sparkilearningDocumentation

Release latest

Jun 16, 2022

Contents

The Sparki robot homepage at Arcbotics may be found here: http://arcbotics.com/products/sparki/

The sparki_learning library github page (with installation instructions) may be found here: https://github.com/radarjd/sparki_learning

This command reference is for version 1.6.8 of the python library

(this library makes use of Python 3; if you're using Python 2, stop!)

Contents

CHAPTER 1

Sparki Commands

1.1 General Commands

1.1.1

The number sign (also called the pound sign or the hash sign) indicates the beginning of a comment. Python ignores everything on a line after #. Comments are used to explain what your code is doing and can be very helpful when someone else is reading your code (or you are re-reading it after not having worked on it in some time).

1.1.2 constrain(n, min_n, max_n)

Returns a value that is greater than or equal to min_n and less than or equal to max_n (i.e. it does bounds checking). This function is used when you have a value (n) that you want to be sure is no less than min_n and no more than max_n. If n is between min_n and max_n, it returns n. If n is less than min_n, it returns min_n. If n is greater than max_n, it returns max_n. Used heavily within the library – you may or may not find it useful. (Moved to sparki_learning.util)

1.1.3 currentTime()

Returns the number of seconds which have occurred since midnight on January 1, 1970. Other functions exist to format this into a more manageable number (like humanTime() below), but this can be used (among other uses) to determine how long a part of a program has taken. For example, to time how long a portion of code takes to execute (in clock time), before the function you could write startingTime = currentTime() and then after the function you could find out the running time with runTime = currentTime() - startingTime. (Moved to sparki_learning.util)

1.1.4 flrange(start, stop, step)

Returns an iterator, very similar to range() in Python 3 or xrange() in Python 2 (see *range* below). None of the arguments have default values. start should be the first value of x you want to calculate, stop should be the last - step value you want to calculate, and step should be the step. If step is negative, this will count down from start to stop (so stop must be less than start). The start, stop, and step values are very similar to the arguments to the range command, but range() allows only integer arguments. (Moved to sparki_learning.util)

1.1.5 from sparki_learning import *

This command imports the sparki_learning module to make all of the Sparki specific commands available. It must appear at the beginning of your Sparki programs. Other modules (also called libraries) exist in python such as string, math, random, sys, os and datetime.

1.1.6 getCommandQueue()

Returns a tuple containing the commands (with arguments) that have been sent to the Sparki.

1.1.7 getUptime()

Returns the number of milliseconds since the Sparki was initialized; returns -1 if Sparki has not been initialized.

1.1.8 getVersion()

Returns a tuple containing the version of the Python sparki_learning library and the version of the software running on the Sparki in that order (i.e. python library version is [0] and sparki software is [1]).

1.1.9 humanTime()

Returns the time in a human readable format like "Fri Apr 5 19:50:05 2016". (Moved to sparki_learning.util)

1.1.10 init(com_port, print_versions=True, auto=False, retries=2)

Connects your computer to the Sparki via Bluetooth. On Windows, com_port will be something like "COM5" or "COM40". On a Mac, instead of using a COM port, you will use a device path which looks something like "/dev/tty.ArcBotics-DevB". You must have paired your computer with Sparki on Bluetooth prior to executing this command. Your computer will assign the COM port or device. On a Mac, you can also use the secret port "mac" and the library will fill in the standard Mac port. This function has become increasingly complicated in order to make it easier to initialize the robot without errors. The print_versions argument (optional, default is True) will print a message upon initialization that tells you the Sparki and python library versions. The auto argument (optional, default is False) will suppress serial errors, and is intended to be used with the initAuto()_ command below. The retries argument (optional, default is 2) specifies the number of times to try to initialize the robot on the given port. The most common reason that initialization appears to fail even though the port is correct appears to have to do with power saving. A modern OS will deactivate the Bluetooth when it's not in use to save power. Giving a couple of tries seems to turn it back on.

