Contents

[**V2 Update** 2](#_Toc197021231)

[**V3 Update** 2](#_Toc197021232)

[**Appendix** 2](#_Toc197021233)

[Rayan\_V3\_oneMotor 2](#_Toc197021234)

[Ryan\_V3\_fiveMotors 5](#_Toc197021235)

[Rayan\_V3\_fiveMotorsEMG 11](#_Toc197021236)

# **V2 Update**

* All the code works as intended
* Motors rotate
* Annoying to wire (ask Orion)
* Waiting on mechanical

# **V3 Update**

Upgrades from V2 include:

1. Individual finger control
2. Pressure sensing and pressure control
3. Switched from rotational motors to linear actuators

Where we are right now:

* Semi functional prototype with functioning pressure sensing and individual control
* The thumb, pointer finger, and ring finger all have current sensing. This current sensor is used to change pressure tolerances
* Has bio pill capabilities (aka reading electrical signals of muscle to function)
* Code is functioning but we need to update it for potentiometer

Things that need to be done:

* Battery integration and management
  + Currently runs off wall power 12V with voltage step down to 5V
* Add potentiometer (changes the strength of linear actuators when engaged)
  + Allows linear actuators to speed up or slow down
* A preassembled PCP with all (ish) components
  + Battery will be independent of the board and potentiometer will be attached to it

# **Appendix**

## Rayan\_V3\_oneMotor

#include <math.h> // For the sin function  
#include <ESP32Servo.h> // For the SERVO  
#define PWM  4  
#define AI1\_PIN  26  
#define AI2\_PIN  25  
#define SWITCHPIN 15  
#define CURRENT1 32  
#define CURRENT2 33  
  
#define PWM\_CHANNEL 4  
#define PWM\_RESOLUTION 8  
#define PWM\_FREQUENCY 4000  
  
int PWM\_val = 1000;  
float current1;  
float current2;  
float currentDiff;  
bool switchOff = 0;  
  
//avging constants  
const int bufferSize = 10; // Size of the moving average buffer, adjust as needed  
float buffer[bufferSize];  
int bufferIndex = 0;  
float sum = 0;  
float movingAverage = 0;  
  
// the setup function runs once when you press reset or power the board  
void setup() {  
  // initialize digital pins as outputs.  
  pinMode(PWM, OUTPUT);  
  pinMode(AI1\_PIN, OUTPUT);  
  pinMode(AI2\_PIN, OUTPUT);  
  pinMode(SWITCHPIN, INPUT);  
  pinMode(CURRENT1, INPUT);  
  pinMode(CURRENT2, INPUT);  
  
  // Setup PWM  
  ledcAttach(PWM\_CHANNEL, PWM\_FREQUENCY, PWM\_RESOLUTION);  
  ledcAttach(PWM, PWM\_CHANNEL, PWM\_RESOLUTION);  
  
  //servo stuff  
  ESP32PWM::allocateTimer(0);  
  ESP32PWM::allocateTimer(1);  
  ESP32PWM::allocateTimer(2);  
  ESP32PWM::allocateTimer(3);  
  
  for (int i = 0; i < bufferSize; i++) {  
  buffer[i] = 0;  
  }  
  
  // for plotting current sensor  
  Serial.begin(115200);  
  
}  
  
// the loop function runs over and over again forever  
void loop() {  
  ledcWrite(PWM\_CHANNEL, PWM\_val);   
    
  if(digitalRead(SWITCHPIN) == HIGH) {  
    digitalWrite(AI1\_PIN, LOW);  
    digitalWrite(AI2\_PIN, HIGH);  
    if(switchOff){ // not currently actually doing anything and needs to be fixed  
      PWM\_val = 100;  
    }  
  } // actuator will stop extending automatically when reaching the limit  
  else{  
  // retracts the actuator  
    switchOff = 0;  
    PWM\_val = 1000;  
    digitalWrite(AI1\_PIN, HIGH);  
    digitalWrite(AI2\_PIN, LOW);  
  } // actuator will stop extending automatically when reaching the limit  
  
  // Current sensor with boxcar average  
  current1 = analogRead(CURRENT1);  
  current2 = analogRead(CURRENT2);  
  currentDiff = current2 - current1;  
  sum -= buffer[bufferIndex];  
  buffer[bufferIndex] = currentDiff;  
  sum += buffer[bufferIndex];  
  bufferIndex = (bufferIndex + 1) % bufferSize;  
  movingAverage = sum / bufferSize;  
  if(movingAverage > 80){  
    switchOff = 1;  
  }  
  
