data. Pictures can be saved and loaded as .jpeg files.

Note

Following standard mathematical convention for a 2D matrix, all references to a pixel are given within an array as [row][col].
 user-defined images may have any size, as long as height ≤ 266 and width ≤ 427 .
 Following standard mathematical convention for a 2D matrix, all references to a pixel are given within an array as [row][col].

Warning

The [Picture](#) struct is defined to be sufficiently large to store several low-resolution camera images (340756 bytes each). Experimentally, an array of up to 94 (not 95) Pictures is allowed. However, the display of images requires that image data be copied, so display of many images may not work. If a program hangs when working with [Picture](#) variables, the issue may involve lack of space on the runtime stack. To utilize a modest number of Pictures, use "ulimit -s" command, as needed, in a terminal window. For example, ulimit -s 32768. Sizes above 32768 may not be allowed in Linux or Mac OS X. Use the camera to take a photo.

Returns

[Picture](#)

4.2.2.40 rTurnLeft()

```
void rTurnLeft (
    double speed,
    double time )
```

turn Scribbler left for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move left linear range: -1.0 specifies right turn at full speed 0.0 specifies no turn 1.0 specifies left turn at full speed
<i>time</i>	specifies the duration of the turn if negative: the robot starts to turn (non-blocking) other processing proceeds, and the robot continues to turn until given another motion command or disconnected (non-blocking) if zero: robot starts turning (non-blocking); other processing proceeds if positive: robot turns for the given duration, in seconds

turn Scribbler left for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move left linear range: -1.0 specifies right turn at full speed 0.0 specifies no turn 1.0 specifies left turn at full speed
<i>time</i>	specifies the duration of the turn if negative: robot continues to turn until given another motion command or disconnected (non-blocking) if nonnegative: robot turns for the given duration, in seconds

4.2.2.41 rTurnRight()

```
void rTurnRight (
    double speed,
    double time )
```

turn Scribbler right for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move right linear range: -1.0 specifies left turn at full speed 0.0 specifies no turn 1.0 specifies right turn at full speed
<i>time</i>	specifies the duration of the turn if negative: the robot starts to turn (non-blocking) other processing proceeds, and the robot continues to turn until given another motion command or disconnected (non-blocking) if zero: robot starts turning (non-blocking); other processing proceeds if positive: robot turns for the given duration, in seconds if nonnegative: robot turns for the given duration, in seconds

turn Scribbler right for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move right linear range: -1.0 specifies left turn at full speed 0.0 specifies no turn 1.0 specifies right turn at full speed
<i>time</i>	specifies the duration of the turn if negative: robot continues to turn until given another motion command or disconnected (non-blocking) if nonnegative: robot turns for the given duration, in seconds

4.2.2.42 rTurnSpeed()

```
void rTurnSpeed (
    char * direction,
    double speed,
    double time )
```

turn Scribbler in direction for a specified time and speed

Parameters

<i>direction</i>	direction of turn, based on looking from the center of the robot and facing forward
<i>speed</i>	the rate at which the robot should move forward linear range: -1.0 specifies turn at full speed 0.0 specifies no turn 1.0 specifies turn at full speed
<i>time</i>	specifies the duration of the turn if negative: the robot starts to turn (non-blocking) other processing proceeds, and the robot continues to turn until given another motion command or disconnected (non-blocking) if zero: robot starts turning (non-blocking); other processing proceeds if positive: robot turns for the given duration, in seconds

Precondition

direction is "left" or "right", case insensitive

turn Scribbler in direction for a specified time and speed

Parameters

<i>direction</i>	direction of turn, based on looking from the center of the robot and facing forward
<i>speed</i>	the rate at which the robot should move forward linear range: -1.0 specifies turn at full speed 0.0 specifies no turn 1.0 specifies turn at full speed
<i>time</i>	specifies the duration of the turn if negative: robot continues to turn until given another motion command or disconnected if nonnegative: robot turns for the given duration, in seconds

Precondition

direction is "left" or "right", case insensitive

4.2.2.43 rWaitTimedImageDisplay()

```
void rWaitTimedImageDisplay ( )
```

Wait until all timed, non-blocking image window timers are complete.

