BOTIMUS

Full-Stack Robotics Rapid Prototyping

Full Design Review (FDR)

Maximus Shurr

maximusshurr.com | maxshurr@gmail.com

Purdue University | Honors Mechanical Engineering ‘26

1/28/2026

Executive Summary

Project Scope & Objectives

This Technical Design Review (TDR) documents the end-to-end engineering lifecycle of "BOTIMUS," a combat robotics platform developed under a strict 72-hour critical path constraint. The objective was to engineer a Minimum Viable Product (MVP) capable of competing against teams with significantly higher capitalization and development lead times. This project serves as a case study in "Extreme Constraint Innovation," demonstrating the application of rapid prototyping, physics-based simulation, and parallel execution strategies to bridge the resource gap.

Technical Methodology

Unlike traditional waterfall development, this project utilized a "Simulation-First" approach. By leveraging Python for kinematic analysis and torque derivation prior to CAD modeling, the design phase eliminated costly trial-and-error loops. The manufacturing strategy employed a Just-In-Time (JIT) workflow, synchronizing 3D printing, electronics validation, and software debugging to maximize throughput during the 43-hour active engineering sprint.

Outcomes & Analysis

The resulting platform successfully secured a division victory, validating the "agility over durability" design thesis. However, a catastrophic structural failure in the finals provided critical data regarding material shear limits in additive manufacturing. This document details the technical decision-making process, the integration of mechanical and IoT systems, and the post-mortem analysis of the structural failure.

Intended Audience

This report is prepared for Engineering Directors and Principal Investigators at high-velocity R&D firms. It demonstrates the capacity to act as a "Generalist Engineer" who can independently bridge the gap between theoretical physics, rapid manufacturing, and deployable software solutions.

Table of Contents

1. Project Description & Constraint Analysis

Operational Context & Resource Asymmetry

The "BOTIMUS" platform was developed within a high-stakes competitive environment designed to simulate rapid R&D cycles. While the standard operational procedure allocated 4-person teams to distribute the mechanical, electrical, and software workload, this project operated under a unique **Vertical Integration Strategy**.

To eliminate communication overhead and maximize design agility, I assumed sole ownership of the full technical stack (Mechanical Design, Electronics Integration, Firmware Development, and Manufacturing). This concentrated the critical path onto a single engineer, creating a significant resource asymmetry compared to competitor teams.

- **Role:** Lead Systems Engineer (Full Stack)
- **Time Constraint:** 72-Hour Hard Stop (Design to Deployment)
- **Resource Gap:** 1 Engineer vs. 4-Engineer Standard Competitor Units

Strategic Resource Allocation (The "Lean" Approach)

With a strictly fixed deadline of 72 hours, a traditional Waterfall development cycle was non-viable. Instead, I implemented a Parallel Execution Workflow designed to overlap manufacturing time with design time.

The core philosophy was "Minimum Viable Simulation." Rather than rushing to CAD, the first 15% of the timeline was dedicated to physics-based constraints analysis. By calculating torque requirements and printer volumetric throughput before modeling, I ensured that the design would fit within the 48-hour print window available.

- **Critical Path Identification:** The 3D printer was identified as the primary bottleneck.
- **Mitigation Strategy:** Design the largest chassis components first to initiate the print job immediately, allowing for firmware development and minor component design to occur in parallel with the print cycle.

System Requirements Definition

To ensure the platform met the rigorous "Combat Class" specifications without scope creep, the competition rulebook was translated into a strict Engineering Requirement Document (ERD). Table 1 outlines the translation of "Customer Wants" into quantifiable "Engineering Specs."

Table 1: Technical Requirement Mapping

Customer Requirement (CR)	**Engineering Requirement (ER)**	**Measurable Output**	**Target**
Physical Constraints			
Must fit within arena size limits	Chassis Dimensions	Length x Width x Height (in)	**Max 10" x 10" x 10"**
Must meet competition weight class	Total System Mass	Weight (lbs)	**Max 5.0 lbs**
Must prevent loose wires (vs. Wire Hook)	Cable Management / Shielding	Exposed Wire Length (mm)	**0 mm (All wires internal/secured)**
Power & Propulsion			
Must use official power source	Power Supply Specification	Voltage / Capacity	**7.4V LiPo (1500mAh) Only**
Must use standard drive actuators	Drive Motor Type	Component Model	**Provided TT Motors**
Must have traction in arena	Wheel Selection	Component Source	**Provided or Custom Designed**
Control & Mobility			
Must be remotely steerable	Control Interface	Communication Protocol	**Microcontroller + Smart Device App**
Must have full directional control	Maneuverability	Drive States	**Fwd, Rev, Left, Right, Stop**
Must remain mobile to avoid disqualification	Reliability / Recovery Time	Time Stationary (sec)	**< 10 seconds**
Combat & Weaponry			
Must damage/control opponent without prohibited items	Weapon System Type	Mechanism Category	**Cold Mechanical Only (No Chem/Elec)**
Must power offensive weapons	Auxiliary Actuation	Motor Type	**Team Choice (From Provided Stock)**
Must survive arena hazards (Flipper)	Stability / Center of Gravity	Recovery Ability	**Self-righting or low COG**
Must not illegally hold opponent	Pinning Duration	Time (sec)	**Max 10 seconds**

2. Conceptualization & Kinematic Analysis

Component Constraints & Force Derivation

The initial system architecture was defined by the strict Commercial Off-The-Shelf (COTS) component list provided by the competition rules (Table 2). A preliminary force analysis identified the MG90s Servo Motor as the critical mechanical bottleneck.

- **Constraint:** The servo provides a maximum stall torque of ~1.8 kg-cm.
- **Requirement:** To be competitive, the weapon system required a lifting force >1.0 kg (20% of opponent mass) to break traction.
- **Analysis:** A direct-drive mechanism was mathematically impossible given the torque limitations. This necessitated the design of a compound lever mechanism to amplify mechanical advantage at the cost of actuation speed. (See Appendix A for Force Derivation)

Table 2: Commercial Off The Shelf (COTS) Components

Part	Quantity
67 mm Wheels	4
Geared TT Yellow Motors (DC)	2
TB6612FNG Motor Controller/driver	1
MG90s Servo Motor	1
ESP32s Micro Processer	1
Toggle Switch	1
7.4V LiPo Battery (with charging cord)	1
Mini Breadboard (with jump wires)	1
L7805CV Voltage Regulator	1

The Python Advantage: Computational Design Optimization

To avoid the "Build-Test-Fail" cycle common in student projects, I developed a custom Python script in a Jupyter Notebook environment to simulate the kinematic performance of the weapon mechanism (See Appendix B). This allowed for the rapid iteration of **geometric variables** (lever length, fulcrum position) against **performance outputs** (tip force, lift height).

Unlike static hand calculations, this script enabled a **parameter sweep analysis**, generating a "Design Space" visualization (Figure 1) that mapped every possible linkage configuration.

Key Optimization Variables:

- **Input:** Servo Torque (1.5 kg-cm safe operating limit)
- **Variable:** Lever Arm Length (L_{lever}) & Flipper Length (L_{flip})
- **Objective Function:** Maximize Tip Force (F_{tip}) and Lift Height (H_{lift}) > 3.0 cm.

Concept Selection: Data-Driven Decision Making

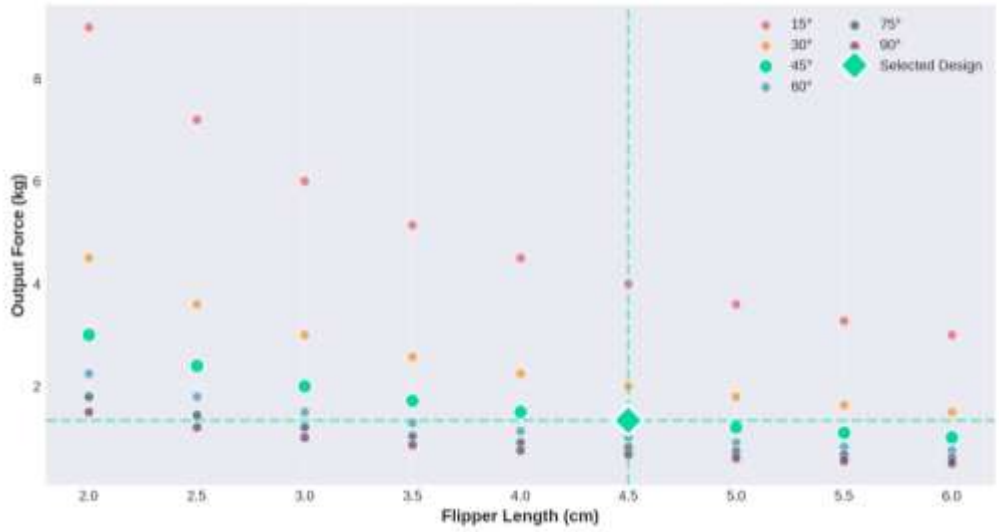


Figure 1: Design Space - Force Output Across All Configurations

The simulation results (Figure 1) revealed a non-linear relationship between lever arm length and effective tip force.

- **Scenario A (Long Reach):** A 6.0 cm flipper provided excellent reach but reduced tip force to <0.8 kg, failing the offensive requirement.
- **Scenario B (High Force):** A 3.0 cm flipper generated >2.0 kg of force but lacked the vertical displacement to destabilize an opponent.
- **Selected Configuration:** The simulation identified a local maximum at a 4.5 cm flipper length.

