mmCEsim

https://mmcesim.org

mmCEsim Documentation & Tutorials

Task-oriented mmWave Channel Estimation Simulation

Version 0.2.2

Wuqiong Zhao (Teddy van Jerry)

January 12, 2024

The application 'mmCEsim' and this document (mmCEsim Documentation & Tutorials) are open source and distributed by an MIT License.

MIT License

Copyright © 2022 – 2024 Wuqiong Zhao (Teddy van Jerry)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The latest edition of this document (mmCEsim Documentation & Tutorials) can be freely accessed online at https://pub.mmcesim.org/mmCEsim-doc.pdf or mmces.im/pdf for short.

Edition 2024/01/12 (corresponding to mmCEsim version 0.2.2).

mmCEsim Website: https://mmcesim.org
Source of This Document: https://github.com/mmcesim/mmcesim-doc

Contents

Preface

Tip

mmCEsim documentation & tutorials are under development!

As a researcher in wireless communications and signal processing, I have always had a passion for programming. Since my first year at university when I started using C++ to accomplish amazing tasks, I have been convinced of the importance of software in research.

Despite this, many researchers underestimate the significance of software in implementation, simulation, and verification of algorithms. Scientific software and programming languages, along with libraries, have been the driving force behind advances in science. Therefore, I am proud to present mmCEsim, an open-source software that is not only easy to use but also free for all.

The idea for mmCEsim originated from the tedious process of writing C++ code for implementing compressed channel estimation for reconfigurable intelligent surface (RIS)-assisted multiple-input multiple-output (MIMO) systems. I was driven by a desire to get rid of these repetitive tasks and eliminate the need to spend so much time setting up simulations. Inspired by NYUSIM, I decided to create my own simulation software.

To make it even easier to use, I designed a programming language called ALG, with simple syntax for describing algorithms. This language can be converted into other languages, such as C++ and MATLAB, for simulation. To use mmCEsim, simply configure your system settings, decide on your channel estimation algorithm, and extend the sounding and estimation process with ALG.

At present, mmCEsim supports channel estimation based on compressed sensing in mmWave and is expected to be more general in the future. It is still under active development and evolving.

I would like to thank my professor, seniors, and fellow students for their help and inspiration. I would also like to express my gratitude to Jinwen Xu for designing the elegant LaTeX template, `beaulivre`, which has made this document possible.

WUQIONG ZHAO
Nanjing, China
January 2024

This page is intentionally left blank

List of Figures

This page is intentionally left blank

List of Tables

This page is intentionally left blank

I

PRELIMINARY

Make preparations before we start.

This page is intentionally left blank

Preview 1

Before diving into documentation details, let's first have a preview of mmCEsim. Maybe you are not sure whether your research or study need this powerful tool, then read this chapter to have a glimpse of mmCEsim.

1.1 Introduction

The application is dedicated to simulate millimeter wave (mmWave) channel estimation:

$$\text{mmCEsim} = \textbf{mm}\text{Wave} + \textbf{C}\text{hannel } \textbf{E}\text{stimation} + \textbf{sim}\text{ulation},$$

where reconfigurable intelligent surface (RIS), also known as intelligent reflecting surface (IRS) [1] is supported for multiple input multiple output (MIMO) systems.

Figure 1.1: mmCEsim banner.

We offer a task-oriented simulation software for researchers to focus on algorithms only without being bothered by coding.

1.2 Features

Here is a list of basic features of mmCEsim:

- Task-oriented mmWave channel estimation formulation;
- Customizable system model;
- Extendable algorithms with our designed ALG language;
- Multiple RISs support;
- Automatic report generation (in plain text and LaTeX PDF);
- Well-written documentation with examples and tutorials.

1.3 Algorithm Background

The task-oriented channel estimation for (RIS-assisted) mmWave MIMO systems is implemented with compressed sensing (CS), which exploits the sparsity of mmWave channels.

1.4 Software Implementation

Based on the algorithm background, we implement this software with command line interface (CLI), graphic user interface (GUI), web application and a VS Code extension. The workflow of mmCEsim is depicted in Fig. 1.2.

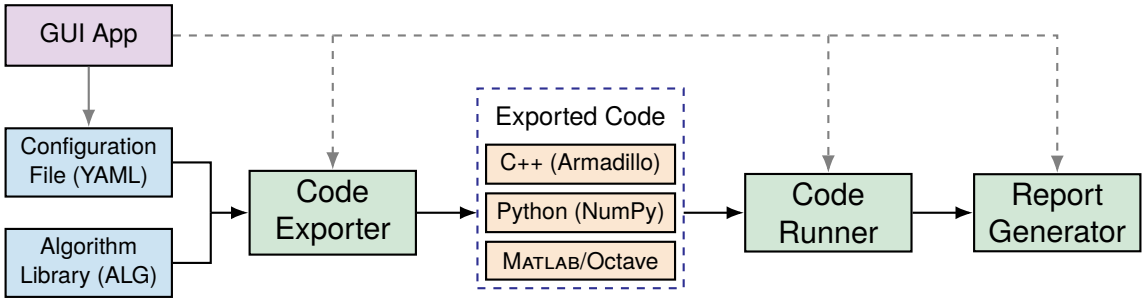


Figure 1.2: mmCEsim workflow.

2 Installation

2.1 Download Binary

You can download the built binary of mmCEsim from GitHub releases. The built CLI binaries include support for Linux (x86), macOS (x64 and arm) and Windows (x86).

They all statically link to libraries, so theoretically no dependency is needed.

> **Note**
>
> Since GitHub Actions currently only provide x86_64 machines, the binary for macOS with arm architecture is built manually on my MacBook Air with an M1 chip.

2.2 Build from Source

Since mmCEsim is built with CMake, so you can easily build the source on Unix-based systems. For Windows, I think there are similar ways.

On a Unix-based system, you can simply use the following code to build and install mmCEsim.

```
git clone https://github.com/mmcesim/mmcesim.git --recurse-submodules
cd mmcesim
mkdir build
cmake . build
cd build
make
sudo make install
```

The option `--recurse-submodules` is required because some dependencies of mmCEsim are managed by Git submodules.

You need to have a C++ compiler that supports C++17 standard, and have installed the Boost library (statically) of minimum version `1.70.0` on your system. You can install them easily on Unix-based systems with your favourite package manager. For Windows users, please follow the official instruction of Boost.

```
# Debian, Ubuntu
sudo apt install libboost-dev
# Arch
sudo pacman -Ss boost
# macOS
sudo port install boost # with MacPorts
brew install boost      # with HomeBrew
```

If you want to build the GUI app as well, you need to install Qt6.

Some options can be configured when calling `cmake`.

- `CMAKE_BUILD_TYPE`: Build type (default as `Release`)
- `CMAKE_INSTALL_PREFIX`: Installation prefix (default as system path)
- `MMCESIM_BUILD_ASTYLE`: Build `astyle` code ormatter (default as `ON`)
- `MMCESIM_BUILD_LOG`: Build mmCEsim log tool (default as `ON`)
- `MMCESIM_BUILD_MAINTAIN`: Build mmCEsim maintenance tool (default as `ON`)
- `MMCESIM_BUILD_GUI`: Build mmCEsim GUI App with Qt (default as `OFF`)
- `MMCESIM_APPLE_COPY_SH`: Copy additional shell script for macOS (default as `OFF`).
- `MMCESIM_TESTS`: Run mmCEsim tests (default as `ON`).

For example, you may use `cmake . build -D CMAKE_INSTALL_PREFIX=usr/mmcesim` to install mmCEsim to the directory `usr/mmcesim`.

2.3 Troubleshooting

2.3.1 macOS Safety Warning

You may view a safety warning after downloading the binary from GitHub Releases. The `trust_mmcesim.sh` is a script to remove that warning. (Give the script proper permission before running in its directory). Technically, it does `xattr -r -d com.apple.quarantine <binary>`.

Cannot find your problems here?

