Electronics Club

IIT KANPUR

Flexe Glove

Summer Project 2025

Mentors

Rachit Agarwal
Yashwi Agarwal
Kushagra Shukla
Shreyas Gupta

Mentees

Name	Roll No.
Antriksh Singhal	240150
Anushka	240160
Arnav Agarwal	240187
B Mukund Advaith	240253
Charitra Jain	240300
Himanshu Gupta	240451
Mohit Kumar Yadav	240658
Naitik Lalchandani	240675
Nishant Kumar	240706
Prakhar	240763
Tarun Goyal	241096

August 11, 2025

Contents

1 Project Timeline

Week	Tasks Completed
Week 1	Basics of Electronics, Introduction to Ubuntu, Designing Hand Model in Fusion 360
Week 2	Introduction to ROS2, Creating Publisher and Subscriber Nodes
Week 3	Converting Hand Model to URDF, Using Joint State Publisher to Control Joints, Simulating Joints in RViz
Week 4	Learning about IMUs, Controlling Hand Movement using IMU Data
Week 5	Introduction to Pure Data (PD), Basics of Machine Learning
Week 6	Creating MIDI Control with Flex Sensors in PD, Training ML Model for Sign Language Recognition
Week 7	Hardware Implementation using Zero PCB, Arduino Nano, Flex Sensors, and IMU
Week 8	Documentation, Project Testing, and Evaluation

2 Objective

- To make a 3D fully controlled hand model that can translate voltage signals to hand movements.
- Use Flexe Glove to detect gestures and translate sign language to English.
- Use audio mixing software to add effects to a song in real time using signals from Flexe Glove.

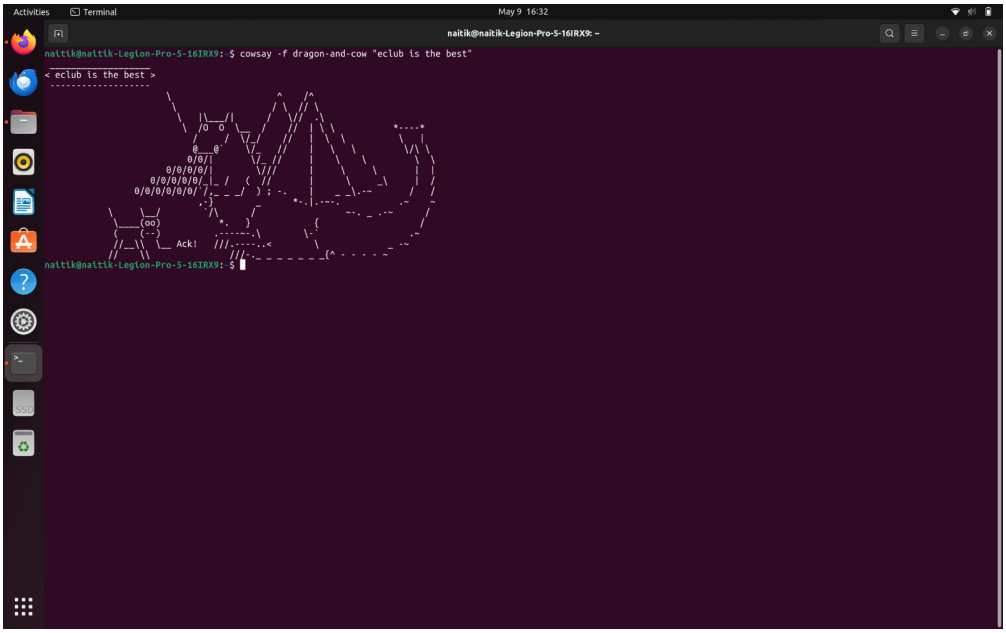
3 Introduction to Linux and Ubuntu Setup

3.1 Assignment 0

We installed packages like `cowsay`, `cmatrix`, `nyancat`, etc., to get started with Linux commands.

3.1.1 Installing `cowsay`

```
$ sudo apt install cowsay
$ cowsay "hello world"
$ cowsay -l
$ cowsay -f dragon "hello world"
```