Final Design Specifications:

- **Mechanical Advantage:** Optimized for 1.33 kg output force.
- **Vertical Displacement:** 3.18 cm (sufficient to lift opponent wheels off-ground).
- **Factor of Safety:** The design assumes a servo efficiency of ~80% (1.5 kg-cm vs 1.8 kg-cm max) to prevent thermal overload during sustained combat.

3. Parallel Just-In-Time (JIT) Manufacturing Strategy

Asynchronous Fabrication Workflow

The 72-hour critical path was defined by the volumetric throughput of the Ender 5 Pro. To mitigate the "Idle Machine" bottleneck, I implemented an **Asynchronous Fabrication Workflow**. The largest chassis components were prioritized and initiated immediately. While the printer executed the 14-hour main chassis job (Figure 2), I utilized the "passive" print time to execute high-value active tasks: designing peripheral modules, writing the C++ firmware, and developing the Human-Machine Interface. This parallel processing effectively doubled the billable engineering hours available within the 3-day constraint.

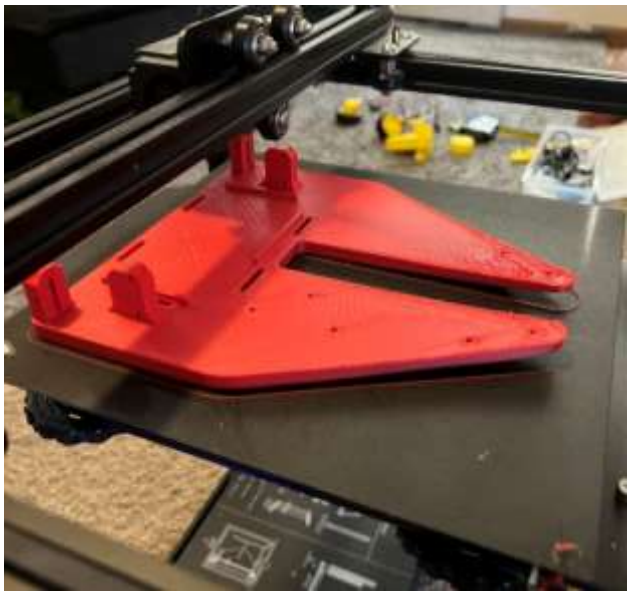


Figure 2: Printing Large Components while Designing Remaining Modules

Motion Sketch Assembly (Top-Down Design)

To support this manufacturing speed, the CAD architecture in PTC Creo utilized a **Top-Down "Master Model" Methodology** rather than traditional Bottom-Up assembly.

- **Skeleton Referencing (Figure 6):** A "Skeleton Model" was created first to define the maximum bounding box and the fixed locations of non-negotiable COTS components (motors, battery, microcontroller).

- **Parametric Updates:** All structural components referenced this central skeleton. If the battery location shifted in the master sketch, the chassis mounting points updated automatically, preventing assembly conflicts.

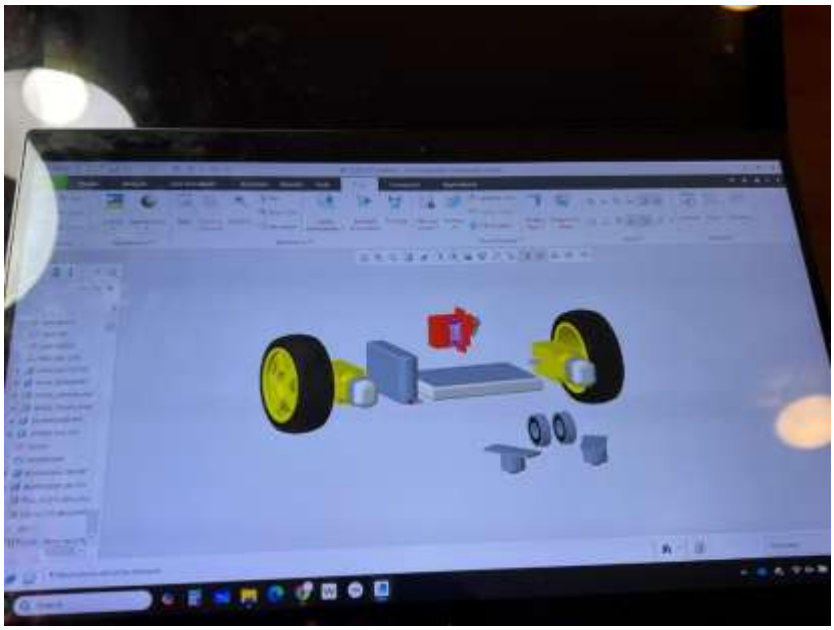


Figure 3: COTS Initial Layout

Modular Component Tolerance Analysis

Empirical Fit Validation ("Unit Testing" Hardware)

Given the variable shrinkage rates of PLA (approx. 0.3-0.5%), relying on nominal CAD dimensions for interference fits presented a critical integration risk. To de-risk the assembly, I implemented a **"Hardware Unit Test" Protocol.**

Before committing to long-duration prints, small "calibration blocks" were manufactured to test critical interfaces—specifically the servo arm spline and ball caster sockets (Figures 4 & 5). This micro-validation identified a required 0.2mm tolerance compensation for the servo horn. Correcting this parameter prior to the 2-hour gear print prevented a catastrophic integration failure that would have cost six hours for three test prints instead of 30 minutes. See Appendix C for full empirical tolerance derivation and calibration.

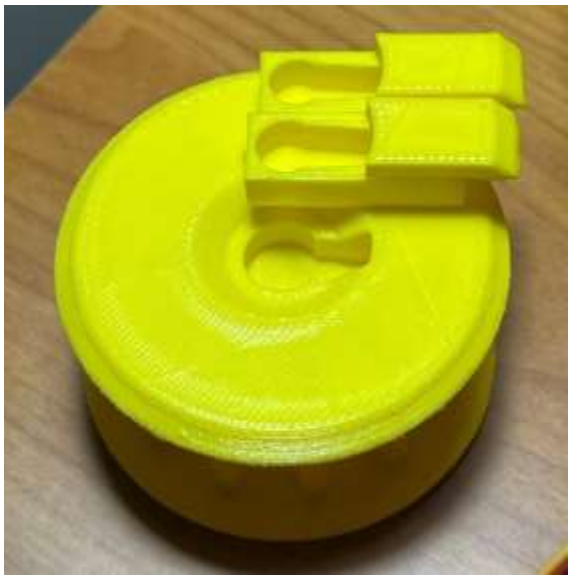


Figure 4: Servo Arm Cavity Tolerance Tests

Figure 5: Ball Caster Iterations

Printer Optimization

Manufacturing parameters were tuned for a specific performance profile: **Maximum Print Velocity** over **Ultimate Tensile Strength**.

- **Layer Height (0.28mm):** Coarse resolution was selected to reduce total print time by ~30% compared to the standard 0.2mm profile.
- **Infill Strategy (10% Grid):** A low-density internal structure minimized mass to maintain the <5.0 lb. weight limit. Structural integrity was preserved by increasing the **Wall Line Count to 3 (1.2mm shell)**, effectively creating a monocoque structure where the skin

bears the impact loads. Furthermore, the competition requires only cold, mechanical weapons so the risk of high-impact or cutting attacks is low.

Figure 6: Optimal Ender-5 Printer Settings for Rapid Printing

- **Support Interface Optimization (Figure 10):** To prevent support structures from fusing to the gear teeth, I manually overrode the 'Support Z-Distance' settings, increasing the air gap to 0.4mm (2x layer height). This ensured supports snapped off cleanly, preserving the involute gear profile without requiring post-processing abrasion.

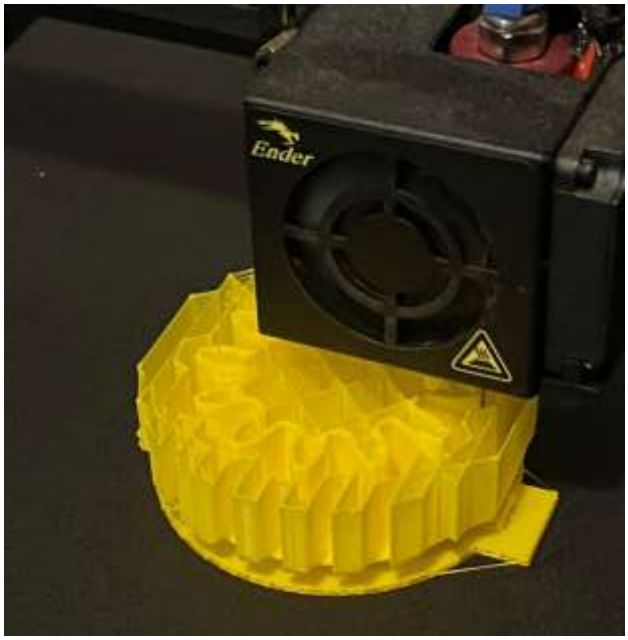


Figure 7: Increased Support Gap, Improving Gear Tooth Surface Finish

4. Electronics & Human-Machine Interface

UX Philosophy: Abstract the Complexity

The Human-Machine Interface (HMI) was designed to abstract electromechanical complexity, reducing operator cognitive load during high-stress combat scenarios.

