To avoid spurious surface heights and accurately capture topography, a 20x objective magnification (3.5 mm x 3.5 mm field of view) was found to be best. High tearing energy cutting conditions produce- periodic surfaces while lower tearing energy creates smooth surfaces. Surface scan results of cut surfaces with smooth and periodic features under 20x lens objective along with their 1D Line-scans are shown in Figure 2-2, clearly depicting the difference in the achieved topography.

A clear rectangular object on a black surface

Description automatically generatedA graph showing a number of data

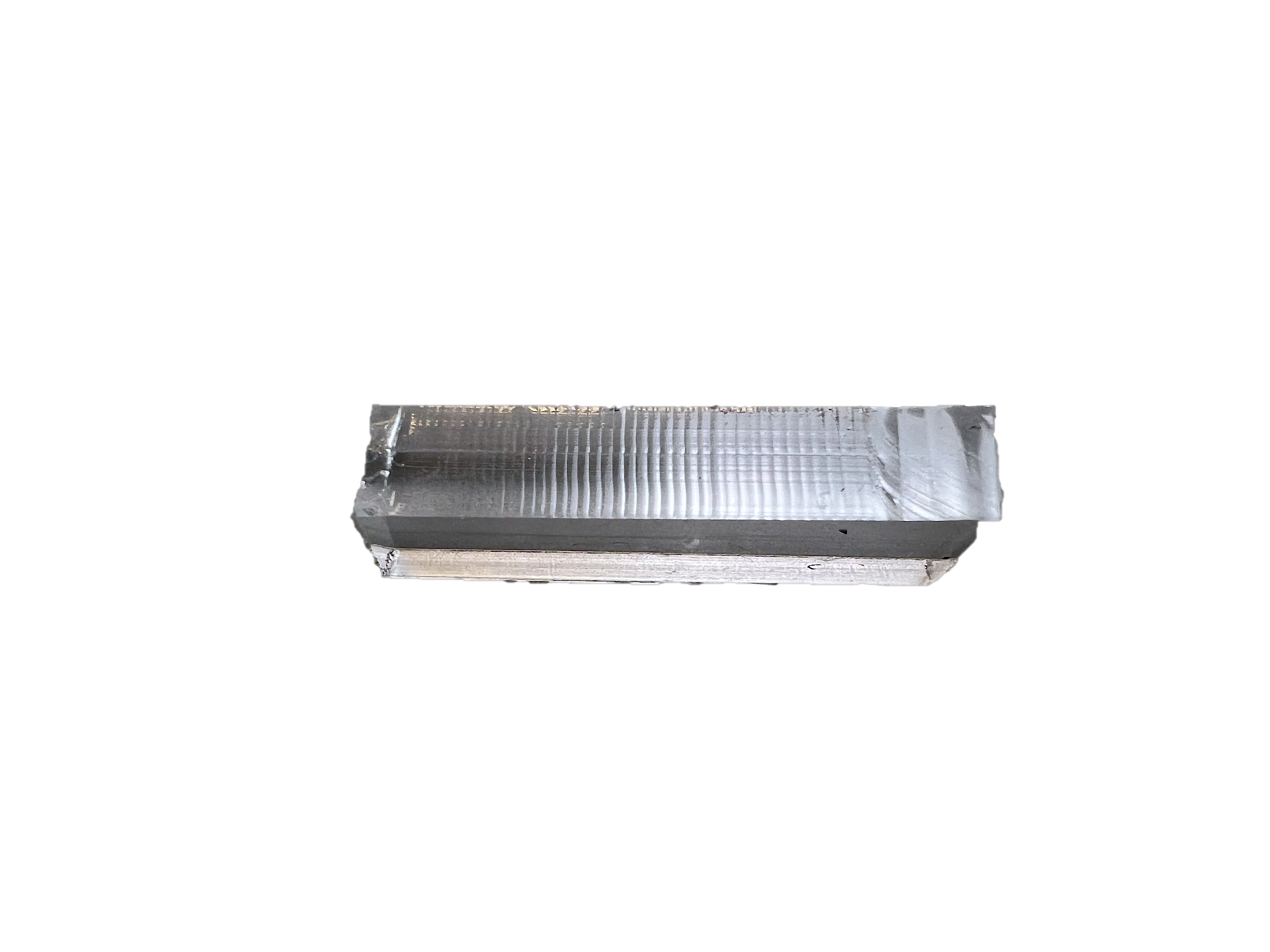
Description automatically generated

1. **Smooth surface**

A blue line graph with black text

AI-generated content may be incorrect.

