For Arduino Mega

```
// Arduino Mega
#include <Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int servoPin = 8;
int PEN_DOWN = 80; // angle of servo when pen is down
int PEN_UP = 20;   // angle of servo when pen is up
Servo penServo;

float wheel_dia = 67.5; // Wheel diameter in mm
float wheel_base = 115; // Wheel base in mm
int steps_rev = 512;    // Steps per revolution for the stepper motor
int delay_time = 6;     // Delay between steps in ms

int L_stepper_pins[] = {12, 10, 9, 11};
int R_stepper_pins[] = {4, 6, 7, 5};

int fwd_mask[][4] = {{1, 0, 1, 0}, {0, 1, 1, 0}, {0, 1, 0, 1}, {1, 0, 0, 1}};
int rev_mask[][4] = {{1, 0, 0, 1}, {0, 1, 0, 1}, {0, 1, 1, 0}, {1, 0, 1, 0}};

// LCD initialization (change address if necessary)
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  Serial.begin(9600);  // Initialize Bluetooth communication

  // Initialize the stepper motor pins
  for (int pin = 0; pin < 4; pin++) {
    pinMode(L_stepper_pins[pin], OUTPUT);
    digitalWrite(L_stepper_pins[pin], LOW);
    pinMode(R_stepper_pins[pin], OUTPUT);
    digitalWrite(R_stepper_pins[pin], LOW);
  }

  // Initialize Servo and LCD
  penServo.attach(servoPin);
  penServo.write(PEN_UP);
  delay(1000);

  // Initialize LCD and show home screen
  lcd.begin();
  displayHomeScreen();
}

void loop() {
   if (Serial.available() > 0) {
    String input = Serial.readStringUntil('\n');  // Read command from Bluetooth
    input.trim();  // Remove any extra spaces or newline characters

    if (input.startsWith("forward(")) {
```

```
      int steps = parseSteps(input);
      forward(steps);
    }
    else if (input.startsWith("backward(")) {
      int steps = parseSteps(input);
      backward(steps);
    }

    else if (input.startsWith("left(")) {
      int degrees = parseSteps(input);
      left(degrees);
    }
    else if (input.startsWith("right(")) {
      int degrees = parseSteps(input);
      right(degrees);
    }
    else if (input.equals("penup()")) {
      penup();
    }
    else if (input.equals("pendown()")) {
      pendown();
    }
    else if (input.equals("tutorial()")) {
      tutorial();
    }
    else if (input.equals("examples()")) {
      examples();
    }
    else if (input.equals("stop()")) {
      done();
    }

    // After all commands are processed, display the home screen
    displayHomeScreen();
  }
}

// Function to display the home screen when the robot is turned on
void displayHomeScreen() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("  TURTLE ROBOT  ");
  lcd.setCursor(0, 1);
  lcd.print("   is Ready!!   ");
}

// Function to extract the number of steps from the command (works for both steps and degrees)
int parseSteps(String input) {
  int openBracket = input.indexOf('(');
  int closeBracket = input.indexOf(')');
  if (openBracket != -1 && closeBracket != -1) {
    String number = input.substring(openBracket + 1, closeBracket);
```

```
    return number.toInt();  // Convert the extracted string to an integer
  }
  return 0;
}

// Movement and servo control functions
int step(float distance) {
  int steps = distance * steps_rev / (wheel_dia * 3.1412);  // Convert distance to steps
  return steps;
}

void forward(float distance) {
  int steps = step(distance);
  lcd.clear();
  for (int p = 0; p <= steps / 2.414741241; p++) {
    lcd.setCursor(0, 0);
    lcd.print("Moving Forward");
    lcd.setCursor(0, 1);
    lcd.print("MM:" + String(p));