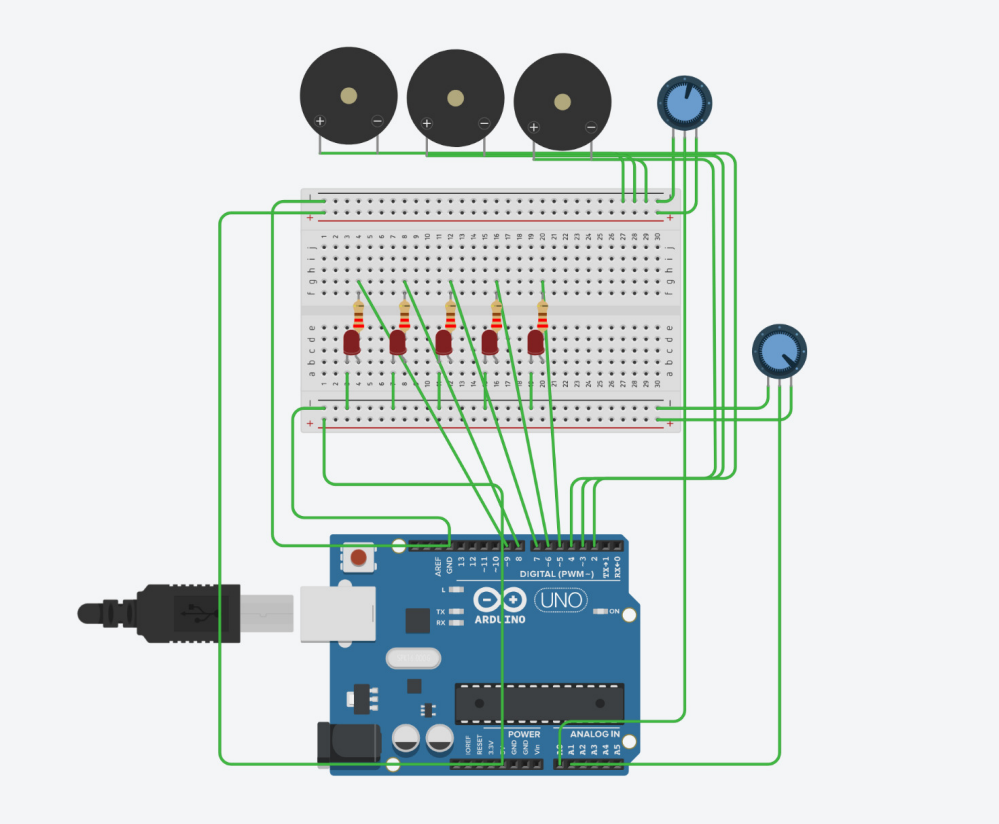


3.2 Second Assignment 0: TinkerCAD Simulation

- Pot1 controls 3 buzzers:
  - 0–341: Buzzer 1
  - 342–682: Buzzer 2
  - 683–1023: Buzzer 3
- Pot2 controls LED delay

Connections:

- Pot1 → A0, Pot2 → A1
- Buzzers: D2, D3, D4
- LEDs: D5 to D9 (through 220Ω resistors)



3.2.1 Arduino Code

```
int leds[] = {5, 6, 7, 8, 9};
int numLeds = 5;
float t;

void setup() {
  pinMode(4, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  Serial.begin(9600);
  for (int i = 0; i < numLeds; i++) {
    pinMode(leds[i], OUTPUT);
  }
}

void loop() {
  int POT1 = analogRead(A0);
  float POT2 = analogRead(A1);
  t = (POT2 + 113.666667)/1.1366667;
  Serial.println(t);

  if (POT1 <= 341) {
    digitalWrite(4, HIGH); digitalWrite(2, LOW); digitalWrite(3, LOW);
  }
  else if (POT1 <= 682) {
```

```
    digitalWrite(4, LOW); digitalWrite(2, HIGH); digitalWrite(3, LOW);
  }
  else {
    digitalWrite(4, LOW); digitalWrite(2, LOW); digitalWrite(3, HIGH);
  }

  for (int i = 0; i < numLeds; i++) {
    digitalWrite(leds[i], HIGH);
    delay(t);
    digitalWrite(leds[i], LOW);
  }
}
```

4 Introduction to Fusion 360

4.1 Types of Joints

1. Rigid: 0 DOF
2. Revolute: 1 rotational DOF
3. Slider: 1 translational DOF
4. Cylindrical: 1T + 1R DOF
5. Pin-slot: 1T + 1R DOF (different axes)
6. Planar: 2T + 1R DOF
7. Ball: 3 rotational DOF

4.2 Assignment 1: Car Model

Created a car model in Fusion 360. Used GrabCAD tyres. Applied revolute joints for wheel movement.

4.3 ACDC4ROBOT Add-in

ACDC4ROBOT converts Fusion 360 models to URDF, SDFormat, MJCF for use in robotics simulation like ROS.

5 Robot Operating System (ROS2)

5.1 Overview

ROS2 offers a distributed system using nodes for various robotic functions. It makes robotics software modular and scalable.

5.2 ROS2 Workspace File Structure

```
my_ros2_ws/
 src/
    my_package/
        setup.py
        package.xml
        resource/
        launch/
        setup.cfg
        my_package/
        test/
 install/
 build/
 log/
```

5.3 Creating a Package and Node

```
$ mkdir my_ros2_ws && cd my_ros2_ws
$ mkdir src
$ cd src
$ ros2 pkg create --build-type ament_python my_package --dependencies
   rclpy
$ cd my_package/my_package
$ touch node_name.py
$ chmod +x node_name.py
```

In setup.py:

```
entry_points={
    'console_scripts': [
        'node_name = my_package.node_name:main',
    ],
}
```

