NAZARBAYEV UNIVERSITY

School of Engineering and Digital Sciences

CSCI502/702 – Hardware/Software Co-Design

Spring Semester 2026

Project Assignment #1

Getting Started with Raspberry Pi and IMU Sensor Interfacing

Team Members:

• Didar Rakhimbay

• Dias Akimbay

• Beibars Ybraiakhyn

• Alim Daniyarov

February 13, 2026

PART 1: RASPBERRY PI INTERFACING

Task 1: Getting Started with Raspberry Pi

The Raspberry Pi 3 Model B is a single-board computer featuring a 1.2GHz quad-core ARM Cortex-A53 processor, 1GB RAM, 802.11n Wi-Fi, Bluetooth 4.1, 4 USB ports, HDMI, and 40 GPIO pins. It runs Linux-based operating systems and serves as an ideal platform for embedded systems development and hardware interfacing projects.

1.1 Safety Rules and Best Practices

Before working with the Raspberry Pi, we reviewed critical safety guidelines:

• **Proper Shutdown:** Never disconnect power by pulling the USB cable. Always use `sudo shutdown -h now` or the OS shutdown option. Improper shutdown can corrupt the SD card filesystem and damage the OS image.
• **Metal Surfaces:** Never place a powered board on conductive surfaces to prevent short circuits.
• **Connection Sequence:** Always connect HDMI, keyboard, and mouse before applying power to avoid damaging ports.
• **Power Supply:** Use a quality 5V, 2.5A power adapter. Insufficient power causes instability and random crashes.
• **ESD Protection:** Handle the board by edges to prevent electrostatic discharge damage to components.

1.2 Operating System Installation and Configuration

We used the Raspberry Pi Imager software to write Raspberry Pi OS (32-bit) to a microSD card. This Debian-based distribution is optimized for Raspberry Pi hardware and includes pre-configured drivers and development tools.

Image Customization Settings:

Setting	Value	Purpose
Username	pi	Default user account
Password	raspberry	Authentication credential
Wi-Fi SSID	ROB_322	Lab network (2.4GHz)
Wi-Fi Password	robotics7block	Network authentication
SSH	Enabled	Remote terminal access
Hostname	raspberrypi	Network identifier

1.3 System Update and Package Management

After booting the Raspberry Pi (indicated by green LED activity and red power LED), we updated the system using APT (Advanced Package Tool), Debian's package manager:

```
sudo apt update
```

Updates the local package index from repositories. This downloads metadata about available packages and their versions but doesn't install anything. Essential before upgrading to ensure latest versions are fetched.

```
sudo apt upgrade
```

Downloads and installs newer versions of all installed packages. This process may take 10-15 minutes depending on network speed and number of updates. Critical for security patches and bug fixes.

1.4 raspi-config Configuration Tool

The `raspi-config` utility provides a text-based interface for system configuration without manually editing config files. Executed with: `sudo raspi-config`

Critical configurations performed:

1. Interface Options → I2C: Enables the I^2C (Inter-Integrated Circuit) kernel driver. This loads the necessary modules (`i2c-bcm2835` and `i2c-dev`) and creates device files in `/dev/i2c-1` for userspace access. Without this, I^2C devices cannot be detected.

2. Interface Options → SSH: Starts the OpenSSH server daemon (`sshd`) and enables it to auto-start on boot. SSH uses port 22 and provides encrypted remote terminal access.

3. Advanced Options → Expand Filesystem: Resizes the root partition to utilize the entire SD card capacity. By default, the OS image is only ~4GB; this expands it to the full card size (8GB, 16GB, 32GB, etc.).

1.5 SSH Remote Access Setup

SSH (Secure Shell) is a cryptographic protocol for secure remote command-line access. It replaces insecure protocols like Telnet and rlogin by encrypting all communications including passwords.

Finding the Raspberry Pi IP Address:

Since the Pi uses DHCP (Dynamic Host Configuration Protocol), it receives an IP address automatically from the router. We found the IP using: `hostname -I` which returned `192.168.0.102`

Establishing SSH Connection:

From a Ubuntu PC on the same network (ROB_322), we connected using:

```
ssh pi@192.168.0.102
```

Format: `ssh username@ip_address`. On first connection, SSH displays the server's RSA fingerprint for verification (prevents man-in-the-middle attacks). After accepting and entering password 'raspberry', we gained shell access.