  Serial.println(movingAverage);  
  
  delay(10);  
}

## Ryan\_V3\_fiveMotors

// Overall  
#include <math.h> // For the sin function  
#define SWITCHPIN 15  
#define PWM\_RESOLUTION 8  
#define PWM\_FREQUENCY 4000  
int switchState;  
const int maxPWM = 1000;  
int strength = 0; // 0=weak, 1=medium, 2=strong  
  
// Index and middle finger variables  
#define AI1\_INDEX 26  
#define AI2\_INDEX 25  
#define BI1\_MIDDLE 2  
#define BI2\_MIDDLE 0  
#define PWM\_IM 4  
#define C1\_IM 32  
#define C2\_IM 33  
  
const int bufferSizeIM = 10; // Size of the moving average buffer, adjust as needed  
float bufferIM[bufferSizeIM];  
int bufferIndexIM = 0;  
float sumIM = 0;  
float movingAverageIM = 0;  
  
int PWM\_IM\_val = 1000;  
float voltage1IM;  
float voltage2IM;  
float currentIM;  
bool thresholdReachedIM = 0;  
int currentThresholdIM[3] = {90, 130, 999}; //tailored to specific motor  
int minPWM\_IM[3] = {100, 500, maxPWM}; //tailored to specific motor  
  
// Ring and pinky finger variables  
#define AI1\_RING 14  
#define AI2\_RING 27  
#define BI1\_PINKY 21  
#define BI2\_PINKY 22  
#define PWM\_RP 17  
#define C1\_RP 38  
#define C2\_RP 39  
  
const int bufferSizeRP = 10; // Size of the moving average buffer, adjust as needed  
float bufferRP[bufferSizeRP];  
int bufferIndexRP = 0;  
float sumRP = 0;  
float movingAverageRP = 0;  
  
int PWM\_RP\_val = 1000;  
float voltage1RP;  
float voltage2RP;  
float currentRP;  
bool thresholdReachedRP = 0;  
int currentThresholdRP[3] = {20, 60, 999}; //tailored to specific motor  
int minPWM\_RP[3] = {100, 200, maxPWM}; //tailored to specific motor  
  
// Thumb variables  
#define AI1\_THUMB 23  
#define AI2\_THUMB 19  
#define PWM\_T 16  
#define C1\_T 36  
#define C2\_T 37  
  
const int bufferSizeT = 10; // Size of the moving average buffer, adjust as needed  
float bufferT[bufferSizeT];  
int bufferIndexT = 0;  
float sumT = 0;  
float movingAverageT = 0;  
  
int PWM\_T\_val = 1000;  
float voltage1T;  
float voltage2T;  
float currentT;  
bool thresholdReachedT = 0;  
int currentThresholdT[3] = {100, 160, 999}; //tailored to specific motor  
int minPWM\_T[3] = {100, 700, maxPWM}; //tailored to specific motor  
  
//--------------------------------------------------------------------------------  
void setup() {  
  // initialize digital pins as outputs.  
  pinMode(SWITCHPIN, INPUT);  
  
  pinMode(PWM\_IM, OUTPUT);  
  pinMode(PWM\_RP, OUTPUT);  
  pinMode(PWM\_T, OUTPUT);  
  
  pinMode(AI1\_INDEX, OUTPUT);  
  pinMode(AI2\_INDEX, OUTPUT);  
  pinMode(BI1\_MIDDLE, OUTPUT);  
  pinMode(BI2\_MIDDLE, OUTPUT);  
  pinMode(AI1\_RING, OUTPUT);  
  pinMode(AI2\_RING, OUTPUT);  
  pinMode(BI1\_PINKY, OUTPUT);  
  pinMode(BI2\_PINKY, OUTPUT);  
  pinMode(AI1\_THUMB, OUTPUT);  
  pinMode(AI2\_THUMB, OUTPUT);  
  
  pinMode(C1\_IM, INPUT);  
  pinMode(C2\_IM, INPUT);  
  pinMode(C1\_RP, INPUT);  
  pinMode(C2\_RP, INPUT);  
  pinMode(C1\_T, INPUT);  
  pinMode(C2\_T, INPUT);  
  
