Let’s break down the updated code using the AccelStepper library to understand how it works and why it should help reduce the jerkiness in your stepper motor movement.

**Code Breakdown**

1. **Include the AccelStepper Library:**

#include <AccelStepper.h>

This library provides advanced control features for stepper motors, including acceleration and deceleration.

1. **Define Pins and Parameters:**

const int motorPinPUL = 2; // PUL+ pin connected to Arduino digital pin 2

const int motorPinDIR = 3; // DIR+ pin connected to Arduino digital pin 3

const int motorPinENA = 4; // ENA+ pin connected to Arduino digital pin 4 (optional)

AccelStepper myStepper(AccelStepper::DRIVER, motorPinPUL, motorPinDIR);

* + motorPinPUL, motorPinDIR, and motorPinENA define the Arduino pins connected to the stepper driver.
  + AccelStepper myStepper(AccelStepper::DRIVER, motorPinPUL, motorPinDIR); initializes an AccelStepper object. The DRIVER parameter specifies that the stepper motor is controlled by a driver that takes step and direction signals.

1. **Setup Function:**

void setup() {

pinMode(motorPinENA, OUTPUT);

myStepper.setMaxSpeed(1000); // Set max speed

myStepper.setAcceleration(500); // Set acceleration

digitalWrite(motorPinENA, LOW); // Enable the driver

}

* + pinMode(motorPinENA, OUTPUT); sets the enable pin as an output.
  + myStepper.setMaxSpeed(1000); sets the maximum speed of the motor (in steps per second). You can adjust this value based on your needs.
  + myStepper.setAcceleration(500); sets the acceleration (in steps per second squared). Adjust this to ensure smooth starts and stops.
  + digitalWrite(motorPinENA, LOW); enables the driver. You can omit this if your driver is always enabled.

1. **Loop Function:**

void loop() {

myStepper.moveTo(800 \* 4); // Move the motor one full revolution clockwise

while (myStepper.distanceToGo() != 0) {

myStepper.run(); // Continuously call run() to move the motor

}

delay(1000); // Delay for 1 second

myStepper.moveTo(-800 \* 4); // Move the motor one full revolution counterclockwise

while (myStepper.distanceToGo() != 0) {

myStepper.run(); // Continuously call run() to move the motor

}

delay(1000); // Delay for 1 second

}

* + myStepper.moveTo(800 \* 4); sets the target position for the motor to move (800 steps \* 4 revolutions). This calculates to 3200 steps for one full revolution (assuming 1/8 microstepping).
  + while (myStepper.distanceToGo() != 0) { myStepper.run(); } continuously calls run() to move the motor towards the target position. This ensures smooth movement.
  + delay(1000); waits for 1 second before the next movement.
  + myStepper.moveTo(-800 \* 4); sets the target position for the motor to move one full revolution in the opposite direction.
  + The second while loop and delay(1000); perform similarly to the first, but for counterclockwise movement.

**Benefits of Using AccelStepper**

* **Smooth Acceleration and Deceleration:** The AccelStepper library handles the acceleration and deceleration of the motor, which reduces the jerkiness and provides smooth starts and stops.
* **Precise Control:** You can specify exact positions for the motor to move to, ensuring precise control over your stepper motor.
* **Versatility:** The library supports various types of stepper motors and control mechanisms, making it versatile for different applications.

**Adjusting Parameters**

* **Max Speed (setMaxSpeed(1000);)**: The maximum speed in steps per second. Adjust this value according to the capabilities of your motor and driver.
* **Acceleration (setAcceleration(500);)**: The acceleration in steps per second squared. Higher values result in faster acceleration but may cause jerkiness if set too high for the motor's capability.

By using AccelStepper and configuring the parameters correctly, you should achieve smoother and more reliable movement for your stepper motor.

To add a feature that allows specifying how many steps you want the stepper motor to rotate, you can introduce a variable that sets the number of steps for each rotation. Then, you can use this variable in the moveTo() function to control the motor's movement.

