Reflection and Traceability Report on TPG

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

1 Changes in Response to Feedback

This section summarizes the changes made over the course of the capstone project in response to feedback from sources such as TAs, the supervisor and other teams. The associated commits can be found by clicking on the associated issue created.

1.1 SRS and Hazard Analysis

Here is the feedback we received on the SRS and Hazard Analysis documents, and the changes we made in response to that feedback.

Table 1: Feedback and Changes for SRS Documentation

Feedback Source	**Feedback Item**	**Response**	**Issue**
TA Feedback	Formalization	**Did Not Fix**: Already formalized to the best of our ability	#311
TA Feedback	Extension of Knowledge	Mentioned and cited sources where terms are taken from.	#310
TA Feedback	Verifiable Requirements	Updated requirements to ensure they were testable and measurable.	#309
TA Feedback	Traceable Requirements	Added traceability matrix to enhance traceability.	#308
TA Feedback	What not How (Abstract)	Revised some requirements to focus on "what" the system should do rather than "how" it should do it.	#307
TA Feedback	Content of SRS (Functionality and Specificity)	Revised functional requirements and clarified ambiguous sections.	#305

Feedback Source	Feedback Item	Response	Issue
TA Feedback	Formatting and Style	Modified formatting according to feedback.	#306
Peer Review	Project Goals	**Did Not Fix**: Decided to focus on main goals of project.	#106
Peer Review	Verifiability	Adjusted specified requirements for verifiability.	#105
Peer Review	User Business	Clarified problem context.	#104
Peer Review	Dev Planning	Updated development planning section with expected deadlines.	#102
Peer Review	Data Dictionary and Scope	**Did Not Fix**	#101
Peer Review	Maintainability, Supportability, Adaptability Requirements	Adjusted requirements for maintainability, supportability, and adaptability.	#107
Peer Review	Fix Functional Requirements	Revised concerned FR-6 for specificity.	#103

Table 2: Feedback and Changes for Hazard Analysis

Feedback Source	Feedback Item	Response	Issue
TA Feedback	Recommended Actions	**Did Not Fix**: Addressed in Peer Feedback	#314
TA Feedback	Hazard Identification	Adjusted concerned sections with feedback.	#313
TA Feedback	Spelling and Grammar	Corrected spelling and grammar errors and implemented other feedback specified.	#312
Peer Review	Inconsistent Hazard Reference	Fixed inconsistency between hazard references.	#136
Peer Review	Potential Missing Hazard for FMEA	**Did Not Fix**: Having these experiments that produce poor results could be a good thing that lets us tweak the TPG algorithm	#135
Peer Review	Priority Assignment	**Did Not Fix**: Not required	#133
Peer Review	No Mitigation Strategy	Modify mitigation strategies for hazards.	#132
Peer Review	Prioritization Justification	Provided detailed justification for hazard prioritization.	#130

Feedback Source	Feedback Item	Response	Issue
Peer Review	SRS Linking Roadmap	Linked SRS in roadmap to hazard analysis.	#128
Peer Review	Ambiguous Terms	Clarified ambiguous terms in the hazard analysis.	#134

1.2 Design and Design Documentation

Here is the feedback we received on the design documents (MG and MIS), and the changes we made in response to that feedback.

Table 3: Feedback and Changes for Module Guide

Feedback Source	Feedback Item	Response	Issue
TA Feedback	Quality Information	Fixed all addressed concerns with issue.	#346
Peer Review	Lack of Links to Other Documents	Added links and references to related documents/sections for better traceability.	#242
Peer Review	Module Decomposition	**Did Not Fix**: Decomposition was deemed unnecessary for the current scope.	#240

Table 4: Feedback and Changes for Module Specification Interface

Feedback Source	Feedback Item	Response	Issue
TA Feedback	Sketches for Enough to Build	**Did Not Fix**: Did not include additional sketchecs or examples, as current level of detail seemed sufficient for our project scope.	#347
Peer Review	Confusion on TPG Experiment Module	Clarified confusing sections in the module specification interface.	#245
Peer Review	Lack of Info for Independent Developer	Added additional details to support independent developers.	#243
Peer Review	Incorrect "Uses" in MIS	Corrected "Uses" subsections in the module specification interface for modules.	#244

1.3 VnV Plan and Report

Here is the feedback we received on the VnV Plan and VnV Report, and the changes we made in response to that feedback.