  // Setup PWMs  
  ledcAttach(PWM\_IM, PWM\_FREQUENCY, PWM\_RESOLUTION); // I changed this from the previous version so suspect error here  
  ledcAttach(PWM\_RP, PWM\_FREQUENCY, PWM\_RESOLUTION);  
  ledcAttach(PWM\_T, PWM\_FREQUENCY, PWM\_RESOLUTION);  
  
  // Initialize boxcars for current sensing  
  for (int i = 0; i < bufferSizeIM; i++) {  
    bufferIM[i] = 0;  
  }  
  for (int i = 0; i < bufferSizeRP; i++) {  
    bufferRP[i] = 0;  
  }  
  for (int i = 0; i < bufferSizeT; i++) {  
    bufferT[i] = 0;  
  }  
  
  // For plotting current sensor  
  Serial.begin(115200);  
  
}  
  
// the loop function runs over and over again forever  
void loop() {  
  ledcWrite(PWM\_IM, PWM\_IM\_val);  
  ledcWrite(PWM\_RP, PWM\_RP\_val);  
  ledcWrite(PWM\_T, PWM\_T\_val);  
    
  switchState = digitalRead(SWITCHPIN);  
  
  if(switchState == 1) {  
    // move linear actuators to close hand  
    digitalWrite(AI1\_INDEX, HIGH);  
    digitalWrite(AI2\_INDEX, LOW);  
    digitalWrite(BI1\_MIDDLE, HIGH);  
    digitalWrite(BI2\_MIDDLE, LOW);  
    digitalWrite(AI1\_RING, HIGH);  
    digitalWrite(AI2\_RING, LOW);  
    digitalWrite(BI1\_PINKY, HIGH);  
    digitalWrite(BI2\_PINKY, LOW);  
    digitalWrite(AI1\_THUMB, HIGH);  
    digitalWrite(AI2\_THUMB, LOW);  
  
    // decrease power to linear actuator if current threshold reached  
    if(thresholdReachedIM){  
      PWM\_IM\_val = minPWM\_IM[strength];  
    }  
    if(thresholdReachedRP){  
      PWM\_RP\_val = minPWM\_RP[strength];  
    }  
    if(thresholdReachedT){  
      PWM\_T\_val = minPWM\_T[strength];  
    }  
  
  } // actuator will stop extending automatically if the limit is reached  
  else{  
    // move linear actuators to open hand  
    digitalWrite(AI1\_INDEX, LOW);  
    digitalWrite(AI2\_INDEX, HIGH);  
    digitalWrite(BI1\_MIDDLE, LOW);  
    digitalWrite(BI2\_MIDDLE, HIGH);  
    digitalWrite(AI1\_RING, LOW);  
    digitalWrite(AI2\_RING, HIGH);  
    digitalWrite(BI1\_PINKY, LOW);  
    digitalWrite(BI2\_PINKY, HIGH);  
    digitalWrite(AI1\_THUMB, LOW);  
    digitalWrite(AI2\_THUMB, HIGH);  
  
    // reset current threshold flag and power  
    thresholdReachedIM = 0;  
    thresholdReachedRP = 0;  
    thresholdReachedT = 0;  
    PWM\_IM\_val = maxPWM;  
    PWM\_RP\_val = maxPWM;  
    PWM\_T\_val = maxPWM;  
  
  } // actuator will stop extending automatically when reaching the limit  
  
  // Current sensor with boxcar average for index and middle finger  
  voltage1IM = analogRead(C1\_IM);  
  voltage2IM = analogRead(C2\_IM);  
  currentIM = voltage2IM - voltage1IM;  
  sumIM -= bufferIM[bufferIndexIM];  
  bufferIM[bufferIndexIM] = currentIM;  
  sumIM += bufferIM[bufferIndexIM];  
  bufferIndexIM = (bufferIndexIM + 1) % bufferSizeIM;  
  movingAverageIM = sumIM / bufferSizeIM;  
  if(movingAverageIM > currentThresholdIM[strength]){  
    thresholdReachedIM = 1;  
  }  
  
  // Current sensor with boxcar average for ring and pinky finger  
  voltage1RP = analogRead(C1\_RP);  
  voltage2RP = analogRead(C2\_RP);  
  currentRP = voltage2RP - voltage1RP;  
  sumRP -= bufferRP[bufferIndexRP];  
  bufferRP[bufferIndexRP] = currentRP;  
  sumRP += bufferRP[bufferIndexRP];  
  bufferIndexRP = (bufferIndexRP + 1) % bufferSizeRP;  
  movingAverageRP = sumRP / bufferSizeRP;  
  if(movingAverageRP > currentThresholdRP[strength]){  
    thresholdReachedRP = 1;  
  }  
  
