A Comprehensive Guide to Designing and Implementing a Scalable, AI-Powered Thrust Vector Control System for Model Rocketry

Section 1: Foundational Principles of Actively Controlled Rocket Flight

The successful development of an advanced model rocket hinges on a deep understanding of the principles that govern its stability and control. This section establishes the theoretical bedrock for a Thrust Vector Control (TVC) system, moving from the fundamental justification for its use to the specific physical models required for its design and implementation. By grounding the project in these core concepts, a scalable and robust system can be engineered.

1.1 From Static Fins to Dynamic Control: An Introduction to TVC

Thrust Vector Control is the ability of a vehicle to manipulate the direction of the thrust from its propulsion system to control its attitude (orientation) and angular velocity.[1] For rockets and ballistic missiles operating in the vacuum of space or at very low airspeeds just after launch, conventional aerodynamic control surfaces like fins are ineffective. In these regimes, TVC is the primary, and often only, means of attitude control.[1]

The fundamental principle of TVC is the generation of a control torque. In a stable, un-vectored rocket, the thrust vector—the line of action of the engine's force—is designed to pass directly through the vehicle's center of mass (CM). This produces linear acceleration without inducing any rotation.[1] TVC works by intentionally deflecting this thrust vector, using mechanisms like a gimbaled nozzle, so that it no longer passes through the CM. This offset thrust creates a lever arm, resulting in a net torque that rotates the vehicle about its CM.[1] This corrective torque can be precisely modulated to counteract external disturbances, such as wind gusts, uneven propellant burn, or thrust misalignments, thereby maintaining the rocket's desired trajectory.[2] It can also be used to execute intentional flight path maneuvers.

This active control paradigm represents a significant departure from traditional model rocketry, which almost exclusively relies on passive stability. Passive stability is achieved by placing fins at the rear of the rocket, which shifts the vehicle's center of pressure (CP) behind its center of mass. Aerodynamic forces acting on these fins automatically generate corrective torques to keep the rocket pointing into the relative wind.[6] While simple and reliable, this method requires high initial velocity to be effective, leading to the characteristic rapid, ballistic-like ascent of model rockets. TVC, by contrast, provides full control authority even at zero airspeed, enabling the slow, controlled, and majestic vertical ascents characteristic of full-scale launch vehicles.[8]

The relationship between the Center of Mass and Center of Pressure is fundamentally altered in a TVC-controlled vehicle. While a passively stable rocket *must* have its CP behind its CM, a vehicle with active control can be designed to be neutrally stable or even slightly unstable.[9] This is a concept known as "active stability," where the control system is continuously working to keep the inherently unstable airframe on its desired path. This liberates the designer from the constraints of large fins and heavy tail sections, allowing for more aerodynamically efficient and lightweight designs. However, it places the entire burden of a successful flight on the absolute reliability and performance of the Guidance, Navigation, and Control (GNC) system.

1.2 A Comparative Analysis of TVC Mechanisms

Several methods have been developed to achieve thrust vectoring, each with distinct advantages and disadvantages. For the scope of a scalable model rocket, understanding these options clarifies why a gimbaled nozzle is the most practical and widely adopted approach.

- **Gimbaled Engines/Nozzles:** This is the most common and efficient method, involving the mechanical pivoting of the entire engine assembly or, more commonly in solid motors, just the nozzle. The deflection is controlled by linear or rotary actuators.[1] This technique has been proven on countless launch vehicles, including the Saturn V and the Space Shuttle, and is the focus of most hobbyist and professional TVC systems today.[1]
- **Exhaust Vanes (Jet Vanes):** This method involves placing heat-resistant, wing-shaped surfaces directly into the engine's exhaust plume.[1] By deflecting these vanes, the exhaust flow is redirected, generating a side force. Jet vanes are advantageous at low airspeeds where aerodynamic fins are useless and can provide roll control with a single nozzle, a feat not possible with a simple two-axis gimbal.[10] However, they suffer from significant drawbacks, including extreme thermal stress, material erosion during the motor burn, and a reduction in total thrust (lower specific impulse) due to the obstruction in the exhaust flow.[10]
- **Liquid Injection TVC (LITVC):** In this system, a secondary fluid (liquid or gas) is injected through ports in the nozzle wall into the main exhaust stream.[1] This injection creates an oblique shock wave that produces an asymmetric pressure distribution

inside the nozzle, resulting in a net side force that vectors the thrust.[10] LITVC was used on early solid-propellant ballistic missiles like the Minuteman II.[1] While it avoids moving parts in the hot exhaust stream, it is generally less efficient than mechanical gimbaling, especially for large control moments, and adds the complexity of a secondary fluid storage and injection system.[10]

- **Vernier Thrusters:** These are small, auxiliary rocket engines that are independently controlled and often gimbaled themselves. They are used to provide fine attitude control and roll control.[1] The R-7 missile family, including the modern Soyuz rocket, famously uses vernier thrusters for steering.[1] However, they add considerable weight, complexity, and plumbing, making them impractical for model rocketry.

1.3 The Physics of TVC: Modeling Torque, Attitude, and Flight Dynamics

To design a control system, one must first have a mathematical model of the system to be controlled. This involves defining coordinate systems and understanding the forces and torques at play.

The rocket's attitude is typically described using two reference frames: a body-fixed frame {B} with its origin at the rocket's CM, and an inertial, space-fixed frame {I} (often a North-East-Down or East-North-Up frame originating at the launch site).[14] The transformation between these frames is defined by a set of three Euler angles: roll (ϕ), pitch (θ), and yaw (ψ).[14]

The core of TVC is the generation of control torque, τ. When the engine nozzle, producing thrust T, is gimbaled by a small angle δ, the thrust vector is split into two components: a large axial component, $T_{axial} = T\cos(\delta)$, which provides the primary propulsive force, and a smaller lateral (or side) component, $T_{lateral} = T\sin(\delta)$, which is perpendicular to the rocket's main axis. This lateral force, acting at a distance L from the gimbal pivot point to the vehicle's CM, creates a control torque:

$$\tau = L \times T_{lateral}$$

For small angles, where $\sin(\delta) \approx \delta$, this torque is approximately proportional to the gimbal angle, providing a means to control the rocket's angular acceleration, α, according to Newton's second law for rotation, $\tau = I\alpha$, where I is the rocket's moment of inertia tensor.[15]

A more advanced model must also account for the "tail-wags-dog" (TWD) effect. This is an inertial coupling phenomenon where the angular acceleration of the gimbaled engine mass exerts reaction forces and torques back onto the rocket's airframe.[16] This effect is particularly important for high-fidelity simulations and the stability analysis of large, flexible launch vehicles, but its principles apply even at model scale.

Ultimately, the rocket's behavior is described by its state—a vector of variables that fully captures its dynamic condition at any instant. This state vector typically includes position, velocity, attitude (often as a quaternion to avoid gimbal lock), and angular rates.[18] The goal of

the control system is to observe this state and apply actions (gimbal commands) to drive it towards a desired state (e.g., zero pitch and yaw angles).

1.4 Principles of Scalability: How Thrust, Mass, and Inertia Influence Design

A key requirement for this project is scalability—the ability to apply the same design principles to rockets of varying sizes and power levels. The physics of flight control dictates a strong interdependency between the rocket's physical characteristics and the demands placed on the TVC system.

- **Thrust and Actuator Force:** The magnitude of the required control torque scales directly with the engine thrust. A high-power motor producing hundreds of Newtons of thrust will generate a much larger lateral force for a given gimbal angle than a small hobby motor. This force must be counteracted by the TVC actuators. Therefore, as thrust increases, the servos must be proportionally stronger, and the gimbal structure must be robust enough to handle these higher loads.[20]
- **Mass, Inertia, and Angular Acceleration:** The rocket's mass and its distribution determine its moment of inertia (I). A larger, heavier rocket will have a significantly higher moment of inertia. According to the rotational dynamics equation $\tau = I\alpha$, a larger torque is required to produce the same angular acceleration for a more massive vehicle.[15] This further drives the need for more powerful actuators on larger rockets.
- **Dynamic Response and Control Speed:** The scale of the rocket also influences its dynamic behavior. Smaller, lighter rockets tend to have very low moments of inertia, making them highly responsive or "twitchy." They can be disturbed and rotate very quickly, demanding an extremely fast and high-bandwidth control system to maintain stability. Conversely, larger vehicles have much higher inertia and thus slower angular dynamics. This phenomenon is sometimes summarized by the observation that "things get easier to control the larger you go".[7] While the forces involved are greater, the slower response time can relax the computational speed and actuator bandwidth requirements of the control loop, though this is a complex trade-off.