    delay(200);
    lcd.clear();
  }

}

void backward(float distance) {
  int steps = step(distance);
  lcd.clear();
  for (int p = 0; p <= steps / 2.414741241; p++) {
    lcd.setCursor(0, 0);
    lcd.print("Moving Backward");
    lcd.setCursor(0, 1);
    lcd.print("MM:" + String(p));
    delay(200);
    lcd.clear();


  }

}

void left(float degrees) {
  float rotation = degrees / 360.0;
  float distance = wheel_base * 3.1412 * rotation;
  int steps = step(distance);
  lcd.clear();
  for (int p = 0; p <= steps / 2.414741241; p++) {
    lcd.setCursor(0, 0);
    lcd.print("Turning Left");
    lcd.setCursor(0, 1);
    lcd.print("Degrees:" + String(p));
    delay(200);
```

```
    lcd.clear();


  }

}

void right(float degrees) {
  float rotation = degrees / 360.0;
  float distance = wheel_base * 3.1412 * rotation;
  int steps = step(distance);
  lcd.clear();
  for (int p = 0; p <= steps / 2.414741241; p++) {
    lcd.setCursor(0, 0);
    lcd.print("Moving Right");
    lcd.setCursor(0, 1);
    lcd.print("Degrees:" + String(p));
    delay(200);
    lcd.clear();


  }

}

void penup() {
  delay(250);
  lcd.clear();
  lcd.print("Pen Up");
  delay(3000);
  lcd.clear();
  penServo.write(PEN_UP);

  delay(250);
}

void pendown() {
  delay(250);
  lcd.clear();
  lcd.print("Pen Down");
  delay(3000);
  lcd.clear();
  penServo.write(PEN_DOWN);

  delay(250);
}

void done() {
  for (int mask = 0; mask < 4; mask++) {
    for (int pin = 0; pin < 4; pin++) {
      digitalWrite(R_stepper_pins[pin], LOW);
      digitalWrite(L_stepper_pins[pin], LOW);
```

```
    }
    delay(delay_time);
  }
}

void tutorial() {
  lcd.clear();
  lcd.print("Use forward(a)");
  lcd.setCursor(0, 1);
  lcd.print("a in mm");
  delay(3000);

  lcd.clear();
  lcd.print("Use backward(a)");
  lcd.setCursor(0, 1);
  lcd.print("a in mm");
  delay(3000);

  lcd.clear();
  lcd.print("Use left(deg)");
  lcd.setCursor(0, 1);
  lcd.print("to move left");
  delay(3000);

  lcd.clear();
  lcd.print("Use right(deg)");
  lcd.setCursor(0, 1);
  lcd.print("to move right");
  delay(3000);

  lcd.clear();
  lcd.print("Use pendown()");
  lcd.setCursor(0, 1);
  lcd.print("for pen down");
  delay(3000);

  lcd.clear();
  lcd.print("Use penup()");
  lcd.setCursor(0, 1);
  lcd.print("for pen up");
  delay(3000);


}

void examples() {
  lcd.clear();
  lcd.print("Eg 1: Home");
  lcd.setCursor(0, 1);
  lcd.print("drawing");
  delay(3000);
```

```
  lcd.clear();
  lcd.print("Eg 2: Face");
  lcd.setCursor(0, 1);
  lcd.print("drawing");
  delay(3000);

  lcd.clear();
  lcd.print("Eg 3: Boat");
  lcd.setCursor(0, 1);
  lcd.print("drawing");
  delay(3000);


}

void eg2() {
        left(90);
        forward(50);
        right(90);
        forward(50);
        right(90);
        forward(50);
        right(90);
        forward(50);
        left(45);
        backward(20);
        right(45);
        backward(20);
        penup();
        right(95);
        forward(25);
        pendown();
        penup();
        left(95);
        forward(23);
        pendown();
        penup();
        forward(30);
```

For Arduino UNO

```
// for arduino uno
#include <Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int servoPin = 8;
int PEN_DOWN = 80; // angle of servo when pen is down
int PEN_UP = 20;   // angle of servo when pen is up
Servo penServo;

float wheel_dia = 67.5; // Wheel diameter in mm
float wheel_base = 115; // Wheel base in mm
