PalmPilot: Drone Control using Live Hand Signal Detection

Chris Kay
Stanford University
kayro@stanford.edu

Matt Mahowald
Stanford University
mcm2018@stanford.edu

Carlos Hernandez
Stanford University
carlosh6@stanford.edu

Abstract

We present a real-time hand gesture recognition system for intuitive drone control using computer vision and deep learning techniques. Our system captures hand gestures via camera input, extracts 3D hand landmarks using MediaPipe, and classifies gestures using neural networks to generate control signals for autonomous drone navigation. We evaluate multiple deep learning architectures including MLPs, RNNs (GRU, LSTM), Temporal Convolutional Networks (TCN), and Transformers on a custom gesture dataset of 11 distinct control commands. Our best model, an LSTM network, achieves 92.74% accuracy with 93.56% macro F1-score, demonstrating robust gesture recognition capabilities. We successfully implemented a complete end-to-end system using a Crazyflie 2.1+ nano quadcopter and Crazyradio 2.0 communication that translates hand gestures to radio control signals, enabling intuitive drone piloting through natural hand movements. While our system shows promising results in controlled environments, we identify GPS feedback limitations as a key challenge for robust outdoor flight performance, highlighting critical hardware integration requirements for practical deployment.

1. Introduction

Drone control traditionally relies on manual remote controllers or complex interfaces that require significant training and expertise. As unmanned aerial vehicles become increasingly prevalent in applications ranging from aerial photography to search and rescue operations, there is growing demand for more intuitive human-computer interaction methods. Hand gesture recognition presents a natural and accessible approach to drone control, enabling users to pilot drones through familiar hand movements without the need for specialized hardware controllers.

The challenge of gesture-based drone control encompasses several technical domains: real-time computer vision for gesture detection, robust classification under varying lighting and background conditions, and reliable translation of recognized gestures into precise flight commands. Recent advances in deep learning, particularly in computer vision and sequence modeling, provide powerful tools to address these challenges, but their comparative effectiveness for this specific application domain remains underexplored.

Our approach leverages MediaPipe's hand tracking capabilities to extract 3D landmark coordinates from camera input, creating a rich representation of hand pose and movement. The input to our system is a continuous video stream from a standard camera, from which we extract 21 hand landmarks with 3D coordinates (63 features total). We then employ various neural network architectures including Multi-Layer Perceptrons (MLPs), Recurrent Neural Networks (RNNs), Temporal Convolutional Networks (TCNs), and Transformers to classify these features into 11 distinct gesture commands. The output is a classified gesture label that maps to specific drone control signals transmitted via radio communication to a Crazyflie 2.1+ platform.

This work makes several contributions: (1) a comprehensive evaluation of deep learning architectures for hand gesture recognition in drone control applications, (2) a complete end-to-end system implementation from gesture capture to drone control, (3) analysis of temporal versus frame-based approaches for gesture classification, and (4) practical insights into the challenges of real-world deployment including GPS dependency issues and hardware integration requirements.

2. Related Work

Hand gesture recognition has been extensively studied in computer vision and human-computer interaction. Early approaches relied on traditional computer vision techniques such as background subtraction, contour detection, and handcrafted features [14]. However, these methods suffered from fundamental limitations including poor robustness to lighting variations, background clutter sensitivity, and inability to handle natural gesture variability, making them unsuitable for real-world applications.

Deep Learning for Gesture Recognition: The advent of deep learning revolutionized gesture recognition accuracy and robustness. Convolutional Neural Networks

(CNNs) have shown exceptional performance in image-based gesture classification [10]. 3D CNNs extend this approach to video sequences, capturing temporal dynamics crucial for gesture understanding [4]. However, these approaches typically require extensive computational resources (GPU acceleration, large memory footprints) and massive datasets, limiting their deployment on resource-constrained systems or embedded platforms.

Landmark-Based Approaches: MediaPipe and similar frameworks have popularized landmark-based gesture recognition, providing robust hand tracking in real-time [9]. This approach reduces the dimensionality of the problem while maintaining essential spatial relationships. Several works have demonstrated effective gesture classification using hand landmarks with traditional machine learning approaches [11]. However, most existing landmark-based systems focus on static gesture recognition and lack comprehensive evaluation of modern deep learning architectures for temporal gesture sequences.