Here’s how you can modify the code to include a variable for the number of steps:

### 

### Modified Code

#include <AccelStepper.h>

// Define stepper motor parameters

const int motorPinPUL = 2; // PUL+ pin connected to Arduino digital pin 2

const int motorPinDIR = 3; // DIR+ pin connected to Arduino digital pin 3

const int motorPinENA = 4; // ENA+ pin connected to Arduino digital pin 4 (optional)

// Initialize AccelStepper

AccelStepper myStepper(AccelStepper::DRIVER, motorPinPUL, motorPinDIR);

// Define the number of steps for the desired rotation

int stepsToMove = 800; // Adjust this value as needed

void setup() {

pinMode(motorPinENA, OUTPUT);

myStepper.setMaxSpeed(1000); // Set max speed

myStepper.setAcceleration(500); // Set acceleration

digitalWrite(motorPinENA, LOW); // Enable the driver

}

void loop() {

// Move the motor clockwise by the specified number of steps

myStepper.moveTo(stepsToMove);

while (myStepper.distanceToGo() != 0) {

myStepper.run();

}

delay(1000); // Delay for 1 second

// Move the motor counterclockwise by the specified number of steps

myStepper.moveTo(-stepsToMove);

while (myStepper.distanceToGo() != 0) {

myStepper.run();

}

delay(1000); // Delay for 1 second

}

The maximum acceleration value you can set for a stepper motor using the AccelStepper library depends on several factors, including the performance characteristics of your specific stepper motor and driver, as well as the limitations of the Arduino and the AccelStepper library itself.

### Factors Influencing Maximum Acceleration

1. **Stepper Motor Specifications**: Each stepper motor has a maximum acceleration it can handle without losing steps. This depends on the motor's physical characteristics, such as its torque and inertia.
2. **Driver Capabilities**: The stepper motor driver (like the TB6600) also has specifications for the maximum current it can supply and the rate at which it can handle step pulses.
3. **Power Supply**: The voltage and current provided by your power supply will affect the maximum acceleration. A higher voltage can generally improve the performance of the motor.
4. **Mechanical Load**: The load on the motor (e.g., the weight it needs to move) will influence how quickly it can accelerate. A heavier load will generally require slower acceleration to avoid missing steps.

### Practical Considerations

While there isn't a hard limit specified by the AccelStepper library for the maximum acceleration value, it is practical to start with conservative values and gradually increase them while testing the motor's performance to ensure it does not skip steps or exhibit erratic behavior.

### Example: Testing Maximum Acceleration

Here is a modified example where you can test different acceleration values to find the maximum stable acceleration for your setup:

cpp

Copy code

#include <AccelStepper.h>

// Define stepper motor parameters

const int motorPinPUL = 2; // PUL+ pin connected to Arduino digital pin 2

const int motorPinDIR = 3; // DIR+ pin connected to Arduino digital pin 3

const int motorPinENA = 4; // ENA+ pin connected to Arduino digital pin 4 (optional)

AccelStepper myStepper(AccelStepper::DRIVER, motorPinPUL, motorPinDIR);

// Define the number of steps per revolution for your stepper motor

const int stepsPerRevolution = 200; // Example for a 1.8° step angle (200 steps/rev)

// Define the number of revolutions to move

const float revolutionsToMove = 2.0; // Example: move 2 revolutions

void setup() {

pinMode(motorPinENA, OUTPUT);

myStepper.setMaxSpeed(1000); // Set max speed (steps per second)

// Experiment with different acceleration values

myStepper.setAcceleration(1000); // Try 1000 steps per second squared

// myStepper.setAcceleration(2000); // Try 2000 steps per second squared

// myStepper.setAcceleration(5000); // Try 5000 steps per second squared

digitalWrite(motorPinENA, LOW); // Enable the driver

}

void loop() {

// Calculate the number of steps to move based on the number of revolutions

int stepsToMove = revolutionsToMove \* stepsPerRevolution;

// Move the motor clockwise by the specified number of steps

myStepper.moveTo(stepsToMove);

while (myStepper.distanceToGo() != 0) {

myStepper.run();

}

delay(1000); // Delay for 1 second

// Move the motor counterclockwise by the specified number of steps

myStepper.moveTo(-stepsToMove);

while (myStepper.distanceToGo() != 0) {

myStepper.run();