A crucial distinction arises when considering the phase of flight. TVC is only effective while the motor is producing thrust.[1] After motor burnout, the rocket is in a coasting phase, and the TVC system has no control authority. In contrast, aerodynamic surfaces like fins are only effective when the rocket has sufficient airspeed.[7] This creates a potential "control authority gap." A pure TVC rocket becomes uncontrollable after burnout, relying on momentum and any residual passive stability to reach apogee. This suggests that a truly robust, high-performance system might be a hybrid, using TVC for the initial powered ascent and then transitioning to aerodynamic control surfaces (either static or active) for the coast phase. This shifts the design problem from simply building a TVC mount to engineering a comprehensive flight control system for all phases of flight.

Section 2: Mechanical Design, Fabrication, and Analysis of a Scalable TVC System

Transitioning from theoretical principles to a functional system requires a rigorous approach to mechanical design and fabrication. This section provides a detailed guide to creating the physical TVC hardware, with a focus on accessible manufacturing techniques like 3D printing, while incorporating the engineering analysis necessary for a reliable, safe, and scalable design.

2.1 Designing the Gimbal Assembly: CAD, Kinematics, and Linkages

The heart of the mechanical system is the two-axis gimbal, which provides independent control over the rocket's pitch and yaw.[2] A typical design consists of an inner ring that holds the motor and pivots along one axis (e.g., pitch), which is itself mounted inside an outer ring that pivots on an orthogonal axis (e.g., yaw).[2]

The use of Computer-Aided Design (CAD) software is indispensable for this task. Programs like Autodesk Fusion 360 or OnShape are ideal for modeling the complex, interlocking parts of the gimbal.[5] Rather than starting from a blank slate, it is highly recommended to study and adapt existing designs. A vibrant community of hobbyists and professionals has made numerous designs available, both open-source and for purchase, which serve as excellent, flight-proven starting points.[20]

A critical aspect of the design is the kinematics of the linkage between the servo motors and the gimbal rings. The goal is to translate the small, rotational motion of a servo horn into precise, controlled tilting of the motor. Since the required gimbal deflection angles are typically small (often just ±5 degrees), it is advantageous to design the linkage with a mechanical advantage that "gears down" the motion.[3] A gear ratio of 2:1 or even as high as 7:1 means that the servo must rotate several degrees to produce one degree of nozzle deflection.[3] This trade-off sacrifices a large range of motion, which is not needed, for a significant increase in both precision and the effective torque applied to the gimbal, allowing smaller servos to control larger forces.[25]

2.2 Material Selection and Additive Manufacturing for Aerospace Applications

For hobbyist-scale TVC systems, additive manufacturing (3D printing) is the key enabling technology, making the fabrication of complex gimbal geometries both affordable and accessible.[2] The choice of material is paramount and must be scaled with the power of the

rocket motor.

- **For Low-to-Mid Power Applications (e.g., Estes D-F class, up to ~40 N thrust):** Polylactic Acid (PLA) is often a suitable material for the gimbal components. It is rigid, strong, and one of the easiest materials to print with high precision.[20] For the airframe components that hold the TVC mount, materials with better layer adhesion and impact resistance, such as PETG, may be preferred.
- **For High-Power Applications (e.g., Aerotech G-class and above):** As thrust levels and airframe stresses increase, more advanced materials are necessary. Acrylonitrile Styrene Acrylate (ASA) is an excellent choice for airframe components due to its superior strength, UV resistance, and higher temperature tolerance compared to PLA.[26] For the gimbal parts themselves, especially those in close proximity to the hot motor casing, high-temperature polymers like Nylon or Polycarbonate, often reinforced with carbon fiber, should be considered. In extreme cases, such as for jet vanes that are directly in the exhaust, professional additive manufacturing techniques like Selective Laser Sintering (SLS) with materials like titanium may be required to withstand the harsh environment.[11]

2.3 Actuator Sizing: A-Priori Torque and Speed Analysis for Servo Selection

Selecting the correct servo motors is one of the most critical steps in designing a scalable and reliable TVC system. An undersized servo will fail to hold the nozzle angle against the engine's thrust, leading to a complete loss of control. The selection process must be based on a careful analysis of both the static and dynamic loads the servo will experience.

The primary load is the static force from the vectored thrust. The torque required from the servo can be estimated using a simple static analysis of the gimbal linkage. The side force generated by the motor is $F_{side} = T_{max} \cdot \sin(\delta_{max})$, where T_{max} is the motor's maximum thrust and δ_{max} is the maximum required gimbal angle. This force creates a torque on the gimbal, which must be balanced by the force from the servo acting through its linkage. A simplified formula for the required linear force from the actuator is:

$$F_{actuator} \approx \frac{T_{max} \sin(\delta_{max}) \cdot L_{gimbal}}{L_{servo}}$$

where L_{gimbal} is the distance from the gimbal pivot to the motor's line of action and L_{servo} is the effective moment arm of the servo linkage.[21] This force is then converted to a required servo torque based on the length of the servo horn.

However, static torque is only half of the story. The system is dynamic; the controller must be able to move the nozzle quickly to react to disturbances. The servo must provide additional torque to overcome the inertia of the gimbaled mass (the motor and nozzle) and achieve the necessary angular acceleration (α_{gimbal}). This dynamic torque is given by $\tau_{dynamic} = I_{motor} \cdot \alpha_{gimbal}$, where I_{motor} is the moment of inertia of the gimbaled components about the pivot axis.[15] This dynamic requirement dictates the necessary *speed* of the servo, typically rated in seconds per 60 degrees. A servo that is strong enough

but too slow will have a low control bandwidth and may not be able to stabilize a fast-acting disturbance, leading to destructive oscillations.[10]
For these reasons, high-quality digital servos with metal gears are strongly recommended. Digital servos offer superior holding torque and precision, while metal gears are essential for durability against the high loads and intense vibrations of a rocket motor burn.[29] The following table provides a starting point for selecting appropriate servos based on the rocket's motor class.

Table 2.1: Servo Motor Selection Matrix for TVC Applications

Motor Class	Typical Max Thrust (N)	Calculated Min. Stall Torque (kg-cm)*	Recommended Servo Class	Example Servo Models
D - F	15 - 40	3 - 8	High-Torque Micro	Hitec HS-5085MG, KST DS215MG
G - H	40 - 200	8 - 30	High-Torque Standard	Savox SA-1283SGP, Hitec D955TW
I - K	200 - 1000	30 - 150+	High-Voltage "Monster"	Savox SB2292SG, Hitec D-840WP

**Note: Calculated torque is an estimate assuming a max gimbal angle of 5 degrees and typical linkage geometry. A safety factor of 1.5-2.0 is highly recommended. Final selection must be based on a detailed analysis of the specific gimbal design.*

2.4 Structural Integration, Load Paths, and Airframe Modifications

The TVC assembly cannot be treated as an isolated component; it must be fully integrated into the rocket's airframe, and all structural load paths must be considered. The entire thrust of the rocket motor is channeled through the gimbal pivot points, into the TVC mount structure, and finally into the main airframe tubes. These connection points must be exceptionally strong.[5]
A common and effective design pattern involves using a series of 3D-printed centering rings connected by rigid, lightweight rods made of carbon fiber or fiberglass.[3] This creates a "power cage" or thrust frame that houses the TVC mechanism, flight computer, and battery. This entire assembly can then be securely mounted within the main body tube of the rocket. This modular approach facilitates assembly, testing, and maintenance.
It is also important to recognize that the mechanical system is not perfectly rigid. The 3D linkages can introduce geometric non-linearities, where the relationship between servo input and nozzle output is not constant across the range of motion, and can even exhibit cross-coupling between the pitch and yaw axes.[28] While a challenge for classical controllers,

these complex, real-world behaviors are precisely the kind of problem that a well-trained reinforcement learning agent can learn to manage implicitly. The neural network policy does not need an explicit mathematical model of these non-linearities; it can learn the correct control outputs required to achieve the desired result through trial and error in a sufficiently accurate simulation.[32]

Section 3: The Guidance, Navigation, and Control (GNC) Electronics Suite

The GNC suite constitutes the "brain and nervous system" of the rocket. It is responsible for sensing the rocket's state, processing that information, and executing the control policy to command the actuators. For a system intended to run an AI model in real-time, the selection of high-performance electronic components is not just a recommendation, but a strict requirement.

3.1 The Flight Controller: Selecting a Microcontroller for Real-Time Embedded AI

The central component of the GNC system is the microcontroller unit (MCU), or flight computer. Its primary job is to execute the high-frequency control loop: read sensors, perform state estimation calculations, run the control algorithm (in this case, a neural network inference), and send updated commands to the servos.[24] The speed at which this loop can be executed, known as the control loop frequency, is a critical performance metric. For a dynamically unstable vehicle like a TVC rocket, frequencies well above 100 Hz are often required.

