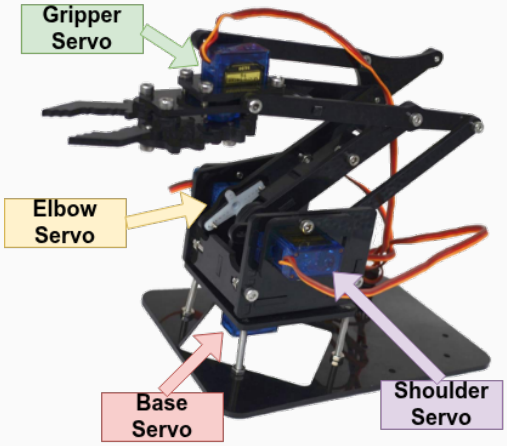
CG2111A Engineering Principles and Practices II

Studio 7: Mini-Project 1: Robot Arm

Introduction

Welcome to your first **mini-project**! In this session, you will assemble the *robot arm* that will be used to manipulate objects in your project, and test that it powers on correctly. Then, you will use a simple command interface using the Arduino Serial Monitor to control the arm. Finally, you will convert the provided code into bare-metal. This will prepare you for the **graded *Robot Arm Checkpoint* (5%)** in the second session of the week.



Learning Objectives

1. Assemble the robot arm with servos for your project
2. Use the Arduino `Serial` library to read user commands to control your servos
3. Use a simple control scheme to move the servos based on user commands
4. Write a bare-metal servo controller for multiple servos using `Timer` interrupts

Equipment Needed

Item	Quantity
Robot Arm Kit	×1
Robot Arm Kit Fasteners (M3 Screws and Nuts, various lengths)	×1

Item	Quantity
Arduino Uno	×1
MG90S Servo and Horn	×4
Phillips Head Screwdriver	×1
Benchtop Power Supply	×1
Breadboard, Jumper Wires, Crocodile Clip Wires	

Supporting Documents

- Servo Midpoint Sketch (servo_midpoint.ino)
- Serial Command Template (serial_arm_template.ino)

Activities

Submission

Ungraded Studio

This is an ungraded studio, so you are not required to submit anything to Canvas. However, the learning objectives relate directly to the projects in this course. To make sure you're prepared for the projects, answer the questions you see in your Learning Journal, which is a document you must create (e.g., on Google Docs) and maintain during CG2111A.

Robot Arm Background

Introduction

You will be building a robot arm based on a popular open-source kit called the **MeArm**, specifically the **MeArm v0.4**. The MeArm designs were created by Ben Gray and the many MeArm versions and variants are documented extensively online.

It's very important to follow the correct set of instructions for this build! We will provide you with our own curated set of instructions below. Following the wrong instructions (e.g., for the wrong arm variant) will lead to many tears and wasted time. Ask us how we know...

The arm's popularity comes from its simplicity. It only requires four servos, some M3-sized fasteners, a Phillips-head screwdriver, and a batch of cheap laser-cut parts which can be packed in a few flat boards.

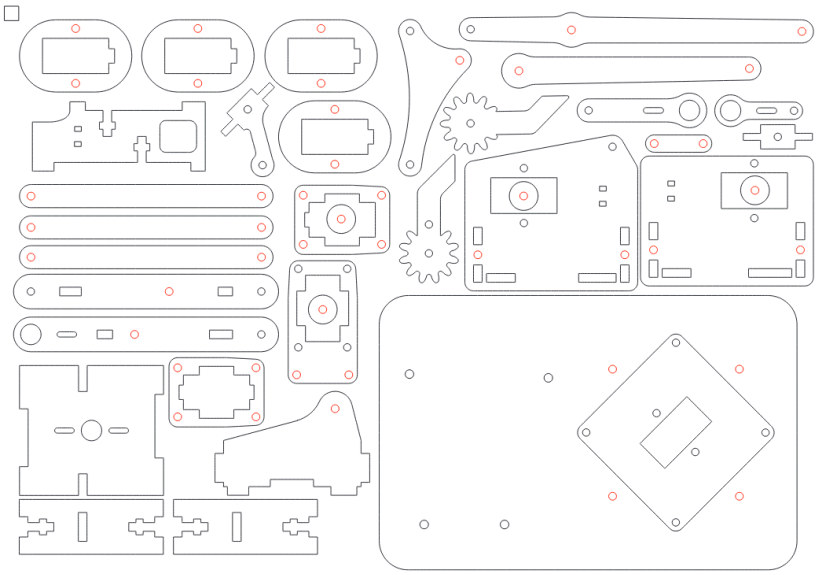


Figure 1: Open-source design of laser-cut acrylic parts included in the MeArm v0.4 kit (source)

