Optimizing Vision-Language-Action Models for Custom Mobile Manipulators: An Exhaustive Analysis of Telemetry, Perception, and Framework Architecture

Executive Summary

The rapid evolution of embodied artificial intelligence has precipitated a paradigm shift in robotic control, moving from modular, hand-engineered pipelines to end-to-end learning systems rooted in Vision-Language-Action (VLA) models. This report addresses the specific engineering and theoretical challenges associated with deploying such models on custom, cost-constrained hardware platforms—specifically, a mobile manipulator composed of a DJI RoboMaster base and a custom robotic arm. The core inquiry examines the feasibility of training imitation learning policies using "commanded actions" as ground truth in the presence of unreliable Z-axis rotation telemetry, the necessity of auxiliary static visual streams for robust state estimation, and the comparative efficacy of the LeRobot framework against alternative architectures like OpenVLA and ALOHA.

The analysis concludes that for holonomic mobile bases with significant mechanical compliance, such as the RoboMaster S1/EP, training on commanded actions is not merely a workaround but a theoretically superior strategy to relying on noisy inertial or odometric telemetry. By leveraging the implicit inverse dynamics capabilities of modern neural networks, the system can learn to map visual intent directly to control signals, effectively bypassing the sensor noise floor. Furthermore, the integration of a secondary, static, allocentric camera view is determined to be non-negotiable for mobile manipulation tasks. This view resolves the "causal confusion" inherent in single-view behavioral cloning by providing a stable global reference frame that anchors the robot's ego-motion and resolves occlusions caused by the manipulator itself. Finally, a rigorous architectural review identifies the LeRobot ecosystem as the optimal development environment. Its modular hardware abstraction layer, standardized LeRobotDataset format, and support for efficient VLA architectures like SmolVLA provide the necessary infrastructure to bridge the gap between custom Python SDKs and state-of-the-art foundation models, outperforming the rigid, research-centric pipelines of OpenVLA or ALOHA for this specific application.

1. The Convergence of Mobile Robotics and Vision-Language-Action Models

The intersection of Large Language Models (LLMs) and robotic control has given rise to a new class of foundation models known as Vision-Language-Action (VLA) models. These systems promise to endow robots with the ability to reason about the world semantically and execute complex, long-horizon tasks through natural language instruction. However, the translation of these capabilities from high-end research laboratories—equipped with industrial-grade hardware and data center compute—to accessible, custom robotic platforms presents a multifaceted engineering challenge.

1.1 The Evolution from Modular Control to End-to-End Learning

Traditionally, mobile manipulation has been treated as a decoupled problem. A classical stack consists of distinct modules: Simultaneous Localization and Mapping (SLAM) for determining position; a global planner for navigation; a local planner for obstacle avoidance; and a manipulation planner for arm trajectory generation. Each module relies on explicit state estimation. The robot must know exactly where it is (x, y, θ), the exact configuration of its joints ($q_1, \dots, q_n$), and the precise location of the target object in a global coordinate frame.[1]

While robust in structured environments, this modular approach is brittle in the face of sensor noise and environmental variability. If the localization module drifts due to wheel slip—a common issue with the mecanum wheels found on the RoboMaster platform—the entire downstream chain fails. The planner generates a trajectory based on a false premise, and the robot misses its target.

VLA models, such as RT-2, OpenVLA, and the recent architectures supported by LeRobot, propose a radical alternative: end-to-end learning.[3] In this paradigm, a single neural network learns a direct mapping from raw sensory observations (pixels from cameras) and high-level instructions (text) to low-level motor commands. The explicit state estimation step is bypassed. The model does not explicitly calculate "I am at coordinate (2.5, 3.4)"; rather, it learns "When the visual scene looks like *this* and the instruction is 'grab the cup', the correct motor command is *that*".

1.2 The Specificity of the RoboMaster Platform

The DJI RoboMaster S1 and EP robots occupy a unique niche in this landscape. Designed primarily for education and competition, they possess high-performance brushless motors and a sophisticated chassis capable of omnidirectional movement. However, they lack the high-fidelity proprioception found in research robots like the Unitree Go2 or the Clearpath Husky.