5.4 Assignment 2: Spiral Motion Using Turtlesim

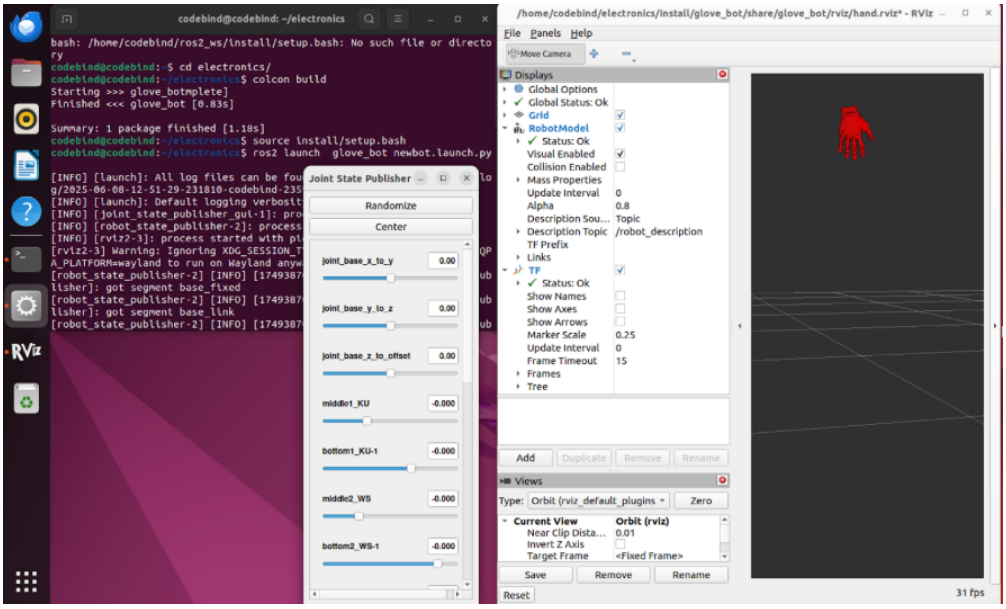
```
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist

class CircleTurtle(Node):
    def __init__(self):
        super().__init__('circle_turtle')
        self.publisher_ = self.create_publisher(Twist, 'turtle1/cmd_vel',
            10)
        timer_period = 0.1
        self.timer = self.create_timer(timer_period, self.move_in_circle)
```

```
    def move_in_circle(self):
        msg = Twist()
        msg.linear.x = 2.0
        msg.angular.z = 1.0
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: Linear x = %.2f, Angular z =
            %.2f' % (msg.linear.x, msg.angular.z))

def main(args=None):
    rclpy.init(args=args)
    node = CircleTurtle()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

6 Importing the Hand Model

Imported a hand model using ACDC4ROBOT, generated '.stl' and '.urdf' files, and launched in RViz.

```
$ colcon build
$ source install/setup.bash
$ ros2 launch my_package newbot.launch.py
```



7 Making new joints in URDF

Created additional links and joints for 3-axis control using URDF.

Listing 1: Added joints in URDF for 3-axis rotation

```
<joint name="base_joint_fixed" type="fixed">
    <parent link="base_fixed"/>
    <child link="base_x"/>
    <origin xyz="0 0 0" rpy="0 0 0"/>
</joint>

<joint name="base_joint_x" type="revolute">
    <parent link="base_x"/>
    <child link="base_y"/>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <axis xyz="1 0 0"/>
    <limit lower="-3.14" upper="3.14" effort="100000000" velocity="
       10000000"/>
</joint>

<joint name="base_joint_y" type="revolute">
    <parent link="base_y"/>
    <child link="base_z"/>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <axis xyz="0 1 0"/>
    <limit lower="-3.14" upper="3.14" effort="10000000" velocity="10000000
       "/>
</joint>

<joint name="base_joint_z" type="revolute">
    <parent link="base_z"/>
    <child link="base_link_offset"/>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <axis xyz="0 0 1"/>
    <limit lower="-3.14" upper="3.14" effort="10000000" velocity="10000000
       "/>
</joint>

<joint name="base_link_offset_joint" type="fixed">
    <parent link="base_link_offset"/>
    <child link="base_link"/>
    <origin xyz="0.25921036 0.01309030 0.99916261" rpy="0 0 0"/>
</joint>
```

8 Moving the model using Arduino

8.1 Using a potentiometer

We attempt to control one joint using a potentiometer. The node for the following is given:

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import JointState
import serial
```

```
import time

class JointPublisherFromSerial(Node):
    def __init__(self):
        super().__init__('joint_state_from_serial')

        # customize this based on your URDF joint name + limits
        self.joint_name = 'middle2_WS'
        self.min_angle = -0.25       # min joint angle in radians
        self.max_angle =  0.7   # max joint angle in radians

        # connect to Arduino (make sure Serial Monitor is CLOSED)
        try:
            self.ser = serial.Serial('/dev/ttyUSB0', 9600, timeout=1)
            time.sleep(2)  # wait for Arduino to reset
        except serial.SerialException:
            self.get_logger().error("Could not open /dev/ttyUSB0. Is it in
                use?")
            exit(1)

        self.joint_pub = self.create_publisher(JointState, '/joint_states'
            , 10)
        self.timer = self.create_timer(0.05, self.timer_callback)

        self.get_logger().info("JointPublisherFromSerial Node Started")

    def timer_callback(self):
        if self.ser.in_waiting > 0:
            try:
                line = self.ser.readline().decode('utf-8').strip()
                pot_val = int(line)

                mapped_angle = self.map_range(pot_val, 0, 1023, self.
                    min_angle, self.max_angle)

                # full joint list
                self.joint_names = [
                    'middle1_KU', 'bottom1_KU-1',
                    'middle2_WS', 'bottom2_WS-1',
                    'middle3_SR', 'bottom3_SR-1',
                    'middle4_SE', 'bottom4_SE-1',
                    'middle5_MA', 'bottom5_MA-1',
                    'base_link_MA-2', 'base_link_SE-2',
                    'base_link_SR-2', 'base_link_WS-2',
                    'base_link_KU-2'
                ]
                positions = [0.0] * len(self.joint_names)

                # just change one joint's value
                idx = self.joint_names.index('middle3_SR')
                positions[idx] = mapped_angle

                msg = JointState()
                msg.header.stamp = self.get_clock().now().to_msg()
```

```
                msg.name = self.joint_names
                msg.position = positions

                self.joint_pub.publish(msg)
                self.get_logger().info(f"Updated middle3_SR: {mapped_angle
                    :.2f} rad")

            except Exception as e:
                self.get_logger().warn(f"Serial read error: {e}")


    def map_range(self, x, in_min, in_max, out_min, out_max):
        return (x - in_min) * (out_max - out_min) / (in_max - in_min) +
            out_min

def main(args=None):
    rclpy.init(args=args)
    node = JointPublisherFromSerial()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

8.2 Using an IMU sensor

Next, using similar logic we do this for changing the orientation of hand using an IMU sensor. For that the ros2 node is given:

```
import math
import rclpy
import numpy as np
from rclpy.node import Node
from sensor_msgs.msg import Imu
from geometry_msgs.msg import TransformStamped
import tf2_ros
from tf_transformations import quaternion_multiply, quaternion_about_axis
from rclpy.parameter import Parameter

class GyroHandPosePublisher(Node):
    def __init__(self):
        super().__init__('gyro_hand_pose_publisher')
        self.get_logger().info("Gyro Hand Pose Publisher Node:
            Initializing...")

        self.tf_broadcaster = tf2_ros.TransformBroadcaster(self)

        self.use_sim_time = self.get_parameter('use_sim_time').value
        self.get_logger().info(f"Gyro Hand Pose Publisher Node:
            use_sim_time: {self.use_sim_time}")
```

```
        self.declare_parameter('hand_frame_id', 'base_link')
        self.hand_frame_id = self.get_parameter('hand_frame_id').value
        self.get_logger().info(f"Gyro Hand Pose Publisher Node: Hand frame
            ID: {self.hand_frame_id}")

        self.current_orientation = np.array([0.0, 0.0, 0.0, 1.0])
        self.last_imu_timestamp = None

        self.imu_subscriber = self.create_subscription(
            Imu,
            '/imu/data_raw',
            self.imu_callback,
            10
        )
        self.get_logger().info(f'Gyro Hand Pose Publisher Node:
           Subscribing to {self.imu_subscriber.topic_name} topic.')

    def imu_callback(self, msg: Imu):
        current_timestamp = msg.header.stamp

        delta_t = 0.0
        if self.last_imu_timestamp is not None:
            last_sec = self.last_imu_timestamp.sec + self.
               last_imu_timestamp.nanosec / 1e9
            current_sec = current_timestamp.sec + current_timestamp.
               nanosec / 1e9
            delta_t = current_sec - last_sec

            if delta_t < 0:
                self.get_logger().warn(
                    f"Negative delta_t ({delta_t:.4f}s) detected. "
                    "Resetting orientation and last timestamp to prevent
                       instability."
                )
                self.current_orientation = np.array([0.0, 0.0, 0.0, 1.0])
                self.last_imu_timestamp = current_timestamp
                return

        self.last_imu_timestamp = current_timestamp

        wx = msg.angular_velocity.x
        wy = msg.angular_velocity.y
        wz = msg.angular_velocity.z

        if delta_t > 0:
            omega_magnitude = math.sqrt(wx**2 + wy**2 + wz**2)

            if omega_magnitude > 1e-6:
                angle_increment = omega_magnitude * delta_t

                axis_x = wx / omega_magnitude
                axis_y = wy / omega_magnitude
                axis_z = wz / omega_magnitude
```

```
                delta_q = quaternion_about_axis(angle_increment, [axis_x,
                    axis_y, axis_z])
                self.current_orientation = quaternion_multiply(self.
                    current_orientation, delta_q)
                self.current_orientation = self.current_orientation / np.
                    linalg.norm(self.current_orientation)

        transform_stamped = TransformStamped()
        transform_stamped.header.stamp = (
            msg.header.stamp if self.use_sim_time else self.get_clock().
                now().to_msg()
        )
        transform_stamped.header.frame_id = 'world'
        transform_stamped.child_frame_id = self.hand_frame_id

        transform_stamped.transform.rotation.x = self.current_orientation
            [0]
        transform_stamped.transform.rotation.y = self.current_orientation
            [1]
        transform_stamped.transform.rotation.z = self.current_orientation
            [2]
        transform_stamped.transform.rotation.w = self.current_orientation
            [3]

        transform_stamped.transform.translation.x = 0.0
        transform_stamped.transform.translation.y = 0.0
        transform_stamped.transform.translation.z = 0.0

        self.tf_broadcaster.sendTransform(transform_stamped)

def main(args=None):
    rclpy.init(args=args)
    gyro_hand_pose_publisher = GyroHandPosePublisher()
    gyro_hand_pose_publisher.get_logger().info("Gyro Hand Pose Publisher
        Node: Entering rclpy.spin()...")
    try:
        rclpy.spin(gyro_hand_pose_publisher)
    except KeyboardInterrupt:
        gyro_hand_pose_publisher.get_logger().info(
            "Gyro Hand Pose Publisher Node: Shutting down cleanly via
                KeyboardInterrupt."
        )
    finally:
        gyro_hand_pose_publisher.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

