

Programming the MoBot

January 6, 2012

Contents

1	The CMobot MoBot Remote Control Library	4
2	Bluetooth Pairing with the MoBot	4
3	The Robot Remote Control Program	10
3.1	The MoBot Diagram and “Move To Zero” Button	10
3.2	Individual Joint Control	10
3.3	Rolling Control	10
3.4	Joint Speeds	10
3.5	Joint Positions	11
3.6	Motions	11
4	Sample Ch MoBot Programs	11
4.1	Basics of a Ch MoBot Program	11
4.1.1	gettingStarted.ch Source Code	11
4.1.2	Demo Code for gettingStarted.ch Explained	12
4.2	Inchworm Gait Demo	12
4.2.1	inchworm.ch Source Code	13
4.2.2	Demo Code for inchworm.ch Explained	13
4.3	Standing Demo	14
4.3.1	stand.ch Source Code	14
4.3.2	stand.ch Explained	15
5	Preprogrammed Motions	15
5.1	inchworm2.ch: A Demo using the motionInchwormLeft() Preprogrammed Motion	16
5.1.1	inchworm2.ch Source Code	16
5.1.2	inchworm2.ch Explained	17
5.2	stand2.ch: A Demo Using the motionStand() Preprogrammed Motion	17
5.2.1	stand2.ch Source Code	17
5.2.2	stand2.ch Explained	17
6	Blocking and Non-Blocking Functions	17
6.1	List of Blocking Movement Functions	18
6.2	List of Non-Blocking Movement Functions	18
7	Controlling Multiple Modules	19
7.1	multipleModules.ch Source Code	19
7.2	Demo Explanation	19

8	Commanding Multiple Robots to Perform Identical Tasks	20
A	Data Types	22
A.1	robotJointId_t	22
A.2	robotJointState_t	22
A.3	robotJointDirection_t	22
B	Macros	22
B.1	robot_joints_t	22
B.2	robot_joint_direction_t	23
C	CMobot API	23
	connect()	26
	connectWithAddress()	26
	disconnect()	27
	getJointAngle()	27
	getJointMaxSpeed()	28
	getJointSpeed()	28
	getJointSpeedRatio()	29
	getJointSpeedRatios()	29
	getJointSpeeds()	30
	getJointState()	30
	isConnected()	31
	isMoving()	31
	motionInchwormLeft()	32
	motionInchwormLeftNB()	32
	motionInchwormRight()	32
	motionInchwormRightNB()	32
	motionRollBackward()	33
	motionRollBackwardNB()	33
	motionRollForward()	34
	motionRollForwardNB()	34
	motionStand()	34

<code>motionStandNB()</code>	34
<code>motionTurnLeft()</code>	35
<code>motionTurnLeftNB()</code>	35
<code>motionTurnRight()</code>	35
<code>motionTurnRightNB()</code>	35
<code>move()</code>	36
<code>moveNB()</code>	36
<code>moveContinuousNB()</code>	36
<code>moveContinuousTime()</code>	37
<code>moveTo()</code>	38
<code>moveToNB()</code>	38
<code>moveJointTo()</code>	38
<code>moveJointToNB()</code>	38
<code>moveJointWait()</code>	39
<code>moveWait()</code>	40
<code>moveToZero()</code>	40
<code>moveToZeroNB()</code>	40
<code>setJointSpeed()</code>	41
<code>setJointSpeedRatio()</code>	41
<code>setJointSpeedRatios()</code>	42
<code>setJointSpeeds()</code>	43
<code>setTwoWheelRobotSpeed()</code>	43
<code>stop()</code>	44

1 The CMobot MoBot Remote Control Library

The **CMobot** library is a collection of functions geared towards controlling the motors and reading sensor values of a MoBot module via the Bluetooth wireless protocol. The functions are designed to be intuitive and easy to use. Various functions are provided to control or obtain the speed, direction, and position of the motors. The API includes C-style functions as well as a C++ class called **CMobot** to facilitate C++ style api function calls.

This documentation introduces the basic computer setup required for controlling the MoBot, as well as several demo programs and a complete reference for all API function provided with the **CMobot** library.

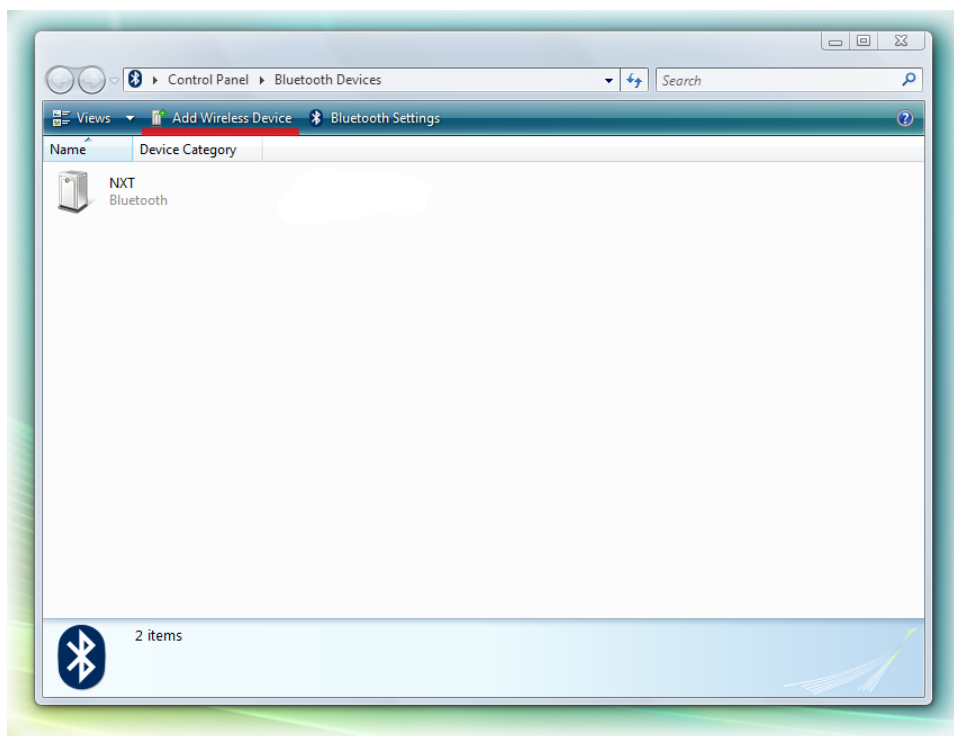
2 Bluetooth Pairing with the MoBot

To control the MoBot with the Bluetooth wireless protocol, the controlling computer must be equipped with Bluetooth. If the computer does not have Bluetooth built-in, an external USB Bluetooth dongle may be used. The following instructions are for a Windows 7 computer with built-in Bluetooth. The basic process is the same for Windows XP and Vista, although the screenshots may appear different.

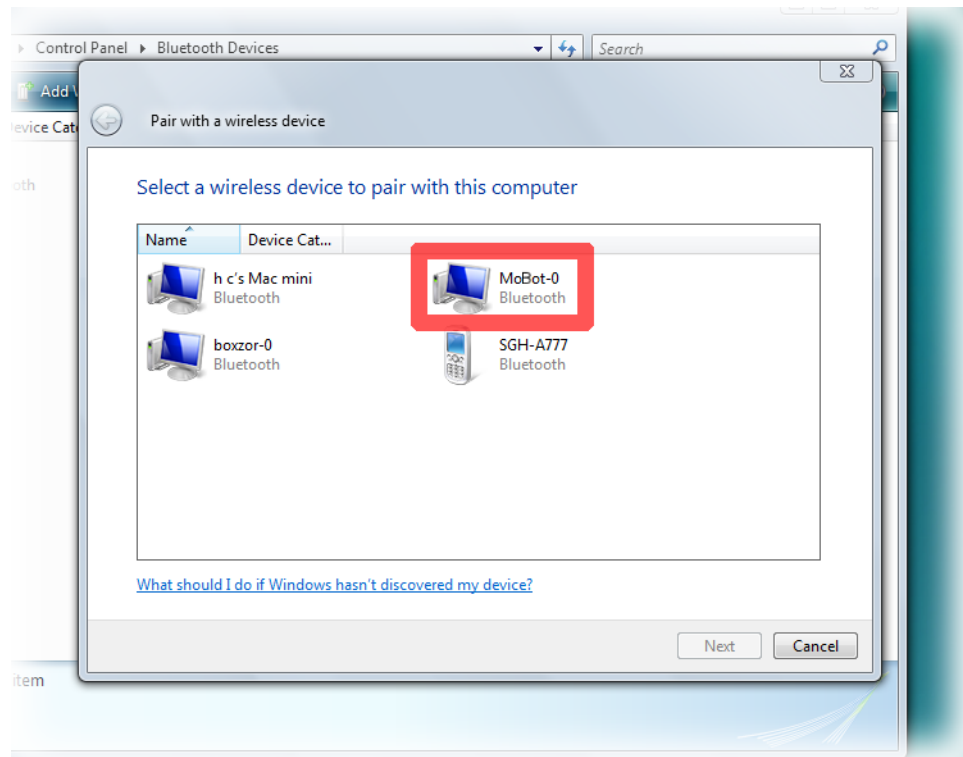
The first step is to open the Bluetooth applet by double-clicking the Bluetooth icon in the applet tray normally found at the bottom right of the screen, as shown highlighted in red in the following figure.



