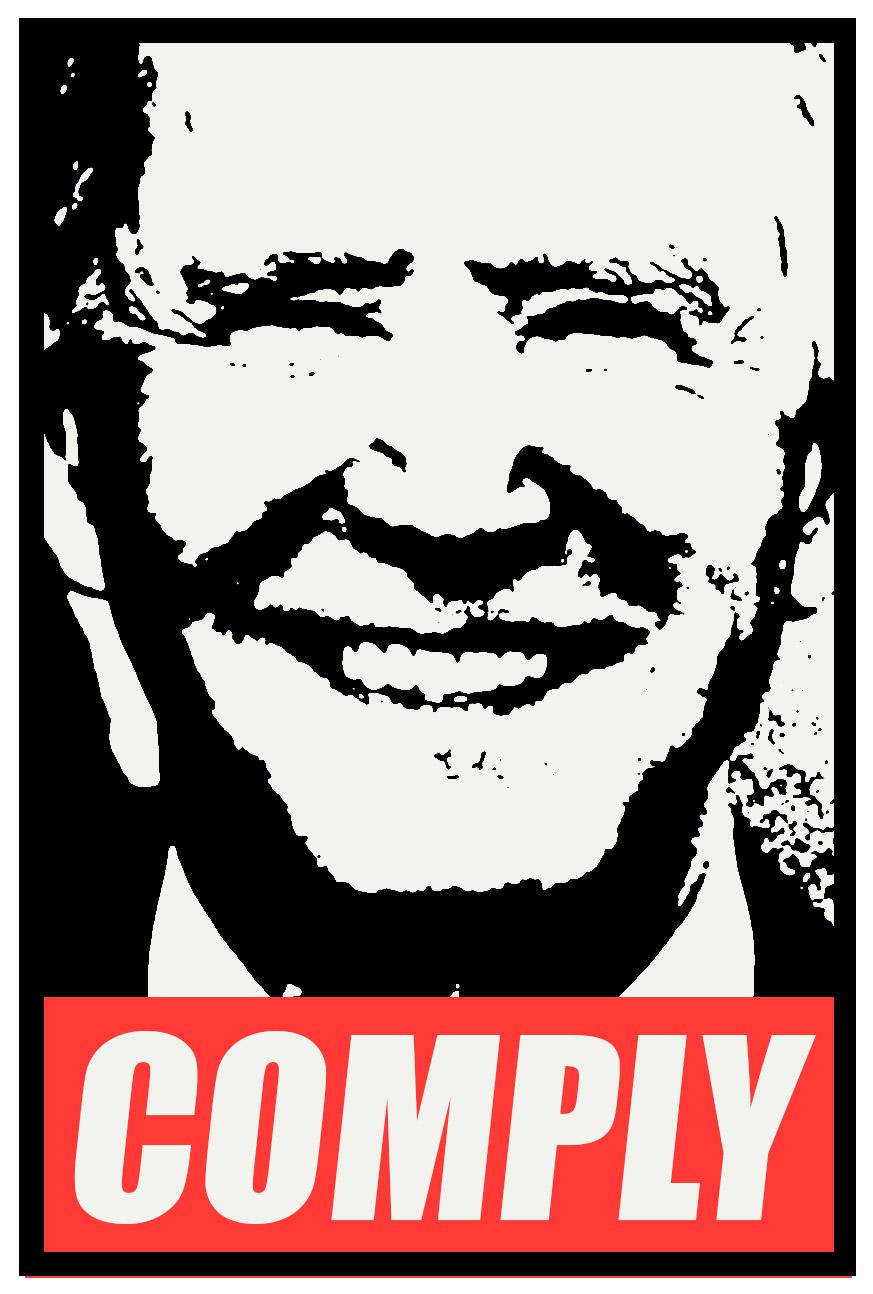
CHRISTIAN’S MAGNIFICENT LaTex Style Guide



Contents

[General Rules: 2](#_Toc83819490)

[Labels: 2](#_Toc83819491)

[Figures 3](#_Toc83819492)

[Single, centered figures: 3](#_Toc83819493)

[Multiple figures, same line: 4](#_Toc83819494)

[Multiple figures, individual subcaptions: 4](#_Toc83819495)

[Equations: 6](#_Toc83819496)

[Single, free-standing equations: 6](#_Toc83819497)

[Multiple equations, multiple equation numbers: 6](#_Toc83819498)

[Multiple equations, single equation number: 7](#_Toc83819499)

[Matrix example: 7](#_Toc83819500)

# General Rules:

## Labels:

* ONLY preface with object type, e.g. \label{eq:foo\_equation} for an equation, \label{fig:foo\_fig} for a figure.
* Labels must be so unique that chapter/section/etc references are unnecessary. Labels should indicate the nature of the object at first glance.
* Numbering is **100% forbidden**. If I see e.g. ”\label{fig:figure4} I *will* find you.
* References to objects always via the \cref command. This will automatically create text of e.g. the type ”Figure 6”.

