Robotics Report: Lab 2

Instituto Superior Técnico

Course: Robotics

Instructor: João Silva Sequeira, Alberto Vale

Shubam Paudel	**116771**
Jakob Schuck	**116789**
Simone Tumminello	**116576**
Matteo Albini	**116888**

18 December 2025

Contents

Chapter 1

Introduction

1.1 Motivation

Human augmentation systems exploit sensing and actuation technologies to assist users in performing simple physical tasks through intuitive interactions. Wearable inertial sensors, such as IMUs, provide a practical and low-cost solution for capturing human motion and enabling gesture-based control. When combined with embedded controllers and electromechanical actuators, these sensors allow the development of compact systems capable of translating human intent into mechanical action. This laboratory work is motivated by the need to understand the fundamental integration of inertial sensing, gesture recognition, and actuator control in the context of human–machine interaction.

1.2 Problem Statement

The goal of this laboratory assignment is to implement a gesture-controlled human augmentation device capable of producing a one–degree-of-freedom linear motion. An IMU attached to the user's arm is used to detect a predefined rotational gesture, which is processed by an Arduino-based controller. Upon detection, a stepper-motor–driven linear actuator is commanded to execute a controlled displacement, emulating a button-pushing action. The system must ensure reliable gesture detection, correct actuator motion, and safe operation within mechanical limits.

Task	**Authors that contributed**
Task 1	Simone, Shubam, Jakob, Matteo
Task 2	Simone, Matteo
Task 3	Simone, Matteo
Task 4	Everyone
Report	Everyone

Chapter 2

System Architecture

This chapter details the architectural design of the human augmentation device and describes how the physical components interface with the logical control structure. The system is designed as a closed-loop controller where a central microcontroller bridges the gap between high-speed sensor data acquisition and precise mechanical actuation.

2.1 Hardware and Electrical Integration

The core of the system is an Arduino Uno, which serves as the central processing unit responsible for interpreting human gestures and commanding the linear actuator. The sensory input is provided by a Grove 9-DoF Inertial Measurement Unit (IMU), specifically utilizing the onboard ICM20600 to capture 3-axis acceleration and angular velocity. This sensor communicates with the Arduino via the I2C bus, ensuring a digital interface that is robust against noise.
On the actuation side, a Nema 17 stepper motor drives the linear slider through a belt-pulley mechanism. The motor is powered by an external 12V supply and controlled by a DRV8825 driver module. The Arduino manages the motor's state by sending digital pulses to the driver; a high signal on the direction pin determines the rotation vector, while distinct pulses on the step pin advance the motor. To ensure operational safety and establish a spatial reference, the system includes two limit switches placed at the extremities of the slider's travel. These switches are wired to digital inputs with internal pull-up resistors enabled, allowing the system to detect physical end-stops.

2.2 Software Architecture and Control Flow

The firmware is implemented in C++ and operates on a continuous cycle of sensing, processing, and actuation. The execution begins with a critical initialization phase. Upon startup, the system performs a gyroscope calibration routine, sampling the sensor while stationary to calculate and subtract the zero-rate bias. This is immediately followed by a mechanical "homing" sequence. The controller drives the slider to the left limit switch and then to the right, measuring the total available travel in steps. The slider is then exactly placed in the center, establishing a calibrated coordinate system where the zero point corresponds to the middle of the rail.
Once initialized, the software enters the main control loop. In every iteration, the system reads raw data from the IMU and processes it through a complementary filter to estimate the user's arm angles in real-time. These angles are fed into the decision logic, which monitors for a rapid roll rotation (X-axis) to trigger the push gesture.

Chapter 3

IMU Data Acquisition and Gesture Detection

This chapter describes the signal processing techniques employed to transform raw inertial measurements into reliable control signals. The system relies on a robust data pipeline encapsulated within the IMUCompFilter and XGestureDetector classes, ensuring that mechanical actuation is driven by stable and drift-free orientation estimates.

3.1 Data Acquisition and Gyroscope Calibration

The fundamental input for the system is derived from the ICM20600 inertial sensor, which provides three-axis acceleration and angular velocity data via the I2C interface. However, consumer-grade MEMS gyroscopes inherently exhibit a zero-rate output error, known as bias, which registers rotation even when the device is stationary. If left uncorrected, this bias accumulates during integration, leading to significant angular drift. To mitigate this, the IMUCompFilter class executes a calibration routine during the system initialization. Over a duration of two seconds, the firmware captures a sequence of samples while the sensor is at rest, computing the average offset for each axis. These calculated bias values are stored and subsequently subtracted from every real-time gyroscope reading, ensuring that the integration process begins with a theoretically zero-error baseline.