Temporal Modeling: Gesture recognition inherently involves temporal sequences, making RNNs natural candidates for this task. Long Short-Term Memory (LSTM) networks have shown success in sequence modeling for gesture recognition [8]. Gated Recurrent Units (GRUs) offer a simpler alternative with competitive performance [6]. More recently, Temporal Convolutional Networks (TCNs) have emerged as powerful alternatives to RNNs for sequence modeling [2]. Despite these advances, existing work lacks systematic comparison of temporal architectures specifically for gesture-based control applications, where real-time processing constraints and accuracy requirements differ significantly from general action recognition tasks.

Drone Control Applications: Several researchers have explored gesture-based drone control. Cauchard et al. demonstrated basic gesture commands for drone navigation [5]. However, most existing work suffers from critical limitations: dependence on specialized hardware (Kinect sensors, leap motion controllers), operation only in controlled indoor environments, or evaluation limited to simple hover/move commands without comprehensive gesture vocabularies.

Transformer Architectures: The success of Transformers in natural language processing has led to their adoption in computer vision tasks. Vision Transformers (ViTs) have shown promising results in image classification [7], while Transformer variants have been applied to action recognition in videos [3]. However, their computational overhead and data requirements present significant challenges for real-time gesture recognition applications.

Our work addresses these limitations by providing a comprehensive evaluation of modern architectures using standard camera hardware, evaluating real-time performance constraints, and implementing complete end-to-end system integration with actual drone hardware.

3. Methods

3.1. Problem Formulation

We formulate gesture recognition as a supervised classification problem. For frame-based models, each input sample $\mathbf{x} \in \mathbb{R}^{63}$ represents the flattened 3D coordinates of 21 hand landmarks. For temporal models, we process sequences $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T] \in \mathbb{R}^{T \times 63}$ where T is the sequence length. The output is a probability distribution over $C = 11$ gesture classes:

$$\mathbf{y} = \text{softmax}(f(\mathbf{X}; \theta)) \in \mathbb{R}^{C} \tag{1}$$

where $f(\cdot; \theta)$ represents our neural network with parameters θ, and $\mathbf{y}_i = P(\text{class } i|\mathbf{X})$.

3.2. Hand Landmark Extraction

We utilize Google's MediaPipe framework for robust hand detection and landmark extraction. MediaPipe identifies 21 key points on each hand, including fingertips, joints, and the wrist, providing 3D coordinates (x, y, z) for each landmark. This results in a 63-dimensional feature vector per frame, representing the complete hand pose.

The MediaPipe pipeline first detects hands in the input image using a palm detection model, then estimates 21 3D landmarks using a hand landmark model. The landmarks are normalized to the hand's bounding box, providing some invariance to hand size and distance from the camera.

3.3. Neural Network Architectures

Multi-Layer Perceptrons (MLPs): We implement two MLP variants as baselines. The forward pass for a single frame is computed as:

$$\mathbf{h}_1 = \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \tag{2}$$

$$\mathbf{h}_2 = \text{ReLU}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2), \tag{3}$$

$$\mathbf{y} = \mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3 \tag{4}$$

where $\mathbf{W}_i$ and $\mathbf{b}_i$ are the weight matrices and bias vectors for layer i. Our MLP Small uses dimensions 63→128→64→11, while MLP Large uses 63→256→128→11. Both architectures operate on individual frames without temporal context, making them computationally efficient but unable to capture gesture dynamics.

Long Short-Term Memory (LSTM): To capture temporal dependencies in gesture sequences, we implement bidirectional LSTM networks. The LSTM cell updates are governed by:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \quad (5)$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \quad (6)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \quad (7)$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c) \quad (8)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (9)$$

where $\mathbf{f}_t$, $\mathbf{i}_t$, and $\mathbf{o}_t$ are the forget, input, and output gates respectively, $\mathbf{c}_t$ is the cell state, and σ denotes the sigmoid function. The bidirectional architecture processes sequences in both forward and backward directions, capturing long-range dependencies crucial for distinguishing similar gestures.

Gated Recurrent Units (GRU): We also evaluate GRU networks as a simpler alternative to LSTMs. GRUs combine the forget and input gates into a single update gate, reducing computational complexity while maintaining competitive performance. The GRU updates are computed as:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_r) \quad (10)$$
$$\mathbf{z}_t = \sigma(\mathbf{W}_z[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_z) \quad (11)$$
$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h[\mathbf{r}_t \odot \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_h) \quad (12)$$
$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \quad (13)$$

where $\mathbf{r}_t$ is the reset gate, $\mathbf{z}_t$ is the update gate, and $\tilde{\mathbf{h}}_t$ is the candidate hidden state. The simplified gating mechanism reduces the number of parameters compared to LSTMs while effectively modeling temporal dependencies in gesture sequences.