# Figures

## Single, centered figures:

LaTeX:

\begin{figure}[h]

\centering

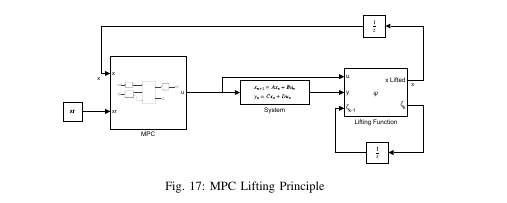
\includegraphics[width=0.8\linewidth]{Graphics/sim\_lifting}

\caption[MPC Lifting Principle]{MPC Lifting Principle}

\label{fig:MPC\_lifting}

\end{figure}

Result:



## Multiple figures, same line:

LaTeX:

\begin{figure}[ht]

\centering

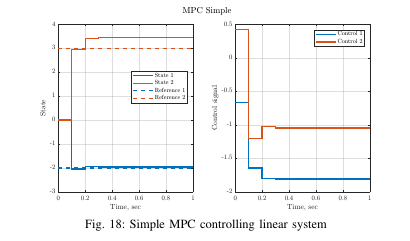
\includegraphics[width=0.7\linewidth]{Graphics/MPC\_Simple\_Fig4}

\caption[Simple MPC]{Simple MPC controlling linear system}

\label{fig:mpc\_simple}

\end{figure}

Result:



## Multiple figures, individual subcaptions:

LaTeX:

\begin{figure}[thpb]

\centering

\begin{subfigure}[t]{0.5\textwidth}

\centering

\includegraphics[height=2in,width=1\linewidth]{"Graphics/IFAC\_controlled\_v02"}

\caption{Control on $T\_{sec}$ and $T\_{SH}$, HDMDc-MPC}

\end{subfigure}%

~

\begin{subfigure}[t]{0.5\textwidth}

\centering

\includegraphics[height=2in,width=1\linewidth]{"Graphics/Dehghan\_Control"}

\caption{Control on $T\_{sec}$ and $T\_{SH}$, \cite{Dehghan2018}}

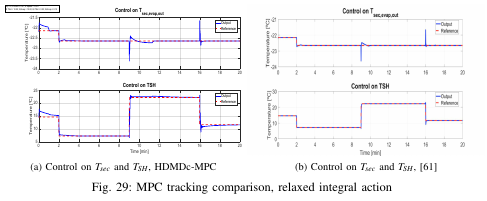
\end{subfigure}

\caption{MPC tracking comparison, relaxed integral action}

\label{fig:MPC\_IFAC\_States\_2}

\end{figure}

Result:



# Equations:

## Single, free-standing equations:

LaTeX code:

\begin{equation}\label{eq:MPCSys}

x\_+ = Ax + Bu

\end{equation}

Result:



## Multiple equations, multiple equation numbers:

**Note that EACH equation has its own label**

LaTeX:

\begin{align}\label{eq:MPC:init}

&x(0) = \bar{x} \\

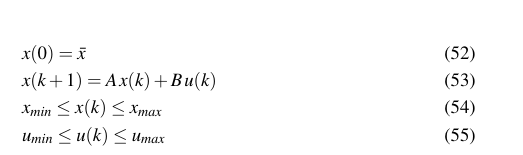
&x(k+1) = A \, x(k) + B \, u(k)

\label{eq:MPC\_sys\_dym} \\

&x\_{min} \leq x(k) \leq x\_{max} \label{eq:MPC\_x\_constraint} \\

&u\_{min} \leq u(k) \leq u\_{max} \label{eq:MPC\_u\_constraint}

\end{align}



## Multiple equations, single equation number:

**Note that the entire set of equations has ONE label**

LaTeX:

\begin{equation}\label{eq:MPCQP}

\begin{alignedat}{2}

&\text{minimise} \quad &&\frac{1}{2}\sum\_{k=1}^{N\_{p}}(x[k]^TQx[k] + u[k]^TRu[k]) + \frac{1}{2} x[N\_{p}+1]^TQ\_{N\_{p}+1} \, x[N\_{p}+1] \\

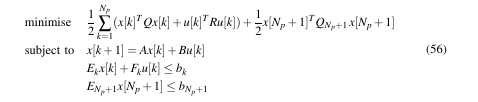
&\text{subject to} \quad &&x[k+1] = Ax[k] + Bu[k] \\