3.2 Sensor Fusion via Complementary Filter

To estimate the user's arm orientation, the system must overcome the limitations of using either accelerometers or gyroscopes in isolation. While the accelerometer provides an absolute reference vector relative to gravity, it is highly susceptible to mechanical vibration and linear acceleration artifacts. Conversely, the gyroscope offers precise responsiveness to rapid motions but suffers from integration drift over time. The firmware addresses this by implementing a digital complementary filter within the updateAnglesXY method. The filter fuses the two data sources using a weighting factor α (set to 0.98), essentially trusting the gyroscope for short-term changes and the accelerometer for long-term stability. The update equation used for the estimation is:

$$\theta_{new} = \alpha \cdot (\theta_{old} + \dot{\theta}_{gyro} \cdot dt) + (1 - \alpha) \cdot \theta_{accel}$$

In this implementation, the term $\dot{\theta}_{gyro} \cdot dt$ represents the angular change measured by the gyroscope, while θ_{accel} is the angle derived from the accelerometer's projection of the gravity vector. This approach effectively filters out high-frequency accelerometer noise while correcting the low-frequency drift of the gyroscope, resulting in a suitable for gesture detection and monitoring.

3.3 Discrete Gesture Recognition

The "Push" command is a binary trigger event detected by the XGestureDetector class, which analyzes the filtered Roll angle (θ_x). The algorithm defines a valid gesture as a rotation that exceeds a specific deadband threshold of 90 degrees. To distinguish intentional gestures from accidental transient spikes, the system employs a time-domain validation logic. The signal must not only cross the threshold but also maintain that state for a minimum holding duration of 120 milliseconds. Once a valid trigger is registered, the detector enters a refractory state for 700 milliseconds, preventing duplicate triggers or "bouncing" while the user resets their arm position to the neutral state.

3.4 Application Logic: The Button Push Sequence

The final stage of the signal processing pipeline is the translation of the detected gesture into a physical interaction. This behavior is governed by a finite state machine that orchestrates the "Button Push" application (Task 4). When the XGestureDetector signals a valid trigger, the system interrupts the continuous control loop to execute a deterministic atomic sequence defined in the doButtonPush routine. It commands the linear actuator to advance a fixed distance (configurable as PRESS_STEPS) in the direction of the button. Upon reaching the target, the system enforces a holding period (defaulting to 1.5 seconds) to ensure a stable mechanical contact. Subsequently, the actuator reverses its trajectory to return exactly to the pre-push coordinates. Finally, the system performs a "soft reset" of the control objects—re-zeroing the center reference and clearing the gesture history—before seamlessly handing control back to the user, allowing for immediate resumption of normal operation.

Chapter 4

Digital Twin & Serial Communication (Processing)

A Processing-based application was implemented to monitor the embedded system in real time, providing (i) a 3D digital twin of the IMU board and (ii) live plots of the main orientation signals. The actuator is controlled only along one linear DoF; therefore, **pitch (Y)** is visualized for monitoring, while **yaw (Z)** is reconstructed on the laptop by integrating the gyroscope Z-rate (relative estimate, subject to drift).

4.1 Data Stream and Synchronization

The microcontroller streams one CSV row at 50 Hz (`STREAM_PERIOD_MS=20`) with the following header:

```
[language={},caption={CSV header (12 fields)},label={lst:dt_csv_header}]
t_ms,ax,ay,az,gx,gy,gz,angleX_deg,angleY_deg,angleY_smooth,gestureX,posSteps
```

To ensure deterministic startup, the application waits for the header before parsing data. While the header is missing (`gotHeader==false`), it periodically sends `'S'` (every 500 ms) to request streaming and force re-synchronization. This choice avoids misalignment when the serial port is opened mid-stream or when the MCU resets on connection.

4.2 Parsing and Robustness

The callback `serialEvent()` filters out non-data serial traffic (e.g., firmware logs such as lines starting with `[` or `Commands:`). A line is accepted as a sample only if:

- it contains at least 12 comma-separated tokens,
- token 0 is numeric (timestamp),
- parsing of the required fields succeeds (protected by `try/catch`).

The application tracks `anyLines`, `okLines`, and `badLines` to quantify parsing quality, and shows these counters in the HUD for quick debugging.

