MIMEH

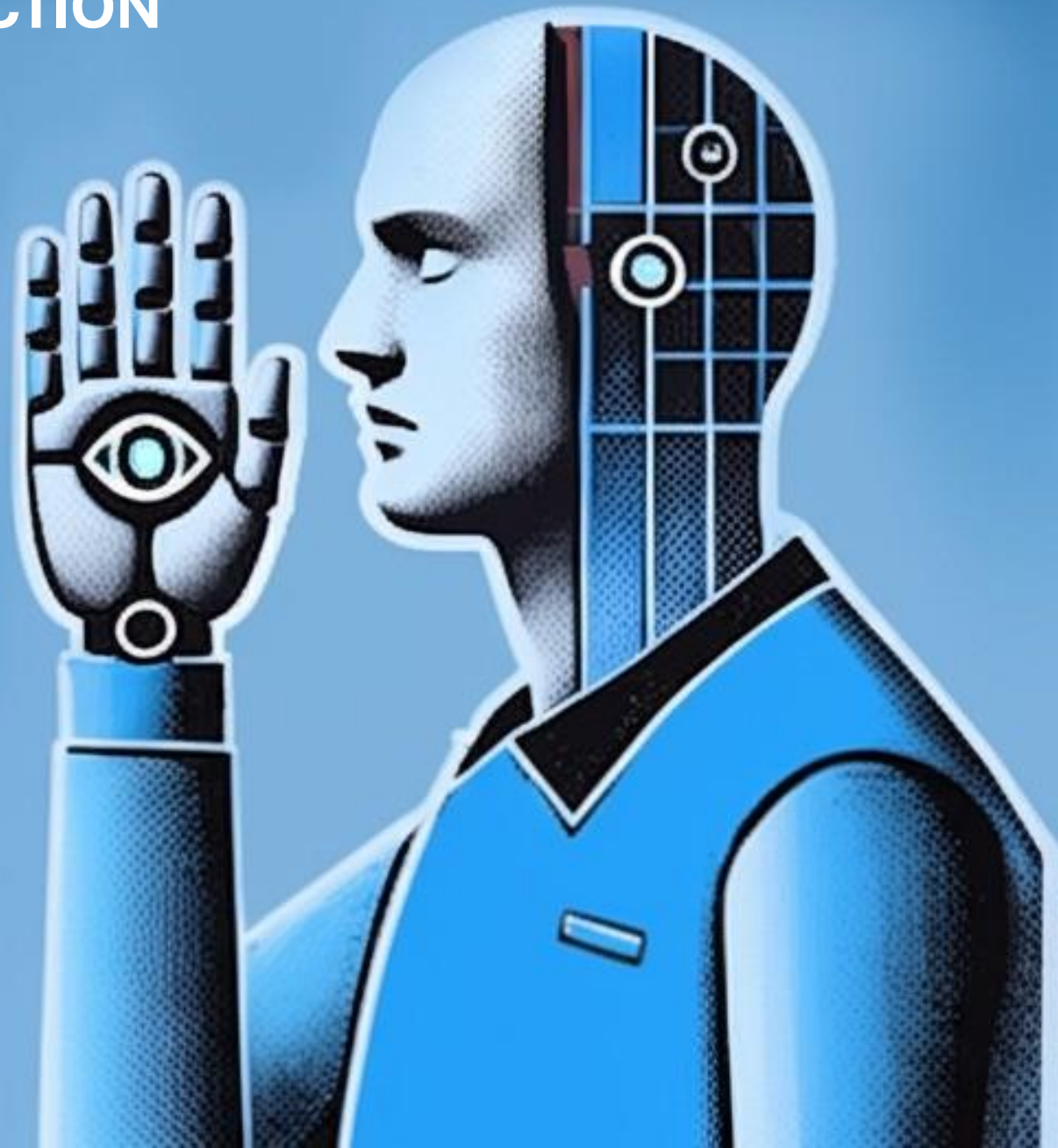
Embedded Software for the Internet of Things Project