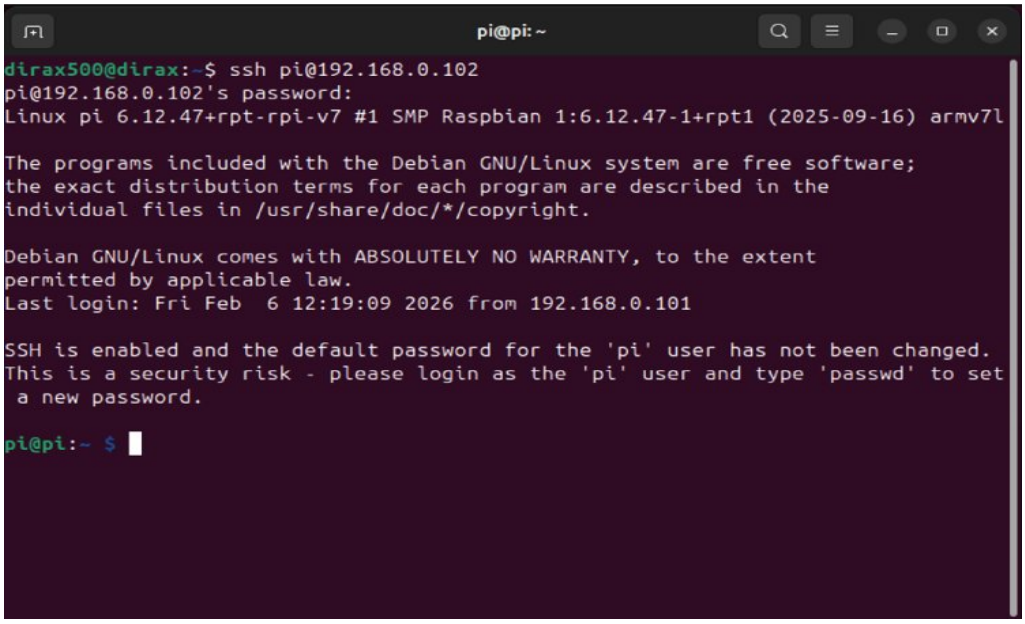

Figure 1: SSH connection established to pi@192.168.0.102. Shows Linux kernel 6.12.47, Raspbian version, and authentication success. The prompt 'pi@pi:~$' indicates we're in the home directory with user privileges.

Why SSH instead of direct connection? SSH enables headless operation (no monitor/keyboard needed), allows file transfer via SCP/SFTP, supports tunneling for remote GUI apps, and permits multiple simultaneous connections. For development, it's more efficient than constantly switching monitor cables.

Task 2: Git Version Control Practice

Git is a distributed version control system used to track code changes, collaborate with teams, and maintain project history. GitHub Classroom provides automated assignment distribution and submission for educational settings.

Git Workflow:

1. `git clone <repo_url>` - Downloads the assignment repository
2. `git add <files>` - Stages files for commit
3. `git commit -m "message"` - Creates a commit snapshot with description
4. `git push` - Uploads commits to GitHub

We uploaded our Hello World C++ program and this report PDF to: `https://classroom.github.com/a/beOzdkCX`

PART 2: IMU SENSOR INTERFACING

Task 3: Setting Up IMU Sensor

What is an IMU? An Inertial Measurement Unit combines multiple sensors to measure motion and orientation. Our Pololu MinIMU-9 v5 includes:

• **LSM6DS33** - 3-axis gyroscope (measures angular velocity in degrees/second) and 3-axis accelerometer (measures linear acceleration in g-forces)
• **LIS3MDL** - 3-axis magnetometer (measures magnetic field in gauss, used as a compass)

Applications: Drones (flight stabilization), robots (balance control), smartphones (screen rotation), VR headsets (head tracking), wearables (step counting, gesture recognition).