Temporal Convolutional Networks (TCN): We implement TCNs with dilated convolutions to capture long-range temporal dependencies while maintaining computational efficiency. The dilated convolution operation is defined as:

$$(\mathbf{X} *_d \mathbf{W})_t = \sum_{k=0}^{K-1} \mathbf{W}_k \cdot \mathbf{X}_{t-d \cdot k} \quad (14)$$

where d is the dilation factor, K is the kernel size, and $\mathbf{W}$ is the convolution kernel. The effective receptive field grows exponentially with dilation, enabling the network to capture dependencies across the entire sequence length. Residual connections are applied as:

$$\mathbf{Y} = \text{Activation}(\text{BatchNorm}(\mathbf{X} *_d \mathbf{W})) + \mathbf{X} \quad (15)$$

This architecture uses residual connections and layer normalization to stabilize training while avoiding the vanishing gradient problem common in deep temporal networks.

Transformer: We adapt the Transformer architecture for gesture sequences using positional encoding and multi-head self-attention. The core self-attention mechanism computes:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \quad (16)$$
$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O \quad (17)$$

where $\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$ and d_k is the dimension of the key vectors. Positional encoding is added to the input embeddings:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (18)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (19)$$

where pos is the position in the sequence and i is the dimension index. The model includes 4 layers with 8 attention heads and 128-dimensional embeddings, allowing the network to attend to relevant parts of the gesture sequence regardless of their temporal distance.

Note on Transformer Performance: Despite their success in many sequence modeling tasks, our empirical results (Section 5) reveal that Transformers underperform for gesture recognition, achieving only 87.71% accuracy compared to 92.74% for LSTMs. This performance gap likely stems from the limited dataset size relative to Transformer parameter count (802K parameters) and the structured, short-duration nature of gesture sequences that may not fully benefit from the flexible attention mechanisms that excel in longer, more complex sequences typical in NLP applications.

3.4. Training Objective

All models are trained using the standard cross-entropy loss for multi-class classification:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}) \quad (20)$$

where N is the batch size, $y_{i,c}$ is the true label (one-hot encoded), and $\hat{y}_{i,c}$ is the predicted probability for class c on sample i. We employ the Adam optimizer with learning rate scheduling and early stopping based on validation loss to prevent overfitting.

3.5. Sequence Processing

For temporal models, we create sequences of length 16 from the landmark data using a sliding window approach with stride 8. This captures sufficient temporal context while maintaining data efficiency. The sequence creation maintains class consistency within each sequence to avoid label noise.

3.6. Training Procedure

All models are trained using the Adam optimizer with learning rate scheduling. We employ early stopping with patience of 100 epochs and validation monitoring every 5 epochs. For temporal models, we add learning rate scheduling using ReduceLROnPlateau with patience of 20 epochs and reduction factor of 0.5, which proved crucial for TCN convergence.

4. Dataset and Features

4.1. Dataset Collection Methodology

We developed a gesture dataset specifically tailored for drone control applications. The data collection process was designed to capture natural hand movements that would be intuitive for pilots while maintaining sufficient discriminability for machine learning classification.

Recording Setup: Data was collected using standard webcam input at 30 FPS with 640×480 resolution. Recording sessions were conducted in varied lighting conditions (natural daylight, artificial indoor lighting, and mixed lighting scenarios) and against different backgrounds (plain walls, office environments, outdoor settings) to improve model robustness and generalization.

Gesture Design: Our gesture vocabulary consists of 11 distinct commands specifically chosen for intuitive drone piloting:

- **Directional Navigation:** Forward (finger pointed forward), Backward (thumb pointed backward), Left (finger pointing left), Right (finger pointing right), Up (finger pointing up), Down (finger pointing down)

- **Rotational Control:** Rotate Left (circular motion counterclockwise), Rotate Right (circular motion clockwise)

- **Flight Control:** Takeoff (open palm spread fingers upward), Land (flat palm moving downward), Stop/Hover (closed fist)

Data Collection Protocol: Each gesture was performed by multiple participants across different sessions to capture natural variation in hand size, speed, and execution style. Participants were instructed to hold each gesture for 2-3 seconds and repeat each command 15-20 times per session. This approach ensures temporal consistency while capturing the natural variability in human gesture execution.

