Smart Glove for VR by Magic Mitts

Team members: Rabiat Sadiq, Blake Novosad, John Navarro, Juan Casillas

Design Package V2 – 10/03/2024

Description: This document contains the details of the software planning in the project

Project Name: Smart Glove for VR.

Team Name: Magic Mitts

Team Members: Rabiat Sadiq, Blake Novosad, John Navarro, Juan Casillas

Instructor: August Allo & Johnathan Votion Ph.D

Smart Glove for VR by Magic Mitts

Team members: Rabiat Sadiq, Blake Novosad, John Navarro, Juan Casillas

Software Overview and Pseudo Code

The software component of our Smart Glove integrates multiple systems, including the ESP32 microcontroller, Unity environment, and electromagnetic braking, to deliver an immersive VR experience.

Pseudo code for Full Hand Control System with Electromagnetic Braking and ESP32 is as followed:

Initialization:

1. Start
    - Setup ESP32 microcontroller
    - Setup Power Pack (Dual 3.7V Battery Cells with Boost Converters)
    - Initialize all 5 flex sensors connected to ESP32 GPIO
    - Initialize the electromagnetic braking system with boost converters and MOSFETs
    - Setup Serial Communication between ESP32 and Unity
    - Initialize all finger joint transforms in Unity
    - Initialize interactable objects in Unity with appropriate colliders

Main Control Loop in ESP32:

1. Read values from all 5 flex sensors (Thumb, Index, Middle, Ring, Pinky)
2. Map sensor values to bending angles for each finger
3. Send mapped angles to Unity via Serial in CSV format
4. Check for incoming commands from Unity:
    - If command == "BRAKE": Activate corresponding electromagnetic brake
    - Else if command == "RELEASE": Deactivate all electromagnetic brakes

Unity Main Loop:

1. Read sensor values from ESP32
2. Parse CSV data into individual finger joint angles
3. Apply Finger Movements:
    - For each finger (Thumb, Index, Middle, Ring, Pinky), apply rotation to each joint (Knuckle, Middle, Tip) based on sensor data
4. Check for Object Interactions:
    - If hand collides with object: Send "BRAKE" command to ESP32
    - Else if hand moves away from the object: Send "RELEASE" command to ESP32

Electromagnetic Braking System Controlled by ESP32:

- If "BRAKE" command received, activate corresponding electromagnet, control MOSFETs, and maintain braking force until the "RELEASE" command.
- If "RELEASE" command received, deactivate all electromagnets and turn off boost converters to minimize power consumption.

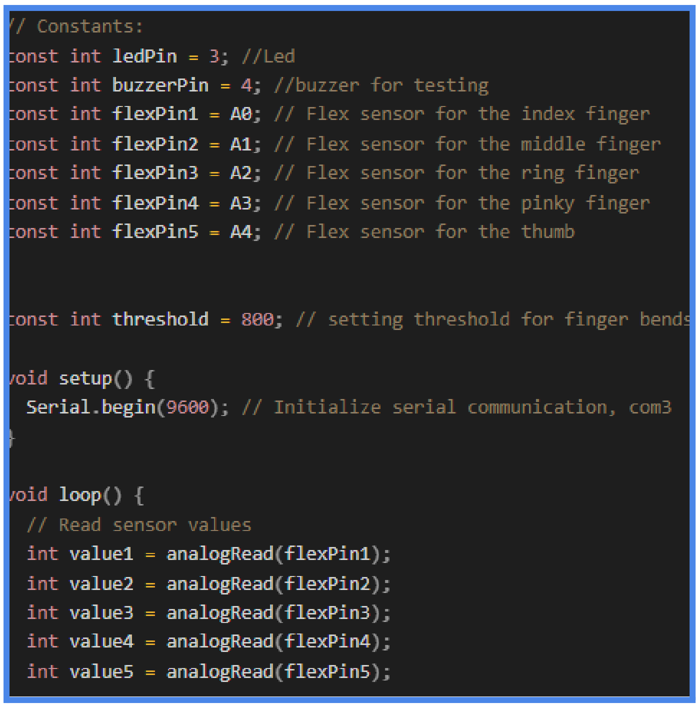
Clean Up on Exit:

Smart Glove for VR by Magic Mitts

Team members: Rabiat Sadiq, Blake Novosad, John Navarro, Juan Casillas

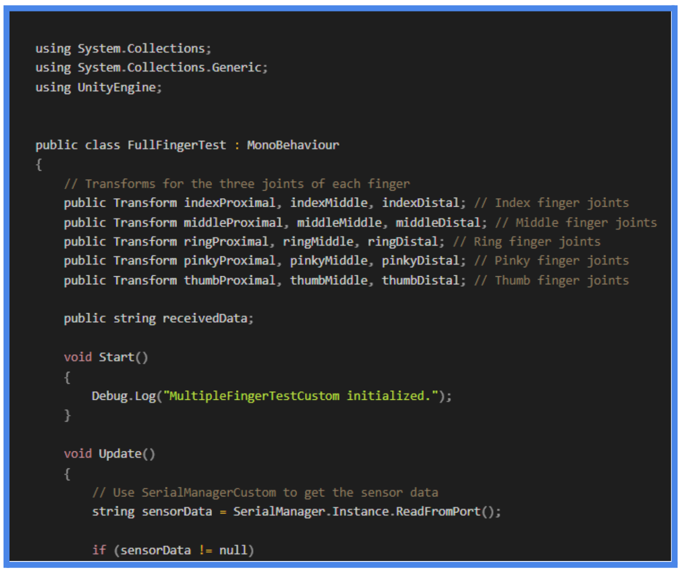
- Close serial ports on ESP32 and Unity
- Deactivate all electromagnetic braking systems
- Power down boost converters

Code Details



```
/ Constants:
onst int ledPin = 3; //Led
onst int buzzerPin = 4; //buzzer for testing
onst int flexPin1 = A0; // Flex sensor for the index finger
onst int flexPin2 = A1; // Flex sensor for the middle finger
onst int flexPin3 = A2; // Flex sensor for the ring finger
onst int flexPin4 = A3; // Flex sensor for the pinky finger
onst int flexPin5 = A4; // Flex sensor for the thumb


onst int threshold = 800; // setting threshold for finger bends

oid setup() {
 Serial.begin(9600); // Initialize serial communication, com3


oid loop() {
 // Read sensor values
 int value1 = analogRead(flexPin1);
 int value2 = analogRead(flexPin2);
 int value3 = analogRead(flexPin3);
 int value4 = analogRead(flexPin4);
 int value5 = analogRead(flexPin5);
```

Figure 01: Arduino Setup

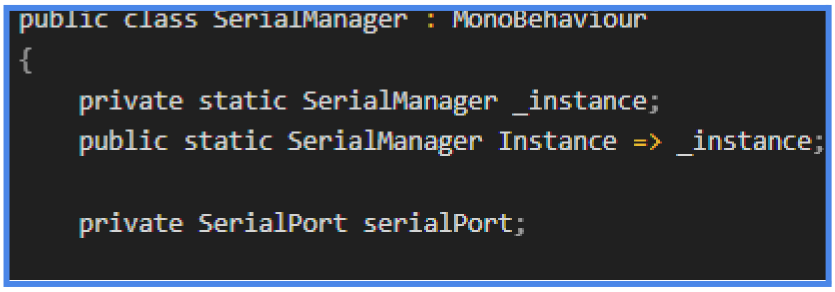


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class FullFingerTest : MonoBehaviour
{
    // Transforms for the three joints of each finger
    public Transform indexProximal, indexMiddle, indexDistal; // Index finger joints
    public Transform middleProximal, middleMiddle, middleDistal; // Middle finger joints
    public Transform ringProximal, ringMiddle, ringDistal; // Ring finger joints
    public Transform pinkyProximal, pinkyMiddle, pinkyDistal; // Pinky finger joints
    public Transform thumbProximal, thumbMiddle, thumbDistal; // Thumb finger joints

    public string receivedData;

    void Start()
    {
        Debug.Log("MultipleFingerTestCustom initialized.");
    }

    void Update()
    {
        // Use SerialManagerCustom to get the sensor data
        string sensorData = SerialManager.Instance.ReadFromPort();