  // Current sensor with boxcar average for thumb  
  voltage1T = analogRead(C1\_T);  
  voltage2T = analogRead(C2\_T);  
  currentT = voltage2T - voltage1T;  
  sumT -= bufferT[bufferIndexT];  
  bufferT[bufferIndexT] = currentT;  
  sumT += bufferT[bufferIndexT];  
  bufferIndexT = (bufferIndexT + 1) % bufferSizeT;  
  movingAverageT = sumT / bufferSizeT;  
  if(movingAverageT > currentThresholdT[strength]){  
    thresholdReachedT = 1;  
  }  
  
  Serial.print(thresholdReachedRP\*500);  
  Serial.print(",");  
  Serial.print(PWM\_RP\_val);  
  Serial.print(",");  
  Serial.print(movingAverageT\*10);  
  Serial.print(",");  
  Serial.println(switchState\*300);  
  
  delay(10);  
}

## Rayan\_V3\_fiveMotorsEMG

// Overall  
#define SWITCHINPUTPIN 18  
#define SWITCHMOTORSPIN 15  
#define BIOPILLPIN 35  
#define PWM\_RESOLUTION 8  
#define PWM\_FREQUENCY 4000  
int switchInputState;  
int switchMotorState;  
float RawEMG;  
const int EMG\_threshold = 80; // set for each person  
int motorPosition;  
const int maxPWM = 1000;  
int strength = 0; // 0=weak, 1=medium, 2=strong  
  
// EMG input from biopill boxcar intialization  
const int bufferSizeEMG = 100; // Size of the moving average buffer, adjust as needed  
float bufferEMG[bufferSizeEMG];  
int bufferIndexEMG = 0;  
float sumEMG = 0;  
float movingAverageEMG = 0;  
  
// Index and middle finger variables  
#define AI1\_INDEX 26  
#define AI2\_INDEX 25  
#define BI1\_MIDDLE 2  
#define BI2\_MIDDLE 0  
#define PWM\_IM 4  
#define C1\_IM 32  
#define C2\_IM 33  
  
const int bufferSizeIM = 10; // Size of the moving average buffer, adjust as needed  
float bufferIM[bufferSizeIM];  
int bufferIndexIM = 0;  
float sumIM = 0;  
float movingAverageIM = 0;  
  
int PWM\_IM\_val = 1000;  
float voltage1IM;  
float voltage2IM;  
float currentIM;  
bool thresholdReachedIM = 0;  
int currentThresholdIM[3] = {90, 130, 999}; //tailored to specific motor  
int minPWM\_IM[3] = {100, 500, maxPWM}; //tailored to specific motor  
  
// Ring and pinky finger variables  
#define AI1\_RING 14  
#define AI2\_RING 27  
#define BI1\_PINKY 21  
#define BI2\_PINKY 22  
#define PWM\_RP 17  
#define C1\_RP 38  
#define C2\_RP 39  
  
const int bufferSizeRP = 10; // Size of the moving average buffer, adjust as needed  
float bufferRP[bufferSizeRP];  
int bufferIndexRP = 0;  
float sumRP = 0;  
float movingAverageRP = 0;  
  
int PWM\_RP\_val = 1000;  
float voltage1RP;  
float voltage2RP;  
float currentRP;  
bool thresholdReachedRP = 0;  
int currentThresholdRP[3] = {20, 60, 999}; //tailored to specific motor  
int minPWM\_RP[3] = {100, 200, maxPWM}; //tailored to specific motor  
  
// Thumb variables  
#define AI1\_THUMB 23  
#define AI2\_THUMB 19  
#define PWM\_T 16  
#define C1\_T 36  
#define C2\_T 37  
  
const int bufferSizeT = 10; // Size of the moving average buffer, adjust as needed  
float bufferT[bufferSizeT];  
int bufferIndexT = 0;  
float sumT = 0;  
float movingAverageT = 0;  
  
int PWM\_T\_val = 1000;  
float voltage1T;  
float voltage2T;  
float currentT;  
bool thresholdReachedT = 0;  
int currentThresholdT[3] = {100, 160, 999}; //tailored to specific motor  
int minPWM\_T[3] = {100, 700, maxPWM}; //tailored to specific motor  
  
