MyroC

MyroC.3.1b

Generated by Doxygen 1.8.8

Tue Feb 23 2016 08:24:21

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

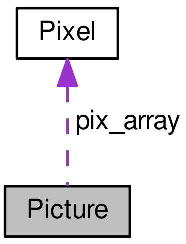
Data Structure Documentation

3.1 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.

3.1.1 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:

- /home/walker/public_html/bluetooth-with-c/MyroC.3.1/MyroC.3.1b/MyroC.h

3.2 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.

3.2.1 Detailed Description

Struct for a pixel.

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

- /home/walker/public_html/bluetooth-with-c/MyroC.3.1/MyroC.3.1b/MyroC.h

Chapter 4

File Documentation

4.1 /home/walker/public_html/bluetooth-with-c/MyroC.3.1/MyroC.3.1b/MyroC.h File Reference

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

Data Structures

- struct Pixel

  Struct for a pixel.
- struct Picture

  Struct for a picture object.

Functions

- int rConnect (const char ∗address)

  connects program to Scribbler
- void rDisconnect ()

  stop Scribbler motion and close Bluetooth
- void rSetConnection (int 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 echoing 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 rWaitTimedImageDisplay ()

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

4.1.1 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 origial 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

4.1.2 Function Documentation

4.1.2.1 void rBackward (double *speed,* double *time*)

moves Scribbler backward for a specified time and speed

Parameters

speed	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
time	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

4.1.2.2 void rBeep (double *duration,* int *frequency*)

Beeps with the given duration and frequency.

Parameters

duration	length of note in seconds
frequency	frequency of pitch in cycles per second (hertz)

Precondition

duration $>$ 0.0

4.1.2.3 void rBeep2 (double *duration,* int *freq1,* int *freq2*)

Generates two notes for the prescribed duration.

Parameters

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

Precondition

duration $>$ 0.0

4.1.2.4 int rConnect (const char $*$ *address*)

connects program to Scribbler

Parameters

address	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

the socket number of communications port

Postcondition

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

4.1.2.5 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

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

Display a picture in a new window.

Parameters

pic	pointer to an RGB picture struct from Scribbler 2 camera
duration	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
windowTitle	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.

4.1.2.7 void rFastForward (double *time*)

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

Parameters

time	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

4.1.2.8 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

4.1.2.9 void rForward (double *speed,* double *time*)

moves Scribbler forward for a specified time and speed

Parameters

speed	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
time	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

4.1.2.10 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

4.1.2.11 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

brightSensors	array to store intensity values
sampleSize	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.

4.1.2.12 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

sensorName	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

sampleSize	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.

4.1.2.13 char∗ rGetForwardness ( )

Gets the forwardness of the Scribbler.

Returns

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

4.1.2.14 void rGetInfo (char ∗ *infoBuffer*)

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

Parameters

infoBuffer	a pre-defined, 60-character array

Postcondition

infoBuffer contains relevant robot information

4.1.2.15 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

irSensors	array to store intensity values
sampleSize	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

4.1.2.16 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

sensorName	name of the light sensor

Precondition

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

Parameters

sampleSize	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

4.1.2.17 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

lightSensors	array to store intensity values
sampleSize	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.

4.1.2.18 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

sensorName	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

sampleSize	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

4.1.2.19 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

lineSensors	array to store line values detected
sampleSize	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

4.1.2.20 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

4.1.2.21 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

obstSensors	array to store intensity values
sampleSize	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.

4.1.2.22 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

sensorName	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

sampleSize	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.

4.1.2.23 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

sampleSize	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.

4.1.2.24 Picture rLoadPicture (char $*$ *filename*)

Load a picture from a .jpeg file.

Parameters

filename	the name of the file

Precondition

file must exist
file must be a 256x192 .jpeg or .jpg

Returns

Picture

4.1.2.25 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

leftSpeed	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
rightSpeed	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

4.1.2.26 void rSavePicture (Picture ∗ *pic,* char ∗ *filename*)

Save a Picture to a .jpeg.

Parameters

pic	pointer to an RGB picture struct from Scribbler 2 camera
filename	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.

4.1.2.27 void rSetBluetoothEcho (char *onOff*)

Turn on and off echoing 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

onOff	char 'y' enables echoing char 'n' disables echoing other character values ignored

4.1.2.28 void rSetConnection (int *new_socket_num*)

set current connection to the socket number

Parameters

new_socket_↩ num	the number of an open socket for communication

Precondition

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

4.1.2.29 void rSetForwardness (char ∗ *direction*)

specifies which end of the Scribbler is considered the front

Parameters

direction	identifies front direction

Precondition

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

4.1.2.30 void rSetIRPower (int *power*)

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

Parameters

power	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

4.1.2.31 void rSetLEDBack (double *led*)

Set the the intensity of the back fluke LED.

Parameters

led	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)

4.1.2.32 void rSetLEDFront (int *led*)

Set the front [fluke] LED on or off.

Parameters

led	value 1 turns on LED value 0 turns off LED

Precondition

led must be 0 or 1

4.1.2.33 void rSetName (const char ∗ *name*)

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

Parameters

name	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

4.1.2.34 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

4.1.2.35 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

4.1.2.36 void rTurnLeft (double *speed*, double *time*)

turn Scribbler left for a specified time and speed

Parameters

speed	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
time	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

4.1.2.37 void rTurnRight (double *speed*, double *time*)

turn Scribbler right for a specified time and speed

Parameters

speed	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
time	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

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

turn Scribbler in direction for a specified time and speed

Parameters

direction	direction of turn, based on looking from the center of the robot and facing forward
speed	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
time	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

4.1.2.39 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

Index