Arduino Code:

```
// Libary
#include <AccelStepper.h>
#include <LiquidCrystal_I2C.h>
// pins
#define size 2 //interupt pin for changing size
#define step 10 // step pin for stepper controller
#define dir 11 // direction pin for stepper controller
#define emergency 3 // interput pin for stopping at end of travel path
#define controlbutton 4 // start pause
#define greenLED 5 // connected to green pin on led
#define redLED 6 // connected to red pin on led
#define esp 7 // connected to esp32 for remote control
#define flowrate_but 8 // controls whether flowrate is controlled by potentiometer
#define forward 12 // manual forward
#define backward 13 // manual reverse
int analog = A0; // reading potentiometer
int read = 0; // potentiometer value
double speed; // speed set by potentiometer
double speed_s; //set program speed
volatile bool sizeChangeFlag = false; //interupt contingency
volatile bool estop = false; // interupt contingency
unsigned long lastLCDUpdate = 0; // Variable to track the last update time
const unsigned long lcdInterval = 1000; // Update LCD every 1 second

// flowrate
#define flowrate 5 // mL
#define twenty_mL 2.28018 // volume pushed per revolution
#define ten_mL 1.40992 // volume pushed per revolution
#define microstep 3200.00 //microstep resolution

AccelStepper stepper(AccelStepper::DRIVER, step, dir); //initialize stepper motor

LiquidCrystal_I2C lcd(0x27,16,2); //contact lcd screen

void setup() {

  //pins
  pinMode(controlbutton, INPUT_PULLUP);
  pinMode(greenLED, OUTPUT);
  pinMode(redLED, OUTPUT);
  pinMode(esp, INPUT_PULLUP);
  pinMode(emergency, INPUT_PULLUP);
```

```
  pinMode(flowrate_but, INPUT_PULLUP);
  pinMode(forward, INPUT_PULLUP);
  pinMode(backward, INPUT_PULLUP);
  pinMode(size, INPUT_PULLUP);

  //interupt
  attachInterrupt(digitalPinToInterrupt(emergency), eStopISR, FALLING);
  attachInterrupt(digitalPinToInterrupt(size), sizeChangeISR, CHANGE);

  stepper.setMaxSpeed(1000); // set maximum stepper motor
  speed_s = (microstep/60) * (flowrate/twenty_mL);
  speed = speed_s;

  lcd.init();
  lcd.clear();
  lcd.backlight();      // Make sure backlight is on

  // Welcome message
  lcd.setCursor(3,0);   //Set cursor to character 2 on line 0
  lcd.print("Welcome ||");
  lcd.setCursor(11,1);   //Move cursor to character 2 on line 1
  lcd.print("_/");

//Serial.begin(9600); // debugging purposes

delay(5000); //start up
}

void loop() {
  int i = 0;
  int j;
// start/pause function
  while (digitalRead(controlbutton) == LOW && digitalRead(emergency)==HIGH) {
    ++i;
    stepper.runSpeed();
    digitalWrite(greenLED, 1);
    digitalWrite(redLED, 0);
    if (i==1) {
    lcd.clear();
    lcd.setCursor(1,0);   //Set cursor to character 2 on line 0
    lcd.print("Moving Forward");
    }
  }
  // start/pause function remote
```

```
  while (digitalRead(esp) == LOW && digitalRead(emergency)==HIGH) {
    ++i;
    stepper.runSpeed();
    digitalWrite(greenLED, 1);
    digitalWrite(redLED, 0);
    if (i==1) {
    lcd.clear();
    lcd.setCursor(1,0);   //Set cursor to character 2 on line 0
    lcd.print("Moving Forward");
    }

  }
  // idle control, moving forward or backward
  if (digitalRead(esp) == HIGH && digitalRead(controlbutton) == HIGH &&
digitalRead(emergency) == HIGH) {

    //idle
    digitalWrite(greenLED, 1);
    digitalWrite(redLED, 1);
    //manual forward control
    while (digitalRead(forward) == LOW && digitalRead(emergency)==HIGH) {
      ++i;
      stepper.setSpeed(600);
      stepper.runSpeed();
      digitalWrite(greenLED, 1);
      digitalWrite(redLED, 0);
    if (i==1) {
    lcd.clear();
    lcd.setCursor(4,0);   //Set cursor to character 2 on line 0
    lcd.print("Forward");
    }
    }
    i=0;
    //manual reverse, capable of being enacted in emergency stop configuration for easy
rewinding
    while (digitalRead(backward) == LOW) {
      ++i;
      stepper.setSpeed(-600);
      stepper.runSpeed();
      digitalWrite(greenLED, 1);
      digitalWrite(redLED, 0);
      if (i==1) {
