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
Naveen Ola	240684
Nishant Kumar	240706
Prakhar	240763
Priyanshu Verma	240811
Saksham Gupta	240912
Tarun Goyal	241096

June 12, 2025

Contents

1 Timeline

- Ubuntu installation and setup, git setup.
- **Task 1:** Made an ASCII art in Linux terminal.
- **Fusion 360:** Learnt about joints by following a tutorial.
- **Task 2:** Made a model of a car (imported tyres from GrabCAD).
- Learnt Turtlesim.
- **Task 3:** Made a spiral using Turtlesim (ROS2).
- In Fusion, learnt to convert Fusion files to URDF and STL files using ACDC4ROBOT add-in in Fusion 360.
- Made launch '.py' files.
- Imported a model of a hand from the internet and converted it similarly.
- Loaded the hand model into RViz.
- Made changes in URDF so that yaw, pitch, and roll rotation could be changed using Joint State Publisher.
- Integrated IMU sensor data to control rotations.

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.

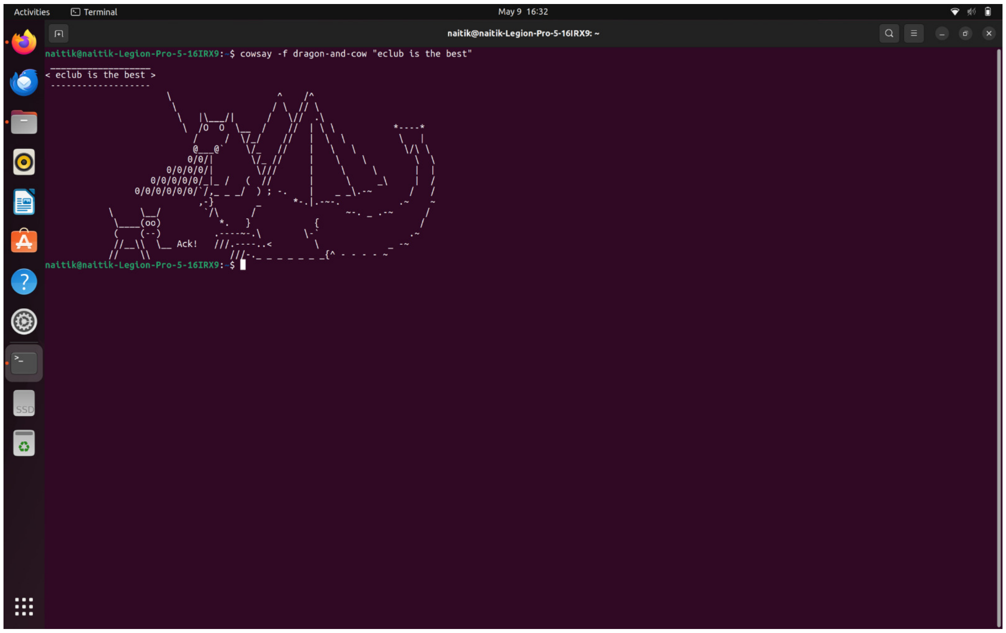
3 Introduction to Linux and Ubuntu Setup

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"
```



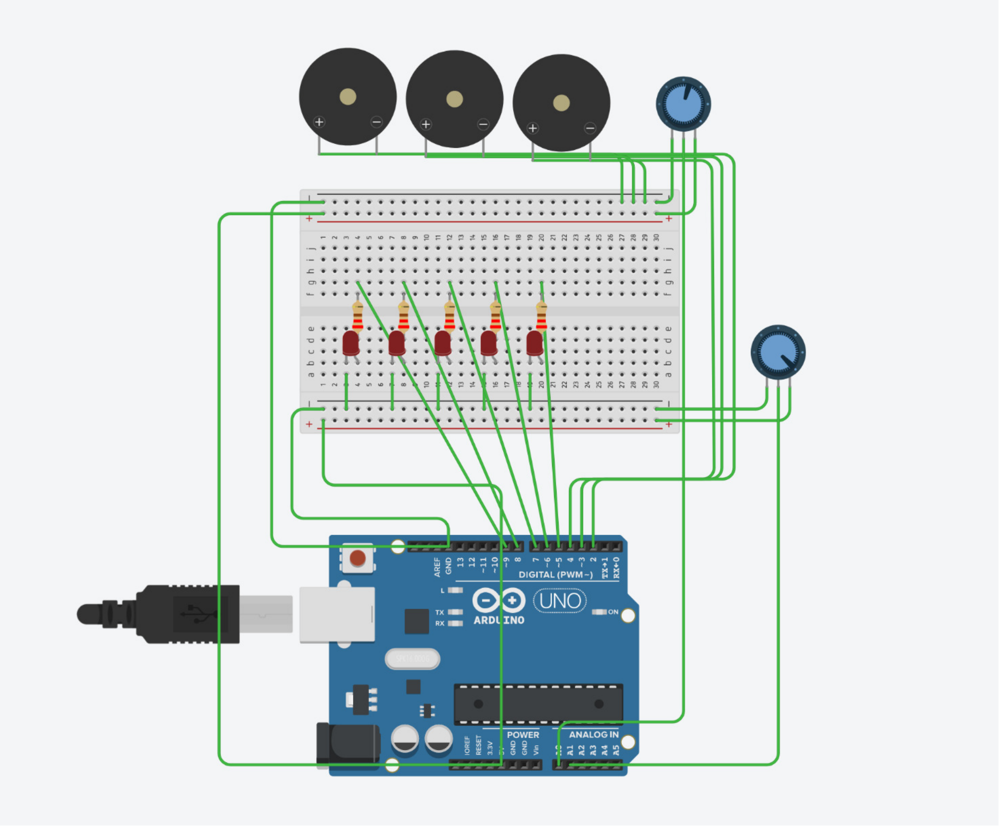
```
naitik@naitik-Legion-Pro-5-16IRX9:~$ cowsay -f dragon-and-cow "eclub is the best"
 ___________________
< eclub is the best >
 -------------------
                       \                    ^    /^
                        \                  / \  // \
                         \   |\___/|      /   \//  .\
                          \  /O  O  \__  /    //  | \ \           *----*
                            /     /  \/_/    //   |  \  \          \   |
                            @___@`    \/_   //    |   \   \         \/\ \
                           0/0/|       \/_ //     |    \    \         \  \
                       0/0/0/0/|        \///      |     \     \       |  |
                    0/0/0/0/0/_|_ /   (  //       |      \     _\     |  /
                 0/0/0/0/0/0/`/,_ _ _/  ) ; -.    |    _ _\.-~       /   /
                             ,-}        _      *-.|.-~-.           .~    ~
             \     \__/        `/\      /                 ~-. _ .-~      /
              \____(oo)           *.   }            {                   /
              (    (--)          .----~-.\        \-`                 .~
              //__\\  \__ Ack!   ///.----..<        \             _ -~
             //    \\               ///-._ _ _ _ _ _ _{^ - - - - ~
naitik@naitik-Legion-Pro-5-16IRX9:~$
```

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',
    ],
}
```

