Hazard Analysis
RoCam

Team #3, SpaceY
Zifan Si
Jianqing Liu
Mike Chen
Xiaotian Lou

October 19, 2025

Table 1: Revision History

Date	**Developer(s)**	**Change**
Date1	Shike Chen	Initial draft
Date2	Shike Chen	Detailed draft

Contents

1 Introduction

Hazard is generally defined as a harm or potential harm or effect caused by a system which may lead to personal injury, property damage, environmental damage etc. As an integrated system with both hardware and software components, hazards may arise due to improper design, malfunction of the system, unexpected user behavior, etc. Therefore, it is critical to identify potential hazards and mitigate them in the early stage of the project. In the case of the RoCam project, the potential hazards could cause physical injury of the user or malfunction of a larger system.

2 Scope and Purpose of Hazard Analysis

The hazard analysis will identify any direct or potential hazards caused by the system which includes hazardous materials, design, user safety, etc. The hazard analysis will also address the general hazardous effects of this system which includes hazards effects in the events of malfunction, misuse or environmental damage. However, since the project is open source, any use case that is beyond the scope of this project will not be included in this hazard analysis.

Such hazard analysis will not only used to identify potential hazards caused by the system, but also to minimize risks related in unforeseen cases. The purpose of this hazard analysis is to properly address the potential harm and risks caused by the product to the user and the external environment.

3 System Boundaries and Components

3.1 Hardware Components

3.1.1 Motion control modules

The motion control module consists of a custom-designed PCB board utilizing an STM32 microcontroller. This module is responsible for controlling the movement of the camera gimbal. It receives commands from our Computer Vision module and translates them into precise motor movements in real time.

3.1.2 Camera gimbal

The camera gimbal used in this project is a existing Camera Gimbal designed by the McMaster Rocketry team. Since our team is not responsible for the design of the gimbal, we can only estimate the potential hazards based on similar products in the market.

3.1.3 Jetson Orin Nano

The Jetson Orin Nano is small, lightweight AI computer designed by NVIDIA. It will be used to deploy our computer vision moodule and communicate directly with the motion control module.

3.1.4 Camera

The camera is a off the shelf camera that is compatible with the Jetson Orin Nano. It is able to provide 4k video input in real time.

3.2 Software Components

3.2.1 Computer Vision Module

Our computer vision module will be developed in Python. The base architecture will be torch and pandas. It will be accerlerated with TensorRT in Jetson to achieve real time stable performance.

3.2.2 User Interface Module

The user interface module will be developed using React. It will provide a web-based interface for users to interact with the system, view camera feeds, and control the camera gimbal.

4 Critical Assumptions

4.1 Assumptions about the Operating Environment

The system will not operating in any extreme environment (e.g., extreme temperature, acidic environments, etc.). The system will be operating within 10 to 25 degree Celsius. The system will operate with a stable power supply.

4.2 Assumptions about the User

Anyone above 14 years old who uses electronic and exposed to software interface on a daily basis for over 2 years with good intent.

4.3 Assumptions about general of the shelf hardware components

All off-the-shelf hardware components will function as advertised and will be compatible with our system. This includes the camera, gimbal, Jetson Orin Nano etc.

4.4 Assumptions about the software libraries

All software libraries used will not have any unknown security vulnerabilities.