- If you have a bug to report, a suggestion for developers, or an issue relating to the software itself, feel free to open an issue on GitHub;
- If you have a general question to ask, you can join the discussions on GitHub;
- Or you can directly send emails to contact@mmcesim.org.

II

DOCUMENTATION

Every syntax and option in details.

This page is intentionally left blank

3 CLI Application

3.1 CLI Options

3.1.1 Help Yourself

With `mmcesim -h`, you can view all supported commands and options.

```
mmCEsim 0.2.1  (C) 2022-2023 Wuqiong Zhao
Millimeter Wave Channel Estimation Simulation
=============================================

Usage: mmcesim <command> <input> [options]

Commands:
  sim [ simulate ]        run simulation
  dbg [ debug ]           debug simulation settings
  exp [ export ]          export code
  cfg [ config ]          configure mmCEsim options
  (Leave empty)           generic use

Allowed options:

Generic options:
  -v [ --version ]        print version string
  -h [ --help ]           produce help message
  --gui                   open the GUI app

Configuration:
  -o [ --output ] arg     output file name
  -s [ --style ] arg      style options (C++ only, with astyle)
  -l [ --lang ] arg       export language or simulation backend
  --value arg             value for configuration option
  -f [ --force ]          force writing mode
  -V [ --verbose ]        print additional information
  --no-error-compile      do not raise error if simulation compiling fails
  --no-term-color         disable colorful terminal contents
```

3.1.2 Command

The allowed commands are explained in the following.

3.1.2.1 exp

Command `exp` exports the `.sim` configuration and corresponding `.alg` algorithms to a selected language. Currently, only export to C++ with Armadillo is supported.

3.1.2.2 sim

Command sim simulates the exported code with the selected backend. Currently, only C++ with Armadillo is supported.

So far, only C++ compiler `g++` (default) and `clang++` are supported which can be configured with option `cfg` cpp. You may also need to configure additional C++ flags with `cfg cppflags` if by default the compiler cannot find armadillo library.

3.1.2.3 dbg

Debug the simulation. This is different from `sim` in that the generated C++ code is compiled with `-g3` instead of `-O3`. Therefore, debug information is retained.

3.1.2.4 cfg

Configure settings.

- Use `mmcesim cfg` *<name>* to show the value of *<name>*.
- Use `mmcesim cfg` *<name>* `--value=<name>` to set the value of *<name>* as *<value>*.

Example 3.1 (Configure C++)

```
mmcesim cfg cpp --value="clang++"
mmcesim cfg cppflags --value="-I/opt/local/include -L/opt/local/lib"
```

Source: https://github.com/mmcesim/mmcesim/blob/master/scripts/mac_config_cppflags_tvj.sh.

3.1.3 Options

3.1.3.1 -v (--version)

Print the version string of mmCEsim.

3.1.3.2 -h (--help)

See §3.1.1.

3.1.3.3 --gui

Open the GUI application.

3.1.3.4 -o (--output)

Set the output file name. No extension name is required, and is added automatically according to your backend settings. `.cpp` for C++, `.py` for Python, `.ipynb` for Jupyter, and `.m` for MATLAB or GNU Octave.

3.1.3.5 -s (--style)

Set C++ AStyle (code formatting) options.

3.1.3.6 -l (--lang)

Set the export language or simulation backend.

3.1.3.7 --value

The value for configuration options.

3.1.3.8 -f (--force)

Enable the force writing mode. This will overwrite existent output files.

3.1.3.9 `-V (--verbose)`

Print additional information.

3.1.3.10 `--no-error-compile`

Do not raise error if compiling fails. This is useful in `sim` and `dbg`.

3.1.3.11 `--no-term-color`

Disable colorful terminal contents.

> **Tip**
>
> mmCEsim also supports the `NO_COLOR` standard: *Command-line software which adds ANSI color to its output by default should check for a `NO_COLOR` environment variable that, when present and not an empty string (regardless of its value), prevents the addition of ANSI color.*

When you have a non-empty `NO_COLOR` environmental variable, the color output is disabled, and you no longer need the `--no-term-color` option.

3.2 Configuration

Configuration is written in a text file with extension `.sim` (actually, can be any file extension) in YAML syntax. [Required] keys need to be filled in, unless provided with a Default value. [Optional] keys can be specified. [Conditional] keys can be either required or optional, depending on other settings.

3.2.1 `version` — Default: `0.2.2` [Required]

This field takes a string value representing the targeted mmCEsim version. For compatibility convenience, this string can be used by the compiler to decide the behavior. The current default value is the same as the compiler version (`0.2.2`).

3.2.2 `meta` — [Optional]

This is a map that provides metadata which can be used in the report. The used fields now include `title`, `description`, `author`.

3.2.3 `physics` — [Optional]

This field is a map that contains physical system settings.

3.2.3.1 `physics/frequency` — Default: `narrow` [Required]

The frequency bandwidth is specified in this field, which can have value `narrow` for narrowband (default) or `wide` for wideband.

3.2.3.2 `physics/off_grid` — Default: `true` [Required]

This is actually about the model. With the geometric channel model with grid, there can be off-grid (or power leakage) problems. Recently, there are also super resolution formulations to solve the problem. But we still adopt the grid representation for its popularity and simplicity. By setting `off_grid` to false, the off grid effect is discarded, i.e. all angles fall on the grid. The default value is `true`.

3.2.3.3 `physics/carriers` — [Conditional]

For a wideband system, you may specify the number of carriers used in OFDM. Its corresponding macro in `CALC` is `` `CARRIERS_NUM` ``.

3.2.4 `nodes` — [Required]

A sequence (array) of nodes in the channel network. Transmitter (Tx), Receiver (Rx), Reconfigurable Intelligent Surface (RIS) are all considered node (channels are the connecting edges to these nodes). For each of its elements, you need to specify the following fields.

3.2.4.1 nodes/id [Required]

The id is used in channels so that we know the direction of channel.

3.2.4.2 nodes/role Default: RIS [Required]

Valid values include transmitter (Tx), receiver (Rx), and RIS (IRS, default value).
There should be one and only one transmitter and one receiver.

3.2.4.3 nodes/num Default: 1 [Required]

The field num can be used to replicate several copies of the same node. This is often used for multi-user scenarios. Currently, this value is discarded, and the number is always 1.

3.2.4.4 nodes/size [Required]

The size means the antenna/element number for a node. For transmitters and receivers, it is the antenna number. For RIS, it is the number of reflecting elements. The value is a scalar for uniform linear array (ULA), and is a 2-value array (for example [8, 4]) for uniform planar array (UPA). For a 2-value array that has the second value set to 1, it is still regarded as a ULA.

Its corresponding macros in CALC are `SIZE.T.x`, `SIZE.T.y`, `SIZE.T`, `SIZE.R.x`, `SIZE.R.y`, `SIZE.R`, `SIZE.*`.

3.2.4.5 nodes/beam [Required]

The number of beams. Dimensions similar to size. Its corresponding macros in CALC are `BEAM.T.x`, `BEAM.T.y`, `BEAM.T`, `BEAM.R.x`, `BEAM.R.y`, `BEAM.R`, `BEAM.*`.

3.2.4.6 nodes/grid Default: same [Required]

The number of grids. Dimensions similar to size. The default value is same, which denotes the number of beams is the same as the number of antennas (or the number of RIS reflecting elements).

Its corresponding macros in CALC are `GRID.T.x`, `GRID.T.y`, `GRID.T`, `GRID.R.x`, `GRID.R.y`, `GRID.R`, `GRID.*`.

3.2.4.7 nodes/beamforming [Required]

For a node with role transmitter (Tx) or receiver (Rx), it is the active beamforming as precoding and combing, respectively. For a RIS node, it is the **passive** reflection tensor.

- nodes/beamforming/variable [Required]
  For Tx or Rx, this sets the variable name of the beamforming matrix (for narrowband) or tensor (for wideband). For RIS, this is the variable name of the reflection tensor.
- nodes/beamforming/scheme Default: random [Required]
  Possible value is random (default) and custom.
- nodes/beamforming/formula [Conditional]
  Custom beamforming formula in ALG. This is required if you set scheme to custom.