While a basic MCU like an Arduino Uno can be used for simple demonstrations with a PID controller, its limited processing power and lack of a hardware Floating Point Unit (FPU) make it entirely unsuitable for running a deep reinforcement learning model.[2] The mathematical operations involved in neural network inference, particularly matrix multiplications, are computationally intensive. A high-performance MCU is essential.

- **Teensy 4.x Series:** This family of MCUs is highly recommended for demanding embedded applications. The Teensy 4.1, for example, features a 600 MHz ARM Cortex-M7 processor, a hardware FPU for both single and double-precision floating-point math, 1 MB of RAM, and 8 MB of flash memory.[34] Its raw computational power, extensive set of peripherals (including multiple UARTs, I2C, SPI, and an SD card slot), and strong community support via the Teensyduino add-on for the Arduino IDE make it an ideal choice for this project.[24]
- **ESP32 Series:** The ESP32 is another powerful and popular choice, known for its dual-core 240 MHz processor and integrated Wi-Fi and Bluetooth capabilities.[34] The

wireless connectivity is a major advantage for in-flight telemetry and on-the-pad configuration. However, its single-core performance and FPU capabilities are generally lower than the Teensy 4.x, which may become a limiting factor for achieving the lowest possible inference latency.[36]

The choice between these platforms depends on the project's priorities. If maximum computational performance for the fastest possible control loop is the goal, the Teensy 4.1 is the superior option. If integrated connectivity and a lower cost are more important, and the control loop requirements are slightly less stringent, the ESP32 is a very capable alternative.

Table 3.1: Comparative Analysis of MCUs for Real-Time Control and ML Inference

Microcontroller	Processor Core	Clock Speed (MHz)	RAM (KB)	Flash (MB)	FPU Support	Key Peripherals	Suitability for TVC/DRL
Arduino Uno	ATmega328P	16	2	0.032	No	I2C, SPI, UART	**Inadequate.** Suitable only for basic PID demonstrations. Lacks memory and processing power for ML inference.
ESP32	Dual-Core Xtensa LX6	240	520	4 - 16	Single-Precision	I2C, SPI, UART, CAN, SDIO, Wi-Fi, Bluetooth	**Viable.** Good processing power and excellent connectivity. A strong choice, but may be outperformed by Teensy in raw inference speed.
Teensy 4.1	ARM Cortex-M7	600	1024	8	Single & Double	I2C, SPI, UART, CAN, SDIO,	**Excellent.** Class-leading

						Ethernet (PHY), USB Host	processing speed and FPU make it ideal for minimizing control loop latency and running complex state estimation and ML models.

3.2 Inertial Sensing and State Estimation: IMU Selection and Data Fusion

The flight controller's perception of the world comes from an Inertial Measurement Unit (IMU). This sensor package typically contains a three-axis gyroscope to measure angular velocity and a three-axis accelerometer to measure linear acceleration.[3]

- **MPU-6050:** A very common and inexpensive 6-DOF (Degrees of Freedom) IMU. It provides raw accelerometer and gyroscope data. While functional for getting started, it can be susceptible to noise and requires the main MCU to perform all sensor fusion calculations.[38]
- **BNO055:** A 9-DOF IMU that adds a three-axis magnetometer for sensing heading relative to Earth's magnetic field. Its key feature is an integrated ARM Cortex-M0+ coprocessor that runs a proprietary sensor fusion algorithm.[41] This allows the BNO055 to output fully processed, stable orientation data in the form of Euler angles or quaternions directly, offloading this computationally expensive task from the main MCU.[6] However, the internal update rate of this fusion algorithm may be a bottleneck for very high-speed control loops, and its "black box" nature offers less customizability.
- **ICM-20948 / BNO086:** These are more advanced 9-DOF sensors that offer higher data rates, lower noise, and more powerful onboard processing than their predecessors. They represent a good middle ground, providing high-quality data and some onboard fusion capabilities while still allowing for custom filtering on the main MCU.[38]

3.3 Deep Dive: Implementing Kalman Filters and Quaternions for

Robust Attitude Estimation

Raw data from an IMU is insufficient for reliable control. Gyroscopes, while excellent at measuring rapid changes in orientation, suffer from bias and drift over time. An initially level gyroscope will slowly report that it is rotating, even when stationary.[42] Accelerometers can measure the direction of gravity to provide a long-term sense of "down," but they cannot distinguish between the force of gravity and the linear acceleration of the rocket itself.[42] During ascent, the accelerometer data is dominated by the engine's thrust.

The solution is **sensor fusion**, a process that combines the strengths of both sensors to produce an optimal estimate of the true state. The most common and effective algorithm for this is the **Kalman Filter**.[3] A Kalman filter is a recursive algorithm that continuously updates its estimate of the rocket's orientation. In each step, it uses the gyroscope data to predict the new orientation and then uses the accelerometer data (and magnetometer, if available) to correct that prediction, effectively filtering out the noise and drift from the individual sensors.[3]

When representing 3D rotations in software, using simple Euler angles (pitch, yaw, roll) can lead to a mathematical singularity known as **gimbal lock**, where two rotational axes align, causing a loss of one degree of freedom.[3] The robust and standard solution in aerospace and robotics is to use
Quaternions. A quaternion is a four-dimensional number that can represent any 3D rotation without singularities.[3] The GNC software pipeline should therefore operate as follows:

1. Read raw angular rates and accelerations from the IMU.
2. Feed these measurements into a Kalman filter.
3. The internal state of the filter, representing the rocket's orientation, should be maintained as a quaternion.
4. For input to the control algorithm or for telemetry, this quaternion can be converted back to Euler angles as needed.[3]

This entire process—reading sensors, running the filter, executing the control policy—is not a set of independent tasks but a single, high-frequency data processing pipeline. The end-to-end latency of this pipeline is the most critical performance metric of the entire GNC system. Each component—the IMU's sample rate, the filter's execution time, the AI model's inference time—contributes to this latency. This holistic view of the system as a pipeline reinforces the need for high-performance components at every stage.

3.4 Power Architecture and System Integration

The power delivery system must be designed to handle the significant and highly variable current demands of the TVC system. The servo motors, especially when working against the high forces of the engine thrust, can draw several amperes of current in brief spikes.[8]

A common and reliable architecture uses a 2S or 3S Lithium Polymer (LiPo) battery (providing a nominal 7.4 V or 11.1 V) to directly power the servos, which are typically designed for these

higher voltages.[8] A separate, dedicated voltage regulator, often called a Battery Eliminator Circuit (BEC), is then used to step this voltage down to a clean, stable 5 V or 3.3 V for the sensitive electronics, including the MCU and IMU. This isolation is critical to prevent the electrical noise and voltage drops caused by the servo motors from interfering with the flight computer, which could lead to a catastrophic processor reset in mid-flight.
While initial prototyping can be done with breadboards and jumper wires, a custom Printed Circuit Board (PCB) is highly recommended for the final flight hardware.[26] A PCB provides a much more reliable, vibration-resistant, and lightweight solution for connecting all the electronic components, significantly increasing the overall robustness of the GNC system.

Section 4: Control Paradigms: From Classical PID to Reinforcement Learning

With the physical hardware and electronic sensing systems defined, the focus shifts to the intelligence that will command them. This section explores the control algorithms, starting with a traditional engineering baseline and then transitioning to the more advanced and flexible paradigm of Reinforcement Learning (RL), which lies at the core of the user's request.

4.1 Establishing a Baseline: Implementing and Tuning a PID Controller

Before diving into AI, it is invaluable to implement a classical controller as a baseline for performance comparison. The Proportional-Integral-Derivative (PID) controller is the workhorse of industrial control systems and provides a solid foundation for understanding the rocket's control problem.[3]
A PID controller operates in a feedback loop, calculating an error value e(t) as the difference between a desired setpoint and the measured process variable. It then computes a corrective output by summing three terms [3]:

- **Proportional (P) Term:** This term is proportional to the current error ($K_p \cdot e(t)$). It provides the primary corrective action. For the rocket, if it is tilted by 2 degrees, the P term commands a proportional gimbal deflection to push it back.[4]
- **Integral (I) Term:** This term accumulates past errors over time ($K_i \cdot \int e(t)dt$). It is responsible for eliminating any steady-state error, such as a slow drift caused by a constant crosswind or a slight thrust misalignment.[3]
- **Derivative (D) Term:** This term is proportional to the rate of change of the error ($K_d \cdot \frac{de(t)}{dt}$). It acts as a damper, resisting rapid changes. If the rocket is tipping over quickly, the D term will command a strong opposing action to slow the rotation, preventing overshoot and oscillations.[3]

For a TVC rocket, the implementation typically involves two or three independent PID loops: one for pitch control and one for yaw control.[3] The input to each loop is the angular error for

that axis (e.g.,
target_pitch - current_pitch), and the output is the commanded angle for the corresponding servo motor.[3]
While simple in concept, the primary challenge of PID control lies in **tuning** the three gain constants (Kp,Ki,Kd). These values are highly dependent on the specific rocket's mass, inertia, and motor thrust. Finding the optimal set of gains that results in a stable, responsive flight without oscillation is a difficult and iterative process, often requiring extensive simulation and trial-and-error.[26] This tuning difficulty is a powerful motivation for exploring machine learning approaches that can automate the discovery of an optimal control strategy.