4.3 Yaw Estimation (Laptop-Side Integration)

Yaw is reconstructed by integrating the gyroscope Z-rate `gz` using the streamed timestamps:

$$\Delta t_k = \frac{t_{ms,k} - t_{ms,k-1}}{1000}, \qquad \psi_k = \psi_{k-1} + \omega_{z,k}\Delta t_k, \tag{4.1}$$

where $\omega_{z,k}$ corresponds to `gz_dps = float(tok[6])`. The estimate is wrapped to $[-180°, 180°]$ and can be reset with key `R`. This design is intentionally simple and sufficient for visualization, with the known limitation of drift (no absolute reference).

4.4 3D Digital Twin and UI

The 3D scene is rendered in `P3D`. Orientation is applied with the same rotation sequence used in the code: `rotateY(yaw)` → `rotateZ(pitch)` → `rotateX(roll)`. A small asymmetric marker is added to the board model to make the orientation visually unambiguous. To improve readability at 60 fps, the rendered angles are smoothed with `lerp` (`VIS_ALPHA=0.25`) without affecting the underlying data.
On the right panel, three live plots are displayed using ring buffers of length `N=420` (≈8.4 s at 50 Hz): roll (`angleX_deg`), pitch (`angleY_smooth`), and yaw (integrated). The HUD reports link status, counters, current angles, `gestureX`, and `posSteps`. When `gestureX==1`, a temporary overlay "GESTURE ACTIVATED" is shown for `GESTURE_MSG_MS`.

4.5 Resulting Interface

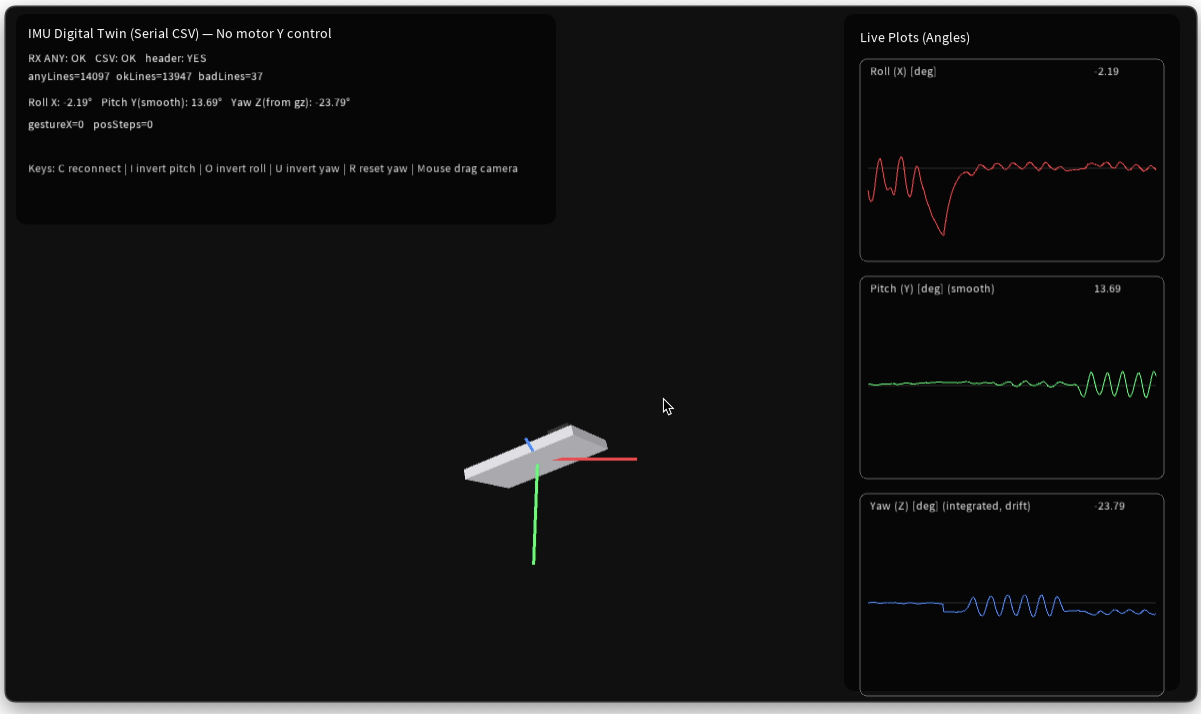

Figure 4.1: Processing digital twin UI with 3D orientation rendering and live plots (roll, pitch smoothed, yaw integrated).