3.2.5 macro [Optional]

User-defined macros. See §6.4 for details.

3.2.6 channels [Required]

This is a sequence (array) of channel links. The settings for each channel link is shown below.

3.2.6.1 channels/id [Required]

Similar to id in nodes, each channel link has a unique identifier as in field id.

3.2.6.2 channels/from [Required]

The node id which the link channel is transmitted from.

3.2.6.3 `channels/to` [Required]

The node `id` which the link channel transmits to.

3.2.6.4 `channels/sparsity` [Required]

The sparsity (number of clusters) of the channel.

3.2.6.5 `channels/gains` Default: `normal` [Required]

3.2.7 `sounding` [Required]

Information related to the sounding procedure is defined here. The position of `sounding` is shown in Fig. 3.1.

3.2.7.1 `sounding/variables` [Required]

Channel variable names are defined here, including

- `received`: received signal
- `noise`: received noise
- `channel`: cascaded channel

3.2.8 `preamble` [Optional]

This is the code part before main simulation (including sounding, estimation and report generation). The position of `preamble` is shown in Fig. 3.1.

> **Tip**
>
> Custom functions can be defined here.

This part is specified using the ALG language.

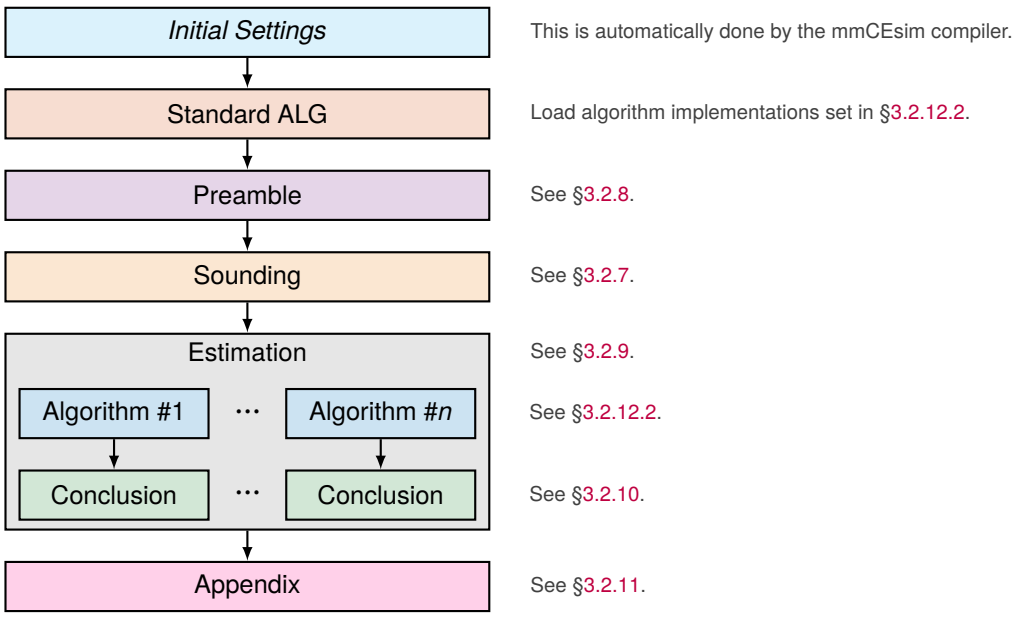


Figure 3.1: Sequence of simulation.

3.2.9 `estimation` Default: `auto` [Required]

This part is specified using the ALG language. The position of `estimation` is shown in Fig. 3.1.

3.2.10 `conclusion` [Optional]

This part is specified using the ALG language. The position of `conclusion` is shown in Fig. 3.1.

3.2.11 `appendix` [Optional]

This is the code part after all jobs are done. The position of `appendix` is shown in Fig. 3.1. This part is specified using the ALG language.

3.2.12 `simulation` [Required]

The main simulation configurations.

3.2.12.1 `simulation/backend` Default: `cpp` [Required]

Possible values are `cpp` (C++ with Armadillo library), `python` (Python with NumPy library), `matlab`, `octave`. This sets the language it exports to and the backend simulation bases on.

3.2.12.2 `simulation/jobs` [Required]

Simulation jobs. Each job is independent of one another.

- `simulation/jobs/name` [Required]
  Simulation job name which is used in the generated report.
- `simulation/jobs/test_num` Default: `50` [Required]
  Number of Monte-Carlo simulations.
- `simulation/jobs/SNR` [Required]
  Signal-to-noise ratio (SNR).
- `simulation/jobs/SNR_mode` Default: `dB` [Required]
  The mode of SNR. Possible values are `dB` (default) and `linear`.
- `simulation/jobs/pilot` [Required]
  The number of pilot overheads.
- `simulation/jobs/algorithms` [Required]

3.2.12.3 `simulation/report` [Optional]

Currently, this field is ignored. Both plain text report (`.rpt`) and LaTeX report will be generated.

3.3 Algorithm

Algorithm are defined in ALG language, please refer to §6 for details.

3.4 Tools

3.4.1 Compose

Compose `.sim` configuration file from command line options.

> NOTE
> This is still under development

3.4.2 Log

View or copy mmCEsim log file with `mmcesim-log`.

3.4.2.1 Log File

The log file is at `<install_prefix>/bin/mmcesim.log`. It stores information about the configuration and many internal processing details. This is especially useful for diagnosis.

Here is an example of the header part of a log file:

```
[INFO] * Time     : 2023-05-06 10:40:17 (UTC +0800)
[INFO] * Version  : 0.2.1
[INFO] * System   : MacOs-ARM
[INFO] * Compiler : clang v13.0.0
[INFO] * CLI Args : mmcesim "-h"
```

3.4.2.2 Usage

The available command line options are listed as follows:

- `-v` (`--version`): produce help message;
- `-h` (`--help`): print version string;
- `-p` (`--print`): print mmCEsim log on terminal;
- `-c` (`--copy`): copy mmCEsim log to clipboard;
- `-f` (`--file`): show mmCEsim log file location.

If no option is provided, it will use the `-p` option to print the log. You can use several options together, such as `-cp` to print and copy.

> **Tip**
>
> You can use `grep` to filter the log, for example use
>
> ```
> mmcesim-log | grep "\[ERROR\]"
> ```
>
> to get all error messages.

3.4.3 Maintain

You can view the latest stable version available with `mmcesim-maintain -l`. This internally invokes `curl https://mmcesim.org/VERSION`, so you need to have `curl` installed.

This page is intentionally left blank

4 GUI Application

The GUI application is written with Qt, and is currently undergoing a major update.

This page is intentionally left blank

5 Web Application

The web app address is https://app.mmcesim.org.
The example web app page is shown in Fig. 5.1.