4.2 Introduction to Reinforcement Learning for Dynamic Control Systems

Reinforcement Learning is a paradigm of machine learning where an autonomous agent learns to achieve a goal by interacting with a complex, uncertain environment.[32] Unlike supervised learning, the agent is not given a dataset of "correct" answers. Instead, it learns through trial and error, receiving a numerical
reward signal from the environment that provides feedback on its actions.[46] The agent's sole objective is to learn a
policy—a strategy for choosing actions—that maximizes the total cumulative reward it receives over time.[32]
Mapping these core RL concepts to the TVC rocket problem provides a clear framework [33]:

- **Agent:** The AI control policy, represented by a neural network, running on the flight computer.
- **Environment:** The rocket itself, along with the laws of physics (gravity, aerodynamics, kinematics) that govern its flight.
- **Action (at):** The output of the policy at a given time t. For TVC, this is a vector containing the commanded gimbal angles for the pitch and yaw servos.
- **State/Observation (st):** The set of data the agent uses to make a decision. This is the output of the GNC system's state estimator, including the rocket's current attitude, angular rates, etc.
- **Reward (rt):** A scalar feedback signal generated after each action. The reward function is designed by the engineer to numerically represent the goals of the mission (e.g., positive reward for staying upright, negative reward for falling over).

The primary advantage of RL over classical control methods is its ability to learn optimal control policies for highly complex and non-linear systems without requiring an explicit, perfect mathematical model of those systems.[32] The neural network policy can learn to control subtle dynamics, such as non-linear actuator responses or complex aerodynamic effects, simply by observing the outcomes of its actions within a simulation.

4.3 Formulating the TVC Problem as a Markov Decision Process (MDP)

The formal mathematical framework for RL is the Markov Decision Process (MDP).[45] An MDP formalizes the agent-environment interaction loop and consists of a set of states, actions, transition probabilities, and a reward function. To apply RL to the TVC problem, we must define these components.

4.3.1 Defining the State Space (Observations)

The state space is the complete set of information the agent receives from the environment to make its decisions. A well-designed state space is crucial; it must contain enough information for the agent to learn an effective policy without being overly complex. For the TVC rocket, a robust state vector would include [18]:

- **Attitude:** The rocket's orientation in 3D space. To avoid the issues of gimbal lock with Euler angles, this is best represented by a 4-element quaternion or by the sine and cosine of the pitch and yaw angles.
- **Angular Velocities:** The rates of rotation around the pitch, yaw, and roll axes. This is critical for the agent to learn damping behavior.
- **Position/Velocity (Optional but recommended):** While not strictly necessary for pure stabilization, including vertical velocity and altitude can help the agent learn different behaviors for different phases of flight.
- **Time/Fuel (Optional):** Including a time-step counter or an estimate of remaining motor burn time can allow the agent to learn time-dependent strategies.[18]

4.3.2 Defining the Action Space

The action space defines the set of all possible commands the agent can issue. This can be either continuous or discrete.

- **Continuous Action Space:** The policy network outputs a continuous floating-point value for each actuator command (e.g., a pitch angle between -5.0 and +5.0 degrees). This allows for very fine and smooth control and is the most natural representation of the problem. It requires RL algorithms specifically designed for continuous spaces, such as PPO or SAC.[50]
- **Discrete Action Space:** The policy chooses one action from a predefined, finite set. For example, the action space for the pitch servo could be {Gimbal -1°, Gimbal 0°, Gimbal +1°}.[19] While this simplifies the learning problem, it can result in less precise, "jerky" control.

For high-performance control, a **continuous action space** is strongly recommended.

4.3.3 The Art of Reward Function Engineering for Rocket Stability

The reward function is the most critical element of the MDP formulation. It is the sole mechanism by which the engineer communicates the desired behavior to the agent. The design of this function is more of an art than a science, often requiring iteration and experimentation. A good reward function for the TVC stabilization task should incentivize staying upright while discouraging inefficient or unstable behavior. A multi-component, or "shaped," reward function is often effective:

- **Primary Objective (Staying Upright):** A large, continuous positive reward for maintaining a small deviation from vertical. A common formulation is an exponential reward based on the total tilt angle, $\theta_{total} = \theta_{pitch}^2 + \theta_{yaw}^2$:
  $R_{attitude} = \exp(-k_{angle} \cdot \theta_{total}^2)$
  This gives the highest reward at zero tilt and smoothly decreases as the rocket tilts over.[18]
- **Penalties (Negative Rewards):**
  - **Termination Penalty:** A large negative reward if the tilt angle exceeds a predefined failure threshold (e.g., 20 degrees). This strongly teaches the agent to avoid catastrophic failure.
  - **Angular Velocity Penalty:** A small negative reward proportional to the square of the angular velocities. This encourages the agent to learn a smooth, stable flight path rather than oscillating wildly.[33]
  - **Control Effort Penalty:** A small negative reward proportional to the square of the commanded action (gimbal angle). This incentivizes the agent to use as little control effort as necessary, leading to more efficient and stable control.[33]

The total reward at each timestep is the sum of these components: $R_{total} = R_{attitude} - C_{vel} \cdot \omega^2 - C_{action} \cdot a^2$. The constants ($k_{angle}, C_{vel}, C_{action}$) are hyperparameters that must be tuned to balance the competing objectives.

This process of reward engineering highlights a fundamental difference between classical control and RL. With a PID controller, the engineer directly specifies the control law by tuning gains. With RL, the engineer specifies the *definition of success* through the reward function, and the RL algorithm's task is to automatically discover a control law that optimizes for that definition. This shifts the engineer's role from low-level parameter tuning to a higher-level, goal-oriented design process.

Section 5: Training the AI Pilot: Deep Reinforcement Learning in a Simulated Environment

Training a reinforcement learning agent for a physical system like a rocket is a process that must occur entirely within a simulated environment. The trial-and-error nature of RL would

make training on real hardware prohibitively expensive, time-consuming, and dangerous.[26] This section details the creation of a suitable digital training ground, the selection of an appropriate DRL algorithm and neural network architecture, and the methodology for training and validating the resulting AI control policy.

5.1 Constructing the Digital Twin: Physics Simulation with PyBullet and RocketPy++

The quality of the learned policy is directly dependent on the fidelity of the simulation. The simulation acts as the agent's "world model," and if this model is inaccurate, the agent will learn a policy that is optimal for the flawed simulation but will fail in the real world. This makes the development of a "digital twin"—a simulation that accurately reflects the real rocket's physics—a critical first step.

For this task, a 3D physics engine is required. Two powerful Python-based tools are particularly well-suited:

- **PyBullet:** This is a robust, open-source physics engine widely used in robotics and reinforcement learning research.[53] It provides fast and accurate simulation of rigid body dynamics, including forces, torques, joints, and collision detection.[55] A model rocket can be defined using a URDF (Unified Robot Description Format) file, and PyBullet can simulate its motion in response to gravity and applied forces from the TVC system. Projects like **PyRocketCraft** have successfully demonstrated the use of PyBullet for simulating and controlling a landing rocket, making it an excellent choice for this project.[55]
- **RocketPy:** This is a specialized, open-source 6-DOF trajectory simulation library created specifically for high-power rocketry.[56] Its strengths lie in its high-fidelity modeling of rocket-specific phenomena, such as aerodynamics based on Barrowman equations, the effects of variable mass as propellant is consumed, and the integration of real-world weather forecast data.[58] While not a general-purpose physics engine like PyBullet, RocketPy's core simulation can be scripted and integrated into an RL training loop, providing highly accurate trajectory and aerodynamic data.[60]

A hybrid approach could leverage the strengths of both: using PyBullet for the core rigid-body dynamics and motor actuation, while using RocketPy's models to calculate and apply realistic aerodynamic forces to the body in each simulation step.