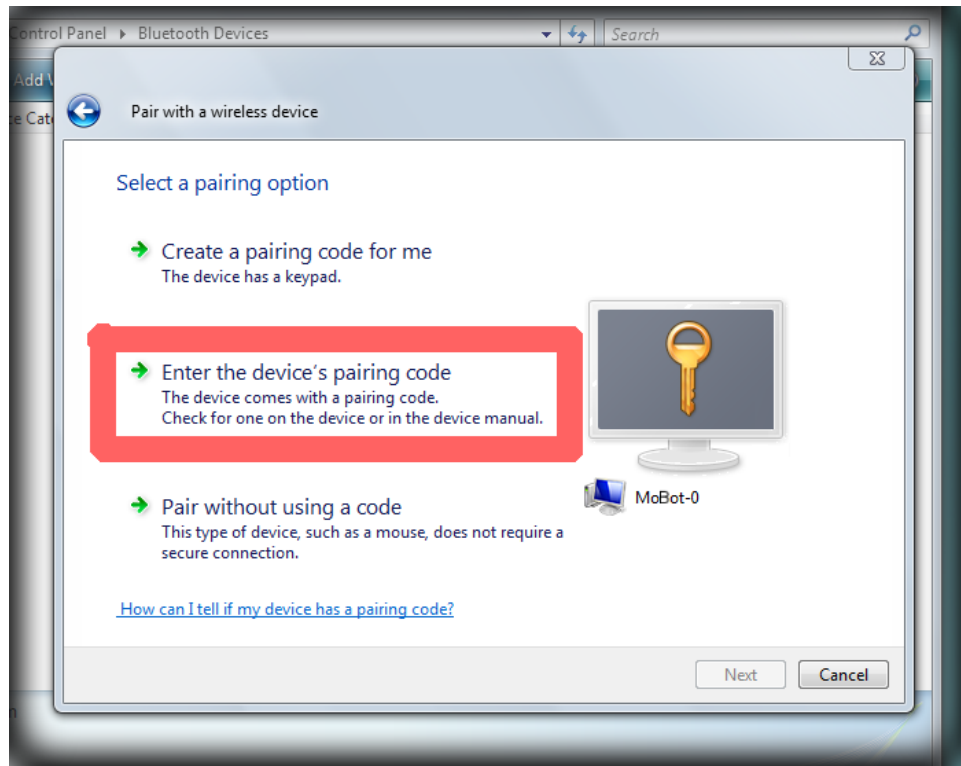
After double-clicking the icon, a new window will appear similar to the following.



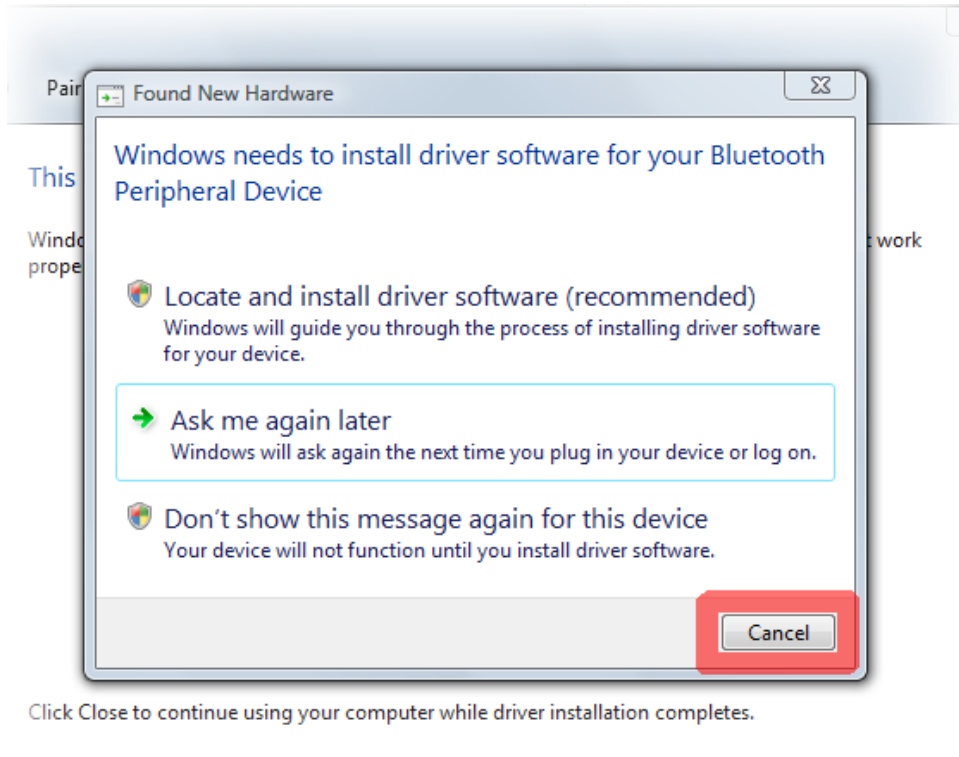
Next, click on the button labeled "Add Wireless Device" towards the top of the window. This will bring up the following dialog.



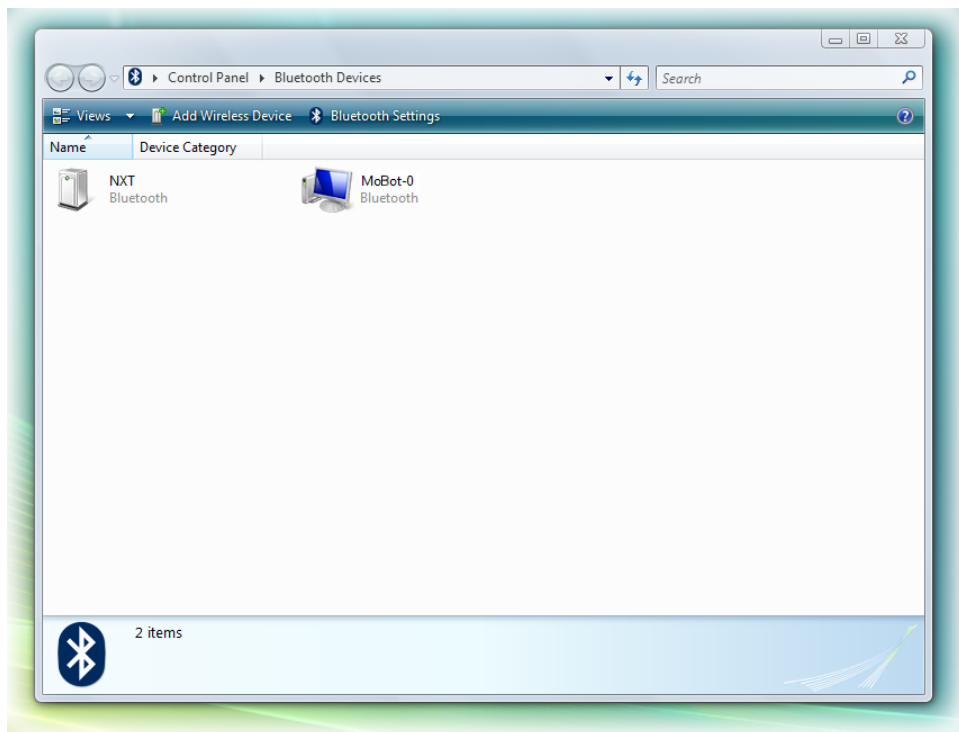
This dialog shows a list of all Bluetooth wireless devices that are in range. Among them should be the MoBot you wish to connect to. If the MoBot does not appear on this list, please ensure that the MoBot is within 10 meters of the connecting computer and that the MoBot is powered on. Double click on the icon representing the MoBot to proceed. Once you have double-clicked the icon, the following dialog box should appear.



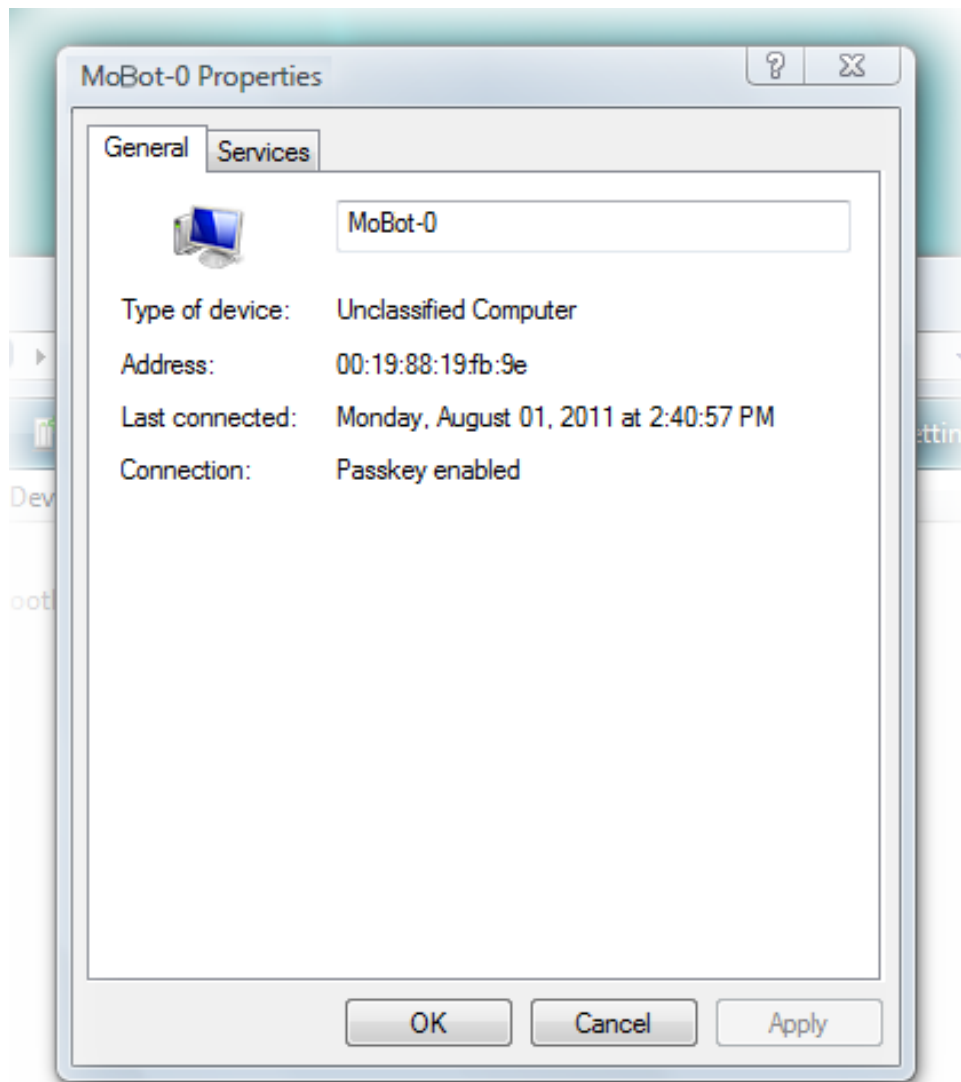
Select the second option, labeled “Enter the device’s pairing code”. MoBot modules come hard-coded with a default pairing code. When prompted for the pairing code, enter “1234”. Once the computer is paired with the MoBot, the following dialog box may pop up asking to install drivers.