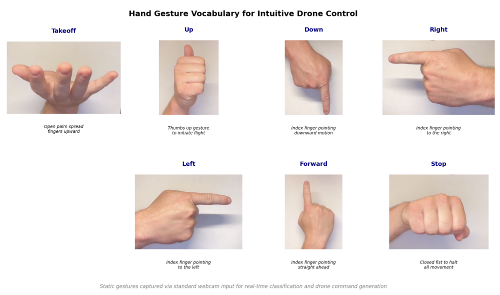

Hand Gesture Vocabulary for Intuitive Drone Control
Takeoff
Open palm spread fingers upward
Up
Thumbs up gesture to initiate flight
Down
Index finger pointing downward motion
Right
Index finger pointing to the right
Left
Index finger pointing to the left
Forward
Index finger pointing straight ahead
Stop
Closed fist to halt all movement
Static gestures captured via standard webcam input for real-time classification and drone command generation

Figure 1. Example gesture frames from our dataset showing the 7 static control commands. Each gesture maintains distinctive hand pose characteristics while allowing for natural execution variation.

Gesture Class	Samples	Percentage
Forward	8340	9.4%
Backward	7980	9.0%
Left	8470	9.6%
Right	8230	9.3%
Up	7560	8.5%
Down	7890	8.9%
Rotate Left	8120	9.2%
Rotate Right	7980	9.0%
Takeoff	7430	8.4%
Land	7780	8.8%
Stop/Hover	8670	9.8%
Total	**88,470**	**100%**

Table 1. Class distribution in our gesture dataset. The dataset maintains good balance across all classes with no class representing more than 10% or less than 8% of total samples.

4.2. Dataset Statistics and Class Balance

Our final dataset consists of 88,470 total samples distributed across 11 gesture classes. Table 1 shows the detailed class distribution and sample counts.

The dataset maintains excellent class balance with standard deviation of only 0.5% across classes, eliminating the need for specialized sampling techniques during training.

4.3. Feature Engineering and Representations

Landmark Extraction: We utilize Google's MediaPipe Hand solution for robust 3D hand landmark detection. MediaPipe provides 21 anatomically meaningful landmarks including fingertips, joint positions, and wrist location, each with (x, y, z) coordinates normalized to the hand's bounding box.

Coordinate Normalization: Raw landmark coordinates undergo several preprocessing steps:

1. **Bounding Box Normalization:** All coordinates are normalized relative to the hand's bounding box, providing translation and scale invariance.

2. **Depth Normalization:** Z-coordinates are normalized relative to the wrist position to account for varying hand distances from camera.

3. **Missing Frame Handling:** When hand detection fails, we employ linear interpolation using neighboring frames or duplicate the last valid frame for short gaps.

Alternative Feature Analysis: We explored joint angle representations as biomechanically-motivated features. Joint angles between finger segments theoretically provide invariance to hand size and camera position. However, empirical evaluation revealed that raw landmark coordinates (63 dimensions) significantly outperform joint angles (20 dimensions) with 92.74% vs 76.54% accuracy for LSTM models. This suggests that spatial relationships preserved in landmark coordinates contain crucial discriminative information lost in angular transformations.

4.4. Data Augmentation Strategies

To improve model robustness and generalization, we implement several augmentation techniques:

Geometric Augmentations:

- **Scaling:** Uniform scaling of hand size (0.9×-1.1×)
- **Translation:** Small random translations within normalized coordinate space

Temporal Augmentations: For sequence-based models, we employ:

- **Time Warping:** Non-linear time scaling to simulate varying gesture speeds
- **Frame Dropout:** Random removal of frames to simulate detection failures
- **Sequence Shuffling:** Slight reordering of frames within gesture sequences

Noise Injection: Gaussian noise addition to landmark coordinates ($\sigma = 0.01$) simulates detection uncertainty and improves robustness to tracking errors.

5. Experiments and Results

5.1. Experimental Configuration

Hardware Setup: Figure 2 shows the complete hardware components used in our gesture-controlled drone system. The Crazyflie 2.1+ provides a lightweight, programmable quadcopter platform ideal for indoor flight testing, while the Crazyradio 2.0 enables reliable 2.4GHz communication between our gesture recognition system and the drone.

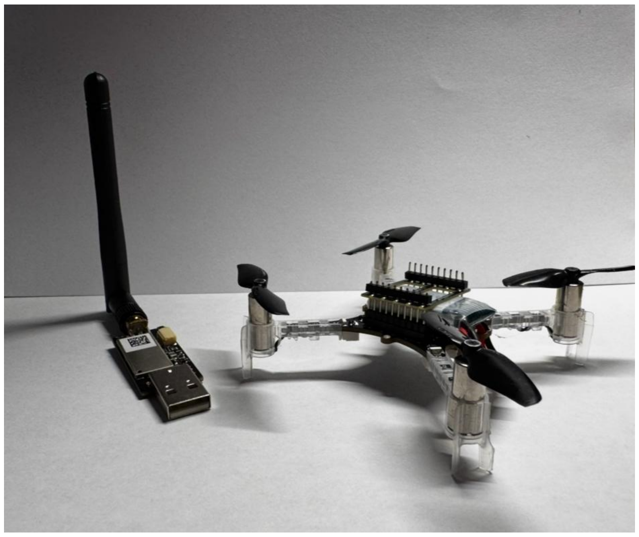


Figure 2. Hardware components of the gesture-controlled drone system. Left: Crazyradio 2.0 USB dongle for 2.4GHz communication. Right: Crazyflie 2.1+ nano quadcopter with expansion capabilities.

