# Module Interface Specification for Optimal EM Placement

Hussein Saad

April 16, 2025

# 1 Revision History

| Date | Version | Notes |
|---|---|---|
| April 16, 2025 | 1.1 | Implement domain expert suggestions |
| March 20, 2025 | 1.0 | Initial Release |

# 2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/husseinsd1/optimal-em-arrangement/blob/main/docs/SRS/SRS.pdf

# Contents

# 3 Introduction

The following document details the Module Interface Specifications for OEMP (Optimal Electromagnet Placement). This document describes, in detail, how the interfaces, assumptions and interactions among the modules of the program.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/husseinsd1/optimal-em-arrangement.

# 4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Optimal EM Placement.

| Data Type | Notation | Description |
|---|---|---|
| character | char | a single symbol or digit |
| integer | $\mathbb{Z}$ | a number without a fractional component in (-$\infty$, $\infty$) |
| natural number | $\mathbb{N}$ | a number without a fractional component in [1, $\infty$) |
| real | $\mathbb{R}$ | any number in (-$\infty$, $\infty$) |

The specification of Optimal EM Placement uses some derived data types: sequences, strings, tuples, and vectors. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. An $n$-dimensional vector is a list of $n$ real numbers. In addition, Optimal EM Placement uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

# 5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

| **Level 1** | **Level 2** |
|---|---|
| Hardware-Hiding Module | |
| Behaviour-Hiding Module | Constant Parameters Module<br>Input Parameters Module<br>Magnetic Field Module<br>Magnetic Force Module<br>Actuation Matrix Module<br>Output Results Module<br>Main (Control) Module |
| Software Decision Module | Optimal Placement Module |

Table 1: Module Hierarchy

# 6 MIS of Constant Parameters Module

## 6.1 Module

ConstantParams

## 6.2 Uses

None

## 6.3 Syntax

### 6.3.1 Exported Constants

| Label | Symbol | Value | Description |
|---|---|---|---|
| MU0 | $\mu_0$ | $4\pi \times 10^{-7}$ | Permeability of free space |

### 6.3.2 Exported Access Programs

None

## 6.4 Semantics

### 6.4.1 State Variables

None

### 6.4.2 Environment Variables

None

### 6.4.3 Assumptions

Constant values are assumed immutable.

### 6.4.4 Access Routine Semantics

None

### 6.4.5 Local Functions

None

# 7 MIS of Input Parameters Module

## 7.1 Module

params

## 7.2 Uses

- Hardware-Hiding Module

## 7.3 Syntax

### 7.3.1 Exported Constants

None

### 7.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| `takeInputs` | - | params of type `Params` | `negativeValueError`, `invalidCurrentError` |

## 7.4 Semantics

`Params` is a data structure used to store the parameter values the user enters into the program.

### 7.4.1 State Variables

params : `Params` = [
$N : \mathbb{N}$,
$I : \mathbb{R}$,
$A : \mathbb{R}$,
$M : \mathbb{N}$,
$K : \mathbb{N}$,
$V : \mathbb{R}$,
$t : \mathbb{R}$,
$m_t : \mathbb{R}$
]
The description of the elements of the above array is found in Section 1.2 of the SRS.

### 7.4.2 Environment Variables

- A console: The medium through which the user will enter the parameter values.

- A keyboard: The module takes input from the user's keyboard.

#### 7.4.3 Assumptions

None

#### 7.4.4 Access Routine Semantics

takeInputs():

- transition:
  - Initialize params : `Params`.
  - Display prompt for user to enter the value of the parameters, one by one, and raise and exception and re-prompt if the entered value is not in line with the constraints outline in Table 2 of the SRS.
- output: params : `Params`
- exception: exc =

| Exception | When |
|---|---|
| `negativeValueError` | When the input for any of the parameters is negative. |
| `invalidCurrentError` | When the input for the current $I$ is greater than 20000 |

#### 7.4.5 Local Functions

None

# 8 MIS of Magnetic Field Module

## 8.1 Module

MagField

## 8.2 Uses

- Constant Parameters Module
- Input Parameters Module

## 8.3 Syntax

### 8.3.1 Exported Constants

None

### 8.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| calculateField | params of type `Params` | vector in $\mathbb{R}^3$ | None |

## 8.4 Semantics

### 8.4.1 State Variables

None

### 8.4.2 Environment Variables

None

### 8.4.3 Assumptions

Assumes the execution of the Input Parameters Module prior to running.

### 8.4.4 Access Routine Semantics

calculateField(params : `Params`):

- transition:
  - Extract necessary parameters $(N, I, A, t)$ from params.
  - Calculate and store magnetic moment using the calculateMoment() local function.

- Find the magnetic field with the given parameters, the calculated moment value, and $\mu_0$ from the Constant Parameters Module using the equation defined in TM2 of the SRS.

- output: This module outputs a real 3D vector describing the magnetic field at some distance $t$ (retrieved from params).

- exception: N/A

#### 8.4.5 Local Functions

**calculateMoment(N : $\mathbb{N}$, I : $\mathbb{R}$, A : $\mathbb{R}$, t : $\mathbb{R}$):** A function to compute the magnetic moment at some distance $t$ as specified in TM1 of the SRS.