How it works

The MeArm has four degrees of freedom (i.e., 4-DoF) driven by one servo for each DoF:

- **BASE**: For *rotation* of the arm's base (yaw)
- **SHOULDER**: Shoulder of the robot, *lifts* the arm up and down
- **ELBOW**: Elbow of the robot, *reaches* forward and backward
- **GRIPPER**: Gripper/claw of the robot, controls *open/close* motion

By manipulating these four servos, you can make the arm move through a wide range of motion. You can also use it to pick up small, light objects using the gripper.

How to approach this studio

Your team will have 4 - 5 members. One option is to split the team into two subgroups so that you can finish the work within the studio session: one subgroup to tackle the arm build first (Activity 1 and 2), and one subgroup to handle the servo control (Activity 3 and 4). During the arm build, not all servos will be used at the start, so Activity 3 and 4 can still be tested to some degree while the build is ongoing. However, this is entirely up to you.

> **! 'No Sponge' Rule**
>
> Please make sure, among yourselves, that all team members contribute to the project.

Activity 1: Building the Arm

> **Build instructions**
>
> Please follow the assembly instructions provided in our Google Slides document:
> CG2111A MeArm v0.4 Assembly Instructions

By the end of this section and by following the instructions, you should have a fully assembled robot arm with the 4 servos installed correctly.

Activity 2: Powering the Arm

Building the Full Circuit

The four servos can draw more current than the Arduino can supply via its digital pins. The Arduino can only supply around 20 mA per pin, and only 100 mA total. However, just *one* servo can draw *700 mA*, so all four could require **2.8 A**. Therefore, you need to use the **benchtop power supply** at your workstation to power the servos.

> **Information**
>
> For your final robot, you will use a different power supply since the benchtop power supply is well... benchtop. This just makes things easier for now.

> ❗ **!! Do not connect anything to the Arduino's 5V pin !!**
>
> Always power the servos from the **benchtop power supply**, *not the Arduino 5V pin.*
>
> If you use the Arduino's 5V pin to power the servos, **you will likely kill the Arduino**.
>
> The power supply's ground and the Arduino's GND pin must be connected together (to form a common ground). The **Arduino's 5V pin should remain unused**. This will ensure that the Arduino will not power the servos.

Procedure:

- Start with no wires connected to the benchtop power supply.
- Turn on and set the benchtop power supply to 5V and current limit to 2.8A.
  - Once you have set this up, **turn off the power supply!**
- Connect all servos' 5V wires to the power supply's positive terminal, and all the servos' GND wires to power supply GND.
- Connect the Arduino GND to the same ground rail as the servos.
- Connect only the servo signal wires to Arduino digital pins.
  - We recommend using the pins A0 - A3 on the Arduino (pins 14 - 17) to save the Timer/PWM pins for other parts of your project.

Follow the circuit diagram exactly as in Figure 2 below. However, you may need jumper wires to extend the servo cables to make the connections shown.

