DESIGNING AN INTUITIVE GUI FOR CONTROLLING A 20 DOF ROBOT

Documentation

*Nikunj Sharma and Ishita Tiwari (Intern: July - August 2023)*

**Aim:**

1. To study various open-source GUIs available in the market for robot.
2. Designing GUI with intuitive controls.
3. GUI should provide real-time feedback from Robot.

**Introduction:**

This project involves designing and developing the user interface (UI) for a robot control system using the Unity game engine. The UI will allow users to interact with the robot's various functionalities and control its movements. It also includes real-time interaction of robot with the UI design. The idea of the project was to control various robotic joints using slider and give them the movement according to the slider value in various directions. This as well contains the user created animations through step-by-step movements of the robot in the animation tabs. The idea is to not just to be able to control the robotic movement in software but as well as to be able to control it in real-time with the actual robot. The movement given in GUI will give the movement in the actual robot and vice-versa.

**ElectronBot:**

****

**Project Overview:**

Peng Zhihui has created a small and adorable robot called the "emoji-bot," showcasing meticulous attention to detail and fine craftsmanship. The robot features intricate 3D printed models, which are intended for Selective Laser Sintering (SLS) printing in nylon. The electronics package consists of custom-designed, small PCBs using Altium Designer, along with readily available modules for a circular LCD and camera.

**Electronic package:**

The main PCB, hosting an STM32F405 microcontroller, manages the display and SD card functions. This choice of STM32 was driven by the need to connect to an external USB3300 high-speed USB PHY. A separate sensor PCB handles various components, including a gesture sensor, a USB hub, an MPU6050 9-axis sensor, and a USB camera module. This sensor PCB connects to the robot's USB-C connector in the base through a flexible flat cable (FFC), allowing the robot to rotate on its base.

**Arm mechanisms and Servos:**

[Peng] has meticulously designed a two-servo shoulder mechanism for the robot's arms. This mechanism is designed to be back-driveable, allowing the host computer to track and record the motor positions for later replay. The servos are connected via I2C, enabling all five servos to share the same bus, thus optimizing resource utilization.

**User Interface:**

The provided software package includes various components for interacting with the ElectronBot:

ElectronBotSDK-LowLevel: This is the foundational library that allows direct communication with the ElectronBot hardware, managing core control and data exchange.

ElectronBotSDK-Player: Building upon the low-level library, this component offers more advanced functions for creating intricate robot behaviors and interactions.

ElectronBotSDK-UnityBridge: Acting as a bridge, this component connects the SDK with the Unity environment, enabling smooth communication and data transfer between the two.

Electron-Studio: Developed in Unity, Electron-Studio is the host computer software with a graphical interface. It facilitates control and interaction with the ElectronBot, allowing users to design robot behaviours visually.

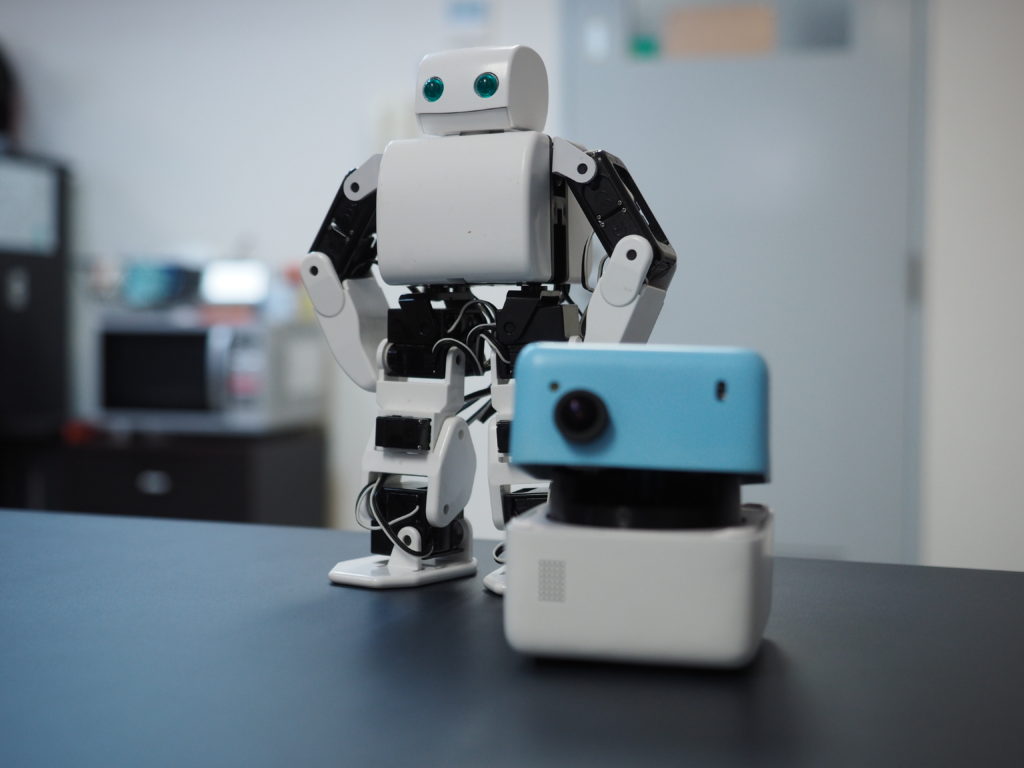
Additionally, the software package includes the BotDriver, a USB driver necessary for establishing a connection between the ElectronBot and the host computer. The driver can be installed by updating it through the device manager and selecting the relevant files from the provided directory.

**GitHub Link**: <https://github.com/peng-zhihui/ElectronBot>

**YouTube Video:** <https://www.youtube.com/watch?v=FmKTiH5Lca4->

**3D Model:** <https://github.com/peng-zhihui/ElectronBot/tree/main/4.CAD-Model>

**Plen 2:**



**Hardware:**

Frame and Structure: PLEN2's frame consists of 3D-printed plastic parts, which provide the foundation for the robot's structure. The design is modular, allowing for easy assembly and disassembly.

Actuators: PLEN2 utilizes servo motors for its movement. These servo motors control the joints of the robot and enable its humanoid-like motions.

Sensors: PLEN2 is equipped with various sensors that provide feedback to the control system. These sensors can include gyroscopes, accelerometers, and potentially other environmental sensors depending on the specific configuration.

