inSphero Chip Tilter - Programming Tutorial

Downloads

The necessary files can be downloaded from GitHub: Click

Introduction

This Python script allows the user to set inSphero's chip tilter parameters, e.g. tilting angles, waiting times, through the USB port of your computer.

It is possible to integrate the script into existing projects and to benefit from synchronization opportunities with other connected devices.

The following tutorial will give a sneak view on the power of this Python class.

First steps

- Make sure that you have installed Python on your computer. You can download WinPython (here), Anaconda (cross-platform) (here) or any other developing platform you prefer.
- Download the files for the inSphero chip tilter from GitHub (link) and copy them into a local folder on your computer.

How to start a program

The best way to run a script in development is to use one of the development platforms mentioned above. This also allows you to go through your code step by step, in case any weird behavior is observed or errors are present.

Once the script or program is finished, it should be possible to start the script by just double clicking the corresponding file, in case you have registered a Python installation on your computer. You can also open Window's command line and navigate to the script folder and type in the name of the script and hit enter:

```
## navigate to folder where script is located:
C:\Windows\system32>cd C:\Users\username\Documents\Python\
C:\Users\username\Documents\Python>

## execute script:
C:\Users\username\Documents\Python>MyTestScript.py  ## --> hit enter here ...
```

Hello world, hello tilter!

Import modules and classes to your project file and create and instance of the tilter class:

```
from ChipTilterCore import ChipTilterCore

# create tilter object and initialize
tilter = ChipTilterCore()
```

When you execute the program, the initialization tries to find an inSphero chip tilter connected to any of your USB ports. The output should look like this, in case **no** tilting device is connected:

```
05/17/2017 02:20:42 PM - libs.ChipTilterCore - INFO - Try to detect inSphero chip
tilter...
05/17/2017 02:20:42 PM - libs.ChipTilterCore - INFO - Could not be found!
```

Whereas when you connect a tilting device it should look like this

```
05/17/2017 02:21:42 PM - libs.ChipTilterCore - INFO - Try to detect inSphero chip
tilter...
05/17/2017 02:21:42 PM - libs.ChipTilterCore - INFO - Found device 'USB Serial Port
(COM9)' on 'COM9'.
05/17/2017 02:21:42 PM - libs.ChipTilterCore - INFO - Initializing serial port.
```

Congratulations, you have just established your first connection to the tilting device!

NOTE: Not only the serial port was initialized in the above example, but a logger was started as well. It enables you yo track all communications initialized by your program, either written to the command prompt or to a file (see ...).

Reset memory of the tilting device

As we now have an initialized serial port to communicate to our tilting device, we can try to initially reset the tilter memory by using the class methods provided through our tilter object.

So let's reset the memory of the tilter:

```
# reset tilter memory to default values
tilter.ResetTilterMemory()
```

The printout on the command prompt should look like the following:

```
05/17/2017 02:21:42 PM - libs.ChipTilterCore - DEBUG - Sent ['ff', '07', '00', '06',
'23'] to tilter
05/17/2017 02:21:42 PM - libs.ChipTilterCore - DEBUG - Sent ['ff', '08', '00', '07',
'23'] to tilter
05/17/2017 02:21:42 PM - libs.ChipTilterCore - DEBUG - Sent ['ff', '09', '00', '08',
'23'] to tilter
05/17/2017 02:21:42 PM - libs.ChipTilterCore - INFO - Tilter setup updated.
```

The characters and numbers in the square brackets are hex code and represent a byte. The bytes encode for:

- 1+2: address

- 3: value set
- 4: checksum
- 5: end of command character '#'

Prepare setup for the tilting device

In order to setup your tilting device, parameters like positive tilting angle or negative waiting time have to be defined and send to the device.