The RoboMaster's odometry is derived from wheel encoders and a consumer-grade Inertial Measurement Unit (IMU). While sufficient for short-duration movements on rigid surfaces, this telemetry suffers from significant drift over time and is highly susceptible to magnetic interference (affecting the compass/yaw) and wheel slippage (affecting position).[5] The user's specific setup, which involves driving the robot via a custom Python SDK and recording velocity commands (V_x, V_y, ω_z), places the system squarely in the domain of "low-cost mobile manipulation." This constraint forces a re-evaluation of standard imitation learning practices, particularly regarding what constitutes "ground truth" data.[6]

1.3 The Vision-Language-Action Paradigm Shift

The emergence of VLA models represents a move towards "generalist" policies. Unlike specialist policies trained via Reinforcement Learning (RL) for a single task (e.g., walking), VLAs leverage the vast semantic knowledge embedded in pre-trained Vision-Language Models (VLMs) like LLaVA or SigLIP.[3] This allows the robot not just to "pick up the object," but to "pick up the *red* object" or "move the *spilled* cup," relying on the model's understanding of "red" and "spilled" derived from internet-scale data.
For a custom robot user, this offers a tantalizing possibility: creating a smart robot without writing thousands of lines of logic for object detection, state machines, and error recovery. However, it also imposes strict requirements on data quality. VLA models are notoriously data-hungry and sensitive to the distribution of the training data. If the relationship between the visual input and the action label is noisy or inconsistent—as it might be with unreliable telemetry—the model will fail to converge or will exhibit hallucinated behaviors.[8]

---

2. Telemetry, State Estimation, and the Action Space

The user's primary technical uncertainty lies in the choice of the training target: *Should the model learn to predict the commands sent to the robot (Commanded Actions) or the motion measured by the robot's sensors (Telemetry)?* This question strikes at the heart of Imitation Learning theory and requires a deep analysis of control systems and causal modeling.

2.1 The Theoretical Basis of Behavioral Cloning Targets

In Behavioral Cloning (BC), we aim to approximate the expert's policy $\pi^*(s)$. The dataset consists of trajectories $\tau = \{(o_0, a_0), (o_1, a_1), \dots, (o_T, a_T)\}$. The neural network with parameters $\theta$ is trained to minimize a loss function, typically the Mean Squared Error (MSE) or Negative Log-Likelihood (NLL) between its prediction and the recorded action $a_t$.[10]
$$ \mathcal{L}(\theta) = \mathbb{E}_{(o, a) \sim \mathcal{D}} [|
| \pi_\theta(o) - a ||^2] $$
The definition of $a$ is the critical variable.

1. **Measured State ($a = \Delta s_{measured}$):** This approach assumes that the "truth" is what the robot physically did. If the expert commanded a velocity of 1.0 m/s but the robot only moved 0.8 m/s due to friction, the label is 0.8.
2. **Commanded Action ($a = u_{cmd}$):** This approach assumes the "truth" is the expert's intent. If the expert commanded 1.0 m/s, the label is 1.0, regardless of the physical outcome.

2.2 The Physics of Mecanum Wheels and Slip Dynamics

To understand why telemetry is problematic for the RoboMaster, we must look at the kinematics of mecanum wheels. A mecanum wheel has rollers mounted at $45^\circ$ angles to the wheel hub. The velocity of the robot chassis $\mathbf{v} = [v_x, v_y, \omega_z]^T$

relates to the wheel velocities $\mathbf{\omega}_{wheels}$ via the kinematic Jacobian J.[11]

$$\mathbf{\omega}_{wheels} = J \cdot \mathbf{v}$$

However, this relationship holds only under the assumption of **no slip**. In reality, the rollers slip continuously. The actual motion of the robot is:

$$\mathbf{v}_{actual} = J^+ \mathbf{\omega}_{wheels} + \mathbf{\epsilon}_{slip}(\mu, N, \mathbf{a})$$

Where $\mathbf{\epsilon}_{slip}$ is a complex error term dependent on the coefficient of friction μ, the normal force N (which changes as the arm moves and shifts the center of mass), and acceleration $\mathbf{a}$.

The robot's onboard odometry estimates $\mathbf{v}_{actual}$ by integrating wheel encoder ticks. Since it cannot measure the slip $\mathbf{\epsilon}_{slip}$ directly (without external sensors like optical flow or high-end GPS), the odometry assumes zero slip. Therefore, the "measured telemetry" is **systematically biased**. It reports where the robot *would have gone* if the floor were perfectly rigid and sticky, not where it actually went.