- **Functional Prioritization:** The interface prioritizes "Intent" over "Actuation." A single "FLIP" command triggers a pre-programmed macro sequence on the microcontroller, rather than requiring the operator to manually toggle servo states (Figure 8).
- **Ergonomics:** The virtual joystick and trigger mechanism were mapped to the natural resting position of the thumbs, allowing for "blind" operation while the operator maintains visual contact with the robot.

Figure 8: Human-Centric Robotic Controls Mockup

IoT Control Architecture

To meet the 72-hour development cycle, the system leveraged the **Blynk IoT Middleware** for rapid packet handling. While cloud-based architectures introduce inherent latency, this was mitigated through a strict **Edge Computing Strategy** (Figure 9).

- **Minimizing Bandwidth:** The network payload was restricted to high-level "Intent Vectors" (e.g., Joystick X, Y integers) rather than raw motor PWM signals.
- **Local Processing:** The ESP32 acted as an intelligent edge node. It received the lightweight vector data and performed all kinematic mixing locally. This ensured that even if network packet loss occurred, the local loop continued to drive the motors smoothly, preventing "stuttering" artifacts common in streaming architectures.

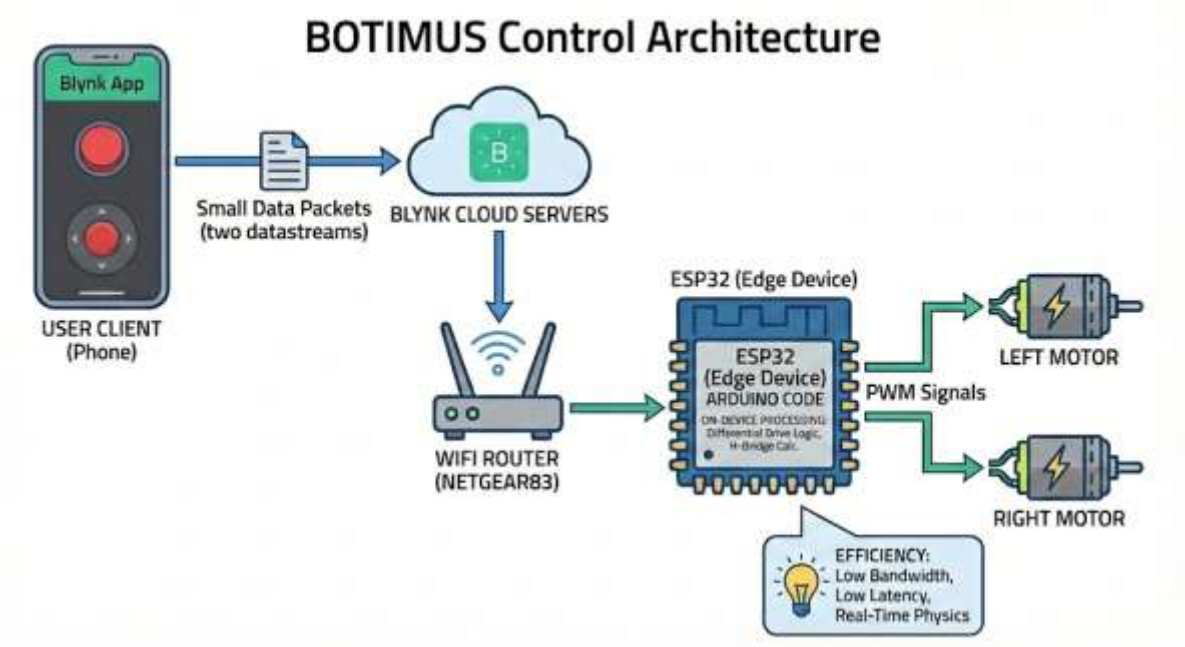


Figure 9: IoT Edge Device Control Architecture

Firmware: Differential Drive Kinematics

The core control logic (Appendix E) utilizes a **Differential Drive Mixing Algorithm** to translate single-vector inputs into dual-motor actuation.

- **Normalization:** Raw input data (0-255) is re-mapped to a signed integer domain (-255 to +255) to establish a true "Zero-Neutral" state.
- **Kinematic Mixing:** The firmware calculates target velocities using a linear superposition of the throttle (V_y) and steering (V_x) vectors:

$$V_{Left} = V_y + V_x \quad [1]$$

$$V_{Right} = V_y - V_x \quad [2]$$

- **H-Bridge Logic:** These target values are then passed to the driver logic, which dynamically switches the H-Bridge polarity and generates the specific PWM duty cycle required to achieve the target velocity.

5. Testing, Validation, Failure Analysis

Customer Requirements Stack Up

Figure 10: Competition Day

Figure 11: Final Product

Post-deployment validation confirms that the platform met 100% of the critical geometric and mass constraints (Table 3). A minor deviation occurred regarding cable shielding (30mm exposed wire), necessitated by the rapid assembly of the upper hull. However, the risk was mitigated through strain-relief zip-ties, resulting in zero electrical disconnects during match play.

Table 3: Final Prototype Requirements Review

Customer Requirement (CR)	**Engineering Requirement (ER)**	**Measurable Output**	**Result**
Physical Constraints			
Must fit within arena size limits	Chassis Dimensions	Length x Width x Height (in)	**9.8" x 9.5" x 4.6"**
Must meet competition weight class	Total System Mass	Weight (lbs)	**3.91 lbs**
Must prevent loose wires (vs. Wire Hook)	Cable Management / Shielding	Exposed Wire Length (mm)	**Risk Mitigated: Strain Relief Applied**
Power & Propulsion			
Must use official power source	Power Supply Specification	Voltage / Capacity	**7.4V LiPo (1500mAh) Only**
Must use standard drive actuators	Drive Motor Type	Component Model	**Provided TT Motors**
Must have traction in arena	Wheel Selection	Component Source	**Provided Wheels**
Control & Mobility			
Must be remotely steerable	Control Interface	Communication Protocol	**Blynk + ESP32**
Must have full directional control	Maneuverability	Drive States	**Arcade controls**
Must remain mobile to avoid disqualification	Reliability / Recovery Time	Time Stationary (sec)	**Moved continuously for the entire 10 minute match**
Combat & Weaponry			
Must damage/control opponent without prohibited items	Weapon System Type	Mechanism Category	**Mechanical Flipper**
Must power offensive weapons	Auxiliary Actuation	Motor Type	**MG90 Servo**
Must survive arena hazards (Flipper)	Stability / Center of Gravity	Recovery Ability	**Wide base and low COG**
Must not illegally hold opponent	Pinning Duration	Time (sec)	**Legally pinned two opponents with flipper**

Operational Performance: Winning The Division

Figure 12: First Division Competitors

The "Simulation-First" design philosophy yielded an immediate tactical advantage in the qualification rounds.

- **Kinematic Superiority:** While competitor robots relied on static wedges, BOTIMUS utilized the optimized lever arm to actively manipulate the Center of Gravity (COG) of opponents.
- **Control Efficiency:** The "Combat Cockpit" UX (Section 4) allowed superior maneuverability. By isolating the weapon control to a single thumb trigger, the driver could maintain evasive maneuvers while simultaneously deploying the flipper.

- **Outcome:** Three decisive victories, one by "Arena Out" (pushing out of bounds), and two 10-second pins with opponent wheels off the ground, securing the Division Championship.

Forensic Failure Analysis (The Finals)

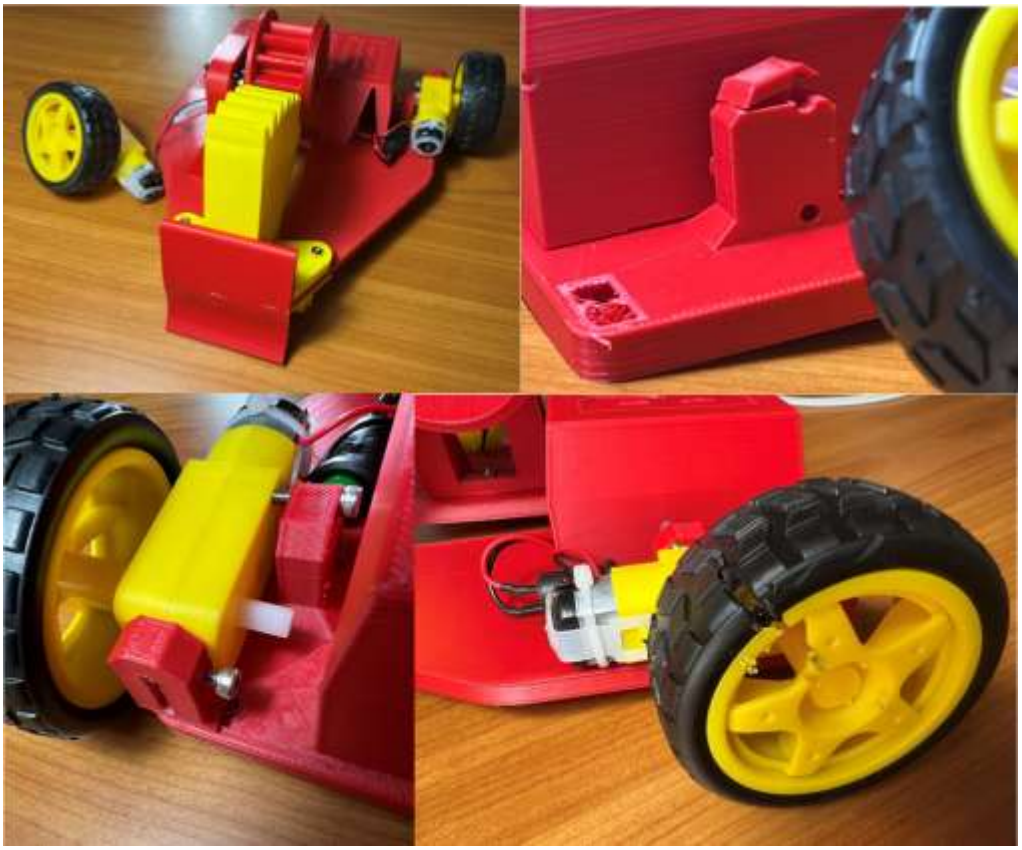


Figure 13: Wheel Attachment Failure Points

Incident Report: Catastrophic Structural Delamination

In the Championship Match, the platform suffered a catastrophic loss of mobility due to the severance of the left drive motor mount (Figure 13).

