Requirements Document

Version 2.0
11/25/2025

REACH
Reinforcement-learning for Enhanced Arm Control in Humans

Dr. Lerner & Dr. Cunha
Mentor: Bailey Hall

Taylor Davis, Victor Rodriguez, Clayton Ramsey, Lucas Larson

Accepted as baseline requirements for the project:

For the client:

Nov 26, 2025

Date

For the team:

CR TJD

Nov 26, 2025

Date

Table of Contents

Introduction

The assistive robotics industry focuses on developing technologies to enhance independence and quality of life for individuals with physical limitations. The global assistive technology market is valued at over $28 billion as of 2025, with robotics growing as a segment. A major area is upper-limb rehabilitation for stroke survivors, who often have persistent motor deficits affecting daily activities like eating, grooming, and reaching. Current solutions include prosthetics and exoskeletons, but they often rely on rigid, preprogrammed movements that struggle in dynamic real-world settings. At Northern Arizona University, Dr. Zach Lerner of Mechanical Engineering and Prof. Carlo R. da Cunha of Electrical Engineering lead research in biomechanics and assistive technologies. Their work bridges mechanical design, control systems, and human-machine interaction, positioning NAU as a hub for rehabilitation robotics. Dr. Lerner's biomechanics expertise informs the mechanical design and human factors, while Prof. da Cunha's control systems work addresses how robotic systems interpret and respond to human intent. Their collaboration aims to address the gap between rigid robotic control and natural human movement.

The REACH project addresses this by developing a simulation-based reinforcement learning framework that enables a robotic arm to learn adaptive, responsive behaviors for assistive tasks. Unlike hardcoded approaches, REACH uses reinforcement learning so the arm learns through trial and error, improving performance over time. The system models the robotic arm and tasks in the MuJoCo physics simulator, trains control policies using algorithms like Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC), and provides a modular framework that supports future task and hardware revisions.

The ultimate goal is a wearable robotic arm that can recognize human gestures, respond appropriately, and assist with daily tasks in a natural and energy-efficient manner. By integrating computer vision (YOLO) for hand detection and gesture recognition, the system will detect when a user raises a hand for a high-five and respond with correctly timed motion. This adaptive, learning-based approach marks a step toward assistive robotics that can seamlessly assist users in real-world environments.

Problem Statement

At Northern Arizona University, the Biomechatronics lab, led by Dr. Zach Lerner, focuses on developing robotic systems that enhance upper-limb mobility and rehabilitation. Their workflow centers on designing, testing, and refining robotic devices that help individuals recover movement and independence after injury or illness.

Current Workflow Overview:

1. Biomechanical Analysis:
   Dr. Lerner's team collects and analyzes human motion data from stroke survivors or healthy subjects performing key upper-limb tasks (e.g., reaching, grasping).

2. Mechanical and Control Design:
   The mechanical team develops prototypes, often exoskeletons or robotic arms, based on this data, while the controls team designs algorithms to drive these devices safely and effectively.

3. Hardware Prototyping and Testing:
   New control strategies are implemented directly on physical hardware. Testing occurs in controlled lab settings with human subjects or mechanical surrogates.

4. Evaluation and Iteration:
   Data from hardware trials is analyzed to improve system response, comfort, and adaptability, before returning to step 2 for another round of design refinement.

Current rehabilitation robotics research at NAU faces a significant bottleneck in control system development. Modern assistive devices often rely on rigid, preprogrammed control schemes that cannot easily adapt to dynamic, real-world human movement. Because the learning and testing of new control algorithms occur primarily on physical prototypes, each experimental iteration is slow, labor-intensive, and sometimes risky. Without a robust simulation-based learning platform, progress in developing natural, responsive, and safe robotic behaviors is limited by both hardware constraints and human resource availability.

Key Pain Points and Deficiencies:

- Manual Control Tuning:
  Each new task requires the team to manually reprogram or recalibrate control parameters. This limits scalability and delays research progress.
- Limited Adaptability:
  Existing robotic systems cannot autonomously learn or generalize new motions from experience, they depend on deterministic, rule-based programming that struggles with variations in human motion.
- Safety and Cost Concerns:
  Early-stage algorithm testing on physical hardware risks damaging expensive

equipment or causing unsafe movements when learning-based controllers behave unpredictably.

- Slow Experimentation Cycle:
  Because testing requires physical setup, each iteration can take hours or days, reducing research throughput.
- Fragmented Integration:
  Mechanical, control, and vision modules often operate independently, with limited shared simulation or testing environments. This makes it difficult to prototype end-to-end assistive behaviors that combine perception and actuation.
- Lack of Scalable Training Environment:
  The absence of a high-fidelity, physics-based simulation environment means researchers cannot perform rapid, parallel experiments, crucial for modern machine learning and control development.

In summary, the core problem is that NAU's current assistive robotics development process is constrained by its dependence on physical prototyping and manually tuned control strategies. This slows innovation, limits adaptive behavior learning, and increases the cost and risk of experimentation.

Solution Vision

The REACH system provides a modular, simulation-based reinforcement learning platform for developing control policies for an assistive robotic arm. Its primary goal is to train the arm to perform cooperative gestures, such as high-fives and handshakes, with a simulated human counterpart. These socially meaningful actions serve as benchmarks for testing perception, coordination, and adaptive control, which will later generalize to more complex rehabilitation and assistive tasks.

At a high level, REACH integrates four main subsystems that interact through structured data and control flow:

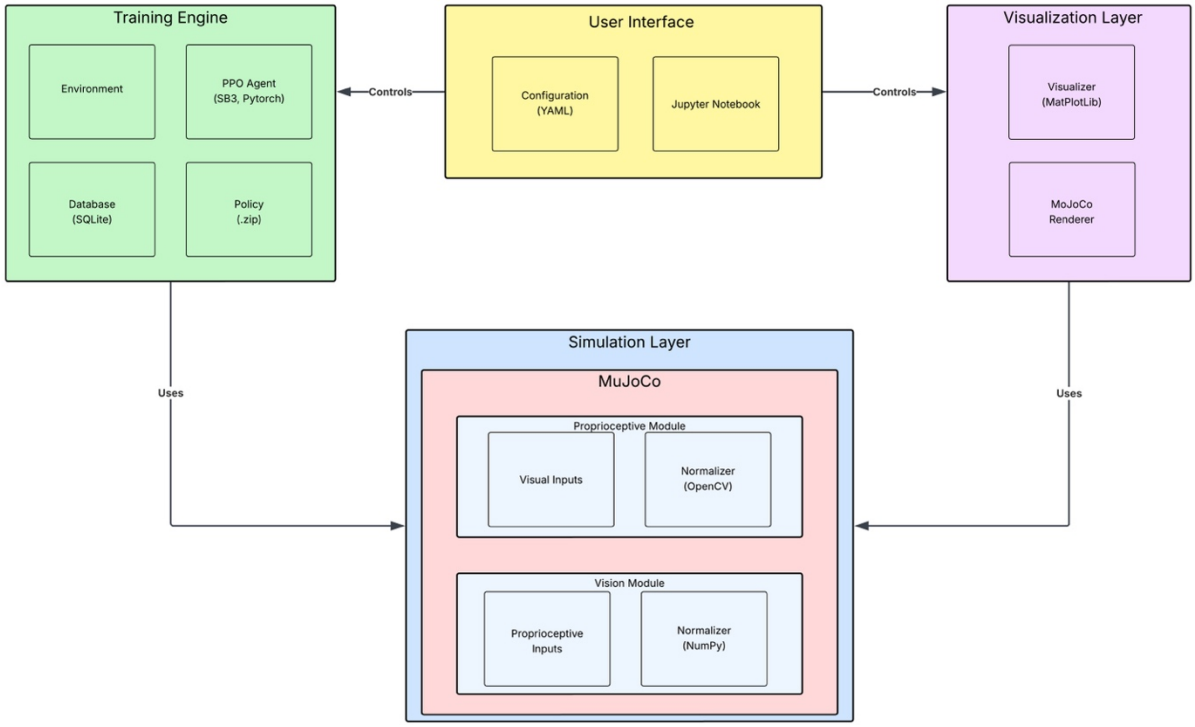
1. User Interface Layer – Provides high-level experiment configuration and control through YAML configuration files and Jupyter notebooks. This layer allows researchers to select environments, policies, and visualization options without modifying source code.
2. Training Engine – Implements the reinforcement learning workflow using Stable-Baselines3 (SB3) and PyTorch. It manages the environment, policy optimization, and data logging. Trained policies are saved as .zip files for reuse and evaluation.
3. Simulation Layer – Built on the MuJoCo physics engine, this layer models the robotic arm and its environment under realistic physical constraints. It contains two complementary sensing modules:
    - Vision Module – Processes visual input using OpenCV and related tools to interpret the position and orientation of the simulated human hand.