//--------------------------------------------------------------------------------  
void setup() {  
  // initialize digital pins as outputs  
  pinMode(SWITCHINPUTPIN, INPUT);  
  pinMode(SWITCHMOTORSPIN, INPUT);  
  pinMode(BIOPILLPIN, INPUT);  
  
  pinMode(PWM\_IM, OUTPUT);  
  pinMode(PWM\_RP, OUTPUT);  
  pinMode(PWM\_T, OUTPUT);  
  
  pinMode(AI1\_INDEX, OUTPUT);  
  pinMode(AI2\_INDEX, OUTPUT);  
  pinMode(BI1\_MIDDLE, OUTPUT);  
  pinMode(BI2\_MIDDLE, OUTPUT);  
  pinMode(AI1\_RING, OUTPUT);  
  pinMode(AI2\_RING, OUTPUT);  
  pinMode(BI1\_PINKY, OUTPUT);  
  pinMode(BI2\_PINKY, OUTPUT);  
  pinMode(AI1\_THUMB, OUTPUT);  
  pinMode(AI2\_THUMB, OUTPUT);  
  
  pinMode(C1\_IM, INPUT);  
  pinMode(C2\_IM, INPUT);  
  pinMode(C1\_RP, INPUT);  
  pinMode(C2\_RP, INPUT);  
  pinMode(C1\_T, INPUT);  
  pinMode(C2\_T, INPUT);  
  
  // Setup PWMs  
  ledcAttach(PWM\_IM, PWM\_FREQUENCY, PWM\_RESOLUTION);  
  ledcAttach(PWM\_RP, PWM\_FREQUENCY, PWM\_RESOLUTION);  
  ledcAttach(PWM\_T, PWM\_FREQUENCY, PWM\_RESOLUTION);  
  
  // Initialize boxcars for current sensing  
  for (int i = 0; i < bufferSizeIM; i++) {  
    bufferIM[i] = 0;  
  }  
  for (int i = 0; i < bufferSizeRP; i++) {  
    bufferRP[i] = 0;  
  }  
  for (int i = 0; i < bufferSizeT; i++) {  
    bufferT[i] = 0;  
  }  
  
  // For plotting current sensor  
  Serial.begin(115200);  
  
}  
  
// the loop function runs over and over again forever  
void loop() {  
  ledcWrite(PWM\_IM, PWM\_IM\_val);  
  ledcWrite(PWM\_RP, PWM\_RP\_val);  
  ledcWrite(PWM\_T, PWM\_T\_val);  
    
  if(digitalRead(SWITCHINPUTPIN)){ //EMG input  
    RawEMG = analogRead(BIOPILLPIN);  
      
    //Boxcar filter for EMG signal  
    float filtered\_EMG = abs(EMGFilter(RawEMG));  
    //updating boxcar array  
    sumEMG -= bufferEMG[bufferIndexEMG];  
    sumEMG += filtered\_EMG;  
    bufferEMG[bufferIndexEMG] = filtered\_EMG;  
    bufferIndexEMG += 1;  
    if (bufferIndexEMG >= bufferSizeEMG)  
      bufferIndexEMG = 0;  
    movingAverageEMG = sumEMG / bufferSizeEMG;  
  
    if(movingAverageEMG > EMG\_threshold){  
      motorPosition = 1;  
    }  
    else{  
      motorPosition = 0;  
    }  
  
    Serial.println(movingAverageEMG);  
  }  
  else{ //Switch input  
    switchMotorState = digitalRead(SWITCHMOTORSPIN);  
    motorPosition = switchMotorState;  
  }  
  
  if(motorPosition == 1) {  
    // move linear actuators to close hand  
    digitalWrite(AI1\_INDEX, HIGH);  
    digitalWrite(AI2\_INDEX, LOW);  
    digitalWrite(BI1\_MIDDLE, HIGH);  
    digitalWrite(BI2\_MIDDLE, LOW);  
    digitalWrite(AI1\_RING, HIGH);  
    digitalWrite(AI2\_RING, LOW);  
    digitalWrite(BI1\_PINKY, HIGH);  
    digitalWrite(BI2\_PINKY, LOW);  
    digitalWrite(AI1\_THUMB, HIGH);  
    digitalWrite(AI2\_THUMB, LOW);  
  
    // decrease power to linear actuator if current threshold reached  
    if(thresholdReachedIM){  
      PWM\_IM\_val = minPWM\_IM[strength];  
    }  
    if(thresholdReachedRP){  
      PWM\_RP\_val = minPWM\_RP[strength];  
    }  
    if(thresholdReachedT){  
      PWM\_T\_val = minPWM\_T[strength];  
    }  
  