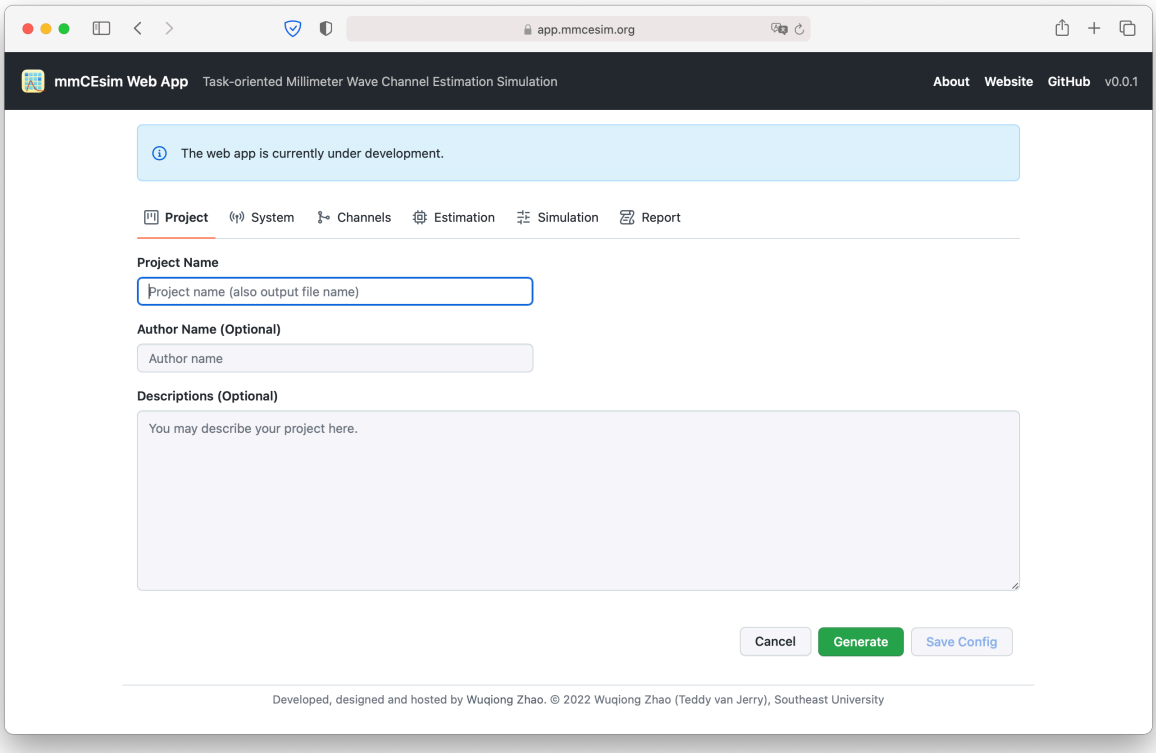


Figure 5.1: Web app interface.

Note

Since this app is hosted on my server, so it can be a little slow.

This page is intentionally left blank

6 ALG Language

6.1 Data Type

6.1.1 Why Need Data Type

Languages Python and Matlab/Octave are weakly typed which can be convenient for writing the code. However, that is problematic for implementation. The efficiency is not satisfactory compared to C++, and sometimes you may encounter ambiguous error information in Matlab. Therefore, for the sake of efficiency and generality, ALG language is designed to be **strongly typed**.

6.1.2 Structure

The type specification is very simple, because ALG language concentrates on matrices. Basically, the structure of ALG language is

prefix + dimension + suffix.

For example, `f2c` means a matrix (dimension is 2) with data type as float and property as a constant.

6.1.3 Specifiers

6.1.3.1 Prefix

Basic Type Prefix Basic type just names the element type. They are shown in Table 6.1.

Table 6.1: ALG variable basic type prefix.

Prefix	Type	C++ Type	Python Type	Matlab/Octave Type
c	Complex	cx_double	complex	complex
f	Float	double	double	double
i	Integer	int	int	int64
u	Unsigned Integer	uword	uint	uint64
b	Boolean	bool	bool	logical
s	String	std::string	str	string
h	Character	char	char	char

Table 6.2: ALG variable alias prefix.

Alias Prefix	Type	Equivalent Two-character Type
`v`	(Column) Vector	`c1`
`r`	Row Vector	`c2`
`m`	Matrix	`c2`
`t`	Tensor	`c3`
`d`	Double	`f0`

Alias Prefix Alias prefixes not only set the element type, but also the dimension. They are the one character alias for a two-character type. A list of alias prefixes is shown in Table 6.2.

> **!** `v`, `r`, `m` and `t` are all for **complex** types. For a non-complex type, you need to use the normal two-character way.
> Row vector (`r`) is actually regarded as a matrix for simplicity, so its dimension is still 2. Only column vector (`v`) is the real vector. But there can be differences in terms of `INIT`, so it should not be confused with `m`.

6.1.3.2 Dimension

Dimensions range from 0 to 3. Details are shown in Table 6.3.

Table 6.3: ALG variable dimension.

Dimension	Type	C++ Type
0	Scalar	—
1	Vector	`Col`
2	Matrix	`Mat`
3	Tensor	`Cube`

> **!** Dimension for a scalar can not be omitted.

Please note that matrices are stored in **column major** order, which is the default order in C++ (Armadillo) and Matlab/Octave. In Python (NumPy), it is equivalent to the option `order='F'`.

> **!** You should always remember the column **major order**, especially if you use are accustomed to Python. The order will make a big difference to matrix reshape and vectorization.

6.1.3.3 Suffix

All suffixes of ALG variables are shown in Table 6.4.

Table 6.4: ALG variable suffix.

Suffix	Meaning	C++	Python	Matlab/Octave
`c`	Constant	`const`	(None)	`persistent`
`r`	Reference	`reference`	(None)	(None)

Tip

Two suffixes cannot be used together and there is also no need to do so. The use of `r` is mainly in `function`, allowing a parameter to be changed inside the function body.

6.2 Function

6.2.1 Syntax Basics

The initiative of proposing a new programming language for algorithm implementation is based on the multi-backend design of mmCEsim. The language is specially designed so that it can be exported to C++ (with Armadillo), Python (with NumPy) and MATLAB/Octave easily.

Every line of ALG language calls a function. Let's first have a look at its basic structure before we cover its details.

```
1 ret1::type1 ret2 = FUNC param1 param2::type2 key1=value1 key2=value2::type3 # com.
```

It may look like an assembly language at the first glance, due to all parameters are separated by space. But it is actually much more convenient. Here are some basic rules:

- All tokens are separated by space.
- Function names are in all upper cases, like `CALC`, `WHILE`.
- Indentation does not matter. Blocks are ended with `END`.
- The function line is mainly composed of three parts: **return values**, **function name**, **parameters**, in the left to right direction.
- Some functions may not have return values, and you may also omit the return values. If there are return values, there is a = between return values and function names.
- Function name is the first word on the right of = (if there are return values) or the first word of line (if there is no return value).
- Like Python, parameters can be passed in by two ways:

  1) **value in position**: Like `param1` and `param2` in the above example. Parameters in different positions correspond to different usages in the function. This is the only way in C++.
  2) **key and value**: Parameters can also be specified using key and its corresponding value. `value1` and `value2` are passed in using this method. It should be noted that there should be no space around the = between key and value.

  There are some special cases that parameters are viewed as a whole, for example `COMMENT` and `CALC`.
- If a parameter contains space or special characters, you need to use the double quotes like `"param with space"` and escape special characters as in C++ and Python.
- You may optionally specify the type of return value and parameters with `::` after the value. For example, in the above example `dtype1`, `dtype2` and `dtype3` are type specifications for `ret1`, `param2` and `value2`, respectively. For more information about data type, please refer to data type of ALG language.
- Like Python, the backslash (\) at the end of the line can be used for continuing the function on next line.
- Comments start with the hash (#) like Python.

! There should be no space around the = between key and value for parameters. For example, `key=val` is valid while `key = val` is forbidden.

Special rules may be applied for different functions. Please refer to the specific documentation for each function.

6.2.2 BRANCH

Declare start of the scope of job algorithms.

Explanations

This is useful in `estimation`. Contents between BRANCH and MERGE will be repeated for different algorithms. So you need to place compressed sensing estimation ESTIMATE and RECOVER inside.

Example

Example of OFDM OMP.

6.2.3 BREAK

Break from a block (for FOR, FOREVER, LOOP, WHILE).

Explanations

The same as `break` in C++, Python and MATLAB/Octave. This function takes no parameter.

Example

Example with FOREVER.

6.2.4 CALC

Make arithmetic calculations.

Explanations

There are two kinds of CALC usage: **inline** and **standalone**:

- **inline**: The contents to be calculated are placed in a set of dollar signs, like LaTeX syntax: `$some operations to be calculated$`.
- **Standalone**: This is like a normal function, with function name as CALC. You may also omit the function name CALC since it is the default function name if nothing is specified. Therefore, `result = CALC your expression` is equivalent to `result = your expression`.