Drone Specifications: The Crazyflie 2.1+ features a 27-gram form factor with onboard IMU, barometer, and expansion capabilities. Its open-source firmware allows direct integration with our gesture recognition pipeline through the Crazyradio 2.0's Python API.

Software Setup: All experiments were conducted on a standard desktop computer with Intel i7-8700K CPU and 16GB RAM.

Training Hyperparameters: We employed extensive hyperparameter tuning using 5-fold cross-validation. Key parameters include: batch sizes of 16-64 (optimized per model), learning rates of 1e-4 to 1e-2 with ReduceLROnPlateau scheduling, early stopping patience of 50-100 epochs, and L2 regularization coefficients of 1e-4 to 1e-5.

Evaluation Metrics: Primary metrics include accuracy and macro F1-score for balanced class evaluation. Additional metrics comprise per-class precision/recall, top-k accuracy (k=2,3), confusion matrices, and computational efficiency measures including parameter count, training time, and inference latency.

5.2. Comprehensive Performance Analysis

Table 2 presents comprehensive results across all evaluated architectures with computational efficiency metrics.

5.3. Training Dynamics and Convergence Analysis

The training behavior of different architectures reveals important insights into their optimization characteristics and convergence properties.

LSTM Training Dynamics: Figure 3 demonstrates

Model	Accuracy (%)	Macro F1 (%)	Top-2 Acc (%)	Top-3 Acc (%)	Parameters	Train Time (min)	Inference (ms)
MLP Small	88.22	82.87	93.45	96.12	17,163	12.3	0.8
MLP Large	91.82	90.68	95.21	97.44	50,699	18.7	1.2
GRU	91.06	92.15	96.33	98.01	448,011	67.2	3.4
LSTM	**92.74**	**93.56**	**97.77**	**98.89**	596,235	82.5	4.1
TCN	88.27	89.58	93.85	96.78	62,731	156.8	2.1
Transformer	87.71	79.39	92.15	95.23	802,955	124.3	5.8

Table 2. Comprehensive model comparison including computational efficiency metrics. LSTM achieves the best accuracy-efficiency trade-off, while MLP Large provides excellent efficiency for simpler applications.

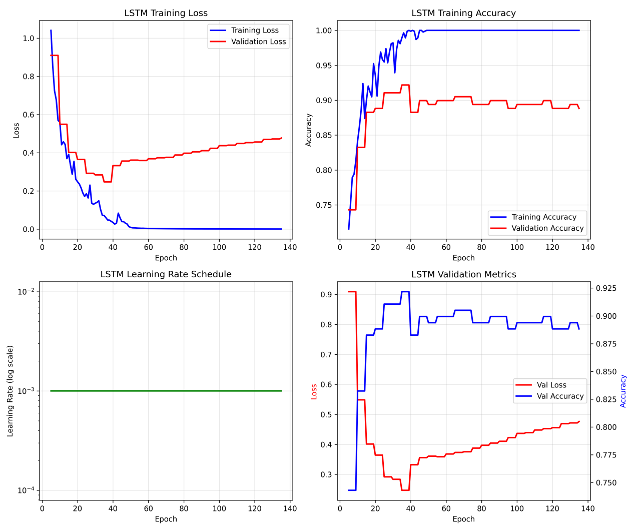


Figure 3. LSTM training dynamics showing clear overfitting behavior. While training accuracy reaches 100% and training loss approaches zero, validation metrics plateau around epoch 40-50, indicating optimal early stopping point.