}

delay(1000); // Delay for 1 second

}

### Tips for Finding the Maximum Stable Acceleration

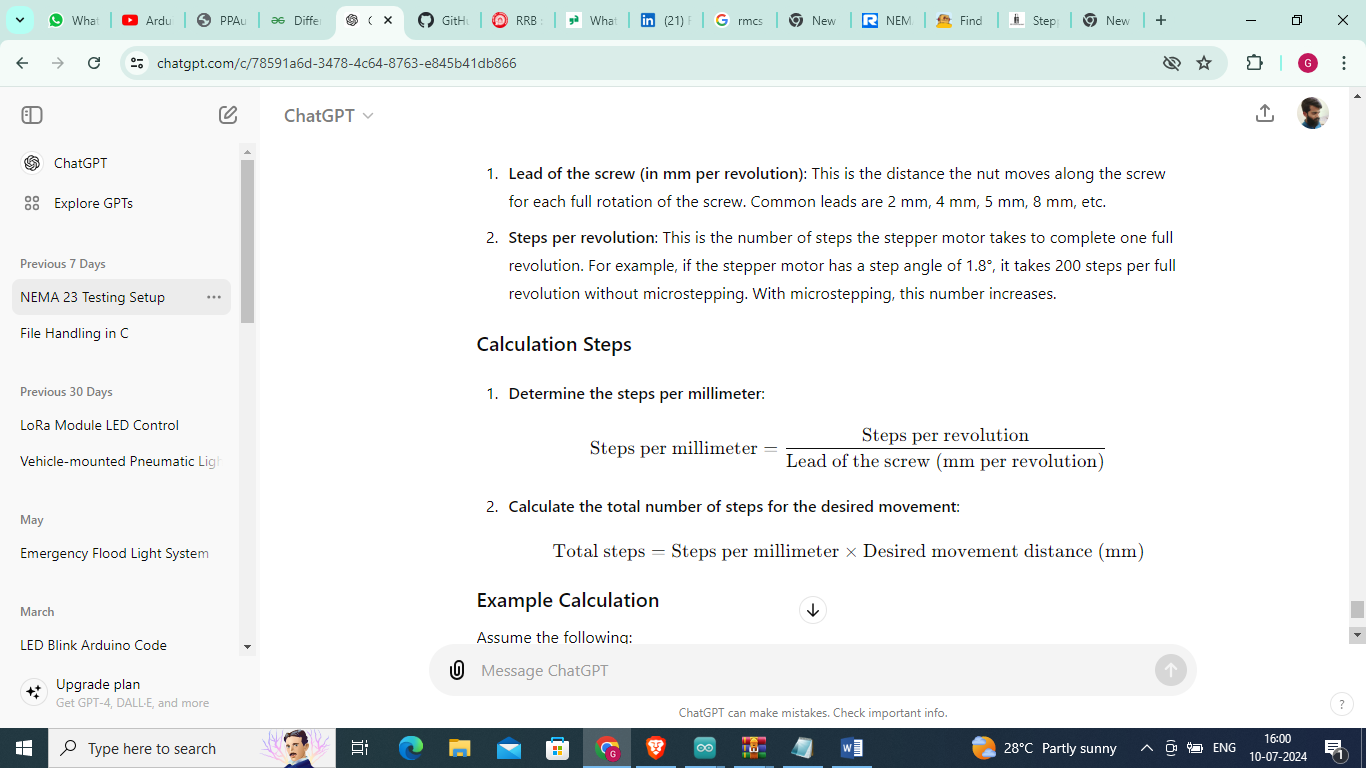
1. **Start Low**: Begin with a low acceleration value, such as 100 steps per second squared.
2. **Increment Slowly**: Gradually increase the acceleration value in small increments, such as 100 or 200 steps per second squared.
3. **Test Each Value**: For each acceleration value, run the motor through several cycles to ensure it moves smoothly without missing steps or stalling.
4. **Monitor Performance**: Pay attention to the motor's behavior, including any unusual sounds or vibrations, which might indicate it's being pushed too hard.

By carefully testing and observing the motor's performance at different acceleration values, you can determine the highest stable acceleration that your specific motor and driver setup can handle.