For more information about the CALC syntax, please refer to §6.3.

> **!** For safety, you should not use anything other than ANSI characters in CALC functions. Otherwise, there can be undefined behaviour.

If you want the calculation result to be a new variable, you may use function NEW.

Example

EXAMPLE 6.1 (Example of CALC)

```
a = CALC b + 2 # explicit CALC function
a = \sin(b) @ c # implicit CALC function
a = b^H + c^{-1} # conjugate transpose and inverse
c = b_{2, 3} # get element of a matrix
c = \abs{b_{:, 3}} + \pow(b_{}, 2) # use : in subscript & use {} for function
\exp2(a + c .* d) ./ e^T -f_{:,3,1:index} # element-wise operator and
 ↪ subscript : range
```

Equivalent C++ Code

```
a = b + 2;
a = arma::sin(b) * c;
a = b.t() + c.i();
```

```
c = b(2, 3);
c = arma::abs(b(arma::span::all, 3)) + arma::pow(b, 2);
arma::exp2(a + c % d) / e.st() - f(arma::span::all, 3, arma::span(1, index));
```

6.2.5 CALL

Call a custom function defined by FUNCTION.

6.2.6 COMMENT

Place a line of comment in the exported code.

Explanations

All contents after the function keyword COMMENT are considered as comments.

Example

Example 6.2 (Example of COMMENT)

```
COMMENT Hi, this is a comment!
```

Equivalent C++ Code

```
// Hi, this is a comment!
```

Equivalent Python Code

```
# Hi, this is a comment!
```

Equivalent MATLAB/Octave Code

```
% Hi, this is a comment!
```

6.2.7 CPP

Write standard C++ contents.

Explanations

All contents after the CPP keywords are copied to exported codes. For backend other than C++, this function is ignored.

Example

Example 6.3 (Example of CPP)

```
CPP std::cout << "Standard C++ Language!" << std::endl;
```

Equivalent C++ Code

```
std::cout << "Standard C++ Language!" << std::endl;
```

For Python, MATLAB/Octave, nothing will happen with the CPP function.

6.2.8 ELIF

ELIF is a shorthand for combining ELSE and IF statements into a continuous sequence.

Explanations

The parameter is the same as IF.

Example

Example with IF.

6.2.9 ELSE

Used in IF blocks.

Explanations

This function implements as `else` in C++, Python and MATLAB/Octave. There is no parameter for the ELSE function.

Example

Example with IF.

6.2.10 END

End of a block for ELSE, ELIF, FUNCTION, FOREVER, IF, LOOP, WHILE.

Explanations

In C++, this functions as `}`, in Python it is the indentation goes back for one block. In MATLAB/Octave, it is the `end` specification.

Example

Example with FOR, FOREVER, IF, LOOP, WHILE.

6.2.11 ESTIMATE

CALL standard ALG functions or your custom algorithms to estimate the sparse channel with compressed sensing (CS).

6.2.12 FOR

Start a `for` loop.

Explanations

The parameters are similar to C++, as listed in Table 6.5.

Table 6.5: ALG FOR function parameters.

Position	Parameter Key	Descriptions
1	`init`	Initialization before entering the loop.
2	`cond`	Condition to continue into the loop.
3	`oper`	Operation after each iteration.

> **!** If there is = or other special characters inside your parameter or there exists space, do remember to place them inside double quotes (").

Example

Example 6.4 (Example of FOR)

```
FOR "i::u0 = INIT 0" "i != 10" "i=i+2" # a for loop taking three parameters
  COMMENT "Do something here in the for loop."
END
```

Equivalent C++ Code

```
for (uword i = 0; i != 10; i = i + 2) {
    // Do something here in the for loop.
}
```

6.2.13 FOREVER

Repeat in the block until `BREAK`.

Example

Example 6.5 (Example of FOREVER)

```
FOREVER # takes no param
  BREAK # Wow, nothing is done when I just break here [Lol]
END
```

Equivalent C++ Code

```
while (1) {
    break;
}
```

6.2.14 FUNCTION

Start a function definition.

Explanations

The function requires an `END` to mark the end of the function.

6.2.15 IF

Conditional statement.

Explanations

This works the same as `if` in C++, Python, MATLAB/Octave. All contents after the `IF` keyword are part of the condition. If you insist using the key value style, the key is `cond`.

Example

Example 6.6 (Example of IF)

```
IF \accu(\pow(\abs(A), 2)) > 0.1 * threshold
  IF b < 0
    b = 0
  ELIF b > 100
    b = 100
  ELSE
```

```
    b = -b
  END
ELSE
  IF cond="c == d" # use key value style if you insist
    A = A * 0.1
  END
END
```

Equivalent C++ Code

```
if (arma::accu(arma::pow(arma::abs(A), 2)) > 0.1 * threshold) {
    if (b < 0) {
        b = 0;
    } else if (b > 100) {
        b = 100;
    } else {
        b = -b;
    }
} else {
    if (c == d) {
        A = A * 0.1;
    }
}
```

6.2.16 INIT

Initialize a variable.

Explanations

This function can initialize a **scalar**, a **vector**, a **matrix** and a **tensor**. The initialization target can be specified in two ways:

- **return value type specification**: You can specify the type of the variable to be initialized by `::`;
- **parameters**: Parameter `dtype` is used for element type, and `dim1`, `dim2`, `dim3` are used for dimension specification.

> **!** Please be **consistent**! The current implementation of the function is fragile and can be fooled by any inconsistent actions. While we are trying to enhance the error detection, you are advised to use the correct dimension.
> However, there are also a few exceptions for user's convenience. Though row vector (`r`) is regarded as a matrix, you can still specify its dimension with only one parameter on `dim1`. For a scalar initialization, the value can directly follow `=`.

The parameters are listed in Table 6.6. For initialization of a row vector (`r`), you may just use one dimension. For initialization of a scalar (dimension as 0), you can specify the value directly after `=`, but if you want to use `fill` and `scale`, you must specify the parameter key.

> **Tip**
>
> Since the development of ALG concentrates on matrix operations, the initialization also performs in a matrix-oriented way. Please refer to data type of ALG language if you are not sure how to use the data type.

Table 6.6: ALG `INIT` function parameters.

Position	Parameter Key	Descriptions
1	`dim1`	Size of the first dimension (for vector).
2	`dim2`	Size of the second dimension (for vector and matrix).
3	`dim3`	Size of the third dimension (for vector, matrix and tensor).
4	`fill`	Element filling mode. `randn` for standard normal distribution $\mathcal{N}(0,1)$, `randu` for standard uniform distribution $\mathcal{U}(0,1)$, `zeros` for filling as 0, `ones` for filling as 1. Default is `zeros`.
5	`scale`	Scale of the value. This works like multiplying a value after the initialization by `fill`.
6	`dtype`	Element data type. This is the one character prefix like `c`, `i`. The default value is complex (`c`).

Example

Example 6.7 (Example of INIT)

```
a = INIT 4 3 fill=ones scale=4 dtype=c # a matrix
b::r = INIT 10 scale="-1+i" # row vector (viewed as a matrix)
pi::f0c = INIT 3.1415926 # a const float
random_number = INIT fill=randn scale=-2
```

Equivalent C++ Code

```
cx_mat a = (4) * arma::ones<cx_mat>(4, 3);
cx_mat b = (-1 + i) * arma::zeros<cx_mat>(1, 10);
const double pi = 3.1415926;
std::complex<double> random_number = (-2) * arma::randn<std::complex<double
 ↪ >>();
```

6.2.17 LOG

Write to log file.

Example

Example 6.8 (Example of LOG)

```
LOG [INFO] A user-defined message.
```

Log file: `mmcesim.log`

```
[INFO] $ LOG [INFO] A user-defined message.
[INFO] * FUNCTION: LOG
[INFO] * PARAMS:
[INFO]     > {}={[INFO]}::{}
[INFO]     > {}={A}::{}
[INFO]     > {}={user-defined}::{}
[INFO]     > {}={message.}::{}
[INFO] A user-defined LOG message.
```

6.2.18 LOOP

Loop with iteration counter.