5 Failure Mode and Effect Analysis

Table 2: FMEA Table

Design Function	Failure Mode	Effect	Cause of Failure	Recommended Action
Gimbal turn as CV control	1. Gimbal does not move 2. Gimbal snaps 3. Gimbal does not turn properly	1. (a) Camera loses the rocket (b) user unable to debug the rocket launch 2. (a) Camera loses the rocket (b) Camera injures user (minor cuts and bruises) (c) Camera and gimbal damages other property (d) Camera, gimbal, motion control module, Jetson Orin Nano damages 3. (a) Camera overshoots or loses the rocket (b) Camera unable to track the rocket smoothly with high quality image	1. (a) failed in communication between the motion control module and the gimbal (b) failed in communication then CV module and the motion control module (c) firmware bug in the motion control module or compatibility issue (d) fried electric components in the motion control module or gimbal 2. (a) improper instruction from the motion control module (b) rocket angular velocity to the gimbal excesses the design limit 3. (a) improper instruction from the motion control module (b) CV module fails to provide proper instruction to the motion control module (c) CV module lost the rocket (d) damaged components in the motion control module or gimbal	1. (a) careful handling in transportation and assembly to avoid damage to the components (b) Integration test before launch to ensure proper communication between all modules 2. (a) Set threshold of the maximum angular velocity to avoid gimbal snapping (b) Set threshold of the maximum angle to avoid gimbal snapping (c) Calculate the expected angular velocity of the camera system and the (SR1) target to ensure they are within the design limit of the system (d) safety goggles should be worn when operating the system (SR2) 3. (a) design CV module to be robust in different environment (b) design motion control module and CV module speed to have self-correcting (SR3) mechanism (c) design multiple tracking algorithms to avoid losing of the rocket
User Interface control monitor tracking	1. UI does not load 2. UI loads but does not display camera feed 3. User unable to understand and operate the UI	1. (a) User unable to control the gimbal and camera (b) Failed to monitor the rocket launch (c) Failed to track the rocket launch (d) User frustration 2. (a) User unable to monitor rocket launch in real time (b) User unable to track the rocket launch in real time (c) User unable to select desired tracking mode and target 3. (a) User unable to control the gimbal and camera (b) Failed to monitor the rocket launch (c) Failed to track the rocket launch (d) User frustration	1. (a) Bug in the UI software (b) Issue in CD pipeline (c) Browser compatibility issue (d) OS environment issue 2. (a) Issue in API between UI and CV module (b) Camera or Jetson Orin Nano hardware issue 3. (a) Poorly designed UI (b) Lack of user manual and documentation	1. (a) Thorough testing in different browsers and OS environment (b) Proper CI/CD pipeline to ensure smooth deployment (c) Provide user manual and documentation for installation and troubleshooting 2. (a) Thorough testing of the API between UI and CV module (b) Provide user manual and documentation for installation and troubleshooting (c) Careful handling in transportation and assembly to avoid damage to the components 3. (a) User testing and feedback to improve UI design (b) Provide user manual and documentation for installation and troubleshooting (c) Provide user with access to an AI Agent that read the user manual and documentation

6 Safety and Security Requirements

1. SR1: The system will be placed in a distance which the expected angular velocity will not exceed the design limit of the gimbal.
2. SR2: all operators must wear safety goggles when operating the system.
3. SR3: The system will have a self-correcting mechanism to ensure the camera can track the rocket smoothly in real time.

7 Roadmap

7.1 Capstone Implementation Plan (Sprints & Milestones)

Table 3: Capstone Roadmap (8 sprints)

Sprint	**Primary Goals**	**Deliverables**	**Safety Focus (Traceability)**	**Verification / Evidence**
1	System bring-up: Jetson Orin Nano, camera, repo CI. Define interfaces (UI $\leftrightarrow$ CV $\leftrightarrow$ Motion).	Env setup scripts; interface spec; smoke tests.	Baseline hazard review; confirm assumptions (power, temp).	Checklist, boot logs, interface mock tests.
2	Motion control MVP: gimbal command path; encoder/limits readout.	Motor driver config; limit-switch handling; teleop tool.	SR1 (angular velocity/angle limits) draft.	Bench test: limit cut-offs; plotted step responses.
3	CV tracking MVP on sample footage (offline $\rightarrow$ near-real-time).	Tracker module; target reacquisition logic.	SR3 draft (self-correction, loss recovery).	Replay tests; failure-injection (occlusion) logs.
4	UI prototype: live feed, mode select, status & alarms.	Web UI with HUD; error banners; safe-state button.	FMEA controls for "UI not loading/displaying".	Cross-browser tests; watchdog for API health.
5	Closed-loop integration: CV $\rightarrow$ Motion with rate limiting.	End-to-end demo; rate limiter & soft stops.	SR1 formalized (hard/soft limits); map FMEA "gimbal snap".	Angular-rate sweep; saturation tests; limit breach proofs.
6	Robustness: fallback trackers, loss detection, auto-recenter.	Multi-strategy tracker; loss timers; homing routine.	SR3 formalized (self-correction).	Scenario matrix (lighting/occlusion); recovery time stats.
7	Safety hardening: interlocks, E-stop, checklists, SOP.	E-stop wiring/command; pre-launch checklist; SOP v1.	SR2 (PPE), SR1 (interlocks), UI hazard cues.	Dry-run with operators; checklist sign-offs; video evidence.
8	Verification & docs: finalize FMEA, traceability, test report.	FMEA v2; SRS trace matrix; safety test report; demo video.	All SRs: SR1–SR3 complete and traced.	Pass/fail summary; residual-risk log; release tag.