1.1.11 initAuto()

Connects your computer to the Sparki via Bluetooth by guessing at the port. On Mac, this will try the most common addresses, but if you custom named the port, this will not be successful. On Windows, this will try COM3 through COM10. It will (by default) print the port that it found the connection on, so that you can put that into *init*

1.1.12 noop()

This does nothing. It happens to be that doing nothing can be helpful on a laptop – laptops have a tendency to drop the bluetooth connection to the robot very quickly, probably for power saving purposes. This command will send a message to the sparki in an attempt to prevent the connection from being dropped.

1.1.13 randint(start, stop)

Returns an integer between (and including) the start and stop bounds. To use this function, you must import it from the random library (e.g. from random import randint). The random library has a variety of other functions to generate (semi-)random numbers. Full documentation may be found at https://docs.python.org/3/library/random.html

1.1.14 range(start, stop, step)

Returns a list of integers from start to stop by step. start is the number at which the range begins. The range ends one value before stop. The range counts by step. For example, if step is 2, range counts by twos. start and step are optional. start defaults to 0 and step defaults to 1. This command is built in to Python, with full documentation at https://docs.python.org/3/library/stdtypes.html#typesseq-range . In Python 2, you should use the xrange() function instead with full documentation at https://docs.python.org/2.7/library/functions.html#xrange

1.1.15 setDebug(level)

Sets the level of debug output for the python library. Possible values (from least verbose to most verbose) are DEBUG_ALWAYS, DEBUG_CRITICAL, DEBUG_ERROR, DEBUG_WARN, DEBUG_INFO, DEBUG_DEBUG. The DEBUG levels are constant integer values defined in the sparki_learning library. The default is DEBUG_WARN, which is a fairly sane level of verbosity. DEBUG_INFO will give a message each time a function is entered. DEBUG_DEBUG will output all messages to and from the robot as well.

1.1.16 setSparkiDebug(level)

Sets the level of debug output for the Sparki itself library. The SPARKI_DEBUGS capability must be set to True (and it is False for all "standard" versions of the Sparki library to save memory on the Sparki). Possible values (from least verbose to most verbose) are DEBUG_ALWAYS, DEBUG_CRITICAL, DEBUG_ERROR, DEBUG_WARN, DEBUG_INFO, DEBUG_DEBUG. The DEBUG levels are constant integer values defined in the sparki_learning library. The default is DEBUG_WARN, which is a fairly sane level of verbosity. Messages will be displayed on the Sparki's LCD. You probably never will use this function.

1.1.17 timer(duration)

Generator which returns (yields) the amount of time which has passed since it was first called. Ends when the time of the original call plus the duration is greater than the current time. In practice, this function is usually used to create a for loop which executes for duration. (e.g. for x in timer(120): [insert code you want to run for 120 seconds]). (Moved to sparki_learning.util)

1.1.18 wait(time)

Waits time seconds before moving to the next command.

1.1.19 waitNoop(time)

Waits time seconds before moving to the next command, and sends noops to the robot every second in order to prevent a timeout. On a Mac, this may be a better command to use than wait, but it will also be less accurate as to the amount of time waited.

1.2 Common Variables

Several commands in the myro module use variables for speed and time, including motors(), forward(), backward(), turnLeft(), and turnRight(). wait() makes use only of time. message is used for output and some input functions. file is used for reading data from files and saving data to them.

distance - a float number of centimeters.

file - a string value containing an absolute path to a file, or a relative path from the directory where IDLE resides. An absolute path on Windows looks like "C:\Music\musicfile.wav" and on Mac looks like "/home/music/musicfile.wav". Even on Windows, the backslashes may appear as front slashes.

message - a value to be output such as a literal string like "Hello, World", a variable, or a combination. If you want message to be a literal string, remember to enclose the string in quotation marks. If you are combining a string plus a numeric value, you need to convert the numeric value to a string using the str() function. For example, if count were a variable which holds the iteration number of a loop, message could be "I am on iteration number " + str(count). In the two special cases of *print(message)* and *speak(message, alsoprint=False)*, message may actually be multiple arguments instead of a string. That is, in the case of those two functions only, you can do something like print("Hello", 2, "you") or speak("Hello", 2, "you").