Explanations

This function uses an iteration counter to control the loop. The parameters are shown in Table 6.7.

Table 6.7: ALG LOOP function parameters.

Position	Parameter Key	Descriptions
1	`begin`	The starting iteration counter.
2	`end`	The end iteration counter (not included).
3	`step`	Iteration counter step.
4	`from`	The starting iteration counter.
5	`to`	The last interaction counter (included if step walks into it).

> **Tip**
>
> Normally, we use the `begin` + `end` format. You may also use `from` + `to` format, but the two formats cannot be used together.

The return value is the iteration counter. If it is not specified, the default one is `i`. You may also use `::` to specify the iteration counter type.

Example

Example 6.9 (Example of LOOP)

```
LOOP 0 10 # implicit counter name as 'i'
  COMMENT 0, 1, 2, ..., 9
  j = LOOP from=10 to=0 step=-1
    COMMENT 10, 9, 8, ..., 0
  END
  k::u0 = LOOP begin=0 end=10 step=2 # specify counter type
    COMMENT 0, 2, 4, 6, 8
  END
END
```

Equivalent C++ Code

```
for (auto i = 0; i < 10; ++i) {
    // 0, 1, 2, ..., 9
    for (auto j = 10; j >= 0; --j) {
        // 10, 9, 8, ..., 0
    }
    for (uword k = 0; k < 10; k += 2) {
        // 0, 2, 4, 6, 8
    }
}
```

6.2.19 PRINT

Print the contents.

Explanations

The function takes a list of parameters, and they can be matrices or scalar values. They are printed out from left to right.

Example

Example 6.10 (Example of PRINT)

```
PRINT 1 '\t' H '\n'
```

Equivalent C++ Code

```
std::cout << 1 << '\t' << H << '\n';
```

6.2.20 RECOVER

Declare the recovered channel.

Explanations

The normalized mean square error (NMSE) or bit error rate (BER) performance is evaluated with the recovered channel specified here (the only parameter that needs to be set).

Example

Example of MIMO OFDM, where `H_hat` is the recovered channel.

6.2.21 MERGE

Declare end the scope of job algorithms.

Explanations

This works with `BRANCH`. View documentation of `BRANCH` for details.
If you do not specify `MERGE`, it is assumed to be end of `estimation` by default.

> **Tip**
>
> It is advised to always specify `MERGE` so that the range between `BRANCH` and `MERGE` is clearer and can be extended by other users more easily.

Example

Example of OFDM OMP.

6.2.22 NEW

Create a new variable from `CALC` result.

Explanations

The parameters are the same as `CALC`, except from the value type is **required**.

Example

Example 6.11 (Example of NEW)

```
y::v = NEW H @ x + n
```

Equivalent C++ Code

```
cx_vec y = H * x + n;
```

6.2.23 WHILE

Repeat while the condition satisfies.

Explanations

This is the same as C++, Python and Matlab/Octave. The function takes only one parameter. (If you do need a key value style, it has key `cond`.)

> **Tip**
>
> Since only one parameter is needed, all contents after the `WHILE` keyword is viewed as the stop condition. So there is no need to quote the condition which is required in `FOR`.

Example

Example 6.12 (Example of WHILE)

```
WHILE i != 100 && result == false # no quote is needed because there will be only one param
  COMMENT "Do something in the while loop."
END
```

Equivalent C++ Code

```
while (i != 100 && result == false) {
    // Do something in the while loop.
}
```

6.3 Calculation (CALC)

6.3.1 Design Inspirations

Let's have a look at the example below.

```
\sqrt(\accu(\pow(\abs(A_{1,:,2:5} - B^{-1}), 2))) * c * A @ (D ./ E^T) + \sin(f_{2})
```

Well, that looks like LaTeX, right? The familiar \ character starting a command (sorry, that's called macro in LaTeX), and inverse as `^{-1}`, transpose `^T`, and subscript with `_{}`!

Well, let's write the above example in LaTeX (through some informal MATLAB style subscript)

Example 6.13 (Maths Representation and LaTeX Code)

$$\left\|\mathbf{A}_{1,:,2:5} - \mathbf{B}^{-1}\right\|_2 \cdot c\mathbf{A} \cdot \left(\mathbf{D} \oslash \mathbf{E}^{\mathsf{T}} + \mathrm{ones}(\sin(\mathbf{f}_2))\right), \tag{6.1}$$

where $\mathbf{A}, \mathbf{B}, \mathbf{D}, \mathbf{E}$ are matrices, c is a scalar, $\mathbf{f}$ is a vector, $\oslash$ represents the element-wise division, and ones represent a matrix with all elements as 1.

```
\left\|\mathbf{A}_{1,:,2:5}-\mathbf{B}^{-1}\right\|_2\cdot c\mathbf{A}\cdot\left(\mathbf{D}\oslash\mathbf{E}^\mathsf{T}+\mathrm{ones}(\sin(\mathbf{f}_2))\right)
```

Tip

As a matter of fact, that can be even more like LaTeX with all brackets can be converted to `{}`, though parameters are separated by `,` not another `{}`. For LaTeX fans, you may even use `^\star` or `^\ast` for conjugate in addition to the normal `^*`. Happy TeXing!
To make the language simple and efficiently convertible to C++, Python and Matlab/Octave, some syntaxes from Python and Matlab are adopted in addition to the LaTeX look.

6.3.2 Syntax Basics

There are commands starting with backslash (\), operators and subscripts, superscripts.

Variable Naming

- Variable names should not contain reserved characters `!@#$%^&*()[]{}\|/-+=~,.<>?:;"'`` `.
- Variable name should also not end with underscore (_).
- Digit can not be the first character of variable name.
- Variable name should not clash with reserved keywords.

Whitespace does not matter in CALC, which are removed before parsing.

6.3.3 Operators

6.3.3.1 Operator List

The operator list is shown in Table 6.8.

Table 6.8: CALC operators.

Operators	Description
+, -	(Unary/Binary) plus/minus
*	Scalar and scalar multiplication, scalar and matrix multiplication
@	Matrix and matrix multiplication
.*, ./	Element-wise multiplication, division
!	Logical Not
(), {}	Command call
<, <=, >, >=	Relational operator $<, \leq, >, \geq$
==, !=	Relational = and $\neq$
&&	Logical And
||	Logical Or
=	Assign

Matrix and matrix multiplication is different from matrix scalar and scalar scalar multiplication. You should distinguish the use of * and @. (This is like the Python syntax.)
Element-wise multiplication .* is different from matrix multiplication @. (This is like the Matlab syntax.)

6.3.3.2 Operator Precedence

Table 6.9 lists the precedence and associativity of ALG CALC operators. Operators are listed top to bottom, in descending precedence.[1]

Table 6.9: CALC operator precedence.

Precedence	Operators	Description	Associativity
1	`(), {}`	Command call	Left-to-right
2	`^T, ^H, ^t, ^i, ^*, ^{-1}`	Matrix superscript	Left-to-right
3	`_{}`	Matrix subscript	Left-to-right
4	`!, +, -`	Logical NOT, unary plus/minus	Right-to-left
5	`*, @, .*, ./`	Matrix (and element-wise) multiplication, division	Left-to-right
6	`+, -`	Addition and subtraction	Left-to-right
7	`<, <=, >, >=`	Relational operator $<, \leq, >, \geq$	Left-to-right
8	`==, !=`	Relational = and $\neq$	Left-to-right
9	`&&`	Logical AND	Left-to-right
10	`||`	Logical OR	Left-to-right
11	`=`	Assign	Right-to-left

6.3.4 Commands

6.3.4.1 Command Usage Basics

Here are the basic rules for command usage:

- Commands start with character `\`;
- Command call has parameter list inside brackets `()`, and you may optionally use `{}` if you want the syntax more like LaTeX;
- Parameters are separated by comma `,`.

