Control of a Humanoid Robot Arm

H. Pérez Pérez

University CEU San Pablo, Madrid, Spain, h.perez10@usp.ceu.es.

Abstract

The objective of this project is to prepare and calibrate a 3D-printed humanoid arm, focusing on the design choices, limitations, and efforts to improve its functionality. The arm can be divided into three parts: hand, forearm, and upper arm. These parts feature articulated joints at the wrist, elbow, and shoulder. All these joints are actuated by single servos, but finger actuation relies on two servos per finger.
This humanoid arm can be used as a base to learn the basics of controlling servos in the context of human motion or to implement and test remote control, potentially allowing remote control to students at other universities.

1. Introduction

The field of bionics aims to merge the intricacies of biology with the power of technology, and bionic arms epitomize this innovative approach. Designed to closely emulate the form and function of a natural human arm, these advanced prosthetics leverage sophisticated EMG sensors, actuators, and algorithms to seamlessly integrate with the user's commands. By forging a direct connection between the user's intentions and the prosthetic limb, bionic arms offer an unprecedented level of control and fluidity of movement, granting users an increased sense of agency and a profound improvement in their quality of life. Examples of this kind of bionic limbs are the Hero Arm [1] or True Limb [2].

One of the most promising examples of above-the-elbow bionic arms would be the Atom Touch [3], which is described to have a near-full range of motion. This means that it implements the same joints a human arm has, allowing natural movements from the elbow onwards, and includes full finger control: allowing the user to not only perform flexing motion with its fingers but also implement the capacity of splaying each finger.

The main goal of this project is to provide a simple interface that facilitates the control of youbionic's prosthetic arm design. This humanoid arm offers a versatile solution to mimic human movements, providing a cost-effective solution tailored toward students. This will allow them to learn the fundamentals of controlling and emulating human motion.

A combination of an Arduino board and an Adafruit PWM/Servo driver board is defined as the primary controller of the arm servos, facilitating the ability to change the program used. This ease of change means that the platform can be reused for other purposes, such as a base for remote control of humanoid limbs or its use as a practice tool and a working prototype to try out and develop software for future, more intricate projects.

Since the whole arm is composed of 3d printed pieces, it provides a simple and easy to work on platform that can be easily improved upon with newly developed parts to work around its possible limitations and expand its capabilities.

2. Humanoid Arm

The design employed is a complete 3D-printed humanoid arm, using the schematics acquired from youbionic. This humanoid arm (as shown in Figure 1) consists of shoulder, elbow, forearm, wrist, and hand.

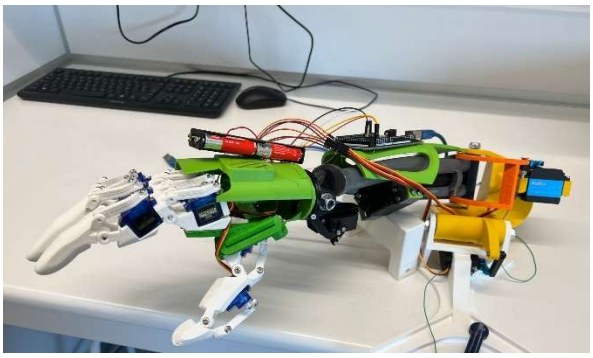

Figure 1. *Humanoid Arm used on the project.*

The arm is actuated by four servos: wrist, forearm, biceps, and shoulder. These servos provide the whole arm with four degrees of freedom, as each servo provides motion in a single plane. In addition to the arm servos, the hand is also articulated by 11 servos. In the case of the thumb, it is actuated by three servos (two of them provide the flexing motion of the finger, and the third one acts as the palm servo); for every other finger, the flexing motion is achieved by two servos, as with the thumb. This means that each finger has two degrees of freedom (as with the arm servos, all of the finger servos only provide motion in a single plane) except for the thumb, which counts with three degrees of freedom since it's actuated by an extra servo.

This project is focused on the hand servos to perform different gestures (Figure 2).