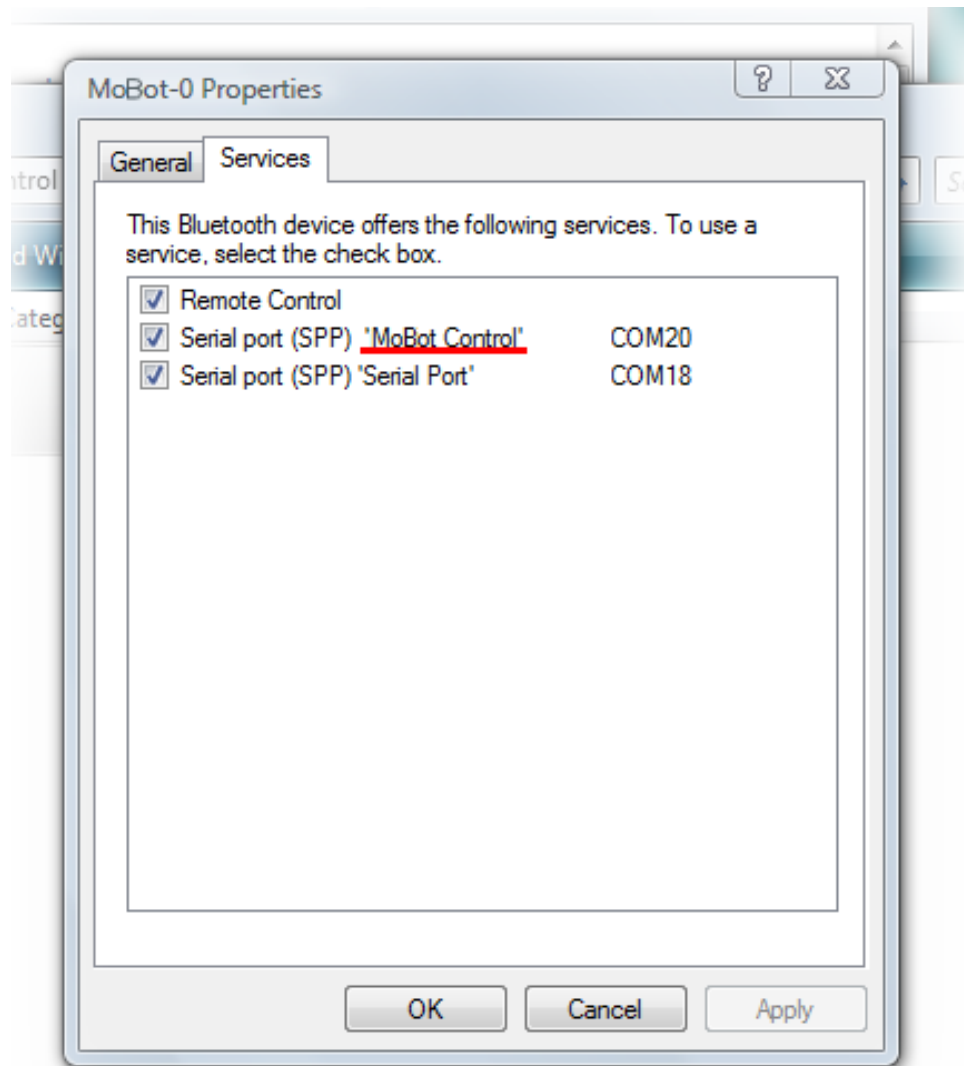
If the previously illustrated dialog box appears, just click the “cancel” button at the bottom right. No extra drivers are necessary for controlling the MoBot module.
At this point, the following dialog should be shown.



The next step is to enable the MoBot control service. Double-click on the icon denoting the MoBot module to bring up the following dialog:

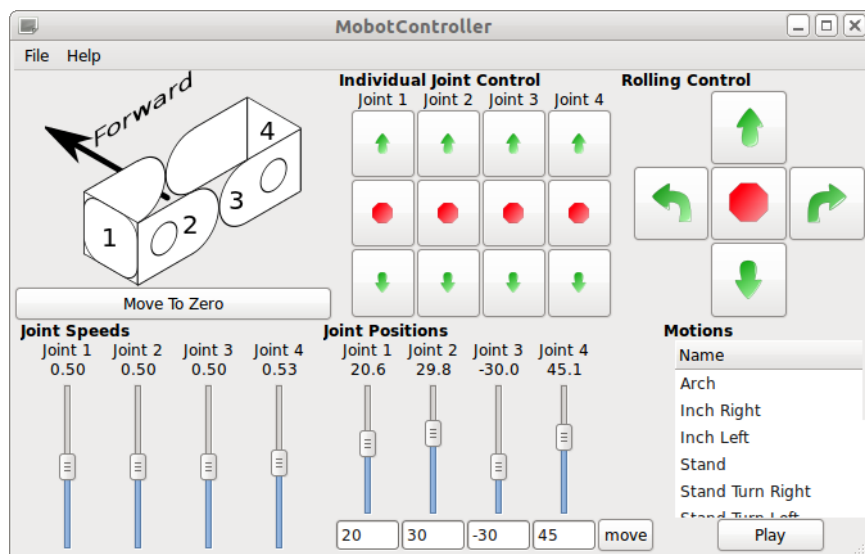


Click on the tab labeled "Services"



Ensure that the service titled “MoBot Control” is enabled. If it is not enabled, click on the check-box to enable it. Click on the “Ok” button to accept the changes and close the dialogs. The MoBot is now ready to be controlled with the CMobot library.

3 The Robot Remote Control Program



The preceding figure shows the MoBot remote control program. The program displays a graphical interface which may be used to display information about the MoBot's joint positions, and also control the speeds and positions of the MoBot's joints. The interface is divided up into six sections; three on the top half of the interface, and three on the bottom half.

3.1 The MoBot Diagram and “Move To Zero” Button

The first section of the GUI located on the top left of the interface displays a schematic diagram of the MoBot, displaying motor positions. Underneath the diagram, there is a large button with the text “Move To Zero”. When clicked, this button will command the connected MoBot to rotate all of its joints to a flat “Zero” position.

3.2 Individual Joint Control

The second section, located at the top-middle section of the interface, is the “Individual Joint Control” section. These buttons command the MoBot to move individual joints. When the up or down arrows are clicked, the MoBot begins to move the corresponding joint in either the positive, or negative direction. The joint will continue to move until the stop button, located between the up and down arrows, is clicked.

3.3 Rolling Control

This section contains buttons for controlling the MoBot as a two wheeled mobile robot. The up and down buttons cause the MoBot to roll forward or backward. The left and right buttons cause the MoBot to rotate towards the left, or towards the right. The stop button in the middle causes the MoBot to stop where it is.

3.4 Joint Speeds

The “Joint Speeds” section, located at the bottom left of the interface, displays and controls the current joint speeds of the MoBot. Joint speeds are a value between 0 and 1, with 1 meaning maximum joint power, and 0 meaning zero joint power. The speed may be set by sliding the vertical sliders to the desired positions.

3.5 Joint Positions

This section, located in the bottom-middle of the interface, is used to display and control the positions of each of the four joints of a MoBot. The joint positions are displayed in the numerical text located above each vertical slider. The displayed joint positions are in units of degrees. There are two methods to control the joints using this interface.

The first method of controlling the joints is by using the vertical sliders. Each vertical slider's position represents a joint's angle. The sliders for the two end joints vary from -180 degrees to 180 degrees, representing one complete rotation. The angles for the two body joints vary from -90 to 90 degrees. When the position of the slider is moved, the MoBot will move its joints to match the sliders.

The second method for moving the joints is by entering the exact angles for the joints. Below each of the four sliders lies a text entry box. Values in degrees may be typed into each of the four entry boxes. When the button on the lower right of the section labeled "Move" is clicked, the MoBot will move its joints to match the values typed into the boxes. If no value is typed into a box, that joint will not move.