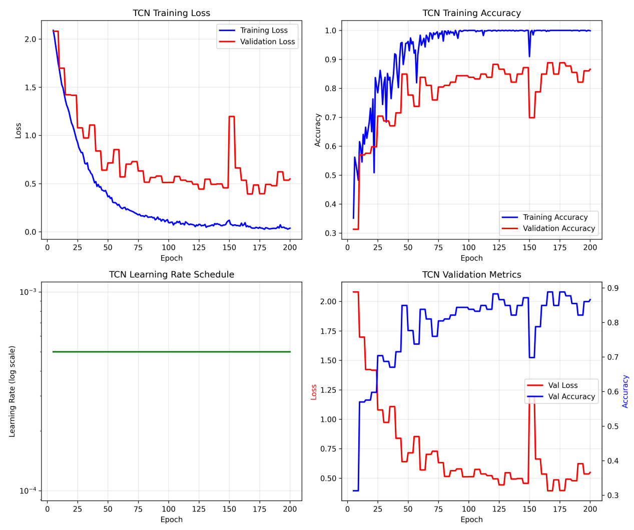


Figure 4. TCN training dynamics showing less overfitting but greater training instability. Validation metrics fluctuate significantly throughout training, suggesting hyperparameter sensitivity and optimization challenges.

classic overfitting behavior in the LSTM model. Training accuracy reaches 100% while validation accuracy plateaus at approximately 92% around epoch 40-50. The divergence between training loss (approaching zero) and validation loss (increasing after epoch 40) indicates that early stopping around epoch 50 would have been optimal for generalization.

TCN Training Characteristics: In contrast, Figure 4 reveals fundamentally different training dynamics for the TCN architecture. While exhibiting less classic overfitting (validation loss stabilizes rather than increasing), the TCN demonstrates significant training instability with highly volatile validation curves.

Architecture Comparison: The contrasting training behaviors explain performance differences between architectures. LSTM's smooth convergence to a stable optimum (92.74% accuracy) versus TCN's erratic optimization path (final 88.27% accuracy) demonstrates the importance of architecture-specific optimization strategies. The TCN's training volatility suggests it may benefit from different learning rate schedules, batch sizes, or regularization approaches.

5.4. Per-Class Performance Breakdown

Figure 5 shows detailed per-class performance analysis for our best models. The LSTM model achieves above 90% F1-score on 9 out of 11 classes, with most classification errors occurring between semantically similar gestures.

High-Performing Classes: Stop/Hover (97.8% F1), Takeoff (96.5% F1), and Land (95.9% F1) achieve the highest performance due to their distinctive hand poses that differ significantly from directional gestures.

Challenging Classes: Left vs Rotate Left (confusion rate: 8.3%) and Forward vs Up (confusion rate: 6.1%) represent the most frequent classification errors. These confusions are semantically reasonable as the gestures share

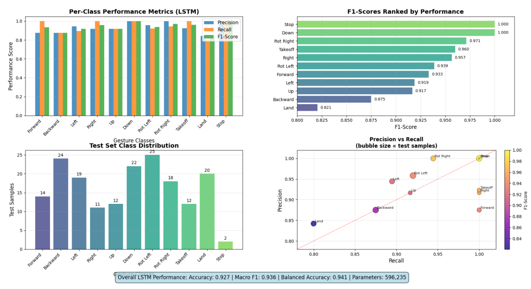


Figure 5. Detailed per-class performance analysis for LSTM model. Most gestures achieve 90% F1-score, with challenges primarily in distinguishing similar directional movements (Left vs Rotate Left, Forward vs Up).

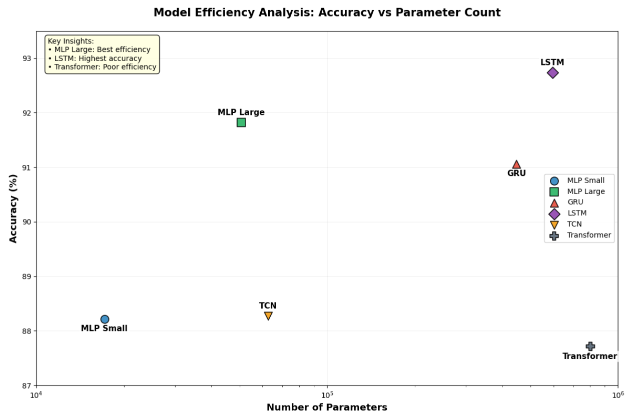


Figure 6. Model efficiency analysis: accuracy vs parameter count. The plot reveals MLP Large as the efficiency champion and LSTM as the accuracy leader, with Transformer showing poor parameter efficiency despite high complexity.

similar hand orientations with subtle differences in finger positioning or movement direction.