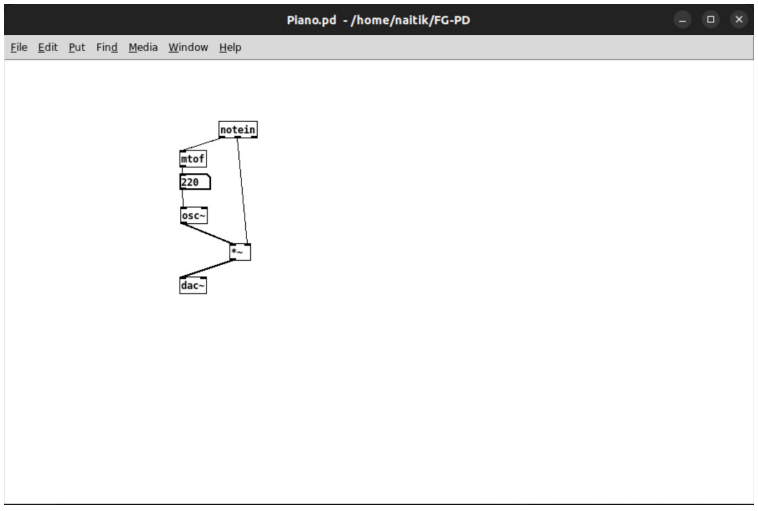
9 Audio Integration

9.1 Introduction to Pure Data

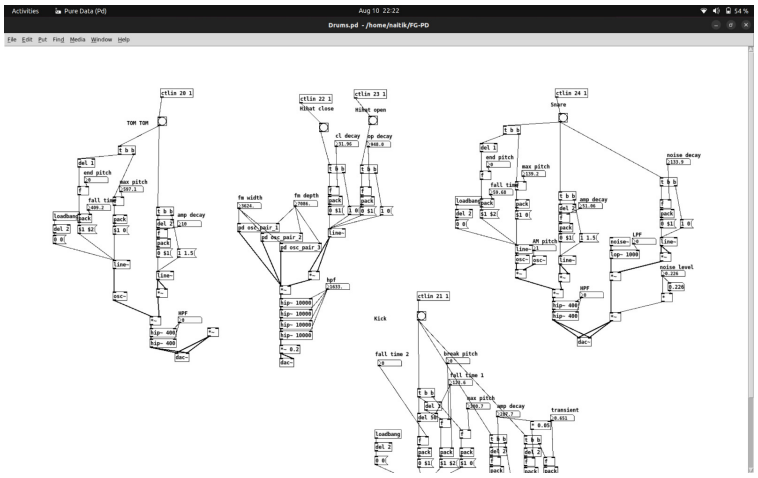
- **Pure Data (Pd)** is a visual programming language used to create interactive multimedia systems, especially suited for audio processing and synthesis.
- We were introduced to Pd through hands-on learning, starting with the creation of **basic additive synthesis patches**.
- These patches allowed us to understand core concepts such as signal generation, oscillators, wave combination, and real-time sound control.

9.2 Instrument Patch Design and Exploration

- We explored two main types of patches:
  - **Pre-made patches** simulating traditional instruments, and
  - **Custom-designed patches** tailored to respond to gesture inputs from our hand model.
- These patches were designed in the Pd environment to respond to control signals, enabling dynamic instrument switching or modulation.
- This phase formed the backbone for building real-time, interactive control over audio output.



Piano patch



Drums patch

9.3 Integration with Hand Tracking

- A major goal of our project was to link Pd with gesture data obtained from a hand tracking system using **ROS and RViz**.

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import JointState
import time
import mido
from mido import Message
port_name = 'Virtual Raw MIDI 2-0:VirMIDI 2-0 24:0'

try:
    outport = mido.open_output(port_name)
    print(f"Connected to: {port_name}")
except Exception as e:
    print(f"Failed to open MIDI port: {e}")
    exit()


# Send a Control Change message: CC 7 (Volume), value 100, channel 0
cc_msg = mido.Message('control_change', control=7, value=100, channel
   =0)
outport.send(cc_msg)
print(f"Sent CC     control: {cc_msg.control}, value: {cc_msg.value},
    channel: {cc_msg.channel}")


class JointToSound(Node):

    def __init__(self):
        super().__init__('joint_to_sound')
        self.subscription = self.create_subscription(
            JointState,
            '/joint_states',
```

```
            self.listener_callback,
            10)
        self.joint_sound_triggered = {}  # To track per-joint state

    def listener_callback(self, msg):
        for i, joint_name in enumerate(msg.name):
            angle = msg.position[i]

            # Play note if joint crosses threshold and wasn't already
                triggered
            if joint_name == 'base_link_WS-2':
                if angle > 0.6 and not self.joint_sound_triggered.get
                    (joint_name, False):
                    self.get_logger().info(f'{joint_name} moved to {
                        angle}     PLAY NOTE')
                    self.play_tom(joint_name)
                    self.joint_sound_triggered[joint_name] = True

                elif angle <= 0.6:
                    self.joint_sound_triggered[joint_name] = False  #
                        Reset when back

            if joint_name == 'base_link_SR-2':
                if angle > 0.6 and not self.joint_sound_triggered.get
                    (joint_name, False):
                    self.get_logger().info(f'{joint_name} moved to {
                        angle}     PLAY NOTE')
                    self.play_kick(joint_name)
                    self.joint_sound_triggered[joint_name] = True

                elif angle <= 0.6:
                    self.joint_sound_triggered[joint_name] = False  #
                        Reset when back

            if joint_name == 'base_link_SE-2':
                if angle > 0.9 and not self.joint_sound_triggered.get
                    (joint_name, False):
                    self.get_logger().info(f'{joint_name} moved to {
                        angle}     PLAY NOTE')
                    self.play_hhc(joint_name)
                    self.joint_sound_triggered[joint_name] = True

                elif angle <= 0.9:
                    self.joint_sound_triggered[joint_name] = False  #
                        Reset when back

            if joint_name == 'base_link_KU-2':
                if angle > 1.0 and not self.joint_sound_triggered.get
                    (joint_name, False):
                    self.get_logger().info(f'{joint_name} moved to {
                        angle}     PLAY NOTE')
                    self.play_snare(joint_name)
                    self.joint_sound_triggered[joint_name] = True
```