**(b) Periodic surface**



A close-up of a graph

AI-generated content may be incorrect.

A graph of a graph

AI-generated content may be incorrect.

**(c) Rough surface**

A red and blue graph

Description automatically generated

A graph of a graph

AI-generated content may be incorrect.

Figure 2-2. Surface topography of characteristic silicone, created by custom Y-shpaed cutting method (a) SMOOTH SURFACE (b) PERIODIC SURFACE (C) ROUGH SURFACE

INSTRUMENTATION, CONTROL AND PROGRAMMING

The experimental study of tribology involves gathering the data of interacting surfacial properties with utmost accuracy under precise control of the experimental matrix. Microtribometers are widely used for this purpose, providing controlled motion along different axes. Their functionality can be enhanced by integrating programming solutions that allow real-time control and simultaneous data collection. However, different experiments require specific stage controls, hence the development of a standardized programming infrastructure simplifies the experimental process, ensuring consistency and ease of use.

This chapter focuses on the implementation of such an infrastructure using the Python programming language. Python’s extensive libraries, adaptability and automation capabilities make it a preferred choice for controlling microtribometers, enabling real-time data acquisition, processing, and visualization. Establishing this infrastructure allows for the proper execution of tribological experiments, offering a reliable methodology for analyzing frictional behavior across various materials.

Data structure and control

The microtribometer's functionality is achieved by interfacing with its hardware components through Python-based control packages. The key elements of this system include:

* **Linear Screw-Driven Stage:** VT-80 screw-driven translation stage by PI-micos (Physik instrumente), operated via a C-863 Mercury DC motor to facilitate precise movement.
* **Cantilever Beam Flexures:** Used to measure surface asperities through deflection.
* **Capacitive Sensors:** Responsible for force data acquisition, providing voltage signals that are later converted into load values.
* **DAQ System:** Communicates sensor data to the main control unit.

To integrate these components, the following software packages are utilized:

* **PIPython Package:** Enables direct communication with the translation stage’s controller, allowing for precise motion control.
* **National Instruments (NI) DAQ Package:** Facilitates real-time data collection from the capacitive sensors.
* **Tkinter Package:** Used to enable a user-interactive setup and provide live data visualization capabilities.

A diagram of a function

Description automatically generated

Figure 3-1. Pseudo code: Logical representation of Python code to control the movemenent of micro tribometer and flow of data.

Psuedo Code

Pseudo code is a high-level description of a program's logic that outlines its structure and data flow in a readable format instead of a detailed syntax

A structured pseudo code representation of the program used to control the microtribometer is shown in Figure 3-1.

A sophisticated control architecture is employed, integrating high-precision stage control with microcontroller-based device management. This approach enables precise mechanical movements while maintaining flexibility in experimental configurations. The first step in the setup process involves initializing the controller and stage using the built-in functions provided by Physik Instrumente’s PIPython library. This initialization establishes a stable connection between the software and hardware components, ensuring smooth operation during experimentation. By leveraging these libraries, researchers can customize the control parameters to suit specific experimental needs.

A step by step approach is shown in the sub-sequent sections with an integrated framework making it a robust solution for control, automation, customization and adaptation to diverse experimental needs.