- **The Threat:** The opponent deployed a high-RPM Kinetic Energy Weapon (Rotary Saw), introducing a **shear load case** that was not present in the initial design requirements (which specified "Cold Mechanical" impact only).

- **Failure Mode:** The failure occurred at the interface between the wheel well and the main chassis. The saw blade penetrated the 1.2mm PLA shell, engaging the low-density (10%) infill lattice.
- **Root Cause:** The "Monocoque" printing strategy (Strong Skin / Weak Core) is optimized for blunt force impact distribution. It possesses near-zero resistance to concentrated shear forces. Once the outer skin was compromised by the saw, the internal lattice offered no structural resistance to the motor torque, causing the entire drive unit to tear free.

Corrective Action (The “Defense Matrix” Update)

This failure highlights a critical gap in the **System Threat Assessment**.

- **Immediate Fix:** Implementation of "Sacrificial Armor." Future iterations must utilize distinct, bolt-on wheel guards made of high-ductility material (TPU or Polycarbonate) to foul rotary weapons before they contact the structural chassis.
- **Design Guideline:** Critical load-bearing interfaces (Motor Mounts) must be printed with **100% Infill** or isolated from the chassis shell to prevent failure propagation.

Second Iteration Design Pivots

Figure 14: Mechanical Flipper Mechanism

Kinematic Inversion

The V1.0 flipper utilized a "Push" vector (Vertical/Backwards). Analysis and performance in the ring shows this configuration often pushed opponents away rather than destabilizing them.

- **The Pivot:** Invert the linkage geometry to create a "Hook and Lift" motion path. By engaging the opponent from the bottom-up, the mechanism can leverage the opponent's own mass to break their traction, rather than relying solely on servo torque.
- **Iteration Ease:** The flipper attachment (see Figure 14) is already modular, so this could be a simple reprint instead of a full mechanism redesign, cutting the next iteration timeframe from eight hours to about two hours.

Material Science Evolution

The reliance on PLA for structural components creates a hard ceiling on durability. The V2.0 roadmap transitions from "Rapid Prototype" to "Design for Manufacturing (DFM)."

- **Sheet Metal Integration:** The chassis floor and motor mounts will be converted to laser-cut 5052 Aluminum. This provides the shear strength required to survive kinetic weapons while maintaining the weight budget.

Conclusion: The “Agile Engineering” Retrospective

The execution of Project BOTIMUS, from blank slate to Division Champion in 43 hours, serves as a validation of the **"Parallel Execution Strategy."**

By decoupling the software development from the manufacturing critical path, the project achieved a velocity impossible under traditional Waterfall methodologies. The use of Python for pre-CAD kinematic optimization eliminated the "Trial and Error" phase, ensuring that the first physical prototype was a combat-ready unit.

While the structural failure in the finals provided a harsh lesson in material limits, the project succeeded in its primary objective: demonstrating that rigorous engineering principles (Constraint Analysis, Simulation, and DFM), when applied with discipline, can deliver high-value hardware solutions under extreme resource constraints.

Figure 15: Me with BOTIMUS after the Three Day Sprint

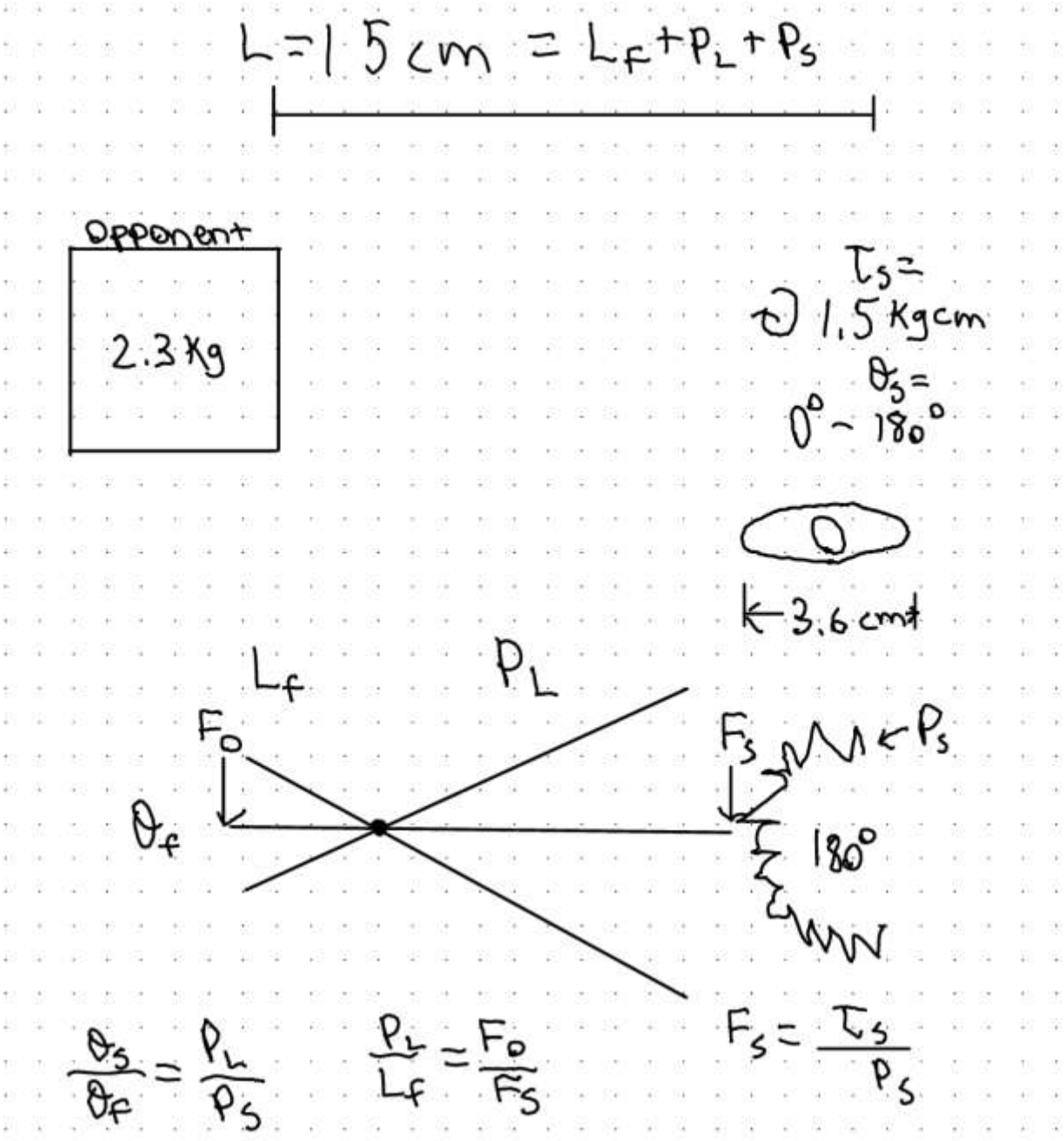
Maximus Shurr

maximusshurr.com | maxshurr@gmail.com

Purdue University | Honors Mechanical Engineering ‘26

Appendix

Appendix A: Preliminary Flipper Mechanism Force Derivation



$$\frac{P_L}{L_F} = \frac{F_0 P_S}{\tau_S} \qquad L = L_F + P_L + P_S$$

$$L_F = L - P_L - P_S$$

$$P_L = \frac{F_0 P_S}{\tau_S}\left[L - P_L - P_S\right]$$

$$P_L\left(1 + \frac{F_0 P_S}{\tau_S}\right) = \frac{F_0 P_S}{\tau_S}\left[L - P_S\right]$$

$$P_L = \frac{F_0 P_S}{\tau_0} \frac{L - P_S}{1 + \frac{F_0 P_S}{\tau_S}}$$

given L_F & θ_F, solve for the rest

$$L - L_F = P_S + P_L$$

$$P_L = P_S \frac{\theta_S}{\theta_F}$$

$$L - L_F = P_S\left(1 + \frac{\theta_S}{\theta_F}\right)$$

$$P_S = \frac{L - L_F}{1 + \theta_S/\theta_F}$$

$$F_S = \frac{\tau_S}{P_S}$$

$$F_0 = \frac{F_S P_L}{L_F}$$

Appendix B: Jupyter Notebook Mechanism Calculations

Botimus Calculations

Below is the math used to calculate the desired dimensions and operating conditions of Botimus