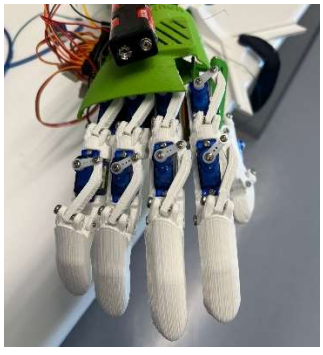

Figure 2. *Robotic hand, composed of 2 servos per finger.*

The arm is actuated using the following components:

- Arduino Mega 2650 (Rev3)
- Adafruit PWM/Servo driver board (PCA9685)
- 11 Micro Servos SG90
- 2 FeeTech Digital Servos FS5323M
- 2 FeeTech 15kg.cm Servos

In order to control these servos, a combination of an Arduino board (Arduino Mega 2560 R3) and an Adafruit 12-bit PWM/Servo driver board is used (Figure 3).

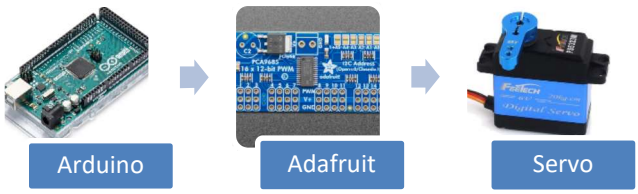

Figure 3. *Control diagram of the servos.*

All these servos are directly connected to the Servo driver board, as depicted in Figure 4, which acts both as the control interface between the servos and the Arduino Mega, and the power source for the servos. To provide power for the servos, the arm uses a four 1.5V battery pack, for a total of 6V.

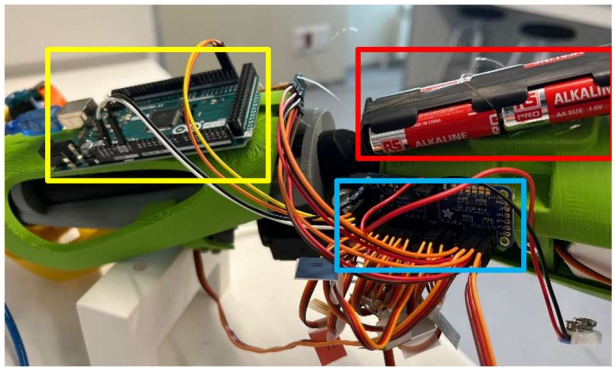

Figure 4. *Actual wiring summarized in fig3. Yellow rectangle: Arduino Board; Blue rectangle: Servo Driver with all servos connected; Red rectangle: battery pack used.*

3. Description

As previously stated, the main goal of this project is to provide an interface to control the arm servos. To achieve this goal, the capability to execute a diverse range of hand and arm movements is required. To perform these movements correctly, all the servos must be calibrated, which will be explained in further detail in the following subsection.

3.1. Servo Calibration

The process of calibrating servos involves adjusting them to move the correct amount when a specific input signal is received. This is done by changing the duration (width) of the pulse signal till the desired range of motion for the servo is obtained. This is because the servo position is controlled by the length of the pulse signal sent (Figure 5). A 1500µs PWM signal will position the servo in the middle position, 90º, increasing the signal length all the way up to 2500µs will position the servo at its max range, 180º; while decreasing the pulse length to 500µs will set the servo position to its minimum value, 0º. This can be a very time-consuming process, but it's essential for the arm to be able to perform tasks accurately.

```
#define SERVOMIN_index_A  122 // index outer servo min          YELLOW1   id6
#define SERVOMAX_index_A  409 // index outer servo max          YELLOW1   id6
#define SERVOMIN_index_B  276 // index inner servo min          YELLOW2   id1
#define SERVOMAX_index_B  503 // index inner servo max          YELLOW2   id1
```

Figure 5. *Example of obtained values after calibrating both servos actuating the index finger.*

One critical task is checking for any problems that would prevent the hand's movement. This kind of issue may occur because of friction between plastic pieces, which leads to impaired movement of the affected finger. To solve this, the involved pieces are sanded down (Figure 6) in order to remove these defects caused by the limitations of the 3d printer used.