3.6 Motions

This section, located on the bottom right of the interface, contains a set of preprogrammed motions for the MoBot. To execute a preprogrammed motion, simply click on the name of the motion you wish to execute, and then click the button labeled "Play".

4 Sample Ch MoBot Programs

To help the user become acquainted with the MoBot control programs, sample programs will be presented to illustrate the basics and minimum requirements of a MoBot control program.

4.1 Basics of a Ch MoBot Program

The first demo presents a minimal program which connects to a MoBot and moves some joints.

4.1.1 gettingStarted.ch Source Code

```
/* Filename: gettingStarted.ch
 * Move the robot endplates. */

#include <mobot.h>

CMobot robot;
double angle1, angle4; // Angles for joints 1 and 4

/* Connect to the paired MoBot */
robot.connect();

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* Rotate each of the faceplates by 90 degrees */
angle1 = deg2rad(90);
angle4 = deg2rad(90);
robot.move(angle1, 0, 0, angle4);
```

4.1.2 Demo Code for `gettingStarted.ch` Explained

The beginning of every MoBot control program will include header files. Each header file imports functions used for a number of tasks, such as printing data onto the screen or controlling the MoBot.

```
#include <mobot.h> // Required for MoBot control functions
```

Next, we must initialize the C++ class used to control the MoBot. This line initializes a new variable named `robot` which represents the remote MoBot module which we wish to control. This special variable is actually an instance of the `CMobot` class, which contains its own set of functions called “methods” or “member functions”.

```
CMobot robot;
```

Next, we initialize two `double` type variables called “`angle1`” and “`angle4`”, which we will use to store some joint angles.

```
double angle1, angle4;
```

The next line will connect our new variable, `robot`, to a MoBot that has been previously paired with the computer.

```
robot.connect();
```

Note that there are two common methods to connect to a remote MoBot. The most common method, demonstrated in the previous line of code, is used to connect to a MoBot that is already paired to the computer. It is also possible to connect to MoBots which are not paired with the computer. This method is necessary for connecting to multiple MoBots simultaneously, as only a single MoBot may be paired with the computer at a time. The second method uses the function `connectWithAddress()`, and its default usage is as such:

```
robot.connectWithAddress("11:22:33:44:55:66", 20);
```

The string “`11:22:33:44:55:66`” represents the Bluetooth address of the MoBot, which must be known in advance. The number 20 represents the Bluetooth channel to connect to. Channel 20 is the default channel MoBots listen on for incoming connections, but may be set to other values.

The next line will use the `moveToZero` member function. The `moveToZero` function causes the MoBot to move all of its motors to the zero position.

```
robot.moveToZero();
```

The next lines of code command joints 1 and 4 to rotate 90 degrees. Note that the member function `move()` expects input angles in radians, so the angles in degrees must be converted to radians using the `deg2rad()` function. If desired, values in radians may also be converted to degrees using the counterpart function, `rad2deg()`. Joints 1 and 4 are the faceplates of the MoBot which are sometimes used to act as “wheels”.

```
angle1 = deg2rad(90);  
angle4 = deg2rad(90);  
robot.move(angle1, 0, 0, angle4);
```

4.2 Inchworm Gait Demo

The next demo will demonstrate a simple gait known as the “Inchworm” gait.

4.2.1 inchworm.ch Source Code

```
/* File: inchworm.ch
 * Perform the "inchworm" gait four times */

#include <mobot.h>

CMobot robot;

/* Connect to the paired MoBot */
robot.connect();

/* Set robot motors to speed of 0.50 */
robot.setJointSpeedRatio(MOBOT_JOINT2, 0.50);
robot.setJointSpeedRatio(MOBOT_JOINT3, 0.50);

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* Do the inchworm gait four times */
int i;
for(i = 0; i < 4; i++) {
    robot.moveJointTo(MOBOT_JOINT2, deg2rad(-45));
    robot.moveJointTo(MOBOT_JOINT3, deg2rad(45));
    robot.moveJointTo(MOBOT_JOINT2, 0);
    robot.moveJointTo(MOBOT_JOINT3, 0);
}
```

4.2.2 Demo Code for inchworm.ch Explained

The first portion of the code is identical to the previous demo, and performs the same function of declaring a MoBot variable and connecting to a paired MoBot.

```
#include <mobot.h>
```

```
CMobot robot;
```

```
/* Connect to the paired MoBot */
robot.connect();
```

The next lines of code set the joint speeds for the two body joints, joints two and three, to 50% speed. They are set to fifty percent speed in order to slow the gait down in order to minimize slippage.

```
/* Set robot motors to speed of 0.50 */
robot.setJointSpeedRatio(MOBOT_JOINT2, 0.50);
robot.setJointSpeedRatio(MOBOT_JOINT3, 0.50);
```

Next, we move the robot into a flat “zero” position.

```
/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();
```

Finally, we perform the actual inchworm gait. The inchworm gait is a gait defined by a sequence of motions performed by the body joints. The motions are as such:

1. The first body joint, referred to as joint A, rotates towards the ground. This drags the MoBot towards the direction of joint A.
2. The other body joint, joint B, rotates towards the ground. Since the center of gravity is currently positioned over joint A, this causes the trailing body joint to slide toward joint A.
3. Joint A moves back to a flat position.
4. Joint B moves back to a flat position.
5. Repeat, if desired.

The direction of travel depends on the selection of the initial body joint. In the following code example, joint 2 is chosen as the initial body joint to move. In this case, the MoBot will traverse towards joint 2. The entire gait is encapsulated in a “for” loop which executes the entire gait four times.

```
/* Do the inchworm gait four times */
int i;
for(i = 0; i < 4; i++) {
    robot.moveJointTo(ROBOT_JOINT2, deg2rad(-45));
    robot.moveJointTo(ROBOT_JOINT3, deg2rad(45));
    robot.moveJointTo(ROBOT_JOINT2, 0);
    robot.moveJointTo(ROBOT_JOINT3, 0);
}
```

4.3 Standing Demo

4.3.1 stand.ch Source Code

```
/* Filename: stand.ch
 * Make a MoBot stand up on a faceplate */

#include <mobot.h>

CMobot robot;

/* Connect to the paired MoBot */
robot.connect();

/* Set robot motors to slow speed of 10 degrees per second */
robot.setJointSpeed(MOBOT_JOINT2, deg2rad(10));
robot.setJointSpeed(MOBOT_JOINT3, deg2rad(10));
/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* Move the robot into a fetal position */
robot.moveJointTo(MOBOT_JOINT2, deg2rad(-85));
robot.moveJointTo(MOBOT_JOINT3, deg2rad(80));

/* Rotate the bottom faceplate by 45 degrees */
robot.moveJointTo(MOBOT_JOINT1, deg2rad(45));
```

```

/* Lift the body up */
robot.moveJointTo(MOBOT_JOINT2, deg2rad(20));

/* Pan the robot around for 3 seconds at 90 degrees per second*/
robot.setJointSpeed(MOBOT_JOINT1, deg2rad(90));
robot.moveContinuousTime(
    MOBOT_FORWARD,
    MOBOT_NEUTRAL,
    MOBOT_NEUTRAL,
    MOBOT_NEUTRAL,
    3000);

```

4.3.2 stand.ch Explained

The first portion of the program performs the necessary setup and connecting, similar to the previous demos. Similar to the previous inchworm demo, the motor speeds are set to a speed of 10 degrees per second, and the function `moveToZero()` is called to put the robot into a flat position. Next, the following lines are executed:

```

robot.moveJointTo(ROBOT_JOINT2, deg2rad(-85));
robot.moveJointTo(ROBOT_JOINT3, deg2rad(80));

```

These movement commands cause the MoBot to curl up into a fetal position with both of its endplates facing toward the ground. Next, the MoBot rotates one of the endplates by 45 degrees.

```

robot.moveJointTo(ROBOT_JOINT1, deg2rad(45));

```

This endplate will eventually become the “foot” of the standing MoBot. Next, the MoBot lifts itself into a standing position, balancing on its endplate.

```

robot.moveJointTo(ROBOT_JOINT2, deg2rad(20));

```

Note that the previous joint angle for Joint 2, a body joint, was -85 degrees. This motion causes joint 2 to rotate all the way to a 20 degree position, which lift up the body of the MoBot such that the MoBot is balancing on faceplate joint 1.

Finally, we rotate joint 1, the foot joint, for three seconds which causes the entire MoBot to rotate in place. The speed is first set to 30% speed to make the rotation a slow rotation. Next, the `moveContinuousTime` member function is used to continuously rotate a joint for a desired amount of time.

```

robot.setJointSpeed(ROBOT_JOINT1, 0.30);
robot.moveContinuousTime(
    ROBOT_FORWARD,
    ROBOT_NEUTRAL,
    ROBOT_NEUTRAL,
    ROBOT_NEUTRAL,
    3000);

```

5 Preprogrammed Motions

The robot API contains functions for executing preprogrammed motions. The preprogrammed motions are motions which are commonly used for robot locomotion. Following is a list of available functions and a brief description about their effect on the robot.

- `motionInchwormLeft()`: This function causes the robot to perform the inchworm gait once, moving the robot towards its left.
- `motionInchwormRight()`: This function causes the robot to perform the inchworm gait once, moving the robot towards its right.
- `motionRollBackward()`: This function causes the robot to rotate its faceplates, using them as wheels to roll backward.
- `motionRollForward()`: This function causes the robot to rotate its faceplates, using them as wheels to roll forward.
- `motionStand()`: This function causes the robot to stand up onto a faceplate, assuming the camera platform position.
- `motionTurnLeft()`: Uses the robot's faceplates as wheels, turning them in opposite directions in order to rotate the robot towards its left.
- `motionTurnRight()`: Uses the robot's faceplates as wheels, turning them in opposite directions in order to rotate the robot towards its right.

