Final Project Report

Joe Pollock
A02272750
Wonyoung Chang
A02300761
ECE3710
Fall 2021

1. Introduction

The device designed in this project utilizes the forearm and hand of the "open-source 3D printed life-size robot," known as InMoov, created by Gaël Langevin, a French sculptor and designer. The hardware is programmed via C++ to move the hand by pressing one of three different buttons.

2. Scope

This document contains the basic instructions and underlying process behind creating this design. The requirements to produce the design, the dependencies of the design for operation, theory of operation to provide an overview of the project, and design alternatives to discuss other approaches to the design that were not taken are discussed in the Design Overview. In the Design Details section, the document addresses the design in detail to promote the reproduction of the design. During the creation of the project, many tests were administered to ensure the project met all requirements. These tests and their conclusions can be found in the Testing section.

This document does not go in-depth on how to produce this design from scratch, such as the creation and printing process of the physical forearm and hand. In addition, this document does not explain the source code, but the source code can be found in the Appendices section.

3. Design Overview

3.1. Requirements

The following requirements for the project were presented by Professor Jonathan Phillips:

- An input, such as the buttons.
- An observable output, such as the movement of the hand.

3.2. Dependencies

The following dependencies are required for the functionality of the project:

- An accurate and precise 3D printed forearm and hand.
- Six servo motors.

- A total of 17 volts.
- A microcontroller not produced by Arduino or Raspberry Pi.
- Source code

3.3. Theory of Operation

The main objective of this project aims to utilize three buttons to move a 3D printed forearm and hand. Five of the servo motors moved one finger each. The sixth servo motor rotated the wrist. The servo motors contained pulleys on the rotor to move the fingers and wrist as they rotated between 0 degrees and 180 degrees. The pulleys contained nylon string, which extended into the fingers, to open and close the hand.

All of the servo motors were connected to a microcontroller and a power source. The microcontroller produced Pulse Width Modulation (PWM) frequencies via C++ code to communicate the position each servo should be in depending on which button had been pressed. The power source, from the microcontroller connected to a PC and the external battery packs, provided the voltage necessary to operate the servo motors.

3.4. Design Alternatives

There were many different paths possible when producing this project. One of those design alternatives included the choice in power supply. This project utilized the voltage from the microcontroller and voltage from two battery packs. Each battery pack contained four AA batteries. Alternatively, the project could receive the necessary voltage from a single power source, such as a large battery. This path was not taken due to the cost-benefit behind buying two battery packs and a pack of AA batteries.

Another design alternative included the 3D printed forearm and hand. Multiple different alternatives were available, but the chosen 3D printed forearm and hand allowed the developers connections to a well-established community if problems arose during the construction of the forearm and hand. In addition, the InMoov files are open-source material under the Creative Commons License, which prevents this development from infringing on any local, state, federal, or international laws.

The microcontroller utilized for the project was not required. The InMoov hand and forearm would work for any microcontroller capable of producing PWM frequencies. An alternative microcontroller was not chosen for two distinct reasons: (1) the project requirements prevented the use of

Arduino, and (2) the utilized microcontroller was readily available to the team.

Another design alternative was the input of the design to make the hand movements and change gestures. Initially, the goal of the project included using an electromyography (EMG) sensor attached to the user's arm to move the hand as the user moves their hand. The developers pursued the buttons as an alternative due to budgetary constraints.

4. Design Details

This section addresses the design in detail, both what it is and why. Enough information should be given so that someone with an engineering background could
implement the design. For example, timing analysis, schematics, and code snippets
are an appropriate level of detail. Datasheets or software listings are not. That would
be too much detail. Still, expect over half of your document (not counting the appendices) to be design details, so use subsections for clarity.

4.1. 3D Printing

The 3D printable hand and forearm produced by Gaël Langevin, under the name InMoov, consisted of 65 different pieces, excluding any hardware, wires, string, and other external equipment. The multitude of pieces allows the hand and forearm to have 3 Degrees of Freedom. In addition, the multitude of pieces allows the hand and forearm to move more life-like than alternative robotic hands, which may not have as many joints as the InMoov hand and forearm. The tools required to put the hand and forearm together included pliers, glue, clamps, drill bits of various sizes, a screwdriver, a knife, and tweezers. The InMoov website (https://inmoov.fr/) instructed the developers on the construction of the forearm and hand. Figure 1 shows the complete construction of the InMoov forearm and hand with the servo motors installed and the string routed along the fingers.