& &&E\_kx[k] + F\_ku[k] \leq b\_k \\

& &&E\_{N\_{p}+1} x[N\_{p}+1] \leq b\_{N\_{p}+1}

\end{alignedat}

\end{equation}



## Matrix example:

LaTeX:

\begin{equation}\label{eq:DenseFormConstraints}

\begin{alignedat}{2}

Q\_D &\triangleq

\begin{bmatrix}

I \otimes Q & \bar{0}\\ \bar{0} & Q\_{N\_{p}}

\end{bmatrix}, \quad R\_D &&\triangleq I \otimes R \\

E\_D &\triangleq \begin{bmatrix}

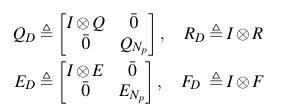
I \otimes E & \bar{0} \\ \bar{0} & E\_{N\_{p}}

\end{bmatrix}, \quad F\_D &&\triangleq I \otimes F

\end{alignedat}

\end{equation}

Result:



# Git

## Git commit messages:

Your commit messages should be extending “*This commit will* “

***Examples (wrong)***

*This commit will* *update valve models*

*This commit will* *remove all bugs in the code and introduce 400 new bugs*

***Examples (right)***

*Update valve models*

*Remove all bugs in the code and introduce 400 new bugs*

If you can’t describe the commit by one sentence, your commit is too big. Keep commits to reduced to one component/section

***Example (wrong)***

*Update valve models ~~and pump models~~*

Should be two commits, to allow for keeping changes of valve alone if the pump update is wrong, hence

***Example (right)***

Commit 1: *Update valve models*

Commit 2: *Update pump models*

# Code:

## Matlab

#### Variable name convention

##### Style

Functions: PascalCase

Variables: camelCase

Classes: ?

Iterator variables: i, j, k,

##### Name

Use names that describe the variable/function

***Example (wrong)***

varTempRasmus123 = 1e6

***Example (right)***

dataPerSec = 1e6

# Figure formats

## Matlab figures

In order to streamline figure format, save the figures by using the specified format:

### Single figure

Script for saving figures in right format:

f = fullfile(savepath,filename)

exportgraphics(figures(i), f, 'Resolution', 400)

***Example***

savepath = ’Bachelor\figures'

filename = "01alpha.png"

f = fullfile(savepath,filename(i))

exportgraphics(figure(1), f, 'Resolution', 400)

Note: if you want to save the figure in a specific format, change the .png of filename to .pdf or whatever you want.

# A bit too extensive..

**Variables**

The names of variables should document their meaning or use.

**Variable names should be in mixed case starting with lower case.**

This is common practice in the C++ development community. TMW sometimes starts variable

names with upper case, but that usage is commonly reserved for types or structures in other

languages.

linearity, credibleThreat, qualityOfLife

An alternative technique is to use underscore to separate parts of a compound variable name. This

technique, although readable, is not commonly used for variable names in other languages.

Another consideration for using underscore in variable names in legends is that the Tex

interpreter in **MATLAB** will read underscore as a switch to subscript.

**Variables with a large scope should have meaningful names. Variables with a small**

**scope can have short names.**

In practice most variables should have meaningful names. The use of short names should be

reserved for conditions where they clarify the structure of the statements. Scratch variables used

for temporary storage or indices can be kept short. A programmer reading such variables should

be able to assume that its value is not used outside a few lines of code. Common scratch variables

for integers are i, j, k, m, n and for doubles x, y and z.

**The prefix *n* should be used for variables representing the number of objects.**

This notation is taken from mathematics where it is an established convention for indicating the

number of objects.

nFiles, nSegments

A MATLAB-specific addition is the use of m for number of rows (based on matrix notation), as

in

mRows

**A convention on pluralization should be followed consistently.**

A suggested practice is to make all variable names either singular or plural. Having two variables

with names differing only by a final letter s should be avoided. An acceptable alternative for the

plural is to use the suffix Array.

point, pointArray

**Variables representing a single entity number can be suffixed by *No* or prefixed by *i*.**

The No notation is taken from mathematics where it is an established convention for indicating an

entity number.

tableNo, employeeNo

The i prefix effectively makes the variables named iterators.

iTable, iEmployee

**Iterator variables should be named or prefixed with *i*, *j*, *k* etc.**

The notation is taken from mathematics where it is an established convention for indicating

iterators.

for iFile = 1:nFiles

:

end

Note that applications using complex numbers should reserve i, j or both for use as the imaginary

number.

For nested loops the iterator variables should be in alphabetical order.

For nested loops the iterator variables should be helpful names.

for iFile = 1:nFiles

for jPosition = 1:nPositions

:

end

:

end

**Negated boolean variable names should be avoided.**

A problem arises when such a name is used in conjunction with the logical negation operator as

this results in a double negative. It is not immediately apparent what ~isNotFound means.

Use isFound

Avoid isNotFound

**Acronyms, even if normally uppercase, should be mixed or lower case.**

Using all uppercase for the base name will give conflicts with the naming conventions given

above. A variable of this type would have to be named dVD, hTML etc. which obviously is not

very readable. When the name is connected to another, the readability is seriously reduced; the

word following the abbreviation does not stand out as it should.