- Proprioceptive Module – Tracks internal joint states and kinematics of the robotic arm using NumPy-based normalizers.

4. Visualization Layer – Provides real-time visual feedback through the MuJoCo Viewer, allowing users to observe both model-only simulations and policy-driven demonstrations. This layer helps evaluate training results and system performance.

Training and evaluation occur entirely within simulation, enabling the system to develop robust and safe control policies before deployment on physical hardware. The learned behaviors will later be transferred to the real assistive robotic arm being developed by the Mechanical and Electrical Engineering teams through a simulation-to-real (sim-to-real) transfer process.

At a high level, the system is modeled in the diagram below.



From a user perspective, REACH will enable researchers and developers to visualize, test, and refine robotic behaviors in a controlled and repeatable environment. The system's modular structure will support ongoing improvements, allowing new behaviors or sensing capabilities to be added as the physical design matures. Ultimately, REACH aims to bridge the gap between intelligent simulation and real-world rehabilitation robotics, providing a scalable platform for future innovation.

Requirements Overview

Before presenting the detailed requirements for this system, it is important to provide a brief overview of how these requirements were identified and how they are organized in this document. The requirements outlined in the following sections represent the formalized outcome of our ongoing process of client interviews, workflow analysis, hands-on prototyping, and iterative validation. Each requirement reflects specific needs discovered through conversations with the client as well as practical insights gained during early prototype demonstrations. As our capstone process emphasizes, the requirements specification is not simply a list of desired features, but a contract that captures the essential behaviors, constraints, and performance expectations that the final system must meet.

To support clarity and traceability, the requirements are presented using a progressive-deepening structure. We begin with high-level domain-level requirements that describe the overarching capabilities the client needs from the system. These broad needs then motivate a structured breakdown into functional requirements, prioritized using the MoSCoW method, followed by non-functional requirements and environmental constraints that the system must operate within. This organization ensures that the detailed requirements are clearly connected to the client's workflow challenges and the solution vision defined in the previous section. In this way, the requirements serve as the bridge between the problem we aim to solve and the concrete specifications that will guide development and acceptance of the final product.

Requirements

These requirements define what the REACH system must accomplish to fulfill its purpose. They establish the measurable goals and constraints that guide both design and implementation, ensuring the final product aligns with the sponsor's needs. These requirements are divided into three categories: functional requirements, which describe the specific behaviors and capabilities the system must exhibit; performance requirements, which outline the expected efficiency, accuracy, and responsiveness of the system; and environment requirements, which specify the hardware, software, and operational conditions necessary for proper deployment and testing.

Functional Requirements

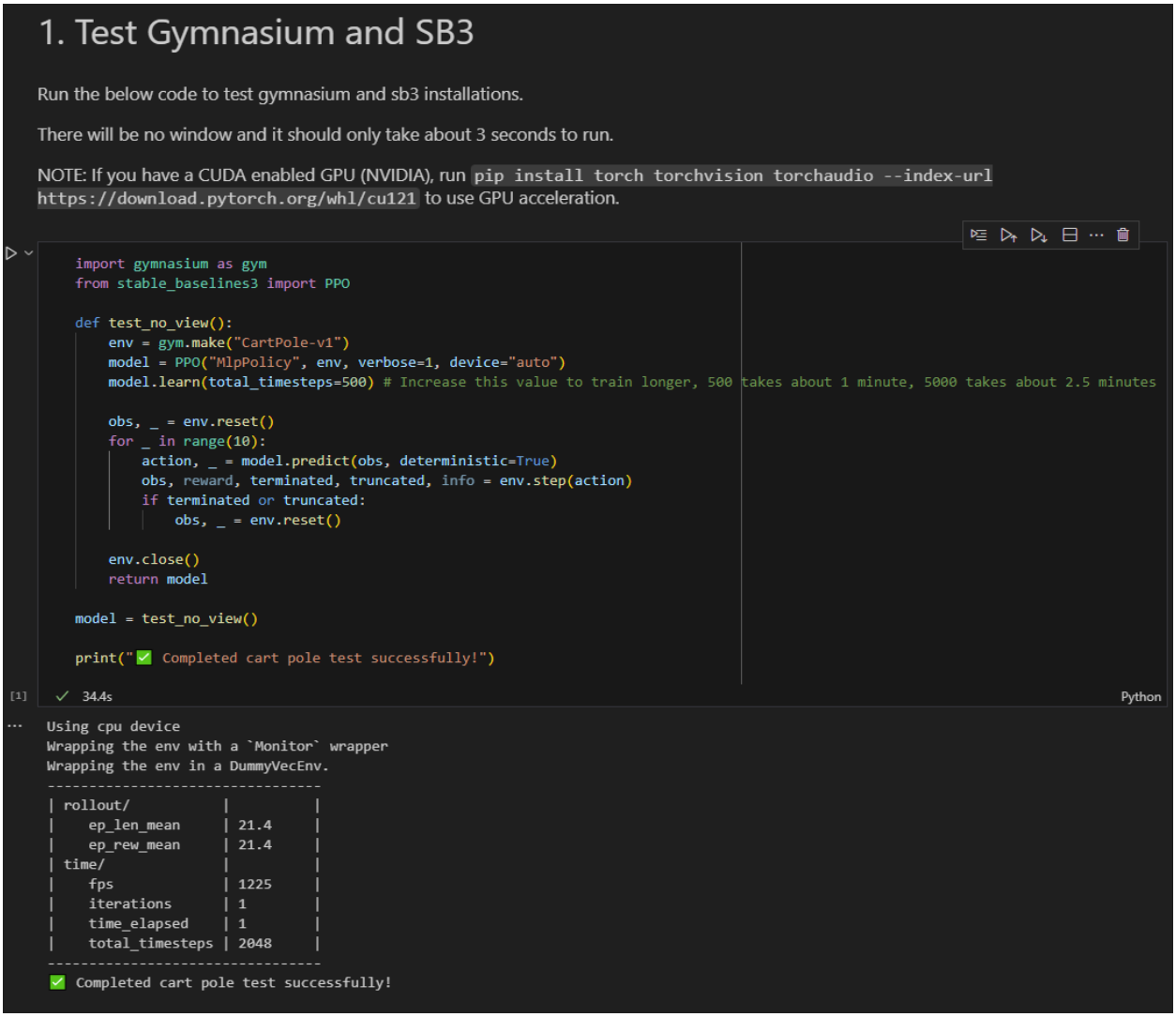
First we will describe the "Must Have" functional requirements for the system. These are the functional requirements that must be completed for our project to be considered complete.