int steps_rev = 512;    // Steps per revolution for the stepper motor
int delay_time = 6;     // Delay between steps in ms

int L_stepper_pins[] = {12, 10, 9, 11};
int R_stepper_pins[] = {4, 6, 7, 5};

int fwd_mask[][4] = {{1, 0, 1, 0}, {0, 1, 1, 0}, {0, 1, 0, 1}, {1, 0, 0, 1}};
int rev_mask[][4] = {{1, 0, 0, 1}, {0, 1, 0, 1}, {0, 1, 1, 0}, {1, 0, 1, 0}};

// LCD initialization (change address if necessary)
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  Serial.begin(9600);  // Initialize Bluetooth communication

  // Initialize the stepper motor pins
  for (int pin = 0; pin < 4; pin++) {
    pinMode(L_stepper_pins[pin], OUTPUT);
    digitalWrite(L_stepper_pins[pin], LOW);
    pinMode(R_stepper_pins[pin], OUTPUT);
    digitalWrite(R_stepper_pins[pin], LOW);
  }

  // Initialize Servo and LCD
  penServo.attach(servoPin);
  penup();
  delay(1000);

  // Initialize LCD and show home screen
  lcd.begin();
  displayHomeScreen();
}

void loop() {
   if (Serial.available() > 0) {
    String input = Serial.readStringUntil('\n');  // Read command from Bluetooth
    input.trim();  // Remove any extra spaces or newline characters
```

```
    if (input.startsWith("forward(")) {
      int steps = parseSteps(input);
      forward(steps);
    }
    else if (input.startsWith("backward(")) {
      int steps = parseSteps(input);
      backward(steps);
    }

    else if (input.startsWith("left(")) {
      int degrees = parseSteps(input);
      left(degrees);
    }
    else if (input.startsWith("right(")) {
      int degrees = parseSteps(input);
      right(degrees);
    }
    else if (input.equals("penup()")) {
      penup();
    }
    else if (input.equals("pendown()")) {
      pendown();
    }
    else if (input.equals("tutorial()")) {
      tutorial();
    }
    else if (input.equals("examples()")) {
      examples();
    }
    else if (input.equals("stop()")) {
      done();
    }

    // After all commands are processed, display the home screen
    displayHomeScreen();
  }
}

// Function to display the home screen when the robot is turned on
void displayHomeScreen() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("  TURTLE ROBOT  ");
  lcd.setCursor(0, 1);
  lcd.print("   is Ready!!   ");
}

// Function to extract the number of steps from the command (works for both steps and degrees)
int parseSteps(String input) {
  int openBracket = input.indexOf('(');
  int closeBracket = input.indexOf(')');
  if (openBracket != -1 && closeBracket != -1) {
```

```
    String number = input.substring(openBracket + 1, closeBracket);
    return number.toInt();  // Convert the extracted string to an integer
  }
  return 0;
}

// Movement and servo control functions
int step(float distance) {
  int steps = distance * steps_rev / (wheel_dia * 3.1412);  // Convert distance to steps
  return steps;
}

void forward(float distance){
  int steps = step(distance);
  Serial.println(steps);
  for(int step=0; step<steps; step++){
    for(int mask=0; mask<4; mask++){
      for(int pin=0; pin<4; pin++){
        digitalWrite(L_stepper_pins[pin], rev_mask[mask][pin]);
        digitalWrite(R_stepper_pins[pin], fwd_mask[mask][pin]);
      }
      delay(delay_time);
    }
  }
}



void backward(float distance){
  int steps = step(distance);
  for(int step=0; step<steps; step++){
    for(int mask=0; mask<4; mask++){
      for(int pin=0; pin<4; pin++){