    lcd.clear();
    lcd.setCursor(4,0);   //Set cursor to character 2 on line 0
```

```
  lcd.print("Backward");
  }
  }

    unsigned long currentMillis = millis();
  if (currentMillis - lastLCDUpdate >= lcdInterval) {
  lcd.clear();
  lcd.setCursor(6,0);   //Set cursor to character 2 on line 0
  lcd.print("Idle");
      lastLCDUpdate = currentMillis; // Reset the timer
  }
 }
// control speed with potentiometer
 while(digitalRead(flowrate_but) == LOW && digitalRead(emergency)==HIGH) {
    read = analogRead(analog); //read potentiomter
  double temp = read / 1024.00 * 10.0 + 1; // translate reading into mL

  // use proper size
  if (size == HIGH) {
  speed = (microstep/60.0) * (temp/twenty_mL);
  } else {
      speed = (microstep/60.0) * (temp/ten_mL);
  }
  stepper.setSpeed(speed);

  // run speed if pressed
  if(digitalRead(controlbutton)==LOW) {
    stepper.runSpeed();
    digitalWrite(greenLED, 1);
    digitalWrite(redLED, 0);
  }
  unsigned long currentMillis = millis();
  if (currentMillis - lastLCDUpdate >= lcdInterval) {
      lcd.clear();
      lcd.setCursor(4, 0); // Set cursor position
      lcd.print("speed(mL)");
      lcd.setCursor(6, 1); // Set cursor position
      lcd.print(temp);
      lastLCDUpdate = currentMillis; // Reset the timer
  }
 }

 //go backward at any point including after an emergency stop
 i=0;
```

```
  while (digitalRead(backward) == LOW) {
      ++i;
      stepper.setSpeed(-600);
      stepper.runSpeed();
      digitalWrite(greenLED, 1);
      digitalWrite(redLED, 0);
      if (i==1) {
    lcd.clear();
    lcd.setCursor(4,0);   //Set cursor to character 2 on line 0
    lcd.print("Backward");
    }
  }

// return to embedded preset speed
  if(digitalRead(flowrate_but) == HIGH) {
    stepper.setSpeed(speed_s);
  }

// interupt for size change
  if (sizeChangeFlag) {
    sizeChange(); // Call the function when the flag is set
    delay(3500);
    sizeChangeFlag = false; // Reset the flag
  }

// interupt for emergency stop
  if (estop) {
    outOfLiquid();
    estop = false; // Reset the flag
  }
}

// reset interupt flags
void sizeChangeISR() {
  sizeChangeFlag = true; // Set the flag
}
void eStopISR() {
  estop = true; // Set the flag
}

// procedure for size change
void sizeChange() {
   if(digitalRead(size) == LOW && digitalRead(controlbutton)==HIGH &&
digitalRead(esp)==HIGH) {
```

```
    speed_s = (microstep/60) * (flowrate/ten_mL);
    lcd.clear();
    lcd.setCursor(4,0);
    lcd.print("size(mL):");
    lcd.setCursor(7,1);
    lcd.print("10");
  }
   if (digitalRead(size) == HIGH && digitalRead(controlbutton)==HIGH &&
digitalRead(esp)==HIGH){
    speed_s = (microstep/60) * (flowrate/twenty_mL);
    lcd.clear();
    lcd.setCursor(4,0);
    lcd.print("size(mL):");
    lcd.setCursor(7,1);
    lcd.print("20");
  }
}

// emergency stop
void outOfLiquid() {
  int i = 0;
  stepper.stop();
  digitalWrite(greenLED, 0);
  digitalWrite(redLED, 1);
  lcd.clear();
  lcd.setCursor(6,0);
  lcd.print("STOP");
  lcd.setCursor(2,1);
  lcd.print("Out of Fluid"); // Call the function when the flag is set

  // stay trapped in emergency stop until system is paused
  // note that system can be reversed in this state
  while (digitalRead(esp) == LOW || digitalRead(controlbutton) == LOW) {
    stepper.stop();
    unsigned long currentMillis = millis();
    if (currentMillis - lastLCDUpdate >= lcdInterval) {
        lcd.clear();
    lcd.setCursor(6,0);
    lcd.print("STOP");
    lcd.setCursor(2,1);
    lcd.print("Out of Fluid"); // Call the function when the flag is set
        lastLCDUpdate = currentMillis; // Reset the timer
    }
    }
```

```
  }
```

ESP32 RECEIVER CODE:

```
#include <esp_now.h>
#include <WiFi.h>
#include <AccelStepper.h>

#define CHANNEL 1

// Init ESP Now with fallback
void InitESPNow() {
  WiFi.disconnect();
  if (esp_now_init() == ESP_OK) {
    Serial.println("ESPNow Init Success");
  }
  else {
    Serial.println("ESPNow Init Failed");
    // Retry InitESPNow, add a counte and then restart?