Regardless of the backend, the simulation must be wrapped in a standardized interface to be compatible with common RL libraries. The **Gymnasium** (formerly OpenAI Gym) interface is the de facto standard.[55] This involves creating a Python class for the environment that has two primary methods: a reset() method that initializes the rocket to its starting state at the beginning of an episode, and a step(action) method that takes an action from the agent, advances the simulation by one time step, and returns the new observation, the reward, and a flag indicating if the episode has ended.

5.2 Algorithm Selection: A Technical Comparison of PPO and SAC for Rocket Control

With the environment defined, the next step is to select a DRL algorithm capable of solving the continuous control problem. Two of the most powerful and widely used algorithms in this domain are Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC). Both are actor-critic methods, meaning they learn both a policy (the actor) and a value function to estimate the quality of states and actions (the critic).[32]

- **Proximal Policy Optimization (PPO):** PPO is an **on-policy** algorithm known for its stability, reliability, and excellent performance across a wide range of tasks.[51] Its key innovation is a "clipped surrogate objective function," which constrains the size of policy updates at each training step.[62] This prevents the agent from making destructively large changes to its policy based on a single batch of data, leading to smoother and more reliable convergence. PPO is often considered a strong default choice for continuous control problems and has been successfully applied to spacecraft attitude control.[49]
- **Soft Actor-Critic (SAC):** SAC is a state-of-the-art **off-policy** algorithm based on the maximum entropy reinforcement learning framework.[65] Unlike traditional RL, which seeks only to maximize the cumulative reward, SAC's objective is to maximize a combination of the expected reward
  and the entropy of the policy. Entropy is a measure of randomness; by also rewarding the policy for being as random as possible (while still achieving the goal), SAC encourages broad exploration.[66] This often leads to two key benefits:
  1. **Higher Sample Efficiency:** Being off-policy allows SAC to reuse old data more effectively, often learning a good policy with fewer simulation steps than PPO.
  2. **Increased Robustness:** The resulting high-entropy policies are less "brittle." They are not over-specialized to a single solution but maintain a degree of randomness, which often makes them more resilient to unexpected disturbances and better able to generalize from simulation to the real world.[65]

Recommendation: While PPO is a robust and proven choice, **SAC is recommended as the more advanced option for this application.** Its emphasis on entropy and robustness is highly desirable for a physical system like a rocket, where the real-world environment will inevitably present conditions not perfectly captured in the simulation.[65] PPO remains an excellent and potentially easier-to-tune alternative.

5.3 Designing the Neural Network Architecture for the Policy and Value Functions

Both PPO and SAC utilize neural networks to approximate the actor (policy) and critic (value)

functions. For a state space like the one defined in Section 4.3, a simple Multi-Layer Perceptron (MLP) is a sufficient and effective architecture.

- **Network Structure:** A typical design for both the actor and critic networks would consist of:
    - An **input layer** with a number of neurons equal to the dimension of the state space (e.g., ~8-12 neurons).
    - Two or three **hidden layers**, each with between 128 and 256 neurons. The Rectified Linear Unit (ReLU) is a standard and effective activation function for these layers.[52]
    - An **output layer** whose structure depends on the network's role:
        - **Actor (Policy) Network:** For a continuous action space with two actions (pitch and yaw gimbal), the output layer would typically output four values: the mean for the pitch action, the standard deviation for the pitch action, the mean for the yaw action, and the standard deviation for the yaw action. The final action is then *sampled* from the Gaussian distributions defined by these means and standard deviations.[51]
        - **Critic (Value) Network:** The output layer consists of a single neuron with a linear activation function, which outputs a single scalar value representing the estimated Q-value (the expected cumulative reward) for the given state and action.[68]

These networks can be easily implemented using standard deep learning frameworks like **PyTorch** or **TensorFlow**.[55]

5.4 The Training Loop: From Environment Interaction to Policy Optimization

The training process is an automated loop that can run for hours or days on a powerful computer. The general steps are as follows:

1. Initialize the actor and critic neural networks with random weights.[66]
2. The agent begins interacting with the simulated environment. For each step, it observes the state, uses its current policy (actor network) to choose an action, and executes that action in the simulation.
3. The simulation returns the next state, the reward for that action, and a termination flag.
4. This (state, action, reward, next_state) tuple, known as an experience, is stored in a large database called a **replay buffer** (this is particularly crucial for off-policy algorithms like SAC).[66]
5. Periodically, the algorithm samples a random mini-batch of these experiences from the replay buffer.
6. This mini-batch is used to train the networks. First, the critic network is updated by minimizing the temporal-difference (TD) error—the difference between its prediction and a more accurate target value calculated from the rewards in the batch.

7. Then, the actor network is updated by performing gradient ascent on the objective function, using the critic's value estimates to guide the policy towards actions that lead to higher expected rewards.[51]
8. This loop of data collection and network optimization is repeated for hundreds of thousands or millions of simulation steps until the agent's performance converges.

5.5 Analyzing Training Performance, Ensuring Convergence, and Validating the Policy

It is essential to monitor the training process to ensure the agent is learning effectively. Tools like **TensorBoard** or **Weights & Biases (wandb)** should be used to log and visualize key metrics in real-time.[69] The most important metric is the average cumulative reward per episode. A successful training run will show this reward curve steadily increasing over time before eventually plateauing, indicating that the policy has converged to an optimal or near-optimal solution.[18]
Once training is complete, the final policy must be rigorously validated. This involves running the trained agent (in deterministic mode, without exploration noise) for a large number of episodes in the simulation, ideally with randomized initial conditions. This helps verify that the policy is not only high-performing but also robust and reliable across a range of scenarios before it is considered for deployment on physical hardware.

Section 6: From Simulation to Reality: Deployment and Low-Latency Onboard Inference

The culmination of the design and training process is the deployment of the AI pilot onto the physical rocket's flight controller. This is a critical and challenging phase that involves bridging the "sim-to-real" gap, optimizing the neural network for an embedded environment, and implementing a low-latency inference engine to make real-time decisions.

6.1 The "Sim2Real" Challenge: Strategies for Bridging the Reality Gap

A policy trained in a pristine, idealized simulation will almost certainly fail when deployed on real hardware. The real world is noisy and imperfect, with factors that are difficult to model perfectly, such as sensor noise, actuator delays, motor thrust variability, and unpredictable wind gusts. The gap between the simulation and reality is known as the "sim-to-real" gap. Several strategies can be employed during training to create a more robust policy that can cross this gap.
The most powerful technique is **domain randomization**. Instead of training the agent in a

single, static simulation, it is trained in a multitude of slightly different simulations. At the start of each training episode, the physical parameters of the simulated environment are randomized within a certain range. For example, the rocket's mass, center of gravity, motor thrust profile, IMU sensor noise characteristics, and servo response latency can all be slightly varied. This forces the agent to learn a control policy that is not over-fitted to one specific set of parameters but is robust and effective across the entire distribution of randomized environments. A policy trained this way is far more likely to generalize successfully to the real world, which it will perceive as just another variation of the environment it has already seen. Additionally, performing **system identification** on the physical rocket is crucial. This involves carefully measuring the true mass, inertia, and dimensions of the assembled rocket and using these precise values in the simulation. The actual thrust curve of the chosen motor type should also be used instead of an idealized one. The more accurately the simulation reflects reality, the smaller the sim-to-real gap will be.

6.2 Model Optimization for Embedded Systems: A Primer on Neural Network Quantization

A neural network trained in a framework like PyTorch or TensorFlow stores its parameters (weights and biases) as 32-bit floating-point numbers. For a typical policy network, this can result in a model file that is several megabytes in size. Running inference with this model requires numerous floating-point matrix multiplications, which are computationally expensive for a microcontroller, even one with a hardware FPU.[71]
To make the model suitable for an embedded device, it must be optimized. The primary technique for this is **quantization**. Quantization is the process of converting the model's 32-bit floating-point numbers into a lower-precision format, most commonly 8-bit signed integers (int8).[71] This optimization yields three significant benefits [72]:

1. **Reduced Model Size:** An int8 model is approximately four times smaller than its float32 counterpart, making it much easier to fit within the limited flash memory of an MCU.
2. **Faster Inference:** Integer arithmetic operations are significantly faster than floating-point operations on most embedded processors. This reduction in computational complexity is the key to achieving the low-latency inference required for a high-frequency control loop.
3. **Lower Power Consumption:** Fewer and simpler computations result in lower energy use, which is beneficial for battery-powered systems.