  } // actuator will stop extending automatically if the limit is reached  
  else{  
    // move linear actuators to open hand  
    digitalWrite(AI1\_INDEX, LOW);  
    digitalWrite(AI2\_INDEX, HIGH);  
    digitalWrite(BI1\_MIDDLE, LOW);  
    digitalWrite(BI2\_MIDDLE, HIGH);  
    digitalWrite(AI1\_RING, LOW);  
    digitalWrite(AI2\_RING, HIGH);  
    digitalWrite(BI1\_PINKY, LOW);  
    digitalWrite(BI2\_PINKY, HIGH);  
    digitalWrite(AI1\_THUMB, LOW);  
    digitalWrite(AI2\_THUMB, HIGH);  
  
    // reset current threshold flag and power  
    thresholdReachedIM = 0;  
    thresholdReachedRP = 0;  
    thresholdReachedT = 0;  
    PWM\_IM\_val = maxPWM;  
    PWM\_RP\_val = maxPWM;  
    PWM\_T\_val = maxPWM;  
  
  } // actuator will stop extending automatically when reaching the limit  
  
  // Current sensor with boxcar average for index and middle finger  
  voltage1IM = analogRead(C1\_IM);  
  voltage2IM = analogRead(C2\_IM);  
  currentIM = voltage2IM - voltage1IM;  
  sumIM -= bufferIM[bufferIndexIM];  
  bufferIM[bufferIndexIM] = currentIM;  
  sumIM += bufferIM[bufferIndexIM];  
  bufferIndexIM = (bufferIndexIM + 1) % bufferSizeIM;  
  movingAverageIM = sumIM / bufferSizeIM;  
  if(movingAverageIM > currentThresholdIM[strength]){  
    thresholdReachedIM = 1;  
  }  
  
  // Current sensor with boxcar average for ring and pinky finger  
  voltage1RP = analogRead(C1\_RP);  
  voltage2RP = analogRead(C2\_RP);  
  currentRP = voltage2RP - voltage1RP;  
  sumRP -= bufferRP[bufferIndexRP];  
  bufferRP[bufferIndexRP] = currentRP;  
  sumRP += bufferRP[bufferIndexRP];  
  bufferIndexRP = (bufferIndexRP + 1) % bufferSizeRP;  
  movingAverageRP = sumRP / bufferSizeRP;  
  if(movingAverageRP > currentThresholdRP[strength]){  
    thresholdReachedRP = 1;  
  }  
  
  // Current sensor with boxcar average for thumb  
  voltage1T = analogRead(C1\_T);  
  voltage2T = analogRead(C2\_T);  
  currentT = voltage2T - voltage1T;  
  sumT -= bufferT[bufferIndexT];  
  bufferT[bufferIndexT] = currentT;  
  sumT += bufferT[bufferIndexT];  
  bufferIndexT = (bufferIndexT + 1) % bufferSizeT;  
  movingAverageT = sumT / bufferSizeT;  
  if(movingAverageT > currentThresholdT[strength]){  
    thresholdReachedT = 1;  
  }  
    
  delay(10);  
}  
  
float EMGFilter(float input)  
{  
  float output = input;  
  {  
    static float z1, z2; // filter section state  
    float x = output - 0.05159732 \* z1 - 0.36347401 \* z2;  
    output = 0.01856301 \* x + 0.03712602 \* z1 + 0.01856301 \* z2;  
    z2 = z1;  
    z1 = x;  
  }  
  {  
    static float z1, z2; // filter section state  
    float x = output - -0.53945795 \* z1 - 0.39764934 \* z2;  
    output = 1.00000000 \* x + -2.00000000 \* z1 + 1.00000000 \* z2;  
    z2 = z1;  
    z1 = x;  
  }  
  {  
    static float z1, z2; // filter section state  
    float x = output - 0.47319594 \* z1 - 0.70744137 \* z2;  
    output = 1.00000000 \* x + 2.00000000 \* z1 + 1.00000000 \* z2;  
    z2 = z1;  
    z1 = x;  
  }  
  {  
    static float z1, z2; // filter section state  
    float x = output - -1.00211112 \* z1 - 0.74520226 \* z2;  
    output = 1.00000000 \* x + -2.00000000 \* z1 + 1.00000000 \* z2;  
    z2 = z1;  
    z1 = x;  
  }  
  return output;  
}