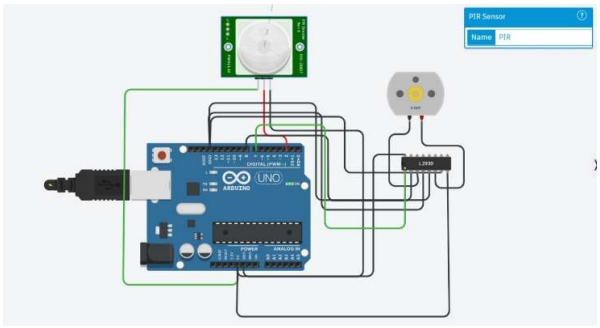
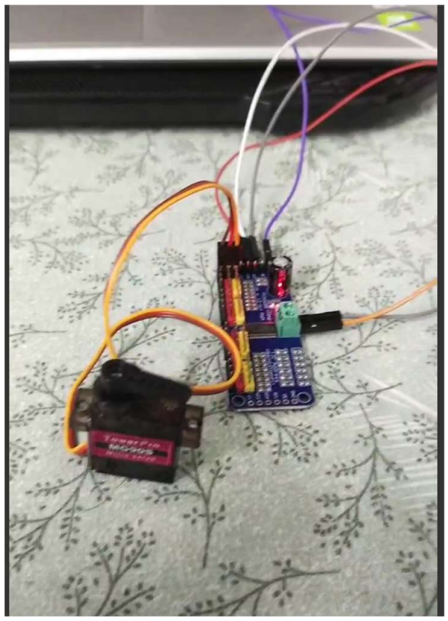
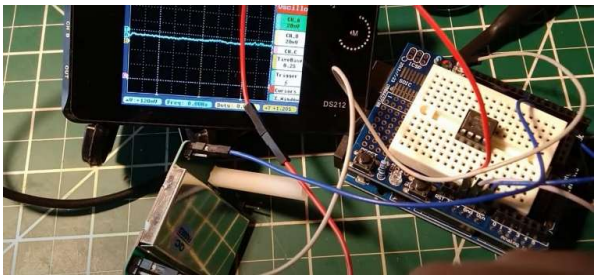
WEEK 6 WEEKLY REPORT:

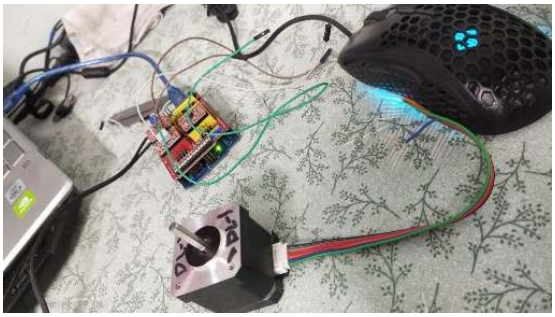
In Week 6, we seamlessly merged art and technology, finalizing the project's core integration. Intricate circuits were meticulously calibrated to synchronize with vibrant artistic elements, ensuring sensors, lights, and interactive visuals harmonized flawlessly. Collaborative creativity bridged aesthetics and functionality, resulting in a cohesive, immersive experience. Testing affirmed stability, marking a milestone where innovation met expression in a dynamic, multisensory installation.

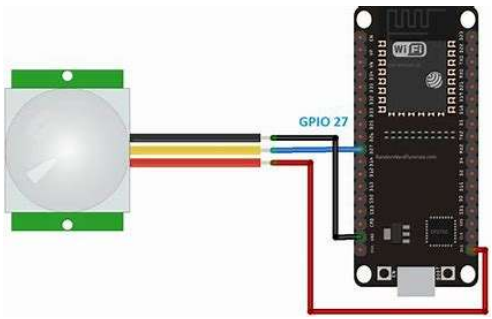
OVERALL CONNECTION











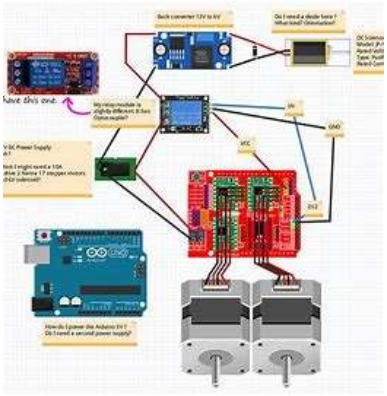
Connecting an ESP32 with a camera for computer vision (CV) involves integrating hardware and software to enable real-time image processing. The ESP32 microcontroller, paired with a compatible camera module (e.g., OV2640), requires precise wiring of GPIO pins for power, data transmission (D0-D7), VSYNC, HREF, PCLK, and XCLK. Libraries like the ESP32 Camera Driver are configured to initialize the camera, adjust resolution (e.g., UXGA, SVGA), and manage frame rates. The captured image data is stored in a buffer, which can be processed using lightweight CV algorithms (e.g., edge detection, motion tracking) or transmitted wirelessly via Wi-Fi for remote analysis. Challenges include optimizing memory usage, reducing latency, and ensuring stable communication between the camera and ESP32. Proper power management and noise reduction are critical to avoid signal interference. Applications range from

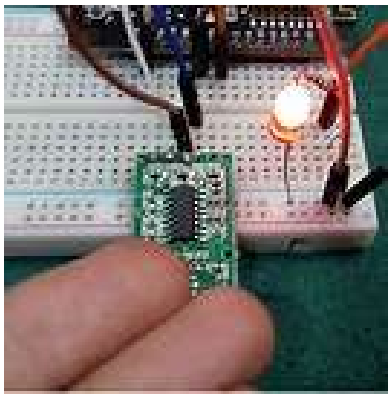
surveillance systems to interactive IoT projects, leveraging the ESP32's low-cost, low-power capabilities for embedded vision solutions.