Table 5: Feedback and Changes for VnV Plan

Feedback Source	Feedback Item	Response	Issue
TA Feedback	Inaccurate descriptions in 2.1 Summary and 2.2 Objective	Modified the two sections to better reflect rubric	#315
TA Feedback	Improve Spelling and Grammar and Style of VnV Plan	Addressed feedback given in issue	#316
Peer Feedback	Add traceability in VnV Plan	**Did Not Fix**: Already have proper traceability	#150
Peer Feedback	Increase Detail in Nonfunctional Test Descriptions	Addressed feedback in issue by adding details	#151
Peer Feedback	Survey Questions for Documentation	Added survey questions	#155

Table 6: Feedback and Changes for VnV Report

Feedback Source	Feedback Item	Response	Issue
Peer Feedback	Limited Explanation of Changes Due to Testing	**Did Not Fix**: We did address the VS Code Dev Containers as a response	#396
Peer Feedback	Limited Explanation of Changes Due to Testing	**Did Not Fix**: Deemed not relevant for VnV Report.	#398
Peer Feedback	Missing Test Description for FR-SNL6 in both VnV documents	Updated the VnVReport with the test's full description in the same format as other tests	#398
Peer Feedback	Traceability Matrix Inconsistency	Removed test to get rid of inconsistency	#398
Peer Feedback	Lack of Criterion for Usability Tests	**Did Not Fix**: already added justifications on what constitutes a pass through the scores we assigned users	#400
Peer Feedback	Missing NFR test case that is present in the VnVPlan	Included proper NFR solution	#401
Peer Feedback	Lack of clarity with feedback changes	**Did Not Fix**: Already talked about the improvements	#402

2 Challenge Level and Extras

2.1 Challenge Level

The challenge level of the project is **advanced** as agreed upon by the course instructor since this project is an extension of the current Tangled Program Graphs repository created by Dr. Stephen Kelly.

2.2 Extras

The extra tackled by this project is a Research Report. This report will cover the research and results obtained from incorporating dynamic memory to enhance reinforcement learning within MuJoCo using Tangled Program Graphs. It explores this through single-task (STL) and multi-task (MTL) experiments on MuJoCo environments such as Inverted Pendulum, Half Cheetah, and Humanoid Standup. The Research Report can be found here.

3 Design Iteration (LO11 (PrototypeIterate))

The design and implementation of our capstone project were driven by the needs of our client, Dr. Kelly, and his research group. As contributors to their existing codebase, our efforts focused on addressing pain points in current workflows and enhancing the overall developer experience, aligning with the goal of facilitating research on the TPG framework within more complex environments.

3.1 MuJoCo Environment Integration and Expansion

Our initial work involved integrating MuJoCo environments to expand the capabilities of the TPG framework. Dr. Kelly's group had previously implemented the "Ant" environment. Recognizing the need to evaluate the TPG framework in more advanced physics engines, we implemented support for additional environments such as Inverted Pendulum, Humanoid Standup, and Half Cheetah. This directly addressed Dr. Kelly's desire to transition to MuJoCo, enabling experimentation within more sophisticated and realistic simulations.

This effort culminated in addressing Dr. Kelly's primary research question: evaluating TPG's performance in multi-task scenarios. Building upon the individual environment implementations, we iterated on combining MuJoCo environments, enabling the evolution of policies capable of learning across multiple tasks. This progression demonstrates a clear evolution driven by the client's research objectives, moving from basic integration to tackling more complex, multi-task learning problems.

3.2 Migration to .csv Logging

A significant change involved modifying the logging system. Dr. Kelly expressed a desire to transition data analysis processing from R to Python-based tools, citing maintainability and the flexibility of Python libraries as key motivations. To accommodate this transition, we migrated the logging format from `.std` and `.err` files to `.csv` files.

The choice of `.csv` over the previous `.std` format offers several advantages from the perspective of user needs:

- **Organization:** `.csv` files provide a structured, tabular format. This makes it easier to import and analyze data using Python libraries like Pandas, streamlining the data processing pipeline. The previous `.std` format required more complex parsing and was less amenable to automated analysis.
- **Accessibility:** `.csv` is a widely supported and easily accessible format. Researchers can readily open and manipulate `.csv` files in various software packages, fostering collaboration and simplifying data sharing.
- **Specific Metric Capture:** We designed the `.csv` logs to capture specific metrics from each stage of the evolutionary process. This allows for targeted analysis of performance and behavior, providing Dr. Kelly and his team with the data they need to evaluate the TPG framework effectively.

By migrating to `.csv` logs, we not only accommodated Dr. Kelly's preference for Python-based tools but also improved the organization, accessibility, and analytical potential of the logged data, directly benefiting the research workflow.

3.3 CLI Tool Development

Initially, the execution of experiments relied on script-based execution, requiring developers to navigate to specific directories and execute commands with specific parameters. This workflow was identified as a pain point, particularly when running multiple experiments concurrently.

