# Module Interface Specification for Inverted Pendulum Control Systems

Morteza Mirzaei

March 18, 2024

## 1 Revision History

| Date | Version | Notes |
|---|---|---|
| 2024-03-18 | 1.0 | Initial Release. |

## 2 Symbols, Abbreviations and Acronyms

See SRS Documentation at:
https://github.com/mirzaim/ipcs/blob/main/docs/SRS/SRS.pdf

# Contents

## 3 Introduction

The following document details the Module Interface Specifications for Inverted Pendulum Control Systems.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/mirzaim/ipcs.

## 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 Inverted Pendulum Control Systems.

| **Data Type** | **Notation** | **Description** |
|---|---|---|
| character | char | a single symbol or digit |
| string | char$^*$ | a sequence of characters |
| 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 (-$\infty$, $\infty$) |
| linguistic variable | ling_var | a linguistic variable that used in the fuzzy logic rules. Its type is string. |
| rule | rule | a rule that used in the fuzzy logic. Its type is string. |
| fuzzy set | fuzzy_set | a fuzzy set that is determined by the membership function $(R \rightarrow R)$. |

The specification of Inverted Pendulum Control Systems uses some derived data types: sequences, strings, and tuples. 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. In addition, Inverted Pendulum Control Systems 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 | Fuzzy Logic Module<br>Parser Module<br>Output Verification Module<br>Simulator Module<br>World Module |
| Software Decision Module | Controller Module<br>GUI Module<br>Sequence Data Structure Module |

Table 1: Module Hierarchy

# 6 MIS of Fuzzy Logic Module

## 6.1 Fuzzy Logic

In this section, we provide some background information on fuzzy logic and fuzzy controllers that will be used in the Inverted Pendulum Control Systems project.

### 6.1.1 Fuzzy Sets

In the realm of control systems, traditional methods often struggle to handle the inherent uncertainty and imprecision present in real-world data. Fuzzy logic offers a solution by introducing the concept of fuzzy sets. Unlike crisp sets, where elements are either fully inside or outside a set, fuzzy sets allow for partial membership. This means that an element can belong to a set to a certain degree, reflecting the uncertainty or vagueness inherent in many real-world scenarios. A fuzzycontroller is a control system based on fuzzy logic, which uses fuzzy sets to represent linguistic variables and rules to make decisions. It consits of three main steps: fuzzification, rule processing, and defuzzification that are described below.

### 6.1.2 Fuzzy Controller Steps

1. **Fuzzification:** Fuzzification is the initial step in a fuzzy controller where crisp inputs are transformed into fuzzy sets. This process involves mapping numerical inputs to linguistic variables and assigning membership degrees to each linguistic variable's corresponding fuzzy set. For instance, in a temperature control system, a crisp input value like "25 degrees Celsius" might be fuzzified into linguistic variables such as "cold," "warm," and "hot," with each linguistic variable having membership degrees reflecting the degree to which the input belongs to each fuzzy set.

2. **Rule Processing:** Once the inputs are fuzzified, fuzzy rules are applied to determine the appropriate control actions. These rules, often expressed in the form of IF-THEN statements, map combinations of fuzzy input variables to fuzzy output variables. For example, a rule might state "IF temperature is cold AND humidity is high THEN heater is high." By evaluating these rules and considering the fuzzy input values, the fuzzy controller determines the appropriate control action to take.

3. **Defuzzification:** After applying fuzzy rules and obtaining fuzzy output sets, the final step is defuzzification. Here, the fuzzy output sets are aggregated to produce a crisp output value that represents the control action to be taken. Various defuzzification methods exist, such as centroid defuzzification or maximum membership principle. These methods translate the fuzzy output sets back into a single numerical value that guides the system's response.

In conclusion, the fuzzy controller method offers a flexible and effective approach to control systems, allowing for the incorporation of human-like reasoning and handling uncertainty inherent in real-world environments.

## 6.2 Module

Fuzzy Logic

## 6.3 Uses

parser module (Section 7)

## 6.4 Syntax

### 6.4.1 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| decide | World | $\mathbb{R}$ | - |
| load_fuzzy_rules | fuzzy_sets: *Array*[ling_var $\times$ fuzzy_set], rules: *Array*[rule] | - | - |

## 6.5 Semantics

### 6.5.1 State Variables

fuzzyInputSets: *Array*[ling_var $\times$ fuzzy_set]
fuzzyOutputSets: *Array*[ling_var $\times$ fuzzy_set]
fuzzyRules: *Array*[rule]

### 6.5.2 Environment Variables

None

### 6.5.3 Assumptions

None

### 6.5.4 Access Routine Semantics

decide(World):

- transition: -
- output: returns the force value.

  1. In the first step, it fuzzifies the world state $x$, $\theta$, $\dot{x}$, and $\dot{\theta}$ to get the fuzzy input sets. This step uses fuzzyInputSets environment variable and measures the membership degree of each linguistic variable's corresponding fuzzy set.

2. In the second step, it applies the fuzzy rules on fuzzified input sets to get the fuzzy output sets. This step uses fuzzyRules environment variable. At the end of this step we have membership degrees of each linguistic variable's corresponding fuzzyOutputSets.
3. In the third step, it defuzzifies the fuzzy output sets to get the force value. In this step we use center of mass method as defuzzification method.