Connectivity: PLEN2 typically includes wireless communication capabilities, enabling it to be controlled remotely via Bluetooth or Wi-Fi.

Power Supply: PLEN2 is powered by a rechargeable battery, which provides the energy required for the robot's movements and operations.

**Software:**

Programming Environments: PLEN2 can be programmed using a variety of programming languages and development environments. This includes languages like Python, as well as graphical programming environments suited for beginners.

Control Interface: Users can interact with PLEN2 through software interfaces on a computer or mobile device. These interfaces allow users to send commands, adjust movements, and control the robot's behavior.

Motion Control: PLEN2's software enables users to create and control various motions and movements. This can range from simple predefined motions to more complex custom routines.

Customization: The open-source nature of PLEN2's software allows for customization and modification. Users can adapt existing code, create new motions, and experiment with different behaviors.

**Firmware:**

Microcontroller: PLEN2's microcontroller is the brain of the robot, responsible for processing commands and controlling the servo motors. The specific microcontroller used may vary, but it's typically a capable and programmable component.

Communication: The firmware handles communication between the control interface (computer or mobile device) and the robot itself. It interprets commands, translates them into motor movements, and provides feedback.

Servo Control: The firmware manages the servo motors, ensuring that they move according to the desired patterns and motions specified by the user.

Sensors Integration: If PLEN2 includes sensors, the firmware handles sensor data processing and integration into the robot's behavior and responses.

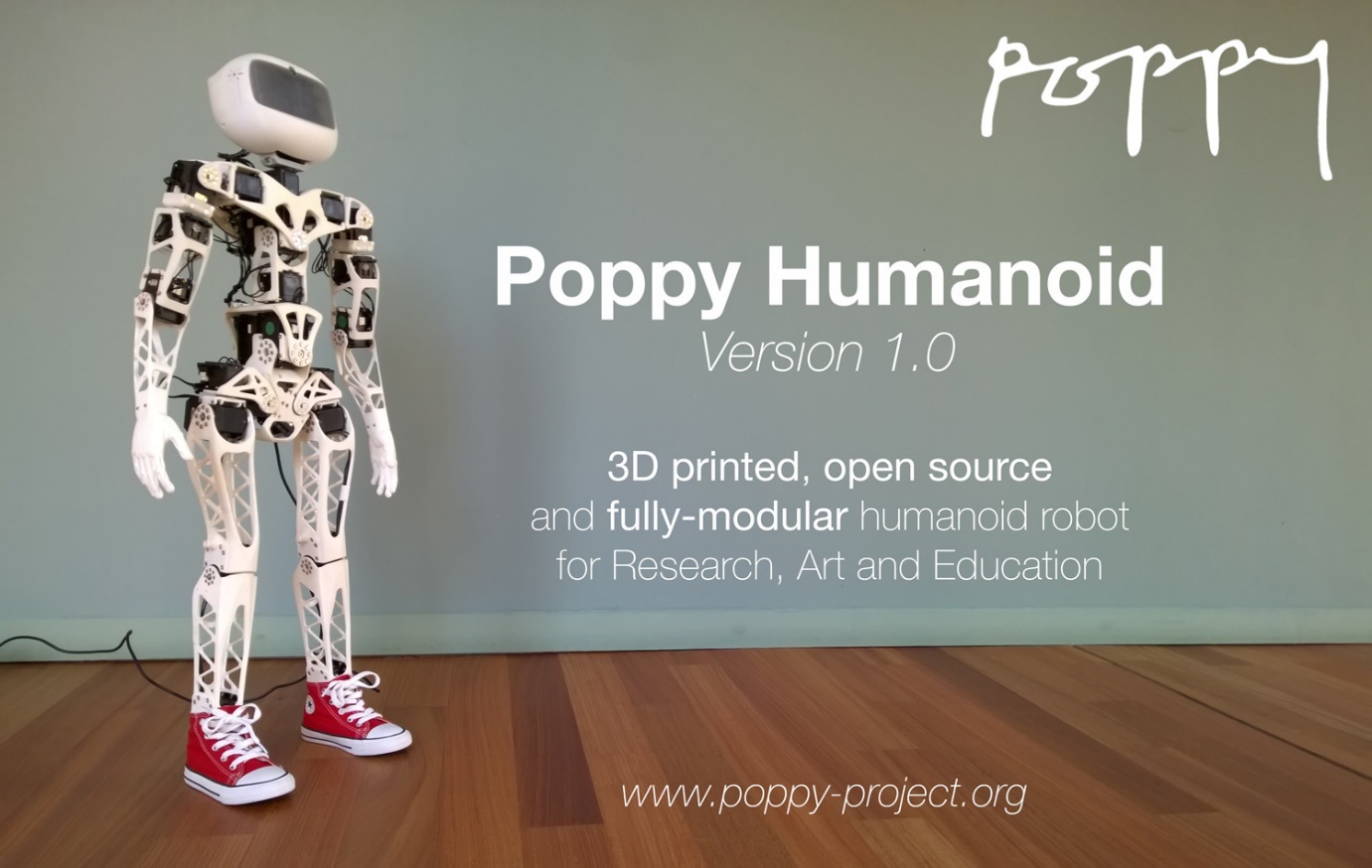
**Website:** <https://plen.jp/en/plen2-2/>

**GitHub:** <https://github.com/plenprojectcompany/PLEN2>

**YouTube Channel**: <https://www.youtube.com/@PLENPROJECT>

**Online Simulator For PLEN2:** <http://plen.jp/playground/motion-editor/>

**Poppy Project:**

****

**Processor:**

Poppy humanoid robots typically use microcontrollers or single-board computers to manage control, sensor integration, and communication. Depending on the model and version, the following are examples of processors that might be used:

Raspberry Pi: Raspberry Pi boards are popular choices for their affordability and versatility. They can run Linux-based operating systems and handle tasks ranging from sensor data processing to high-level control.

Arduino: Arduino microcontrollers provide real-time control capabilities. They are often used for interfacing with actuators and processing sensor data.

STM32 Microcontrollers: STM32 microcontrollers from STMicroelectronics are commonly used in robotics projects due to their capabilities in motor control and sensor interfacing.