There are two main approaches to quantization:

- **Post-Training Quantization (PTQ):** This is the simplest method. A fully trained float32 model is fed into a converter tool, which quantizes the weights and activations to int8. This can be done with little effort but may sometimes result in a slight loss of model accuracy.[71]
- **Quantization-Aware Training (QAT):** This method simulates the effects of quantization *during* the training or fine-tuning process. It allows the model to adapt to

the lower-precision format, often resulting in higher accuracy for the final quantized model at the cost of a more complex training setup.[75]

6.3 A Step-by-Step Guide to Deploying with TensorFlow Lite for Microcontrollers (TFLM)

TensorFlow Lite for Microcontrollers (TFLM) is a specialized version of the TensorFlow framework designed to run machine learning models on resource-constrained devices like MCUs.[76] It provides the necessary tools and libraries to take a trained model and execute it efficiently on bare-metal hardware.

The deployment workflow is a multi-step process:

1. **Train and Save the Model:** After the DRL training process is complete, the final policy network is saved in its native framework format (e.g., TensorFlow SavedModel or PyTorch checkpoint).
2. **Convert to TensorFlow Lite Format:** The saved model is then passed through the TensorFlow Lite Converter. This tool converts the model into the highly efficient .tflite FlatBuffer format. This is also the stage where post-training quantization is typically applied.[71]
3. **Convert to a C Array:** The .tflite file, which is a binary file, is then converted into a C/C++ header file (.h) using a tool like xxd. This header file contains the entire model represented as a large unsigned char array.[77]
4. **Integrate into Firmware Project:** This generated header file is included in the flight controller's firmware project (e.g., an Arduino sketch for Teensy or an ESP-IDF project for ESP32).
5. **Implement the Inference Code:** Within the firmware, the TFLM C++ library is used to run the model. The key steps in the code are [77]:
    - Include the TFLM library headers and the model's header file.
    - Statically allocate a region of memory, called the tensor_arena, which TFLM will use for all its operations, including storing the input and output tensors and intermediate activation values.
    - Instantiate the TFLM MicroInterpreter.
    - In the main control loop, copy the latest state vector data (from the Kalman filter) into the model's input tensor.
    - Call interpreter->Invoke() to run a single forward pass of the neural network.
    - Read the resulting action vector from the model's output tensor and send the values to the servo motors.

This entire process transforms the abstract, Python-based AI model into a single, static C-array that is compiled directly into the rocket's firmware. The complex, adaptive learning phase is entirely offline; the onboard "intelligence" is a highly optimized, fixed mathematical function represented by this array of numbers.

6.4 Achieving Fast Decisions: The Mechanics of Low-Latency Inference on an MCU

The user's query about making "fast decisions" is addressed directly by this deployment pipeline. The speed of the decision-making process is the inference latency—the time it takes for the MCU to execute the interpreter->Invoke() function.
By leveraging a powerful MCU with a high clock speed and hardware FPU (like the Teensy 4.1) and running a model that has been heavily optimized through 8-bit quantization, this inference time can be reduced to the sub-millisecond range for a moderately sized MLP. This extremely low latency is what enables the GNC control loop to run at hundreds of Hertz. By making corrective decisions much faster than the rocket's physical dynamics can evolve, the AI controller can maintain stability even in the face of sudden disturbances, effectively staying ahead of any potential instability.[65] The "fast decision" is therefore not a result of on-the-fly learning, but of the rapid execution of a pre-compiled, highly optimized, and pre-learned function.

Section 7: System Integration, Testing, and Advanced Topics

The final phase of the project involves bringing together the mechanical, electronic, and software subsystems into a cohesive whole. This requires a methodical approach to integration, calibration, and testing, with safety as the paramount concern. Once a stable system is achieved, it becomes a powerful platform for exploring even more advanced control objectives.

7.1 End-to-End System Integration, Calibration, and Ground Testing

Before any flight attempt, the fully assembled rocket must undergo rigorous calibration and ground testing.

- **Component Calibration:** This is a non-negotiable first step. The IMU must be calibrated on a level surface to determine and correct for any inherent gyroscope and accelerometer biases.[24] The TVC servos must also be calibrated; their mechanical linkages should be adjusted so that the neutral '90-degree' command corresponds to a perfectly aligned motor nozzle, and the relationship between commanded servo angle and actual nozzle deflection should be measured and accounted for in the control software.[2]
- **Static Fire / Hover Test:** The most important ground test is a static fire test.[9] The rocket should be securely fastened to a robust test stand that allows it to operate under

full thrust without taking off. This allows the entire GNC loop—sensors, state estimator, AI controller, and actuators—to be tested under the realistic, high-vibration conditions of a motor burn. The system's ability to maintain a vertical orientation can be visually observed, and crucial data can be logged to validate the controller's performance before committing to a free flight.

7.2 A Progressive Testing Protocol: From Benchtop to Tethered Tests to Free Flight

A safety-first, incremental testing methodology is essential to mitigate risks and maximize the chances of success.

1. **Benchtop "Wiggle" Test:** With the system fully assembled but without a motor installed, power on the flight computer. Manually tilt the rocket body along its pitch and yaw axes. The GNC system should be active, and one should observe the TVC gimbal moving in the correct direction to counteract the imposed tilt. This simple test verifies the correct orientation of the IMU and the proper sign convention in the control logic.
2. **Tethered Test:** For smaller rockets, a tethered flight can be a valuable intermediate step. The rocket is flown with a live motor but is attached to a guide wire or tether system that prevents it from flying away in the event of a control failure.[24] This allows for testing the controller's performance in free air while providing a crucial safety backup.
3. **First Free Flight:** The first untethered launch is the ultimate test of the system. It should be conducted with a relatively low-power motor in a large, open field, in compliance with all safety codes established by rocketry organizations like the National Association of Rocketry (NAR) or Tripoli Rocketry Association (TRA).[23] Comprehensive onboard data logging is absolutely critical. Every flight, whether successful or not, is a data collection opportunity. Post-flight analysis of the logged sensor and controller data is essential for diagnosing issues and improving the system.[8]

This entire development process is not linear but cyclical. Data from a flight test reveals discrepancies between the simulation and reality. This information is used to update the simulation model to be more accurate. The AI agent is then retrained in the improved simulation, and the new, more robust policy is deployed for the next flight test.[26] This iterative loop of
fly -> analyze -> update -> retrain is the core methodology for developing a high-performance active control system.

7.3 Future Directions: Propulsive Landing, Trajectory Optimization, and Advanced Maneuvers

Once a stable vertical ascent has been reliably achieved, the TVC platform becomes a foundation for exploring far more complex and ambitious control tasks, mirroring the

capabilities of modern commercial spaceflight.

- **Propulsive Landing:** The ultimate challenge in reusable rocketry is propulsive landing. This would involve adding deployable landing legs to the airframe and training a completely new RL agent for the terminal descent phase. The agent would need to learn a "hoverslam" or "suicide burn" maneuver, executing a perfectly timed, high-thrust engine burn to nullify all vertical and horizontal velocity just moments before touching down on a designated landing pad.[8] This requires a more complex state space (including target position) and a sparse reward function that heavily rewards a soft, upright landing.[49]
- **Trajectory Optimization:** The agent can be trained for objectives beyond simple vertical stability. By modifying the reward function, the agent could learn to execute a gravity turn, pitching over after launch to follow an efficient trajectory that maximizes altitude or targets a specific downrange impact point.[10]
- **Aggressive Maneuvering and Disturbance Rejection:** The robustness of the trained policy can be pushed to its limits by training it to follow aggressive attitude commands or to recover from large, externally-induced disturbances.

It is imperative to recognize that as the capabilities of the model rocket advance, so do the associated responsibilities. An actively guided rocket is a technologically sophisticated device. Its development and operation must be conducted with a rigorous focus on safety and in full compliance with all applicable local and national regulations. Certain advanced guidance technologies can be subject to regulations such as the International Traffic in Arms Regulations (ITAR) in the United States, and while hobby-scale projects are not the primary focus of such laws, developers should remain aware of the legal landscape surrounding their work.[7]

Conclusion

The development of a scalable, AI-powered Thrust Vector Control system for model rocketry is an ambitious but achievable multidisciplinary engineering challenge. It requires a synthesis of mechanical design, embedded electronics, control theory, and artificial intelligence. The project's success hinges on a methodical, iterative approach that begins with a solid theoretical foundation and progresses through careful design, simulation-based training, and rigorous, safety-conscious testing.

The transition from classical PID control to a Deep Reinforcement Learning paradigm represents a significant leap in capability. An RL agent can learn to manage the complex, non-linear dynamics of a rocket in flight, automatically discovering a robust control policy that can adapt to the specific characteristics of the vehicle. The key to this process is the creation of a high-fidelity "digital twin" simulation, which serves as the training ground for the AI, and the careful engineering of a reward function that accurately encodes the mission's objectives. For deployment, achieving the "fast decisions" necessary for real-time control is accomplished not through onboard learning, but through highly efficient onboard inference.