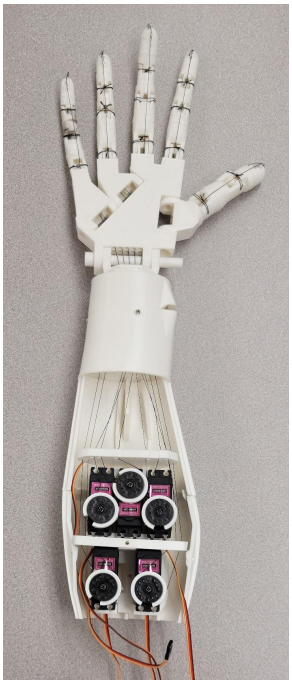

Figure 1: The InMoov Forearm & Hand

4.2. Servo Motors

One portion of hardware utilized for this project included six MG996R servo motors. The servo motor must be able to move from 0 degrees to 90 degrees minimum for the fingers, but a servo motor moving up to 180 degrees will allow the hand to close much further. PWM frequencies were sent to the MG996R each time a button was pressed to alter the position of the hand. The full PWM period completed was 50 Hz. An open hand meant all the servos were operating at 0 degrees. A 0 degree servo motor resulted by sending a 1 (or high frequency) through the PWM connection for 0.6 ms. An open hand can be achieved using the *paper()* function from the ServoMotor.c code in the Appendices.

4.3. Materials

The type of string utilized in this project was crucial to how well the project operated. We used black nylon string for the project, which slipped more than we preferred, so we glued the string to the servo motor wheels. Frequently, fishing string was recommended during our research for the project, but this type of wire stretches out over time, which prevents the servo motors from closing the hands. The string utilized must be strong enough to hold anything placed inside the hand. Our research stated that the minimum weight the string must endure is around 15 kg.

The three buttons operated for the cycling between hand gestures utilized the SysTick Timer and interrupts to detect when each button was pressed. When a specific button was pressed, the hardware would call the interrupt connected to the GPIO pin. Inside the handler function, the code calls another function, which alters the position of the servo motors via Timer 2 in the microcontroller. This function alters the length of time a high wave is sent to the servo motors, which determines the degree output. All three buttons utilized a single resistor each with a minimum of 2 kilohms. The pull-up resistors provide the definite position necessary for the system.

4.4. C++ Code

Our code, located in the Appendix, differentiates between the initialization of the PWM period and the initialization of the servo motors. The *PWM_Init()* function sets the PWM period. The *Servo_Init()* function initializes the Timer 2 channels. Our code utilizes all four channels of Timer 2. Four of the fingers are tied to an individual channel to promote independent movement. The pinky is tied to channel 1 with the thumb. This optimized the hardware because the alternative would have been to initialize a second timer. The thumb and pinky did not need to operate independently because both fingers would move together regardless of the button pressed.

