mmCEsim

https://mmcesim.org

mmCEsim Documentation & Tutorials

Task-oriented mmWave Channel Estimation Simulation

Version 0.1.1

Wuqiong Zhao (Teddy van Jerry)

January 11, 2023

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

Edition 2023/01/11 (corresponding to mmCEsim version 0.1.1).

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

Contents

III
TUTORIALS

APPENDIX

Preface

mmCEsim documentation & tutorials are under development!

I would like to thank Jinwen Xu for designing the elegant LaTeX template `beaulivre`, which empowers this document.

Wuqiong Zhao
Nanjing, China
January 2023

List of Figures

List of Tables

I

PRELIMINARY

Make preparations before we start.

1 Preview

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.

Installation 2

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
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 Formatter (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`).

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>`.

II

DOCUMENTATION

Every syntax and option in details.

CLI Application 3

GUI Application 4

5 Web Application

The example web app page is shown in Fig. 5.1.

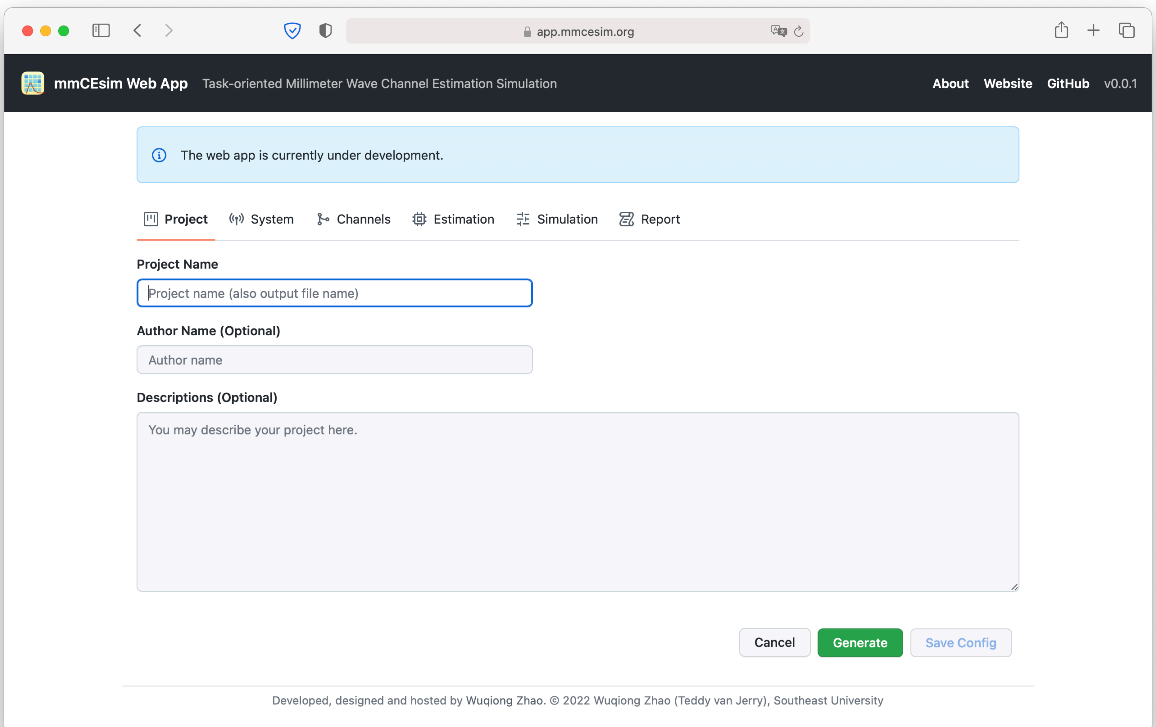

Figure 5.1: Web app interface.

ALG Language 6

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 (`c`) 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.

```
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 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`.

6.2.9 END

End of a block for `ELSE`, `ELIF`, `FUNC`, `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`.

6.2.10 ESTIMATE

`CALL` standard ALG functions to estimate the sparse channel with compressed sensing (CS).

6.2.11 FUNCTION

Start a function definition.

Explanations

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

6.2.12 IF

Conditional statement.

6.3 Calculation (CALC)

6.4 Macro

6.5 ALG Library

III TUTORIALS

Step-by-step guide on using mmCEsim.

This page is intentionally left blank

7 Millimeter Wave Channel Estimation

Millimeter wave channel estimation for multiple input multiple output (MIMO) systems techniques are discussed in [2].

8 CLI Application Tutorials

This page is intentionally left blank

GUI Application Tutorials 9

Web Application Tutorials 10

This page is intentionally left blank

VS Code Extension Tutorials 11

APPENDIX

Additional information about mmCEsim.

Additional Resources A

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`.

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 (number one) 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 assisted editing the book *Channel Codes for 5G Wireless Systems* and the chapter *Stochastic Computation for Baseband Processing*.

B Change History

B.1 HEAD 2023/01/11

B.2 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.3 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.4 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] 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.

This page is intentionally left blank

Index

R

S

T

U

V