Complete Researcher Interface Using Jupyter Notebooks and Configuration Files
Priority Level: MUST

“As a graduate student researcher, I want to conduct all experiments through Jupyter notebooks and YAML configuration files, so that I can work effectively without modifying core code or risking system breakage.”

Provide pre-configured Jupyter notebooks for all major research workflows: environment testing and visualization, trained policy analysis, reward function experimentation, training data analysis, and visualization for reports. Each notebook must include comprehensive markdown explanations of concepts, step-by-step instructions, expected outputs, and troubleshooting guidance All training hyperparameters, reward weights, task settings, environment parameters, and vision settings must be configurable through YAML files without code modification. Configuration files must include inline comments explaining each parameter's purpose, acceptable value ranges, and impact on training System must provide configuration validation that catches invalid settings before training begins, with clear error messages guiding users to correct values Support for multiple experiment configurations that researchers can copy and modify. Enable parameter modification through notebook variables and configuration files without touching source code in src/ directory. Include comprehensive error handling with actionable error messages that guide non-expert users to solutions.

Prototype:



1. Test Gymnasium and SB3

Run the below code to test gymnasium and sb3 installations.

There will be no window and it should only take about 3 seconds to run.

NOTE: If you have a CUDA enabled GPU (NVIDIA), run `pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121` to use GPU acceleration.

```python
import gymnasium as gym
from stable_baselines3 import PPO

def test_no_view():
    env = gym.make("CartPole-v1")
    model = PPO("MlpPolicy", env, verbose=1, device="auto")
    model.learn(total_timesteps=500) # Increase this value to train longer, 500 takes about 1 minute, 5000 takes about 2.5 minutes

    obs, _ = env.reset()
    for _ in range(10):
        action, _ = model.predict(obs, deterministic=True)
        obs, reward, terminated, truncated, info = env.step(action)
        if terminated or truncated:
            obs, _ = env.reset()

    env.close()
    return model

model = test_no_view()

print("✅ Completed cart pole test successfully!")
```

[1] ✓ 34.4s Python

```
Using cpu device
Wrapping the env with a `Monitor` wrapper
Wrapping the env in a DummyVecEnv.
---------------------------------
| rollout/           |          |
|    ep_len_mean     | 21.4     |
|    ep_rew_mean     | 21.4     |
| time/              |          |
|    fps             | 1225     |
|    iterations      | 1        |
|    time_elapsed    | 1        |
|    total_timesteps | 2048     |
---------------------------------
✅ Completed cart pole test successfully!
```

An example of a Jupyter notebook that demonstrates how global variables can be modified, but no source code changes are needed to easily read and run the code.

Acceptance Criteria:
Researchers can complete the entire research workflow using only notebooks and configuration files. All notebook examples run without errors and include clear expected outputs. The configuration system allows modification of all training parameters without code access. Invalid configurations produce descriptive error messages within 5 seconds. New users can create experiment configurations by copying and modifying examples. Error messages guide users to solutions without requiring code inspection.

MuJoCo Simulation Environment with Robotic Arm and Training Framework

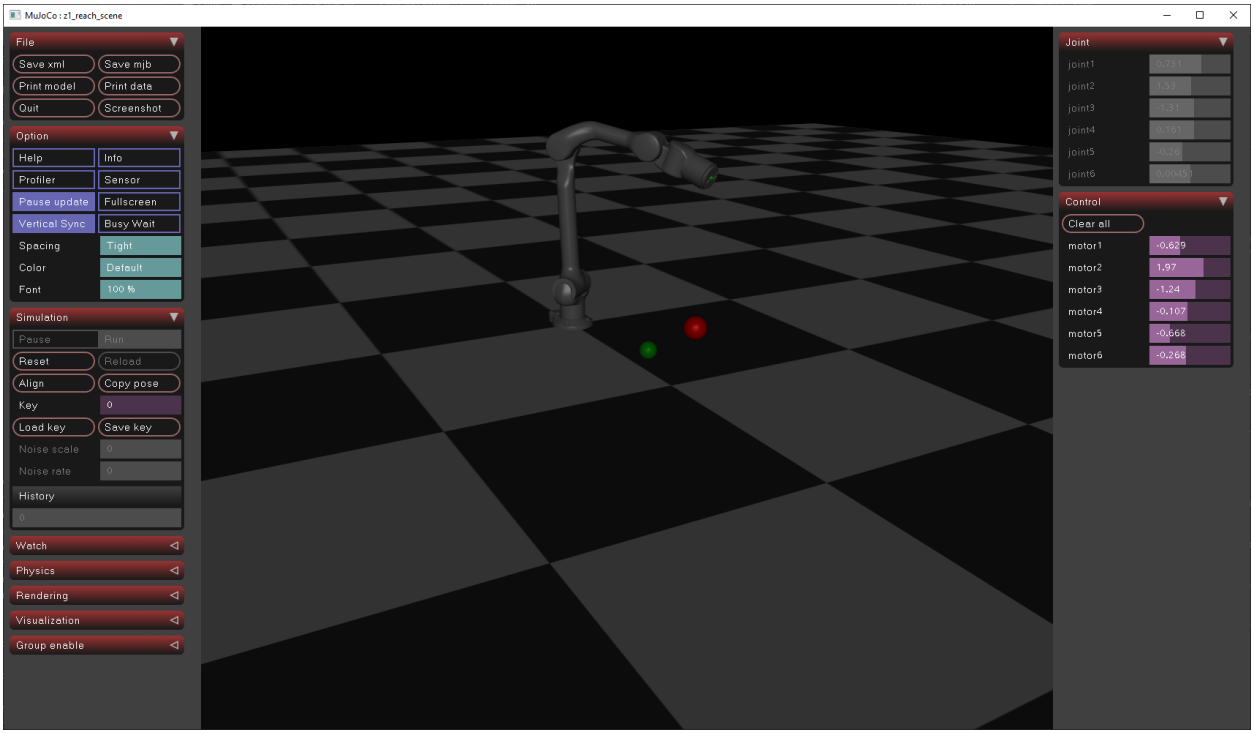
Priority Level: MUST

"As a researcher, I want a physics-accurate simulation environment with a robotic arm model that I can visualize, test, and train policies on, so that I can conduct reinforcement learning experiments before deploying to real hardware."

Model the robotic arm in MuJoCo XML format with accurate kinematics matching final ME team specifications including 6-7 degrees of freedom, proper segment lengths, joint limits, and mass properties. Include human arm models for collaborative scenarios and gesture interaction testing. Support waist-mounted attachment point as specified by ME team design. Implement physics constraints including joint limits, damping, friction, and collision detection. Provide sensor simulation including joint position sensors, joint velocity sensors, end-effector position tracking, and tactile/force feedback capabilities. Implement Gymnasium-compatible environment interface that follows standard RL environment protocols. Support visualization through Jupyter notebooks, standalone visualization scripts, and MuJoCo viewer. Implement basic reaching tasks with configurable reward functions that include distance-to-target rewards, energy consumption penalties, smoothness penalties, and success bonuses. Support random target generation within configurable workspace bounds with 3D workspace defined in configuration. Implement episode termination based on success threshold with configurable distance tolerance and timeout limits. Provide training script that accepts configuration file path and supports both local training and Monsoon HPC deployment via SLURM. Implement PPO and SAC agent wrappers using Stable-Baselines3 for policy training. Support automatic logging of training metrics to TensorBoard for real-time monitoring. Implement model checkpoint saving at regular intervals with resume capability. Support parallel environment training for faster data collection.