2.3 The "Z-Rotation" (Yaw) Telemetry Failure Mode

The user specifically identifies Z-rotation telemetry as unreliable. This is consistent with the hardware limitations. While linear velocity errors accumulate linearly, angular velocity errors can accumulate catastrophically, leading to heading drift. On a platform like the RoboMaster, Z-rotation is often fused with a magnetometer. Indoors, ferrous metals in building structures cause hard-iron and soft-iron distortions, rendering the magnetometer unreliable.

If we were to use this noisy telemetry as the training target (a), we would introduce two fatal flaws into the VLA:

1. **High Entropy / Noise Injection:** The model would attempt to learn a mapping from visual inputs to a noisy signal. A consistent visual rotation might be labeled as $10^\circ/s$ in one sample and $2^\circ/s$ in another (due to sensor lag or drift). This forces the model to "average" inconsistent data, leading to "indecisive" or washed-out behaviors.[8]
2. **The Inverse Dynamics Mismatch:** If the model predicts the "measured" velocity, we need an inverse controller to convert that back into a command to send to the motors. Since the low-level controller on the RoboMaster expects a "desired velocity" command, feeding it a "measured velocity" (which includes slip effects) closes the loop on the wrong variable.[12]

2.4 The Superiority of Commanded Actions: Implicit Inverse Dynamics

Training on **Commanded Actions** (u_{cmd}) solves both problems by shifting the burden of handling physics to the neural network.

When the VLA is trained on u_{cmd}, it learns the function:

$$\pi_\theta(Image_t) \rightarrow u_{cmd}$$

Crucially, this function implicitly incorporates the **Inverse Dynamics** of the system.

$$u_{cmd} \approx f^{-1}_{physics}(\text{Desired Motion inferred from Vision})$$
Consider a scenario where the robot is on a thick carpet. To turn right at a visible rate of $30^\circ/s$, the human expert pushes the joystick all the way to the right (Command = 1.0). On a slick tile floor, to achieve the same visual rotation, the expert might only push the joystick halfway (Command = 0.5).

- **Case A (Telemetry Training):** The model learns to predict "30 deg/s" in both cases. But at inference time, if it outputs "30 deg/s" to the robot's controller, the low-level PID might not push hard enough on the carpet.
- **Case B (Command Training):** The model sees the texture of the carpet (via the camera) and learns "When I see carpet and need to turn, output 1.0. When I see tile, output 0.5."

The neural network, with its high capacity, learns to compensate for the environment's friction and the robot's mechanical compliance. It learns the *intent* required to overcome the physics.[13]

Conclusion on Telemetry: For the specific case of the RoboMaster S1/EP with unreliable Z-telemetry, training on **Commanded Actions** is the only viable path. It bypasses the sensor noise and leverages the VLA's capacity to learn implicit inverse dynamics. The user should record the exact values sent to the chassis.drive_speed() function in the Python SDK and use those as the action labels.[5]

2.5 Designing the Input State Vector

While the *output* labels must be commanded actions, the *input* state vector (observation.state) requires careful curation. VLA models typically fuse visual embeddings with proprioceptive data.

- **Do NOT include Z-rotation telemetry:** Feeding the unreliable sensor data back into the network as an input feature is dangerous. It creates a "conflicting supervisor" where the visual data says one thing (e.g., "we are rotating") and the telemetry says another ("we are still"). Neural networks often overfit to the easiest feature; if the telemetry is easier to parse than pixels, the network might rely on the garbage telemetry and fail in the real world.[15]
- **Do include Arm State:** The encoders on the robotic arm joints are typically rigid and reliable. Include joint positions and gripper state.
- **Do rely on Visual Odometry:** Modern Vision Transformers (ViTs) used in VLAs (like SigLIP in SmolVLA) are highly effective at extracting optical flow and ego-motion from image sequences. The model will internally estimate the base's motion from the changing camera view, effectively learning a "Visual Gyroscope" that is far more robust than the hardware sensor.[16]

---

3. Visual Perception Architectures in Robotics

The second critical design decision concerns the vision system. The user asks if a second static camera is necessary. In the context of mobile manipulation, where the base moves and the arm interacts, the visual architecture is the primary determinant of policy robustness.

