Module Interface Specification for Inverted Pendulum Control Systems (IPCS)

Morteza Mirzaei

April 15, 2024

1 Revision History

Date	Version	Notes
2024-03-18	1.0	Initial Release.
2024-04-14	2.0	Address comments.

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
boolean	$\mathbb{B}$	a binary variable that can be either true or false
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
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 ($\mathbb{R} \to \mathbb{R}$).
array	Array[T]	a sequence of elements of type T.
dictionary	Dict[K, V]	a collection of key-value pairs, where K is the key type and V is the value type.

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 Config Reader Module World Checker Module Simulator Module World Module Manager Module Main Module
Software Decision Module	Controller Module GUI Module File Output 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

World module (Section 10)

6.4 Syntax

6.4.1 Exported Access Programs

Name	In	Out	Exceptions
decide	World	$\mathbb{R}$	-
read_fuzzy_rules	file_path: char*	Array[rule]	-

6.5 Semantics

6.5.1 State Variables

fuzzyInputSets: *Dict*[ling_var, fuzzy_set]
fuzzyOutputSets: *Dict*[ling_var, fuzzy_set]
fuzzyRules: *Array*[rule]
(The fuzzyInputSets, fuzzyOutputSets are hardcoded in the module, but the fuzzyRules are read from a file.)

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, θ, $\dot{x}$, 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 state 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: -

read_fuzzy_logic(file_path):

- transition: -
- output: It acts like a compiler and reads the file and. returns: fuzzyRules Array[rule]
- exception: -

7 MIS of Config Reader Module

7.1 Module

Config Reader

7.2 Uses

None

7.3 Syntax

7.3.1 Exported Access Programs

Name	In	Out	Exceptions
load_config	file_path: char*	World	FileNotFoundException

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
)

8 MIS of World Checker Module

8.1 Module

World Checker

8.2 Uses

World Module (Section 10)

8.3 Syntax

8.3.1 Exported Access Programs

Name	In	Out	Exceptions
check	World	World	ValueError

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

check(world):

- transition: -
- output: world
- exception: $exc :=$ (
  if not (all parameters in world is float) then ValueError
  |
  if not $(0 \leq m \leq 80)$ then ValueError
  |
  if not $(0 \leq M \leq 80)$ then ValueError
  |

if not $(0 \leq l \leq 20)$ then ValueError
|
if (min_x $\leq$ x $\leq$ max_x) then ValueError
)

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	$\mathbb{R}$	-	ValueError

9.4 Semantics

9.4.1 State Variables

world: World
(The 'world' variable could either reside in the simulator module or the manager module. However, since the simulator module is the only one with write access to the 'world' variable, it is more logical to keep it there.)

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:
  - Calculate and update the acceleration of the cart and the pendulum.
  - Update the 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:

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

This module solves the above equations using the Runge-Kutta method. It treat the above equations as a system of linear equations.

There are more advanced equations that considers the friction and the damping of the cart and the pendulum. https://github.com/jitendra825/Inverted-Pendulum-Simulink, This repository contains matlab code that simulates the inverted pendulum with the friction and the damping of the cart and the pendulum. If it's easier to implement the friction and the damping, we can use the code in this repository.

- output: -
- exception: $exc :=$ (
  if $dt \leq 0$ then ValueError
)

10 MIS of World Module

10.1 Module

World

10.2 Uses

None

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: $\mathbb{R}$
$\dot{x}$: $\mathbb{R}$
$\ddot{x}$: $\mathbb{R}$
θ: $\mathbb{R}$
$\dot{\theta}$: $\mathbb{R}$
$\ddot{\theta}$: $\mathbb{R}$
F: $\mathbb{R}$

10.4.2 Environment Variables

None

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

None

11 MIS of Manager Module

11.1 Module

Manager

11.2 Uses

Simulator Module (Section 9)
Controller Module (Section 13)

11.3 Syntax

11.3.1 Exported Access Programs

Name	In	Out	Exceptions
run	-	-	-
resume	-	$\mathbb{B}$	-
draw	World	-	-
wait	-	-	-
quit	-	-	-

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

run():

- transition: It follows the following steps:
  - Get world from Simulator.
  - give it to Controller to get the force.
  - Give the force to Simulator to update the world state.

  - Call draw() to draw the updated world.
  - Call wait() to wait for some time.
  - Repeat the above steps until resume() returns false.

- output: -
- exception: -

resume():

- transition: -
- output: It's abstract method that will be implemented by the derived class. It returns true if the simulation should continue, otherwise false.
- exception: -

draw(world):

- transition: It's abstract method that will be implemented by the derived class. It draws the world state.
- output: -
- exception: -

wait():

- transition: It's abstract method that will be implemented by the derived class. It waits for some time.
- output: -
- exception: -

quit():

- transition: It's abstract method that will be implemented by the derived class. It quits the simulation.
- output: -
- exception: -

12 MIS of Main Module

12.1 Module

Main

12.2 Uses

Config Reader Module (Section 7)
World Module (Section 10)
Manager Module (Section 11)
Simulator Module (Section 9)
Controller Module (Section 13)

12.3 Syntax

12.3.1 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	-

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

None

12.4.3 Assumptions

None

12.4.4 Access Routine Semantics

main():

- transition: This is the main function of the program. It follows the following steps:
  - Read the configuration file using Config Reader.
  - Create the world using the configuration.
  - Create the Manager, Simulator, and Controller.
  - Call the run() method of the Manager.

- output:
- exception:

12.4.5 Access Routine Semantics

None

13 MIS of Controller Module

13.1 Module

13.2 Uses

Fuzzy Logic Module (Section 6)

13.3 Syntax

13.3.1 Exported Access Programs

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

13.4 Semantics

13.4.1 State Variables

None

13.4.2 Environment Variables

None

13.4.3 Assumptions

None

13.4.4 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: -

14 MIS of GUI Module

GUI module is child of Manager module.

14.1 Module

GUI

14.2 Uses

World Module (Section 10)
Manager Module (Section 11)

14.3 Syntax

14.3.1 Exported Access Programs

Name	In	Out	Exceptions
resume	-	$\mathbb{B}$	-
draw	World	-	-
wait	-	-	-
quit	-	-	-

14.4 Semantics

14.4.1 State Variables

None

14.4.2 Environment Variables

win: 2D diagram displayed on the screen

14.4.3 Assumptions

None

14.4.4 Access Routine Semantics

resume():

- transition: -
- output: It checks if the user clicked on the close button of the window. If yes, it returns false, otherwise true.

- exception: -

draw(world):

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

wait():

- transition: The wait time is dynamically determined by the GUI module to meet the frame rate requirements.
- output: -
- exception: -

quit():

- transition: Closes the gui window.
- output: -
- exception: -

15 MIS of File Output Module

File Output module is child of Manager module.

15.1 Module

File Output

15.2 Uses

World Module (Section 10)
Manager Module (Section 11)

15.3 Syntax

15.3.1 Exported Access Programs

Name	In	Out	Exceptions
resume	-	$\mathbb{B}$	-
draw	World	-	-
wait	-	-	-
quit	-	-	-

15.4 Semantics

15.4.1 State Variables

None

15.4.2 Environment Variables

file: output csv file.

15.4.3 Assumptions

None

15.4.4 Access Routine Semantics

resume():

- transition: -
- output: It returns true for n steps of simulation time.
- exception: -

draw(world):

- transition: Write the current world state to a file.
- output: -
- exception: -

wait():

- transition: Do nothing.
- output: -
- exception: -

quit():

- transition: Closes the file.
- 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.