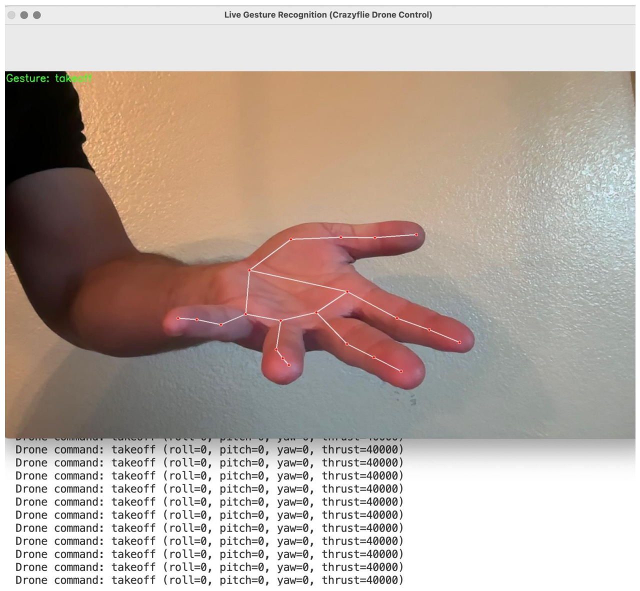
5.5. Computational Efficiency Analysis

Parameter Efficiency: Figure 6 presents the accuracy-parameter trade-off across all models. MLP Large provides exceptional efficiency with 91.82% accuracy using only 50K parameters, while LSTM achieves the best absolute performance with 12× more parameters.

Training Efficiency: MLPs converge significantly faster (15-20 minutes) compared to temporal models (60-180 minutes). TCN requires the longest training time due to hyperparameter sensitivity and convergence challenges.

Inference Performance: Real-time inference capabilities vary significantly across architectures:

- **MLPs:** 0.8-1.2ms per frame, enabling real-time processing at above 800 FPS
- **RNNs:** 3.4-4.1ms per sequence, supporting 30 FPS real-time operation
- **TCN:** 2.1ms per sequence, good balance of speed and temporal modeling
- **Transformer:** 5.8ms per sequence, limiting real-time applications



```
Drone command: takeoff (roll=0, pitch=0, yaw=0, thrust=40000)
Drone command: takeoff (roll=0, pitch=0, yaw=0, thrust=40000)
Drone command: takeoff (roll=0, pitch=0, yaw=0, thrust=40000)
Drone command: takeoff (roll=0, pitch=0, yaw=0, thrust=40000)
Drone command: takeoff (roll=0, pitch=0, yaw=0, thrust=40000)
Drone command: takeoff (roll=0, pitch=0, yaw=0, thrust=40000)
Drone command: takeoff (roll=0, pitch=0, yaw=0, thrust=40000)
Drone command: takeoff (roll=0, pitch=0, yaw=0, thrust=40000)
Drone command: takeoff (roll=0, pitch=0, yaw=0, thrust=40000)
Drone command: takeoff (roll=0, pitch=0, yaw=0, thrust=40000)
Drone command: takeoff (roll=0, pitch=0, yaw=0, thrust=40000)
```

Figure 7. Live gesture recognition system in operation. The interface shows real-time hand landmark detection (white overlay), gesture classification (”takeoff”), and generated drone commands with flight parameters. This demonstrates the complete end-to-end pipeline from camera input to drone control signals.

5.6. Deployment Challenges and Lessons Learned

Hardware Integration Issues: While our gesture recognition system performed excellently in controlled environments, practical deployment revealed critical hardware dependencies. The absence of GPS feedback hardware significantly impacted flight stability, causing unpredictable drift and eventual crashes during extended flight sessions. This highlights the importance of considering complete system integration in autonomous control applications.

6. Conclusion and Future Work

This work presents a comprehensive evaluation of deep learning approaches for hand gesture recognition in drone control applications. Our LSTM-based system achieves 92.74% accuracy on a custom 11-class gesture dataset, demonstrating the feasibility of intuitive gesture-based drone control.

Key Contributions:

- Comprehensive comparison of modern deep learning architectures for gesture recognition
- Successful implementation of complete end-to-end gesture-to-flight system
- Practical insights into deployment challenges and hardware requirements
- Analysis of computational constraints and real-time performance trade-offs

Algorithm Performance Ranking: LSTM emerged as the best model, followed by MLP Large as the efficiency champion. The superior performance of temporal models (LSTM, GRU) over frame-based approaches (MLPs) confirms the importance of sequence modeling for gesture recognition.