    // InitESPNow();
    // or Simply Restart
    ESP.restart();
  }
}

// config AP SSID
void configDeviceAP() {
  const char *SSID = "Slave_1";
  bool result = WiFi.softAP(SSID, "Slave_1_Password", CHANNEL, 0);
  if (!result) {
    Serial.println("AP Config failed.");
  } else {
    Serial.println("AP Config Success. Broadcasting with AP: " + String(SSID));
    Serial.print("AP CHANNEL "); Serial.println(WiFi.channel());
  }
}

void setup() {
  // pins
  pinMode(27, OUTPUT);
  digitalWrite(27, HIGH);


  // wifi
```

```
  Serial.begin(115200);
  Serial.println("ESPNow/Basic/Slave Example");
  //Set device in AP mode to begin with
  WiFi.mode(WIFI_AP);
  // configure device AP mode
  configDeviceAP();
  // This is the mac address of the Slave in AP Mode
  Serial.print("AP MAC: "); Serial.println(WiFi.softAPmacAddress());
  // Init ESPNow with a fallback logic
  InitESPNow();
  // Once ESPNow is successfully Init, we will register for recv CB to
  // get recv packer info.
  esp_now_register_recv_cb(OnDataRecv);

}

// callback when data is recv from Master
void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "C8:2E:18:25:16:D1",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  Serial.print("Last Packet Recv from: "); Serial.println(macStr);
  Serial.print("Last Packet Recv Data: "); Serial.println(*data);
  Serial.println("");
  if (*data == 1) {
  digitalWrite(27, LOW);
  } else {
    digitalWrite(27, HIGH);
  }
}

void loop() {
  // debugging
//  digitalWrite(27, HIGH);
 // delay (3000);
  //digitalWrite(27, LOW);
  //delay(3000);
}
```

ESP32 TRANSMITTER CODE:

```
#include <esp_now.h>
#include <WiFi.h>
```

```
#include <esp_wifi.h> // only for esp_wifi_set_channel()

// Global copy of slave
esp_now_peer_info_t slave;
#define CHANNEL 1
#define PRINTSCANRESULTS 0
#define DELETEBEFOREPAIR 0


// Init ESP Now with fallback
void InitESPNow() {
  WiFi.disconnect();
  if (esp_now_init() == ESP_OK) {
    Serial.println("ESPNow Init Success");
  }
  else {
    Serial.println("ESPNow Init Failed");
    // Retry InitESPNow, add a counte and then restart?
    // InitESPNow();
    // or Simply Restart
    ESP.restart();
  }
}

// Scan for slaves in AP mode
void ScanForSlave() {
  int16_t scanResults = WiFi.scanNetworks(false, false, false, 300, CHANNEL); // Scan only on
one channel
  // reset on each scan
  bool slaveFound = 0;
  memset(&slave, 0, sizeof(slave));

  Serial.println("");
  if (scanResults == 0) {
    Serial.println("No WiFi devices in AP Mode found");
  } else {
    Serial.print("Found "); Serial.print(scanResults); Serial.println(" devices ");
    for (int i = 0; i < scanResults; ++i) {
      // Print SSID and RSSI for each device found
      String SSID = WiFi.SSID(i);
      int32_t RSSI = WiFi.RSSI(i);
      String BSSIDstr = WiFi.BSSIDstr(i);
      if (PRINTSCANRESULTS) {
        Serial.print(i + 1);
```

```
      Serial.print(": ");
      Serial.print(SSID);
      Serial.print(" (");
      Serial.print(RSSI);
      Serial.print(")");
      Serial.println("");
    }
    delay(10);
    // Check if the current device starts with `Slave`
    if (SSID.indexOf("Slave") == 0) {
      // SSID of interest
      Serial.println("Found a Slave.");
      Serial.print(i + 1); Serial.print(": "); Serial.print(SSID); Serial.print(" [");
Serial.print(BSSIDstr); Serial.print("]"); Serial.print(" ("); Serial.print(RSSI); Serial.print(")");
Serial.println("");
      // Get BSSID => Mac Address of the Slave
      int mac[6];
      if ( 6 == sscanf(BSSIDstr.c_str(), "%x:%x:%x:%x:%x:%x",  &mac[0], &mac[1], &mac[2],
&mac[3], &mac[4], &mac[5] ) ) {
        for (int ii = 0; ii < 6; ++ii ) {
          slave.peer_addr[ii] = (uint8_t) mac[ii];
        }
      }

      slave.channel = CHANNEL; // pick a channel
      slave.encrypt = 0; // no encryption

      slaveFound = 1;
      // we are planning to have only one slave in this example;
      // Hence, break after we find one, to be a bit efficient
      break;
    }
  }
 }

 if (slaveFound) {
  Serial.println("Slave Found, processing..");
 } else {
  Serial.println("Slave Not Found, trying again.");
 }

 // clean up ram
 WiFi.scanDelete();
}
```

```
// Check if the slave is already paired with the master.