        digitalWrite(L_stepper_pins[pin], fwd_mask[mask][pin]);
        digitalWrite(R_stepper_pins[pin], rev_mask[mask][pin]);
      }
      delay(delay_time);
    }
  }
}



void right(float degrees){
  float rotation = degrees / 360.0;
  float distance = wheel_base * 3.1412 * rotation;
  int steps = step(distance);
  for(int step=0; step<steps; step++){
    for(int mask=0; mask<4; mask++){
      for(int pin=0; pin<4; pin++){
        digitalWrite(R_stepper_pins[pin], rev_mask[mask][pin]);
        digitalWrite(L_stepper_pins[pin], rev_mask[mask][pin]);
      }
      delay(delay_time);
```

```
    }
  }
}


void left(float degrees){
  float rotation = degrees / 360.0;
  float distance = wheel_base * 3.1412 * rotation;
  int steps = step(distance);
  for(int step=0; step<steps; step++){
    for(int mask=0; mask<4; mask++){
      for(int pin=0; pin<4; pin++){
        digitalWrite(R_stepper_pins[pin], fwd_mask[mask][pin]);
        digitalWrite(L_stepper_pins[pin], fwd_mask[mask][pin]);
      }
      delay(delay_time);
    }
  }
}


void done(){ // unlock stepper to save battery
  for(int mask=0; mask<4; mask++){
    for(int pin=0; pin<4; pin++){
      digitalWrite(R_stepper_pins[pin], LOW);
      digitalWrite(L_stepper_pins[pin], LOW);
    }
    delay(delay_time);
  }
}


void penup() {
  delay(250);
  lcd.clear();
  lcd.print("Pen Up");
  delay(2000);
  lcd.clear();
  penServo.write(PEN_UP);

  delay(250);
}

void pendown() {
  delay(250);
  lcd.clear();
  lcd.print("Pen Down");
  delay(2000);
  lcd.clear();
  penServo.write(PEN_DOWN);

  delay(250);
```

```
}


void tutorial() {
  lcd.clear();
  lcd.print("Use forward(a)");
  lcd.setCursor(0, 1);
  lcd.print("a in mm");
  delay(2000);

  lcd.clear();
  lcd.print("Use backward(a)");
  lcd.setCursor(0, 1);
  lcd.print("a in mm");
  delay(2000);

  lcd.clear();
  lcd.print("Use left(deg)");
  lcd.setCursor(0, 1);
  lcd.print("to move left");
  delay(2000);

  lcd.clear();
  lcd.print("Use right(deg)");
  lcd.setCursor(0, 1);
  lcd.print("to move right");
  delay(2000);

  lcd.clear();
  lcd.print("Use pendown()");
  lcd.setCursor(0, 1);
  lcd.print("for pen down");
  delay(2000);

  lcd.clear();
  lcd.print("Use penup()");
  lcd.setCursor(0, 1);
  lcd.print("for pen up");
  delay(2000);


}

void examples() {
  lcd.clear();
  lcd.print("Eg 1: Home");
  lcd.setCursor(0, 1);
  lcd.print("drawing");
  delay(2000);

  lcd.clear();
  lcd.print("Eg 2: Face");
```

```
  lcd.setCursor(0, 1);
  lcd.print("drawing");
  delay(2000);

  lcd.clear();
  lcd.print("Eg 3: Boat");
  lcd.setCursor(0, 1);
  lcd.print("drawing");
  delay(2000);


}
```

For Processing

```
import processing.serial.*;
import javax.swing.JOptionPane;

Serial myPort;
PFont font;
boolean portAvailable;
String commandText = "";
boolean tutorialPressed = false;
boolean examplesPressed = false;
boolean sendPressed = false;

void setup() {
  fullScreen();  // Set to full screen
  font = createFont("Roboto Medium", 18, true);
  textFont(font);

  // Check for available serial ports
  String[] portList = Serial.list();
  printArray(portList); // Print out the available ports in the console

  if (portList.length > 0) {
    String portSelection = selectPortDialog(portList); // Show port selection dialog
    if (portSelection != null) {
      myPort = new Serial(this, portSelection, 9600);
      myPort.clear();
      portAvailable = true;
      println("Connected to: " + portSelection);