3.1 The Limits of Ego-Centric (Onboard) Perception

An onboard camera (the FPV camera on the RoboMaster) provides an "ego-centric" view. While intuitive for teleoperation, it suffers from several severe limitations for autonomous policy learning:

1. **The "Copycat" Problem and Causal Confusion:** When a robot moves forward, the visual scene expands (optical flow). If the robot is trained only on this view, it struggles to distinguish between "I am moving forward" and "The world is zooming in." More critically, if the robot stops moving, the visual signal of motion disappears. This can lead to a state where the robot stops, sees no motion, and thus predicts "stop" actions, freezing in place. This is a well-documented failure mode in Behavioral Cloning known as "causal confusion".[9]
2. **Occlusion by the Manipulator:** In a mobile manipulator setup, the arm is mounted on the chassis. As the arm reaches out to grasp an object, it inevitably moves into the field of view of the onboard camera. Often, at the critical moment of grasping, the gripper completely occludes the target object. A single-view policy becomes blind at the most crucial step of the task, leading to low success rates.[17]
3. **Field of View (FoV) Tunnel Vision:**
   The RoboMaster's camera has a fixed forward gaze. It cannot see obstacles to the side or behind it. Since the platform is holonomic (can move sideways), a policy might command a lateral move (V_y) to align with a target, unaware that it is crashing into a table leg just out of frame.

3.2 The Strategic Necessity of Allocentric (Static) Vision

Adding a second, static camera (allocentric view) resolves these issues fundamentally.

1. **Global Reference Frame:** A static camera acts as a visual anchor. It sees the robot *and* the target in the same coordinate frame. The VLA can learn a geometric relationship: "Minimize the pixel distance between the robot's gripper and the red cup." This is a much simpler function to learn than inferring relative positions from a moving, pitching camera.[2]
2. **Occlusion Robustness:** When the arm blocks the onboard camera, the static camera typically retains a clear line of sight to the workspace. Attention mechanisms in Transformer-based VLAs (like the Cross-Attention layers in ACT or OpenVLA) automatically learn to attend to the camera view that provides the most information at any given timestep. Research on the ALOHA system demonstrates that multi-camera fusion increases success rates from ~40% (single view) to >90% (multi-view) on complex tasks.[18]
3. **Drift Correction:** Given the unreliable Z-rotation telemetry, the static camera provides the "ground truth" for orientation. If the robot drifts 10° yaw due to wheel slip,

the onboard camera just sees the scene shift. The static camera sees the robot *rotated* relative to the environment. This visual feedback allows the policy to correct the mechanical drift of the mecanum wheels, acting as an external localization system.[20]

3.3 Camera Placement: Angled vs. Top-Down

The user asks specifically about "angled/top-down."

- **Strict Top-Down (Orthographic):** This view is excellent for 2D navigation (x, y plane). It turns the navigation problem into a 2D map traversal. However, it compresses the Z-axis (height), making it difficult for the robot to judge the height of objects for grasping or stacking.
- **Angled Allocentric (e.g., 45-degree shoulder view):** This is the optimal configuration for mobile manipulation. It captures the robot's position on the floor (for navigation) while preserving 3D perspective cues (for manipulation). It allows the model to perceive depth through perspective distortion and parallax.[17]

Recommendation: A second static camera is **mandatory** for a robust system. An angled view ($30^\circ-45^\circ$ elevation) covering the workspace is superior to a strict top-down view.

3.4 The Challenge of Wifi Camera Synchronization

The user intends to use a "static wifi camera." This introduces a significant technical hazard: **Network Latency and Jitter.**

Behavioral Cloning relies on the assumption of **temporal alignment**: $Action_t$ was caused by $Observation_t$. If the Wifi camera stream lags by 200ms, the policy will associate the *current* action with a *past* visual state.

- **Variable Latency (Jitter):** Wifi latency is stochastic. It fluctuates based on network traffic. This means the lag is not constant, preventing simple calibration.
- **The "Stop-Go" Oscillation:** If the camera lags, the robot might overshoot a target because the video feed shows it hasn't arrived yet. Then, when the video catches up, it sees it has gone too far and corrects. This leads to oscillation.

Engineering Solution:

1. **Timestamping:** The data collection pipeline must timestamp every frame upon arrival.
2. **Software Synchronization (The "Delta" Method):** When training the VLA, the data loader should not just grab the "next" frame. It should look for the frame with the timestamp closest to the action's timestamp. LeRobotDataset v3.0 supports this via the delta_timestamps configuration, allowing the user to request frames relative to the current time (e.g., $t=0, t=-0.1$).[19]
3. **Hold-Last Strategy:** If a frame is dropped or delayed, the training pipeline should repeat the last known good frame rather than halting or using a frame from the future.
4. **Hardware Preference:** If at all possible, replacing the Wifi camera with a USB camera connected to the main compute unit (even via a long active extension cable) will eliminate this source of error and significantly improve training stability.[22]

4. The LeRobot Ecosystem and Framework Analysis

The choice of software framework is as critical as the hardware. The user asks if LeRobot is the best choice or if alternatives exist. This requires a comparative analysis of the current robot learning landscape.