The function naming convention mainly follows that of Armadillo which is also similar to MATLAB.

6.3.4.2 `\dictionary`

Generate a dictionary matrix for beamspace (virtual) representation.

Math Representation The complex dictionary matrix is used in compressed sensing (CS). For a uniform linear array (ULA) with size M and grid size M^G, the dictionary $\mathbf{V} \in \mathbb{C}^{M \times M^G}$ is defined as

$$\frac{1}{\sqrt{M}} \exp\left(-2\pi i \cdot d \cdot \mathbf{x}_M \mathbf{u}_{M^G}^{\mathsf{H}}\right), \tag{6.2}$$

where $\mathbf{x}_M = [0, 1, 2, \cdots, M-1]^{\mathsf{T}}$, $\mathbf{u}_{M^G} = [-1, -1 + \frac{2}{M^G}, -1 + \frac{4}{M^G}, \cdots, 1 - \frac{2}{M^G}]^{\mathsf{T}}$ and d is the antenna spacing which is assumed to be $1/2$ in the current version.

For a uniform planar array (UPA) with size $M = M_x M_y$, the dictionary is defined as

$$\mathbf{V}_M = \mathbf{V}_{M_x} \otimes \mathbf{V}_{M_y}, \tag{6.3}$$

where $\otimes$ denotes the Kronecker product, and $\mathbf{V}_{M_x}$ and $\mathbf{V}_{M_y}$ can both be calculated with (6.2).

[1] The table is so similar to that of C++. Well, indeed, and it is adapted from C++ Reference.

ALG Implementation The return value is the generated dictionary matrix which has type `c2c` (const complex matrix). The parameters are shown in Table 6.10.

Table 6.10: CALC `\dictionary` parameters.

Position	Meaning	Description
1	M_x	ULA size or UPA x dimension size.
2	M_y	1 for ULA and y dimension size for UPA.
3	M_x^G	ULA grid size or UPA x dimension grid size.
4	M_y^G	1 for ULA and y dimension grid size for UPA.

Example This is how you can use `\dictionary` in ALG.

Example 6.15 (Example of \dictionary)

```
# UPA Antenna size: 16x6, Grid size: 16x16
D = \dictionary(16, 8, 16, 16)
# This creates a new instance of dictionary matrix
D::mc = NEW \dictionary(16, 8, 16, 16)
# Use macro to create a dictionary at the transmitter side
D::mc = NEW `DICTIONARY.T`
```

Tip

For simplicity, you may use macro to simplicity the `\dictionary` command. The `` `DICTIONARY.T` `` and `` `DICTIONARY.R` `` macros can be used to represent the dictionary at the transmitter side and receiver side without specifying the parameters as long as they are specified in the `nodes` section of `.sim` configuration.

6.3.5 Subscripts

Subscripts take the subview of vector/matrix/tensor. The syntax is `_{<sub>}` (a leading underscore `_` as in LaTeX), where the brackets cannot be omitted even if `<sub>` contains only one character.

! The subscript syntax imposes a requirement for variables: variables cannot be ended with character underscore (_).
Due to internal underscore is allowed for variable names (see §6.3.2), the subscript is only recognized when brackets {} exist.

6.3.5.1 Foo Subscripts

Well, some subscripts just do nothing, so they are removed. They are `_{}`, `_{:}`, `_{:,:}`, `_{:,:,:}`.

6.3.5.2 Valid Subscripts

The contents inside `_{}` of valid subscripts are similar to MATLAB syntax except for indices start from 0.

- For a vector `x`, use `x_{n}` to access the n-th element, and `x_{begin:end}` to access a range of elements.
- For a matrix `A`, use `A_{i,j}` to access an individual element, `A_{i,:}` to access an entire row, `A_{:,j}` to access an entire column, `A_{i:j,l:m}` to access a range of rows and columns, and `A_{:,:,k}` to access a slice of the matrix.

- For a tensor `Z`, use `Z_{i,j,k}` to access an individual element, `Z_{:,:,k}` to access a slice of the tensor, and `Z_{:,:,indices}` to access multiple slices where `indices` is of type `u1`. Accessing indices vector of type `u1` for a tensor is only supported for the last dimension.

Here are some examples:

Example 6.16 (Example of Subscripts)

```
# x is a vector (dim = 1)
# A is a matrix (dim = 2)
# Z is a tensor (dim = 3)
x_{2} # [scalar] the 2-nd element of vector
x_{1:4} # [vector] elements 1,2,3,4 of vectors
x_{index} # [scalar] (index of type u0) the index-element of vector
x_{indices} # [vector] (indices of type u1) elements of indices in vector
A_{3,2} # [scalar] the element at position (3,2)
A_{2,index} # [scalar] (index of type u0)
A_{2,:} # [rowvec] the 2-nd row
A_{:,3} # [vector] the 3-nd column
A_{1,3:5} # [vector] elements (1,3),(1,4),(1,5)
A_{2:4,indices} # [matrix] (indices of type u1)
A_{15} # [scalar] the 15-th element in the flattened matrix as vector
A^T_{1,2} # [scalar] element (1,2) of A^T, i.e. element (2,1) of A
A_{1:3,:}^H # [scalar] the conjugate transpose of sub matrix (Mind the
 ↪ sequence!)
Z_{0,0,0} # [scalar] element (0,0,0) of tensor
Z_{:,:,2} # [matrix] the 2-nd slice of tensor
Z_{:,:,indices} # [tensor] (indices of type u1) slices of tensor
```

> **!** Using indices vector of type `u1` for a tensor is only supported for the last dimension (i.e. the slice).

6.3.6 Superscripts

Supper scripts are led by caret (`^`) as in LaTeX.

6.3.6.1 Transpose and Conjugate Transpose

Real Matrices The transpose for a real matrix is `^t` or `^{t}`.

Complex Matrices The transpose for a complex matrix is `^T` or `^{T}`, and the conjugate transpose (Hermitian transpose) for a complex matrix is `^H` or `^{H}`.

> **!** There can be C++ compiling error if `^T` and `^t` are misused!

6.3.6.2 Conjugate

The conjugate of a complex number/vector/matrix/tensor is `^*`, `^{*}`, `^\star`, `^{\star}`, `^\ast` or `^{\ast}`. (Wow, so TeXy!!!)

> **!** You should **not** calculate the conjugate for a real number/vector/matrix/tensor.

6.3.6.3 Inverse

The inverse of a square matrix is `^{-1}`, `^i` or `^I`.
For a non-square matrix, you may use `\pinv` (pseudo-inverse) command.

6.4 Macro

For simulation convenience, there are macros defined above the level of ALG language. That is, some configuration variables may be used in ALG.

6.5 ALG Library

6.5.1 Basics

ALG Library is a collection of standard ALG functions that can be safely `CALL`ed. Moreover, in function `ESTIMATE`, predefined compressed sensing algorithms are called which are also in the ALG library.

Like MATLAB, the inside function name should be the same as the file name except for the `.alg` file extension. For example, OMP function is defined in file `/include/mmcesim/OMP.alg`.

6.5.2 Algorithms

6.5.2.1 OMP

Orthogonal matching pursuit. (`OMP.alg`)

6.5.2.2 OMPL

OMP List. (`OMPL.alg`)

This algorithm is proposed in *OMPL-SBL Algorithm for Intelligent Reflecting Surface-Aided mmWave Channel Estimation* (TVT'23) [2], extending OMP with the k-best idea.

6.5.2.3 Oracle LS

Oracle least square (LS). (`Oracle_LS.alg`)

6.5.2.4 LS Support

Least square (LS) with specified support. (`LS_Support.alg`)

6.5.3 Other Utilities

6.5.3.1 max_n

Obtain the largest n elements from a vector. (`max_n.alg`)

This page is intentionally left blank

III

TUTORIALS

Step-by-step guide on using mmCEsim.