speed - a value between -1 and 1. Any value less than -1 will be made -1; any value more than 1 will be made 1. A value of 1 means to turn the wheels at full power. A value of -1 means to turn the wheels in the opposite direction at full power. Decimal values mean to use proportionately less than full power (so .5 is half power).

time - the time in seconds to perform the action. Fractional values will perform for fractional seconds. A value of -1 means to perform the action forever. If time is optional for a command, the value defaults to -1 (meaning that if you omit time, the robot performs the command until you tell it to *stop()*).

1.3 Movement Commands

In addition to the below, the *Grid Commands* also move the robot.

1.3.1 backward(speed, time = -1)

Move the robot backward at *speed* speed for *time* seconds. time is optional and may be omitted. See the section on *Common Variables* for an explanation of speed and time.

1.3.2 forward(speed, time = -1)

Move the robot forward at *speed* speed for *time* seconds. time is optional and may be omitted. See the section on *Common Variables* for an explanation of speed and time.

1.3.3 gripperClose(distance = 7)

Closes the gripper on the robot *distance* centimeters. distance is optional and may be omitted – if omitted, the gripper will be totally closed.

1.3.4 gripperOpen(distance = 7)

Opens the gripper on the robot *distance* centimeters. distance is optional and may be omitted – if omitted, the gripper will be totally opened.

1.3.5 gripperStop()

Stops the gripper motor.

1.3.6 isMoving()

Returns True if the Python library thinks the Sparki's wheel motors are turning. Note that this is a guess and errors or connection problems could throw this off.

1.3.7 motors(left_speed, right_speed, time = -1)

Starts the robot's wheel motors. The left wheel will move at left_speed. The right wheel will move at right_speed. The wheels will move for *time* seconds. time is optional and may be omitted. left_speed and right_speed are *speed* variables, and time is a *time* variable, as defined in the section on *Common Variables*.

1.3.8 moveBackwardcm(distance)

Moves the robot backward *distance* centimeters.

1.3.9 moveForwardcm(distance)

Moves the robot backward *distance* centimeters.

1.3.10 servo(position) (defined below) will turn the Sparki's head

1.3.11 stop()

Stops the robot's motors immediately. Only necessary if the robot is presently in motion (e.g. for the motors() or move() commands).

1.3.12 turnLeft(speed, time = -1)

Turn the robot left at *speed* speed for *time* seconds. time is optional and may be omitted. See the section on *Common Variables* for an explanation of speed and time.

1.3.13 turnRight(speed, time = -1)

Turn the robot right at *speed* speed for *time* seconds. time is optional and may be omitted. See the section on *Common Variables* for an explanation of speed and time.

1.4 Grid Commands

The grid commands implement a pseudo-coordinate plane for use with the Sparki. When you turn the robot on, the robot is assumed to be a 0,0 and facing the positive direction on the y axis. You can use the moveTo() commands to move to a specific point on the grid. Each integer position on the grid is 1cm from the next or previous integer position. For example, 0,1 would be 1cm forward from the starting position of the robot. The robot only updates its grid position when using the grid commands. For example, if you used forward(1,1), that would not update the grid position. If you want to ensure the robot stays somewhere on the grid, only use grid movement commands. If you don't care about the grid position, mixing commands is perfectly fine!

1.4.1 drawFunction(function, xvals, scale = 1)

This is a complicated function. drawFunction() draws the function given by the function argument on the coordinate plane. The function argument should be a lambda function. The lambda function given should return the value of the y coordinate given the x. For example, lambda x: x**2 given as the function would graph y=x2. xvals should be an iterator of the values of x you want to use. You may find the flrange() function helpful. For example, drawFunction(lambda x: math.sin(x), flrange(-2, 2.1, .1)) would draw the sin x from -2 to 2 going a tenth at a time. scale increases the size of the drawing for visibility.