4.1 The Framework Landscape

Feature	**LeRobot**	**OpenVLA (Native)**	**ACT / ALOHA**	**Isaac Lab / Orbit**
Primary Focus	Democratization, Hardware Abstraction, Standard Datasets	VLA Fine-tuning, Research	Bimanual Manipulation	Sim-to-Real, RL
Data Format	**LeRobotDataset** (Parquet + MP4)	**RLDS** (TensorFlow Records)	**HDF5** (Custom Structure)	USD / HDF5
Model Support	ACT, Diffusion, **SmolVLA**, TDMPC	OpenVLA-7B	ACT (CNN/ResNet)	PPO, SAC, Dreamer
Compute Req.	Scalable (Consumer GPU)	High (A100/H100 recommended)	Low (RTX 3060 sufficient)	High (Physics Sim)
Custom Hardware	**High** (Modular Robot class)	Low (Assumes Bridge data)	Low (Rigid ALOHA config)	N/A (Simulation focused)

4.2 The Case for LeRobot

LeRobot stands out as the optimal framework for this specific project for three distinct reasons:

4.2.1 Hardware Abstraction and the Python SDK Wrapper

LeRobot provides a Python-native Robot class designed to be subclassed. The user is already using a "custom Python SDK." Integrating this into LeRobot is a matter of writing a wrapper class:

Python

```
class RoboMasterRobot(lerobot.Robot):
    def send_action(self, action):
        # Map normalized action to SDK commands
        vx, vy, omega = action[:3]
        self.sdk.chassis.drive_speed(x=vx, y=vy, z=omega)
```

This wrapper allows the RoboMaster to inherit all of LeRobot's tooling: data recording scripts, visualization dashboards, and safety limits. In contrast, using the original ACT repository or OpenVLA codebase would require the user to write their own data collection pipeline from scratch, handle thread locking for cameras, and manually format data into HDF5 or RLDS files—a process prone to subtle bugs.[24]

4.2.2 The LeRobotDataset Standard (v3.0)

The user has a complex data signature: "Vx, Vy, Z commands, arm moves, onboard video, static wifi video."
LeRobotDataset v3.0 is designed for exactly this heterogeneity. It uses **Apache Parquet** for the tabular data (actions, states) and **MP4** for the video streams.

- **Automatic Sync:** It handles the synchronization of the multiple video streams (onboard + wifi) and the action logs automatically via timestamp alignment.
- **Flexibility:** It allows defining arbitrary action spaces. The user can define an action vector of size 7: [vx, vy, omega, arm_1, arm_2, arm_3, gripper]. Other frameworks like ALOHA often hardcode the action space for two arms (14 DoF) and require deep code refactoring to change.[24]

4.2.3 Native Support for SmolVLA

The user explicitly mentions wanting to train a **VLA**.

- **OpenVLA-7B:** The standard OpenVLA model is 7 billion parameters. Fine-tuning this requires massive VRAM (typically 80GB A100s) and complex LoRA (Low-Rank Adaptation) setups. Inference on a mobile robot's onboard computer (likely a laptop or Jetson) would be extremely slow (<1Hz).
- **SmolVLA:** LeRobot supports **SmolVLA**, a highly efficient VLA architecture (typically ~1B parameters or less) built on the **SigLIP** vision encoder and **SmolLM** language backbone. This model is designed to run on consumer-grade hardware (like an RTX 3090 or 4090) while maintaining the semantic grounding capabilities of larger models. For a custom project, this balance of capability and efficiency is crucial.[29]

4.3 Why Not Alternatives?

- **OpenVLA Repo:** While powerful, it is a research codebase tailored for large-scale pre-training on the Open X-Embodiment dataset. It lacks the tooling for collecting data on a custom robot and quickly training a policy. The "data friction" of converting custom logs to the RLDS format is significant.[25]
- **ALOHA/ACT:** The original ACT repo is excellent for bimanual manipulation but is rigid. Adapting it to a mobile base with a different action space involves significant hacking of the imitate_episodes.py script. LeRobot implements the ACT policy within a modular structure, giving the user the best of both worlds.[19]

Verdict: LeRobot is the superior choice. It provides the "glue" code to connect the custom Python SDK to state-of-the-art policies (SmolVLA, Diffusion) without requiring the user to reinvent the data engineering wheel.