**Sensors:**

Poppy humanoid robots incorporate various sensors to perceive their environment and interact with it:

Cameras: Cameras are essential for visual perception. They allow the robot to capture images and potentially perform tasks like object recognition and navigation.

Depth Sensors: Depth sensors, such as time-of-flight cameras, enable the robot to perceive distances to objects and create depth maps.

Inertial Measurement Units (IMUs): IMUs, which include accelerometers, gyroscopes, and sometimes magnetometers, provide information about the robot's orientation, acceleration, and angular velocity.

Force/Torque Sensors: These sensors measure forces and torques applied to the robot's limbs or end-effectors, enabling more precise interaction with the environment.

**Actuators:**

Actuators are responsible for generating movement in the Poppy humanoid robots:

Servo Motors: Poppy robots often use servo motors to actuate joints. These motors offer precise control over joint angles and positions.

Pneumatic Actuators: Some Poppy robots utilize pneumatic actuators for fluid and compliant movements, mimicking biological systems.

**Software:**

Software plays a crucial role in controlling and programming Poppy humanoid robots:

Robot Operating System (ROS): ROS is commonly used for robotic software development. It provides tools, libraries, and a framework for building robot applications.

Python: Python is a popular programming language for controlling Poppy robots. It offers an accessible environment for coding and interfacing with sensors and actuators.

C/C++: For real-time control and performance-critical tasks, C and C++ languages might be used to program the microcontrollers.

Simulation Software (CoppeliaSim): Simulation software like CoppeliaSim (formerly V-REP) allows developers to test and simulate robot behaviors in a virtual environment before deploying them to the physical robot.

Poppy Libraries: The Poppy Project provides its own libraries and resources tailored for programming and controlling their humanoid robots.

**Website:** <https://www.poppy-project.org/en/>

**CoppeliaSim(V-Rep):** [Downloads - Coppelia Robotics](https://www.coppeliarobotics.com/downloads)

**UNITY 3D**



Unity serves as a versatile platform for creating graphical user interfaces (GUIs) tailored to robot control. Leveraging its 3D rendering and UI capabilities, Unity enables the development of user-friendly interfaces for robot operations. These interfaces can be customized, integrated with robotics APIs, and deployed across various platforms. Unity's simulation features aid in testing and training, while its visualization tools enhance data representation from robot sensors. Overall, Unity empowers developers to design effective GUIs that facilitate real-time control, monitoring, and interaction with robots.

**How to get started with Unity:**

1. Installing Unity Hub and Unity Editor: Download and install Unity Hub, a tool that helps you manage different versions of Unity. Through Unity Hub, you can install and launch the Unity Editor, where you'll create and edit your projects.

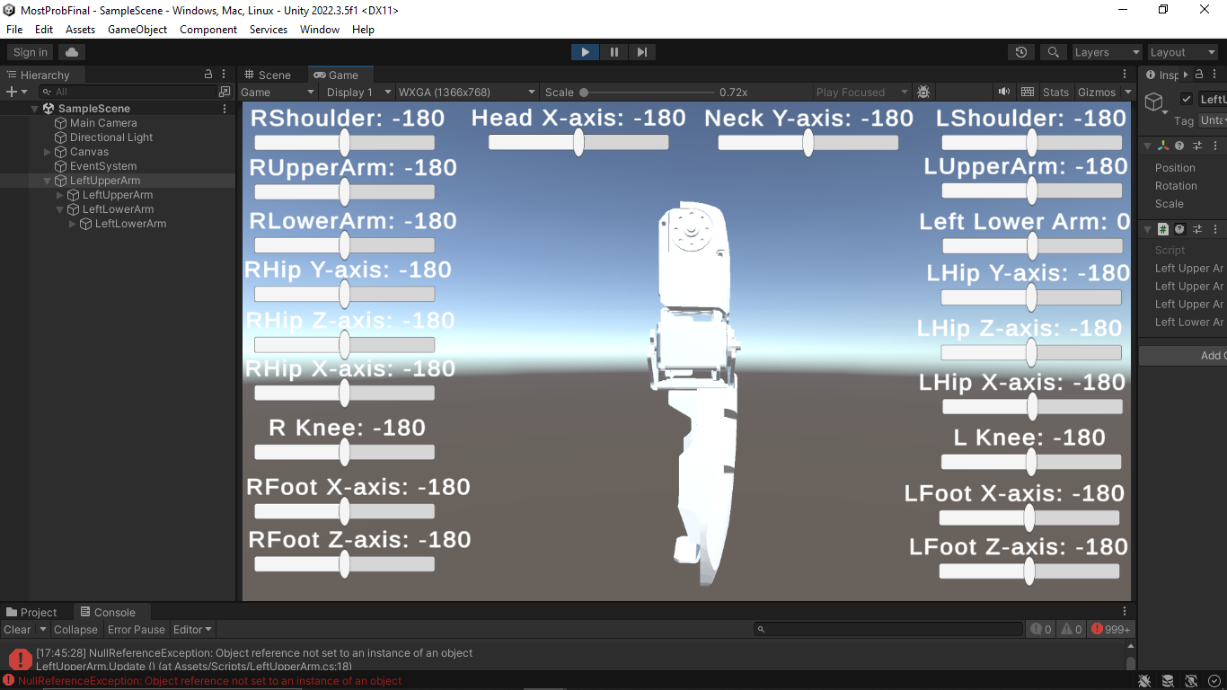
Download link: [Start Your Creative Projects and Download the Unity Hub | Unity](https://unity.com/download)

1. Programming Fundamentals: Learn the basics of programming and scripting. Unity uses C# as its primary scripting language, so understanding concepts like variables, loops, conditionals, and functions is crucial.
2. C# Programming: Develop a good understanding of C# programming language syntax and concepts. Learn about classes, objects, methods, and inheritance, which are essential for scripting in Unity.

Refer: <https://www.youtube.com/watch?v=N775KsWQVkw&list=PLPV2KyIb3jR4CtEelGPsmPzlvP7ISPYzR&pp=iAQB>

**Using Unity to control 3D model of Darwin Op2:**

To control the 3D model of Darwin we used sliders to control different joints of robot. Darwin consists of 20 DOF. But for the same to do it was important to learn how to integrate the C# code with different sliders for all joints and to move the parts in different axis according to the slider value.