2.1 I^2C Communication Protocol

I^2C (Inter-Integrated Circuit) is a synchronous, multi-master serial bus invented by Philips. It uses only 2 wires: SDA (Serial Data) and SCL (Serial Clock), plus ground. Multiple devices share the same bus, each identified by a unique 7-bit address.

I^2C Advantages:

• Minimal wiring (2 wires vs 8+ for parallel)
• Multiple devices on one bus (up to 127)
• Built-in addressing (no chip select pins needed)
• Acknowledgment mechanism (error detection)

Speed modes: Standard (100 kHz), Fast (400 kHz), Fast-plus (1 MHz). Our configuration uses Fast mode.

2.2 Hardware Connections

We connected the MinIMU-9 v5 to the Raspberry Pi GPIO header following I^2C specifications:

IMU Pin	RPi Pin	GPIO	Function	Notes
SCL	5	GPIO 3	I2C Clock	Synchronized by master (RPi)
SDA	3	GPIO 2	I2C Data	Bidirectional data line
GND	6	Ground	Common ground	Essential for signal reference
VDD	1	3.3V	Power supply	Regulated 3.3V output
VIN	Not connected	-	5V input (unused)	See explanation below

Why VIN is NOT connected:

The MinIMU-9 v5 has two power inputs: VIN (5V) and VDD (3.3V). The VIN pin feeds a voltage regulator that outputs 3.3V to the sensors. However, Raspberry Pi's I^2C GPIO pins operate at 3.3V logic levels - they are NOT 5V tolerant. If we powered the sensor with 5V via VIN while connecting I^2C to RPi's 3.3V pins, voltage mismatch could damage the GPIO pins. By using VDD (3.3V) directly, both

power and logic levels match, ensuring safe operation.

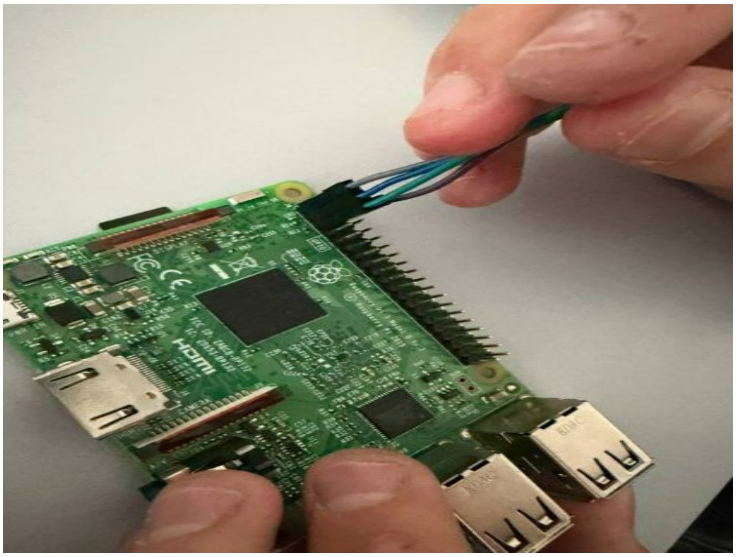

Figure 2: Physical wiring of IMU sensor to Raspberry Pi GPIO pins. Green/blue wires connect to I^2C pins (SCL/SDA), power supplied via 3.3V VDD pin.

2.3 I²C Detection and Verification

After physical connections, we verified I²C functionality using command-line tools from the `i2c-tools` package.

```
i2cdetect -y 1
```

Scans the I²C bus #1 for connected devices. The `-y` flag auto-confirms (skips interactive prompt). Raspberry Pi 3 has two I²C buses: bus 0 (reserved for HATs) and bus 1 (GPIO pins 3 & 5). We use bus 1.