---

5. Systems Integration and Engineering Strategy

Having established the theoretical approach (Commanded Actions, Multi-View) and the software foundation (LeRobot), this section provides a concrete engineering roadmap for implementation.

5.1 Designing the Action Space and Normalization

The neural network expects normalized inputs and outputs, typically in the range $[-1, 1]$. The RoboMaster SDK expects physical units (m/s).

- **Normalization Strategy:**
  Define the physical limits of the robot:
  - $V_{max} \approx 3.5$ m/s (RoboMaster S1 top speed)
  - $\Omega_{max} \approx 600$ deg/s
  - Arm Joint Limits (in degrees or radians)
    The LeRobot Robot class configuration allows setting these min and max values. The framework will automatically normalize the data recorded from the joystick during collection and denormalize the network's predictions during inference.
  - *Critical Detail:* Ensure the joystick used for teleoperation covers the full dynamic range. If the human only ever pushes the stick to 50%, the normalized action will never exceed 0.5, and the robot will move slowly. Calibrate the joystick to map full deflection to the desired maximum training speed.[6]

5.2 Data Collection Protocol

To train a robust VLA, the data must be diverse.

1. **Teleoperation:** Use a high-quality gamepad (Xbox/PS5) mapped to the Python SDK. Drive the robot smoothly. Jerky human motions will be cloned by the policy.
2. **Episode Duration:** Keep episodes focused (15-60 seconds). Long episodes increase the accumulation of drift error in the LSTM/Transformer horizons.
3. **Task Variations:**
   - *Visual Diversity:* Collect data with different lighting conditions and background clutter.
   - *Spatial Diversity:* Start the robot from different positions and angles relative to the target. This forces the model to learn correction behaviors, not just a single open-loop trajectory.
4. **Handling "Stop":** The dataset must explicitly contain "stop" actions (all zeros) at the end of a task. Without this, the robot might drift indefinitely after completing the task. LeRobot handles this with next.done flags, but the recorder must explicitly capture a few seconds of station-keeping at the end of each episode.[31]

5.3 Training Configuration: The "VLA" Pipeline

While the user's goal is a VLA, starting directly with a language-conditioned model can be hard to debug. A staged approach is recommended:

Phase 1: Visual Behavioral Cloning (Diffusion Policy)

- Train a **Diffusion Policy** (available in LeRobot) using only the image and state inputs, ignoring the language instructions.
- **Why:** Diffusion policies are extremely robust to multimodal action distributions (e.g., "go left OR right" to avoid an obstacle). They handle the continuous action space of the mobile base exceptionally well.[32]
- **Goal:** Verify that the "Commanded Action" + "Multi-View" hypothesis works. If this policy cannot navigate to the object, the VLA won't either.

Phase 2: Language Conditioning (SmolVLA)

- Once the base policy works, switch to the **SmolVLA** architecture.
- **Data Annotation:** Annotate the collected episodes with text instructions (e.g., "Drive to the red cube," "Push the blue box"). LeRobot's dataset format supports a tasks.jsonl file for this purpose.[33]
- **Training:** Fine-tune the SmolVLA model. The pre-trained SigLIP visual encoder will help the model generalize to new objects (e.g., recognizing a "pear" even if it only saw "apples" during training, provided the pre-training set was large enough).[29]

5.4 Implementation of the "State" Exclusion

In the LeRobot configuration file (e.g., robomaster_smolvla.yaml), the feature mapping should be explicit:

YAML

```
features:
  observation.images.onboard:
    dtype: video
    shape:
  observation.images.static:
    dtype: video
    shape:
  observation.state:
    dtype: float32
    shape:   # ONLY arm joints + gripper
  action:
    dtype: float32
    shape:   # Vx, Vy, W, Arm1, Arm2, Arm3, Grip
```

Note the exclusion of chassis odometry from observation.state. This forces the model to rely on observation.images for navigation, effectively learning visual odometry.[19]

---

6. Theoretical Implications and Future Directions

The architecture proposed here—training on intent rather than measurement, utilizing multi-view fusion, and leveraging efficient VLA foundations—aligns with the broader trend in robotics towards "Software 2.0."