Steps calculation

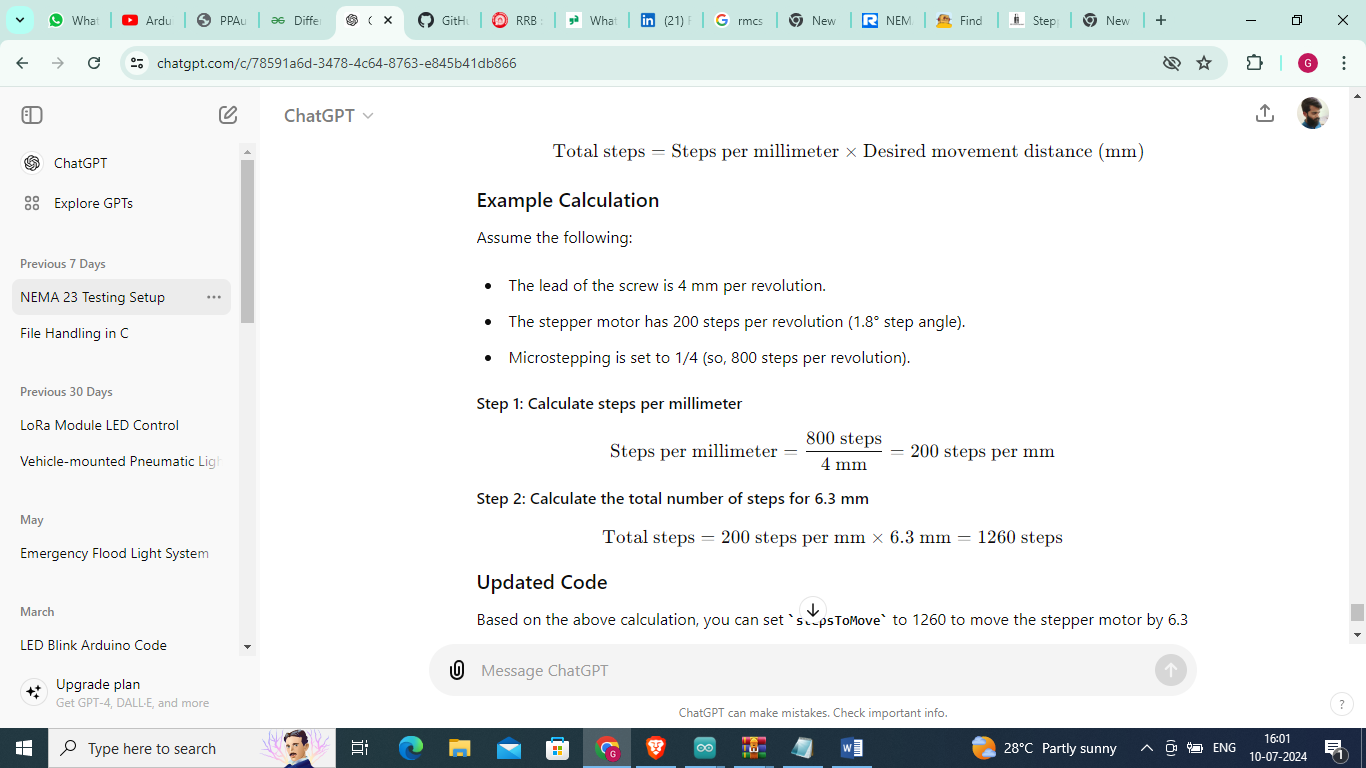
To convert the desired movement distance (e.g., 6.3 mm) into steps, you need to know the following parameters:

1. **Lead of the screw (in mm per revolution)**: This is the distance the nut moves along the screw for each full rotation of the screw. Common leads are 2 mm, 4 mm, 5 mm, 8 mm, etc.
2. **Steps per revolution**: This is the number of steps the stepper motor takes to complete one full revolution. For example, if the stepper motor has a step angle of 1.8°, it takes 200 steps per full revolution without microstepping. With microstepping, this number increases.



ssume the following:

* The lead of the screw is 4 mm per revolution.
* The stepper motor has 200 steps per revolution (1.8° step angle).
* Microstepping is set to 1/4 (so, 800 steps per revolution).



You can stop the stepper motor after it completes the specified number of steps without using the enable pin by ensuring that no further steps are commanded once the desired movement is completed. You can achieve this by exiting the loop() function or by having the loop() function do nothing once the movement is completed.

Here's the modified version of your code:

cpp

Copy code

#include <AccelStepper.h>

// Define stepper motor parameters

const int motorPinPUL = 2; // PUL+ pin connected to Arduino digital pin 2

const int motorPinDIR = 3; // DIR+ pin connected to Arduino digital pin 3

AccelStepper myStepper(AccelStepper::DRIVER, motorPinPUL, motorPinDIR);

// Define the number of steps to move

const int stepsToMove = 12600; // Calculated steps to move 63 mm

void setup() {

myStepper.setMaxSpeed(2000); // Set max speed

myStepper.setAcceleration(5000); // Set acceleration

// Move the motor clockwise by the specified number of steps

myStepper.moveTo(stepsToMove);

}

void loop() {

// Run the motor until it reaches the target position

if (myStepper.distanceToGo() != 0) {

myStepper.run();

} else {

// Do nothing, effectively stopping the motor after it completes the movement

while (true) {

// Optional: You can add some code here if you want to perform other tasks after the motor stops

}

}

}

**Explanation**

1. **Initialization**: The setup() function initializes the stepper motor and commands it to move to the target position by calling myStepper.moveTo(stepsToMove).
2. **Loop Execution**: In the loop() function, myStepper.run() is called repeatedly until myStepper.distanceToGo() returns 0, indicating the motor has reached its target position.
3. **Stopping the Motor**:
   * Once the motor has reached its target position (myStepper.distanceToGo() == 0), the else block is executed.
   * The while (true) {} loop ensures that the loop() function does not issue any further commands, effectively stopping any further motion.

This approach ensures the motor completes its specified movement and stops afterward. This method does not disable the motor driver, so the motor will still hold its position with holding torque.