Assoc. Prof. Kasim Sinan Yildirim

Group 10: *De Marco Matthew, Lo Iacono Andrea, Pezzo Andrea*

INTRODUCTION

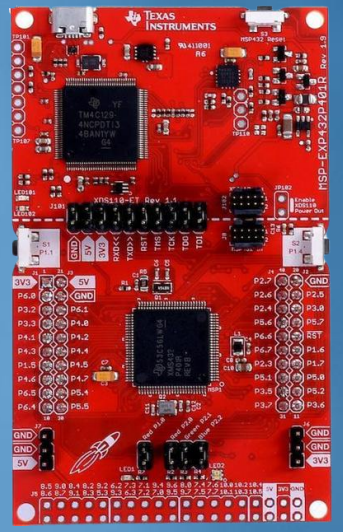
M.I.M.E.H. (*Microcontroller-Integrated Motion Embedded-software Hand*) is an embedded software application designed to control and manage a physical prosthetic hand, whether actively used by an individual or as an external device. The core functionality of this application is to enable precise finger control using two development boards: the ***Texas Instruments MSP432P401R LaunchPad*** **with the** ***Educational BoosterPack MKII*** and the ***Arduino R4 MINIMA***. Most importantly, it provides intelligent gesture replication through Python's ***MediaPipe*** library, ensuring seamless and responsive hand movement.

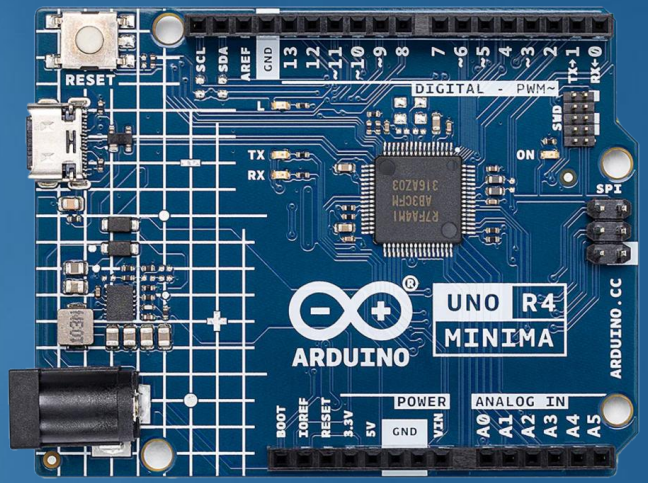


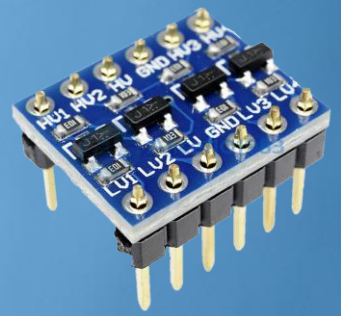
HARDWARE AND SENSORS USED

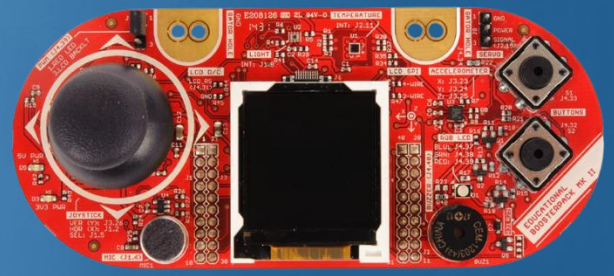
This project required some hardware components to be carried out:

- ***Arduino R4 Minima***;
- ***TI MSP432P401R*** **and** ***TI Educational BoosterPack MKII***;
- ***Logical level shifter***;
- ***Breadbords***;
- ***Wires***;
- ***5x Servo motors***;
- **3D-printed hand model** (open-source design from *InMoov*)
- ***Personal Computer***.