```
                elif angle <= 1.0:
                    self.joint_sound_triggered[joint_name] = False  #
                        Reset when back

            if joint_name == 'base_link_MA-2':
                if angle > 0.6 and not self.joint_sound_triggered.get
                    (joint_name, False):
                    self.get_logger().info(f'{joint_name} moved to {
                        angle}     PLAY NOTE')
                    self.play_hho(joint_name)
                    self.joint_sound_triggered[joint_name] = True

                elif angle <= 0.6:
                    self.joint_sound_triggered[joint_name] = False  #
                        Reset when back




    def play_tom(self, joint_name):

        msg = Message('control_change', control=20, value=127,
            channel=0)
        outport.send(msg)

    def play_kick(self, joint_name):

        msg = Message('control_change', control=21, value=127,
            channel=0)
        outport.send(msg)

    def play_hhc(self, joint_name):

        msg = Message('control_change', control=22, value=127,
            channel=0)
        outport.send(msg)

    def play_hho(self, joint_name):

        msg = Message('control_change', control=23, value=127,
            channel=0)
        outport.send(msg)

    def play_snare(self, joint_name):

        msg = Message('control_change', control=24, value=127,
            channel=0)
        outport.send(msg)

def main(args=None):
    rclpy.init(args=args)
    node = JointToSound()
    rclpy.spin(node)
    node.destroy_node()
```

```
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Code for ROS2 node that takes input from Joint State Publisher and sends to Pure Data to play sound.

- The hand model's gestures and movements were mapped to audio control logic, such as:
  - Switching between different instrument patches,
  - Modifying frequency, amplitude, or filter parameters in real-time.
- Example mappings:
  - **Open hand** → Switch to Instrument A,
  - **Closed fist** → Switch to Instrument B,
  - **Vertical movement** → Modulate pitch.

```
    import rclpy
from rclpy.node import Node
from sensor_msgs.msg import JointState
import serial
import time

class JointPublisherFromFlex(Node):
    def __init__(self):
        super().__init__('flex_joint_publisher')

        self.joint_names = [
            'middle1_KU', 'bottom1_KU-1',
            'middle2_WS', 'bottom2_WS-1',
            'middle3_SR', 'bottom3_SR-1',
            'middle4_SE', 'bottom4_SE-1',
            'middle5_MA', 'bottom5_MA-1',
            'base_link_MA-2', 'base_link_SE-2',
            'base_link_SR-2', 'base_link_WS-2',
            'base_link_KU-2'
        ]

        # Joint mapping constants (adjust if needed)
        self.min_angle = -0.261799
        self.max_angle = 0.308997

        try:
            self.ser = serial.Serial('/dev/ttyACM0', 9600,
                timeout=1)
            time.sleep(2)  # Give time to reset
        except serial.SerialException:
            self.get_logger().error("    Failed to open /dev/
                ttyUSB0")
```

```
            exit(1)

        self.joint_pub = self.create_publisher(JointState, '/
            joint_states', 10)
        self.timer = self.create_timer(0.05, self.timer_callback)

        self.get_logger().info("    Flex Sensor Joint Publisher
            Node Started")

    def timer_callback(self):
        if self.ser.in_waiting > 0:
            try:
                raw_line = self.ser.readline().decode('utf-8').
                    strip()
                tokens = raw_line.split(',')

                if len(tokens) != 6:
                    self.get_logger().warn(f"       Invalid data:
                        {raw_line}")
                    return

                # Parse and map values individually
                val_ku = self.map_range(int(tokens[0]), 245, 280,
                    self.min_angle, self.max_angle)
                val_ws = self.map_range(int(tokens[1]), 0, 100,
                    self.min_angle, self.max_angle)
                val_sr = self.map_range(int(tokens[2]), 195, 235,
                    self.min_angle, self.max_angle)
                val_se = self.map_range(int(tokens[3]), 13, 190,
                    self.min_angle, self.max_angle)
                val_ma = self.map_range(int(tokens[4]), 230, 260,
                    self.min_angle, self.max_angle)

                positions = [0.0] * len(self.joint_names)
                positions[self.joint_names.index('base_link_KU-2'
                    )] = val_ku
                positions[self.joint_names.index('base_link_WS-2'
                    )] = val_ws
                positions[self.joint_names.index('base_link_SR-2'
                    )] = val_sr
                positions[self.joint_names.index('base_link_SE-2'
                    )] = val_se
                positions[self.joint_names.index('base_link_MA-2'
                    )] = val_ma

                msg = JointState()
                msg.header.stamp = self.get_clock().now().to_msg
                    ()
                msg.name = self.joint_names
                msg.position = positions

                self.joint_pub.publish(msg)
                self.get_logger().info(
                    f"Angles: KU={val_ku:.2f}, WS={val_ws:.2f},
```

```
                        SR={val_sr:.2f}, SE={val_se:.2f}, MA={
                        val_ma:.2f}"
                )

            except Exception as e:
                self.get_logger().warn(f"       Error reading
                    serial: {e}")

    def map_range(self, x, in_min, in_max, out_min, out_max):
        return (x - in_min) * (out_max - out_min) / (in_max -
            in_min) + out_min

def main(args=None):
    rclpy.init(args=args)
    node = JointPublisherFromFlex()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Code for ROS2 node that takes data from serial monitor and sends to RViz Joint State Publisher

- This resulted in a contactless, expressive audio interface that combined vision-based gesture recognition with modular sound synthesis.

10 Machine Learning Development

10.1 Introduction

To facilitate intuitive human-computer interaction for individuals with hearing or speech impairments, we explored machine learning-based sign language recognition systems. Our objective was to interpret hand gestures and convert them into meaningful textual representations using trained models on time-series data from gesture inputs.

10.2 Initial Learning Phase

We began by studying foundational machine learning models, including Linear Regression, and applied them on simple datasets using Kaggle notebooks, such as the MNIST digit classification notebook. This allowed us to understand the structure and flow of supervised learning pipelines.

Listing 2: Example: Simple MNIST classification training

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train/255.0, x_test/255.0
y_train, y_test = to_categorical(y_train), to_categorical(y_test)

model = Sequential([
    Flatten(input_shape=(28,28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[
   'accuracy'])
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

10.3 Transition to LSTM-based Sequence Modeling

After grasping the basics, we shifted our focus to Long Short-Term Memory (LSTM) networks due to their superior performance on sequential data. We experimented with multiple random gesture-based datasets, achieving accuracy levels close to 90%. The LSTM's ability to learn temporal patterns made it ideal for interpreting sign sequences.

Listing 3: Example: Basic LSTM sequence classification

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential([
    LSTM(64, input_shape=(time_steps, features)),
    Dense(num_classes, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[
   'accuracy'])
model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val))
```