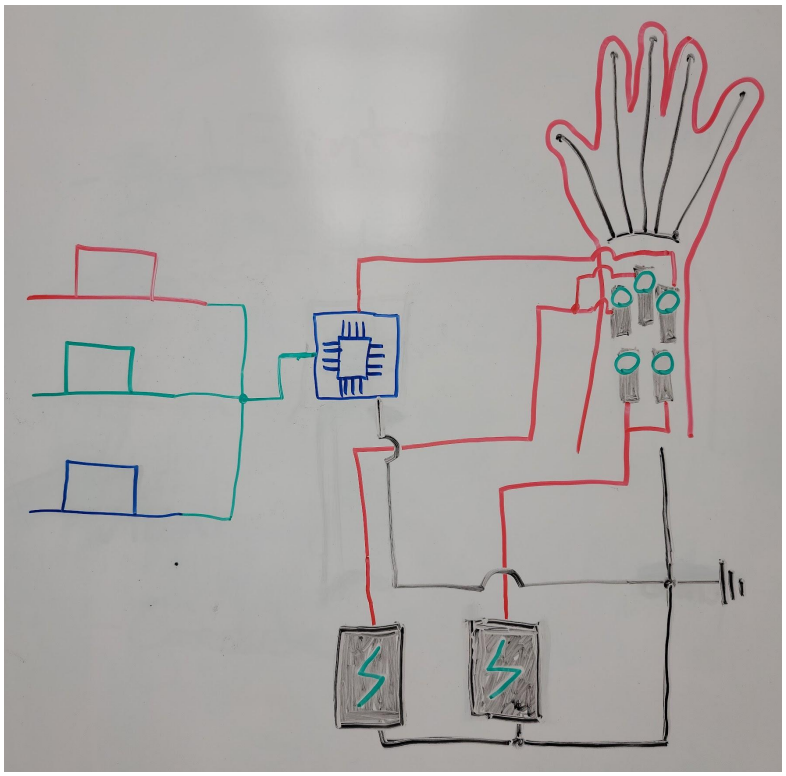

Figure 2: InMoov Hand & Forearm Operations Diagram

4.5. The System

Figure 2 illustrates the system connections. The three buttons can be observed on the left side, which the buttons are connected to the microcontroller in the middle of the diagram. The bottom of the diagram presents the two separate power banks that provided voltage of around 6 V to two servo motors each. The microcontroller powered the fifth servo motor and the buttons. All hardware is connected to the breadboard with a common ground. The black lines in the hand represent the string pulled by the servo motors to mobilize each finger. Figure 3 depicts the live model of the final product.

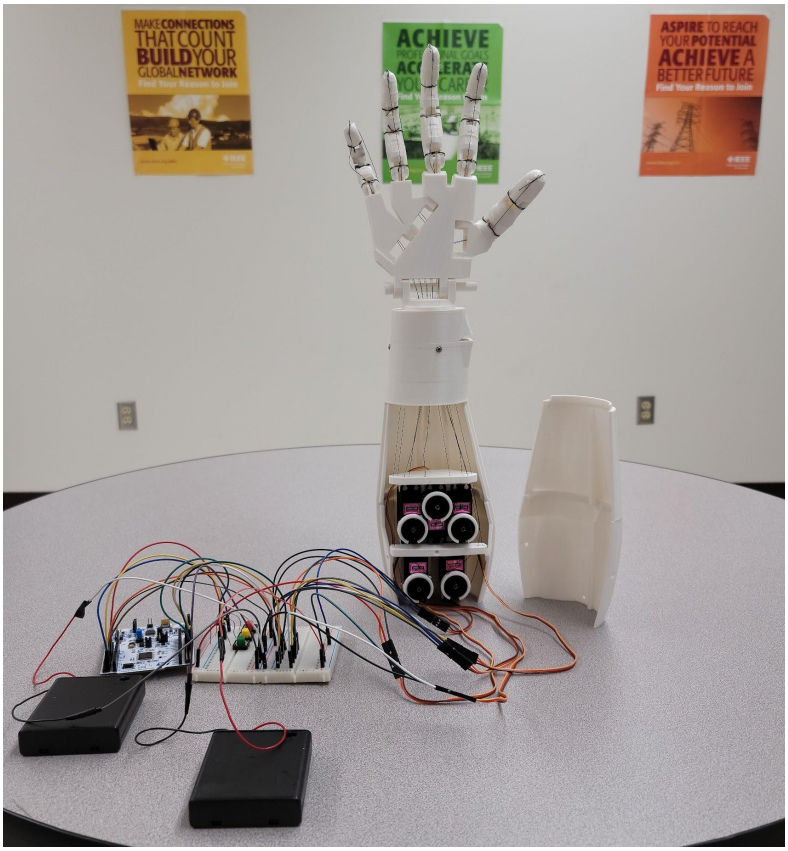

Figure 3: Final Product of InMoov Forearm & Hand Project

5. Testing

To ensure our project has accomplished each requirement, we ran multiple tests throughout the development process. Our first test involved ensuring we had output for our system. As intended, we incorporated servo motors to manipulate the hand gestures of the model. Our testing procedure involved connecting a single servo motor to the STM32L476RG microcontroller. One pin was connected to the 5 V output of the microcontroller, the next pin connected to the ground port

of the microcontroller, and the final pin connected to a GPIO pin on the microcontroller. Testing each servo motor ensured none of our servo motors were faulty. The observations necessary to verify were that the servo motors would move from 0 degrees to 90 degrees, then back to 0 degrees. During initial testing, the servo motors did not operate, so nothing happened. Eventually, the servo motor would turn a specific number of degrees, which allowed us to manipulate our equation to either turn further to the right or turn less than the servo motor did during the test. The applicable requirement to this test was that we had a valid output for the system.