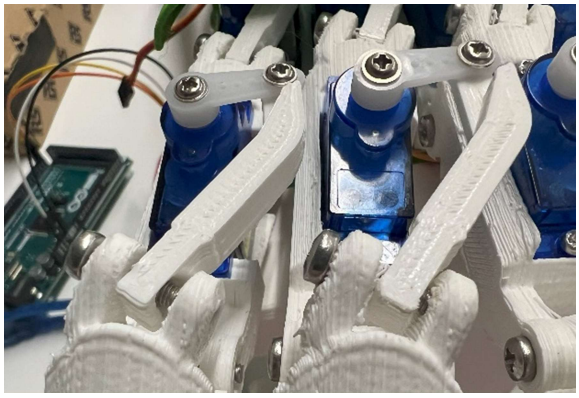

Figure 6. *The edges of the plastic piece connected to the servo have been sanded down to facilitate finger movement.*

Once this process was completed, all the servos were recalibrated to reflect the changes in range gained from fixing the 3D-printed imperfections of the actuator pieces.

After this second calibration, the first iteration of the control code was developed. Which, while functional, had a severe problem that had to be solved: Memory usage. The control code, together with the problems and solutions, will be explained in further detail in the following subsection.

3.2. Control Software

As stated previously, all servos are managed using an Arduino and adafruit board. So as to use the provided adafruit libraries to control the servos, their maximum and minimum pulse lengths are required (as these are needed to move the servo into a certain position using the setPMW [4] function). These values should be something like the ones previously shown in Figure 5.

With this pulse length values ready, these values had to be mapped to a 180-degree range. For this, the Arduino function map() [5] was used, storing the resulting values on an array, as shown in Figure 7.

```
for (i = 0; i < 181; i++){      You, last week • setup …
    for(servoid=0; servoid<10; servoid++){
    pulselen_from_degrees[servoid][i] = map(i,0,180,SERVOMINS_Fingers[servoid],SERVOMAXS_Fingers[servoid]);
    }
}
```

Figure 7. *Code used to map the minimum and maximum values for each servo to its 180-degree range.*

This array stores the data as follows:

value = pulselen_from_degrees[servoid][degrees]

- servoid - [index] The servo stored data values correspond to.
- degrees - [index] Degree you want the selected servo to move to.
- value - [stored value] The pulse length that the setPMW() function uses for servo movement.

This first iteration of value mapping has been deprecated since, while simple to use, it's very inefficient memory-wise. The cause is that for each servo 180 16-bit values are stored in the array. While this is no issue when dealing with just one servo, in a more complex scenario, it can quickly take up large amounts of memory. In this case, since 15 servos were used, the memory requirement for the mapped values was 5.4 kB, which took a considerable portion of the memory available for the Arduino board (8kB).

The problem was resolved through a straightforward and concise solution: A simple C++ script can be developed to obtain an approximation of the step Arduino map function is adding to the minimum pulse length for each servo. These step values can then be used directly on the code with the intention of calculating the mapped value previously stored on the array.

This change reduces the memory usage from 180 16-bit values (unsigned 16bit int) to 2 16-bit values and a 32-bit value (float), which in turn means a reduction of approximately 5.37kB (97.81%), more than half of the SRAM the Arduino board posseses (8kB).

```
void openCloseHand(){
    uint8_t i=0;
    while (i<180){
        pwm.setPWM(0, 0, i*STEP_SERVOID_0 + SERVOMIN_thumb_B);
        pwm.setPWM(1, 0, i*STEP_SERVOID_1 + SERVOMIN_index_B);
        pwm.setPWM(2, 0, i*STEP_SERVOID_2 + SERVOMIN_middle_B);
        pwm.setPWM(3, 0, i*STEP_SERVOID_3 + SERVOMIN_ring_B);
        pwm.setPWM(4, 0, i*STEP_SERVOID_4 + SERVOMIN_pinky_B);
        pwm.setPWM(5, 0, i*STEP_SERVOID_5 + SERVOMIN_thumb_A);
        pwm.setPWM(6, 0, i*STEP_SERVOID_6 + SERVOMIN_index_A);
        pwm.setPWM(7, 0, i*STEP_SERVOID_7 + SERVOMIN_middle_A);
        pwm.setPWM(8, 0, i*STEP_SERVOID_8 + SERVOMIN_ring_A);
        pwm.setPWM(9, 0, i*STEP_SERVOID_9 + SERVOMIN_pinky_A);
        pwm.setPWM(11, 0, -i*STEP_SERVOID_11+SERVOMAX_palm);
        i++;
    }
    i=0;
    while (i<180){
        pwm.setPWM(0, 0, -i*STEP_SERVOID_0 + SERVOMAX_thumb_B);
        pwm.setPWM(1, 0, -i*STEP_SERVOID_1 + SERVOMAX_index_B);
        pwm.setPWM(2, 0, -i*STEP_SERVOID_2 + SERVOMAX_middle_B);
        pwm.setPWM(3, 0, -i*STEP_SERVOID_3 + SERVOMAX_ring_B);
        pwm.setPWM(4, 0, -i*STEP_SERVOID_4 + SERVOMAX_pinky_B);
        pwm.setPWM(5, 0, -i*STEP_SERVOID_5 + SERVOMAX_thumb_A);
        pwm.setPWM(6, 0, -i*STEP_SERVOID_6 + SERVOMAX_index_A);
        pwm.setPWM(7, 0, -i*STEP_SERVOID_7 + SERVOMAX_middle_A);
        pwm.setPWM(8, 0, -i*STEP_SERVOID_8 + SERVOMAX_ring_A);
        pwm.setPWM(9, 0, -i*STEP_SERVOID_9 + SERVOMAX_pinky_A);
        pwm.setPWM(11, 0, i*STEP_SERVOID_11+ SERVOMIN_palm);
        i++;
    }
}
```