1.4.2 getAngle()

Return the number of degrees that Sparki has turned using the turnBy() command, or since setAngle() was last called. turnRight(), turnLeft() and motors() do not update the angle. Increases on positive angle turns and decreases on negative angle turns.

1.4.3 getCentimetersMoved()

Returns the float number of centimeters Sparki has moved using moveForwardcm(), moveBackwardcm(), and the grid commands. Always increases.

1.4.4 getPosition()

Return Sparki's current position on the grid as a list. The x position is the first element of the list (i.e. [0]) and the y position is the second element of the list (i.e. [1]).

1.4.5 moveBy(x, y, turnBack = False)

Move the robot to grid position x, y as though the current position were 0,0. For example, if the robot is at 3,4, moveBy(1,1) would move Sparki to 4,5. If turnBack is True (not the default), the robot will turn back to the heading it was on prior to the command.

1.4.6 moveTo(x, y, turnBack = False)

Move the robot to grid position x, y. If turnBack is True (not the default), the robot will turn back to the heading it was on prior to the command.

1.4.7 resetPosition()

Sets Sparki's current position on the grid to 0,0 and its angle to 0. The same as calling setAngle(0) and setPosition(0,0). Does not move the robot.

1.4.8 setAngle(newAngle)

Sets the number of degrees that Sparki has turned, which is used by moveBy() and moveTo(). When Sparki is initialized, the angle is 0, so to reset the angle as though the robot were just turned on, use setAngle(0). newAngle defaults to 0. turnRight(), turnLeft() and motors() do not update the angle. Does not move Sparki - if you want to turn to an angle relative to the robot's starting position, use turnTo(); if you want to turn to an angle relative to the robot's current position, use turnBy()

1.4.9 setPosition(x, y)

Sets Sparki's current position on the grid. Does not move the robot.

1.4.10 turnBy(degrees)

Turns the robot by a number of degrees. degrees can be any number. A negative degrees turns the robot left (counterclockwise). A positive degrees turns the robot right (clockwise). Note that this behavior is different than the Myro library - for the Myro library, a negative value turns right (clockwise).

1.4.11 turnTo(degrees)

Turns the robot to the specified heading relative to the value returned by getAngle(). degrees can be any number greater than or equal to 0 and less than 360. A value greater than or equal to 360 or less than 0 will be wrapped around. When the robot is initialized, that heading is defined as 0.

1.5 Input Commands

Several of the commands (ask(), askQuestion(), joystick(), pickAFile() and yesorno()) will create GUI windows if tkinter is available. If it is not, the library will fall back to pure text.

1.5.1 ask(message, title = "Question")

Creates window with message. User is allowed to input a response. Returns the user's response. See the section on Common Variables for an explanation of message. title is optional and defaults to "Question". (Note that if this appears to do nothing, the window with the output may be hidden behind other windows.) (Moved to sparki_learning.gui in v1.6.0.)

1.5.2 askQuestion(message, options, title = "Question")

Creates window with message, with buttons labeled with options. options must be a list of strings. Returns the user's response. See the section on Common Variables for an explanation of message. Loops until user gives a value in options. title is optional and defaults to "Question". (Note that if this appears to do nothing, the window with the output may be hidden behind other windows.) (Moved to sparki_learning.gui in v1.6.0.)

1.5.3 gamepad()

Control Sparki using the remote control. The Sparki will not accept commands while it is under remote control. Press + or - on the remote to end remote control.

1.5.4 input(message)

Prints message to the shell. User is allowed to input a response. Returns the user's response. See the section on Common Variables for an explanation of message. This command is built in to Python. The full documentation is available at python.org at https://docs.python.org/3/library/functions.html#input

1.5.5 joystick()

Control Sparki using a GUI window. Allows movement and opening and closing of the gripper. (Note that if this appears to do nothing, the window with the output may be hidden behind other windows.)

1.5.6 messageWindow(message, title = "Message")

Strictly speaking, this is not an input command, but it does display a message to the user and "pause" your program until the user clicks okay.