Term Definitions

Symbol	Definition
theta_f	flipper arm degree range
F_o	Opponent force applied at flipper tip
L_f	Length of flipper from fulcrum
F_s	Servo force applied at other end of flipper lever
P_l	Pitch circle radius of lever
P_s	Pitch circle radius of servo gear
tau_s	Servo max safe operating torque
theta_s	servo arm degree range
L	mechanism length

Known Properties

```
L = 15 #cm
theta_s = 180 #degrees
tau_s = 1.5 #kg-cm
```

Target Properties

```
F_o = 3 #kg
P_s = 2 #cm
```

Design Calculations

```
F_s = tau_s/P_s #cm
P_l = ((F_o*P_s)/tau_s)*((L-P_s)/(1+((F_o*P_s)/tau_s))) #cm
L_f = L - P_l - P_s #cm
theta_f = (theta_s*P_s)/P_l #deg

assert round(P_l*F_s, 5) == round(L_f*F_o, 5)

print("Lever Info:")
print(f"Force: {F_o} kg; Flipper Angle Range: {theta_f} deg")
print(f"Flipper: {L_f} cm; Lever: {P_l} cm; Servo: {P_s}")
```

```
Lever Info:
Force: 3 kg; Flipper Angle Range: 34.61538461538461 deg
Flipper: 2.5999999999999996 cm; Lever: 10.4 cm; Servo: 2
```

Bulk Scenario Testing

Lever from Opponent Force and Servo Pitch Radius

```
def get_lever_info(F_o, P_s, L = 15, theta_s = 180, tau_s = 1.5):
    F_s = tau_s/P_s #cm
    P_l = ((F_o*P_s)/tau_s)*((L-P_s)/(1+((F_o*P_s)/tau_s))) #cm
    L_f = L - P_l - P_s #cm
    theta_f = (theta_s*P_s)/P_l #deg

    assert round(P_l*F_s, 5) == round(L_f*F_o, 5)

    return {
        "Force": F_o,
        "Flipper Angle Range": theta_f,
        "Flipper": L_f,
        "Lever": P_l,
        "Servo": P_s
    }

print(get_lever_info(1, 2))
print(get_lever_info(1.7, 2))
print(get_lever_info(1.4, 2))
print(get_lever_info(2.3, 2))
print(get_lever_info(3, 2))
print(get_lever_info(4, 2))
print(get_lever_info(5, 2))
```

```
{'Force': 1, 'Flipper Angle Range': 48.46153846153846, 'Flipper':
5.571428571428571, 'Lever': 7.428571428571429, 'Servo': 2}
{'Force': 1.7, 'Flipper Angle Range': 39.90950226244344, 'Flipper':
3.979591836734693, 'Lever': 9.020408163265307, 'Servo': 2}
{'Force': 1.4, 'Flipper Angle Range': 42.527472527472526, 'Flipper':
4.534883720930232, 'Lever': 8.465116279069768, 'Servo': 2}
{'Force': 2.3, 'Flipper Angle Range': 36.72240802675585, 'Flipper':
3.196721311475409, 'Lever': 9.80327868852459, 'Servo': 2}
{'Force': 3, 'Flipper Angle Range': 34.61538461538461, 'Flipper':
2.5999999999999996, 'Lever': 10.4, 'Servo': 2}
{'Force': 4, 'Flipper Angle Range': 32.88461538461539, 'Flipper':
2.052631578947368, 'Lever': 10.947368421052632, 'Servo': 2}
{'Force': 5, 'Flipper Angle Range': 31.846153846153843, 'Flipper':
1.695652173913043, 'Lever': 11.304347826086957, 'Servo': 2}
```

```
print(get_lever_info(2.3, 1))
print(get_lever_info(2.3, 2))
print(get_lever_info(2.3, 3))
print(get_lever_info(2.3, 4))
print(get_lever_info(2.3, 5))
```

```
{'Force': 2.3, 'Flipper Angle Range': 21.242236024844722, 'Flipper':
5.526315789473685, 'Lever': 8.473684210526315, 'Servo': 1}
{'Force': 2.3, 'Flipper Angle Range': 36.72240802675585, 'Flipper':
3.196721311475409, 'Lever': 9.80327868852459, 'Servo': 2}
{'Force': 2.3, 'Flipper Angle Range': 54.78260869565218, 'Flipper':
2.142857142857144, 'Lever': 9.857142857142856, 'Servo': 3}
{'Force': 2.3, 'Flipper Angle Range': 76.12648221343873, 'Flipper':
1.5420560747663554, 'Lever': 9.457943925233645, 'Servo': 4}
{'Force': 2.3, 'Flipper Angle Range': 101.73913043478262, 'Flipper':
1.153846153846155, 'Lever': 8.846153846153845, 'Servo': 5}
```

Output Force and Lever Design from Flipper Length and Flipper Angle Range

```
from math import sin, radians

def get_lever_design(L_f, theta_f, L = 15, theta_s = 180, tau_s =
1.5):
    P_s = (L-L_f)/(1 + (theta_s/theta_f)) #cm
    P_l = L - L_f - P_s #cm
    F_s = tau_s/P_s #cm
    F_o = (P_l*F_s)/L_f #kg

    assert round(P_l*F_s, 5) == round(L_f*F_o, 5)

    return {
        "Force": F_o,
        "Max Height": sin(radians(theta_f))*L_f,
        "Servo": P_s,
        "Lever": P_l,
        "Flipper Angle Range": theta_f,
        "Flipper": L_f,
    }

info = get_lever_info(2.3, 1)
print(info)
print(get_lever_design(info["Flipper"], info["Flipper Angle Range"]))
```

```
{'Force': 2.3, 'Flipper Angle Range': 21.242236024844722, 'Flipper':
5.526315789473685, 'Lever': 8.473684210526315, 'Servo': 1}
{'Force': 2.2999999999999994, 'Max Height': 2.0022490973867195,
'Servo': 1.0, 'Lever': 8.473684210526315, 'Flipper Angle Range':
21.242236024844722, 'Flipper': 5.526315789473685}
```

```
print(get_lever_design(5, 15))
print(get_lever_design(5, 30))
print(get_lever_design(5, 45))
print(get_lever_design(5, 60))
print(get_lever_design(5, 75))
print(get_lever_design(5, 90))
```

```
print(get_lever_design(4.5, 15))
print(get_lever_design(4.5, 30))
print(get_lever_design(4.5, 45))
print(get_lever_design(4.5, 60))
print(get_lever_design(4.5, 75))
print(get_lever_design(4.5, 90))
```

```
{'Force': 3.599999999999999, 'Max Height': 1.2940952255126037,
'Servo': 0.7692307692307693, 'Lever': 9.23076923076923, 'Flipper Angle
Range': 15, 'Flipper': 5}
{'Force': 1.8, 'Max Height': 2.4999999999999996, 'Servo':
1.4285714285714286, 'Lever': 8.571428571428571, 'Flipper Angle Range':
30, 'Flipper': 5}
{'Force': 1.2, 'Max Height': 3.5355339059327378, 'Servo': 2.0,
'Lever': 8.0, 'Flipper Angle Range': 45, 'Flipper': 5}
{'Force': 0.9, 'Max Height': 4.330127018922193, 'Servo': 2.5, 'Lever':
7.5, 'Flipper Angle Range': 60, 'Flipper': 5}
{'Force': 0.72, 'Max Height': 4.8296291314453415, 'Servo':
2.9411764705882355, 'Lever': 7.0588235294117645, 'Flipper Angle
Range': 75, 'Flipper': 5}
{'Force': 0.5999999999999999, 'Max Height': 5.0, 'Servo':
3.3333333333333335, 'Lever': 6.666666666666666, 'Flipper Angle Range':
90, 'Flipper': 5}
{'Force': 4.0, 'Max Height': 1.1646857029613433, 'Servo':
0.8076923076923077, 'Lever': 9.692307692307692, 'Flipper Angle Range':
15, 'Flipper': 4.5}
{'Force': 2.0, 'Max Height': 2.2499999999999996, 'Servo': 1.5,
'Lever': 9.0, 'Flipper Angle Range': 30, 'Flipper': 4.5}
{'Force': 1.3333333333333333, 'Max Height': 3.1819805153394642,
'Servo': 2.1, 'Lever': 8.4, 'Flipper Angle Range': 45, 'Flipper': 4.5}
{'Force': 1.0, 'Max Height': 3.8971143170299736, 'Servo': 2.625,
'Lever': 7.875, 'Flipper Angle Range': 60, 'Flipper': 4.5}
{'Force': 0.8, 'Max Height': 4.346666218300808, 'Servo':
3.088235294117647, 'Lever': 7.411764705882353, 'Flipper Angle Range':
75, 'Flipper': 4.5}
{'Force': 0.6666666666666666, 'Max Height': 4.5, 'Servo': 3.5,
'Lever': 7.0, 'Flipper Angle Range': 90, 'Flipper': 4.5}
```

```
print(get_lever_design(2, 45))
print(get_lever_design(2.5, 45))
print(get_lever_design(3, 45))
print(get_lever_design(3.5, 45))
print(get_lever_design(4, 45))
print(get_lever_design(4.5, 45))
print(get_lever_design(6, 45))
```

```
{'Force': 3.0, 'Max Height': 1.4142135623730951, 'Servo': 2.6,
'Lever': 10.4, 'Flipper Angle Range': 45, 'Flipper': 2}
{'Force': 2.4, 'Max Height': 1.7677669529663689, 'Servo': 2.5,
'Lever': 10.0, 'Flipper Angle Range': 45, 'Flipper': 2.5}
```

```
{'Force': 2.0, 'Max Height': 2.121320343559643, 'Servo': 2.4, 'Lever':
9.6, 'Flipper Angle Range': 45, 'Flipper': 3}
{'Force': 1.7142857142857142, 'Max Height': 2.4748737341529163,
'Servo': 2.3, 'Lever': 9.2, 'Flipper Angle Range': 45, 'Flipper': 3.5}
{'Force': 1.5, 'Max Height': 2.8284271247461903, 'Servo': 2.2,
'Lever': 8.8, 'Flipper Angle Range': 45, 'Flipper': 4}
{'Force': 1.3333333333333333, 'Max Height': 3.1819805153394642,
'Servo': 2.1, 'Lever': 8.4, 'Flipper Angle Range': 45, 'Flipper': 4.5}
{'Force': 1.0, 'Max Height': 4.242640687119286, 'Servo': 1.8, 'Lever':
7.2, 'Flipper Angle Range': 45, 'Flipper': 6}
```

Design Selection

We have shown that a small flipper can generate up to 5kg of opponent force, double the weight limit, but it's impractical because the flipper would have to be so short. Instead, we decided to go with a flipper that balances length and angle range to lift the opponent off of one wheel, pinning them in the air. We only need about half of the weight limit on our side since the normal force of the ground can supply the remaining force.