Note that all of the functions listed above are “blocking” functions, meaning they will not return until the motion has completed. These functions also have non-blocking equivalents which are discussed in Section 6.

5.1 inchworm2.ch: A Demo using the `motionInchwormLeft()` Preprogrammed Motion

This demo performs the same motion as the previous `inchworm.ch` demo, except it uses a built-in preprogrammed motion.

5.1.1 inchworm2.ch Source Code

```
/* File: inchworm.ch
 * Perform the "inchworm" gait four times */

#include <mobot.h>

CMobot robot;

/* Connect to the paired MoBot */
robot.connect();

/* Set robot motors to speed of 0.50 */
robot.setJointSpeedRatio(MOBOT_JOINT2, 0.50);
robot.setJointSpeedRatio(MOBOT_JOINT3, 0.50);

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* Do the inchworm gait four times */
int i;
for(i = 0; i < 4; i++) {
    robot.motionInchwormLeft();
}
```



```
}
```

5.1.2 inchworm2.ch Explained

This program is identical to the previously discussed `inchworm.ch` demo, except for the final lines of the program, which appear as such:

```
int i;
for(i = 0; i < 4; i++) {
    robot.motionInchwormLeft();
}
```

Note that instead of controlling each joint in sequence explicitly, the function `motionInchwormLeft()` is used instead. The `motionInchwormLeft()` function automatically executes the inchworm gait on the MoBot module, and the encapsulating `for` loop executes the motion four times.

5.2 stand2.ch: A Demo Using the `motionStand()` Preprogrammed Motion

This demo is a simple demonstration of the `motionStand()` member function.

5.2.1 stand2.ch Source Code

```
/* Filename: stand2.ch
 * Make a MoBot stand up on a faceplate */
#include <mobot.h>

CMobot robot;
/* Connect to the paired MoBot */
robot.connect();
/* Run the built-in motionStand function */
robot.motionStand();
```

5.2.2 stand2.ch Explained

The source code for this demo is straightforward. After the initialization and connection as seen in previous demos, it executes the following line of code:

```
robot.motionStand();
```

This line of code causes the MoBot to perform a sequence of motions causing it to stand up on a faceplate. The function is a blocking function and will wait until the entire motion sequence is completed before continuing. There are also non-blocking versions of the motion functions, documented in Section 6.

6 Blocking and Non-Blocking Functions

All of the MoBot movement functions may be designated as either “blocking” functions or “non-blocking” functions. A blocking function is a function which does not return while operations are being performed. For instance, the `moveWait()` function is a blocking function. When called, the function will hang, or “block”, until all the joints have stopped moving. After all joints have stopped moving, the `moveWait()` function will return, and the rest of the program will execute.

Furthermore, some functions have both a blocking version and a non-blocking version. For these functions, the suffix **NB** denotes that the function is non-blocking. For instance, the function `motionStand()` is blocking, meaning the function will not return until the motion is completed, whereas the function `motionStandNB()` is non-blocking, meaning the function returns immediately and the robot performs the “standing” motion asynchronously.

The function `moveNB()` is an example of a non-blocking function. When the `moveNB()` function is called, the function immediately returns as the joints begin moving. Any lines of code following the call to `moveNB()` will be executed even if the current motion is still in progress.

Demos for the non-blocking functions are located in the next section of this document.

6.1 List of Blocking Movement Functions

- `move()`
- `moveContinuousTime()`
- `moveJointTo()`
- `moveTo()`
- `moveToZero()`
- `moveJointWait()`
- `moveWait()`
- `motionInchwormLeft()`
- `motionInchwormRight()`
- `motionRollBackward()`
- `motionRollForward()`
- `motionStand()`
- `motionTurnLeft()`
- `motionTurnRight()`

6.2 List of Non-Blocking Movement Functions

- `moveNB()`
- `moveContinuousNB()`
- `moveJointToNB()`
- `moveToNB()`
- `moveToZeroNB()`
- `motionInchwormLeftNB()`
- `motionInchwormRightNB()`
- `motionRollBackwardNB()`
- `motionRollForwardNB()`

- motionStandNB()
- motionTurnLeftNB()
- motionTurnRightNB()

7 Controlling Multiple Modules

The MoBot control software is designed to be able to control multiple modules simultaneously. There are a couple important differences in the program which enable the control of multiple modules. A small demo program which controls two modules simultaneously will first be presented, followed by a detailed explanation of the program elements.

7.1 multipleModules.ch Source Code

```
/* Filename: multipleModules.ch
 * Control two modules and make them stand simultaneously. */

#include <mobot.h>

CMobot robot1;
CMobot robot2;

/* For multiple robots, we need to use the "connectWithAddress" function to
 * connect to separate robots. Substitute the string "11:11:11:11:11:11" with
 * the address of the first MoBot and the string "22:22:22:22:22:22" with the
 * string of the second MoBot.*/
int defaultChannel = 1;
string_t bluetoothAddress1 = "11:11:11:11:11:11";
string_t bluetoothAddress2 = "22:22:22:22:22:22";
robot1.connectWithAddress(bluetoothAddress1, defaultChannel);
robot2.connectWithAddress(bluetoothAddress2, defaultChannel);

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot1.moveToZeroNB();
robot2.moveToZeroNB();

robot1.moveWait();
robot2.moveWait();

/* Instruct the first robot to stand and the second robot to inchworm
 * simultaneously. */
robot1.motionStandNB();
robot2.motionInchwormLeftNB();
robot1.moveWait();
robot2.moveWait();
```

7.2 Demo Explanation

The first two lines of interest appear as such:

```
CMobot robot1;  
CMobot robot2;
```

These two lines declare two separate variables which will represent the two separate MoBot modules. Next, we need to connect each variable to a physically separate MoBot. This is done with the following lines.

```
robot1.connectWithAddress("11:11:11:11:11:11", 20);  
robot2.connectWithAddress("22:22:22:22:22:22", 20);
```

These lines connect the first variable, `robot1`, to the MoBot with address `11:11:11:11:11:11`. When running this demo, this demo address will need to be replaced with the actual address of the MoBot. The second argument, `20`, is the channel to connect to. By default, the MoBot will listen on channel `20` for incoming connections.

A similar process is done with `robot2`, causing it to connect to a second MoBot with address `22:22:22:22:22:22`.

```
robot1.moveToZeroNB();  
robot2.moveToZeroNB();
```

These two lines command the two robots to move to their zero positions. Note that these functions are non-blocking. This means that the `moveToZeroNB()` function will return immediately, and will not wait for the first robot to finish completing the motion before commanding the second robot to begin. In a normal program, this effectively causes both robots to move to their zero positions simultaneously.

```
robot1.moveWait();  
robot2.moveWait();
```

Since the `moveToZeroNB()` functions are non-blocking, we would like the program to wait until the motions are complete before continuing. By calling `moveWait()` on both of the robots, we can be assured that the robots have finished moving before the program continues.

```
robot1.motionStandNB();  
robot2.motionInchwormLeftNB();  
robot1.moveWait();  
robot2.moveWait();
```

Similar to the calls to `moveToZeroNB()`, this block of code instructs the first MoBot to stand and the second MoBot to perform the inchworm gait. Note that we call the non-blocking versions of the functions, `motionStandNB()` and `motionInchwormLeftNB`. Since these functions are non-blocking, both robots will effectively perform the motions simultaneously. Again, we call the `moveWait()` function on both robots to ensure that the robots have finished standing before the code continues.

8 Commanding Multiple Robots to Perform Identical Tasks

There is a specialized class called `CMobotGroup` which may be used to control multiple modules simultaneously. The `CMobotGroup` represents a group of robots. Any command that is given to the group of modules is duplicated to each member of the group.

The majority of the movement functions available in the `CMobot` class are also available in the `CMobotGroup` class. For documentation of each member function, please consult the documentation for the corresponding functions in the `CMobot` class. Following is a complete listing of the available member functions in the `CMobotGroup` class.

Function Name	Description
<code>addRobot()</code>	Adds a robot to the group.
<code>move()</code>	For the remaining functions, please see corresponding <code>CMobot</code> member function.
<code>moveContinuousNB()</code>	
<code>moveContinunousTime()</code>	
<code>moveJointContinuousNB()</code>	
<code>moveJointContinuousTime()</code>	
<code>moveJointTo()</code>	
<code>moveJointWait()</code>	
<code>moveTo()</code>	
<code>moveWait()</code>	
<code>moveToZero()</code>	
<code>setJointSpeed()</code>	
<code>setJointSpeedRatio()</code>	
<code>setTwoWheelRobotSpeed()</code>	
<code>stop()</code>	
<code>motionInchwormLeft()</code>	
<code>motionInchwormLeftNB()</code>	
<code>motionInchwormRight()</code>	
<code>motionInchwormRightNB()</code>	
<code>motionRollBackward()</code>	
<code>motionRollForwardNB()</code>	
<code>motionStand()</code>	
<code>motionStandNB()</code>	
<code>motionTurnLeft()</code>	
<code>motionTurnLeftNB()</code>	
<code>motionTurnRight()</code>	
<code>motionTurnRightNB()</code>	

A Data Types

There are data types which are used by the MoBot library to represent certain values, such as joint id's and motor directions.