Figure 8. *New code used to make the hand open and close. Note the difference with the original implementation shown in Fig.9*

With the previous work done, the setPMW function from the adafruit library can be used to control what position the servo should move to. In order to move the servo to a desired angle, the code will need to iterate through all the degrees between the servo current position and the target position, as can be seen in Figure 9.

```
while (i<180){      You, last week • setup
    delay(3);
    pwm.setPWM(11, 0, -i*STEP_SERVOID_11+SERVOMAX_palm);
    i++;
}
```

Figure 9. *Code needed for the purpose of making a finger perform a controlled sweep from 0º to 180º.*

This is because, in each iteration, the servo is moving into the next angle until the target angle is reached (i.e.: to reach 89º, if it was at 87º it would need to move into 88º and then tell move again into the 89º position)

With the aim of modulating the speed at which the servos actuate, the step used for the iteration needs to be adjusted: Since the setPMW function will move the servo into the desired position, the bigger the steps the faster the servo will rotate. In the cases shown on Figure 8 and 10, the servo will perform 180 steps, moving from 0º to 180º one step at a time. This speed could be further slowed down by adding a delay after each call to the setPWM() function.

```
void openCloseHand_array(){      You, last week • setup
  for (i = 0; i < 181; i++){
    for(servoid=0; servoid<11; servoid++){
      pwm.setPWM(servoid, 0, pulselen_from_degrees[servoid][i]);
    }
  }

  delay(250);

  for (i = 180; i < 181; i--){
    for(servoid=0; servoid<11; servoid++){
      pwm.setPWM(servoid, 0, pulselen_from_degrees[servoid][i]);

    }
  }
}
```

Figure 10. *Original implementation of code used to make the hand open and close.*

Figure 8 and Figure 10 shows two iterations of the code used to perform a simple gesture: opening and closing the arm's hand (The only minor difference between both codes is that the new implementation, Figure 8, also includes the code required to actuate the palm servo, which is not included in Figure 10). In both cases, the step is a single degree, making the hand open and close as slowly as it is physically capable of. This lower speed limit is forced by the mechanical nature of how the fingers are pushed and pulled by the servos, since if the speed was lowered, as previously mentioned, some fingers might get stuck.

4. Results

To test the hand capabilities, new code was developed to perform a compound gesture (Fig. 11): Closing the hand and pushing a button with the index finger. (All code used during this project can be found at GitHub [6]).

```
void loop() {

  pwm.setPWM(15, 0, 30*STEP_SERVOID_15+SERVOMIN_wrist);
  delay(2000);

  pointIndex();
  delay(5000);


  lowerHand();
  delay(500);


  raiseHand();
  delay(500);
}
```

Figure 11. *Implementation of the three previously defined methods.*

The desired gesture is divided into four smaller gestures: The hand is first positioned by tilting the wrist to the 30º position, then the hand will close, leaving the index finger extended. Once this is done the hand will tilt by the wrist servo all the way down, and after half a second it will return to the original tilt of 30º.

As previously shown, the main body of the Arduino program (Fig. 11) combines a single use of the setPWM function and three methods; this is because the desired gesture is divided into four smaller gestures (Fig.12).