Our second test involved testing our input. As initially intended, we instantiated the SysTick Timer to incorporate interrupts to trigger our servo motors when a button is pressed. The test procedure involved connecting a single servo motor to the microcontroller, then connecting all three buttons to the microcontroller. As the program ran, we intended on verifying the servo motor position changed to the desired degree for each button pressed. For example, our closed hand function must move the servo motor to 90 degrees and our open hand function must move the servo motor to 0 degrees. During testing, our observations demonstrated that pressing the red button moved the servo motor to 90 degrees and pressing the yellow button moved the servo motor to 0 degrees. Our intentions behind this test were to ensure our system had input.

6. Conclusion

The testing portion of the project demonstrated the hand moved from an open position to a closed position, then the independent movement of the fingers as the green button demonstrated an erected index and middle finger with the remaining fingers closed. The hand performed its task, but the hand did not perform optimally due to the servo motors recommended by the InMoov designer. The servo motors used were unable to perform a full 180 degrees, which would allow the InMoov hand to further close.

An additional decrease in optimality occurred due to the lack of precision used with the 3D printer. The fingers were intended to contain holes on the inside of each segment, but the holes were closed too tight to line string through them as instructed on the InMoov website. As a result, we lined string along the outer edges and inside edges of the fingers, which allowed the fingers to open and close as necessary. As the outer edge string pulls tighter, the inside edge string loosens to create an erect finger. As the outer edge string loosens, the inside edge string tightens to create a closed finger.

Appendices

Main.c:

```c
#include "interrupts.h"
#include "ServoMotor.h"

int main()
{
    Interrupt_Init();    //initialize button interrupts
    systick_init(4000);    //Initialize SysTick Timer
    PWM_Init();    //Initialize PWM frequency
    Servo_Init();    //Initialize Servo Motors

    while(1);
}
```

Interrupts.h:

```c
#ifndef __STM32L476R_NUCLEO_INTERRUPTS_H
#define __STM32L476R_NUCLEO_INTERRUPTS_H

#include "stm32l476xx.h"
#include "ServoMotor.h"

//Port: GPIOC
//Buttons:
//    - Red (Rock): PC0
//    - Yellow (Paper): PC1
//    - Green (Scissors): PC4

void Interrupt_Init(void);
void systick_init(unsigned int ticks);
void SysTick_Handler(void);
void EXTI0_IRQHandler(void);
void EXTI1_IRQHandler(void);
void EXTI4_IRQHandler(void);

#endif
```

ServoMotor.h

```c
#ifndef __STM32L476R_NUCLEO_SERVOMOTOR_H
#define __STM32L476R_NUCLEO_SERVOMOTOR_H

#include "stm32l476xx.h"
#include "interrupts.h"

//Port: GPIOA & GPIOB
//Timer: TIM2
//Input_channel for:
//    - Thumb(PA0): Channel 1
//    - Index Finger(PA1): Channel 2
//    - Middle Finger(PB10): Channel 3
//    - Ring Finger(PB11): Channel 4
//    - Pinky(PA5): Channel 1

extern int PRESCALER;
extern int AUTO_RELOAD_VALUE;

extern double DUTY_CYCLE_0_DEG;
extern double DUTY_CYCLE_90_DEG;
extern double DUTY_CYCLE_180_DEG;

void PWM_Init(void);
void Servo_Init(void);

void rock(void);
void paper(void);
void scissors(void);
void delay_ms(unsigned int ms);

#endif
```