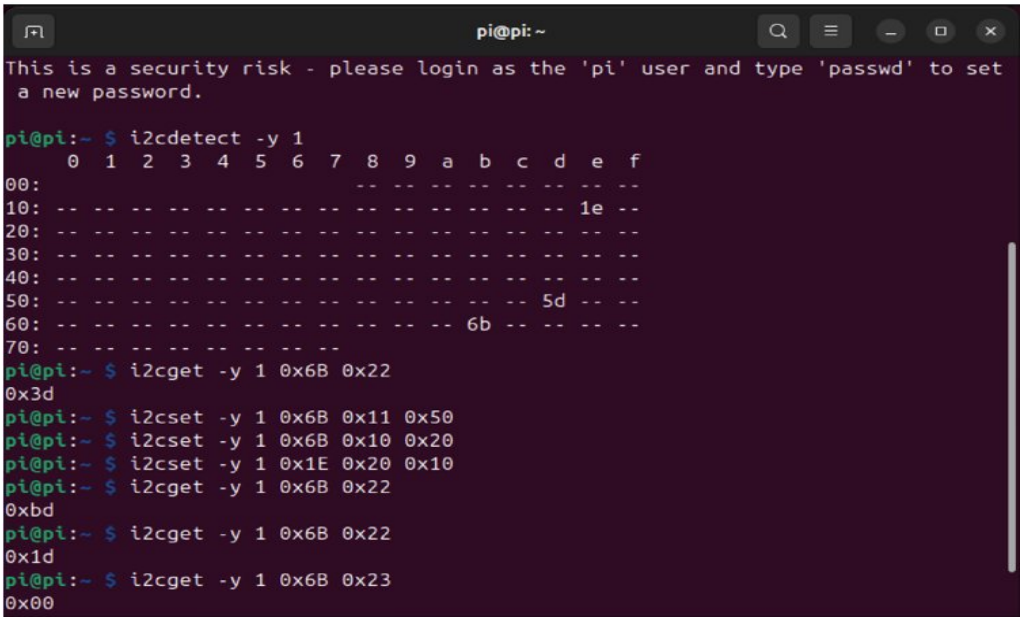

Figure 3: i2cdetect output showing two devices: 0x1E (LIS3MDL magnetometer) and 0x6B (LSM6DS33 gyro/accel). '--' indicates no device at that address.

Device Addresses Detected:

- `0x6B` - LSM6DS33 (gyroscope + accelerometer)
- `0x1E` - LIS3MDL (magnetometer)

These 7-bit addresses are hardcoded in the sensor chips. If detection failed, we'd check: wiring, I²C enabled in raspi-config, power supply, or damaged sensors.

Why value '1' in i2c commands?

Raspberry Pi 3 has two I²C controllers: i2c-0 and i2c-1. The GPIO header pins (physical pins 3 and 5) are connected to i2c-1. That's why all our commands use bus number 1: `i2cdetect -y 1`, `i2cget -y 1 ...`, `i2cset -y 1 ...`

2.4 Sensor Configuration Registers

IMU sensors power on in a disabled state to conserve energy. We must write specific control words to configuration registers to activate sensors and set parameters (data rate, sensitivity range, operating mode).

Configuration Register Table:

Sensor	Register	Address	Control Word	Configuration Details
LSM6DS33 Accelerometer	CTRL1_XL	0x10	0x20 (00100000)	ODR=26Hz, FS=±2g, BW auto, All axes enabled
LSM6DS33 Gyroscope	CTRL2_G	0x11	0x50 (01010000)	ODR=26Hz, FS=±500dps, All axes enabled
LIS3MDL Magnetometer	CTRL_REG1	0x20	0x10 (00010000)	TEMP disabled, XY ultra-perf, DO=0.625Hz (overridden by REG3)
LIS3MDL Magnetometer	CTRL_REG2	0x21	0x00 (00000000)	FS=±4 gauss, Normal mode, No reboot/reset

Detailed Control Word Breakdown:

CTRL1_XL = 0x20 (Accelerometer):

Binary: `0010 0000`
• Bits [7:4] = 0010 → Output Data Rate = 26 Hz (measurements per second)
• Bits [3:2] = 00 → Full Scale = ±2g (sensitivity range, 1g = 9.81 m/s^2)
• Bits [1:0] = 00 → Bandwidth filter auto-selected based on ODR

Why 26Hz? Balance between responsiveness and power consumption. Higher rates (52Hz, 104Hz) drain more power.