Current Implementation Status:
MuJoCo model structure created with sensor definitions. The base environment class template exists with the Gymnasium interface structure. Task framework created with reward structure template. The configuration system includes environment settings, task parameters, workspace bounds, and reward weights. The training script template exists with a command-line interface. SLURM script created for Monsoon HPC deployment. Agent wrappers structured with SB3 integration points. Logging framework configured for TensorBoard integration. Visualization script exists. Model documentation provided. Arm model refinement to match ME team specifications in progress.

Prototype:



Acceptance Criteria:
Arm model loads and renders correctly in simulation with all sensors functioning. The environment follows Gymnasium interface standard and integrates with Stable-Baselines3. Researchers can visualize arm behavior and sensor data through notebooks without code modifications. Reaching tasks achieves ≥80% success rate after training (1M timesteps). Training can be initiated with a single command using a configuration file. Training progress visible in TensorBoard with reward curves, success rates, and other metrics. Model checkpoints save automatically and can resume training. The system supports parallel environment training for faster data collection. All task parameters including workspace bounds, success threshold, and reward weights configurable via YAML files.

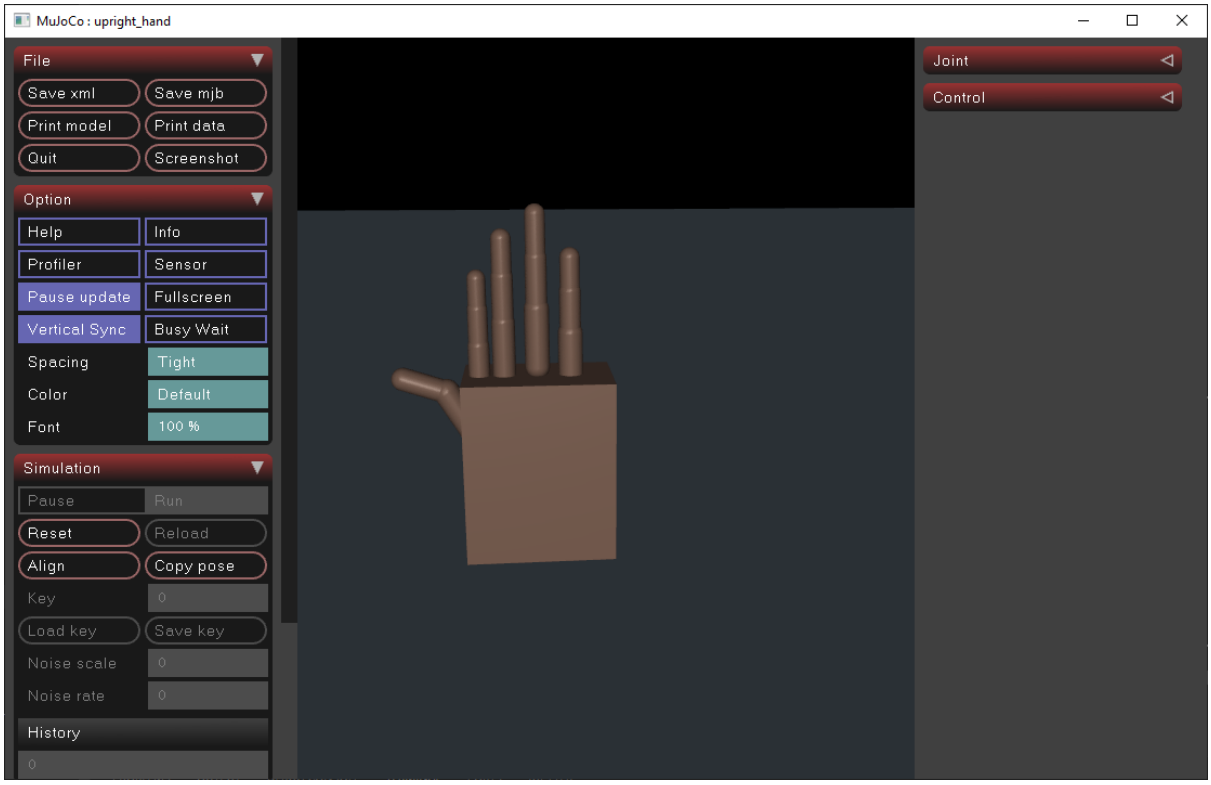
Gesture Recognition and Response
Priority Level: MUST

"As a user, I want the simulated robot arm to recognize when I raise my hand for a high-five, so that it can respond with a correctly timed motion."

Detect human hand position and orientation in the simulation environment using computer vision. Classify hand gestures including raised hand for high-five and extended hand for handshake. Generate appropriate motion plan based on detected gesture type. Execute robotic arm motion with correct timing and trajectory alignment to match human gesture. Support visualization of gesture detection process showing detected hand position, gesture classification, and arm response trajectory. Provide configurable gesture types and response behaviors through configuration files. Enable testing of gesture recognition through interactive interfaces without code modification.

Current Implementation Status:
Vision integration framework established with hand detection capabilities. Camera simulation module supports MuJoCo camera feeds for gesture detection. Gesture classification and motion planning framework designed. Integration with reinforcement learning policy for adaptive gesture responses planned.

Prototype:



Example of a hand in the “high-five” position simulated in Mujoco. This will ideally show a CV bounding box around the hand showing that it has been correctly identified.

Acceptance Criteria:
System detects hand position within 5cm accuracy of actual position. Gesture classification achieves ≥85% accuracy for high-five and handshake gestures. Motion execution completes within 2 seconds of gesture detection. Arm successfully aligns end-effector with hand position for a high-five gesture. Researchers can test gestures through configuration without modifying code.

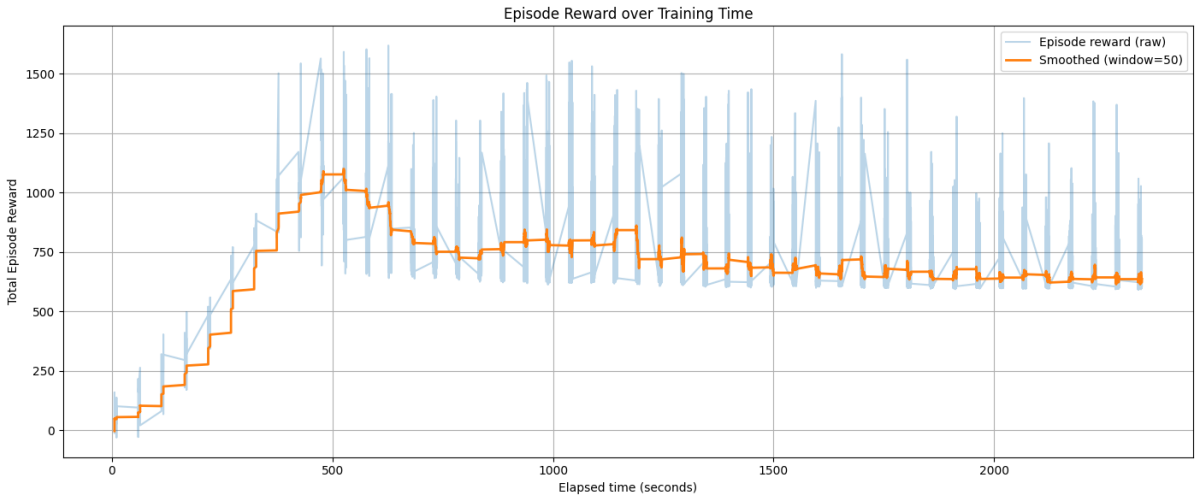
Reinforcement Learning Training Loop
Priority Level: MUST

"As a developer, I want the simulation to train a policy that learns from the success or failure of gestures, so that the arm's movements improve over time."