1.5.7 pickAFile(prompt = "Choose a file")

Creates window with a file dialog so that the user can pick a file. Might be useful for reading from or saving to a file. (Moved to sparki_learning.gui in v1.6.0.). prompt is an optional message prompt to the user (added in 1.6.3)

1.5.8 pickAFolder(prompt = "Choose a folder")

Creates window with a dialog so that the user can pick a folder. prompt is an optional message prompt to the user.

1.5.9 yesorno(message)

Creates window with message, with buttons labeled with "yes" and "no". Returns the user's response (which will be either "yes" or "no"). See the section on Common Variables for an explanation of message. (Moved to sparki_learning.gui in v1.6.0.)

1.6 Output Commands

1.6.1 beep(time = 200, freq = 2800)

Plays a tone at freq for time milliseconds through the Sparki. Both are optional (freq defaults to 2800 and time to 200). A list of common frequencies can be found at http://en.wikipedia.org/wiki/Piano_key_frequencies.

1.6.2 LCDclear(update = True)

Clears (makes blank) Sparki's LCD. Note that you must call LCDupdate() to display the message once it's printed if you set update to False.

1.6.3 LCDdrawLine(X1, Y1, X2, Y2, update = True)

Draws a line from X1, Y1 to X2, Y2 on the LCD. The X coordinates can be from 0 to 127. The Y coordinates can be from 0 to 63. Note that you must call LCDupdate() to display the line once it's drawn if you set update to False.

1.6.4 LCDdrawPixel(X,Y, update = True)

Draws the pixel at X, Y on the LCD. The X coordinate can be from 0 to 127. The Y coordinate can be from 0 to 63. Note that you must call LCDupdate() to display the dot once it's drawn if you set update to False.

1.6.5 LCDdrawRect(X1, Y1, X2, Y2, update = True)

Draws a rectangle having opposite corners at X1, Y1 and X2, Y2 on the LCD. The X coordinates can be from 0 to 127. The Y coordinates can be from 0 to 63. Note that you must call LCDupdate() to display the line once it's drawn if you set update to False.

1.6.6 LCDdrawString(X, Y, message, update = True)

Prints message to the LCD on the back of Sparki at the X, Y coordinate given. X is the pixel to begin the drawing – can be from 0 to 121. Y is the line to begin the drawing – can be from 0 to 7. Note that you must call LCDupdate() to display the message once it's printed if you set update to False.

1.6.7 LCDerasePixel(X,Y, update = True)

Erases the pixel at X, Y on the LCD. The X coordinate can be from 0 to 127. The Y coordinate can be from 0 to 63. Note that you must call LCDupdate() to display the dot once it's drawn if you set update to False.

1.6.8 LCDprint(message, update = True)

Prints message to the LCD on the back of Sparki. See the section on Common Variables for an explanation of message. Note that you must call LCDupdate() to display the message once it's printed if you set update to False.

1.6.9 LCDprintLn(message, update = True)

Prints message to the LCD on the back of Sparki, and go to the next line. See the section on Common Variables for an explanation of message. Note that you must call LCDupdate() to display the message once it's printed if you set update to False.

1.6.10 LCDsetColor(color)

Sets the drawing color of the LCD command. Can be used to erase previously drawn things. color of 0 is black, and that's how it starts. color of 1 is white, and would erase things that are black.

1.6.11 LCDreadPixel(X,Y)

Returns True if the color of the pixel at X, Y on the LCD is black; otherwise returns False. The X coordinate can be from 0 to 127. The Y coordinate can be from 0 to 63.

1.6.12 LCDupdate()

Updates the Sparki's LCD with anything new you've printed to it since the last update.