For a basic tilting setup six values are mandatory:

- A+ - positive tilting angle (1 - 90)
- A- - negative tilting angle (1 - 90)
- M+ - positive moving time (1 - 100 s)
- M- - negative moving time (1 - 100 s)
- P+ - positive waiting time (0 - 99:59)
- P- - negative waiting time (0 - 99:59)

Here's the code for this:

```
##############################
### - SET TILTING ANGLES - ###
##############################
# set positive tilting angle to 45 degree
tilter.SetValue('posAngle', 45)
# set negative tilting angle to 30 degree
tilter.SetValue('negAngle', 30)
##############################
### - SET MOTION TIMINGS - ###
##############################
# set positive motion time to 12 sec
tilter.SetValue('posMotion', 12)
# set negative motion time to 15 sec
tilter.SetValue('negMotion', 15)
##############################
### - SET PAUSE TIMINGS - ###
##############################
# set positive waiting time to 30 sec
tilter.SetValue('posPause', '30')
# set negative waiting time to 1:30
tilter.SetValue('negPause', '1:30')
```

Write setup to the tilting device

Currently the tilter doesn't know anything about this values, it's just stored internally in the class members to allow easy multiple writing of the same setup.

So we still need to send the values to the tilting device. We can do that with the following code snippet:

```
# now let's write the setup to the tilter
tilter.WriteSetup()
```

The values on the screen of the tilting device should have been changed accordingly and a printout on the screen (or file) should be visible, similar to this:

```
05/17/2017 02:21:43 PM - libs.ChipTilterCore - DEBUG - Sent ['ff', '06', '1e', '23',
'23'] to tilter
05/17/2017 02:21:43 PM - libs.ChipTilterCore - DEBUG - Sent ['ff', '05', '01', '05',
'23'] to tilter
05/17/2017 02:21:43 PM - libs.ChipTilterCore - DEBUG - Sent ['ff', '07', '1e', '24',
'23'] to tilter
05/17/2017 02:21:43 PM - libs.ChipTilterCore - INFO - Tilter setup updated.
```

In total, nine should be printed on the screen, since eight registers were updated. Minutes and seconds of waiting times are stored individually in the tilting device, thus separate commands need to be sent.

NOTE: By pressing the select and up/down buttons you can verify the correct settings for the positive as well as negative side.

Start tilting

So far we setup the most important parameters of the tilting device, what allows us to start tilting:

```
# start tilting with current setup
tilter.StartTilter()
```

Since we not defined a total run time, the tilting device runs forever with the current setup or until we send the stop command.

The corresponding output look like this:

```
05/17/2017 02:02:32 PM - libs.ChipTilterCore - DEBUG - Sent ['ff', '0c', '01', '0c',
'23'] to tilter
05/17/2017 02:02:32 PM - libs.ChipTilterCore - INFO - Started tilting.
```

Stop tilting

Just for debugging purposes, we will wait for 10 seconds before we call the stop command. The tilting device will return to its initial position (horizontal).

```
# import sleep to delay stop command
from time import sleep


# sleep for 10 seconds
sleep(10)

# stop tilting
tilter.StopTilter()
```

And the corresponding output:

```
05/17/2017 02:02:42 PM - libs.ChipTilterCore - DEBUG - Sent ['ff', '0c', '00', '0b',
'23'] to tilter
05/17/2017 02:02:43 PM - libs.ChipTilterCore - INFO - Stopped tilting.
```

Defining events

It is maybe necessary to execute some functions on certain events of the tilter in order to synchronize other devices or events. A list of the supported events can be found under '*Setup parameters - Supported events*'.

To show the abilities of this functionality we first need to define some functions (callbacks) to be executed on the defined events. We should define four functions and a global variable (for simplicity):

```
# global var for start time of waiting event
sT = None

# to be called when tilting device is waiting on the positive side
def onPosWait():
    print('I am waiting on the positive side...')

# to be called when tilting device moves down on the negative side
def onNegDown():
    print('I am moving down on the negative side...')

# to be called directly when tilting device is waiting on the negative side to
calculate the delay
def onNegWait():
    global sT
    sT = time()

# to be called when tilting device is waiting on the negative side, but delayed by x
seconds
def onNegWaitDelay():
    print('I was waiting for %2.3f s to show this.' % (time()-sT))
```

After defining the functions, we can tell the tilter class that we want to execute them on a special event. Accordingly we have to define four events and the lines should look like the following:

```
# define event for waiting on the positive side
tilter.SetTilterEvent( 'onPosWait', onPosWait              )

# define event for moving down on the negative side
tilter.SetTilterEvent( 'onNegDown', onNegDown              )

# define event for waiting on the negative side to start the counter for the delay
tilter.SetTilterEvent( 'onNegWait', onNegWait              )

# define event for waiting on the negative side, but delayed by 5 seconds
tilter.SetTilterEvent( 'onNegWait', onNegWaitDelay, delay=5 )
```

NOTE: The third definition is only necessary because we want to show the delayed time in the function '*onNegWaitDelay()*'. So we have to store the starting time first to calculate the delay.