CTRL2_G = 0x50 (Gyroscope):

Binary: `0101 0000`
• Bits [7:4] = 0101 → Output Data Rate = 26 Hz
• Bits [3:2] = 01 → Full Scale = ±500 degrees/second
• Bits [1:0] = 00 → Default (unused bits)

Why ±500 dps? Suitable for moderate rotation speeds. Slower movements use ±245 dps, faster (drones) use ±2000 dps.

CTRL_REG1 = 0x10 (Magnetometer):

Binary: `0001 0000`
• Bit [7] = 0 → Temperature sensor disabled
• Bits [6:5] = 00 → XY-axis ultra-high performance mode
• Bits [4:2] = 010 → Partial ODR setting (combined with CTRL_REG3 for 5Hz)
• Bits [1:0] = 00 → Fast ODR disabled

CTRL_REG2 = 0x00 (Magnetometer):

Binary: `0000 0000`
• Bits [6:5] = 00 → Full Scale = ±4 gauss (Earth's magnetic field is ~0.5 gauss)
• Bit [3] = 0 → No reboot
• Bit [2] = 0 → No soft reset

Why ±4 gauss? Earth's field is weak; ±4 gauss provides good resolution without saturation.

2.5 Writing Configuration to Sensors

We used `i2cset` to write control words to registers:

```
i2cset -y 1 0x6B 0x11 0x50 # Gyroscope

i2cset -y 1 0x6B 0x10 0x20 # Accelerometer

i2cset -y 1 0x1E 0x20 0x10 # Magnetometer REG1

i2cset -y 1 0x1E 0x21 0x00 # Magnetometer REG2
```

Command format: `i2cset -y [bus] [device_addr] [register_addr] [value]`
The `-y` flag skips confirmation prompts (auto-yes).

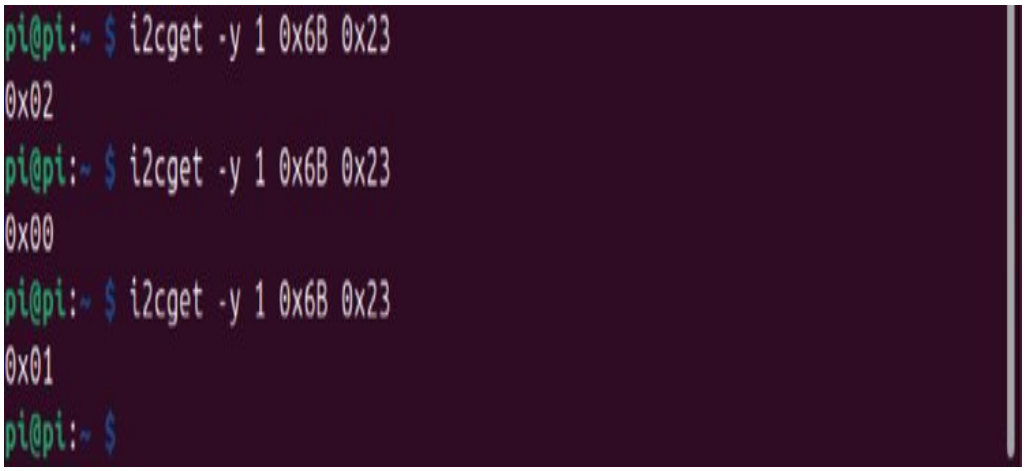

Figure 4: i2cset commands configuring sensors, followed by i2cget reading gyro data registers showing changing values (0x3d, 0xbd, 0x1d, 0x00, 0x02, 0x01) as sensor moves.

2.6 Reading Sensor Data

After configuration, sensors actively measure and store data in output registers. Each axis (X, Y, Z) uses two 8-bit registers for 16-bit resolution:

Gyroscope X-axis data:

- `OUTX_L_G (0x22)` - Low byte
- `OUTX_H_G (0x23)` - High byte

Combined they form a 16-bit signed integer representing angular velocity. Similar registers exist for Y (0x24/0x25) and Z (0x26/0x27) axes.

```
i2cget -y 1 0x6B 0x22 # Read low byte

i2cget -y 1 0x6B 0x23 # Read high byte
```

Example readings from Figure 4: When reading register 0x22 multiple times while rotating the sensor, values changed (0x3d → 0xbd → 0x1d → 0x00), confirming the gyroscope is actively measuring rotation.