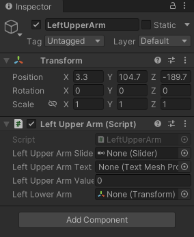


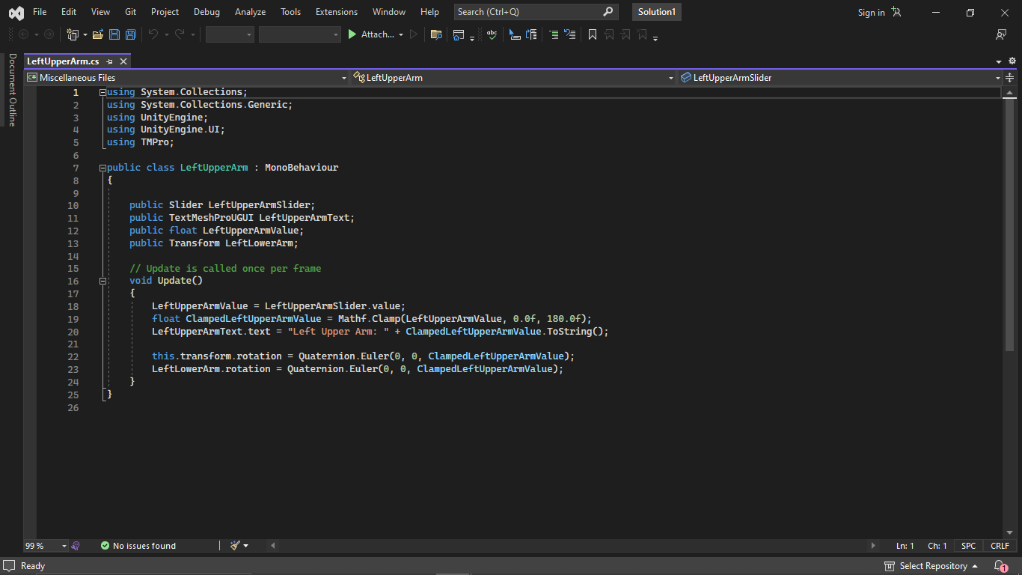
1. **Adding Sliders in Unity:**

* Create Canvas: Add a Canvas to your scene if you don't have one (GameObject > UI > Canvas).
* Add Slider: Inside the Canvas, create a Slider (GameObject > UI > Slider).
* Position and Customize: Adjust the Slider's position, size, and values in the Inspector.
* Optional Text: Add a Text element to display the value (GameObject > UI > Text).

1. **Integrating Slider with The Object:**

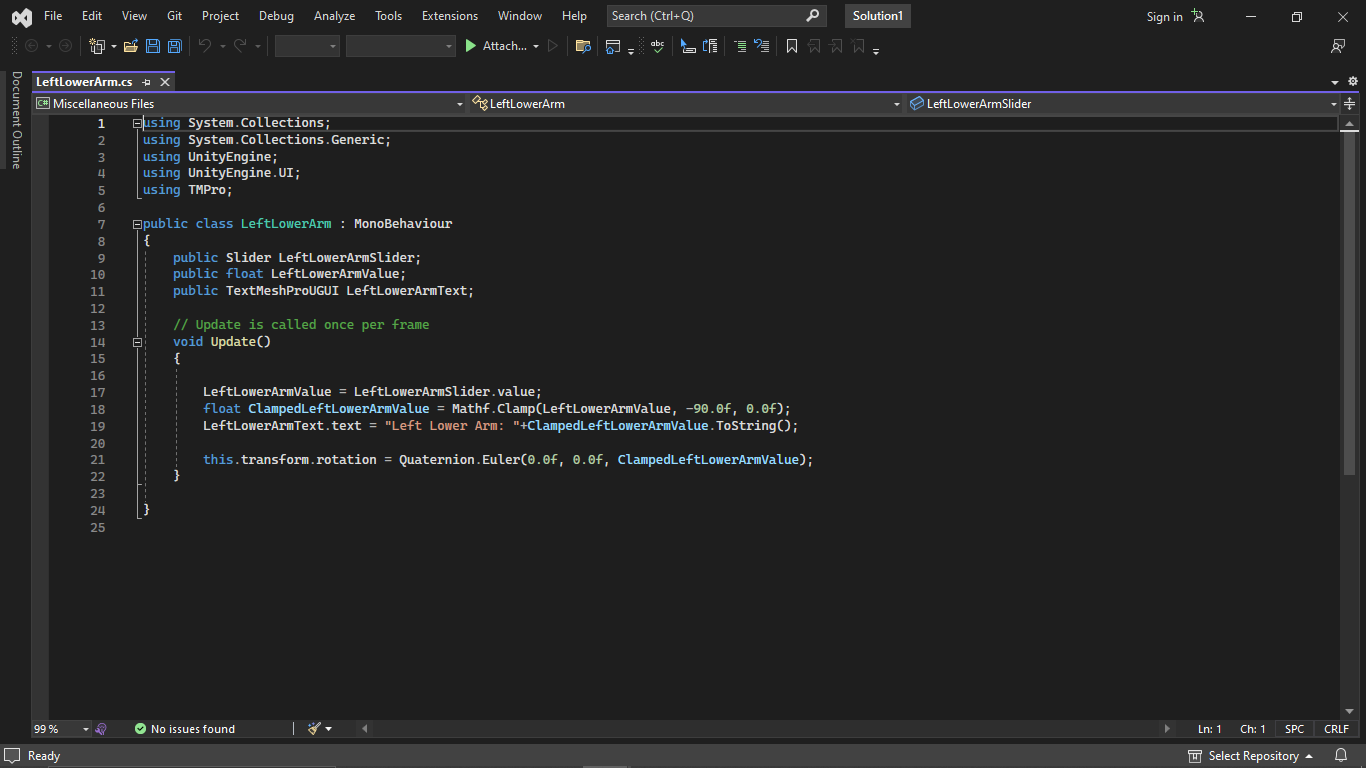
Integrating a slider with an object in Unity involves linking the slider's value to a specific property of the object, allowing real-time adjustments to affect the object's behavior, such as its size, speed, or intensity.