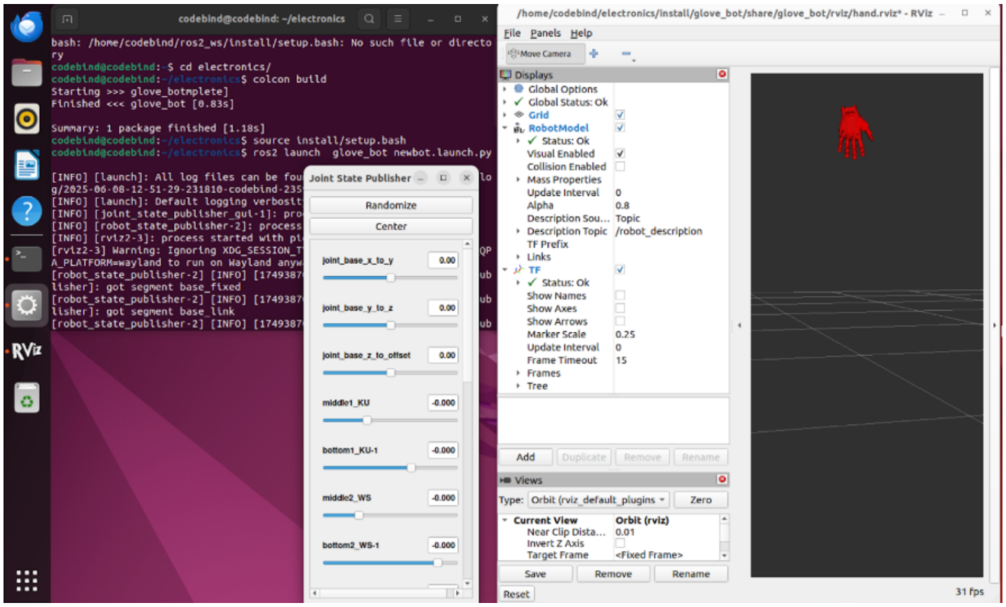
5.4 Assignment 2: Spiral Motion Using Turtlesim

```
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
```



```
bash: /home/codebind/ros2_ws/install/setup.bash: No such file or directo
ry
codebind@codebind:~$ cd electronics/
codebind@codebind:~/electronics$ colcon build
Starting >>> glove_botmplete]
Finished <<< glove_bot [0.83s]

Summary: 1 package finished [1.18s]
codebind@codebind:~/electronics$ source install/setup.bash
codebind@codebind:~/electronics$ ros2 launch  glove_bot newbot.launch.py

[INFO] [launch]: All log files can be fou                            lo
g/2025-06-08-12-51-29-231810-codebind-235
[INFO] [launch]: Default logging verbosit
[INFO] [joint_state_publisher_gui-1]: pro
[INFO] [robot_state_publisher-2]: process
[INFO] [rviz2-3]: process started with pi
[rviz2-3] Warning: Ignoring XDG_SESSION_T                            QP
A_PLATFORM=wayland to run on Wayland anyw
[robot_state_publisher-2] [INFO] [1749387                            ub
lisher]: got segment base_fixed
[robot_state_publisher-2] [INFO] [1749387                            ub
lisher]: got segment base_link
[robot_state_publisher-2] [INFO] [1749387                            ub
```

joint_base_x_to_y 0.00
joint_base_y_to_z 0.00
joint_base_z_to_offset 0.00
middle1_KU -0.000
bottom1_KU-1 -0.000
middle2_WS -0.000
bottom2_WS-1 -0.000

- Global Options
- ✓ Global Status: Ok
- Grid ☑
- RobotModel ☑
  - ✓ Status: Ok
  - Visual Enabled ☑
  - Collision Enabled □
  - Mass Properties
  - Update Interval 0
  - Alpha 0.8
  - Description Sou... Topic
  - Description Topic /robot_description
  - TF Prefix
  - Links
- TF ☑
  - ✓ Status: Ok
  - Show Names □
  - Show Axes □
  - Show Arrows □
  - Marker Scale 0.25
  - Update Interval 0
  - Frame Timeout 15
  - Frames
  - Tree

Type: Orbit (rviz_default_plugins

- Current View Orbit (rviz)
  - Near Clip Dista... 0.01
  - Invert Z Axis □
  - Target Frame <Fixed Frame>

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()
```