Connecting a microwave Doppler sensor involves integrating it into a circuit to detect motion or velocity using the Doppler effect. Begin by identifying the sensor's pins: typically, a power supply (5-12V DC), ground, and an output signal pin. Connect the power and ground to a stable voltage source, ensuring correct polarity to avoid damage. The output pin, which carries the detected frequency shift as a voltage signal, should be linked to a microcontroller or amplifier for processing. Incorporate filtering components, such as capacitors or resistors, to minimize noise interference. Calibrate the sensor by adjusting its sensitivity and detection range, often via onboard potentiometers or software settings. Ensure proper shielding and positioning to avoid false triggers from environmental factors. Test the setup by observing signal changes in response to movement, verifying real-time responsiveness and accuracy for applications like security systems, automatic lighting, or speed measurement.

Connecting a passive infrared (PIR) sensor involves integrating its three primary pins: power (VCC), ground (GND), and output (OUT). The VCC pin is typically connected to a 5V power source, while GND links to the system's common ground. The OUT pin, which sends a high signal (3.3V or 5V) upon detecting motion, is wired to a microcontroller's digital input pin (e.g., Arduino's GPIO). Adjusting onboard potentiometers allows calibration of sensitivity and detection delay. Proper placement ensures optimal coverage, avoiding obstructions or direct interference from heat sources. A pull-down resistor may be added to stabilize the signal, ensuring reliable motion-triggered responses in applications like security systems or automated lighting.

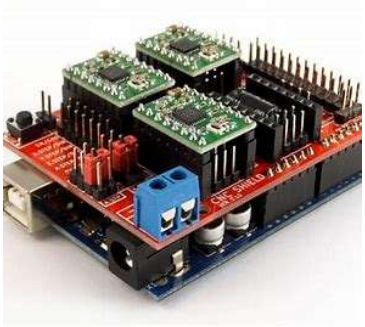
Connecting a servo motor to an ESP microcontroller, such as an ESP32 or ESP8266, involves three primary components: power, ground, and signal. The servo's power (VCC) and ground (GND) pins are connected to a stable 5V power source, preferably an external supply, to avoid overloading the ESP's onboard regulator. The signal wire (typically yellow or orange) links to a GPIO pin capable of pulse-width modulation (PWM), such as GPIO2 or GPIO5. Using the Arduino IDE, the Servo library simplifies control by generating PWM signals to rotate the servo (0°–180°). Ensure common grounding between the ESP and power supply to prevent voltage discrepancies. For precise movement, calibrate the servo's pulse width in code and avoid abrupt voltage drops by using capacitors or a dedicated servo shield for multiple motors.





*Connecting a stepper motor to an Arduino involves several key steps. First, identify the motor type (unipolar or bipolar) and use a compatible driver module, such as the ULN2003 for smaller motors or the A4988/DRV8825 for precise control. Connect the driver's control pins (e.g., IN1-IN4 for ULN2003 or STEP/DIR for A4988) to the Arduino's digital pins. Power the motor separately via an external supply, as stepper motors typically require higher voltage/current than the Arduino's 5V output. Use the Arduino IDE's Stepper library or third-party libraries like *Backstepped* to program movement sequences, defining steps per revolution, speed, and direction. Ensure proper grounding between the Arduino, driver, and power supply to avoid noise or damage. Test the circuit incrementally, verifying wiring and code logic to prevent overheating or missed steps. Adjust micro stepping settings (if supported) for smoother motion. Always double-check datasheets for pin configurations and current limits to ensure compatibility and safe operation.*

CNC SHIELD:



A CNC shield is an expansion board designed to interface microcontrollers (like Arduino or ESP-based boards) with CNC machines, 3D printers, or laser cutters. It simplifies wiring by providing dedicated slots for stepper motor drivers (e.g., A4988, DRV8825), which control individual stepper motors for precise axis movement. The shield routes power, step/direction signals, and enables/disables pins between the microcontroller and drivers. It typically includes terminals for motor power (12-24V), spindle control (PWM), and end-stop switches for homing. Some versions offer additional features like cooling fan headers or probe inputs. Compatibility depends on pin mapping; Arduino-focused shields may require firmware adjustments (like GRBL) when used with ESP boards. Proper grounding, current limiting via driver potentiometers, and heat management are critical to avoid motor stalling or driver overheating. This modular setup streamlines complex motion control projects by centralizing connections and enabling scalable multi-axis automation.

HOW TO CONNECT 6 WIRE STEPPER MOTOR TO CNC SHIELD:

To connect a 6-wire stepper motor to a CNC shield, first identify the motor's coils using a mustimeter. A 6-wire motor has two coils with centre taps: measure resistance between wires to find pairs with equal resistance (these are the coil ends) and those with half the resistance (centre taps). Ignore the centre taps (leave them disconnected) and use the four coil ends (A+, A-, B+, B-). Connect these to the corresponding driver on the CNC shield (e.g., X-axis driver pins: 1A, 1B, 2A, 2B). Ensure the driver (e.g., A4988/DRV8825) is configured for bipolar mode. Power

the motor via the shield's external supply, matching the motor's voltage/current rating. Adjust the driver's current limit using its potentiometer to prevent overheating. Test with a simple motion code, verifying direction and step consistency. Double-check wiring against the motor's datasheet to avoid misconnections.