In response to this usability issue, we developed a Command Line Interface (CLI) tool to simplify experiment execution. This change was presented to Dr. Kelly, who agreed that a streamlined workflow would significantly improve the developer experience.

The initial "MVP" version of the CLI tool supported essential functions such as:

- `tpg evolve [environment]`: Evolving a policy for a given environment.
- `tpg replay [environment]`: Replaying the best-performing agent in an environment.
- `tpg plot [environment]`: Plotting relevant statistics.

These commands mirrored the functionality of the original scripts but offered a more intuitive and descriptive interface. After merging the initial version and providing documentation, Dr. Kelly's research group provided valuable feedback. They requested additional functionality, such as:

- `tpg clean`: Removing old log files to maintain a clean working environment.
- `tpg kill`: Terminating MPI processes.

Based on this feedback, we iterated on the CLI tool, incorporating these new features to further streamline the user experience and address the specific needs of the research team. The final CLI tool represents a significant improvement in usability, allowing researchers to execute, manage, and analyze experiments with greater ease and efficiency.

In summary, our design iterations were continuously guided by the needs of Dr. Kelly and his research group. By focusing on improving existing workflows, accommodating new technologies, and responding to user feedback, we were able to develop a system that directly supports their research goals and enhances the overall developer experience.

4 Design Decisions (LO12)

This section justifies the key design decisions made throughout the project, explaining how they were influenced by our initial assumptions, the limitations we encountered, and the constraints imposed upon us. The aim was to evolve the TPG framework into a more robust, usable, and extensible system suitable for advanced research and easier collaboration.

4.1 Build System Migration (SCons to CMake)

- **Initial State & Limitation:** The original TPG project utilized `SCons`. While functional, `SCons` is less common in the modern C++ ecosystem than `CMake`. We found it presented limitations regarding straightforward integration with standard C++ tooling (like IDEs), effective dependency management (especially for testing frameworks like `Catch2`), and seamless configuration within modern CI/CD pipelines (GitHub Actions).
- **Assumption:** We assumed that migrating to `CMake`, a widely adopted industry standard, would significantly improve long-term maintainability, ease developer onboarding, and simplify the integration of necessary components. We also assumed the team could acquire the necessary `CMake` proficiency.
- **Constraint:** The need to reliably build a complex C++ project involving multiple external dependencies (Eigen, Boost, MuJoCo, MPI, YAML-CPP, `Catch2`) across potentially different setups (even within a container) mandated a robust and well-supported build system.
- **Decision & Justification:** We decided to invest the effort in migrating the build system entirely to `CMake`. This addressed the limitations of `SCons` by providing a standardized platform. It simplified adding unit tests, enabled easier configuration for code coverage tools (though challenges remained), and made integration with GitHub Actions workflows much more straightforward. The upfront effort was justified by the significant expected gains in project health, developer productivity, and CI/CD integration, aligning with the project goal of improving software engineering practices.

4.2 Development Environment Standardization (VS Code Dev Containers)

- **Limitation/Constraint:** A major initial hurdle, identified early on and confirmed by user feedback, was the inconsistency in development environments across team members (macOS, Windows with WSL, Linux) and within Dr. Kelly's research group. This led to significant time lost on configuration, dependency conflicts (especially graphics libraries, MuJoCo setup), and a steep onboarding curve, directly hindering productivity and collaboration.
- **Assumption:** We assumed that developers would be amenable to using VS Code and that Docker containerization could provide a fully consistent, Linux-based development environment regardless of the host OS, encapsulating all complex dependencies.
- **Decision & Justification:** We implemented and documented a VS Code Dev Container setup. This directly tackled the limitation of environment inconsistency and the cross-platform constraint. It ensured all developers worked with the exact same dependencies and tools,

drastically reducing setup time (from potentially weeks to minutes, per usability feedback) and eliminating "works on my machine" issues. This was crucial for enabling effective teamwork, reliable testing, and predictable CI/CD runs.

4.3 Logging Framework Overhaul (.std/.err to Typed CSVs)

- **Limitation:** The original logging mechanism, primarily redirecting `stdout`/`stderr`, made automated data extraction and analysis difficult and inefficient, hindering the research workflow which requires plotting and comparing metrics. Dr. Kelly also desired a shift towards Python-based analysis.
- **Assumption:** We assumed that structuring log output into typed CSV files (one per metric type) would be significantly easier to parse programmatically (e.g., using Pandas) and that separating metrics by type would improve organization.
- **Decision & Justification:** We designed and implemented an event-driven logging system where specific metric events are emitted and captured by loggers writing to dedicated, typed `.csv` files in a structured `logs/` directory hierarchy (split by metric type). This decision directly addressed the parsing limitations and facilitated Python-based analysis, enabling the creation of plotting scripts (`tpg-plot-evolve.py`) and systematic analysis for the research report extra. Usability feedback confirmed the usefulness but highlighted areas for documentation improvement (e.g., explaining metric names).