6.1 Implicit Physics Learning

By training on commanded actions, we are effectively asking the neural network to internalize the physics of the RoboMaster chassis. The network learns a combined model of $\pi(o) = \text{Controller}^{-1}(\text{Planner}(o))$. This collapse of the planning and control layers into a single weight matrix is what allows VLAs to execute dynamic maneuvers (like drifting or pushing) that are difficult to model analytically.[14]

6.2 The Commoditization of Robot Learning

The ability to run this stack on a RoboMaster (a <$1000 robot) with LeRobot (open-source software) signifies a democratization of technology previously reserved for Google DeepMind or OpenAI. The use of **SmolVLA** specifically points to a future where "edge VLAs" run directly on the robot, removing the need for cloud connectivity and its associated latency risks.[34]

6.3 Scaling Laws

As the user collects more data, the performance of the VLA should scale. Unlike classical KF/SLAM approaches, which hit a ceiling defined by sensor noise, VLA performance is bounded primarily by data diversity. This suggests that the most effective way to improve the system is not to buy better sensors, but to collect more diverse teleoperation data—a fundamental shift in robotic engineering philosophy.[33]

---

7. Conclusion

The deployment of a Vision-Language-Action model on a custom RoboMaster platform is a feasible and scientifically sound endeavor, provided specific architectural choices are made to mitigate hardware limitations.

1. **Telemetry Strategy:** The unreliability of Z-rotation telemetry necessitates a training strategy based on **Commanded Actions**. This approach leverages the neural network's capacity to learn implicit inverse dynamics, effectively treating the robot's physics and low-level controller as a single learnable block.
2. **Visual Architecture:** A **second, static, allocentric camera** is mandatory. It resolves the ambiguity of ego-motion and provides the global spatial context required for robust navigation and manipulation, preventing the "causal confusion" that plagues single-view systems.
3. **Framework: LeRobot** is the superior framework for this application. Its hardware abstraction layer facilitates the integration of the custom Python SDK, while its

standardized dataset format and support for efficient models like SmolVLA provide the most direct path to a functional, language-conditioned policy.

By adhering to this roadmap—prioritizing intent over noisy measurement, ensuring multi-view perception, and utilizing the modular power of LeRobot—the user can construct a sophisticated mobile manipulation system that transcends the limitations of its consumer-grade hardware.

Citations: [1]

Works cited

1. Pure Vision Language Action (VLA) Models: A Comprehensive Survey - arXiv, accessed February 2, 2026, https://arxiv.org/html/2509.19012v1
2. AI Robotics: A Field Report on Imitation Learning with LeRobot - ML6, accessed February 2, 2026, https://www.ml6.eu/en/blog/ai-robotics-a-field-report-on-imitation-learning-with-lerobot
3. VLA-0: Building State-of-the-Art VLAs with Zero Modification - arXiv, accessed February 2, 2026, https://arxiv.org/html/2510.13054v1
4. From Words to Actions: The Rise of Vision-Language-Action Models in Robotics - Marvik.ai, accessed February 2, 2026, https://www.marvik.ai/blog/from-words-to-actions-the-rise-of-vision-language-action-models-in-robotics
5. RoboMaster S1 - Programming Guide - DJI, accessed February 2, 2026, https://www.dji.com/robomaster-s1/programming-guide
6. 5. Getting Started with the RoboMaster SDK - EP, accessed February 2, 2026, https://robomaster-dev.readthedocs.io/en/latest/python_sdk/beginner_ep.html
7. SmolVLA: Efficient Vision Language Action Model - LeRobot - Learn OpenCV, accessed February 2, 2026, https://learnopencv.com/smolvla-lerobot-vision-language-action-model/
8. Decisiveness in Imitation Learning for Robots - Google Research, accessed February 2, 2026, https://research.google/blog/decisiveness-in-imitation-learning-for-robots/
9. Constrained Behavior Cloning for Robotic Learning - arXiv, accessed February 2, 2026, https://arxiv.org/html/2408.10568v1
10. Ch. 21 - Imitation Learning - Underactuated Robotics, accessed February 2, 2026, https://underactuated.mit.edu/imitation.html
11. Sim2Real Transfer of Imitation Learning of Motion Control for Car-like Mobile Robots Using Digital Twin Testbed - MDPI, accessed February 2, 2026, https://www.mdpi.com/2218-6581/14/12/180
12. Deformable Object Manipulation with a Tactile Reactive Gripper Neha Sunil - DSpace@MIT, accessed February 2, 2026, http://dspace.mit.edu/bitstream/handle/1721.1/139946/Sunil-nsunil-smme-meche-2021-thesis.pdf?sequence=1&isAllowed=y
13. Robot Motion Planning in Dynamic, Uncertain Environments - ResearchGate,