Interrupts.c:

```c
#include "interrupts.h"

void Interrupt_Init()
{
    /*Enable GPIO Clock*/
```

```
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;

    /*Set moder to input for PC0, PC1, PC4*/
    GPIOC->MODER &= 0xFFFFFCF0;

    /*Set to Pull down for PC0, PC1, PC4*/
    GPIOC->PUPDR &= 0xFFFFFCF0;
    GPIOC->PUPDR |= 0x0000020A;

    /*Enable Interrupts*/
    NVIC_EnableIRQ(EXTI0_IRQn);
    NVIC_EnableIRQ(EXTI1_IRQn);
    NVIC_EnableIRQ(EXTI4_IRQn);

    /*Red Button*/
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    SYSCFG->EXTICR[0] &= ~(SYSCFG_EXTICR1_EXTI0);
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI0_PC;
    EXTI->IMR1 |= EXTI_IMR1_IM0;    //Interrupt Mask Register
    EXTI->RTSR1 |= EXTI_RTSR1_RT0;    //Rising trigger selection

    /*Yellow Button*/
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    SYSCFG->EXTICR[0] &= ~(SYSCFG_EXTICR1_EXTI1);
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI1_PC;
    EXTI->IMR1 |= EXTI_IMR1_IM1;    //Interrupt Mask Register
    EXTI->RTSR1 |= EXTI_RTSR1_RT1;    //Rising trigger selection

    /*Green Button*/
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    SYSCFG->EXTICR[1] &= ~(SYSCFG_EXTICR2_EXTI4);
    SYSCFG->EXTICR[1] |= SYSCFG_EXTICR2_EXTI4_PC;
    EXTI->IMR1 |= EXTI_IMR1_IM4;    //Interrupt Mask Register
    EXTI->RTSR1 |= EXTI_RTSR1_RT4;    //Rising trigger selection
}

/*Initialize SysTick Timer*/
void systick_init(unsigned int ticks)
{
```

```
    SysTick->CTRL = 0;    //Disable SysTick
    SysTick->LOAD = ticks - 1;    //Set reload register
    SysTick->VAL = 0;    //Reset SysTick counter value
    SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk;    //Select
processor clock
    SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;    //Enables SysTick
interrupt
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;    //Enable SysTick
}

/*SysTick interrupt service routine*/
void SysTick_Handler()
{

}

/*Red Button ISR*/
void EXTI0_IRQHandler(void)
{
    rock();
    delay_ms(200);    //Delay
    EXTI->PR1 |= EXTI_PR1_PIF0;
}

/*Yellow Button ISR*/
void EXTI1_IRQHandler(void)
{
    paper();
    delay_ms(200);    //Delay
    EXTI->PR1 |= EXTI_PR1_PIF1;
}

/*Green Button ISR*/
void EXTI4_IRQHandler(void)
{
    scissors();
    delay_ms(200);    //Delay
    EXTI->PR1 |= EXTI_PR1_PIF4;
}
```