4.4 User Interface (Script-based to Python CLI)

- **Limitation:** As evidenced by the project's early README versions and initial team experience, interacting with the TPG framework relied on executing various shell scripts (`.sh`), requiring specific knowledge of arguments, paths, and context. This was cumbersome, error-prone, and presented a barrier to easy experimentation and onboarding.
- **Assumption:** A unified Command Line Interface (CLI) built with Python (common in the ML/research ecosystem) and a library like Click would provide a more intuitive, discoverable, and consistent user experience, abstracting away script complexities.
- **Decision & Justification:** We developed the `tpg` CLI tool, migrating away from the collection of shell scripts. This provided a single, user-friendly entry point, integrated the loading of environment configurations (initially `.txt`, later migrated to `.yaml`), handled directory setup automatically, and offered clear commands with help messages. The initial design addressed the core limitations, and subsequent refinements (adding `kill`, `debug`, `clean` commands) were directly driven by feedback from Dr. Kelly's research group, demonstrating its value in improving the practical research workflow.

4.5 MuJoCo Environment Integration (Adapter/Environment Classes)

- **Constraint:** A primary project requirement was to integrate the MuJoCo physics simulator.
- **Limitation:** TPG's engine needed a robust way to interact with the MuJoCo C API, managing its specific data structures (`mjModel*`, `mjData*`), simulation steps, and differing state/action spaces, without cluttering the core TPG logic.

- **Assumption:** Encapsulating environment-specific logic within dedicated C++ classes inheriting from a common base (`MujocoEnv`, itself inheriting `TaskEnv`) would provide effective abstraction and allow the core TPG algorithm to remain generic. Configuration could be managed externally via files.
- **Decision & Justification:** We implemented specific classes for each target MuJoCo environment (Ant, Half-Cheetah, etc.) within `src/src/environments/mujoco/`. These classes handle MuJoCo model loading, API interactions (stepping, state/action handling), reward calculation, and termination logic, isolating these details from the main TPG engine. Configuration was managed via `.yaml` files. This object-oriented design successfully met the integration constraint while managing the inherent complexity of the task through abstraction and encapsulation.

5 Economic Considerations (LO23)

Since the project will eventually be open-sourced once the research work is finished, its primary market does not include direct commercial sales but rather the research community, AI practitioners, and robotics engineers and enthusiasts.

Attracting users will require a combination of strategies to enhance community engagement, increase project visibility, and improve usability. A few key approaches are:

- **Open-Source Repository**: Hosting the project on GitHub with a well-organized README, issue tracker and contribution guidelines would encourage potential contributors.
- **Conference & Workshop Presentations**: Presenting the project at ML and robotic conferences such as GECCO and Conference on Artificial Life would increase its visibility among researchers.
- **Publishing Research & Documentation**: Continue releasing research papers and technical reports explaining TPG's capabalities and how well it compares with alternatives.
- **Showcasing Notable Results**: Publishing blog posts, YouTube videos, as well as engaging with the reinforcement learning community through Hugging Face forums, GitHub discussions, Reddit and other online platforms.

TPG's potential lies in academic research, robotics applications, and open-source adoption. The long-term strategy would focus on gaining traction in research and industry collaborations, which could lead to funding opportunities, grants, or potential commercial applications in reinforcement learning for real-world robotic systems.

While it's difficult to estimate exact numbers of potential users, open-source RL-based GitHub projects often have thousands of users. For example, OpenAI's Gym has 35.6k stars GitHub, while Stable Baselines3 has around 10.2k stars. Given TPG's unique focus, it has the potential to attract a niche but highly engaged user base of researchers, engineers, and developers in the AI and robotics space.

6 Reflection on Project Management (LO24)

6.1 How Does Your Project Management Compare to Your Development Plan

Our team properly followed our development plan with respect to the workflow plan, team member roles, and communication plan. As stated in our Development Plan, we used the "TPG Capstone" Discord server for team communication, which was consistently utilized throughout the project to discuss ideas, host meetings and notify the team for PR requests. Additionally, the development stack and technologies mentioned in the Development Plan, such as Git, GitHub, and C++, were integral to the development of our project.