1. **Parent-Child Relationship:**

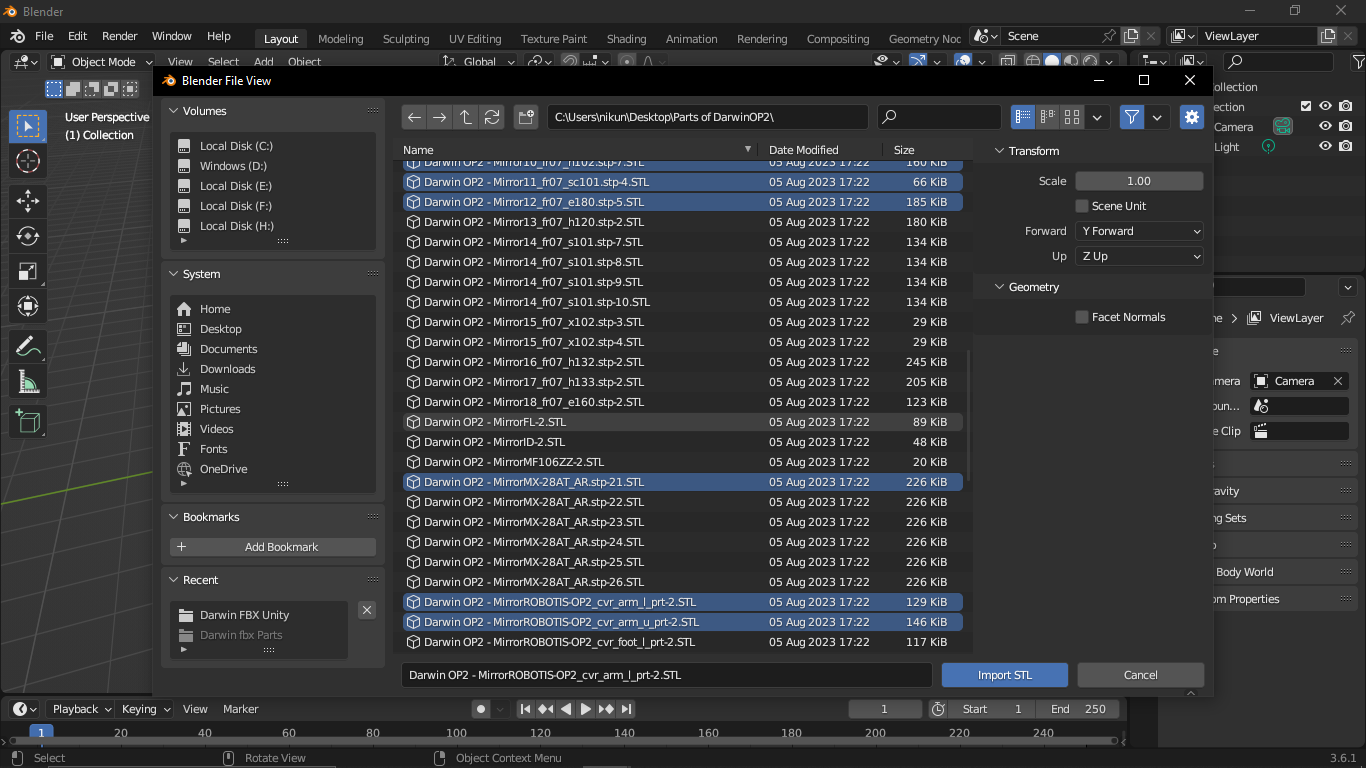
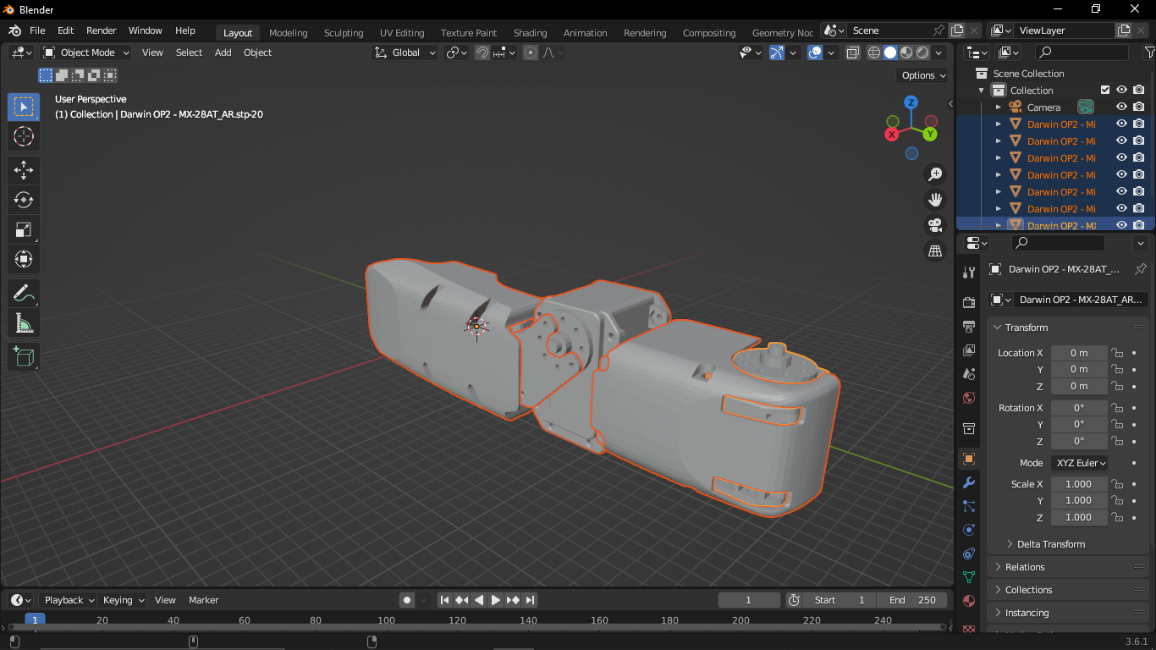
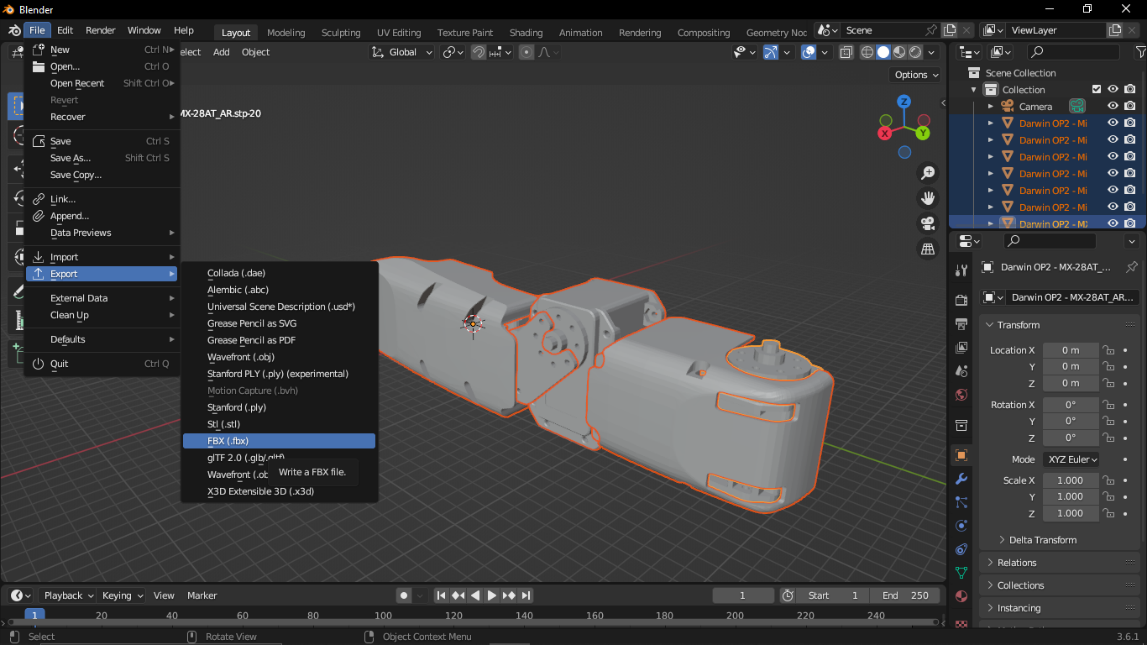
In Unity, the parent-child relationship is a fundamental concept where game objects can be connected in a hierarchical manner. When one game object is designated as the parent of another, the child object's position, rotation, and scale become relative to the parent. This means that if the parent object moves or rotates, its child objects move or rotate with it, maintaining their spatial relationship.