ServoMotor.c:

```
#include "ServoMotor.h"

////////////////////////////////////////////////////////////////////
//////
//Use MSI
//PWM Mode 1
//ARR = period
//System clock = 4MHz
//Prescaler = 64 -> 4MHz / 64 = 62,500 = 62.5 kHz
//ARR = (62.5 kHz / 50 Hz) - 1 = 1250 - 1 = 1,249
//Compare register for compare output:
//CCR = (ARR + 1) * 0.5 -> 0.5 represents a 50% duty cycle
//CCR = (ARR + 1) * Duty Cycle
//Calculate the Duty Cycle to calculate the CCR
//Set the CCR, the computer will use that to set the PWM high when
desired
//DUTY_CYCLE_MIN = 0.03 -> 3% of 20 ms is when servo is at -90
degrees
//DUTY_CYCLE_MID = 0.078 -> 7.8% of 20 ms is when servo is at 0
degrees
//DUTY_CYCLE_MAX = 0.125    -> 12.5% of 20 ms is when servo is at +90
degrees
////////////////////////////////////////////////////////////////////
///////

//Goal is to reach an output of 50 Hz (20 ms) for the total PWM
Period
int PRESCALER = 64;    // Set PSC -> PSC = System_Clock_Freq / __ Hz
-> The __ Hz can be randomly chosen to get closer to the desired ARR
//ARR = __ Hz / 50 Hz -> This is the same __ Hz as in the line above
and 50 Hz is the desired PWM period
int AUTO_RELOAD_VALUE = 1250 - 1;    // PWM period = (ARR + 1) *
1/(System_Clock_Freq / PSC) = __ sec = __ Hz

/*Set Duty Cycle -> CCR = (ARR + 1) * DUTY_CYCLE*/
double DUTY_CYCLE_0_DEG = ((100 * 0.0006) / 0.02) / 100;    //To go
right -> Use 0.6 ms -> Duty Cycle = (100% * 0.6 ms) / 20 m
```

```c
double DUTY_CYCLE_90_DEG = ((100 * 0.00156) / 0.02) / 100;    //To point upward -> Use 1.56 ms -> Duty Cycle = (100% * 1.56 ms) / 20 ms
double DUTY_CYCLE_180_DEG = ((100 * 0.0025) / 0.02) / 100;    //To go right -> Use 2.5 ms -> Duty Cycle = (100% * 2.5 ms) / 20 ms

void PWM_Init(void)
{
    /*Enable MSI as System Clock*/
    RCC->CR |= RCC_CR_MSION;    //Clock init RCC_CR using MSI
    while (!(RCC->CR&RCC_CR_MSIRDY));    //Wait until MSI is ready

    /*Enable Peripheral Clocks via RCC Registers*/
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;    // Enable GPIOA RCC
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;    // Enable GPIOB RCC
    RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;    // Enable TIM2 RCC

    /*Configure GPIO (Pins PA0, PA1, PA5, PB10, PB11)*/
    GPIOA->MODER &= 0xFFFFF3F0;    //Clear bits in PA0, PA1, and PA5
    GPIOA->MODER |= 0x0000080A;    //Set PA0, PA1, and PA5 set to Alternate Function (10)

    GPIOB->MODER &= 0xFF0FFFFF;    //Clear bits in PB10 and PB11
    GPIOB->MODER |= 0x00A00000;    //Set PB10 and PB11 to Alternate Function (10)

    GPIOA->AFR[0] &= 0xFF0FFF00;    //Clear bits in PA0, PA1, and PA5
    GPIOA->AFR[0] |= 0x00101111;    //Set alternate function pins (PA0, PA1, PA5) as TIM2_CHx (AF1)

    GPIOB->AFR[1] &= 0xFFFF00FF;    //Clear bits in PB10 and PB11
    GPIOB->AFR[1] |= 0x00001100;    //Set alternate function pins (PB10 and PB11) as TIM2_CHx (AF1)

    /*Set pins to push-pull with no pull-up and no pull-down*/
    //The reset-state of the microcontroller is push-pull, so no need to invoke OTYPER
    GPIOA->PUPDR &= 0xFFFFF3F0;    //Clear bits
    GPIOA->PUPDR |= 0x00000000;    //Set to no pull-up or pull-down
    GPIOB->PUPDR &= 0xFF0FFFFF;    //Clear bits PB10 and PB11
```

```
    GPIOB->PUPDR |= 0x00000000;    //Set to no pull-up or pull-down
}

/*Configure Timers(TIM2)*/
void Servo_Init(void)
{
    /*Select the counting direction: 0 = up-counting, 1 =
down-counting*/
    TIM2->CR1 &= ~TIM_CR1_DIR;    //Set to up-counting

    TIM2->PSC = (uint32_t)PRESCALER;    //Prescaler, slow down the
input clock by a factor of (1 + prescaler)
    TIM2->ARR = (uint32_t)AUTO_RELOAD_VALUE;    //Auto-reload

    /*Channel 1*/
    TIM2->CCMR1 &= ~TIM_CCMR1_OC1M;    //Clear output compare bits
    TIM2->CCMR1 |= TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2;    //Select
PWM Mode 1 output (OC1M = 110)

    /*Channel 2*/
    TIM2->CCMR1 &= ~TIM_CCMR1_OC2M;    //Clear output compare bits
    TIM2->CCMR1 |= TIM_CCMR1_OC2M_1 | TIM_CCMR1_OC2M_2;    //Select