Implement Proximal Policy Optimization (PPO) training algorithm using Stable-Baselines3 framework. Support Soft Actor-Critic (SAC) as an alternative training algorithm. Collect trajectory data from environment interactions including state observations, actions taken, rewards received, and episode outcomes. Update policy network based on reward signals and advantage estimation. Implement value function estimation for baseline subtraction and variance reduction. Support automatic model checkpoint saving at regular intervals during training. Enable training resume from saved checkpoints without losing progress. Provide comprehensive logging of training metrics including episode rewards, success rates, policy loss, value loss, and entropy. Support training on local machines and high-performance computing clusters via SLURM job submission. Implement parallel environment training for faster data collection and improved sample efficiency. Support configuration-based hyperparameter tuning without code modification.

Current Implementation Status:
PPO agent wrapper framework established with Stable-Baselines3 integration. SAC agent wrapper framework designed for alternative algorithm support. Training script interface accepts configuration files for experiment management. Checkpoint callback system planned for automatic model saving. Evaluation callback framework designed for periodic policy assessment. Logging infrastructure configured for TensorBoard integration. SLURM job script template created for Monsoon HPC deployment.

Prototype:



Here we see the training reward over time on a chart. We can see the policy converges at about 500 seconds after starting. This is a great start to smooth policy training.

Acceptance Criteria:
The training loop successfully runs on both local machines and Monsoon HPC cluster. Policy shows measurable improvement in average reward over training with monotonic increase for at least the first 500k timesteps. Model checkpoints save and load correctly, enabling training resumes. Training metrics logged and accessible via TensorBoard or similar visualization tool. The system achieves >80% task success rate after 1 million timesteps of training. Hyperparameters configurable through YAML configuration files without code changes.

Vision Integration System
Priority Level: SHOULD

"As a user, I want the arm's camera to detect the human hand position, so that the motion planner can target accurate alignment for gestures."

Integrate YOLO object detection model (YOLOv8) for hand and object detection in camera feeds. Implement camera simulation within the MuJoCo environment providing RGB and depth image feeds. Extract hand position coordinates from camera feed using bounding box detection. Estimate 3D hand position using depth information from camera for accurate world coordinates. Transform camera coordinates to world/arm coordinate system for motion planning. Pass detected hand position and orientation to

RL agent as observation inputs. Support configurable vision enable/disable through configuration flags. Provide visualization of camera feed with overlaid detection bounding boxes and 3D position estimates. Support real-time processing at ≥30 fps for responsive interaction. Enable integration with Oak-D Pro spatial AI camera for future real-world deployment.

Current Implementation Status:
YOLO detector framework established with hand detection capabilities. The camera simulation module provides MuJoCo camera integration. Depth-aware detection method implemented for 3D position estimation. Coordinate transformation framework designed for camera-to-world conversion. The configuration system supports vision toggle and parameter settings. GPU acceleration support planned for real-time processing.

Acceptance Criteria:
Hand detection achieves ≥80% confidence in the simulation environment. Position detection accuracy within 10cm of actual hand position in 95% of detections. Camera feed processes at ≥30 fps without blocking control loop. Coordinate transformation accuracy validated against known test positions. Vision systems can be enabled/disabled via configuration without code modification. Detection results passed correctly to the motion planner for gesture targeting.

Visualization Dashboard
Priority Level: SHOULD

"As a researcher, I want to view graphs of reward curves and joint movements, so that I can analyze the performance and stability of the training process."

Display real-time reward curves during training showing episode rewards, average rewards, and reward trends over time. Visualize joint angle trajectories over time for all arm joints during episodes. Show end-effector position tracking in 3D space over episode duration. Provide training statistics including current success rate, average episode length, energy consumption trends, and policy entropy. Support visualization of reward component breakdown including distance reward, energy penalty, smoothness penalty, and success bonus. Enable export of all graphs and visualizations in publication-ready formats including PNG, PDF, and CSV. Integrate with TensorBoard for real-time metric visualization during training. Provide interactive policy analysis tools for exploring specific episodes and trajectories. Support comparison visualization of

multiple trained models side-by-side. Display training progress indicators including training time, steps completed, and estimated time to completion.

Current Implementation Status:
Monitoring dashboard framework established for metric visualization. TensorBoard logging integration configured for real-time training visualization. Logging utilities support comprehensive metric collection. Visualization notebook templates designed for interactive analysis. Dashboard visualization components and automatic metric logging implementation planned.

Acceptance Criteria:
Dashboard updates in real-time during training with metrics refreshing at least once per second. All key metrics displayed and accessible including reward curves, success rates, joint trajectories, and energy consumption. Graphs exportable in common formats (PNG, PDF, CSV) for reports and publications. Researchers can access the dashboard without code modifications through standard interface. Visualization generates within 30 seconds for responsive interactive analysis. Multiple models can be compared side-by-side with statistical metrics.

Simulation Reset and Replay Controls
Priority Level: SHOULD

"As a developer, I want to reset and replay episodes, so that I can test different learning parameters without restarting the entire environment."

Implement environment reset functionality that returns simulation to initial state with configurable randomization. Support random target generation within workspace bounds on each reset. Save episode trajectories including state history, action sequence, reward sequence, and outcome. Provide controls to replay saved episodes with exact reproduction of original episodes. Support manual reset during active simulations for immediate testing. Maintain episode history and statistics including success/failure counts and average performance. Enable reset with different random seeds for testing reproducibility. Support reset to specific saved states for debugging and analysis. Provide episode comparison tools for analyzing differences between successful and failed episodes.

Current Implementation Status:

Environment reset framework established with Gymnasium-compatible reset interface. Episode management system designed for trajectory tracking and history maintenance. Reset functionality template includes state initialization and target randomization. Episode replay framework planned for trajectory reproduction. Manual reset controls and episode history management implementation in progress.

Acceptance Criteria:
Reset completes within 100ms for responsive workflow. Episode replay accurately reproduces the original episode with exact state and action sequence. Manual reset works during active simulations without system errors. Episode history is maintained correctly with statistics and metadata. Researchers can reset and replay episodes through the interface without code modification. Multiple episodes can be compared side-by-side for analysis.

Multiple Simulation Environments and Advanced Features
Priority Level: SHOULD

"As a researcher, I want to train policies in multiple environments (kitchen, bathroom sink, table) and use advanced features like vision integration and gesture recognition, so that I can test policy generalization and explore more complex interaction scenarios."

Implement pre-configured simulation environments for kitchen scenarios including cups, utensils, countertops, and typical kitchen objects. Implement bathroom sink environments with toothbrush, faucet, mirror, and sink objects. Implement table environment for seated tasks with table-top objects and reachable workspace. Support environment switching via configuration file selection without code modifications. Ensure shared observation/action space across environments for policy transfer testing. Implement vision integration with YOLO object detection that can be enabled/disabled via configuration flag. Support hand detection and object detection through camera simulation with 3D position estimation using depth information. Implement gesture recognition for high-five and handshake gestures with configurable response behaviors. Provide visualization of gesture detection and arm response for debugging and analysis. Include environment-specific documentation and examples for each scenario.

Current Implementation Status:
Modular task framework exists designed for multiple environments. Environment factory pattern implemented for environment creation from config. Vision module structure

exists with detection, depth fusion, and 3D position estimation methods. Camera simulation module created for MuJoCo camera integration. The configuration system includes vision settings with toggle flags. Specific environment models and gesture recognition implementation planned.

Acceptance Criteria:
Researchers can switch between at least 3 different environments by changing config files. Vision modules can be enabled/disabled via configuration without code changes. Gesture recognition achieves ≥85% accuracy for high-five and handshake detection. The arm responds appropriately to detected gestures within 2 seconds. All environments share compatible observation/action spaces for policy transfer. Environment-specific documentation and examples provided for each scenario.