By leveraging powerful microcontrollers like the Teensy 4.1 and optimizing the trained neural network through quantization with frameworks like TensorFlow Lite for Microcontrollers, the complex intelligence developed in simulation can be compiled into a compact, lightning-fast function capable of running hundreds of times per second.
Ultimately, this project is not a linear path to a single launch but a cyclical process of design, simulation, and real-world testing. Each flight provides invaluable data that feeds back into the development loop, progressively refining the system's performance. By embracing this iterative methodology and adhering to strict safety protocols, it is possible to create a sophisticated and scalable active control system that pushes the boundaries of model rocketry, opening the door to advanced capabilities such as trajectory optimization and even autonomous propulsive landing.

Works cited

1. Thrust vectoring - Wikipedia, accessed on July 28, 2025, https://en.wikipedia.org/wiki/Thrust_vectoring
2. THRUST VECTOR CONTROL, accessed on July 28, 2025, https://www.kscst.org.in/spp/46_series/46s_spp/02_Exhibition_Projects/208_46S_BE_3913.pdf
3. Developing a Control Algorithm and Simulation for Thrust Vector Controlled Rockets, accessed on July 28, 2025, https://www.nmas.org/wp-content/uploads/2023/03/2022_NMJS_Kim.pdf
4. How does thrust vectoring work? : r/rocketry - Reddit, accessed on July 28, 2025, https://www.reddit.com/r/rocketry/comments/1g5w99v/how_does_thrust_vectoring_work/
5. Making a Thrust Vector Control Mount, Part 1: Learning | by August Fulton Spitz | Medium, accessed on July 28, 2025, https://medium.com/@augustfspitz/making-a-thrust-vector-control-mount-part-1-learning-bd603981440a
6. Thrust Vector Control Mount : 4 Steps - Instructables, accessed on July 28, 2025, https://www.instructables.com/Thrust-Vector-Control-Mount/
7. Thrust Vectoring at Model Scale - Hacker News, accessed on July 28, 2025, https://news.ycombinator.com/item?id=24887754
8. Build Your Own Thrust Vectored Rockets For Vertical Landings Like SpaceX - Make:, accessed on July 28, 2025, https://makezine.com/article/maker-news/build-your-own-thrust-vectored-rockets-for-vertical-landings-like-spacex/
9. University team looking for advice: Movable fins or thrust vector control (TVC) for an active control system in a model rocket? : r/rocketry - Reddit, accessed on July 28, 2025, https://www.reddit.com/r/rocketry/comments/1fxn978/university_team_looking_for_advice_movable_fins/
10. Thrust Vector Controller Comparison for a Finless Rocket - MDPI, accessed on July 28, 2025, https://www.mdpi.com/2075-1702/11/3/394
11. Thrust Vectoring of Small-scale Solid Rocket Motors Using Additively

Manufactured Jet Vanes | AIAA Propulsion and Energy Forum, accessed on July 28, 2025, https://arc.aiaa.org/doi/10.2514/6.2021-3228
12. STUDY OF SELECTED THRUST VECTOR CONTROL SYSTEMS FOR SOLID PROPELLANT MOTORS - NASA Technical Reports Server, accessed on July 28, 2025, https://ntrs.nasa.gov/api/citations/19660001347/downloads/19660001347.pdf
13. Thrust Vector Control for Nuclear Thermal Rockets - NASA Technical Reports Server, accessed on July 28, 2025, https://ntrs.nasa.gov/api/citations/20140002890/downloads/20140002890.pdf
14. Thrust vector control and state estimation architecture for low-cost small-scale launchers - arXiv, accessed on July 28, 2025, https://arxiv.org/pdf/2303.16983
15. How to Determine What Torque You Need for Your Servo Motors - Automatic Addison, accessed on July 28, 2025, https://automaticaddison.com/how-to-determine-what-torque-you-need-for-your-servo-motors/
16. ADVANCED MODELING OF CONTROL-STRUCTURE INTERACTION IN THRUST VECTOR CONTROL SYSTEMS - NASA Technical Reports Server, accessed on July 28, 2025, https://ntrs.nasa.gov/api/citations/20230000427/downloads/AAS_2023_P3_Orr_TVC_Modeling_paper.pdf
17. Figure 4.1 TVC engine for a Saturn-5 rocket, accessed on July 28, 2025, https://www.flixan.net/wp-content/uploads/2015/03/Actuators.pdf
18. jiupinjia/rocket-recycling: Rocket-recycling with ... - GitHub, accessed on July 28, 2025, https://github.com/jiupinjia/rocket-recycling
19. taherfattahi/ppo-rocket-landing: Proximal Policy ... - GitHub, accessed on July 28, 2025, https://github.com/taherfattahi/ppo-rocket-landing
20. Thrust Vector Control 3D Files - BPS.Space, accessed on July 28, 2025, https://bps.space/products/thrust-vector-control
21. HELP with TVC kinematics: I am working on college project on how to steer a rocket and get it back on trajectory to orbit as fast as possible to minimize the energy loss, I am working on the inverse kinematics of a Delta robot like model of two arms 90° apart and any help would be appreciated. : r - Reddit, accessed on July 28, 2025, https://www.reddit.com/r/rocketry/comments/lnkj45/help_with_tvc_kinematics_i_am_working_on_college/
22. Torque calculation - Motors, Mechanics, Power and CNC - Arduino Forum, accessed on July 28, 2025, https://forum.arduino.cc/t/torque-calculation/109896
23. Thrust Vector Control for Rockets - YouTube, accessed on July 28, 2025, https://www.youtube.com/watch?v=cNFJp6psQ6E
24. TVC Model Rocket : 5 Steps - Instructables, accessed on July 28, 2025, https://www.instructables.com/TVC-Model-Rocket/
25. What servos should I use for a TVC mount? : r/rocketry - Reddit, accessed on July 28, 2025, https://www.reddit.com/r/rocketry/comments/g55ux8/what_servos_should_i_use_for_a_tvc_mount/

26. Thrust Vector Controlled Rocket Model, accessed on July 28, 2025, https://dspace.cvut.cz/bitstream/handle/10467/109193/F3-BP-2023-Malek-Lukas-Vektorovani_tahu_modelu_rakety.pdf?sequence=-1&isAllowed=y
27. thrust vectoring for model rocket : r/arduino - Reddit, accessed on July 28, 2025, https://www.reddit.com/r/arduino/comments/14uarra/thrust_vectoring_for_model_rocket/
28. TB-03: Derivation of Thrust Vector Control (TVC) Actuator-Force / Gimbal-Torque Transformation Matrix - NASA Technical Reports Server, accessed on July 28, 2025, https://ntrs.nasa.gov/api/citations/20240012525/downloads/20240012525-rev01.pdf?attachment=true
29. Choosing the Correct Servo - Savox USA, accessed on July 28, 2025, https://www.savoxusa.com/pages/choosing-the-correct-servo
30. Thrust Vectored Model Rocket : r/rocketry - Reddit, accessed on July 28, 2025, https://www.reddit.com/r/rocketry/comments/sdd67r/thrust_vectored_model_rocket/
31. ius thrust vector control (tvc) servo system, accessed on July 28, 2025, https://ntrs.nasa.gov/api/citations/19790014390/downloads/19790014390.pdf
32. Mastering Control with Reinforcement Learning - Number Analytics, accessed on July 28, 2025, https://www.numberanalytics.com/blog/mastering-control-with-reinforcement-learning
33. Reinforcement Learning for Control Systems Applications - MATLAB & Simulink, accessed on July 28, 2025, https://www.mathworks.com/help/reinforcement-learning/ug/reinforcement-learning-for-control-systems-applications.html
34. Technical Comparison Report: ESP32 vs. Teensy 4 Series - Copperhill Technologies, accessed on July 28, 2025, https://copperhilltech.com/blog/technical-comparison-report-esp32-vs-teensy-4-series/
35. “Teensy 4.0 Unleashed: Pushing the Boundaries of Microcontroller Performance and Flexibility” | by Aniket Fasate | Medium, accessed on July 28, 2025, https://medium.com/@fasateaniket5/teensy-4-0-unleashed-pushing-the-boundaries-of-microcontroller-performance-and-flexibility-319d94cf0b40
36. Get to Know ESP32 and ESP8266 Microcontrollers in 2024 - PCBONLINE, accessed on July 28, 2025, https://www.pcbonline.com/blog/esp32-and-esp8266-microcontrollers.html
37. When will Teensy marry ESP32?, accessed on July 28, 2025, https://forum.pjrc.com/index.php?threads/when-will-teensy-marry-esp32.57531/
38. IMU for rocketry - EEVblog, accessed on July 28, 2025, https://www.eevblog.com/forum/beginners/imu-for-rocketry/
39. IMU Comparison - SlimeVR Docs, accessed on July 28, 2025, https://docs.slimevr.dev/diy/imu-comparison.html
40. Can't decide what IMU I should choose for a TVC rockey anyone got a suggestion? : r/rocketry - Reddit, accessed on July 28, 2025,