11 ROS Integration and Sign Language Recognition

11.1 Dataset Customization and Preprocessing

We chose the *ASL Sensor Dataglove Dataset*[1] as our primary dataset. This dataset was preprocessed and tailored to align with our project's specific needs. Each gesture was represented as a row in a CSV file, and we published this data row-wise to the serial monitor for real-time streaming.

Listing 4: Data Publisher Node - streams CSV data via ROS2

```
class DataPublisherNode(Node):
```

[1] ASL Sensor Dataglove Dataset on Kaggle

```
    def __init__(self):
        super().__init__('data_publisher_node')
        self.publisher_ = self.create_publisher(Float32MultiArray, '
            original_sensor_data', 10)

        csv_path = os.path.join(get_package_share_directory('hand_model'),
            'SignSpeak.csv')
        df = pd.read_csv(csv_path).fillna(0)
        self.feature_df = df.drop(columns=['_id', 'word'])

        self.timer = self.create_timer(0.1, self.timer_callback)
        self.row_index = 0

    def timer_callback(self):
        if self.row_index < len(self.feature_df):
            row_data = self.feature_df.iloc[self.row_index].values.tolist
                ()
            self.publisher_.publish(Float32MultiArray(data=row_data))
            self.row_index += 1
```

11.2 Model Behavior and Prediction Mechanism

Our final LSTM model was trained on the customized dataset. During inference, it operated on a sliding window of 10 consecutive rows. It analyzed the sequence and identified the most frequently occurring gesture over that span, outputting it as the predicted gesture.

Listing 5: Prediction Node with sliding window and majority voting

```
class PredictionAndSaveNode(Node):
    def __init__(self):
        super().__init__('prediction_and_save_node')
        self.SEQUENCE_LENGTH = 15
        self.data_buffer = collections.deque(maxlen=self.SEQUENCE_LENGTH)
        self.final_data_to_save = []

        model_dir = os.path.join(get_package_share_directory('hand_model')
            , 'model_files')
        self.model = load_model(os.path.join(model_dir, 'asl_lstm_model.h5
            '))
        self.scaler = joblib.load(os.path.join(model_dir, 'scaler.joblib')
            )
        self.encoder = joblib.load(os.path.join(model_dir, 'label_encoder.
            joblib'))

        self.subscription = self.create_subscription(Float32MultiArray, '
            original_sensor_data', self.listener_callback, 10)

    def listener_callback(self, msg):
        self.data_buffer.append(np.array(msg.data))
        if len(self.data_buffer) == self.SEQUENCE_LENGTH:
            clean_sequence = np.array(self.data_buffer)
            scaled_sequence = self.scaler.transform(clean_sequence)
```

```
            prediction_input = scaled_sequence.reshape(1, self.
                SEQUENCE_LENGTH, -1)
            prediction = self.model.predict(prediction_input, verbose=0)
            predicted_label = self.encoder.classes_[np.argmax(prediction)]
            self.final_data_to_save.append(predicted_label)
```

11.3 Realtime Gesture Prediction

For real-world testing, a dedicated node processed incoming live sensor data, ran predictions, and saved detected gestures.

Listing 6: Realtime Predictor Node - live inference with Attention-LSTM model

```
class RealtimePredictorNode(Node):
    def __init__(self):
        super().__init__('realtime_predictor_node')
        self.SEQUENCE_LENGTH = 1500
        self.data_buffer = collections.deque(maxlen=self.SEQUENCE_LENGTH)

        package_share_dir = get_package_share_directory('hand_model')
        self.model = load_model(os.path.join(package_share_dir, '
            asl_dataglove_advanced_model.h5'),
                                custom_objects={'Attention': Attention})
        self.scaler = joblib.load(os.path.join(package_share_dir, 'scaler.
            joblib'))
        self.encoder = joblib.load(os.path.join(package_share_dir, '
            label_encoder.joblib'))

        self.output_file_path = os.path.expanduser('~/Desktop/
            predicted_output.txt')
        open(self.output_file_path, 'w').close()

        self.subscription = self.create_subscription(Float32MultiArray, '
            imu_flex_data', self.listener_callback, 10)

    def listener_callback(self, msg):
        self.data_buffer.append(np.array(msg.data))
        if len(self.data_buffer) == self.SEQUENCE_LENGTH:
            sequence_scaled = self.scaler.transform(np.array(self.
                data_buffer))
            prediction_input = sequence_scaled.reshape(1, self.
                SEQUENCE_LENGTH, -1)
            prediction_probs = self.model.predict(prediction_input,
                verbose=0)[0]
            predicted_label = self.encoder.classes_[np.argmax(
                prediction_probs)]
            with open(self.output_file_path, 'a') as f:
                f.write(predicted_label.upper() + '\n')
            self.data_buffer.clear()
```

