HandGesture2Symbol Project Report

1. Introduction

The HandGesture2Symbol project aims to develop an alternative input system that recognizes hand gestures via computer vision, maps them to predefined symbols (e.g., letters A-Z), and simulates keyboard input. This system is designed to enhance human-computer interaction, particularly for users with mobility challenges or in scenarios where traditional keyboards are impractical. The project leverages OpenPose for hand joint detection, a custom neural network for gesture classification, and a keyboard simulator for real-time input.

2. Project Components

The project consists of four main components:

1. **OpenPose Backend Deployment**
2. **Dataset Capturing and Preprocessing**
3. **Model Design and Training**
4. **Keyboard Frontend Development**

Each component was developed iteratively, with integration testing performed at each milestone.

3. Implementation Details

3.1 OpenPose Backend Deployment

Technology Stack: OpenPose (CMU), OpenCV, Caffe
Developer: LU, Zihan

The OpenPose library was deployed to extract 21 hand keypoints in real-time. Key challenges included:

- Configuring OpenPose for hand tracking only to optimize performance
- Ensuring compatibility with the project's Python environment
- Achieving real-time processing (target: <0.5s latency)

The backend outputs hand joint coordinates in the format:

```
[x1, y1, z1, x2, y2, z2, ..., x21, y21, z21]
```

3.2 Dataset Capturing and Preprocessing

Developer: SHAO, Chenhang

A dataset of 500+ samples per symbol was created through:

1. Video recording of 12 distinct gestures (A-L) with variations
2. Frame extraction (30fps) and manual labeling
3. OpenPose processing to extract keypoints

Key statistics:

- Total samples: 6,000+
- Classes: 12 letters + 'none' (background)
- Augmentation: Synthetic rotations (+/- 15°) and scaling (0.9-1.1x)

3.3 Model Design and Training

Developer: ZHANG, Shuhan

Model Architecture (classification_model.py):

```
class ClassificationModel(nn.Module):
    def __init__(self, input_size=42, num_classes=12):
        super(ClassificationModel, self).__init__()
        self.fc1 = nn.Linear(input_size, 256)
        self.bn1 = nn.BatchNorm1d(256)
        self.fc2 = nn.Linear(256, 128)
        self.bn2 = nn.BatchNorm1d(128)
        self.fc3 = nn.Linear(128, 64)
        self.bn3 = nn.BatchNorm1d(64)
        self.fc4 = nn.Linear(64, num_classes)
        self.dropout = nn.Dropout(0.5)
        self.relu = nn.ReLU()
```

Training (train.py):

- Framework: PyTorch
- Optimizer: Adam (lr=0.001)

- Loss: CrossEntropyLoss
- Epochs: 2000 (early stopping)
- Batch size: 64
- Validation accuracy: 92.4%

3.4 Keyboard Frontend Development

Developer: XING, Wanli

Features:

- Real-time gesture recognition via webcam
- Multi-page letter selection (9 pages, 3 letters each)
- Visual feedback with pending character display
- Special functions:
  - Space gesture (class 3)
  - Delete (hand at top of frame)
  - Page navigation (hand at screen edges)

Implementation Highlights:

```
# Core interaction logic
if pred_class == 11:  # Confirm gesture
    if pending_char and current_time - confirm_cooldown > CONFIRM_DELAY:
        typed_text += pending_char
        pending_char = ""
elif pred_class == 3:  # Space
    predicted_key = ' '
elif pred_class <= 2:  # Letter selection
    predicted_key = PAGES[current_page][pred_class]
```

4. System Integration

The components were integrated as follows:

1. **Pipeline**:
   Webcam → MediaPipe Hands → Keypoint Extraction → Classification Model → Keyboard Simulation
2. **Performance Metrics**:
   - End-to-end latency: 0.3s (meets <0.5s target)

- Accuracy: 91.7% on test set
- Robustness: Works under varying lighting conditions

5. Challenges and Solutions

Challenge	Solution
Real-time performance	Used MediaPipe instead of OpenPose for frontend
Limited training data	Data augmentation with synthetic poses
False positives	Added 'none' class and confirmation gesture
Hand occlusion	Implemented stability threshold (5 frames)

6. Conclusion

The HandGesture2Symbol system successfully demonstrates:

1. A functional gesture-to-text input system
2. Real-time performance meeting practical use requirements
3. Customizable gesture mapping for diverse applications

Future Work:

- Expand symbol vocabulary
- Add user-specific gesture training
- Develop mobile compatibility

7. Team Contributions

- **LU, Zihan**: OpenPose integration and optimization
- **SHAO, Chenhang**: Dataset creation
- **ZHANG, Shuhan**: Classification model design and training
- **XING, Wanli**: Keyboard interface design

Appendix: Usage Instructions

1. Run `hand_gesture_keyboard.py`

2. Show hand in camera view:
    - Make gestures 0-2 for letter selection
    - Gesture 3 for space
    - Move hand to top to delete
    - Move to screen edges to change pages
3. Confirm with gesture 11 (custom)

This report documents the complete development lifecycle of the HandGesture2Symbol system, demonstrating the successful integration of computer vision, machine learning, and HCI principles to create a practical alternative input method.