7.2 Milestones

1. M1 (End Sprint 2): Motion MVP with enforceable soft limits (SR1 draft).
2. M2 (End Sprint 4): UI prototype with health/status and safe-state control.
3. M3 (End Sprint 6): Closed-loop tracking with recovery/self-correction (SR3).
4. M4 (End Sprint 8): Safety package complete (FMEA v2, SR1–SR3 verified, docs).

7.3 Traceability Plan

- FMEA → SR mapping: "Gimbal snaps/overshoots" → SR1; "Tracking lost" → SR3; "UI unavailable/unclear" → UI alarms, SOP, tests.
- Each sprint closes with: (a) updated FMEA row(s), (b) test artifacts linked to SR IDs, (c) residual risk review.

7.4 Acceptance Criteria

- **SR1**: Max angular velocity/angle never exceeded in any commanded or fault path; evidence: logs + plotted limits across stress tests.
- **SR2**: Operators use PPE; SOP and checklist enforced; evidence: signed checklist + demo footage.
- **SR3**: From target-loss events, system recovers or safely recenters within $T_{\max}$ seconds in $\geq 95\%$ of trials.

7.5 Post-Capstone Backlog (Future Work)

1. Hardware redundancy (limit sensors, power rails), formal E-stop certification.
2. Model optimization (TensorRT INT8), thermal profiling, enclosure/EMI review.
3. Expanded UI: guided wizards, operator training mode, automated pre-flight.
4. Data logging for incident analysis; automated regression test farm.
5. Security hardening: signed releases, SBOM, dependency scanning.

Appendix — Reflection

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 listed risks had your team thought of before this deliverable, and which did you think of while doing this deliverable? For the latter ones (ones you thought of while doing the Hazard Analysis), how did they come about?
4. Other than the risk of physical harm (some projects may not have any appreciable risks of this form), list at least 2 other types of risk in software products. Why are they important to consider?

1. **What went well while writing this deliverable? (CV perspective)** From the computer vision side, what went well was how the hazard analysis helped us think beyond just performance and accuracy. Normally, we focus on improving tracking or inference speed, but this deliverable made us consider how our module's failures could affect safety and the physical system. For example, if the CV model misidentifies or loses the rocket, the gimbal might swing too far or too fast. Listing these in the FMEA table showed clear links between the vision system and hardware movement. It also encouraged us to add fallback ideas, like recovery tracking or self-correction, which became part of SR3. The process improved our system awareness — we now design the CV pipeline not only to be fast, but also predictable and safe.
2. **What pain points did you experience during this deliverable, and how did you resolve them? (Development/UI perspective)** From the development side, one major challenge was figuring out how to document software-related hazards in a meaningful way. It was easier to picture a gimbal breaking than a user interface "failing," so at first, the risks in our module sounded less serious. We realized, however, that a software crash or unclear UI could lead to real operational problems if users can't control or monitor the camera. Formatting the FMEA table in LaTeX was also tricky — the entries for causes and effects got very long. We solved this by using short numbered lists and grouping related issues under one failure mode. Collaborating with the hardware and CV members also helped, because they could explain how our software might contribute to physical or timing failures, which gave us better context for our part of the analysis.
3. **Which of your listed risks had your team thought of before this deliverable, and which did you think of while doing this deliverable? How did they come about? (Hardware perspective)** From the hardware perspective, we had already identified the obvious mechanical hazards before writing this document — such as motor overdriving, gimbal snapping, or loose mounts. These were discussed early in the semester when testing servo drivers. However, new risks came up during the hazard analysis that we hadn't thought about before. For instance, we realized that communication loss between the motion control board and the Jetson could cause unpredictable movements, and that power fluctuations might damage sensitive components. These ideas surfaced while building the FMEA table, since it forced us to trace every "what if" scenario from signal input to physical motion. It also made us think about how environmental assumptions (like stable temperature and power) affect reliability. Writing this section helped the hardware team plan safer wiring, limit switches, and emergency stops for future testing.

4. **Other than the risk of physical harm, list at least 2 other types of risk in software products. Why are they important to consider? (QA perspective)** From a quality assurance perspective, two major risks beyond physical harm are *data integrity risk* and *maintainability risk*. Data integrity risk means that logs or sensor readings might be lost, corrupted, or inconsistent — which can make debugging almost impossible. If the system fails but there's no reliable data record, identifying the cause becomes guesswork. Maintainability risk comes from unclear code structure, missing documentation, or complex dependencies that make it hard to update safely. These are important to consider because a project that cannot be maintained or verified quickly becomes unsafe, even if it appears to "work." Good QA practices like regression testing, version control, and traceable requirements help prevent these issues. Recognizing these risks early will help us build a more dependable system over time.