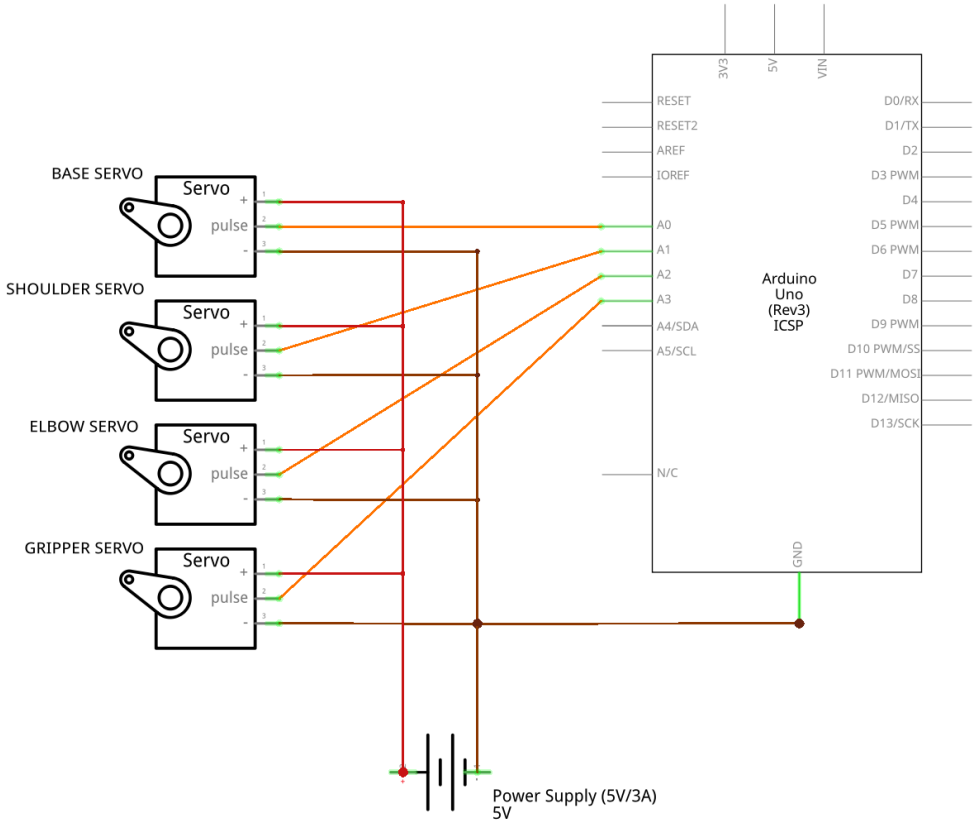


Figure 2: Circuit diagram for powering the four servos from the benchtop power supply (remember: do not connect anything to the Arduino's 5V pin)

Uploading the Midpoint Code

Do not turn on the power supply yet. Upload the provided file servo_midpoint.ino to the Arduino Uno. If you changed the pins used for your servos, change the sketch accordingly.

Keep your Arduino powered on via the USB cable before proceeding to the next step.

First Full Power-On

Only once your circuit is complete, the Arduino code has been uploaded, and the Arduino is on, turn on the power supply.

You should observe that all four axes of your robot arm move to some initial position.

> ⚠ **Warning**
>
> If you see any "magic smoke", or the arm doesn't move, or the servos get hot, **turn off the power supply** and ask for help from your TA and/or instructor.

Testing a Different Arm Position

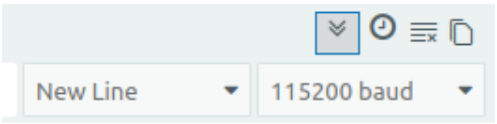
Change the servo values in your servo_midpoint.ino file *slightly* (e.g., change 90 to 95, don't modify it too far). Re-upload the code to the Arduino and verify that all **four** servos move.

> ⚠ **Servo Limits**
>
> The servos will blindly try to move to whatever angle you command. If the values are changed too much, the servos may move beyond their safe range, hit mechanical limits, or cause possible damage to the robot arm. At this point, change them only a bit!
>
> **Warning**: The gripper servo typically only has a 10 degree rotation limit.

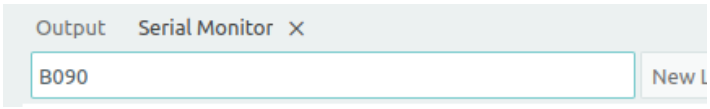
Activity 3: Controlling the Arm with Serial Commands

In this activity, you will use a **Serial command interface** so the arm can be controlled from the Arduino **Serial Monitor**. The goal is to be able to enter a *command* (a sequence of characters) into the Serial Monitor, press ENTER, and something should happen.

> **Serial Settings**
>
> Set the Serial Monitor line ending to **New Line** so that commands are sent as a line, and the baud rate to **115200**, so the commands are processed quickly. The baud rate defines how fast data is sent across the serial link, and both the Arduino and Serial Monitor need to agree on the baud value. We will explore these concepts further in another studio.



Serial Monitor settings



Command (B090) being entered into the input field

Command Format

We will use a small, human-readable command set:

- Commands start with a single letter.
- Commands with numbers must use exactly three digits with no spaces (use leading zeros).
- The `V` command sets the movement speed (delay per degree in milliseconds).
- The `B`/`S`/`E`/`G` commands move the base, shoulder, elbow, and gripper servos respectively.