// If not, pair the slave with master
bool manageSlave() {
  if (slave.channel == CHANNEL) {
    if (DELETEBEFOREPAIR) {
      deletePeer();
    }

    Serial.print("Slave Status: ");
    // check if the peer exists
    bool exists = esp_now_is_peer_exist(slave.peer_addr);
    if ( exists) {
      // Slave already paired.
      Serial.println("Already Paired");
      return true;
    } else {
      // Slave not paired, attempt pair
      esp_err_t addStatus = esp_now_add_peer(&slave);
      if (addStatus == ESP_OK) {
        // Pair success
        Serial.println("Pair success");
        return true;
      } else if (addStatus == ESP_ERR_ESPNOW_NOT_INIT) {
        // How did we get so far!!
        Serial.println("ESPNOW Not Init");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_ARG) {
        Serial.println("Invalid Argument");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_FULL) {
        Serial.println("Peer list full");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_NO_MEM) {
        Serial.println("Out of memory");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_EXIST) {
        Serial.println("Peer Exists");
        return true;
      } else {
        Serial.println("Not sure what happened");
        return false;
      }
    }
```

```
  } else {
    // No slave found to process
    Serial.println("No Slave found to process");
    return false;
  }
}

void deletePeer() {
  esp_err_t delStatus = esp_now_del_peer(slave.peer_addr);
  Serial.print("Slave Delete Status: ");
  if (delStatus == ESP_OK) {
    // Delete success
    Serial.println("Success");
  } else if (delStatus == ESP_ERR_ESPNOW_NOT_INIT) {
    // How did we get so far!!
    Serial.println("ESPNOW Not Init");
  } else if (delStatus == ESP_ERR_ESPNOW_ARG) {
    Serial.println("Invalid Argument");
  } else if (delStatus == ESP_ERR_ESPNOW_NOT_FOUND) {
    Serial.println("Peer not found.");
  } else {
    Serial.println("Not sure what happened");
  }
}

uint8_t data = 0;
// send data
void sendData() {
  //data++;
  const uint8_t *peer_addr = slave.peer_addr;
  Serial.print("Sending: "); Serial.println(data);
  esp_err_t result = esp_now_send(peer_addr, &data, sizeof(data));
  Serial.print("Send Status: ");
  if (result == ESP_OK) {
    Serial.println("Success");
  } else if (result == ESP_ERR_ESPNOW_NOT_INIT) {
    // How did we get so far!!
    Serial.println("ESPNOW not Init.");
  } else if (result == ESP_ERR_ESPNOW_ARG) {
    Serial.println("Invalid Argument");
  } else if (result == ESP_ERR_ESPNOW_INTERNAL) {
    Serial.println("Internal Error");
  } else if (result == ESP_ERR_ESPNOW_NO_MEM) {
    Serial.println("ESP_ERR_ESPNOW_NO_MEM");
```

```
  } else if (result == ESP_ERR_ESPNOW_NOT_FOUND) {
    Serial.println("Peer not found.");
  } else {
    Serial.println("Not sure what happened");
  }
}

// callback when data is sent from Master to Slave
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "C8:2E:18:25:16:D1", // C8:2E:18:25:16:D1
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  Serial.print("Last Packet Sent to: "); Serial.println(macStr);
  Serial.print("Last Packet Send Status: "); Serial.println(status ==
ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup() {
  pinMode(26, INPUT_PULLUP);
  Serial.begin(115200);
  //Set device in STA mode to begin with
  WiFi.mode(WIFI_STA);
  esp_wifi_set_channel(CHANNEL, WIFI_SECOND_CHAN_NONE);
  Serial.println("ESPNow/Basic/Master Example");
  // This is the mac address of the Master in Station Mode
  Serial.print("STA MAC: "); Serial.println(WiFi.macAddress());
  Serial.print("STA CHANNEL "); Serial.println(WiFi.channel());
  // Init ESPNow with a fallback logic
  InitESPNow();
  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);
}

void loop() {
  if (digitalRead(26)== HIGH) {
    data = 0;
  } else {
    data = 1;
  }
  // In the loop we scan for slave
  ScanForSlave();
  // If Slave is found, it would be populate in `slave` variable
  // We will check if `slave` is defined and then we proceed further
```

```
  if (slave.channel == CHANNEL) { // check if slave channel is defined
    // `slave` is defined
    // Add slave as peer if it has not been added already
    bool isPaired = manageSlave();
    if (isPaired) {
      // pair success or already paired
      // Send data to device
      sendData();
    } else {
      // slave pair failed
      Serial.println("Slave pair failed!");
    }
  }
  else {
    // No slave found to process
  }

  // wait for 3seconds to run the logic again
  //delay(300);
}
```