Chapter 5

Experimental Results and Discussion

This chapter reports the experimental validation of the proposed human augmentation device, focusing on the two required demonstrations: (i) activation of the button-push motion through an X-axis (roll) gesture and (ii) real-time IMU tracking on the Processing digital twin with live plots. These experiments address Tasks 2–4 and provide evidence of the complete pipeline operation.

5.1 Experimental Setup

The IMU (Grove 9-DoF board) was attached to the user's forearm, while the linear actuator (slider + Nema17 + DRV8825) was worn on the opposite arm, with a rigid pointer aligned to a push-button target. Two limit switches were placed at the slider extremes to enforce safe travel and to support the homing routine (center reference). The Processing application was executed on the laptop to visualize the orientation state and to monitor the serial stream.

5.2 Video Demonstration

A video recording was produced as required by the assignment, showing one or more trials of the complete system:

- X-axis gesture activation (roll) and corresponding button-push motion;
- digital twin orientation tracking and live plots (roll, pitch(smooth), yaw-integrated).

The video is available at:

https://ulisboa-my.sharepoint.com/:v:/g/personal/ist1116576_tecnico_ulisboa_pt/IQDUA8SBHxZaSoPk7r-XxIboAcXx14r3nCe3euTzuyyeJAQ?e=yCdat8

5.3 Gesture Trigger on X Axis (Roll)

The chosen gesture is a discrete event derived from the estimated roll angle. The firmware implements a threshold-and-hold strategy:

- threshold (deadband): $|\theta_x| > 90°$;
- minimum hold time: 120 ms;
- refractory time: 700 ms (prevents re-triggering while returning to neutral).

This logic makes the trigger robust against brief spikes and reduces false positives during normal arm motion. In the recorded trials, the gesture activation is visible both on the mechanical motion (start of the push sequence) and on the Processing HUD through the `gestureX` event flag.

5.4 Digital Twin Tracking and Live Plots

The Processing tool receives the serial CSV stream at 50 Hz and renders the IMU board orientation in a 3D scene. Figure 4.1 shows the interface used during the experiments, including:

- a 3D digital twin (board model + world axes);
- a HUD with counters (`okLines/badLines`) and current angles;
- three live plots: roll (X), pitch (Y, smoothed), and yaw (Z, integrated).

Pitch is displayed only for monitoring, while yaw is obtained by integrating `gz` on the laptop side; as expected, yaw is a relative estimate and may drift over time without an absolute reference (magnetometer fusion). This behavior is visible in the yaw plot and is consistent with the adopted design choice.

5.5 Button Push Motion and Safety Behavior

Upon a valid gesture trigger, the actuator executes a deterministic push routine:

1. advance by a fixed number of steps toward the button;
2. maintain contact for a fixed hold time;
3. return to the initial position and re-center the internal step reference.

The end switches enforce safe operation by preventing motion beyond the mechanical limits. Additionally, the homing sequence establishes the midpoint reference before user interaction, enabling repeatable push actions starting from a known configuration.

5.6 Discussion and Limitations

The experiments confirm correct end-to-end behavior (gesture → trigger → push) and demonstrate real-time visualization and debugging capabilities through the digital twin. The main limitations are:

- **Yaw drift:** laptop-side integration accumulates bias over time; adding magnetometer-based fusion (or a full attitude filter) would provide an absolute heading reference.
- **Fixed gesture thresholds:** 90° and timing parameters are tuned for the current user/-mounting; an adaptive threshold or a calibration step could improve generalization.
- **Stepper motion profile:** step timing is constant; acceleration ramps could reduce vibration and improve perceived smoothness.

Chapter 6

Conclusion

This laboratory work presented a gesture-controlled human augmentation device that combines inertial sensing and a one-degree-of-freedom linear actuator to perform a button-push action. The system uses a wearable IMU to estimate arm orientation in real time (complementary filter with gyroscope bias calibration) and detects a discrete roll (X-axis) gesture via a threshold–hold–refractory strategy.

On the actuation side, a stepper-driven slider executes a deterministic push sequence and relies on limit switches and a homing procedure to guarantee safe operation and a repeatable center reference. A Processing-based tool was developed to stream the system state over serial CSV and to provide a practical monitoring interface, including a 3D digital twin and live plots.

Experimental trials (documented in the submitted video) demonstrate the correct end-to-end behavior: the X-axis gesture reliably triggers the mechanical push and the IMU state is tracked in real time on the laptop interface. Future improvements include absolute yaw estimation through magnetometer fusion, more adaptive gesture tuning, and smoother stepper motion profiles via acceleration ramps.