Postcondition

wait until all timed [non-blocking] images have closed
robot motion is unaffected by this function

Warning

images opened with duration 0.0 do not close until images are updated or until the program terminates
all other [timed, non-blocking] image windows are closed by this function

Postcondition

wait until all timed [non-blocking] images have closed
robot motion is unaffected by this function

Warning

images opened with duration 0.0 do not close until images are updated or until the program terminates
all other image windows are closed by this function

Index

bth_inquiry_record, [5](#)

GLimageset, [5](#)

image_timing_data, [6](#)

motion_timing_data, [7](#)

my_error_mgr, [7](#)

MyroC-image-file.c, [11](#)

 rLoadPicture, [13](#)

 rSavePicture, [14](#)

MyroC.h, [15](#)

 rBackward, [18](#)

 rBeep, [19](#)

 rBeep2, [19](#)

 rConnect, [20](#)

 rDisconnect, [22](#)

 rDisplayImageProcessingErrors, [22](#)

 rDisplayPicture, [23](#)

 rFastForward, [24](#)

 rFinishProcessing, [25](#)

 rForward, [25](#)

 rGetBattery, [26](#)

 rGetBrightAll, [26](#)

 rGetBrightTxt, [27](#)

 rGetForwardness, [28](#)

 rGetIRAll, [29](#)

 rGetIRTxt, [30](#)

 rGetInfo, [29](#)

 rGetLightTxt, [32](#)

 rGetLightsAll, [31](#)

 rGetLine, [33](#)

 rGetName, [35](#)

 rGetObstacleAll, [35](#)

 rGetObstacleTxt, [36](#)

 rGetStall, [37](#)

 rHardStop, [39](#)

 rLoadPicture, [39](#)

 rMotors, [40](#)

 rSavePicture, [40](#)

 rSetBluetoothEcho, [42](#)

 rSetConnection, [42](#)

 rSetForwardness, [43](#)

 rSetForwardnessTxt, [43](#)

 rSetIRPower, [43](#)

 rSetLEDBack, [44](#)

 rSetLEDFront, [45](#)

 rSetName, [45](#)

 rSetVolume, [45](#)

 rStop, [46](#)

 rTakePicture, [46](#)

 rTurnLeft, [47](#)

 rTurnRight, [48](#)

 rTurnSpeed, [49](#)

 rWaitTimedImageDisplay, [49](#)

Picture, [7](#)

Pixel, [8](#)

rBackward

 MyroC.h, [18](#)

rBeep

 MyroC.h, [19](#)

rBeep2

 MyroC.h, [19](#)

rConnect

 MyroC.h, [20](#)

rDisconnect

 MyroC.h, [22](#)

rDisplayImageProcessingErrors

 MyroC.h, [22](#)

rDisplayPicture

 MyroC.h, [23](#)

rFastForward

 MyroC.h, [24](#)

rFinishProcessing

 MyroC.h, [25](#)

rForward

 MyroC.h, [25](#)

rGetBattery

 MyroC.h, [26](#)

rGetBrightAll

 MyroC.h, [26](#)

rGetBrightTxt

 MyroC.h, [27](#)

rGetForwardness

 MyroC.h, [28](#)

rGetIRAll

 MyroC.h, [29](#)

rGetIRTxt

 MyroC.h, [30](#)

rGetInfo

 MyroC.h, [29](#)

rGetLightTxt

 MyroC.h, [32](#)

rGetLightsAll

 MyroC.h, [31](#)

rGetLine

 MyroC.h, [33](#)

rGetName

- MyroC.h, [35](#)
- rGetObstacleAll
 - MyroC.h, [35](#)
- rGetObstacleTxt
 - MyroC.h, [36](#)
- rGetStall
 - MyroC.h, [37](#)
- rHardStop
 - MyroC.h, [39](#)
- rLoadPicture
 - MyroC-image-file.c, [13](#)
 - MyroC.h, [39](#)
- rMotors
 - MyroC.h, [40](#)
- rSavePicture
 - MyroC-image-file.c, [14](#)
 - MyroC.h, [40](#)
- rSetBluetoothEcho
 - MyroC.h, [42](#)
- rSetConnection
 - MyroC.h, [42](#)
- rSetForwardness
 - MyroC.h, [43](#)
- rSetForwardnessTxt
 - MyroC.h, [43](#)
- rSetIRPower
 - MyroC.h, [43](#)
- rSetLEDBack
 - MyroC.h, [44](#)
- rSetLEDFront
 - MyroC.h, [45](#)
- rSetName
 - MyroC.h, [45](#)
- rSetVolume
 - MyroC.h, [45](#)
- rStop
 - MyroC.h, [46](#)
- rTakePicture
 - MyroC.h, [46](#)
- rTurnLeft
 - MyroC.h, [47](#)
- rTurnRight
 - MyroC.h, [48](#)
- rTurnSpeed
 - MyroC.h, [49](#)
- rWaitTimedImageDisplay
 - MyroC.h, [49](#)
- robot_motion_camera_record, [9](#)
- timing_thread_info, [9](#)