Data Type	Description
<code>robotJointId_t</code>	An enumerated value that indicates a MoBot joint.
<code>robotJointState_t</code>	The current state of a MoBot joint.
<code>robotJointDirection_t</code>	The current motion direction of a MoBot joint.

A.1 `robotJointId_t`

This datatype is an enumerated type used to identify a joint on the MoBot. Valid values for this type are:

```
typedef enum mobot_joints_e {  
    ROBOT_JOINT1 = 1,  
    ROBOT_JOINT2 = 2,  
    ROBOT_JOINT3 = 3,  
    ROBOT_JOINT4 = 4  
} robotJointId_t;
```

The joints are enumerated in order from one side of the MoBot to the other. Joints 1 and 4 are faceplate joints, and joints 2 and 3 are body joints.

A.2 `robotJointState_t`

This datatype is an enumerated type used to designate the current movement state of a joint. Valid values are:

- `ROBOT_JOINT_IDLE`: This value indicates that the joint is not moving.
- `ROBOT_JOINT_MOVING`: This value indicates that the joint is currently moving.
- `ROBOT_JOINT_GOALSEEK`: This value indicates that the joint is currently moving towards a predefined goal.

A.3 `robotJointDirection_t`

This datatype designates a MoBot joint's commanded direction of travel. Valid values are

- `ROBOT_NEUTRAL`: There is no predesignated direction for the joint. If the joint is commanded to move to a specific location, the MoBot will decide the best direction to move the joint in order to achieve the goal with the smallest motion.
- `ROBOT_FORWARD`: Move the joint in the direction which increases its angular position.
- `ROBOT_BACKWARD`: Move the joint in the direction which decreases its angular position.

B Macros

B.1 `robot_joints_t`

The data type `robot_joints_t` contains the following macro datatypes.

Value	Description
ROBOT_JOINT1	Joint number 1 on the MoBot, which is a face-plate joint.
ROBOT_JOINT2	Joint number 2 on the MoBot, which is a body joint.
ROBOT_JOINT3	Joint number 3 on the MoBot, which is a body joint.
ROBOT_JOINT4	Joint number 4 on the MoBot, which is a face-plate joint.

B.2 robot_joint_direction_t

The data type `robot_joint_direction_t` indicates the commanded direction of a joint on the MoBot.

Value	Description
ROBOT_NEUTRAL	This value indicates automatic direction control for a joint. The MoBot will choose the best direction to attain the commanded joint position.
ROBOT_FORWARD	Move the joint in the forward direction.
ROBOT_BACKWARD	Move the joint in the backward direction.

C CMobot API

The header file `mobot.h` defines all the data types, macros and function prototypes for the robot API library. The header file declares a class called `CMobot` which contains member functions which may be used to control the robot.

Table 1: CMobot Member Functions.

Function	Description
<code>CMobot()</code>	The CMobot constructor function. This function is called automatically and should not be called explicitly.
<code>~CMobot()</code>	The CMobot destructor function. This function is called automatically and should not be called explicitly.
<code>connect()</code>	Connect to a remote robot module. This function connects to the first robot listed in the Barobo configuration file. To edit the configuration file, use the robot control graphical user interface, and select the menu item “Robot → Configure Robot Bluetooth”.
<code>connectWithAddress()</code>	Connect to a robot module by specifying its Bluetooth address.
<code>disconnect()</code>	Disconnect from a robot module.
<code>getJointAngle()</code>	Gets a joint’s angle.
<code>getJointMaxSpeed()</code>	Gets a joint’s maximum speed in radians per second.
<code>getJointSpeed()</code>	Gets a motor’s current speed setting in radians per second.
<code>getJointState()</code>	Gets a motor’s current status.
<code>isConnected()</code>	This function is used to check the connection to a robot.
<code>move()</code>	Move all four joints of the robot by specified angles.
<code>moveNB()</code>	Identical to <code>move()</code> but non-blocking.
<code>moveContinuousNB()</code>	Move joints continuously. Joints will move until stopped.
<code>moveContinuousTime()</code>	Move joints continuously for a certain amount of time.
<code>moveTo()</code>	Move all four joints of the robot to specified absolute angles.
<code>moveToNB()</code>	Identical to <code>moveTo()</code> but non-blocking.
<code>moveJointTo()</code>	Set the desired motor position.
<code>moveJointToNB()</code>	Identical to <code>moveJointTo()</code> but non-blocking.
<code>moveJointWait()</code>	Wait until the specified motor has stopped moving.
<code>moveWait()</code>	Wait until all motors have stopped moving.
<code>moveToZero()</code>	Instructs all motors to go to their zero positions.
<code>moveToZeroNB()</code>	Identical to <code>moveToZero()</code> but non-blocking.
<code>setJointSpeed()</code>	Sets a motor’s speed setting in radians per second.
<code>setJointSpeedRatio()</code>	Set a joints speed setting to a fraction of its maximum speed.
<code>stop()</code>	Stop all currently executing motions of the robot.

Table 2: CMobot Member Functions for Compound Motions.

Compound Motions	These are convenience functions of commonly used compound motions.
<code>motionInchwormLeft()</code>	Inchworm gait towards the left.
<code>motionInchwormLeftNB()</code>	Identical to <code>motionInchwormLeft</code> but non-blocking.
<code>motionInchwormRight()</code>	Inchworm gait towards the right.
<code>motionInchwormRightNB()</code>	Identical to <code>motionInchwormRight</code> but non-blocking.
<code>motionRollBackward()</code>	Roll on the faceplates toward the backward direction.
<code>motionRollBackwardNB()</code>	Identical to <code>motionRollBackward()</code> but non-blocking.
<code>motionRollForward()</code>	Roll on the faceplates forwards.
<code>motionRollForwardNB()</code>	Identical to <code>motionRollForward()</code> but non-blocking.
<code>motionStand()</code>	Stand the robot up on its end.
<code>motionStandNB()</code>	Identical to <code>motionStandNB()</code> but non-blocking.
<code>motionTurnLeft()</code>	Rotate the robot counterclockwise.
<code>motionTurnLeftNB()</code> ..	Identical to <code>motionTurnLeft()</code> but non-blocking.
<code>motionTurnRight()</code> ...	Rotate the robot clockwise.
<code>motionTurnRightNB()</code> .	Identical to <code>motionTurnRight()</code> but non-blocking.

CMobot::connect()

Synopsis

```
#include <mobot.h>
int CMobot::connect();
```

Purpose

Connect to a remote robot via Bluetooth.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function is used to connect to a robot. The function looks inside of a Barobo configuration file and connects to the first robot listed in the file. The configuration file may be created and/or modified using the Robot Controller Interface, and selecting the “Robot → Configure Robot Bluetooth” menu item.

Example

Please see the example in Section 4.1.2 on page 12.

See Also

connectWithAddress(), disconnect()

CMobot::connectWithAddress()

Synopsis

```
#include <mobot.h>
int CMobot::connectWithAddress(char address[], int channel);
```

Purpose

Connect to a remote robot via Bluetooth by specifying the specific Bluetooth address of the device.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

address The Bluetooth address of the robot.

channel The Bluetooth channel that the listening program is listening on. The default channel is channel 1.

Description

This function is used to connect to a robot.

Example

See Also

connect(), disconnect()

CMobot::disconnect()

Synopsis

```
#include <mobot.h>
int CMobot::disconnect();
```

Purpose

Disconnect from a remote robot.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function is used from disconnect to a robot. A call to this function is not necessary before the termination of a program. It is only necessary if another connection will be established within the same program at a later time.

Example

See Also

connect(), connectWithAddress()

CMobot::getJointAngle()

Synopsis

```
#include <mobot.h>
int CMobot::getJointAngle(robotJointId_t id, double &position);
```

Purpose

Retrieve a robot joint's current angle.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

- | | |
|-----------------------|---|
| <code>id</code> | The joint number. This is an enumerated type discussed in Section A.1 on page 22. |
| <code>position</code> | A variable to store the current position of the robot motor. The contents of this variable will be overwritten with a value that represents the motor's angle in radians. |

Description

This function gets the current motor position of a robot's motor. The position returned is in units of degrees

and is accurate to roughly ± 0.1 degrees.

Example

See Also

CMobot::getJointMaxSpeed()

Synopsis

```
#include <mobot.h>
int CMobot::getJointMaxSpeed(robotJointId_t id, double &speed);
```

Purpose

Get the maximum speed of a joint on the robot.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

- id** The joint number. This is an enumerated type discussed in Section A.1 on page 22.
- speed** A variable of type **double**. The value of this variable will be overwritten with the maximum speed setting of the joint, which is in units of radians per second.

Description

This function is used to find the maximum speed setting of a joint. This is the maximum speed at which the joint will accept speed setting from the function **setJointSpeed()**. The values are in units of radians per second.

Example

See Also

getJointSpeed(), **getJointMaxSpeedRatio()**, **setJointSpeed()**, **setJointSpeedRatio()**

CMobot::getJointSpeed()

Synopsis

```
#include <mobot.h>
int CMobot::getJointSpeed(robotJointId_t id, double &speed);
```

Purpose

Get the speed of a joint on the robot.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

- id** The joint number to pose. This is an enumerated type discussed in Section A.1 on page 22.
- speed** A variable of type **double**. The value of this variable will be overwritten with the current speed setting of the joint, which is in units of radians per second.

Description

This function is used to find the current speed setting of a joint. This is the speed at which the joint will move when given motion commands. The values are in units of radians per second.