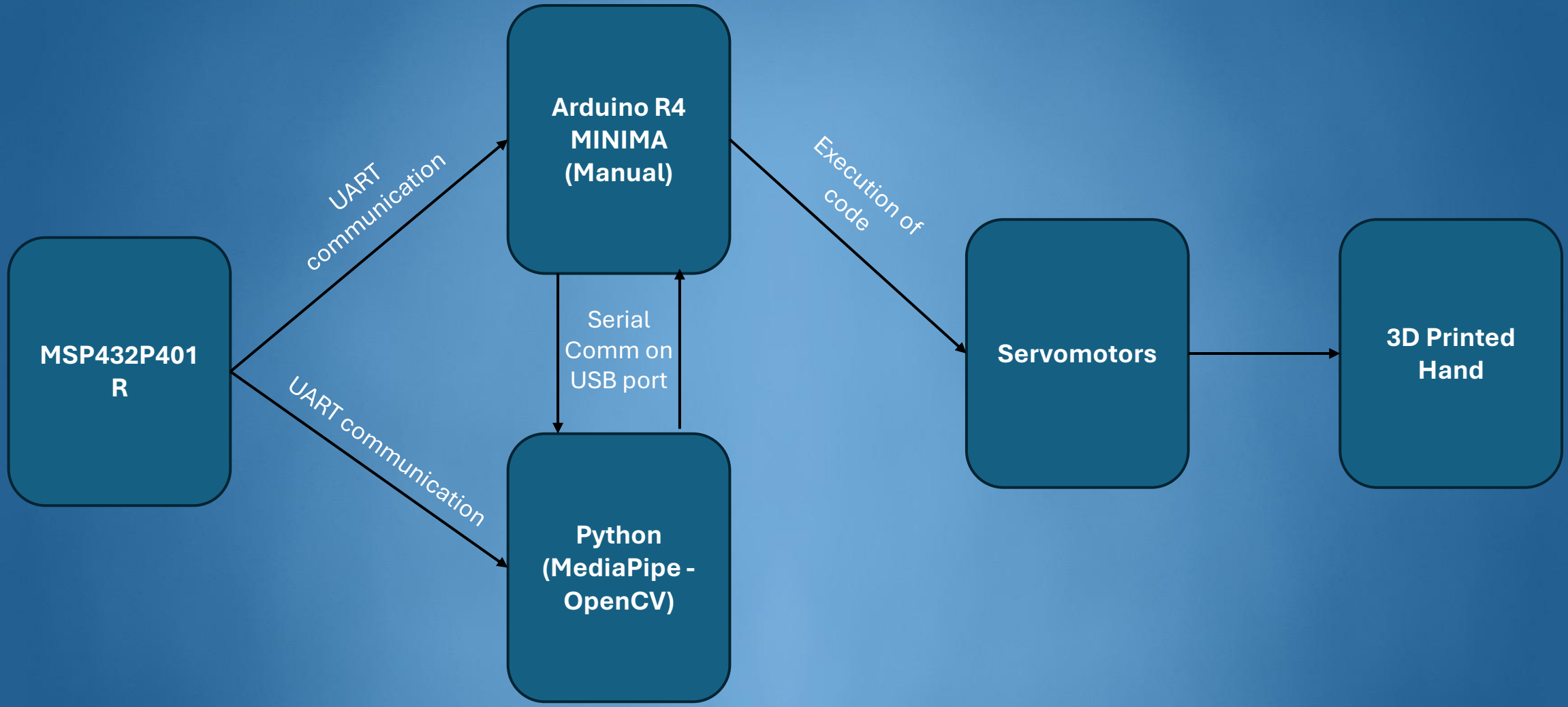








WORKFLOW AND USER INTERACTION



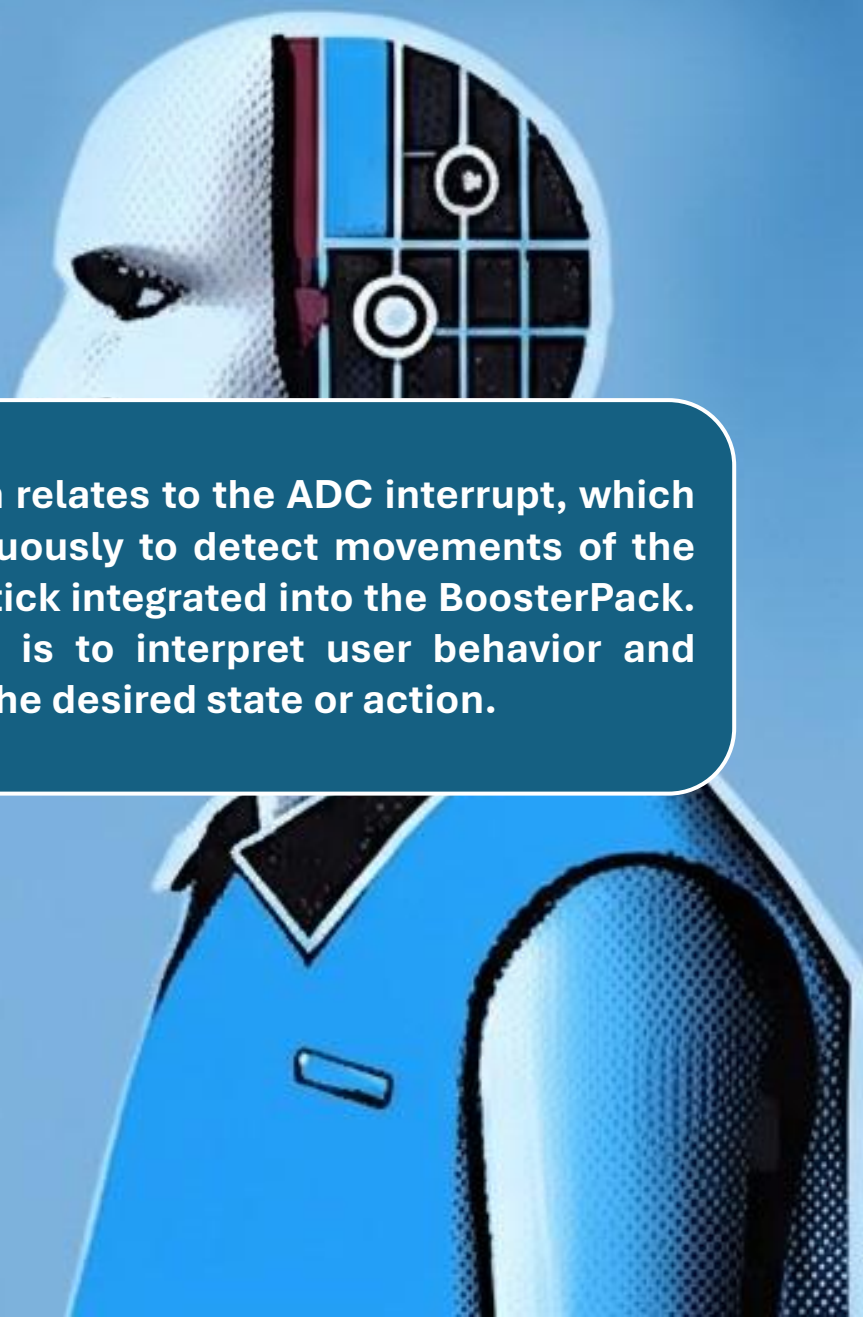
IMPLEMENTATIONS

```
void ADC14_IRQHandler(void)
{
    uint64_t status;
    status = ADC14_getEnabledInterruptStatus();
    ADC14_clearInterruptFlag(status);

    /* ADC_MEM1 conversion completed */
    if(status & ADC_INT1)
    {
        /* Store ADC14 conversion results */
        resultsBuffer[0] = ADC14_getResult(ADC_MEM0);
        resultsBuffer[1] = ADC14_getResult(ADC_MEM1);

        if(isControllingServo)
        {
            if(resultsBuffer[1] > JOY_Y_HIGH_SERVO || resultsBuffer[1] < JOY_Y_HIGH_SERVO)
           {
               //The user has selected one of the options
               //The interrupt can be disable at the moment
               ADC14_disableInterrupt(ADC_INT1);  // Disable specific ADC MEM1 interrupt

           }
           if(resultsBuffer[0] < JOY_X_LOW)
           {
               //Exit from the sections
               ADC14_disableInterrupt(ADC_INT1);  // Disable specific ADC MEM1 interrupt
           }
        }
        else
```



This section relates to the ADC interrupt, which runs continuously to detect movements of the analog joystick integrated into the BoosterPack. Its purpose is to interpret user behavior and determine the desired state or action.