Command	Format	Action	Example
B	Bddd	Move base servo to target angle (degrees).	B090
S	Sddd	Move shoulder servo to target angle (degrees).	S120
E	Eddd	Move elbow servo to target angle (degrees).	E145
G	Gddd	Move gripper servo to target angle (degrees).	G030
V	Vddd	Set speed as delay per degree (ms).	V010
H	H	Set all servos to a 90 degree position.	H

While most of the commands are self-explanatory, the `V` command needs some explanaton. It sets the **speed** of servo movement. The speed is defined as a delay (in milliseconds) per degree of movement. For example, if the speed is set to `V010` (10 ms/deg), and the servo is commanded to move from `B090` to `B120` (a change of `30 deg`), then the total time taken to move will be `30 deg * 10 ms/deg = 300 ms`. The `H` command returns all servos to the 90 degree ("home") position.

Serial Command and Servo Code

> 📝 **Code Provided**
>
> We have provided most of the code for this part, so that you can focus on the next activity.

Open the file serial_arm_template.ino, and upload it to your Arduino.

Your task: try out the commands that we specified in the table above, and confirm that the robot arm moves as intended. Change the movement speed and test that the functionality works as well. Finally, make sure the H command works to home the arm.

> ⚠️ **Range of movement**
>
> Don't move the joints too far yet. Stick to around 80 to 100 degrees.

Adding Limits (Important!)

Your task: Find the limits of your robot arm's axes by choosing different angle values for each of the servos. Once you find the angle limit for each joint, modify the provided code to **prevent the arm moving past these limits**. This will help prevent damage to your servos even if you enter an incorrect command.

Activity 4: Moving to Bare-Metal Control

> 📝 **Time Management**
>
> This activity will take some time, since you have to convert the existing code to a bare-metal version for the servos, and add new functionality. Please plan appropriately.

Now that you've verified the functionality of the arm and the serial control code, in this activity, you will **stop using the Servo.h** library and switch to bare-metal control for your servos.

Requirements for this activity:

1. Your code should not use any **Servo.h** functions (e.g., `attach`, `write`, etc)
2. You should use a **Timer1 interrupt** (similar to the last question of the Timer studio) to generate pulses that will control the position of multiple servos.
    1. The MG90S servos can take pulses between around 0.5 ms and 2.5 ms to move through its full range. You do not have to keep to a pulse width of 1 - 2 ms if it limits your servo's range.
3. You should be able to **change the movement speed** of the servos with the V command. ~~even *while* the servo is moving~~. You may use the **`delay`** function solely to control the servo rotation speed (but not how you set its position) during this mini-project.
4. You should be able to set the position of all four servos within each 20 millisecond period. However, it's alright if only one servo's position is actively changing at a time (it's OK to service one movement command before accepting the next movement command). ~~move **multiple servos** at the **same time** e.g., if the commands `B120` and `S120` are entered in quick succession (and assuming a slow enough speed that `B120` does not finish immediately), both the base and shoulder servos should be able to move at the same time from a human perspective.~~
    - For avoidance of doubt, this means that you should not move the base servo first to completion, and only then attempt to move the shoulder servo to completion.

> 📝 **Save your code!**
>
> A significant part of the code you write here will be used for your project. Please make sure to save it. Also, to make your life easier during the rest of your project, try to write it in a modular way (with functions, defines, etc).

Preparing for the Checkpoint Demo

The activities above are to prepare you for the checkpoint demo that happens in the second studio of the week. In the checkpoint demo, your TAs and instructor will verify at least the following behaviors (subject to changes):

- ☑ The robot arm is assembled correctly with all servos included.
- ☑ Bare-metal servo control is implemented (no `Servo.h` usage), using timers to generate servo pulses.
- ☑ The following commands function correctly and move the expected joint:
  - ☑ `Bddd` (base/yaw), `Sddd` (shoulder), `Eddd` (elbow), `Gddd` (gripper), `H` (home).
- ☑ The speed command works as intended:
  - ☑ `Vddd` updates the movement speed (delay per degree) and the effect is observable.
- ☑ Your code limits the movement angle of the servos to what is physically achievable.
- ☑ Multiple servos can move concurrently (e.g., issuing `B120` and `S120` in quick succession results in both joints moving without waiting for the other to finish).
- ☑ Servo movement remains smooth and responsive while commands are issued (including speed changes during motion).

Summary

In this studio, you have built and tested a robot arm that will be used in your project. You have also implemented bare-metal servo control using Timer interrupts, and a Serial command interface to control the arm. This prepares you for the upcoming checkpoint demo in the next studio session. We hope this was a fun experience!