        if (sensorData != null)
```

Figure 02: Unity Glove inputs

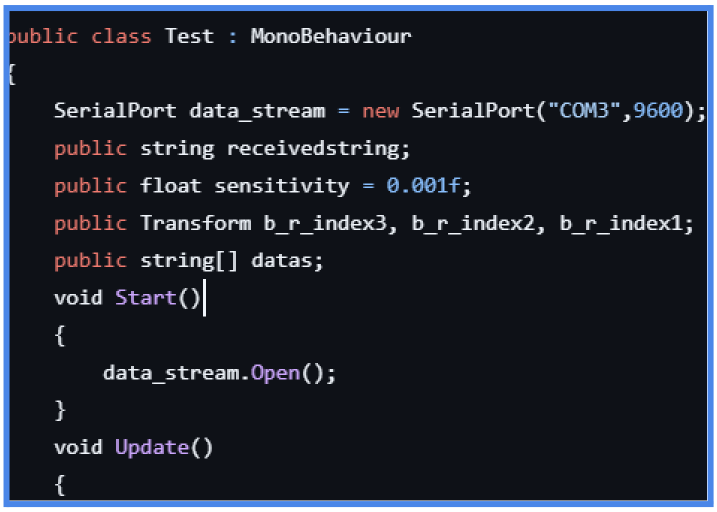
Smart Glove for VR by Magic Mitts

Team members: Rabiat Sadiq, Blake Novosad, John Navarro, Juan Casillas



```
public class SerialManager : MonoBehaviour
{
    private static SerialManager _instance;
    public static SerialManager Instance => _instance;

    private SerialPort serialPort;
```

Figure 03: Unity serial Manager class



```
public class Test : MonoBehaviour
{
    SerialPort data_stream = new SerialPort("COM3",9600);
    public string receivedstring;
    public float sensitivity = 0.001f;
    public Transform b_r_index3, b_r_index2, b_r_index1;
    public string[] datas;
    void Start()
    {
        data_stream.Open();
    }
    void Update()
    {
```

Figure 04: Starter code

<u>Arduino Setup: (*Figure 01*)</u>

- Functions:
  - `setup()`: Initializes pin modes, serial communication at 9600 baud, and sets up pins for LED and buzzer.
  - `loop()`: Reads flex sensor values, maps them to bending angles, and sends data via serial communication. Controls LED and buzzer based on flex sensor thresholds.

<u>Unity C# Script:(*Figure 02*)</u>

- Functions:
  - `Start()`:Initializes serial communication, references the SerialManager, and connects to the specified COM port to manage data exchange between Unity and the Arduino (ESP32).
  - `Update()`: Reads incoming data from Arduino, parses it, and maps values to control virtual hand's finger joints.
  - `Map()`: Utility function for mapping sensor values to rotation angles.
  - `OnApplicationQuit()`: Ensures proper closure of the serial port.

- `OnCollisionEnter()`: Detects collisions and sends feedback signals.
- `SendBuzzCommand()`: Sends a specific signal ("buzz") to the Arduino when an object is grabbed or touched, simulating haptic feedback or braking system.

- Software Code Progression:
  - **Single Finger:** Initial setup for one flex sensor, controlling one finger in Unity.
  - **Multiple Fingers:** Expanded to handle input from five flex sensors, controlling all fingers with realistic bending.
  - **Feedback Integration:** Added buzzer feedback for interactions, simulating future electromagnetic braking or haptic feedback systems.
  - **Full Hand Tracking:** Integrated with VR hand tracking, allowing for full hand locomotion, including collision detection and object interaction.

Modifications and Updates:

1. **Serial Communication**:(*Figure 03)*
   - Integrated SerialManager in Unity for robust handling of input and output.
   - Improved latency for more responsive control.
2. **Finger Movement**:
   - Expanded from single to multi-finger tracking.
   - Optimized for real-time input from all five flex sensors.
   - Removed delays in Arduino code for real-time control.
3. **Object Interaction and Collision Detection**:
   - Added collision detection and interaction logic.
   - Introduced tags for interactable objects and colliders for each finger joint.
4. **Hardware Interaction Feedback**:
   - Integrated external feedback (buzzing, LED activation) for collision detection.
   - Adjusted control system for easy scaling with additional actuators.
5. **VR Demo and Full-Glove Testing**:
   - Updated demo to display real-time interaction.
   - Added new scenarios for comprehensive testing.

Modification Description:

- **Modification from Starter Code:(***Figure 04)*
  - The code expanded from a single-finger setup to handle five fingers, with sensor values adapted for flex sensors and smoothed for realistic joint movement. Integration with Oculus OVR rig and 3D hand models enabled lifelike virtual hand manipulation, and colliders ensure proper object interaction. Added collision detection and feedback, triggering signals like the buzzer when objects are touched. SerialManager was implemented for efficient serial communication, ensuring smooth data handling between hardware and Unity.

Note: Full code has been attached to the Design Package V2.