Visualization Dashboard and Training Monitoring
Priority Level: SHOULD

"As a researcher, I want a real-time visualization dashboard showing training metrics, so that I can monitor training progress, identify issues early, and make informed decisions about when to stop training or adjust parameters."

Implement real-time TensorBoard integration displaying reward curves over training, success rate trends, average episode length, and other key metrics. Provide visualization of joint angle trajectories during training episodes. Display end-effector position tracking over time. Show training statistics including current success rate, average reward, energy consumption trends, and policy entropy. Support export capabilities for all metrics and visualizations in formats suitable for reports (PNG, PDF, CSV). Include automated metric logging that updates the dashboard during training. Provide alerts for training issues such as reward plateaus, divergence, or failure to learn.

Current Implementation Status:
Monitoring module structure exists. TensorBoard logging framework configured in training system. Logging utilities exist. Dashboard visualization components and automatic metric logging implementation planned.

Acceptance Criteria:

Dashboard updates in real-time during training with metrics refreshing at least once per second. Researchers can access the dashboard without code modifications through the standard TensorBoard interface. All key metrics displayed and accessible including reward curves, success rates, and joint trajectories. Visualizations exportable in publication-ready formats. Training issues automatically detected and reported.

Domain Randomization and Energy Optimization
Priority Level: COULD

"As an advanced researcher, I want to enable domain randomization for improved sim-to-real transfer and energy-efficient policies, so that trained policies generalize better to real-world conditions and are suitable for battery-powered devices."

Implement configurable domain randomization parameters including physics randomization with friction, mass, and damping variations. Support visual randomization including lighting conditions and textures. Enable task parameter randomization for target positions and workspace bounds. Support gradual randomization schedules that increase diversity over training. Implement energy consumption calculation from joint torques and velocities. Include energy penalty in reward function with configurable weight. Track and report energy consumption metrics during training. Provide documentation on when and how to use domain randomization effectively.

Current Implementation Status:
A wrapper system exists for environment modifications. Energy weight parameter included in reward configuration. Reward framework supports multi-objective reward composition. Domain randomization wrapper and power calculation implementation planned.

Acceptance Criteria:
Domain randomization can be enabled via configuration without code changes. Trained policies show ≥20% reduction in energy consumption compared to baseline. Randomization improves policy generalization as measured by performance across varied test conditions. Energy metrics tracked and reported throughout training.

Real Hardware Deployment
Priority Level: COULD

“As a researcher, I want the trained policies to be deployable on the physical robotic arm in future semesters, so that the simulation framework remains compatible with real-world implementation.”

Although physical deployment is **not required** for this project phase, the system should be designed with future real-hardware integration in mind. The simulation framework must support exporting trained policies in formats compatible with embedded platforms such as the NVIDIA Jetson Orin Nano. The architecture should maintain consistent observation and action spaces so that policies trained in simulation can later be executed on the physical arm without major restructuring. Model interfaces must follow clear modular boundaries so that hardware modules — such as joint encoders, actuators, and the Oak-D Pro camera — can be integrated in subsequent work. Basic ROS messaging and control-loop compatibility should be considered to reduce integration friction for future teams. Documentation must clearly describe how the simulation environment maps to expected hardware signals, including joint ordering, limits, and coordinate frames.

Current Implementation Status:

Foundational simulation interfaces designed with hardware compatibility in mind. Observation/action spaces use consistent formatting suitable for later deployment. Reward structure and control frequency aligned with typical embedded robotics constraints. ROS communication design and hardware driver integration planned for future teams.

Acceptance Criteria:

Simulation-trainable policies exportable in a portable model format (e.g., PyTorch .pt or SB3 .zip). Observation and action spaces documented to align with ME/EE hardware specifications. Control-loop frequency configurable to match hardware execution rates. Documentation included describing how future teams can connect hardware sensor streams and actuator outputs to the existing control framework.

Performance Requirements

Performance requirements specify measurable performance characteristics the system must meet for effective researcher use.

Researcher Usability and Onboarding

Researchers with basic Python knowledge must be able to complete their first successful experiment within 1 week of initial setup following the getting started guide Common research tasks must require less than 3 commands or notebook cells: training initiation, model evaluation, and result visualization. Error messages must be clear and actionable, guiding users to solutions without requiring code inspection or deep

technical knowledge. Configuration validation must catch invalid settings within 5 seconds of training start with specific guidance on correction.

Verification Method: Test with 5 graduate students having basic Python knowledge but no prior RL experience, measure time to first successful experiment, count questions asked, and evaluate error message clarity.

Current Status: Getting started guide provides step-by-step instructions targeting 2-hour onboarding goal.

Simulation Performance
Simulation must run at ≥60 frames per second (fps) in headless mode on standard hardware for efficient training data collection. Visualization mode must maintain ≥30 fps for smooth, real-time rendering during interactive testing. Physics timestep must be configurable with default supporting stable simulation at 500 Hz (0.002s timestep).

Verification Method: Benchmark simulation with 1000 timesteps, measure average fps across 10 runs, test notebook responsiveness with various visualization operations.

Current Status: MuJoCo timestep configured at 0.002s (500 Hz) in simple_arm.xml, supporting high-frequency simulation.

Training Performance
Training must process at least 1000 environment steps per second on Monsoon HPC with at least 8 parallel environments for efficient learning. Researchers must be able to start training on Monsoon HPC with a single command (sbatch scripts/slurm_train.sh) after initial one-time setup. Training progress should be visible without SSH access through TensorBoard logs accessible via web interface.

Verification Method: Measure training throughput over 10,000 steps on Monsoon, verify SLURM script execution, confirm TensorBoard log accessibility.

Current Status: Configuration supports parallel environments (n_envs: 4), SLURM script exists in scripts/slurm_train.sh for Monsoon deployment.

Model Performance and Learning

Trained policy must achieve >80% task success rate on reaching task after 3 million timesteps of training for validation of learning framework. Training should converge to stable performance (success rate plateau with <5% variation) within 1 million timesteps for basic reaching tasks. Gesture recognition accuracy must be ≥85% for high-five detection when vision integration is implemented. Policies must show measurable improvement over training (monotonic increase in success rate for at least first 500k timesteps).

Verification Method: Evaluate trained model on 100 independent test episodes, compute success rate and confidence intervals, analyze training curves for convergence behavior, test gesture detection on 100 test cases.

Current Status: Success threshold defined in config (success_threshold: 0.05 meters), evaluation framework planned, training metrics logging configured.

Vision System Performance (When Implemented)
Hand detection must complete within 33ms (≥30 fps processing) for real-time operation without blocking control loop. Position estimation accuracy must be within 10cm of actual hand position in 95% of detections for reliable motion planning. Camera feed processing must not introduce >100ms latency to control loop to maintain responsive interaction. Detection confidence threshold must be configurable with default ensuring ≥80% precision (true positive rate).

Verification Method: Benchmark detection pipeline with 1000 test frames, measure processing time distribution, validate position accuracy against ground truth, test control loop latency.

Analysis and Evaluation Performance
Model evaluation on 100 episodes must complete within 10 minutes on standard hardware for practical workflow. Notebook visualizations must generate within 30 seconds for responsive interactive analysis. Policy analysis must provide comprehensive results without requiring code modifications, accessible through notebook interface. Metric calculations must be deterministic and reproducible across runs with same random seed.

Verification Method: Benchmark evaluation script execution time, measure notebook cell execution times, test reproducibility across 5 runs, verify no code changes required for analysis.

Current Status: Evaluation script template exists, analysis notebooks framework established, visualization tools planned.

