Documentation Guide

For the Portland State University lab under Ralf Widenhorn developing Pozyx

File and Folder Organization

The Pozyx-Arduino-library-master and Pozyx-Python-library-master folders contain all of the original code, including the tutorials, from the Pozyx development team.

House Code contains the code made by our PSU team

> Tutorials Altered has the tutorial files from Pozyx altered to do different things
>
> Original Programs has original programs written by our PSU team
>
> Modules has Python modules of useful functions, classes, and other pieces of code for importation and utilization in other programs
>
> Debug contains test debugging files
>
> Documentation contains documentation files, including this one

Data contains data recorded from various tests.

Python PEP8 Styling

We format our Python code with adherence to the PEP8 style guide for Python (https://www.python.org/dev/peps/pep-0008/). Here are some highlights from that style guide to simplify the rules to be followed:

Guideline	Example
Functions are lowercase with words separated by underscores for readability, no camelCase()	`get_time()`, `find_average_speed()`, `log_sensor_data()`
Variables are lowercase with words separated by underscores for readability, no camelCase	`foo_bar`, `foobar`, `start_time`, `speed`, `long_but_specific_variable_name`
Limit all lines to a maximum of 79 characters.	Every line of code
For flowing long blocks of text with fewer structural restrictions, limit lines to 72 characters.	This applies to docstrings, comments, and similar blocks of text
Top level function and class definitions are preceded by 2 blank lines	Top level means not nested within another class, object, or other

Method definitions are preceded by 1 blank line	Functions within classes are methods

Add whitespace where it is appropriate	`i = i + 1`
	`submitted += 1`
	`x = x*2 - 1`
	`hypot2 = x*x + y*y`
	`c = (a+b) * (a-b)`

Do not add extra whitespace or not enough whitespace	`i=i+1`
	`submitted +=1`
	`x = x * 2 - 1`
	`hypot2 = x * x + y * y`
	`c = (a + b) * (a - b)`

Python Commenting

Please, please, PLEASE comment the code that you write. Not only will it help others to understand what you have done, thus reducing the time and effort needed to explain things to others in the future, it will also help you to get a better understanding of the code that you have written and make things more readable.

For single line comments, use a hash before the line.

Example: `#this line is a comment`

For multi-line comments, surround the commenting text with two delimiters (""")

Example:

```
"""
This is the first line of a multi-line comment.
This is the second line of a multi-line comment.
This is the third line of a multi-line comment.
This is the last line of a multi-line comment.
"""
```

Python Docstrings

See the PEP 257 Docstring Conventions guide (https://www.python.org/dev/peps/pep-0257/) for more on the semantics and conventions associated with Python docstrings.

When commenting on the nature, purpose, input and output, implementation, or something else of a module, function, class, or method, use a document string or docstring. The purpose of docstring standardization is to provide a standard and easy way to read documentation about a module, function, class, or method and to provide a uniform documentation style so that IDEs like PyCharm can look through your code and know what the documentation is supposed to say to make pop-up documentation boxes look nice.

A docstring is a string literal placed at the first line of a module, function, class, or method definition. Docstrings will need to be identified by using delimiters or """triple double quotes""" around it. Use r"""raw triple double quotes""" if you use any backslashes in a docstring.

For single-line docstrings, simply put the docstring and its delimiters on one line. For example:

```
def kos_root():
    """Return the pathname of the KOS root directory."""
    global _kos_root
    if _kos_root: return _kos_root
    ...
```

For multi-line docstrings, include a summary line just like a one-line docstring, followed by a blank line, followed by a more elaborate description. The summary line may be used by automatic indexing tools; it is important that it fits on one line and is separated from the rest of the docstring by a blank line. The summary line may be on the same line as the opening quotes or on the next line. The entire docstring is indented the same as the quotes at its first line.

We use the reStructuredText/Sphinx method of styling docstrings. For example, look at this docstring documentation of a function. The docstring starts with a one line summary followed by an empty line. On the next line, “param” indicates the line documents a parameter, “paramtype” is the type (str, int, float) of the first parameter, “param_1” matches the name of the first parameter, and then the parameter is described. The next line does the same for the second parameter. The following line uses “return” to specify a description for the return value, and the next line uses “rtype” to document the type of the returned object. The key “raises” specifies any raised exceptions. Take note of where the full colons are on each line and what they surround as they are critical to an IDE’s ability to read the docstring.

```
def my_function(param_1, param_2):
    """
    One line summary of what this function does.

    :param paramtype param_1: param 1 description
    :param paramtype param_2: param 2 description
    :return: return description
    :rtype: type of the returned object
    :raises: description of an error raised by the function
    """

    ...
    return None
```