IMPLEMENTATIONS

```
if (UART_getInterruptStatus(UART_MODULE2, EUSCI_A_UART_RECEIVE_INTERRUPT)) {
    char receivedChar = UART_receiveData(UART_MODULE2); // Read received character

    if (receivedChar == '\n') { // End of message
        receivedData[index] = '\0'; // Null-terminate string
        int num = atoi(receivedData);   // Convert string to integer

        if(currentState==MANUAL)
        {
            printf("Received: %d\n", num); // Print received data
            //Numbers correspond to the Fingers
            if(num >= 1 && num <= 5)
            {
                //Meaning the Arduino is letting control the servos to MSP
                isControllingServo = 1;
            }
        }
```

This code handles an EUSCI_A2_BASE interrupt, triggered by a response from the Arduino after a message was sent via EUSCI_A1_BASE. If the response falls within a specific range, it indicates that the connection is fully operational, allowing the MSP to control the servos (fingers).

```
if(Serial.available() > 0 && isVisionConnected)
{
  // Read the servo angles (they are sent as a formatted string)
  int thumbAngle = Serial.parseInt();
  int indexAngle = Serial.parseInt();
  int middleAngle = Serial.parseInt();
  int ringAngle = Serial.parseInt();
  int pinkyAngle = Serial.parseInt();
  // Move the servos based on the received angles
  thumbServo.write(thumbAngle);
  indexServo.write(indexAngle);
  middleServo.write(middleAngle);
  ringServo.write(ringAngle);
  pinkyServo.write(pinkyAngle);
}
```

This code comes from the Arduino IDE and handles serial communication between Arduino and Python via USB. When Python accesses the port, it sends an array of five elements representing real-time angles calculated from the PC's webcam using MediaPipe.

IMPLEMENTATIONS

```python
# Serial Configuration
ARDUINO_PORT = "/dev/cu.usbmodem11301"  # Change this based on your setup
BAUD_RATE = 9600

arduino = None
# Try connecting to Arduino


try:
    arduino = serial.Serial(ARDUINO_PORT, baudrate: 9600, timeout=1)
    time.sleep(2)  # Allow time to establish connection
    print("Connected to Arduino")
except serial.SerialException:
    print("Error: Could not open serial port.")

# Pass the serial instance to controller.py
initialize_serial(arduino)
```

```python
1 usage
def listen_arduino():
    global arduino_thread_running
    while arduino_thread_running:
        try:
            if arduino and arduino.in_waiting > 0:
                command = arduino.readline().decode('utf-8').strip()
                if command == "1":
                    root.after( ms: 0, handle_webcam_processing)
                    break

        except:
            pass
```

These scripts manage the connection between the Arduino and the Python script for gesture recognition.

TESTING & PROBLEMS

The testing approach focused on verifying individual functionalities before integrating them. For instance, before testing **MSP432 communication**, **UART communication** on the designated pins (*3.2* and *3.3*) was examined, revealing that pin 3.3 was not transmitting messages to the **Arduino**. Additionally, a major challenge in board-to-board communication was handling voltage level differences.

Python implementation was initially tested with the **Arduino** alone to verify serial communication, which highlighted an issue where the serial port became occupied if the **Arduino's Serial Monitor** was enabled. The final testing phase involved sending messages from the **MSP432** to the **Arduino** while **Python** monitored the USB port to check if the **MSP432** granted access.

FUTURE IMPROVEMENTS

For future improvements, we plan to integrate the **OV7670 camera sensor**. This sensor requires multiple Arduino pins, making it necessary to expand the already limited pin set of the **Arduino R4 Minima**. The goal is to transmit frames to Python for processing while also displaying the processed frames on the **MSP432's LCD screen**.

Another potential enhancement is implementing a **gesture recording** feature in Python, allowing gestures to be stored on the **MSP board**. This would enable replaying a recorded gesture without the need to control each finger individually in real time.

CONTRIBUTIONS

Matthew De Marco

- Contributed to the Texas Instruments program by developing the operating menu and integrating Python configuration with the Texas Instruments system.

Andrea Pezzo

- Contributed to the hand construction and Arduino implementation.
- Assisted with Python integration and Texas Instruments development.

Andrea Lo Iacono

- Worked on hand construction and Python-Arduino integration.
- Remodeled 3D files for printing hand components.