System Resource Usage
Training logs must not exceed 10GB per training run (1M timesteps) to manage storage on Monsoon HPC. System must support training runs up to 10 million timesteps without memory leaks or degradation. Configuration files must be ≤1MB each to support easy sharing and version control.

Verification Method: Monitor log file sizes during extended training runs, measure checkpoint file sizes, test 10M timestep training for memory stability, verify config file sizes.

Current Status: Logging system configured with checkpoint frequency (checkpoint_freq: 50000), storage management planned, config files currently well within size limits.

Environmental Requirements

Environmental requirements specify constraints, dependencies, and integration needs imposed by the client, hardware, and research environment.

Target User Environment
The primary users of this project are graduate students in Dr. Lerner's Biomechatronics Lab and Prof. da Cunha's research lab at Northern Arizona University. These users should have basic Python programming skills, experience using Jupyter notebooks for interactive computing, and a foundational understanding of robotics concepts such as joints, actuators, and sensors, regardless if they are experts in reinforcement learning or deep learning or not. The system should be designed so that users can run experiments and make modifications through configuration files and notebooks without editing core implementation code in the src/ directory. The expected development environment is macOS and Linux, aligning with the current team setup and Monsoon HPC requirements.

Hardware Platform Requirements

The system will be developed on macOS and Linux machines equipped with multi-core processors (4 or more cores recommended) to support parallel environment training, while large-scale training runs will utilize the NAU Monsoon HPC cluster, where access to a CUDA-compatible GPU is preferred but not required for basic functionality. A future stretch goal for deployment includes running real-time policy execution on an NVIDIA Jetson Orin Nano for control of the physical robot, so the system architecture should be designed with this eventual integration in mind, despite it not being a requirement. The hardware design must also remain compatible with the final robotic arm specifications, joint configs, link lengths, etc, which will be defined through ongoing collaboration with the Mechanical and Electrical Engineering teams.

Software Dependencies and Compatibility

The project requires Python 3.9 or higher with clear installation instructions and dependency management through a requirements.txt file, which mandates the use of Jupyter Notebook/Lab for interactive researcher workflows. Core software dependencies include the MuJoCo 2.3+ physics engine, PyTorch for neural network implementation, Stable-Baselines3 for PPO/SAC algorithms, and Gymnasium as the standard RL environment interface. Some additional required packages include the Ultralytics YOLOv8 model for object and hand detection, NumPy and SciPy for numerical computations, and TensorBoard for logging and visualizing training metrics. All dependencies must be reproducible and installable with a single command “pip install -r requirements.txt” and with pinned versions specified.

Documentation and Usability Requirements

The project’s documentation must include a Getting Started Guide that enables new users to complete their first experiment, which includes covering installation, environment setup, configuration, an initial training run, and basic evaluation, within two hours without needing assistance. In addition, configuration documentation should provide inline comments for all YAML parameters, describing their purpose, acceptable ranges, typical values, and how they affect system behavior. Each Jupyter notebook must contain clear markdown explanations outlining its purpose with step-by-step instructions, expected outputs, and troubleshooting tips. API documentation is required for any functions researchers are likely to use, which include function signatures, parameter and return value descriptions for application in usage examples and common use cases. Additionally, a Troubleshooting Guide must address at least ten common issues such as installation, configuration, training, and visualization problems to go

along with clear, actionable solutions. Lastly, in order to keep confusion to a minimum, non-technical explanations for RL, MuJoCo, PPO, and reward functions should be created in order for users with basic Python knowledge but little to no ML experience to understand.

Safety, Error Prevention, and Code Protection

The system must validate all configuration parameters before training begins in order to be successful in detecting issues such as negative reward values, impossible workspace bounds, or incompatible settings within five seconds and provide clear, actionable error messages that guide non-expert users toward solutions without requiring code inspection or advanced debugging skills. To prevent accidental changes to core functionality, the implementation code in the src/ directory should be protected to ensure that researchers perform all experimentation exclusively through configuration files and notebooks.

Hardware Integration Requirements (Future/Planned)

Future hardware integration requirements include support for the Oak-D Pro spatial AI camera to enable real-world deployment and compatibility with joint encoder data, providing proprioceptive feedback aligned with Mechanical Engineering sensor specifications. The system may also require Robot Operating System (ROS) message formats if hardware communication relies on ROS, which will be confirmed in collaboration with the ME/EE teams in the future. Additionally, the architecture must be designed to allow future deployment on the NVIDIA Jetson Orin Nano for real-time execution, including model optimization and the necessary hardware interface support.

Research Workflow and Collaboration Support

The system must support a collaborative research workflow by ensuring full experiment reproducibility through saved configuration files, fixed random seeds, and comprehensive training logs. Researchers should be able to share results easily using saved models, configuration files, evaluation logs, and visualization outputs. While the codebase is version-controlled through Git/GitHub at github.com/lucaslarson25/reach, researchers are expected to work primarily with configuration files and notebooks rather than modifying core code. The workflow should enable multiple researchers to work in parallel without conflict between each other by using separate configuration files, experiment directories, and clear model naming conventions.

Compliance, Standards, and Reproducibility
The project must adhere to established standards for compliance and reproducibility, following PEP8 Python style guidelines as documented in CONTRIBUTING.md along with using Markdown for all technical documentation. Dependency management must ensure reproducibility by listing all required packages with specific version numbers in requirements.txt. Along with that, the system should run consistently across different development environments like macOS and Linux. Additionally, all third-party libraries and external code must be properly attributed in accordance with their respective licenses.

Network and Storage Infrastructure
The system must support access to the NAU Monsoon HPC cluster for large-scale training that involves SLURM job submission, Python and CUDA module loading, and efficient data transfer workflows. Local development machines should have enough storage (≥50GB recommended) available for models, training logs, evaluation results, and visualization outputs. Internet access is required to download pre-trained models (such as YOLO weights), Python packages, and documentation resources, and the workflow must support transferring large model files and logs between local environments and the Monsoon HPC cluster.

Potential Risks

As with any research-oriented development project, several risks could impact the success of REACH. These risks range from technical challenges to inter-team dependencies and integration issues with external systems. Identifying and mitigating these early helps ensure stable progress throughout the semester.

Hardware Dependency Risk
The physical robotic arm that is being developed by the Mechanical and Electrical Engineering teams for this framework has not yet been finalized. Our simulation framework for REACH is dependent on these designs for the arm and the parameters for its joint configuration, limits, and different link lengths. With this delay, the reliability of the framework cannot be tested or improved on, even with provided testing arms and parameters given via the simulation software MuJoCo. This could lead to delays in the completion of the framework or make the delivered product an unfinished mess within the time to deliver it. In order to work around this constraint, we are focusing on building the framework as adaptable as possible so that any parameters and designs given can easily be implemented and tested without further delays.

Sim-to-Real Transfer Risk
A long-term goal that we are working towards is transferring the control policies from our simulation into a physical robot. With this in mind, the idealistic parameters and situations are not as available in a real world environment. Physics, latency, and sensor noise are just a few things that a sim-to-real transfer has to deal with. These factors could cause the robot's training to worsen over time, alter the stress load of the robot without anyone's knowledge, or even degrade the functionality of it all, making the framework and robot unstable. To mitigate this risk, we have made the framework employ domain randomization techniques, along with continuous tuning, to design it in a way that allows learning policies to be more generalizable.