1.6.13 print(message)

Prints message to the computer screen. See the section on Common Variables for an explanation of message. print() is built in to Python and has other useful options, but they go beyond what you're likely to encounter routinely. The full documentation is available at python.org at https://docs.python.org/3/library/functions.html#print

1.6.14 printDebug(message, priority = DEBUG_WARN, output = sys.stderr)

Prints message to output if the current debug level (set by *setDebug(level)* and defaulting to DEBUG_WARN) is greater than or equal to priority. output defaults to standard error. Included for convenience of writing your own functions. message must be a string (i.e. message does not accept multiple arguments like speak or print). (Moved to sparki_learning.util in 1.5.2.dev2)

1.6.15 setRGBLED(red, green, blue)

Sets the RGB LED to red, green, blue, where each value is a number from 0 to 100. For example, [100, 0, 0] would turn the light fully red, [0, 100, 0] would be fully green, and [0, 0, 100] would be fully blue. The values can be mixed to make most colors. Hardware limitations prevent this function from working to its full capability. The LED simply cannot display all values, and in particular cannot display values where red is equal to the other values. If you want to display "white", Arcbotics recommends values of 60,100,90 – the library contains suggested values for other colors in the setRGBLED function code.

1.6.16 setStatusLED(brightness)

Sets the status LED to brightness, where brightness is a number from 0 to 100. For the "standard" versions of the Sparki software, the status LED is illuminated when the robot is processing a command, and turned off when it is not, so this is of limited use.

1.6.17 speak(message, alsoprint=False)

Speaks the given message. Can be given an argument like print – e.g. speak("Hello", 2, "you") should work as well as speak("Hello to you"). You can also give the keyword argument alsoprint=True if you want the speak commend to print out the message immediately prior to speaking. So, you could use speak("Hello", 2, "you", alsoprint=True) as well as speak("Hello to you", alsoprint=True) *Relies on operating system features, not python – only implemented for Mac and Windows.* On Mac, this uses the say command. On Windows, this command may create a window which is open briefly during the actual speech and requires write access to the user's home folder. The speak function is implemented with a hack and may be unreliable.

1.7 Sensor Commands

1.7.1 compass()

Returns Sparki's current compass heading. *Inaccurate*

1.7.2 getAccel()

Returns the current values of Sparki's 3 accelerometers in a list. Accelerometers measure acceleration (primarily due to gravity) and can tell you the orientation of Sparki. The library includes convenience functions getAccelX(), getAccelY(), and getAccelZ() if you only want one sensor value.

1.7.3 getBattery()

depricated due to inaccuracies in 1.5.1 – always returns -1. Returns the current voltage of Sparki's batteries. ArcBotics reports that the underlying system call is unreliable, and as such any value returned by this function is suspect (a value < 2 should always be disregarded). If it is working, anything below around 4 is low. Should be just below 6 with new batteries.

1.7.4 getBright(position)

Returns the brightness value in front of the robot from position. A higher number indicates more light. position is the literal string “left”, “center”, or “right” (or the number 0 for “left”, 1 for “center”, or 2 for “right”). position may be omitted, in which case this returns a list of three values representing the “left”, “center”, and “right” sensors.

1.7.5 getLine(position)

Returns the value of the line sensor in position. The sensor is actually testing the reflection of an infrared light, and is only accurate with a high amount of contrast (e.g. a black line on a white surface). position may be any number 0 through 4: 0 is the line sensor on the left edge, 1 is the left middle, 2 is the middle, 3 is the right middle, and 4 is the left edge. position may be omitted, in which case getLine() returns a list of five values representing all line sensors.

1.7.6 getMag()

Returns the current values of Sparki’s 3 magnetometers in a list. Magnetometers measure magnetic fields around the robot. The library includes convenience functions getMagX(), getMagY(), and getMagZ() if you only want one sensor value.

1.7.7 getObstacle(position)

Returns a number which is the number of centimeters the Sparki believes the closest object is. position is the literal string “left”, “center”, or “right” (or the number of degrees you want to turn the servo – see the servo command for more information). position may be omitted, in which case getObstacle() returns a list of three values representing the “left”, “center”, and “right” values.

1.7.8 ping()

Returns the number of centimeters to the closest object directly in front of Sparki’s head. You can turn the head with servo(position) and then get a distance with ping().