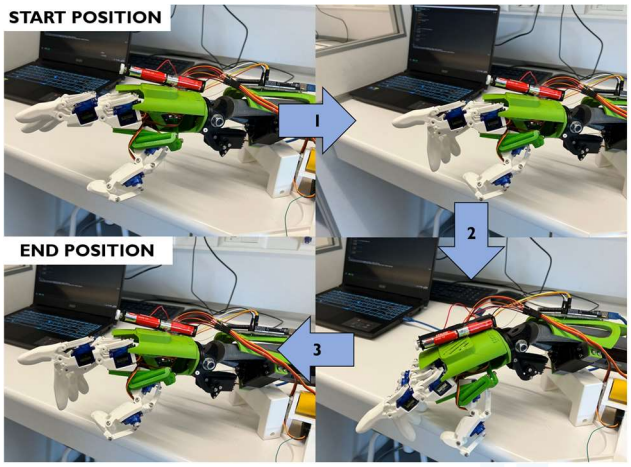

Figure 12. *Depiction of the three movements (1→2 →3) that add up to form the desired gesture.*

The hand is first positioned by tilting the wrist to the 30º position, then the hand will close, leaving the index finger extended. Once this is done the hand will tilt by the wrist servo all the way down, and after half a second it will return to the original tilt of 30º.

5. Conclusions

As stated previously, Youbionic arm can mimic human-like motion, but it has some limitations: a narrow range of movement on the forearm and wrist servos, combined with a rather limited grip capability that hinders its capacity to hold smaller-sized objects, such as a pen.

This limited grip capability is caused because of the design of the actuation mechanism of the finger joints. Finger movement is provided via single servos pushing and pulling a plastic piece that connects the servo with the finger. This issue happens because friction from both plastic pieces hinders the effective range of motion and sensibility that the fingers ultimately have. To mitigate this problem as much as possible, plastic pieces were sanded down, but the issue still prevails. This issue means that the fingers must be moved at a set speed as if moved slower, they might get stuck at a non-desired angle and can cause permanent damage on the servo responsible for actuating said finger (as it will still try to push or pull without taking into consideration the fact that the finger is stuck).

An area that still requires further attention and effort is adding some kind of memory for the servos, as right now, once the script ends, the servo position is not being stored in memory. This has a big impact on how the arm behaves, as when the script starts running, some servos might make a quick 180-degree sweep that could cause damage to the components of the arm. This should only happen in the extreme cases in which the servo stopped at the 0º position and the script considers the starting position as 180º, making the servo perform a 180-degree sweep in a single step.

Lastly, an additional concern regarding the implementation used to control this bionic arm arises from the pulse length sensitivity of the servos, which in combination with the choices taken to control the servos can result in a loss of accuracy in servo movement in certain scenarios. As previously stated, the servo is controlled by a pulse signal of a given duration. If the servo is to be moved a certain number of degrees, for each degree, the duration of the pulse signal sent to the servo is increased (or decreased) by a step calculated from the minimum and maximum pulse lengths for the servo. If this step happens to be smaller than the sensitivity of the actuated servo, it could be ignored by the servo meaning that the servo will stay at the same position. This leads to a wide range of servo accuracies, ranging from 0.5º to 1.5º, depending on the servo specifications and the maximum and minimum values obtained during the calibration process.

In addition, there is another minor aspect that requires attention: development and implementation of a GUI. This should allow a more intuitive and user-friendly interface to interact and control the movement of the arm servos, making it easier to perform tests on the arm.

References

[1] Liarokapis, M., Zisimatos, A.G., Mavrogiannis, C.I. and Kyriakopoulos, K.J., 2014. Openbionics: An open-source initiative for the creation of affordable, modular, light-weight, underactuated robot hands and prosthetic devices.

[2] Unlimited Tomorrow TrueLimb: https://www.unlimitedtomorrow.com/truelimb/ (Consulted: May 2023)

[3] Atom Limbs Atom Touch bionic arm: https://www.atomlimbs.com/touch (Consulted: May 2023)

[4] Adafruit PMW Servo Driver Library: https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library (Consulted: May 2023)

[5] Hughes, J.M., 2016. Arduino: a technical reference: a handbook for technicians, engineers, and makers. " O'Reilly Media, Inc." (ISBN: 1491934506).

[6] Code used for all the project: https://github.com/hugoperezgi/proyectos2_HumanoidArm (Consulted: June 2023)