Reinforcement Learning Stability Risk
Our training of the simulation framework uses Reinforcement Learning (RL) methods in order to accurately simulate the training process of the framework. However, these methods can become very unstable during said training, particularly in high-dimensional and unexpected parameter-rich environments like robotic control. Any improper tuning or lack of environment exploration leads to things like repetitive behaviors that often result in failure, as well as lackluster convergence between the framework and the robot. Both could lead to unstable learning dynamics for both the framework's productivity and the robot's operability, which can become unsafe, jerky, and erratic. The team will be addressing the risk by implementing RL algorithms obtained through Stable Baselines3 and by using systematic performance monitoring to allow us to catch and swiftly correct instabilities early.

Integration and Communication Risk
REACH as a whole requires multiple and varying uses of communication between a variety of systems, including things like hardware interfaces, sensor data, and vision modules for environment detection. These widespread and layered streams of communication between systems bring up issues of compatibility or synchronization that could arise when these various components interact. Things like mismatching formats concerning data, where one system gives an answer that another system is expecting in a specific way that doesn't match, or timing, as these systems can be fast-acting or slow-acting, and can get bogged down waiting for a different system to catch up in order to progress. In order to mitigate these issues, the team will establish and enact clear interface protocols, along with testing routines, before beginning integration with the ME and EE teams' systems.

Project Scope and Time Management Risk
With an interdisciplinary project such as this and the multiple groups that are a part of it, the overall size of this project is a big concern, especially with the ambitious goals of machine learning and robotics integration. As the project progresses, many things could be added or removed as time passes, leading to an increasing scope for the project, the work it would require, and the number of possible problems down the road that wouldn't exist if the scope creep were accounted for. To combat this risk, the team will focus on achieving a functioning simulation that demonstrates the learned gestures needed for the minimum viable outcome by the end of the semester. Any goals, tasks, or advanced additions will only be pursued once the core objectives of the project have been completed ahead of schedule.

Project Plan

The REACH execution roadmap spans Fall 2025 through Spring 2026 and follows a phased cadence tied directly to the functional requirements and course deliverables. Each phase culminates in a milestone review with the clients and mentor to verify readiness before advancing. Week numbers refer to the CS476 calendar (Week 1 begins August 25, 2025).

Phase 1: Simulation Foundations (Weeks 1–6). The team finalizes the MuJoCo model, validates joint limits against the emerging ME/EE arm specifications, and brings the Gymnasium-compatible environment online with sensor streams and deterministic resets. By the close of Week 6 the scripted reaching task achieves at least 50% success under a baseline PPO configuration, providing the prerequisite for reinforcement learning work. Fall course checkpoints, proposal presentations, mentor meeting, and team standards updates, serve as progress gates for this phase.

Integration Buffer 1 (Week 7). A dedicated review week consolidates lessons learned, updates documentation, and secures sponsor sign-off before training infrastructure work proceeds.

Phase 2: Training Infrastructure (Weeks 7–12). Focus shifts to the reinforcement learning pipeline. The team hardens PPO and SAC wrappers, enables automatic checkpoint/resume behavior, and deploys the training scripts to the Monsoon HPC cluster via SLURM. TensorBoard logging is exposed for remote monitoring so researchers can track performance without direct cluster access. Success criteria include sustaining a 1M step run with at least eight parallel environments and attaining a 60% reach success rate. This phase aligns with midterm course deliverables such as the feasibility analysis update and peer evaluation.

Integration Buffer 2 (Week 13). The week before fall break is reserved for cross-team integration, updating the requirements document, and planning the winter work sprint.

Phase 3: Vision and Gesture Loop (Weeks 14–18, continuing through Winter Break to Week 23). The vision module is integrated with YOLO-based detection, synthetic datasets are expanded, and depth fusion produces hand position estimates within 10 cm accuracy. The gesture classifier then drives motion planning so the arm can respond to high-five and handshake cues in under two seconds. Winter break is leveraged for data labeling, detector tuning, and asynchronous experimentation. A mid-January review ensures the perception stack meets accuracy and latency targets before spring classes resume.

Integration Buffer 3 (Week 24). A short buffer absorbs holiday fallout, synchronizes with ME/EE teams, and locks the spring backlog.

Phase 4: Monitoring and Advanced Features (Weeks 24–30). Real-time dashboards visualize reward curves, joint trajectories, and energy usage; simulation reset and replay tooling captures full trajectories for debugging; and configuration-driven environment switching (kitchen, sink, table) enables robustness testing. Outputs must be exportable in publication-ready formats for upcoming design reviews and poster drafts. This phase runs in parallel with the course progress report, design walkthrough, and poster preparation milestones.

Phase 5: Validation and Documentation (Weeks 31–36). The team executes the acceptance test suite (100-episode evaluations, configuration validation, reproducibility checks), conducts domain randomization studies that target a 20% energy reduction relative to baseline, and finalizes all user-facing documentation. Sponsor sign-off occurs the first week of April, satisfying course final presentation and poster session deadlines.

The following Gantt chart reflects our schedule visually:

Critical Path. Three items remain gating factors for downstream work: completing the MuJoCo environment (Phase 1), achieving the first stable 1M-step training run (Phase 2), and integrating gesture recognition with the policy (Phase 3). Buffer weeks and mid-phase reviews provide contingency for unexpected integration issues or evolving hardware specifications. A Gantt chart accompanying this section visualizes the schedule, dependencies, and buffer placement for quick reference.

Conclusion

The REACH project aims to make meaningful progress in the field of rehabilitation robotics by combining reinforcement learning, computer vision, and simulation technology into one adaptive system. Through this work, the team seeks to lay the foundation for a robotic arm that can intuitively assist with upper-limb rehabilitation exercises and cooperative gestures.

This document outlined the key elements guiding the project's next phase, including the problem background, the motivation for a simulation-first approach, and the core requirements needed for success. The proposed system will focus on training a virtual robotic arm in MuJoCo to perform simple, socially meaningful movements like high-fives and handshakes in coordination with simulated human motion.

By prioritizing a robust simulation environment, the REACH team ensures that the foundation for later sim-to-real transfer is both technically sound and adaptable to ongoing hardware development. The identified risks, particularly those related to integration, learning stability, and hardware dependencies, will be actively managed through modular design, careful testing, and close collaboration with the Mechanical and Electrical Engineering teams.

As the semester continues, the REACH project will move toward developing functional prototypes, refining reinforcement learning methods, and establishing a verified requirements baseline for future expansion into real-world robotic systems. With this approach, the team is positioned to contribute a meaningful and technically grounded step forward in assistive robotics for rehabilitation.

Glossary

- NAU Monsoon HPC cluster: a Linux-based system with the SLURM workload manager.
- SLURM: an open-source workload manager and job scheduler used to manage high-performance computing clusters.
- HPC: High-Performance Computing.
- CUDA: Compute Unified Device Architecture; a parallel computing platform and programming model.
- StableBaselines3: an open-source Python library that provides reliable and high-quality implementations of deep reinforcement learning (RL) algorithms.
- MuJoCo: a physics engine for simulating articulated systems like robots, biomechanics, and animations.
- PyTorch: an open-source machine learning library primarily used for developing and training deep neural networks.

- Gymnasium: an open-source Python library for developing and comparing reinforcement learning (RL) algorithms by providing a standard API for RL environments.
- TensorBoard: an open-source visualization toolkit for machine learning that helps users track, visualize, and compare experiments and debug models.
- NumPy: an open-source Python library fundamental for numerical computing.
- SciPy: an open-source Python library used for scientific computing.
- Ultralytics YOLOv8: state-of-the-art, open-source computer vision model for tasks like object detection, segmentation, and pose estimation.
- PPO/SAC: modern algorithms used in reinforcement learning to train agents.
- YAML: a human-readable data serialization language commonly used for configuration files and data exchange.
- RL: a type of machine learning where an agent learns to make a sequence of decisions by taking actions in an environment to maximize a cumulative reward.