Example**See Also**

getJointMaxSpeed(), getJointSpeedRatio(), setJointSpeed(), setJointSpeedRatio()

CMobot::getJointSpeedRatio()

Synopsis

```
#include <mobot.h>
int CMobot::getJointSpeedRatio(robotJointId_t id, double &ratio);
```

Purpose

Get the speed ratio settings of a joint on the robot.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

- id** Retrieve the speed ratio setting of this joint. This is an enumerated type discussed in Section A.1 on page 22.
- ratio** A variable of type double. The value of this variable will be overwritten with the current speed ratio setting of the joint.

Description

This function is used to find the speed ratio setting of a joint. The speed ratio setting of a joint is the percentage of the maximum joint speed, and the value ranges from 0 to 1. In other words, if the ratio is set to 0.5, the joint will turn at 50% of its maximum angular velocity while moving continuously or moving to a new goal position.

Example**See Also**

setJointSpeeds(), getJointSpeedRatio(), getJointSpeed()

CMobot::getJointSpeedRatios()

Synopsis

```
#include <mobot.h>
int CMobot::getJointSpeedRatios(double ratio[4]);
```

Purpose

Get the speed ratio settings of all joints on the robot.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

speed An array of type double. The four elements of the array will be overwritten with the four speed ratio settings of the robot's joints, which are expressed in as a value from 0 to 1.

Description

This function is used to retrieve all four joint speed ratio settings of a robot simultaneously. The speeds are as a value from 0 to 1.

Example

See Also

setJointSpeeds(), getJointSpeedRatios(), getJointSpeed()

CMobot::getJointSpeeds()

Synopsis

```
#include <mobot.h>
int CMobot::getJointSpeeds(double speed[4]);
```

Purpose

Get the speed settings of all joints on the robot.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

speed An array of type double. The four elements of the array will be overwritten with the four speed settings of the robot's joints, which are expressed in radians per second.

Description

This function is used to retrieve all four joint speed settings of a robot simultaneously. The speeds are in radians per second.

Example

See Also

setJointSpeeds(), getJointSpeedRatios(), getJointSpeed()

CMobot::getJointState()

Synopsis

```
#include <mobot.h>
int CMobot::getJointState(robotJointId_t id, robotJointState_t &state);
```

Purpose

Determine whether a motor is moving or not.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

id The joint number. This is an enumerated type discussed in Section A.1 on page 22.
state An integer variable which will be overwritten with the current state of the motor. This is an enumerated type discussed in Section A.2 on page 22.

Description

This function is used to determine the current state of a motor. Valid states are:

- **ROBOT_JOINT_IDLE** : 0: The motor is idle.
- **ROBOT_JOINT_MOVING** : 1: The motor is moving.
- **ROBOT_JOINT_GOALSEEK** : 2: The motor is heading towards a specified position.

Example

See Also

CMobot::isConnected()

Synopsis

```
#include <mobot.h>
int CMobot::isConnected();
```

Purpose

Check to see if currently connected to a remote robot via Bluetooth.

Return Value

The function returns zero if it is not currently connected to a robot, 1 if the robot is connected, or -1 if an error has occurred.

Parameters

None.

Description

This function is used to check if the software is currently connected to a robot.

Example

See Also

CMobot::isMoving()

Synopsis

```
#include <mobot.h>
int CMobot::isMoving();
```

Purpose

Check to see if a robot is currently moving any of its joints.

Return Value

This function returns 0 if none of the joints are being driven, 1 if any joint is being driven, and -1 if there was an error communicating with the MoBot.

Parameters

None.

Description

This function is used to determine if a robot is currently moving any of its joints.

Example**See Also**

**CMobot::motionInchwormLeft()
CMobot::motionInchwormLeftNB()****Synopsis**

```
#include <mobot.h>
int CMobot::motionInchwormLeft();
int CMobot::motionInchwormLeftNB();
```

Purpose

Perform the inch-worm gait to the left.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function causes the robot to perform a single cycle of the inchworm gait to the left. This function comes in two flavors; a blocking and a non blocking version. The function `motionInchwormLeft()` is blocking, and the function will hang until the motion has finished. The alternative function, `motionInchwormLeftNB()` will return immediately, and the motion will execute asynchronously.

See Also

`motionInchwormRight()`

**CMobot::motionInchwormRight()
CMobot::motionInchwormRightNB()****Synopsis**


```
#include <mobot.h>
int CMobot::motionInchwormRight();
int CMobot::motionInchwormRightNB();
```

Purpose

Perform the inch-worm gait to the right.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function causes the robot to perform a single cycle of the inchworm gait to the right.

This function has both a blocking and non-blocking version. The blocking version, `motionInchwormRight()`, will block until the robot motion has completed. The non-blocking version, `motionInchwormRightNB()`, will return immediately, and the motion will be performed asynchronously.

See Also

`motionInchwormLeft()`

`CMobot::motionRollBackward()` `CMobot::motionRollBackwardNB()`

Synopsis

```
#include <mobot.h>
int CMobot::motionRollBackward();
int CMobot::motionRollBackwardNB();
```

Purpose

Use the faceplates as wheels to roll backward.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function causes each of the faceplates to rotate 90 degrees to roll the robot backward.

This function has both a blocking and non-blocking version. The blocking version, `motionRollBackward()`, will block until the robot motion has completed. The non-blocking version, `motionRollBackwardNB()`, will return immediately, and the motion will be performed asynchronously.

See Also

`motionRollForward()`

CMobot::motionRollForward() CMobot::motionRollForwardNB()

Synopsis

```
#include <mobot.h>
int CMobot::motionRollForward();
int CMobot::motionRollForwardNB();
```

Purpose

Use the faceplates as wheels to roll forward.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function causes each of the faceplates to rotate 90 degrees to roll the robot forward.

This function has both a blocking and non-blocking version. The blocking version, `motionRollForward()`, will block until the robot motion has completed. The non-blocking version, `motionRollForwardNB()`, will return immediately, and the motion will be performed asynchronously.

See Also

`motionRollBackward()`

CMobot::motionStand() CMobot::motionStandNB()

Synopsis

```
#include <mobot.h>
int CMobot::motionStand();
int CMobot::motionStandNB();
```

Purpose

Stand the robot up on a faceplate.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function causes the robot to motionStand up into the camera platform.

This function has both a blocking and non-blocking version. The blocking version, `motionStand()`, will block until the robot motion has completed. The non-blocking version, `motionStandNB()`, will return immediately, and the motion will be performed asynchronously.

See Also

`CMobot::motionTurnLeft()`
`CMobot::motionTurnLeftNB()`

Synopsis

```
#include <mobot.h>
int CMobot::motionTurnLeft();
int CMobot::motionTurnLeftNB();
```

Purpose

Rotate the robot using the faceplates as wheels.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function causes the robot to rotate the faceplates in opposite directions to cause the robot to rotate counter-clockwise.

This function has both a blocking and non-blocking version. The blocking version, `motionTurnLeft()`, will block until the robot motion has completed. The non-blocking version, `motionTurnLeftNB()`, will return immediately, and the motion will be performed asynchronously.

See Also

`motionTurnRight()`

`CMobot::motionTurnRight()`
`CMobot::motionTurnRightNB()`

Synopsis

```
#include <mobot.h>
int CMobot::motionTurnRight();
int CMobot::motionTurnRightNB();
```

Purpose

Rotate the robot using the faceplates as wheels.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function causes the robot to rotate the faceplates in opposite directions to cause the robot to rotate clockwise.

This function has both a blocking and non-blocking version. The blocking version, `motionTurnRight()`, will block until the robot motion has completed. The non-blocking version, `motionTurnRightNB()`, will return immediately, and the motion will be performed asynchronously.

See Also

`motionTurnLeft()`

CMobot::move() CMobot::moveNB()

Synopsis

```
#include <mobot.h>
int CMobot::move(double angle1, double angle2, double angle3, double angle4);
int CMobot::moveNB(double angle1, double angle2, double angle3, double angle4);
```

Purpose

Move all of the joints of a robot by specified angles.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

<code>angle1</code>	The amount to move joint 1, expressed in radians.
<code>angle2</code>	The amount to move joint 2, expressed in radians.
<code>angle3</code>	The amount to move joint 3, expressed in radians.
<code>angle4</code>	The amount to move joint 4, expressed in radians.

Description

This function moves all of the joints of a robot by the specified number of degrees from their current positions.

The function `move()` is a blocking function, which means that the function will not return until the commanded motion is completed. The function `moveNB()` is the non-blocking version of the `move()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more information on blocking and non-blocking functions, please refer to Section 6 on page 17.

Example

Please see the demo at Section 4.1.2 on page 12.

See Also

CMobot::moveContinuousNB()

Synopsis

```
#include <mobot.h>
int CMobot::moveContinuousNB(
    robotJointDirection_t dir1,
    robotJointDirection_t dir2,
    robotJointDirection_t dir3,
    robotJointDirection_t dir4);
```

Purpose

Move the joints of a robot continuously in the specified directions.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

Each integer parameter specifies the direction the joint should move. The types are enumerated in `mobot.h` and have the following values:

- `MOBOT_NEUTRAL` : The joint should not move.
- `MOBOT_FORWARD` : The joint will begin moving in the positive direction.
- `MOBOT_BACKWARD`: The joint will begin moving in the negative direction.

More documentation about these types may be found at Section A.3 on page 22.

Description

This function causes joints of a robot to begin moving at the previously set speed. The joints will continue moving until the joint hits a joint limit, or the joint is stopped by setting the speed to zero. This function is a non-blocking function.