Design Parameters

```
final_design_info = get_lever_design(4.5, 45)
print(f"Flipper Length: {final_design_info['Flipper']} cm")
print(f"Lever Length: {final_design_info['Lever']} cm")
print(f"Servo Length: {final_design_info['Servo']} cm")
print(f"Max Height: {final_design_info['Max Height']:.2f} cm")
print(f"Force: {final_design_info['Force']:.2f} kg")
print(f"Flipper Angle Range: {final_design_info['Flipper Angle
Range']} deg")
print(f"Servo Angle Range: {theta_s} deg")
print(f"Servo Torque: {tau_s} kg-cm")
```

```
Flipper Length: 4.5 cm
Lever Length: 8.4 cm
Servo Length: 2.1 cm
Max Height: 3.18 cm
Force: 1.33 kg
Flipper Angle Range: 45 deg
Servo Angle Range: 180 deg
Servo Torque: 1.5 kg-cm
```

Appendix C: Empirical Tolerance Derivation & Calibration

Objective: To determine the precise Geometric Dimensioning and Tolerancing (GD&T) offsets required for interference and clearance fits using the specific Ender 5 Pro / PLA hardware-material combination.

Methodology: A "Rotational Interference Calibration Artifact" (RICA) was designed to validate three critical fit types:

1. **Press Fit (Transition):** For the 608 Bearings and Hex Nuts.
2. **Clearance Fit:** For the Omni-Directional Ball Caster mechanism.
3. **Thread Engagement:** For M3/M4 fasteners.

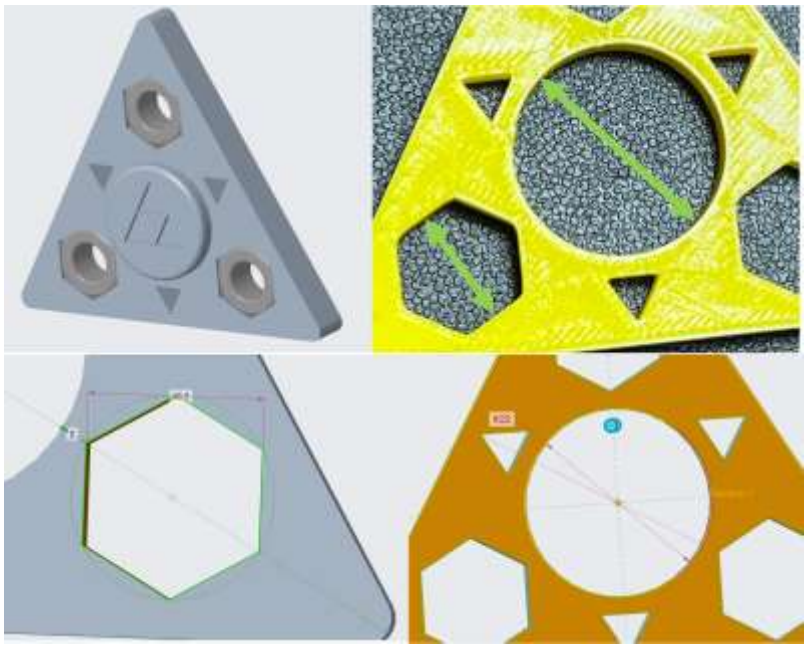


Figure C.1: Rotational Interference Calibration Artifact

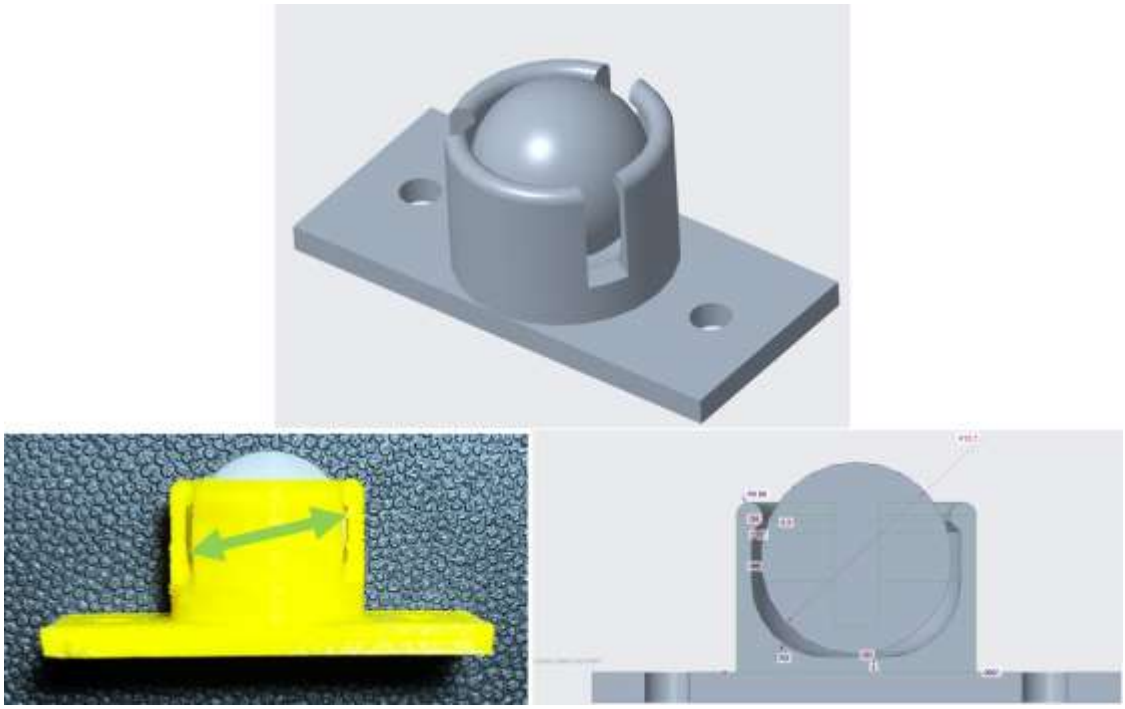


Figure C.2: Ball Caster CAD, Real Part and Desired Dimensions

Data & Analysis: Table C.1 compares nominal CAD dimensions against physical printed outputs.

Table C.1: GD&T Nominal Dimension Tolerance Data Collection

Nominal Dimension	**Physical dimension (measured from COTS)**	**Fitting hole dimension specified in CAD**	**Expected Dimension from 3D Printed Part**	**Measured 3D Printed Part Dimensions**
608 Bearing D = 22mm	22.00mm	22.00mm	21.7mm	21.76mm
Hex nut Socket Size = 12.7mm	12.55mm	12.50mm	12.2mm	11.95mm
Bearing Ball d = 12.7 mm	12.71 mm	13.30 mm	13.0 mm	12.86 mm

Findings:

- **Dimensional Drift:** The Z-axis (inter-layer) dimensions exhibited higher variance than XY-plane dimensions due to layer squish.

- **Ball Caster Optimization:** The initial 0.1mm tolerance resulted in a seized joint. A second iteration with a **0.3mm radial clearance** was identified as the optimal setpoint, allowing free rotation while retaining the ball bearing during high-G impacts (Figure C.3).

Figure C.3: First and Final Iterations of Ball Caster

- **Guideline Formulation:** This study established a project-wide design rule: **+0.2mm offset for interference fits** and **+0.3mm offset for clearance moving parts**.

