Module Interface Specification for TPG

Team 3, Tangle
Calvyn Siong
Cyruss Allen Amante
Edward Gao
Richard Li
Mark Angelo Cruz

January 8, 2025

1 Revision History

Date	Version	Notes
01/15/2025	0.0	Revision 0

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at this link. This section records information for easy reference.

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
DNN	Deep Neural Network
FR	Functional Requirement
M	Module
MG	Module Guide
MIS	Module Interface Specification
Multi-Task RL	Multi-Task Reinforcement Learning
NFR	Non-Functional Requirement
OS	Operating System
R	Requirement
RL	Reinforcment Learning
SC	Scientific Computing
SRS	Software Requirements Specification
TPG	Tangled Program Graphs
UC	Unlikely Change

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

The structure of the MIS for modules comes from **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. 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 TPG.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
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 (-∞, ∞)

The specification of TPG 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, TPG 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	N/A
Behaviour-Hiding Module	Classic Control Module MuJoCo Module Visualization Module Logging Module
Software Decision Module	TPG Experiment Module Framework Module

Table 1: Module Hierarchy

6 MIS of MuJoCo Module

6.1 Module

MujocoEnv

6.2 Uses

MuJoCo Framework (mujoco/mujoco.h), Task Environment (TaskEnv.h)

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	Input	Output	Exceptions
mujocoEnv	-	-	-
~mujocoEnv	-	-	-
reset	{rng : mt19937&}	-	-
terminal	-	bool	-
sim_step	{action : std::vector<double>}	Results	-
initialize_simulation	-	-	Unable to load binary model
set_state	{qpos : std::vector<double>, qvel : std::vector<double>}	-	-
do_simulation	{ctrl : std::vector<double>, n_frames: int}	-	-
GetObsSize	-	int	-

6.3.3 State Variables

Name	Type	Description
m_	mjModel*	Pointer to the MuJoCo model
d_	mjData*	Pointer to the MuJoCo data
cam_	mjvCamera	Camera used for visualization
opt_	mjvOption	Visualization options
scn_	mjvScene	Visualization scene
con_	mjrContext	Custom GPU rendering context
init_qpos_	std::vector¡double¿	Initial positions
init_qvel_	std::vector¡double¿	Initial velocities
model_path_	String	Absolute path to model xml file
frame_skip_	Integer	Number of frames per simulation step
obs_size_	Integer	Size of the observation space

6.3.4 Environment Variables

None

6.3.5 Assumptions

- The MuJoCo environment is properly installed on the user's device.

6.3.6 Access Routine Semantics

mujocoEnv():

- output: *out* := self
- exception: none

~mujocoEnv():

- transition: self → destroyed
- exception: none

reset(*rng*):

- transition: resets the environment to initial state using a random number generator, *rng*.
- exception: none

terminal():

- transition: environment in terminal state ? terminal := true : terminal := false
- output: *out* := terminal
- exception: none

sim_step(*action*):

- transition: advances the simulation by one step using the given *action*.
- output: *out* := Results object containing simulation outcomes.
- exception: none

initialize_simulation():

- transition: initializes the simulation by loading the MuJoCo model and creating its data structure.
- exception: unable to load binary model error.

set_state(*qpos*, *qvel*):

- transition: sets the positions and velocities, *qpos* and *qvel*, in the simulation state.
- exception: none

do_simulation(*ctrl*, *n_frames*):

- transition: executes *n_frames* of simulation steps with the given control inputs, *ctrl*.
- exception: none

6.3.7 Local Functions

None

7 MIS of Visualization Module

7.1 Module

The visualization module implements visualization functionality of a TPG experiment with graphs, charts, video playback, and a realtime simulation.

7.2 Uses

The visualization module is used to visualize the results of a TPG experiment while it is running or after the experiemnt has completed. This information contains data pertaining to what is going on in an experiment, and performance metrics.

7.3 Syntax

7.3.1 Exported Constants

N/A

7.3.2 Exported Access Programs

Name	**Input**	**Output**	**Exceptions**
plot-stats	{fitnessValues: int, generationTime: int, programCount: int, teamCount: int}	{PDF file}	Returns a PDF file of plot statistics for an experiment
run-mpi	{fitnessValues: int, generationTime: int, programCount: int, teamCount: int}	{OpenGL animation}	An OpenGL animation starts to run

7.4 Semantics

7.4.1 State Variables

N/A

7.4.2 Environment Variables

Name	**Type**	**Description**
tpg.[experimentId].std	File(s)	output log files generated for each experiment

7.4.3 Assumptions

- Format of the log files is known and will not change.

7.4.4 Access Routine Semantics

N/A

7.4.5 Local Functions

Name	Input	Output	Description
extractMetrics	{logFile: file}	{dataFile: file}	Saves extracted data into .rslt files
transformData	{logFile: file}	{dataFile: file}	Reverse processes the log files, pattern matches, reformatting, and aggregating the data and saving to a .rslt file
plotData	{logFile: file}	{plotFile: file}	Plots the data from the .rslt file and saves the plot to a .pdf file

8 MIS of TPG Experiment Module

8.1 Module

TPGExperimentMPI

8.2 Uses

The TPG Experiment module is responsible for managing and evolving different policies, evaluating different tasks, and tracking experiments. It is used during the execution of experiments through the commands: **tpg-run-slurm** and **tpg-run-mpi**, which are used for executing tasks in virtual and local machines respectively.

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

None

8.4 Semantics

8.4.1 State Variables

Name	Type	Description
world_rank	Integer	The rank of the MPI process within the environment. It is used to distinguish between the master and evaluator processes.
n_task	Integer	The current number of tasks (environments) running.
active_task	Integer	The index of the current active task.
phase	Integer	The numerical representation of the current phase of the program.
t_current	Integer	The current generation or iteration of the task.
t_start	Integer	The starting generation or iteration of the task.
task_to_replay	Integer	The index of the task to replay.

8.4.2 Environment Variables

Name	Type	Description
COMET_API_KEY	String	The API key used for Comet experiment tracking service

8.4.3 Assumptions

- Experimental parameters such as **mj_model_path** and **active_tasks** are well-defined.

8.4.4 Access Routine Semantics

None

8.4.5 Local Functions

None

9 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

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

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?
2. What pain points did you experience during this deliverable, and how did you resolve them?
3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?
4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?
5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)
6. 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)