Example

See Also

CMobot::moveContinuousTime()

Synopsis

```
#include <mobot.h>
int CMobot::moveContinuousTime(int dir1, int dir2, int dir3, int dir4, int msec);
```

Purpose

Move the joints of a robot continuously in the specified directions.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

Each integer direction parameter specifies the direction the joint should move. The types are enumerated in `mobot.h` and have the following values:

- `MOBOT_NEUTRAL` : The joint should not move.

- **MOBOT_FORWARD** : The joint will begin moving in the positive direction.
- **MOBOT_BACKWARD**: The joint will begin moving in the negative direction.

The **msecs** parameter is the time to perform the movement, in milliseconds.

Description

This function causes joints of a robot to begin moving. The joints will continue moving until the joint hits a joint limit, or the time specified in the **msecs** parameter is reached. This function will block until the motion is completed.

Example

See Also

CMobot::moveTo()
CMobot::moveToNB()

Synopsis

```
#include <mobot.h>
int CMobot::moveTo(double angle1, double angle2, double angle3, double angle4);
int CMobot::moveToNB(double angle1, double angle2, double angle3, double angle4);
```

Purpose

Move all of the joints of a robot to the specified positions.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function moves all of the joints of a robot to the specified absolute positions.

The function **moveTo()** is a blocking function, which means that the function will not return until the commanded motion is completed. The function **moveToNB()** is the non-blocking version of the **moveTo()** function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 6 on page 17.

Example

Please see the demo at Section 4.1.2 on page 12.

See Also

CMobot::moveJointTo()
CMobot::moveJointToNB()

Synopsis

```
#include <mobot.h>
int CMobot::moveJointTo(int id, double position);
int CMobot::moveJointToNB(int id, double position);
```

Purpose

Move a joint on the robot to an absolute position.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

`id` The joint number to wait for.
`position` The absolute angle to move the motor to.

Description

This function commands the motor to move to a position specified in degrees at the current motor's speed. The current motor speed may be set with the `setJointSpeed()` member function. Please note that if the motor speed is set to zero, the motor will not move after calling the `moveJointTo()` function.

The function `moveJointTo()` is a blocking function, which means that the function will not return until the commanded motion is completed. The function `moveJointToNB()` is the non-blocking version of the `moveJointTo()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 6 on page 17.

Example

Please see the example in Section 4.1.2 on page 12.

See Also

`connectWithAddress()`

CMobot::moveJointWait()

Synopsis

```
#include <mobot.h>
int CMobot::moveJointWait(int id);
```

Purpose

Wait for a joint to stop moving.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

`id` The joint number to wait for.

Description

This function is used to wait for a joint motion to finish. Functions such as `poseJoint()` and `moveJoint()`

do not wait for a joint to finish moving before continuing to allow multiple joints to move at the same time. The `moveJointWait()` or `moveJointWait()` functions are used to wait for robotic motions to complete.

Please note that if this function is called after a motor has been commanded to turn indefinitely, this function may never return and your program may hang.

Example

Please see the example in Section 4.1.2 on page 12.

See Also

`moveWait()`

CMobot::moveWait()

Synopsis

```
#include <mobot.h>
int CMobot::moveWait();
```

Purpose

Wait for all joints to stop moving.

Return Value

The function returns 0 on success and non-zero otherwise.

Description

This function is used to wait for all joint motions to finish. Functions such as `poseJoint()` and `moveJoint()` do not wait for a joint to finish moving before continuing to allow multiple joints to move at the same time. The `moveWait()` or `moveJointWait()` functions are used to wait for robotic motions to complete.

Please note that if this function is called after a motor has been commanded to turn indefinitely, this function may never return and your program may hang.

Example

See the sample program in Section 4.1.2 on page 12.

See Also

`moveWait()`, `moveJointWait()`

CMobot::moveToZero() CMobot::moveToZeroNB()

Synopsis

```
#include <mobot.h>
int CMobot::moveToZero();
int CMobot::moveToZeroNB();
```

Purpose

Move all of the joints of a robot to their zero position.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function moves all of the joints of a robot to their zero position. Please note that this function is non-blocking and will return immediately. Use this function in conjunction with the `moveWait()` function to block until the movement completes.

The function `moveToZero()` is a blocking function, which means that the function will not return until the commanded motion is completed. The function `moveToZeroNB()` is the non-blocking version of the `moveToZero()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 6 on page 17.

Example

Please see the demo at Section 4.1.2 on page 12.

See Also

CMobot::setJointSpeed()

Synopsis

```
#include <mobot.h>
int CMobot::setJointSpeed(int id, double speed);
```

Purpose

Get the speed of a joint on the robot.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

`id` The joint number to pose.
`speed` An variable of type `double` indicating the requested speed.

Description

This function is used to set the speed of a joint of a robot. Valid speed values range from 0 to 1.

Example**See Also**

CMobot::setJointSpeedRatio()

Synopsis

```
#include <mobot.h>
int CMobot::setJointSpeedRatio(robotJointId_t id, double ratio);
```

Purpose

Set the speed ratio settings of a joint on the robot.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

- id** Set the speed ratio setting of this joint. This is an enumerated type discussed in Section A.1 on page 22.
- ratio** A variable of type double with a value from 0 to 1.

Description

This function is used to set the speed ratio setting of a joint. The speed ratio setting of a joint is the percentage of the maximum joint speed, and the value ranges from 0 to 1. In other words, if the ratio is set to 0.5, the joint will turn at 50% of its maximum angular velocity while moving continuously or moving to a new goal position.

Example**See Also**

setJointSpeeds(), setJointSpeedRatio(), getJointSpeed()

CMobot::setJointSpeedRatios()

Synopsis

```
#include <mobot.h>
int CMobot::setJointSpeedRatios(double ratios[4]);
```

Purpose

Set the speed ratio settings of all joints on the robot.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

- ratios** An array of type double. Each element of the array represents the speed ratio for a joint, expressed in a value from 0 to 1.

Description

This function is used to simultaneously set the angular speed ratio settings of all four joints of a robot. The speed ratio is a percentage of the maximum speed of a joint, expressed in a value from 0 to 1.

Example**See Also**

getJointSpeeds(), setJointSpeed(), getJointSpeed()

CMobot::setJointSpeeds()

Synopsis

```
#include <mobot.h>
int CMobot::setJointSpeeds(double speeds[4]);
```

Purpose

Set the speed settings of all joints on the robot.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

speeds An array of type double. Each element of the array represents the speed, expressed in radians per second, to set a joint.

Description

This function is used to simultaneously set the angular speed settings of all four joints of a robot.

Example

See Also

getJointSpeeds(), setJointSpeed(), getJointSpeed()

CMobot::setTwoWheelRobotSpeed()

Synopsis

```
#include <mobot.h>
int CMobot::setTwoWheelRobotSpeed(double speed, double radius, char unit[]);
```

Purpose

Roll the robot at a certain speed in a straight line.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

- speed** The speed at which to roll the robot. The units used will be the units specified in the **unit** parameter.
- radius** The radius of the wheels attached to the robot. The units of the parameter should match the units provided in the **unit** parameter.
- unit** Specify the units used for the **speed** and **radius** parameters.
- "cm" : The speed is expressed in centimeters per second, and the radius in centimeters.
 - "m" : The speed is expressed in meters per second, and the radius in meters.
 - "inch" : The speed is expressed in inches per second, and the radius in inches.
 - "foot" : The speed is expressed in feet per second, and the radius in feet.

Description

This function is used to make a two wheeled robot roll at a certain speed. The desired speed and radius of the wheels is provided and the function will rotate the wheels at the appropriate rate in order to achieve the desired speed.

Example**See Also**

CMobot::stop()**Synopsis**

```
#include <mobot.h>
int CMobot::stop();
```

Purpose

Stop all current motions on the robot.

Return Value

The function returns 0 on success and non-zero otherwise.

Description

This function stops all currently occuring movements on the robot. Internally, this function simply sets all motor speeds to zero. If it is only required to stop a single motor, use the `setJointSpeed()` function to set the motor's speed to zero.

Example**See Also**

`setJointSpeed()`

Index

connect(), 26
connectWithAddress(), 26

disconnect(), 27

getJointAngle(), 27
getJointMaxSpeed(), 28
getJointSpeed(), 28
getJointSpeedRatio(), 29
getJointSpeedRatios(), 29
getJointSpeeds(), 30
getJointState(), 30

isConnected(), 31
isMoving(), 31

motionInchwormLeft(), 32
motionInchwormRight(), 32
motionInchwormRightNB(), 32
motionRollBackward(), 33
motionRollBackwardNB(), 33
motionRollForward(), 34
motionRollForwardNB(), 34
motionStand(), 34
motionStandNB(), 34
motionTurnLeft(), 35
motionTurnLeftNB(), 35
motionTurnRight(), 35
motionTurnRightNB(), 35
move(), 36
moveContinuousNB(), 36
moveContinuousTime(), 37
moveJointTo(), 38
moveJointToNB(), 38
moveJointWait(), 39
moveNB(), 36
moveTo(), 38
moveToNB(), 38
moveToZero(), 40
moveToZeroNB(), 40
moveWait(), 40

ROBOT_BACKWARD, 23
ROBOT_FORWARD, 23
ROBOT_JOINT1, 22
ROBOT_JOINT2, 22
ROBOT_JOINT3, 22
ROBOT_JOINT4, 22
robot_joint_direction_t, 23
robot_joints_t, 22
ROBOT_NEUTRAL, 23

setJointSpeed(), 41
setJointSpeedRatio(), 41
setJointSpeedRatios(), 42
setJointSpeeds(), 43
setTwoWheelRobotSpeed(), 43
stop(), 44