PWM Mode 1 output (OC2M = 110)

    /*Channel 3*/
    TIM2->CCMR2 &= ~TIM_CCMR2_OC3M;    //Clear output compare bits
    TIM2->CCMR2 |= TIM_CCMR2_OC3M_1 | TIM_CCMR2_OC3M_2;    //Select
PWM Mode 1 output (OC3M = 110)

    /*Channel 4*/
    TIM2->CCMR2 &= ~TIM_CCMR2_OC4M;    //Clear output compare bits
    TIM2->CCMR2 |= TIM_CCMR2_OC4M_1 | TIM_CCMR2_OC4M_2;    //Select
PWM Mode 1 output (OC4M = 110)

    TIM2->CR1 |= TIM_CR1_ARPE;    //Enable TIM2_ARR register preload

    /*Enable Output*/
    TIM2->CCER |= TIM_CCER_CC1E;    //Channel 1
    TIM2->CCER |= TIM_CCER_CC2E;    //Channel 2
```

```
    TIM2->CCER |= TIM_CCER_CC3E;    //Channel 3
    TIM2->CCER |= TIM_CCER_CC4E;    //Channel 4

    /*Select output polarity: 0 = Active high, 1 = Active Low*/
    TIM2->CCER &= ~TIM_CCER_CC1P;    //Channel 1
    TIM2->CCER &= ~TIM_CCER_CC2P;    //Channel 2
    TIM2->CCER &= ~TIM_CCER_CC3P;    //Channel 3
    TIM2->CCER &= ~TIM_CCER_CC4P;    //Channel 4

    TIM2->CR1 |= TIM_CR1_CEN;    //Enable counter

}

/*--Hand Orientation--*/
/*Rock (Hand closed in Fist)*/
//    - Thumb(PA0->CH1): Closed (180 degrees)
//    - Index Finger(PA1->CH2): Closed (180 degrees)
//    - Middle Finger(PB10->CH3): Closed (180 degrees)
//    - Ring Finger(PB11->CH4): Closed (180 degrees)
//    - Pinky(PA5->CH1): Closed (180 degrees)

/*Paper (Hand open flat)*/
//    - Thumb(PA0->CH1): Open (0 degrees)
//    - Index Finger(PA1->CH2): Open (0 degrees)
//    - Middle Finger(PB10->CH3): Open (0 degrees)
//    - Ring Finger(PB11->CH4): Open (0 degrees)
//    - Pinky(PA5->CH1): Open (0 degrees)

/*Scissors (Index and Middle Finger straight; remainder closed)*/
//    - Thumb(PA0->CH1): Closed (180 degrees)
//    - Index Finger(PA1->CH2): Open (0 degrees)
//    - Middle Finger(PB10->CH3): Open (0 degrees)
//    - Ring Finger(PB11->CH4): Closed (180 degrees)
//    - Pinky(PA5->CH1): Closed (180 degrees)

/*Rock (Hand closed in Fist)*/
void rock(void)
{
    /*Channel 1    (Thumb, Pinky)*/
```

```
    TIM2->CCR1 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_180_DEG);    //Move Servo to +90 degrees

    /*Channel 2 (Index Finger)*/
    TIM2->CCR2 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_180_DEG);    //Move Servo to +90 degrees

    /*Channel 3 (Middle Finger)*/
    TIM2->CCR3 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_180_DEG);    //Move Servo to +90 degrees

    /*Channel 4 (Ring Finger)*/
    TIM2->CCR4 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_180_DEG);    //Move Servo to +90 degrees
}

/*Paper (Hand open flat)*/
void paper(void)
{
    /*Channel 1 (Thumb, Pinky)*/
    TIM2->CCR1 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_0_DEG);    //Move Servo to 0 degrees

    /*Channel 2 (Index Finger)*/
    TIM2->CCR2 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_0_DEG);    //Move Servo to 0 degrees

    /*Channel 3 (Middle Finger)*/
    TIM2->CCR3 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_0_DEG);    //Move Servo to 0 degrees

    /*Channel 4 (Ring Finger)*/
    TIM2->CCR4 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_0_DEG);    //Move Servo to 0 degrees
}

//Scissors (Index and Middle Finger straight; remainder closed)
void scissors(void)
{
```

```
    /*Channel 1    (Thumb, Pinky)*/
    TIM2->CCR1 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_180_DEG);    //Move Servo to +90 degrees

    /*Channel 2 (Index Finger)*/
    TIM2->CCR2 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_0_DEG);    //Move Servo to 0 degrees

    /*Channel (Middle Finger)*/
    TIM2->CCR3 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_0_DEG);    //Move Servo to 0 degrees

    /*Channel 4 (Ring Finger)*/
    TIM2->CCR4 = (uint32_t)((AUTO_RELOAD_VALUE + 1) *
DUTY_CYCLE_180_DEG);    //Move Servo to +90 degrees
}

/*Delay function*/
void delay_ms(unsigned int ms)
{
    unsigned int i, j;
    for(i=0; i < ms; i++)
    {
       volatile int temp = 0;
       for(j=0;j<800;j++)
       {
            temp++;
       }
    }
}
```