- exception: -

load_fuzzy_rules(fuzzy_sets, rules):

- transition: It sets the fuzzyInputSets, fuzzyOutputSets and fuzzyRules state variables.
- output: -
- exception: -

# 7 MIS of Parser Module

## 7.1 Module

Parser

## 7.2 Uses

None

## 7.3 Syntax

### 7.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| load_config | file_path: char* | World | FileNotFoundException, Invalid-FileFormatException |
| load_Fuzzy_logic | file_path: chat* | fuzzy_sets: *Array*[ling_var × fuzzy_set], rules: *Array*[rule] | FileNotFoundException, Invalid-FileFormatException |

## 7.4 Semantics

### 7.4.1 State Variables

None

### 7.4.2 Environment Variables

file: A text file.

### 7.4.3 Assumptions

None

### 7.4.4 Access Routine Semantics

load_config(file_path):

- transition: -
- output: Read initial world configuration from the file and returns a World.

- exception: $exc :=$ (
  if file_path does not exist then FileNotFoundException
  |
  if file_path is not in the correct format then InvalidFileFormatException
  )

load_Fuzzy_logic(file_path):

- transition: -
- output: It acts like a complier and reads the file and returns the fuzzy sets and rules.
  returns:
  fuzzy_sets: Array[LinguisticVariable $\times$ FuzzySet], rules: fuzzyRules: Array[Rule]
- exception: $exc :=$ (
  if file_path does not exist then FileNotFoundException
  |
  if file_path is not in the correct format then InvalidFileFormatException
  )

# 8 MIS of Output Verification Module

## 8.1 Module

Output Verification

## 8.2 Uses

World Module (Section 10)

## 8.3 Syntax

### 8.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| verify | world | - | InvalidAngleException, InvalidPositionException |

## 8.4 Semantics

### 8.4.1 State Variables

None

### 8.4.2 Environment Variables

None

### 8.4.3 Assumptions

None

### 8.4.4 Access Routine Semantics

verify(world):

- transition: -
- output: -
- exception: $exc :=$ (
  if $\theta \leq -\pi$ or $\theta \geq \pi$ then InvalidAngleException
  |
  if $x \leq$ min_x or $x \geq$ max_x then InvalidPositionException
  )

## 9 MIS of Simulator Module

### 9.1 Module

Simulator

### 9.2 Uses

World Module (Section 10)

### 9.3 Syntax

#### 9.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| tick | R | - | - |

### 9.4 Semantics

#### 9.4.1 State Variables

world: World

#### 9.4.2 Environment Variables

None

#### 9.4.3 Assumptions

None

#### 9.4.4 Access Routine Semantics

tick(dt):

- transition: It updates the world state by one time step of size $dt$. It follows the following steps:
  - Caclulate and update the acceleration of the cart and the pendulum.
  - Update teh velocity of the cart and the pendulum.
  - Update the position of the cart and the pendulum.
  - Reset the forces acting on the cart and the pendulum.

It uses the two following equations to update the world state:

$$F(t) = (M+m)\ddot{x}(t) - ml\ddot{\theta}(t)\cos\theta(t) + ml(\dot{\theta}^2(t))\sin\theta(t)$$
$$l\ddot{\theta}(t) + g\sin\theta(t) = \ddot{x}(t)\cos\theta(t)$$

- output: -
- exception: -

# 10 MIS of World Module

## 10.1 Module

World

## 10.2 Uses

Parser Module (Section 7)

## 10.3 Syntax

### 10.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| - | - | - | - |

## 10.4 Semantics

### 10.4.1 State Variables

m: $\mathbb{R}$
M: $\mathbb{R}$
l: $\mathbb{R}$
g: $\mathbb{R}$
x: $\mathbb{R}$
$\dot{x}$: $\mathbb{R}$
$\ddot{x}$: $\mathbb{R}$
$\theta$: $\mathbb{R}$
$\dot{\theta}$: $\mathbb{R}$
$\ddot{\theta}$: $\mathbb{R}$
F: $\mathbb{R}$

### 10.4.2 Environment Variables

None

### 10.4.3 Assumptions

None

### 10.4.4 Access Routine Semantics

None

# 11 MIS of Controller Module

## 11.1 Module

## 11.2 Uses

Fuzzy Logic Module (Section 6)

## 11.3 Syntax

### 11.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| decide | World | $\mathbb{R}$ | - |

## 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

Because this module is interface so access routines doesn't have exact procedure.
decide(world):

- transition: -
- output: Based on the current world situation, it should return proper force value to keep the pendulum up.
- exception: -

# 12 MIS of GUI Module

## 12.1 Module

GUI

## 12.2 Uses

World Module (Section 10)

## 12.3 Syntax

### 12.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| draw | World | - | - |

## 12.4 Semantics

### 12.4.1 State Variables

None

### 12.4.2 Environment Variables

win: 2D diagram displayed on the screen

### 12.4.3 Assumptions

None

### 12.4.4 Access Routine Semantics

draw(world):

- transition: Modify the win to display the current world state. The location and angle of pendulum should be clear.
- output: -
- exception: -

## 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.

## 13 Appendix

## 14 Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design. Please answer the following questions:

1. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)
2. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)