1. **Solid works and Blender:**

Transferring specially crafted parts from SolidWorks to Blender involves importing .stlasm files. Once in Blender, these parts are organized into logical groups like Head, Lower Arm, and Upper Arm, with attention to proper orientation. This step is crucial for creating a clear structure.

Once the parts are arranged, they are readied for Unity by converting them into .fbx files, which Unity understands well. These .fbx files retain the organized groups and orientations, making them suitable for Unity's animation and control features. This process ensures a smooth transition from design to interactive experiences.



**ROBOT OPERATING SYSTEM (ROS)**

Robot Operating System (ROS) is an open-source middleware framework specifically designed for robotics software development. ROS provides a collection of tools, libraries, and conventions that simplify the process of creating complex robot applications. It's not an operating system in the traditional sense, but rather a software framework that runs on top of a traditional operating system.

**ROS AND GAZEBO INTEGRATION:**

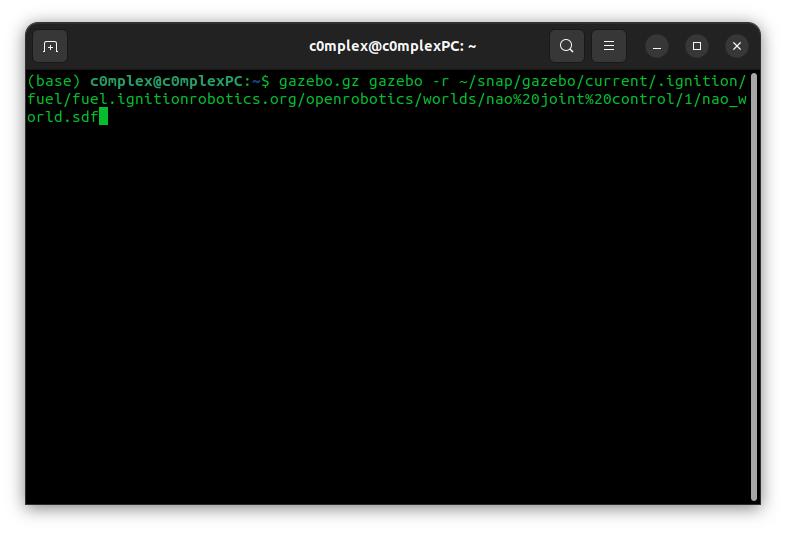
ROS (Robot Operating System) and Gazebo are two separate but highly complementary tools used in robotics development, often integrated together to provide a comprehensive simulation and development environment for robotic systems.

ROS is a flexible and powerful framework for developing robot software. It provides a set of tools, libraries, and conventions that enable communication and coordination between different components of a robotic system. ROS simplifies tasks like hardware abstraction, communication between nodes, and data processing.

Gazebo is a popular open-source robotics simulator that allows you to simulate and visualize complex robotic systems in a virtual environment. It provides a physics engine for realistic interactions, sensors for simulating data from cameras and other devices, and a variety of pre-built models and environments.

ROS and Gazebo are integrated to provide a comprehensive solution for robotics development, offering a seamless transition from software development and testing to real-world robot deployment. The combination of ROS's powerful communication and control capabilities with Gazebo's realistic simulation environment makes it a valuable tool for roboticists, researchers, and hobbyists alike.

**Gazebo Snap:**





Above is installation and Nao simulation.

**Gazebo and ROS installation:**

Refer link: <https://classic.gazebosim.org/tutorials?tut=ros_installing&cat=connect_ros>

**Useful links:**

MoveIT:  <https://moveit.picknik.ai/main/index.html>

This library is a super helpful tool for robots. It makes robots move smoothly from one spot to another, sort of like guiding them on a clear and perfect path. It's like giving a robot a map with a starting point and an ending point, and MoveIt figures out the best way for the robot to travel between those points in URDF files.  
  
YouTube Video: <https://www.youtube.com/watch?v=EosEikbZhiM>   
  
YouTube Channel: <https://www.youtube.com/@ArticulatedRobotics>

These YouTube links are a great way to see how ROS, Gazebo, and MoveIt work together for robots. These videos show you visually how everything fits, making it easier to understand how they control and move robots.