We made sure that each team member adhered to their designated roles and responsibilities by assigning each member tasks that matched their roles. Our team meetings were held regularly, and we successfully coordinated schedules to accommodate everyone, leading to regular team attendance. Overall, our project management was carried out efficiently, staying true to our original plan.

6.2 What Went Well?

Our project management was effective, particularly in the implementation of MuJoCo environments, which proceeded smoothly. This success allowed us to build upon our initial work by exploring multi-tasking for the TPG-Mujoco interface. Additionally, we enhanced our project with the development of new CLI tools, which streamlined our workflow and improved the overall efficiency of our processes. Our team communication and collaboration were strong throughout the project. This ensured that all team members were aligned with project goals and timelines. The use of GitHub for version control and project management also contributed to our success, allowing us to track progress effectively and manage tasks efficiently through issues. Overall, these factors combined to create a productive working environment.

6.3 What Went Wrong?

One of the challenges we faced was the inability to implement all the MuJoCo environments we initially planned due to unforeseen complications. Additionally, the process of migrating from SCons to CMake proved to be quite challenging due to a lack of updated documentation. Debugging C++ was particularly difficult as changes were occurring regularly in the TPG framework, and our limited experience with C++ added to the complexity. These issues highlighted areas where we needed to improve our skills and adapt our strategies to better handle such challenges in the future.

6.4 What Would you Do Differently Next Time?

In future projects, we would aim to diversify our roles within the team, allowing each member to explore multiple areas of software development. This would encourage a more varied approach and broaden our collective skill set, as it challenges us to step outside our comfort zones and tackle different aspects of the development process. Additionally, experimenting with a different development technologies and frameworks could be both exciting and beneficial, as it presents new challenges and learning opportunities. While our current stack was effective, exploring newer frameworks or languages might enhance performance and developer efficiency.

We would also likely place a greater emphasis on more comprehensive unit testing to ensure robust code quality, as our testing scope was relatively limited. This would involve more inclusive testing strategies that cover a wider range of scenarios and edge cases. We would likely also have implemented more complex MuJoCo environments for evaluation, which would have provided a deeper understanding of the TPG framework's capabilities and limitations in handling sophisticated simulations.

7 Reflection on Capstone

This section focuses on the key learnings and reflections gained during the course of the capstone project.

7.1 Which Courses Were Relevant

Several courses proved highly relevant to the successful completion of our capstone project. Among the most impactful were:

1. **2AA4 - Introduction to Software Design:** This course provided a foundational understanding of collaborative software development practices. The experience of working on a group project using GitHub, including managing issues, utilizing Kanban boards, and creating pull requests, was invaluable. It instilled the fundamentals of effective team-based development. Furthermore, the emphasis on software design principles, such as SOLID and GRASP, was directly applicable to the capstone project. Although the course was primarily focused on Java, the object-oriented design patterns learned were transferable to our project, which utilized Python and C++. Specifically, we leveraged the Observer pattern to implement a refined logging system within the TPG codebase.

2. **3RA3 - Requirements Engineering:** This course provided a solid foundation in defining project requirements prior to implementation. Learning to articulate and prioritize user needs was instrumental in guiding our development efforts. The user-based design principles learned in 3RA3 helped us focus our efforts and prioritize features that aligned with Dr. Kelly's research objectives. The reports we submitted throughout the capstone project mirrored the structure and content of assignments completed in 3RA3, demonstrating the practical application of the course's principles.

7.2 Knowledge/Skills Outside of Courses

Beyond the formal curriculum, our capstone project required the acquisition of specific knowledge and skills in areas not explicitly covered in our coursework. These included:

1. **Genetic Programming:** Gaining an understanding of genetic programming, a specialized research area, was essential. Since none of us had prior exposure to this domain, our learning primarily involved direct instruction from our supervisor, Dr. Kelly, as well as in-depth analysis of the existing TPG codebase. This process required us to quickly grasp complex concepts and apply them to our project.

2. **C++ Proficiency:** C++ was the language of choice for performance-critical aspects of the project, a decision driven by Dr. Kelly's preference. While our curriculum included object-oriented languages like Java, C++ was not a primary focus. We therefore had to develop proficiency in C++ independently, leveraging our understanding of object-oriented principles to navigate the language's syntax and intricacies.

3. **Docker Containerization:** The project made extensive use of Docker for containerization, a topic not typically covered in our academic coursework. While some of us had gained exposure to Docker during co-op placements, a deeper understanding of how containers work and their benefits was necessary. We independently learned how to install, run, and utilize Docker, developing a valuable skill essential in modern software development workflows. Given the prevalence of containerization in industry, incorporating some introductory material on this topic into the curriculum could benefit future students.