https://www.reddit.com/r/rocketry/comments/1fgy3rl/cant_decide_what_imu_i_should_choose_for_a_tvc/
41. Capturing IMU Data with a BNO055 Absolute Orientation Sensor - Projects, accessed on July 28, 2025, https://www.allaboutcircuits.com/projects/bosch-absolute-orientation-sensor-bno055/
42. Resources on model Rocket Thrust Vectoring (in particular attitude estimation) - Reddit, accessed on July 28, 2025, https://www.reddit.com/r/rocketry/comments/14laheo/resources_on_model_rocket_thrust_vectoring_in/
43. Utilizing Deep Reinforcement Learning in Control: Release the Power of Intelligent Systems | by Furkan Ayık | Medium, accessed on July 28, 2025, https://medium.com/@ayikfurkan1/utilizing-deep-reinforcement-learning-in-control-unleashing-the-power-of-intelligent-systems-a0e2d690cbd7
44. Reinforcement Learning: An Introduction by Richard S. Sutton | Goodreads, accessed on July 28, 2025, https://www.goodreads.com/book/show/739791.Reinforcement_Learning
45. Reinforcement Learning, second edition: An Introduction|eBook - Barnes & Noble, accessed on July 28, 2025, https://www.barnesandnoble.com/w/reinforcement-learning-second-edition-richard-s-sutton/1137255927
46. Reinforcement Learning: Machine Learning Meets Control Theory - YouTube, accessed on July 28, 2025, https://www.youtube.com/watch?v=0MNVhXEX9to
47. Reinforcement Learning: A Tutorial - Electrical Engineering and Computer Science, accessed on July 28, 2025, https://web.eecs.umich.edu/~baveja/NIPS05RLTutorial/NIPS05RLMainTutorial.pdf
48. Reinforcement Learning - Control Theory - MIT Fab Lab, accessed on July 28, 2025, https://fab.cba.mit.edu/classes/865.21/topics/control/07_reinforcement_learning.html
49. Rocket Landing Control with Random Annealing Jump Start Reinforcement Learning - arXiv, accessed on July 28, 2025, https://arxiv.org/html/2407.15083v1
50. (PDF) Intelligent Attitude Control of Satellites via Deep Reinforcement Learning, accessed on July 28, 2025, https://www.researchgate.net/publication/377396432_Intelligent_Attitude_Control_of_Satellites_via_Deep_Reinforcement_Learning
51. Proximal Policy Optimization (PPO) Agent - MATLAB & Simulink - MathWorks, accessed on July 28, 2025, https://www.mathworks.com/help/reinforcement-learning/ug/proximal-policy-optimization-agents.html
52. alxndrTL/Landing-Starships: Make autonomous landing ... - GitHub, accessed on July 28, 2025, https://github.com/alxndrTL/Landing-Starships
53. Bullet Real-Time Physics Simulation | Home of Bullet and PyBullet: physics simulation for games, visual effects, robotics and reinforcement learning., accessed on July 28, 2025, https://pybullet.org/

54. Documentation | Bullet Real-Time Physics Simulation - PyBullet, accessed on July 28, 2025, https://pybullet.org/wordpress/index.php/forum-2/
55. jnz/PyRocketCraft: Rocket Simulation, Reinforced Learning ... - GitHub, accessed on July 28, 2025, https://github.com/jnz/PyRocketCraft
56. First Simulation with RocketPy, accessed on July 28, 2025, https://docs.rocketpy.org/en/latest/user/first_simulation.html
57. RocketPy-Team/RocketPy: Next generation High-Power Rocketry 6-DOF Trajectory Simulation - GitHub, accessed on July 28, 2025, https://github.com/RocketPy-Team/RocketPy
58. RocketPy Documentation — RocketPy 1.10.0 documentation, accessed on July 28, 2025, https://docs.rocketpy.org/
59. Rocket Simulation by using Machine learning - Kaggle, accessed on July 28, 2025, https://www.kaggle.com/code/rajeevsinghsisodiya/rocket-simulation-by-using-machine-learning
60. Scripting with Python and JPype - OpenRocket wiki, accessed on July 28, 2025, http://wiki.openrocket.info/Scripting_with_Python_and_JPype
61. The Best Tools for Reinforcement Learning in Python You Actually Want to Try - Neptune.ai, accessed on July 28, 2025, https://neptune.ai/blog/the-best-tools-for-reinforcement-learning-in-python
62. Proximal Policy Optimization (PPO) in Reinforcement Learning - GeeksforGeeks, accessed on July 28, 2025, https://www.geeksforgeeks.org/machine-learning/a-brief-introduction-to-proximal-policy-optimization/
63. Mastering Proximal Policy Optimization - Number Analytics, accessed on July 28, 2025, https://www.numberanalytics.com/blog/mastering-proximal-policy-optimization
64. Deep Reinforcement Learning Policies for Underactuated Satellite ..., accessed on July 28, 2025, https://arxiv.org/abs/2505.00165
65. Soft Actor Critic—Deep Reinforcement Learning with Real-World Robots, accessed on July 28, 2025, https://bair.berkeley.edu/blog/2018/12/14/sac/
66. Mastering Soft Actor-Critic in ML - Number Analytics, accessed on July 28, 2025, https://www.numberanalytics.com/blog/ultimate-guide-to-soft-actor-critic
67. Rocket Trajectory Optimization with Reinforcement Learning - Medium, accessed on July 28, 2025, https://medium.com/@arnobgr/rocket-trajectory-optimization-with-reinforcement-learning-6637f5af21bf
68. kaifishr/RocketLander: A simple framework equipped with ... - GitHub, accessed on July 28, 2025, https://github.com/kaifishr/RocketLander
69. OpenRL-Lab/openrl: Unified Reinforcement Learning Framework - GitHub, accessed on July 28, 2025, https://github.com/OpenRL-Lab/openrl
70. Rolv-Arild/rocket-learn - GitHub, accessed on July 28, 2025, https://github.com/Rolv-Arild/rocket-learn
71. Optimizing Neural Networks: Unveiling the Power of Quantization - Analytics Vidhya, accessed on July 28, 2025, https://www.analyticsvidhya.com/blog/2024/01/optimizing-neural-networks-unvei

ling-the-power-of-quantization-techniques/
72. Advances in the Neural Network Quantization: A Comprehensive Review - MDPI, accessed on July 28, 2025, https://www.mdpi.com/2076-3417/14/17/7445
73. Quantization For Embedded Systems - Meegle, accessed on July 28, 2025, https://www.meegle.com/en_us/topics/quantization/quantization-for-embedded-systems
74. Deep Learning Network Quantization for Deployment to Embedded Targets - MATLAB, accessed on July 28, 2025, https://www.mathworks.com/videos/deep-learning-network-quantization-for-deployment-to-embedded-targets-1620162112212.html
75. Neural Network Model quantization on Mobile - AI blog - Arm Community, accessed on July 28, 2025, https://community.arm.com/arm-community-blogs/b/ai-blog/posts/neural-network-model-quantization-on-mobile
76. TensorFlow Lite Tutorial — SensiML Documentation, accessed on July 28, 2025, https://sensiml.com/documentation/sensiml-python-sdk/additional-tutorials/tensorflow-lite-tutorial.html
77. Get started with microcontrollers | Google AI Edge - Gemini API, accessed on July 28, 2025, https://ai.google.dev/edge/litert/microcontrollers/get_started
78. TinyML: Getting Started with TensorFlow Lite for Microcontrollers - DigiKey, accessed on July 28, 2025, https://www.digikey.com/en/maker/projects/tinyml-getting-started-with-tensorflow-lite-for-microcontrollers/c0cdd850f5004b098d263400aa294023
79. Multiagent Reinforcement Learning-Based Multimodel Running Latency Optimization in Vehicular Edge Computing Paradigm | Request PDF - ResearchGate, accessed on July 28, 2025, https://www.researchgate.net/publication/383771007_Multiagent_Reinforcement_Learning-Based_Multimodel_Running_Latency_Optimization_in_Vehicular_Edge_Computing_Paradigm
80. Low-Precision Reinforcement Learning: Running Soft Actor-Critic in Half Precision - Computer Science Cornell, accessed on July 28, 2025, https://www.cs.cornell.edu/gomes/pdf/2021_bjorck_icml_sac.pdf
81. accessed on January 1, 1970, https://arxiv.org/pdf/2407.15083v1.pdf