The core logic of the code is broadly divided into four phases:

**Part 1: Initialization phase**

* Setting up of GUI framework, import required packages and libraries (e.g. Tkinter, matplotlib, numpy, queue, and threading).
* Initiating DAQ, and running of startup code for scaling up the stage
* Initializing stage, motor controller and associated components

**Part 2: Setup phase**

* User inputs test parameters:
  + User identity, sample, probe and experiment information.
  + User entries (for setting up the experimental matrix- velocity, stroke length, number of strokes, initial stage position and cantilever beam information),
* Labels, frames and logos.
* Database initialization, creation of buttons, variables and empty graphs.
* The system validates all inputs before proceeding.
* Performs initial calibration of force sensors, and provides structure to sub-functions, delegating tasks and processing their outputs.
* Acts as the central controller thread for sending and receiving data.

**Part 3: Test execution logic**

* Main test loop operates in a controlled sequence:
  + Move stage to starting position
  + Begin data collection (Position tracking, force measurements and time data collection)
  + Execute strokes (forward and reverse stroke) and record measurements.
* Basic functions before executing the main test loop:
  + Validation whether the user entries are in range (e.g. within stage limit of 0 to 150 mm, speed- 0.1 to 20 mm/s and right data base allocation) and all the functions that handle actions while buttons are triggered.

**Part 4: Data processing logic**

* Real-time data analysis:
  + Converts raw voltage readings to force measurements (Normal and friction loads).
  + Statistical calculations for friction coefficient for each cycle, data separation and processing.
* Data storage:
  + Saves detailed measurements and calculates the average of each cycle data.
  + Plots the required data simultaneously on the GUI-graphical user interface.

This structured approach ensures modularity, ease of debugging, and clear functionality, allowing researchers to replicate or modify the setup for further tribological investigations.

The acquired data is then distributed and saved into two separate files. One contains the whole data (all the data from the start of the stage motion to its end, including the raw data of initial experimental setup) this helps in debugging if incurred any error during or after the experiment. The second file contains the middle 60 % set of the data of each cycle (single forward and backward stroke of stage) and this data is used for all further calculations as shown in Figure 3-2.

A diagram of data flow

Description automatically generated

Figure 3-2.Data flow, separation of data and basic background calculations

Development of GUI (Graphical User Interface)

A Python-based Graphical User Interface (GUI) was developed using the Tkinter package to facilitate live data visualization and user interaction. The GUI enables researchers to input experimental parameters, monitor real-time data acquisition, and validate results instantaneously. It provides an intuitive interface for controlling the microtribometer and ensures ease of use for conducting tribological experiments.

**User Input Window**

Upon running the program, a pop-up window appears, prompting the user to input essential experimental parameters, as shown in Figure 3-3. This functionality is implemented in Part-1 of the program (as discussed in previous section), which creates the root window and aligns required buttons and input fields in a structured grid format. The interface includes user identity, experimental motion parameters (such as stage location, velocity, stroke length, cycle count, and database location), and load adjustments.

Once all mandatory inputs are provided, pressing the ‘Start Test’ button initiates the experiment through main function (Part-3 of the program). If necessary, an Emergency Stop button allows immediate halting of both stage motion and data acquisition. The required normal load is then manually applied at the setup but can be automated using a load cell as a future improvisation.

A screenshot of a computer

Description automatically generated

Figure 3-3. Graphical User Interface (GUI), user inputs, buttons and database

**Live Data Visualization**

Real-time data monitoring can be achieved through dynamic graphs, as shown in Figure 3-4. These plots provide instataneous validation of the experimental process:

* **Stage Position vs. Time:** Ensures correct stage movement and verifies data acquisition sufficiency through stroke lengths and equal time per every cycle. It also represents the whole data (blue) and separated data (red).
* **Cycle vs. µ (Coefficient of Friction):** Displays the final output per cycle, offering real-time insights into frictional behavior. The calculations depend on the number of acquired datapoints, which is governed by the user-defined parameters.

A screenshot of a computer

Description automatically generated

Figure 3-4. Live data representation with logo and dynamic console for simultaneous monitoring.

The GUI offers flexibility in visualization, similarly, additional data plots can be incorporated or even console/debug sections as needed. These interactive features significantly enhance the efficiency and reliability of tribological measurements by providing instant feedback on data validity and experiment progression.

A screenshot of a computer

AI-generated content may be incorrect.