As you might have noticed, it is possible to define multiple functions for one event, independent from each other (with or without delay), like in definition 3+4.

After starting the tilter again, we should see this on the command prompt:

```
05/17/2017 03:57:13 PM - libs.ChipTilterCore - DEBUG - Sent ['ff', '0c', '01', '0c',
'23'] to tilter
05/17/2017 03:57:13 PM - libs.ChipTilterCore - INFO - Started tilting.
I am waiting on the positive side...
I am moving down on the negative side...
I was waiting for 5.000 s to show this.
```

We can even stop the tilter after one cycle using the '*onNegUp*' event and the '*StopTilter*' function:

```
# define stop command when the tilter moves up on the negative side - one cycle
finished
tilter.SetTilterEvent( 'onNegUp'  , tilter.StopTilter       )
```

NOTE: The chip tilter only sends the status data packages I+II every two seconds, in an alternating fashion (I, II, I, II, I, ...). This might cause a small delay to recognize the events and execute the callback functions.

Besides the delay it is also possible to define an iteration counter for a particular function and the event on that it should be called. In case you want to execute the function not everytime the tilter waits on the positive side, just set the parameter '*it*' > 1. An example could look like this:

```
# to be called when tilting device is waiting on the positive side, but only every
second time
def onPosWaitEveryTwo():
    print('I am waiting on the positive side...\nBut only every second time I am
executed.')

# define event to execute the function every second time the tilter is waiting on the
positive side
tilter.SetTilterEvent( 'onPosWait', onPosWaitEveryTwo       )
```

The result looks then like this, according to the function we defined before:

```
--> the first time the tilter is waiting on the positive side
I am waiting on the positive side...

--> the second time the tilter is waiting on the positive side
I am waiting on the positive side...
I am waiting on the positive side...
But only every second time I am executed.
```

Receiving parameters from the tilting device

Once the serial port to the tilter is established, we can receive the current setup from the tilter. That might be helpful when we don't wish to reset the setup always, but reset only on request. A list of all supported parameters can be found under '*Setup parameters - Supported parameters*'

We might want to get the current angle and the corresponding motion time:

```
# print current positive angle received from the tilter
print( 'A+ %s' % tilter.GetParameter('A+')   )

# print current positive motion time received from the tilter
print( 'M+ %s s' % tilter.GetParameter('M+') )
```

You might also define your own events depending on a countdown, e.g. motion or pause countdown.

NOTE: As previously mentioned, the chip tilter only sends the status data packages I+II every two seconds, in an alternating fashion (I, II, I, II, I, ...). This might lead to some delay for certain parameters to be updated.

Use non-verbose mode

By default the tilter logging module is initialized to print all messages starting from the lowest level 'DEBUG', as you could see in many of the previously shown examples. If you wish to hide some of these messages, you can change the level that should be shown. You should know that there are multiple levels defined but only three are used by the class: ERROR, INFO and DEBUG (highest to lowest level).

To completely suppress messages of level 'DEBUG' or even 'INFO', just pass the argument '*logLevel*' upon creation of your own tilter object to the class initialization:

```
# suppress debug messages and only show info + error
tilter = inSpheroChipTilterCore(logLevel='INFO')

# suppress debug and info messages, only show errors
tilter = inSpheroChipTilterCore(logLevel='ERROR')
```

You can also activate or inactivate certain levels during program execution, if necessary. Just use the method '*setLevel*' of the logger object, instead of passing it on initialization:

```
# create tilter object and initialize
tilter = ChipTilterCore()

# your code here
# ...
# ...

# somewhere in your program
tilter.logger.setLevel('INFO')
```

Defining multiple setups

Inheriting from *inSpheroChipTilterCore* class

In case you want your class be able to use the functionalities provided by the chip tilting class, you can easily inherit from it:

```
from libs.ChipTilterCore import ChipTilterCore

# define new class which inherits from 'ChipTilterCore'
class tilterControllerApp(ChipTilterCore):
```

You can download a class example '*tilterClassExample.py*' that uses this functionality and creates three buttons employing the tkinter module of Python.