accessed February 2, 2026, https://www.researchgate.net/publication/220396955_Robot_Motion_Planning_in_Dynamic_Uncertain_Environments
14. Combining Trajectory Optimization, Supervised Machine Learning, and Model Structure for Mitigating the Curse of Dimensionality in the Control of Bipedal Robots | Request PDF - ResearchGate, accessed February 2, 2026, https://www.researchgate.net/publication/320920179_Combining_Trajectory_Optimization_Supervised_Machine_Learning_and_Model_Structure_for_Mitigating_the_Curse_of_Dimensionality_in_the_Control_of_Bipedal_Robots
15. Top 4 Techniques for Handling Missing Values in Machine Learning - Paperspace Blog, accessed February 2, 2026, https://blog.paperspace.com/top-4-techniques-for-handling-the-missing-values-in-machine-learning/
16. [2408.10568] Constrained Behavior Cloning for Robotic Learning - arXiv, accessed February 2, 2026, https://arxiv.org/abs/2408.10568
17. VBM-Net: Visual Base Pose Learning for Mobile Manipulation using Equivariant TransporterNet and GNNs - arXiv, accessed February 2, 2026, https://arxiv.org/html/2510.04171v1
18. Exploring Active Vision in Bimanual Robotic Manipulation - arXiv, accessed February 2, 2026, https://arxiv.org/html/2409.17435v1
19. Robot Learning: A Tutorial - a Hugging Face Space by lerobot, accessed February 2, 2026, https://huggingface.co/spaces/lerobot/robot-learning-tutorial
20. RoboManipBaselines: A Unified Framework for Imitation Learning in Robotic Manipulation across Real and Simulated Environments - arXiv, accessed February 2, 2026, https://arxiv.org/html/2509.17057v1
21. What are the correct angles in the Top-down perspective? - Game Development Stack Exchange, accessed February 2, 2026, https://gamedev.stackexchange.com/questions/70368/what-are-the-correct-angles-in-the-top-down-perspective
22. [1812.09366] Wireless Software Synchronization of Multiple Distributed Cameras - ar5iv, accessed February 2, 2026, https://ar5iv.labs.arxiv.org/html/1812.09366
23. Lekiwi in Lerobot | Seeed Studio Wiki, accessed February 2, 2026, https://wiki.seeedstudio.com/lerobot_lekiwi/
24. LeRobot: Making AI for Robotics more accessible with end-to-end learning - GitHub, accessed February 2, 2026, https://github.com/huggingface/lerobot
25. openvla/README.md at main - GitHub, accessed February 2, 2026, https://github.com/openvla/openvla/blob/main/README.md
26. Isaac Lab: A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning - arXiv, accessed February 2, 2026, https://arxiv.org/html/2511.04831v1
27. Bring Your Own Hardware - Hugging Face, accessed February 2, 2026, https://huggingface.co/docs/lerobot/integrate_hardware
28. LeRobotDataset v3.0 - Hugging Face, accessed February 2, 2026, https://huggingface.co/docs/lerobot/en/lerobot-dataset-v3
29. VLA-0-Smol - Robot Learning Collective, accessed February 2, 2026, https://robot-learning-collective.github.io/vla-0-smol

30. SmolVLA: Efficient Vision-Language-Action Model trained on Lerobot Community Data, accessed February 2, 2026, https://huggingface.co/blog/smolvla
31. LeRobot Dataset Format - phospho starter pack documentation, accessed February 2, 2026, https://docs.phospho.ai/learn/lerobot-dataset
32. Visuomotor Policy Learning via Action Diffusion - arXiv, accessed February 2, 2026, https://arxiv.org/html/2303.04137v5
33. LeRobot Community Datasets: The "ImageNet" of Robotics — When and How?, accessed February 2, 2026, https://huggingface.co/blog/lerobot-datasets
34. Fine Tuning SmolVLA for New Environments (Code included!) | by Xavier O'Keefe - Medium, accessed February 2, 2026, https://medium.com/correll-lab/fine-tuning-smolvla-for-new-environments-code-included-af266c56d632
35. Multimodal Mobile Robotic Dataset for a Typical Mediterranean Greenhouse: The GREENBOT Dataset - MDPI, accessed February 2, 2026, https://www.mdpi.com/1424-8220/24/6/1874