System on Chip compatible with ROS (Robot operating system)

Raspberry Pi 4

1. 40 pin GPIO header
2. CPU Broadcom Quad core A72 (ARMv8-A), 64-bit CPU @1.5GHz
3. 1, 2, 4 GB versions
4. Bluetooth 5
5. Full gigabit ethernet.
6. A pair of 3.0 and a pair of 2.0 ports
7. Dual micro-HDMI ports
8. Supported communication protocols- UART, SPI, I2C, and USB
9. Cost – I.N.R 7719 (Amazon)
10. Robotic Applications: Suitable for a wide range of robotic applications, especially those requiring a balance between performance and affordability.

To install ROS on Raspberry pi 4 refer –

1. ROS- [ROSberryPi/Installing ROS Kinetic on the Raspberry Pi - ROS Wiki](http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi)
2. ROS2- [ROS 2 on Raspberry Pi — ROS 2 Documentation: Foxy documentation](https://docs.ros.org/en/foxy/How-To-Guides/Installing-on-Raspberry-Pi.html)



Jetson Nano

1. Quadcore ARM Cortex A-57
2. 40 pin expansion headers
3. GP10 python library
4. Ethernet (USB 3)
5. HDMI and DP display
6. Barrel jack
7. UART console
8. Micro ID
9. Supported by Linux4Tegra
10. Cost- INR 19014(Amazon)\*
11. I2C, SPI, UART, PWM, CAN
12. Robotic Applications: Ideal for AI and computer vision tasks in robotics due to its dedicated GPU and AI-focused features.

6/6

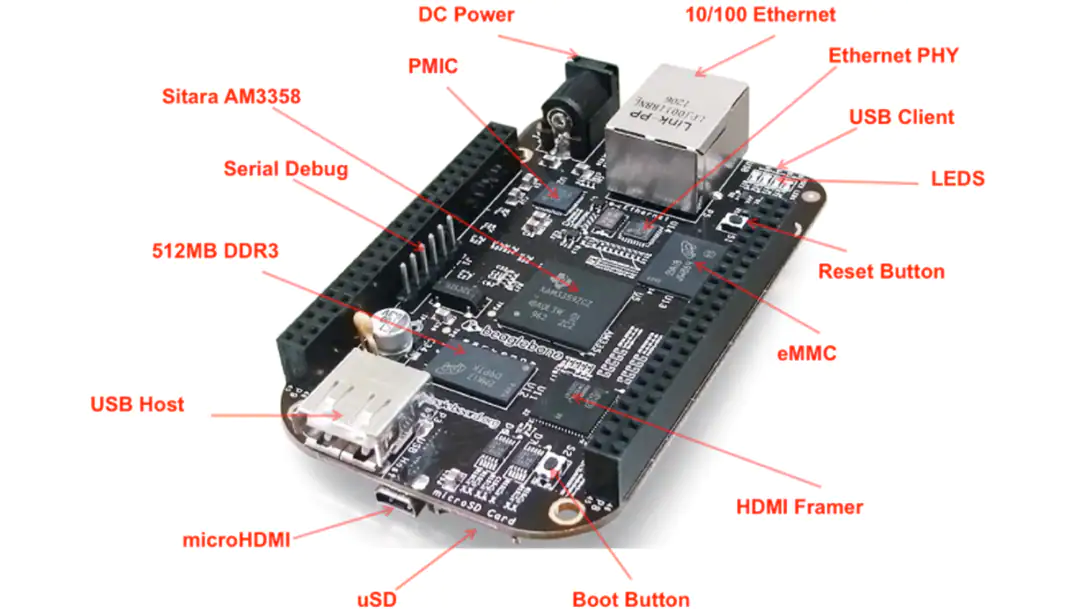
Odroid XU4



1. $53 (₹13,513 on Amazon)
2. Octa core ARM Cortex A-15 and A-7
3. 2GB LPDDR3 RAM
4. eMMC5.0
5. 2Xusb 3.0, 1x USB2.0
6. GB ethernet port
7. HDMI
8. Ubuntu
9. I2C, SPI, UART, PWM, CAN, ethernet
10. Robotic Applications: Suitable for various robotic applications, including tasks that require higher computing power.

