Matlab Code Standard for Robotics 41013

Code Standardisation:

- Lower case first letter all variables (e.g. thisIsMyVariable, anotherVariable)
- For loops you can use i and j, everywhere else name variables descriptively (avoid non-descript names such as a, b, c, x1, t1)
- Upper case first letter for all functions, classes and namespaces, such as "MyFunctioName?()", and "AnotherFunctionName?()"
- Use only classes for all core functionality and objects
- Put all classes into appropriate namespace (notice in the following example how the following classes could be confused since the words can mean different fundamental categories)

```
namespace Fruits { class Apple{}; class Orange{}; class Mango{}; }
namespace Computers { class Apple{}; class Windows{}; class Linux{}; }
namespace RegionalTowns { class Dubbo{}; class Orange{}; class Scone{} }
```

- For class member attributes use "self" and the name of the object before the name. For example self.myVariableName.
- Never use underscores in function names or anywhere else in any variables except in certain cases
    - Constants such as MILLIMETERS_PER_METER so it is easier to read
    - handles "_h" such as a figure handle "fig_h"
- Always use "<" for "less than" symbols for comparisons. (i.e. if (0 < i).... not if (i > 0))
- Always write *greater than equal to* and *less than equal to* in form of "a <= b" so still always use this sign " < " (not b >= a)
- For loops should be written as (note: spaces, incrementing of i; and the comparator <)

```
for i = 2:2:10
  /// Do something
  /// Do something else
end
```

- In the case of *if statements*, then write as follows

```
if variable1 < variable2
  /// Do one thing
  /// Do another thing
end
```

- For a constant or a define use all caps and underscores between word i.e. VARIABLE_NAME
- For function that get or set internal variables.
    - Use "Set" and "Get" before the variable name.
    - For a variable such as stepHeight, the set function would have the variable's first letter capitalised (i.e. SetStepHeight?)
    - So `SetStepHeight()` that sets the parameter stepHeight
    - `GetStepHeight()` to get a protected internal variable, or one which must be calculated
- Upper case first letter of all words in variable name (e.g. heightValue, and metersToSurface)
- When taking up multiple lines for ease of reading, See the example

```
robotBaseTransform =[1, 0, 0, 0 …
                   , 0, 1, 0, 0 …
                   , 0, 0, 1, 0];
```

Tests

- Use the unit testing framework https://sourceforge.net/projects/mlunit/?source=navbar
- Write test cases for all classes. These have the suffix Tests after the classname and are in the same directory as the class. So test a class named ArmControl? with a test case ArmControlTests?.
- All tests should be simple and self contained. No longer than 1 page on screen long
- Each test must test something. Log files can be used for debug, but don't expect others to check outcomes in log files
- Ideally there should be a test for all externally accessible class functions

Doxygen

- Install doxygen and graphviz. Document your code using doxygen and build using https://github.com/simgunz/doxymatlab. Note: Make sure the doxygen is added to the system path.

Logging

- Download and use the Matlab logger
- http://au.mathworks.com/matlabcentral/fileexchange/33532-log4matlab
- Remember though that, even when the macro is off these log macros take some time, so for functions called millions of times, and those that are so simple there is no chance of a crash, then don't put a log message.
- In order to set the file with which to log to

```
L = log4matlab('logFileName.log');
```

- To log a debug message about a class called 'myClassName'

```
L.mlog = {L.DEBUG,'myClassName','This is a debug message'};
```

- To log a warning message about a function called 'SomeFunction'

```
L.mlog = {L.WARN,'SomeFunction','This is a warning message'};
```

- To log an error message about a script called 'ThisIsArbitrary'

```
L.mlog = {L.ERROR,'ThisIsArbitrary','This is an error message'};
```

- To set the logger level for class called 'myClassName' so that only warnings and errors are shown. Then to send 3 log messages of which only 2 will now be logged

```
L.SetLoggerLevel('myClassName',L.WARN)
L.mlog = {L.DEBUG,'myClassName','This is a debug message'};
L.mlog = {L.WARN,'myClassName','This is a warning message'};
L.mlog = {L.ERROR,'myClassName','This is an error message'};
```

- To set the logger level for function called 'SomeFunction' so that only errors are shown. Then to send 3 log messages of which only 1 will now be logged

```
L.SetLoggerLevel('SomeFunction',L.ERROR)
L.mlog = {L.DEBUG,'SomeFunction','This is a debug message'};
L.mlog = {L.WARN,'SomeFunction','This is a warning message'};
L.mlog = {L.ERROR,'SomeFunction','This is an error message'};
```

- To get the logger level for myClassName, SomeFunction and ThisIsArbitrary and see they are equal to what we expect

```
L.GetLoggerLevel('myClassName') == L.WARN
L.GetLoggerLevel('SomeFunction') == L.ERROR
L.GetLoggerLevel('ThisIsArbitrary') == L.DEBUG
```

- To allow WARN and ERROR messages be printed to the command window

```
L.SetCommandWindowLevel(L.WARN)
```

- To set back to no message printed to the command window

```
L.SetCommandWindowLevel(L.NONE)
```

- To log a matrix

```
L.mlog = {L.DEBUG,'myClassName',['The transform
is',L.MatrixToString(eye(4))]};
```

- To Log an exception (such as generated when trying to access a zeroth (0th) index)

```
try
   a(0);
catch ME
   L.mlog = {L.DEBUG,'myClassName',['There was an
error',L.ExceptionToString(ME)]};
end
```

- To get the name of the current function inside a class (MUST RUN IN THE example in TestClass2.m)

```
self.L.mlog = {self.L.DEBUG,mfilename('class'),[self.L.Me,'a
=',num2str(a)]};
```