1.7.9 senses()

Displays a window with (or prints out) data from all of the sensors on Sparki. By default, it updates every two seconds. Program execution is paused while the window is displayed. If tkinter is not available, no window will be displayed, but the status of the sensors will be output in text. (Note that if this appears to do nothing, the window with the output may be hidden behind other windows.)

1.7.10 servo(position)

Turns the servo (sparki’s head) to position. position is a number between -90 and 90, where -90 is directly to the left, 0 is straight ahead, and 90 is directly to the right.

1.8 IR Commands

1.8.1 receiveIR()

Returns an int received from the IR sensor on the front of the Sparki. You can then convert this int to a more meaningful piece of information. Returns a -1 if no data is available *Not well tested*

1.8.2 sendIR(data)

Sends the int data via the IR emitter on the front of the Sparki. *Not well tested*

1.9 EEPROM Commands

The EEPROM is non-volatile (long-term) storage located on the processor on Sparki (the processor is the chip labeled ATMEGA32U4). The ATMEGA32U4 has 1024 bytes of EEPROM. We use this primarily to store the name of the Sparki (at byte 20), but you can use it to store other stuff. These commands allow you to read and write from the EEPROM.

1.9.1 bluetoothRead()

Returns the bluetooth address of the Sparki *if* it has been previously stored on the robot. The address must be written to byte 80 in the EEPROM using either *EEPROMwrite(location, data)* or preferably *bluetoothWrite(address)*.

1.9.2 bluetoothWrite(address)

Writes the bluetooth address to the EEPROM. address must be a properly formatted bluetooth address (i.e. it must be six pairs of hexidecimal numbers separated by either - or :). The method of determining your address varies on your operating system.

1.9.3 EEPROMread(location, amount)

Reads amount bytes of data at location in the EEPROM. location must be greater than or equal to 0 and less than 1024. Note that it is likely that there's nothing interesting at a particular location unless you've put it there.

1.9.4 EEPROMwrite(location, data)

Writes data to location in the EEPROM. location must be greater than or equal to 0 and less than 1024. The length of data plus the location must be less than 1024 (since the EEPROM stops at 1024). Be careful with writing data to the EEPROM, as you don't want to overwrite important things. As a general guideline, keep your writing location greater than 100.

1.9.5 getName()

Returns the name of the physical Sparki robot. Note that you must have set the name on each physical robot at least once prior to getting the name, or you'll get a garbage value.

1.9.6 setName(name)

Sets the name of the physical Sparki robot. Note that you must have set the name on each physical robot at least once prior to getting the name, or you'll get a garbage value.

CHAPTER 2

Related Commands

2.1 Synchronization Commands

The synchronization commands are provided to allow several computers to do something at the same time. In the case of sparki_learning, they are usually used to allow multiple computers to command their respective robots so that the robots can do things together (at the same time). One computer acts as the server, and all other computers act as the clients. The server is told how much time to wait, and then the server communicates that to any clients that connect to the server. When the time expires, the server and clients all return from the synchronization function. These are found in sparki_learning.sync_lib

2.1.1 get_client_start(server_ip, server_port = 32216)

Connects to a sync server returns the amount of time before the synchronization event should happen. server_ip should be the server's ip address (which is printed to the server's screen when the server calls start_sync_server()). server_port defaults to 32216 (the same as the server). server_port must be the same on the client and server.

2.1.2 start_sync_server(time = 15, server_port = 32216)

Starts the sync server to execute the commands following after time seconds. Opens a network socket on the computer, so you may be asked if you want Python to be allowed to open and listen on a port. server_port defaults to port 32216. server_port must be the same on the client and server.

2.1.3 start_sync_client(server_ip, server_port = 32216)

Connects to a sync server and executes the commands following after the period of time specified by the server. server_ip should be the server's ip address (which is printed to the server's screen when the server calls start_sync_server()). server_port defaults to 32216 (the same as the server). server_port must be the same on the client and server.

2.1.4 syncWait(server_ip, server_port = 32216)

Wait for a time specified by a sync server over a network. Uses the *get_client_start(server_ip, server_port = 32216)* function above. One computer must be designated the sync server and execute the start_sync_server command.