    } else {
      println("No port selected. Running without serial connection.");
      portAvailable = false;
    }
  } else {
    println("No serial ports found. Running without serial connection.");
    portAvailable = false;
  }
}

void draw() {
  background(245, 248, 255);  // Light background for professional look

  // Title with subtle shadow
  fill(50, 50, 100);
  textSize(42);
  textAlign(CENTER);
  text("TURTLE ROBOT", width / 2, 80);

  fill(120);
  textSize(20);
  text("made by Arnav & Malank", width / 2, 120);
```

```
  // Buttons with pressed effect
  drawButton(width / 4 - 90, 160, 180, 60, "TUTORIAL", tutorialPressed);
  drawButton(3 * width / 4 - 90, 160, 180, 60, "EXAMPLES", examplesPressed);

  // Command Input Box Label
  fill(50, 50, 100);
  textSize(20);
  textAlign(LEFT, CENTER);
  text("Enter your commands here:", width / 2 - 180, 270);

  // Command Input Box
  fill(255);
  stroke(200);
  strokeWeight(1);
  rect(width / 2 - 180, 290, 480, 240, 12);
  fill(80);
  textSize(18);
  textAlign(LEFT, TOP);
  text(commandText, width / 2 - 170, 300);

  // Send Button with pressed effect
  drawButton(width / 2 + 200, 540, 100, 50, "SEND", sendPressed);
}

// Function to draw a button with pressed effect
void drawButton(float x, float y, float w, float h, String label, boolean pressed) {
  // Button Shadow and Background based on pressed state
  if (pressed) {
    fill(70, 130, 230);  // Darker color when pressed
    noStroke();
  } else {
    fill(100, 170, 255);
    stroke(180, 180, 200);
    strokeWeight(2);
  }
  rect(x, y, w, h, 15);

  // Button Text
  fill(255);
  textSize(18);
  textAlign(CENTER, CENTER);
  text(label, x + w / 2, y + h / 2);
}

void mousePressed() {
  // Check if Tutorial button is clicked
  if (mouseX > width / 4 - 90 && mouseX < width / 4 + 90 && mouseY > 160 && mouseY < 220)
{
    tutorialPressed = true;
    sendSerialCommand("tutorial");
  }
```

```
  // Check if Examples button is clicked
  if (mouseX > 3 * width / 4 - 90 && mouseX < 3 * width / 4 + 90 && mouseY > 160 && mouseY
< 220) {
    examplesPressed = true;
    sendSerialCommand("examples");
  }

  // Check if Send button is clicked
  if (mouseX > width / 2 + 200 && mouseX < width / 2 + 300 && mouseY > 540 && mouseY <
590) {
    sendPressed = true;
    sendSerialCommand(commandText);
    commandText = "";  // Clear the commandText after sending
  }
}

void mouseReleased() {
  // Reset button press states
  tutorialPressed = false;
  examplesPressed = false;
  sendPressed = false;
}

void keyTyped() {
  if (key == BACKSPACE) {
    if (commandText.length() > 0) {
      commandText = commandText.substring(0, commandText.length() - 1);
    }
  } else if (key == ENTER || key == RETURN) {
    commandText += "\n";
  } else {
    commandText += key;
  }
}

void sendSerialCommand(String command) {
  if (portAvailable) {
    myPort.write(command + "\n");
    println("Command sent: " + command);
  } else {
    println("Port not available. Simulating sending:\n" + command);
  }
}

// Function to display a COM port selection dialog
String selectPortDialog(String[] ports) {
  Object selection = JOptionPane.showInputDialog(null,
      "Select a COM Port",
      "COM Port Selection",
      JOptionPane.QUESTION_MESSAGE,
      null,
      ports,
```

```
      ports[0]);

  if (selection != null) {
    return selection.toString();
  }
  return null;
}
```