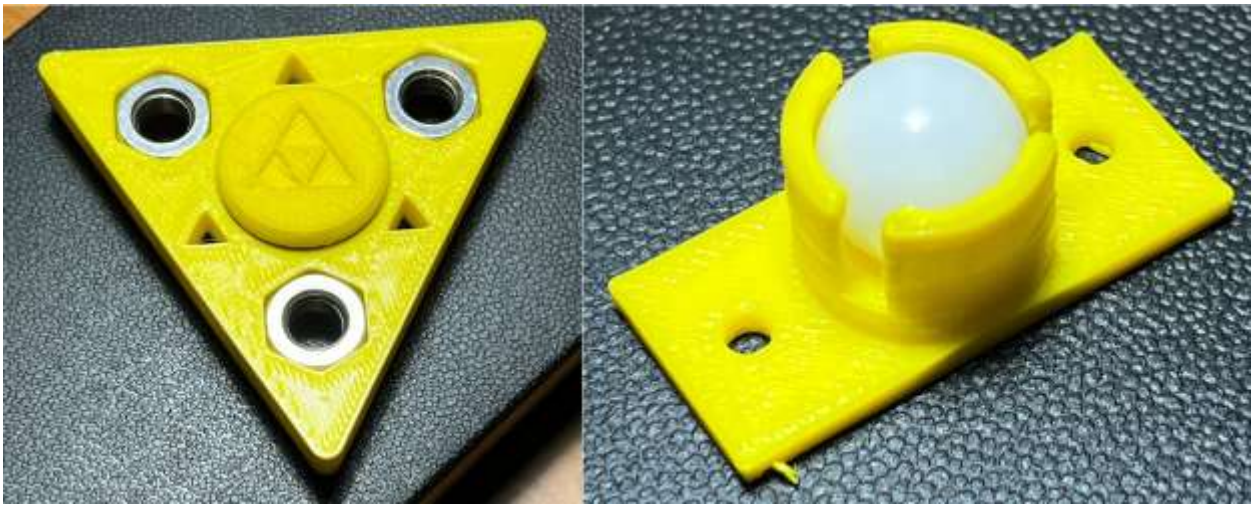


Figure C.4: Final RICA and Ball Caster

Appendix D: CAD Screenshots

Full Creo assembly and .stl files are available on request. See https://maximusshurr.com.