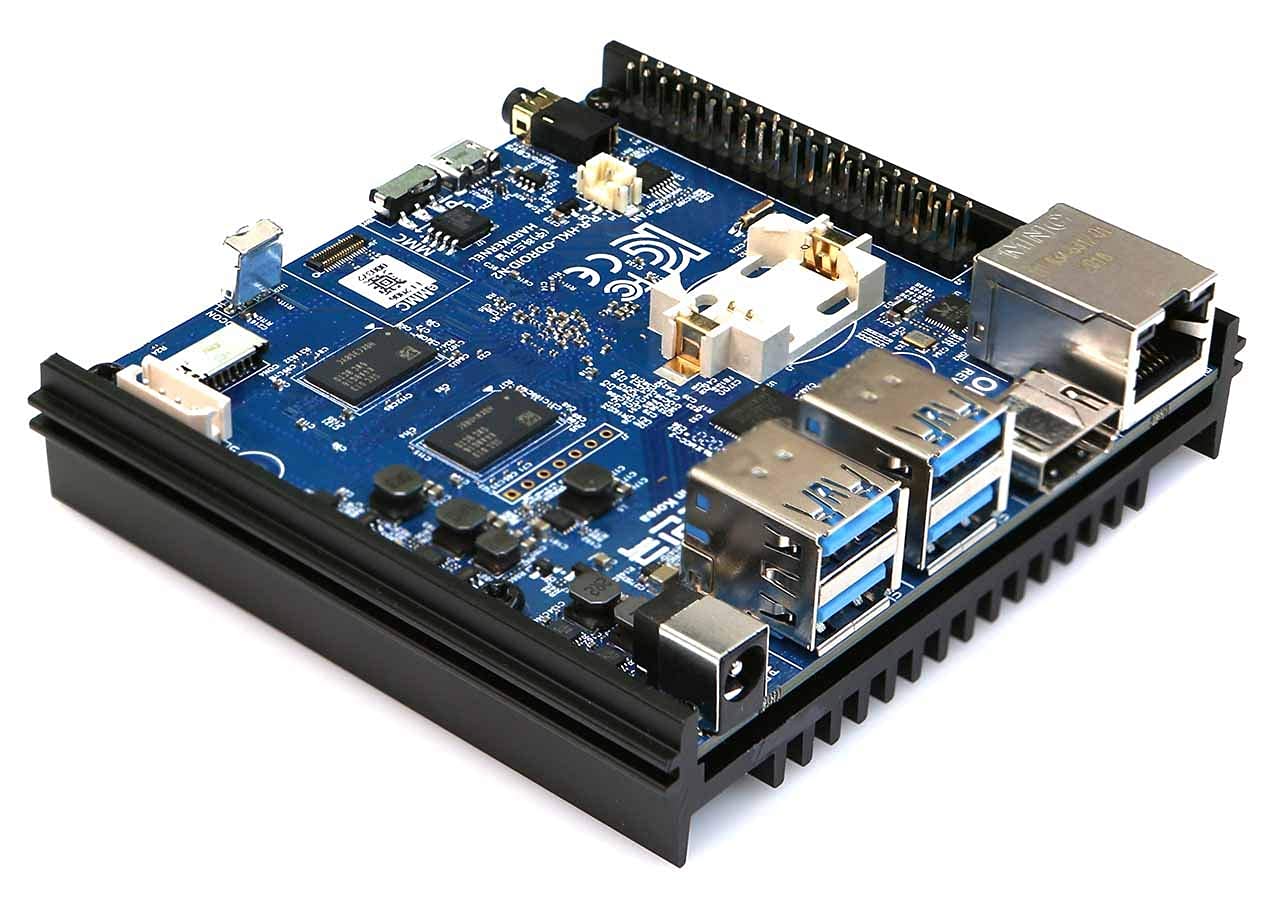
Beagle Bone Black

1. Single Core ARM Cortex A-8
2. 1GHz ARM Cortex A-8
3. 1HDMI Mini
4. UART, I2C, SPI, ADC, eMMC
5. ₹6285
6. DDR3 Memory
7. 1 x USB 3.0, 1X USB 2.0
8. Communication Protocols: Gigabit Ethernet, UART, SPI, I2C
9. Robotic Applications: Suitable for smaller robotic projects and learning purposes, though limited in performance compared to other options.



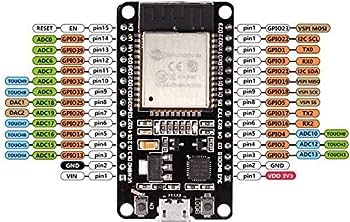
Odroid N2+

1. Hexacore
2. 950 MHz
3. $66-$83
4. Heat sink and case
5. No built in WI-FI, Bluetooth
6. 4 USB 3.0 ports
7. I.R
8. Robotic Applications: Well-suited for various robotic tasks that require higher computational capabilities.



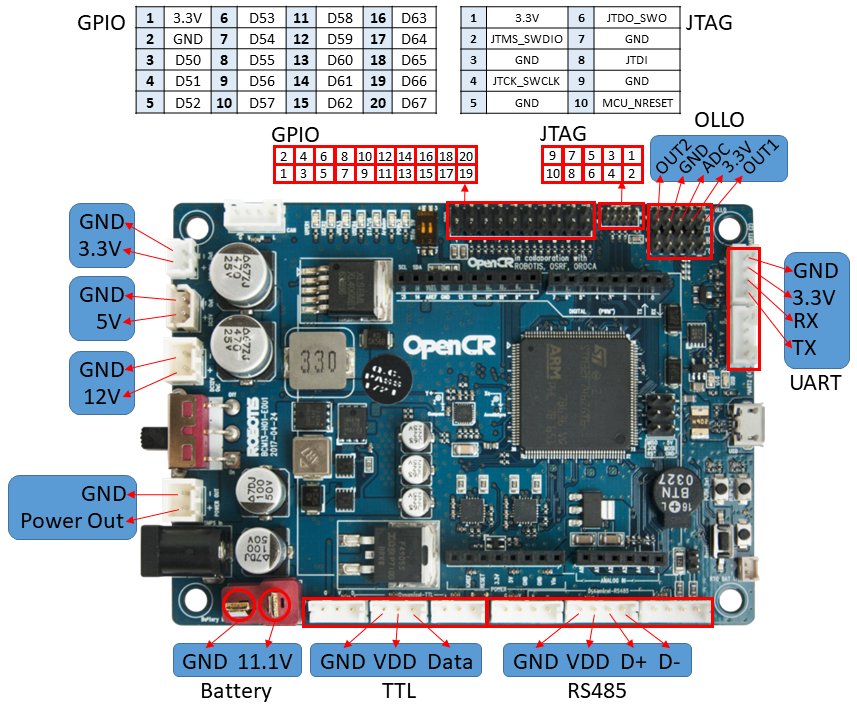
ESP32:

1. Microcontroller: ESP32-D0WDQ6 (dual-core, 32-bit MCU)
2. Clock Speed: Up to 240 MHz
3. Flash Memory: 4 MB
4. RAM: 520 KB
5. GPIO Pins: Varies based on board variant
6. Interfaces: UART, I2C, SPI, WiFi, Bluetooth
7. Supported Communication Protocols: UART, I2C, SPI, WiFi (802.11b/g/n), Bluetooth (Classic and BLE)
8. Compatibility with ROS: ROS2 (Foxy and later versions)



Robotis OpenCR 1.0:

1. Microcontroller: STM32F746ZG (ARM Cortex-M7 core)
2. Clock Speed: 216 MHz
3. Flash Memory: 1 MB
4. RAM: 340 KB
5. GPIO Pins: Varies based on board design
6. Interfaces: UART, I2C, SPI, Dynamixel Bus
7. Supported Communication Protocols: UART, I2C, SPI, Dynamixel Protocol
8. Compatibility with ROS: ROS (via rosserial or custom integration)



Teensy 4.1:

1. Microcontroller: NXP i.MX RT1062 (ARM Cortex-M7 core)
2. Clock Speed: 600 MHz
3. Flash Memory: 8 MB
4. RAM: 1 MB
5. GPIO Pins: 39
6. Interfaces: UART, I2C, SPI
7. Supported Communication Protocols: UART, I2C, SPI
8. Compatibility with ROS: ROS (via rosserial or custom integration) 