Upon the initialization of the class '*tilterControllerApp*' the basic parameters for the tilting setup are written to the device. When you hit the '*Start tilter*' button the tilter will start running and when you hit '*Stop tilter*' it will stop.


```
05/17/2017 06:52:31 PM - libs.ChipTilterCore - DEBUG - Sent ['ff', '0c', '01', '0c',
'23'] to tilter
05/17/2017 06:52:31 PM - libs.ChipTilterCore - INFO - Started tilting.
05/17/2017 06:52:36 PM - libs.ChipTilterCore - DEBUG - Sent ['ff', '0c', '00', '0b',
'23'] to tilter
05/17/2017 06:52:36 PM - libs.ChipTilterCore - INFO - Stopped tilting.
```

And, of course, you can also reset the tilter with the '*Reset tilter*' button.

The corresponding code snippet look like this:

```
class tilterControllerApp(ChipTilterCore):
 def __init__(self, master)

 ...

        ################################
        ### - DEFINE GUI COMPONENTS - ###
        ################################

        # create three buttons: reset, start, stop tilter
        self.resetButton = tk.Button( master, text='Reset tilter' )
        self.startButton = tk.Button( master, text='Start tilter' )
        self.stopButton  = tk.Button( master, text='Stop tilter'  )

        # define functions that should be executed when button is clicked
        self.resetButton.config( command=self.onResetClick )
        self.startButton.config( command=self.onStartClick )
        self.stopButton.config ( command=self.onStopClick )

        # place the buttons on the window
        self.resetButton.pack()
        self.startButton.pack()
        self.stopButton.pack()

    def onResetClick(self):
        self.ResetTilterSetup()

    def onStartClick(self):
        self.StartTilter()

    def onStopClick(self):
        self.StopTilter()
```

Troubleshooting

Update problems

In case you should notice any irregularities while writing updating the tilter setup, it is also possible to use a '*force mode*', which writes the setup multiple time (default is three).

```
# use force mode to reset tilting setup
tilter.ResetTilterMemory(mode='force')

# similar here: use force mode to write setup
tilter.WriteSetup(mode='force')
```

NOTE: If you select this option, the update might appear to be quite slow for sending just a few commands through the USB port - in fact, it is. Each value has to be addressed individually followed by a waiting time of 50 ms. Let's assume we reset the device, then we need to reset 13 values. If we assume 2 ms for the command to be sent + 50 ms waiting and all this three times, then we end up with ~2 s for the reset

Setup Parameters

The default parameters can be found at the beginning of the class definition. You should not change them unless you want to extend functionality.

Addresses

Supported keywords to assign values to the corresponding parameters on the tilting device:

Address (names)	Valid values
'posAngle'	1-90 degrees
'negAngle'	1-90 degrees
'posMotion'	1-100 seconds
'negMotion'	1-100 seconds
'posPause'	0-99:59 (mm:ss)
'negPause'	0-99:59 (mm:ss)
'horPause'	0-99:59 (mm:ss)
'totTime'	0-99:59 (hh:mm)
'status'	See status bits

Status bits

The following list contains the status bits:

- 'stopTilter',
- 'startTilter',
- 'plusMinusOff',
- 'plusMinusOn',
- 'resetErrors' ,
- 'soundOff',
- 'tryReconnect',
- 'noControl'

Supported parameters

With the '*GetParameter*' method it is possible to access these parameters and handle them in your script.

Parameter (key)	Description
'ID'	Chip tilter ID
'A+'	Positive tilting angle
'A-'	Negative tilting angle
'M+'	Positive motion time
'M-'	Negative motion time
'm'	Current motion countdown (if not in motion => 0)
'P+'	Positive pause time
'P-'	Negative pause time
'p'	Current pause countdown (if not pausing=> 0)
'H'	Horizontal pause time
'T'	Total run time
't'	Current run time countdown (also contains seconds)
'S'	3 status bytes (see inSphero Operating Manual)

Supported events

In order to execute arbitrary (callback) functions synchronized with the tilter you can define events (multiple are possible). The following are supported:

Event	Description (function execution)
'onPosDown'	When chip tilter moves down on the positive side
'onPosUp'	When chip tilter moves up on the positive side
'onNegDown'	When chip tilter moves down on the negative side
'onNegUp'	When chip tilter moves up on the negative side
'onPosWait'	When chip tilter waits on the positive side
'onNegWait'	When chip tilter waits on the negative side