- output: A magnetic moment vector in $\mathbb{R}^3$

- exception: None

# 9 MIS of Magnetic Force Module

## 9.1 Module

MagForce

## 9.2 Uses

- Input Parameters Module
- Magnetic Field Module

## 9.3 Syntax

### 9.3.1 Exported Constants

None

### 9.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| calculateForce | params of type `Params` | vector in $\mathbb{R}^3$ | None |

## 9.4 Semantics

### 9.4.1 State Variables

None

### 9.4.2 Environment Variables

None

### 9.4.3 Assumptions

None

### 9.4.4 Access Routine Semantics

calculateForce(params : `Params`):

- transition:
  - Invoke and store the magnetic field returned by the Magnetic Field Module.
  - Extract the magnetic moment of the target object from params.

- Compute the force vector as described in TM3 of the SRS

- output: A real 3D vector describing the magnetic force on some target.
- exception: N/A.

#### 9.4.5 Local Functions

None

# 10 MIS of Actuation Matrix Module

## 10.1 Module

ActuationMatrix

## 10.2 Uses

- Input Parameters Module
- Magnetic Field Module
- Magnetic Force Module

## 10.3 Syntax

### 10.3.1 Exported Constants

None

### 10.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| constructMatrix | params of type `Params` | in $\mathbb{R}^6$ | None |

## 10.4 Semantics

### 10.4.1 State Variables

None

### 10.4.2 Environment Variables

None

### 10.4.3 Assumptions

None

### 10.4.4 Access Routine Semantics

constructMatrix(params : `Params`):

- transition:
  - Extract parameters from the params argument.

- Generate a set of random positions (of size $M$ from params) using the local function generatePos.
- For each candidate position, calculate the magnetic force and field vectors using Magnetic Force Module and Magnetic Field Module, respectively.
- Sum up the force and field vectors of the candidate positions, the result is two 3D vectors.
- Concatenate the two vectors such that a $6 \times 1$ matrix is formed.

- output: A $6 \times 1$ real matrix.
- exception: N/A

#### 10.4.5 Local Functions

**generatePos(M :** $\mathbb{N}$): A function to generate random candidate positions for the EMs.

- output: A set of $M$ [x, y, z] coordinates.
- exception: None

# 11 MIS of Optimal Placement Module

## 11.1 Module

FindOptPositions

## 11.2 Uses

- Actuation Matrix Module
- Input Parameters Module

## 11.3 Syntax

### 11.3.1 Exported Constants

None

### 11.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| solve | vector in $\mathbb{R}^6$ , params of type `Params` | binary vector in $\mathbb{R}^M$ | SolverException |

## 11.4 Semantics

### 11.4.1 State Variables

None

### 11.4.2 Environment Variables

None

### 11.4.3 Assumptions

None

### 11.4.4 Access Routine Semantics

solve($\mathcal{U}$ : $\mathbb{R}^6$, params : `Params`):

- transition:
  - Compute and store $\mathcal{U}\mathcal{U}^\top$
  - Extract $M$ and $K$ from params.

  - Pass $\mathcal{U}\mathcal{U}^\top$, $M$ and $K$ into a `cvxpy` solver.

- output: A vector $x \in \{0,1\}^M$ such that:
  - $\mathbb{1}_M^\top x = K$ ($\mathbb{1}$ is a ones vector).
  - $\lambda_{\min}$ of $\sum_{i=1}^K x_i \mathcal{U}_i \mathcal{U}_i^\top$ is maximized.
- exception: Any exceptions raised by the solver.

#### 11.4.5 Local Functions

None

# 12 MIS of Output Results Module

## 12.1 Module

OutputResults

## 12.2 Uses

- Hardware-Hiding Module
- Optimal Placement Module

## 12.3 Syntax

### 12.3.1 Exported Constants

None

### 12.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| output | $x \in \{0,1\}^M$ | - | None |

## 12.4 Semantics

### 12.4.1 State Variables

None

### 12.4.2 Environment Variables

Console: this module prints the vector $x$ onto the console for the user to see.

### 12.4.3 Assumptions

None

### 12.4.4 Access Routine Semantics

output($x : \{0,1\}^M$):

- transition: Prints the given vector onto the console.
- output: N/A
- exception: None

### 12.4.5 Local Functions

None

# 13 MIS of Main (Control) Module

## 13.1 Module

main

## 13.2 Uses

- Hardware-Hiding Module
- Constant Parameter Module
- Input Parameters Module
- Magnetic Field Module
- Magnetic Force Module
- Actuation Matrix Module
- Optimal Placement Module
- Output Results Module

## 13.3 Syntax

### 13.3.1 Exported Constants

None

### 13.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| main | - | - | Various |

## 13.4 Semantics

### 13.4.1 State Variables

- params : `Params`
- $\mathcal{U}$ : $\mathbb{R}^6$

### 13.4.2 Environment Variables

None

### 13.4.3 Assumptions

None

### 13.4.4 Access Routine Semantics

main():

- transition:
  - Call and store params from Input Parameters Module.
  - Invoke the Actuation Matrix Module and store the returned $\mathcal{U}$ vector (Actuation Matrix will itself invoke the modules responsible for the magnetic field/force and constant parameters).
  - Provide the $\mathcal{U}$ vector and params to Optimal Placement Module, and store the returned $x$ vector.
  - Pass the returned $x$ vector to Output Results Module.
- output: N/A
- exception: Exceptions arising from submodules.

### 13.4.5 Local Functions

None

# References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.