11.4 Sentence Construction and Spell Correction

To enhance the usability of the system, we implemented a spell-check mechanism that constructed coherent sentences using a buffer of the last five predicted words. This not only improved readability but also accounted for minor prediction errors by forming grammatically meaningful outputs.

Listing 7: Sentence construction and spell correction

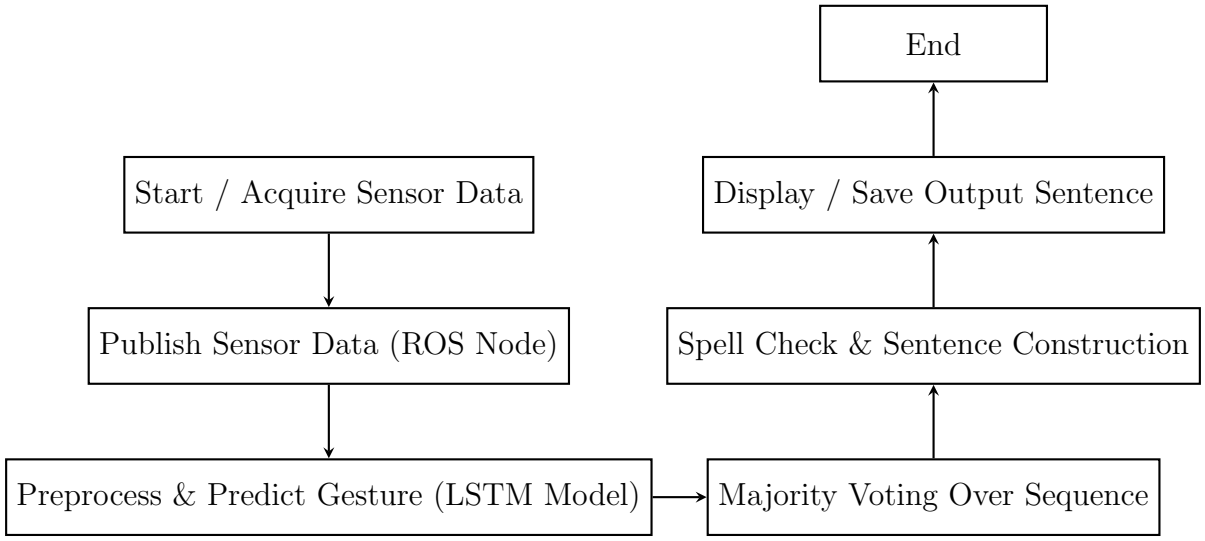
```
from spellchecker import SpellChecker
spell = SpellChecker()
corrected = [spell.correction(w) for w in last_words]
sentence = ' '.join(corrected)
```

11.5 Hardware Design

The hardware setup includes a sensor-enabled glove with multiple flex sensors and an IMU. The microcontroller collects raw flex + orientation data and streams it via serial to a connected PC for processing.

11.6 System Flowchart

The following flowchart illustrates the overall workflow of the sign language recognition system, from data acquisition to sentence construction.



12 Project Constraints and Limitations

The primary aim of the project was to design and implement a fully functional hand model capable of recognizing sign language and providing audio output. However, the final physical implementation faced several constraints:

- Initially, the STM microcontroller was planned to be used due to its low latency advantage. However, writing and modifying uploaded data on it proved difficult. When additional sensors were connected, it often returned garbage values. Considerable time was spent troubleshooting this issue, but it could not be resolved, leading to a switch to the Arduino Nano.
- During hardware assembly, minute soldering work was required. It is suspected that unintentional solder bridges between components may have caused hardware malfunction, rendering the system non-operational.
- The project plan included creating a custom dataset for sign language and training the ML model on it. Due to the non-functional hardware, this was not achieved.

As a result, the complete setup was implemented and tested in a software environment:

- The hand model was designed in Fusion 360 and imported into ROS for visualization and simulation.
- A sign prediction model using machine learning was developed and integrated with audio playback functionality via Pure Data.
- The system was tested using simulated or pre-given data to successfully predict signs/words and play corresponding audio.
- Partial hardware testing was carried out on a breadboard using flex sensors to control finger movements and IMU sensor to move the hand model, which worked in a simulated setup.

13 Conclusion

Despite hardware challenges, the project successfully demonstrated the integration of robotic hand modeling, machine learning, and audio output in a simulated environment. The ROS-based hand model, combined with the LSTM-based sign recognition system and Pure Data audio integration, worked effectively when tested with pre-given input data. Although the complete physical glove with integrated sensors and microcontroller was not achieved due to hardware constraints, the software simulation provided proof of concept and validated the system's functionality. This project demonstrated how embedded systems, robotics, and machine learning can be combined to create a functional assistive device, potentially improving communication accessibility for individuals with hearing or speech impairments. The work lays a solid foundation for future iterations where hardware reliability can be improved, a custom dataset can be collected, and the glove can be fully implemented for real-time sign language recognition and audio output.