Use html, isUsaSpecific, checkTiffFormat()

Avoid hTML, isUSASpecific, checkTIFFFormat()

**Avoid using a keyword or special value name for a variable name.**

MATLAB can produce cryptic error messages or strange results if any of its reserved words or

builtin special values is redefined. Reserved words are listed by the command iskeyword. Special

values are listed in the documentation.

**Constants**

**Named constants (including globals) should be all uppercase using underscore to**

**separate words.**

This is common practice in the C++ development community. Although TMW may appear to use

lower case names for constants, for example pi, such builtin constants are actually functions.

MAX\_ITERATIONS, COLOR\_RED

**Constants can be prefixed by a common type name.**

This gives additional information on which constants belong together and what concept the

constants represent.

COLOR\_RED, COLOR\_GREEN, COLOR\_BLUE

**Structures**

**Structure names should begin with a capital letter.**

This usage is consistent with C++ practice, and it helps to distinguish between structures and

ordinary variables.

**The name of the structure is implicit, and need not be included in a fieldname.**

Repetition is superfluous in use, as shown in the example.

Use Segment.length

Avoid Segment.segmentLength

**Functions**

The names of functions should document their use.

**Names of functions should be written in lower case.**

It is clearest to have the function and its m-file names the same. Using lower case avoids potential

filename problems in mixed operating system environments.

getname(.), computetotalwidth(.)

There are two other function name conventions commonly used. Some people prefer to use

underscores in function names to enhance readability. Others use the naming convention

proposed here for variables.

**Functions should have meaningful names.**

There is an unfortunate MATLAB tradition of using short and often somewhat cryptic function

names—probably due to the DOS 8 character limit. This concern is no longer relevant and the

tradition should usually be avoided to improve readability.

Use computetotalwidth()

Avoid compwid()

An exception is the use of abbreviations or acronyms widely used in mathematics.

max(.), gcd(.)

Functions with such short names should always have the complete words in the first header

comment line for clarity and to support lookfor searches.

**Functions with a single output can be named for the output.**

This is common practice in TMW code.

mean(.), standarderror(.)

**Functions with no output argument or which only return a handle should be named after**

**what they do.**

This practice increases readability, making it clear what the function should ( and possibly should

not) do. This makes it easier to keep the code clean of unintended side effects.

plot(.)

**The prefixes *get/set* should generally be reserved for accessing an object or property.**

General practice of TMW and common practice in C++ and Java development. A plausible

exception is the use of set for logical set operations.

getobj(.); setappdata(.)

**The prefix *compute* can be used in methods where something is computed.**

Consistent use of the term enhances readability. Give the reader the immediate clue that this is a

potentially complex or time consuming operation.

computweightedaverage(); computespread()

**The prefix *find* can be used in methods where something is looked up.**

Give the reader the immediate clue that this is a simple look up method with a minimum of

computations involved. Consistent use of the term enhances readability and it is a good substitute

for get.

findoldestrecord(.); findheaviestelement(.);

**The prefix *initialize* can be used where an object or a concept is established.**

The American *initialize* should be preferred over the British *initialise*. Abbreviation *init* should be

avoided.

initializeproblemstate(.);

**The prefix *is* should be used for boolean functions.**

Common practice in TMW code as well as C++ and Java.

isoverpriced(.); iscomplete(.)

There are a few alternatives to the is prefix that fit better in some situations. These include the

has*,* can and should prefixes:

hasLicense(.); canEvaluate(.); shouldSort(.);

**Complement names should be used for complement operations.**

Reduce complexity by symmetry.

get/set, add/remove, create/destroy, start/stop, insert/delete,

increment/decrement, old/new, begin/end, first/last, up/down,

min/max, next/previous, old/new, open/close, show/hide,

suspend/resume, etc.

**Avoid unintentional shadowing.**

In general function names should be unique. Shadowing (having two or more functions with the

same name) increases the possibility of unexpected behavior or error. Names can be checked for

shadowing using which -all or exist.

**General**

**Names of dimensioned variables and constants should usually have a units suffix.**

Using a single set of units is an attractive idea that is only rarely implemented completely.

Adding units suffixes helps to avoid the almost inevitable mixes.

incidentAngleRadians

**Abbreviations in names should be avoided.**

Using whole words reduces ambiguity and helps to make the code self-documenting.

Use computearrivaltime(.)

Avoid comparr(.)

Domain specific phrases that are more naturally known through their abbreviations or acronyms

should be kept abbreviated. Even these cases might benefit from a defining comment near their

first appearance.

html, cpu, cm

**Consider making names pronounceable.**

Names that are at least somewhat pronounceable are easier to read and remember.

**All names should be written in English.**

The MATLAB distribution is written in English, and English is the preferred language for

international development.