Converting raw values to physical units: The raw 16-bit value must be multiplied by sensitivity factor from the datasheet. For ±500 dps range: sensitivity = 17.50 mdps/LSB (milli-degrees per second per least significant bit). Example: if raw value = 1000, actual angular velocity = 1000 × 0.0175 = 17.5 degrees/second.

2.7 Data Register Summary

Sensor	Measurement	Register	Address	Format
LSM6DS33	Gyro X-axis	OUTX_L_G, OUTX_H_G	0x22, 0x23	16-bit signed
LSM6DS33	Gyro Y-axis	OUTY_L_G, OUTY_H_G	0x24, 0x25	16-bit signed
LSM6DS33	Gyro Z-axis	OUTZ_L_G, OUTZ_H_G	0x26, 0x27	16-bit signed
LSM6DS33	Accel X-axis	OUTX_L_XL, OUTX_H_XL	0x28, 0x29	16-bit signed
LSM6DS33	Accel Y-axis	OUTY_L_XL, OUTY_H_XL	0x2A, 0x2B	16-bit signed
LSM6DS33	Accel Z-axis	OUTZ_L_XL, OUTZ_H_XL	0x2C, 0x2D	16-bit signed
LIS3MDL	Mag X-axis	OUT_X_L, OUT_X_H	0x28, 0x29	16-bit signed
LIS3MDL	Mag Y-axis	OUT_Y_L, OUT_Y_H	0x2A, 0x2B	16-bit signed
LIS3MDL	Mag Z-axis	OUT_Z_L, OUT_Z_H	0x2C, 0x2D	16-bit signed

CONCLUSION

This project successfully demonstrated complete Raspberry Pi setup and I^2C sensor interfacing. We gained practical experience with Linux system administration (APT updates, raspi-config), network protocols (SSH, DHCP), version control (Git), hardware interfacing (GPIO, I^2C), and low-level sensor configuration (register programming, binary control words).

Key Technical Skills Acquired:
• Embedded Linux system configuration and command-line proficiency
• Understanding I^2C protocol timing, addressing, and multi-device bus sharing
• Reading and interpreting sensor datasheets to determine register addresses and bit fields
• Binary/hexadecimal conversion and bit manipulation for control word construction
• Hardware debugging skills (checking connections, verifying power, detecting I^2C devices)
• Safe practices for embedded systems (proper shutdown, ESD protection, voltage level matching)
• Version control workflow for collaborative software development

The sensor configuration process highlighted the importance of datasheet comprehension. Each control word's bits directly map to hardware settings (ODR, sensitivity, operating modes). Incorrect values could disable sensors, saturate readings, or waste power. This low-level understanding is essential for embedded systems engineering where software directly controls hardware behavior.

Individual Contributions:

• **Didar Rakhimbay:** Raspberry Pi OS installation, SSH configuration, system updates, and network setup.

• **Dias Akimbay:** Physical GPIO connections, I^2C wiring verification, and i2cdetect testing.

• **Beibars Ybraiakhyn:** Sensor register configuration, control word calculation, and data reading validation.

• **Alim Daniyarov:** GitHub repository management, documentation, and final report compilation.

REFERENCES

1. Molloy, D. (2016). *Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux*. Wiley.

2. Raspberry Pi Foundation. (2026). *Raspberry Pi Documentation*. https://www.raspberrypi.com/documentation/

3. STMicroelectronics. (2019). *LSM6DS33 Datasheet: iNEMO inertial module: always-on 3D accelerometer and 3D gyroscope*.

4. STMicroelectronics. (2015). *LIS3MDL Datasheet: Digital output magnetic sensor: ultra-low-power, high-performance 3-axis magnetometer*.

5. Pololu Corporation. (2024). *MinIMU-9 v5 Gyro, Accelerometer, Compass, and Altimeter*. https://www.pololu.com/product/2739

6. GitHub, Inc. (2026). *GitHub Documentation*. https://docs.github.com/

7. NXP Semiconductors. (2014). *UM10204: I2C-bus specification and user manual*.