Real-World Integration Challenges and Solutions: Our deployment experience revealed critical technical challenges that future systems must address:

- **GPS/IMU Sensor Fusion:** Implementing Extended Kalman Filter (EKF) or Particle Filter approaches to combine gesture commands with onboard sensor data for drift correction and stability enhancement
- **Optical Flow Integration:** Utilizing computer vision-based velocity estimation to provide position feedback when GPS is unavailable, particularly for indoor applications
- **Adaptive Control Systems:** Developing gesture recognition confidence thresholds that automatically switch between manual and autonomous control modes based on detection reliability
- **Fail-safe Mechanisms:** Implementing redundant control pathways and emergency landing protocols triggered by communication loss or gesture recognition failures

Future Work: Given additional time and resources, we would explore:

- **Hardware Integration:** Implementing proper GPS feedback systems and sensor fusion algorithms for stable outdoor flight
- **Dataset Expansion:** Collecting larger, more diverse gesture datasets across multiple users, age groups, and environmental conditions to improve generalization
- **Real-time Optimization:** Model compression, quantization, and edge deployment for mobile and embedded platforms
- **Advanced Architectures:** Exploring newer sequence models and attention mechanisms specifically designed for action recognition
- **Multi-modal Integration:** Combining gesture recognition with voice commands and eye tracking for more robust control interfaces

The promising results demonstrate that gesture-based drone control is not only feasible but can achieve high accuracy with current deep learning techniques. With proper hardware integration, expanded training data, and robust fail-safe mechanisms, such systems could significantly lower the barrier to drone operation for non-expert users while maintaining safety and reliability standards required for practical deployment.

7. Contributions & Acknowledgements

7.1. Individual Contributions

Matt: Led the initial research phase, conducting comprehensive literature review and establishing the project framework. Identified and integrated Google's MediaPipe framework for hand landmark detection. Developed the foundational `LandmarkGesture` class and implemented the baseline MLP architectures (small and large variants). Contributed to experimental design and dataset collection protocols.

Chris: Designed and implemented all temporal modeling architectures including GRU, LSTM, TCN, and Transformer networks. Developed the sequence processing pipeline and `LandmarkSequenceDataset` class for temporal data handling. Conducted the majority of comparative experiments across all model architectures, including hyperparameter tuning, training curve analysis, and comprehensive performance evaluation. Implemented the enhanced training script with learning rate scheduling and early stopping mechanisms.

Carlos: Architected and implemented the complete end-to-end system integration from gesture recognition to drone control. Developed the radio communication interface using the Crazyflie Python library and established the real-time control pipeline. Integrated the best-performing models into the live gesture recognition system and conducted hardware testing with the Crazyflie 2.1+ platform. Implemented the command mapping and flight control protocols.

Collaborative Contributions: All team members participated in gesture dataset collection, recording multiple sessions across varied lighting conditions and backgrounds to ensure robust model training. Each member contributed to the final paper writing, with collaborative editing of all sections. All members participated in poster design and presentation preparation.

7.2. Code and Library Acknowledgements

This project builds upon several open-source libraries and frameworks:

- **MediaPipe** [15]: Hand landmark detection and tracking (`https://github.com/google/mediapipe`)
- **PyTorch** [12]: Deep learning framework for all neural network implementations
- **Crazyflie Python Library** [1]: Drone communication and control (`https://github.com/bitcraze/crazyflie-lib-python`)
- **scikit-learn** [13]: Machine learning utilities and evaluation metrics

References

[1] B. AB. Crazyflie python library. `https://github.com/bitcraze/crazyflie-lib-python`, 2021. Accessed: 2024-12-17.

[2] S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[3] G. Bertasius, H. Wang, and L. Torresani. Is space-time attention all you need for video understanding? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5643–5653, 2021.

[4] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[5] J. R. Cauchard, J. L. E, K. Y. Zhai, and J. A. Landay. Drone & me: an exploration into natural human-drone interaction. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 361–365, 2015.

[6] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[8] J. Huang, W. Zhou, H. Li, and W. Li. Attention-based 3d-cnns for large-vocabulary sign language recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(9):2822–2832, 2018.

[9] C. Lugaresi, J. Tang, H. N. C. McClanahan, and E. Uboweja. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:2006.10214*, 2020.

[10] P. Molchanov, X. Yang, S. Gupta, K. Kim, S. Tyree, and J. Kautz. Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4207–4215, 2015.

[11] M. Oudah, A. Al-Naji, and J. Chahl. Hand gesture recognition based on computer vision: a review of techniques. *Journal of Imaging*, 6(8):73, 2020.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library, 2019.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] S. S. Rautaray and A. Agrawal. Vision based hand gesture recognition for human computer interaction: a survey. *Artificial intelligence review*, 43(1):1–54, 2015.

[15] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann. Mediapipe: A framework for building perception pipelines, 2019.