This page is intentionally left blank

Millimeter Wave Channel Estimation

Millimeter wave channel estimation for multiple-input multiple-output (MIMO) systems techniques are discussed in [3].

A novel channel estimation algorithm *orthogonal matching pursuit list-sparse Bayesian learning* (OMPL-SBL) is proposed in [2], which also reviews the compressed channel estimation scheme for RIS-aided MIMO systems. The structure of angular domain sparsity is also discussed.

The employed automatic beamforming scheme is proposed in [4].

This page is intentionally left blank

8 CLI Application Tutorials

The tutorials are being developed. Please refer to §3 for now.

This page is intentionally left blank

9 GUI Application Tutorials

The tutorials are being developed. Please refer to §4 for now.

This page is intentionally left blank

10 Web Application Tutorials

The tutorials are being developed. Please refer to §5 for now.

This page is intentionally left blank

11 VS Code Extension Tutorials

11.1 Installation

The extension `mmCEsim` is published at the VS Code Marketplace, and you can view it at https://marketplace.visualstudio.com/items?itemName=mmcesim.mmcesim.

11.2 Features

Currently, there is syntax highlight support for `.sim` and `.alg`, and the YAML schema is also provided for the `.sim` configuration.

This page is intentionally left blank

APPENDIX

Additional information about mmCEsim.

This page is intentionally left blank

Additional Resources

A.1 Publications

A brief introduction of mmCEsim is given in the poster at the 2022 National Postdoc Seminar in Nanjing, which I attend as the only undergraduate student, and got the Honorable Mention award.

This document is also published online at https://pub.mmcesim.org/mmCEsim-doc.pdf.

Related research papers: [2], [4].

A.2 Websites

A.2.1 For Users

If you are the user of mmCEsim and wants to know more, you may find the following websites in Table A.1 useful.

Table A.1: Websites for users.

Website	URL
Homepage	https://mmcesim.org
Web Application	https://app.mmcesim.org
Blog	https://blog.mmcesim.org
Publications	https://pub.mmcesim.org
VS Code Extension	https://marketplace.visualstudio.com/items?itemName=mmcesim.mmcesim

A.2.2 For Developers

If you are a developer and maybe want to contribute to the mmCEsim project, you can find additional websites in Table A.2.

Table A.2: Websites for developers.

Website	URL
GitHub Organization	https://github.com/mmcesim
C++ Dev Documentation	https://dev.mmcesim.org
CLI App Wiki	https://github.com/mmcesim/mmcesim/wiki

A.3 Author

Wuqiong Zhao (*Student Member, IEEE*) is an undergraduate student pursuing the Bachelor's Degree in communications engineering, working at Lab of Efficient Architectures for Digital-communication and Signal-processing (LEADS) and National Mobile Communications Research Laboratory, Southeast University. He is the honors student of Chien-Shiung Wu College and earned the National Scholarship and Cyrus Tang Scholarship in 2021. From 2020 to 2021, he also served as the Special Student Assistant to President of Southeast University. He was also nominated as the most influential undergraduate student of Southeast University in 2022. His research interest includes channel estimation, Bayesian algorithms, and the intelligent reflecting surface (IRS) in wireless communication of 5G and 6G. He is also the reviewer of IEEE TCAS II and ISCAS 2023.

Change History B

B.1 v0.2.2 2024/01/12

New Features

- ALG library enhancement (#49, PR #51):
  - Add OMPL [2] algorithm to the library.
  - ALG dependency check.
  - Enhanced log and TEX report (PR #55).
- Update Doxygen website configurations (including more graphs).

Bug Fixes

- Rename configuration file to `mmcesim.cfg` (#50, PR #53).
- Fix C++ (Armadillo) runtime error when simulating with `-O2` or `-O3` on macOS by patching `arma::randperm` (#52, PR #54).

News

- The research paper entitled 'Beam Pattern and Reflection Pattern Design for Channel Estimation in RIS-Assisted Mmwave MIMO Systems' [4] is online as an early access paper of IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY (Sep. 8, 2023): https://ieeexplore.ieee.org/document/10243635 (#79).
- The research paper entitled 'Beam Pattern and Reflection Pattern Design for Channel Estimation in RIS-assisted mmWave MIMO Systems' [4] has been accepted by IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY (May 21, 2023);
- The research paper entitled 'OMPL-SBL Algorithm for Intelligent Reflecting Surface-Aided mmWave Channel Estimation' [2] is online as an early access paper of IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY (Jun. 27, 2023): https://ieeexplore.ieee.org/document/10164645 (#69).

B.2 v0.2.1 2023/03/31

New Features

- Colorful terminal (#22).
- `mmcesim-maintain` tool support (#25).
- Log system support with cleaner terminal output (#32, PR #36).

- `mmcesim-log` tool support (#37, PR #39, PR #47).
- Support `NO_COLOR` standard (#42, PR #48).
- Syntax highlight for ALG language on mmcesim.org with a custom lexer (`mmcesim.org#1`).

Bug Fixes

- Fix the Docker `entrypoint` command.
- Fix *total lines* badge display in README (#40, PR #41).
- Fix Ubuntu 20 release error (#46).

News

- Short domain mmces.im has been registered alongside mmcesim.org (#43).

B.3 v0.2.0 2023/01/20

New Features

- Active beam pattern design support (#18).
- Report generation of RIS-assisted systems (#3).

Bug Fixes

- Fix inconsistency of LaTeX report destination directory.

News

- The research paper entitled 'OMPL-SBL Algorithm for Intelligent Reflecting Surface-Aided mmWave Channel Estimation' [2] has been accepted by IEEE Transactions on Vehicular Technology (Jan. 13, 2023).

B.4 v0.1.1 2023/01/11

New Features

- Multi RIS assisted systems support (#3);
- RIS pattern design support (#17).

Bug Fixes

- Fix `cmake install` configurations.

News

- Automated release process with a better CI workflow (#20).

B.5 v0.1.0 2022/10/16

New Features

- Basic mmWave MIMO systems channel estimation support;
- Design of ALG language;
- Export of code with Armadillo library;
- Auto simulation (#5).

B.6 v0.0.1 2022/07/27

Though the app has not been fully developed, the task-oriented concept has already been established.

Bibliography

[1] Q. Wu and R. Zhang, "Towards smart and reconfigurable environment: Intelligent reflecting surface aided wireless network", *IEEE Commun. Mag.*, vol. 58, no. 1, pp. 106–112, Jan. 2020.

[2] W. Zhao, Y. You, L. Zhang, X. You, and C. Zhang, "OMPL-SBL algorithm for intelligent reflecting surface-aided mmWave channel estimation", *IEEE Trans. Veh. Technol.*, vol. 72, no. 11, pp. 15 121–15 126, Nov. 2023.

[3] J. Lee, G.-T. Gil, and Y. H. Lee, "Channel estimation via orthogonal matching pursuit for hybrid MIMO systems in millimeter wave communications", *IEEE Trans. Commun.*, vol. 64, no. 6, pp. 2370–2386, Jun. 2016.

[4] Y. You, W. Zhao, L. Zhang, X. You, and C. Zhang, "Beam pattern and reflection pattern design for channel estimation in RIS-assisted mmWave MIMO systems", *IEEE Trans. Veh. Technol.*, pp. 1–6, 2023, to be published. DOI: TVT.2023.3309950.

This page is intentionally left blank

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

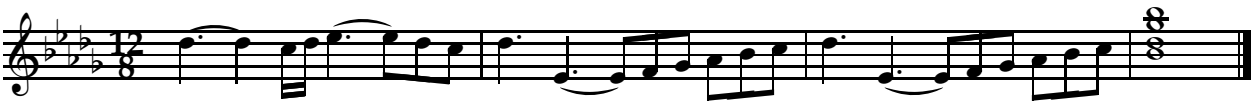
U

V

W

Y

Z



Finale of Götterdämmerung by Richard Wagner. Step into the mythical world of gods and heroes with our millimeter wave channel estimation simulation software.