Designing an application architecture for developing a marker-less motion tracking system for humans to track position of joints in suits and fleet management of humans in suits

Mahedi Hasan Master of Artificial Intelligence, RMIT University
Email: s3903731@student.rmit.edu.au

Kiriti Rambhatla, Supervisor
Email: kiriti@metakosmos.com.au

1 Introduction

Since our spacesuits are equipped with an Inertial Measurement Unit (IMU) with Global Positioning System (GPS), it would be possible to track the positions and movements of their limbs and other body parts. This combination of IMU and GPS technology allows for a comprehensive tracking system that can provide both local and global position information. The purpose of this project is to track the movement of humans in suits without the need for external markers and managing a fleet of these individuals, possibly for applications like virtual reality, simulations, or other contexts where monitoring human movement is crucial.

2 Functional Requirements

Markerless Motion Tracking System: This involves monitoring the realtime movements of body joints of astronauts wearing our spacesuits. The project vision is to develop a system that can track the positions and movements of joints in real-time. Traditional motion tracking systems often require markers (such as reflective dots) placed on specific body joints or objects to capture movement accurately. Markerless systems, on the other hand, without the need for these external markers.

Fleet Management: It involves managing the movement and status of individuals wearing space suits. Develop mechanisms for coordinating and managing multiple individuals wearing motion-tracking suits simultaneously. Address data synchronization challenges to ensure that the motion tracking data from each individual is accurately represented in the overall system.

User Interface (UI): Design a user-friendly interface for operators or users to interact with the system. Include features for selecting and monitoring specific individuals within the group. Implement controls for starting, pausing, or adjusting tracking parameters for individual suits.

Data Security and Privacy: Implement measures to secure the motion tracking data, especially if it involves sensitive information related to astronauts. Address privacy concerns associated with the collection and storage of motion data from multiple individuals.

Application Areas: Identify and articulate potential application areas, such as astronaut training, biomechanical analysis, or simulation for space missions. Consider how the motion tracking system could contribute to enhancing the effectiveness and safety of activities involving individuals in space suits.

.

3 Solution Approach & Justification

Initial development will be conducted using raspberry pi 4B equipped with senseHAT IMU and GPS. Then further modifications of code and testing will be carried out to original equipment(suit computer)

3.1 Hardware Setup and Data Capture

3.1.1 Connectivity

Wire the IMU and GPS modules to the Raspberry Pi 4B, adhering to datasheets or manuals for correct GPIO pin connections.

3.1.2 Development Environment

Set up the Raspberry Pi's development environment, including the installation of necessary libraries and tools for the chosen programming language, such as Python.

3.2 Script Development (IMU)

Write a script to read data from the IMU, utilizing libraries like `smbus` or `gpiozero` for I2C or GPIO communication, respectively. Process IMU data to extract information on orientation and movement.

3.2.1 Script Development (GPS)

Utilize a library like `gpsd` to interface with the GPS module, extracting position and velocity data. Process GPS data to obtain relevant information such as latitude, longitude, altitude, and speed.

3.2.2 Sensor Fusion

Implement sensor fusion techniques, such as Kalman filtering, to combine data from the IMU and GPS for a more accurate representation of the Raspberry Pi's position and orientation.

3.3 Communication and Data Transmission

3.3.1 Communication Setup

Establish a communication method for transmitting sensor data. Preferably, use socket programming with the `socket` library, designating the laptop running MacOS as the server and Raspberry Pi devices as clients.

3.3.2 Data Exchange Protocol

Develop a robust protocol for data exchange between the Raspberry Pi and other devices in the system. Utilize socket programming functions for creating, binding, listening, accepting, and sending/receiving data over sockets.

3.4 Database Integration

3.4.1 Database Setup

Configure a database on the Raspberry Pi or an external server to store motion tracking data. Choose a suitable database management system such as SQLite, MySQL, or MongoDB.

3.5 Database Interaction

Write Python scripts to insert and retrieve data from the database, ensuring seamless interaction between the motion tracking system and the storage backend.

3.6 Real-Time Visualization

3.6.1 Visualization Component

Develop a visualization component capable of displaying real-time motion tracking data. Depending on project requirements, consider a simple graphical interface or a more complex 3D visualization. Prefer using Java and React.js for scalability, or alternatively, incorporate the `pygame` module for visualization.

3.7 Software Development for Suit Computers

3.8 Adaptation for Suit Computers

Apply similar techniques to develop software on suit computers for receiving, processing, and transmitting data from the IMU with GPS. Adjust code as per suit requirements, ensuring compatibility and performance.

3.8.1 Code Modifications

Make minor modifications to the existing codebase, considering the specific needs of suit computers. Conduct thorough testing to validate the adapted software's functionality.

3.9 Testing and Optimization

3.9.1 Testing Phase

Conduct extensive testing to ensure the reliability and accuracy of the motion tracking system. Verify data transmission, processing, and storage across multiple devices.

3.9.2 Optimization

Identify areas for optimization in both code and system architecture. Fine-tune algorithms, communication protocols, and database interactions for improved performance.

4 System Documentation and Planning

4.1 Build Developer Requirements

- **Requirements Document:** Create a comprehensive requirements document outlining guidelines and specifications for developers. Detail hardware specifications, software dependencies, data format standards, and coding conventions. Save this document in Microsoft Word format to serve as a reference for future development.

4.2 Architecture Mapping

- **Architecture Diagrams:** Utilize diagramming tools Visio to create detailed architectural diagrams. Illustrate the connections between hardware components, data flow, and system modules. Clearly define communication protocols and data exchange points in the architecture to guide future development.

4.3 User Interface Design

- **Design UI for Expected Architecture:** Leverage design tools like Figma or similar applications to create user interface prototypes based on the expected system architecture. Design UI components for both the Raspberry Pi interface and the suit computer application, ensuring a user-friendly and intuitive experience.

4.4 Future Development Guidelines

- **Documentation Repository:** Establish a centralized repository for documentation, housing essential documents like the requirements set, architecture diagrams, and UI design prototypes. This repository will serve as a knowledge hub for current and future developers.
- **Version Control:** Implement version control, using platforms like Git, to track changes in documentation and code. Ensure that the repository is well-maintained and accessible to developers for continuous collaboration.
- **Developer Onboarding:** Develop an onboarding process for new developers, providing them with access to the documentation repository and offering training sessions on the system's architecture, coding standards, and best practices.

4.5 Continuous Improvement

- **Feedback Mechanism:** Establish a feedback mechanism for developers to contribute insights and improvements to the documentation and architecture. Encourage regular reviews and updates to keep the documentation relevant.
- **Knowledge Transfer:** Conduct knowledge transfer sessions periodically to share insights, lessons learned, and updates with the development team. Foster an environment of continuous learning and improvement.