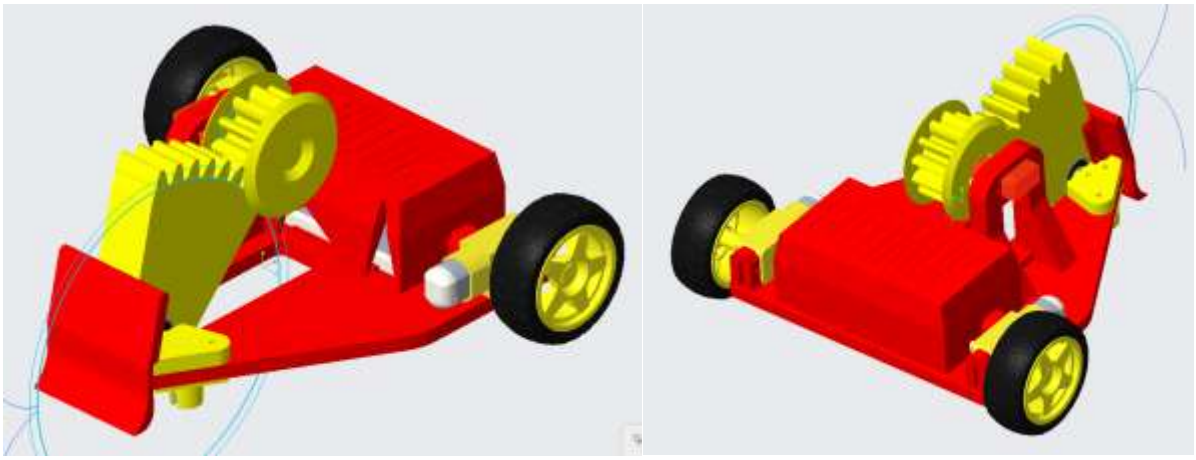


Figure D.1: Full Assembly

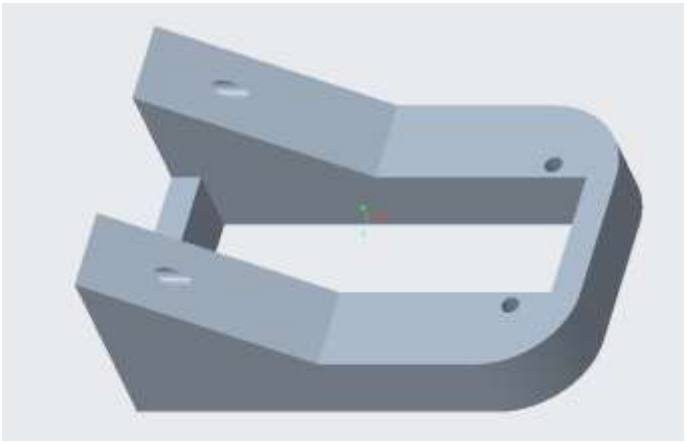


Figure D.2: Servo Mount

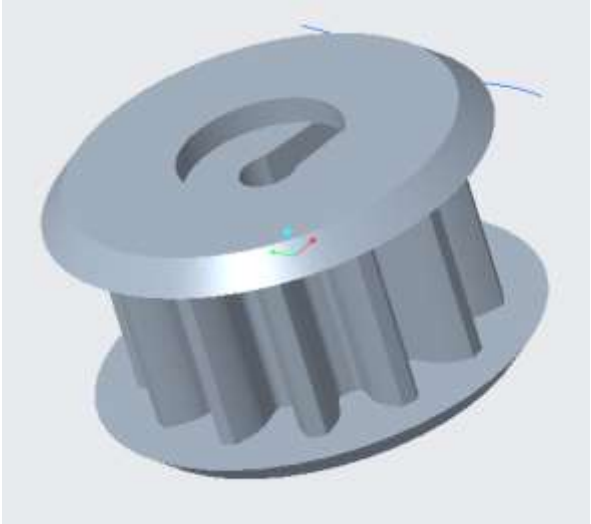


Figure D.3: Servo Gear

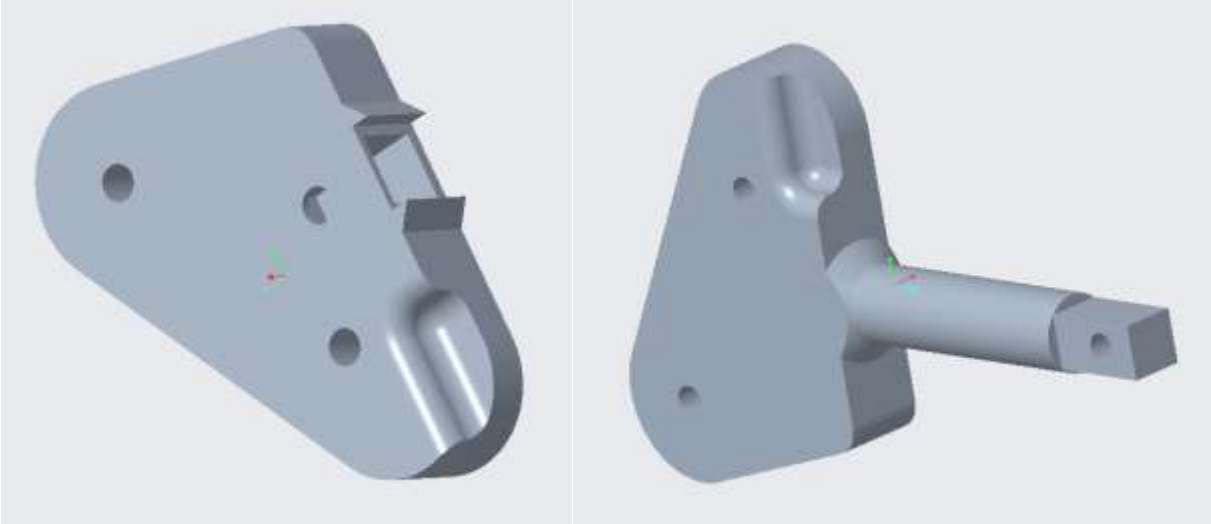


Figure D.4: Flipper Arm Axel

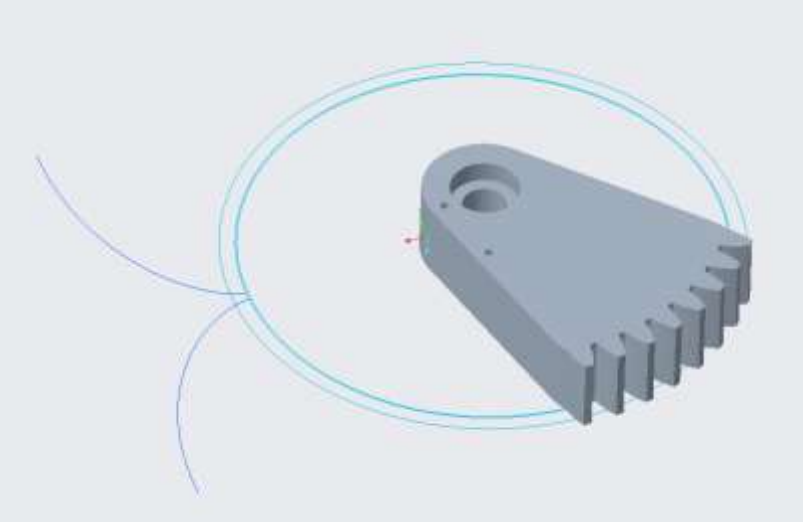


Figure D.5: Flipper Arm

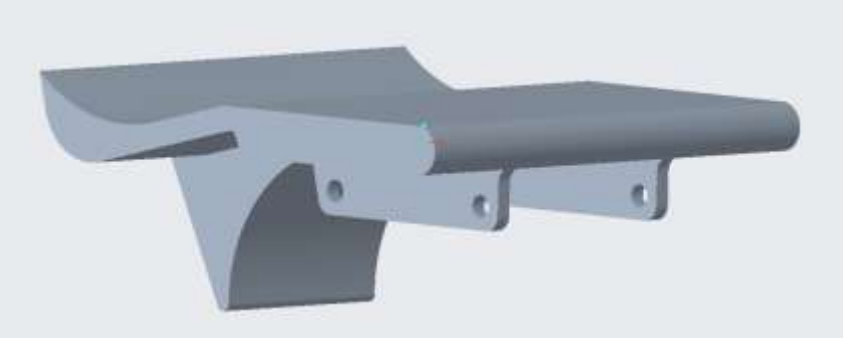


Figure D.6: Flipper Attachment

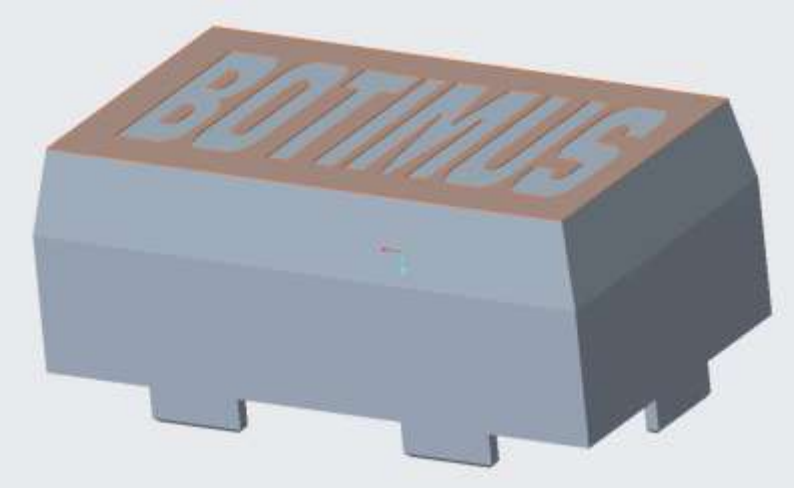


Figure D.7: Electronics Housing

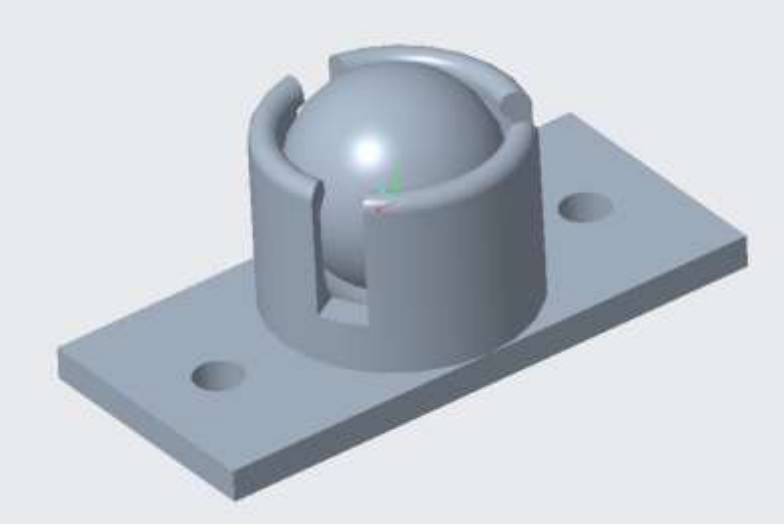


Figure D.8: Ball Caster

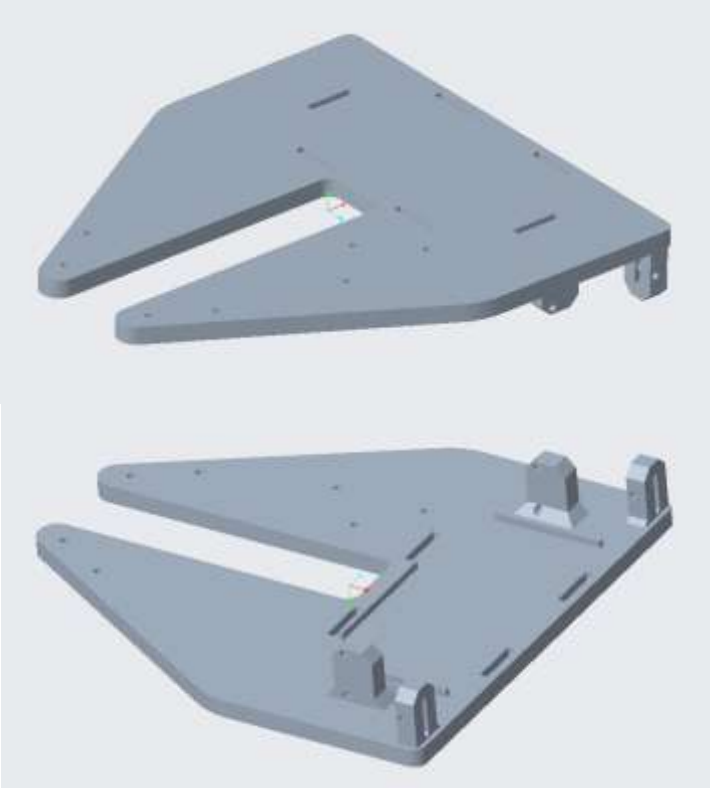


Figure D.9: Baseplate Chassis

Appendix E: Edge Device Embedded Code

Below is “Botimus_Code.ino”, the Arduino file runs on the robot during operation and allows the robot to listen to the Blynk datastreams and drive the motors on command.

```
#define BLYNK_PRINT Serial

#define BLYNK_TEMPLATE_ID "REDACTED"
#define BLYNK_TEMPLATE_NAME "REDACTED"
#define BLYNK_AUTH_TOKEN "REDACTED"

#include <BlynkSimpleEsp32.h>

// Auth token from Blynk
char auth[] = BLYNK_AUTH_TOKEN;

// Competition WiFi credentials
char ssid[] = "NETGEAR83";
char pwd[] = "quaintbreeze400";

#include "ESP32Servo.h"

#define SERVO_PIN 27
#define A_PWM 15
#define A_IN1 2
#define A_IN2 0
#define B_PWM 13
#define B_IN1 12
#define B_IN2 14

Servo myServo;  //claim the servo motor object

void setup() // setup function runs onces whenever the arduino board is powered
on or the "EN" button is pressed
{
  Serial.begin(9600);
  while(!Serial);

  Blynk.begin(auth, ssid, pwd);  // Initialize the connection to Blynk using
auth-token and WiFI credentials.

  myServo.attach(SERVO_PIN);    //set up the servo pin as pin 27
  myServo.write(0);   //initialize the servo motor at zero degrees
  pinMode(A_PWM, OUTPUT);
```

```
  pinMode(A_IN1, OUTPUT);    //set A_IN1(2) as an output pin
  pinMode(A_IN2, OUTPUT);    //set A_IN2(0) as an output pin
  pinMode(B_PWM, OUTPUT);
  pinMode(B_IN1, OUTPUT);    //set B_IN1(12) as an output pin
  pinMode(B_IN2, OUTPUT);    //set B_IN2(14) as an output pin
  digitalWrite(A_IN1, LOW);
  digitalWrite(A_IN2, LOW);
  digitalWrite(B_IN1, LOW);
  digitalWrite(B_IN2, LOW);
  analogWrite(A_PWM, 0);
  analogWrite(B_PWM, 0);
}

BLYNK_WRITE(V0)
{
  int servoTarget = param.asInt();
  myServo.write(servoTarget);
  // Serial.print("servo motor goes to");
  // Serial.println(servoTarget);
}

BLYNK_WRITE(V1) {
  // 0-255
  // 0-127 128-255
  int joystick_x = param[0].asInt();//[-127,128]
  int joystick_y = param[1].asInt();
  int x = map(joystick_x, 0, 255, -255, 255);
  int y = map(joystick_y, 0, 255, -255, 255);
  int leftMotorSpeed = constrain(y + x, -255, 255);
  int rightMotorSpeed = constrain(y - x, -255, 255);
  // Serial.print("V1: x = ");
  // Serial.print(x);
  // Serial.print("joy_x = ");
  // Serial.print(joystick_x);
  // Serial.print("; LMS: ");
  // Serial.print(leftMotorSpeed);
  // Serial.print("\t y=");
  // Serial.print(y);
  // Serial.print("joy_y = ");
  // Serial.print(joystick_y);
  // Serial.print("; RMS: ");
  // Serial.println(rightMotorSpeed);

  if (leftMotorSpeed < 0) {
    digitalWrite(A_IN1, HIGH);
```

```
    digitalWrite(A_IN2, LOW);
  } else {
    digitalWrite(A_IN1, LOW);
    digitalWrite(A_IN2, HIGH);
  }

  if (rightMotorSpeed < 0) {
    digitalWrite(B_IN1, HIGH);
    digitalWrite(B_IN2, LOW);
  } else {
    digitalWrite(B_IN1, LOW);
    digitalWrite(B_IN2, HIGH);
  }

  analogWrite(A_PWM, leftMotorSpeed);
  analogWrite(B_PWM, rightMotorSpeed);
}

void loop() {
  Blynk.run();
}
```

Appendix F: Unexpected Anomalies

On the day of the finals, I flipped over the bot to start charging, and I realized the bottom of the flipper mechanism kind of looks like a chicken… ball caster eyes, a red flipper beak with a gear head of feathers. It's not a bug, it's a feature 😁