

```

1  #include <Adafruit_PWMServoDriver.h>
2  #include <LiquidCrystal.h>
3
4  Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
5  //String lastReceivedMessage = "";
6  bool handShakeSuccessful = false;
7  int received;
8  int startStage = 0;
9  bool started = false; //once started == true then start swapping from the serial reads
10 bool CommandInProgress = false;
11 String inputString = "";
12 bool stringComplete = false;
13 byte insertNumber = "";
14
15 const int AddOnType_MMU = 0;
16 const int AddOnType_BlendBox = 1;
17 int AddOnType = AddOnType_MMU;
18
19 const int servo_pwm_max = 2900; // 2376;
20 const int servo_pwm_min = 600; //484;
21
22
23 //servos enums
24 const int eePinNum = 0;
25 const int eeMaxAngle = 1;
26 const int eeCurrentAngle = 2;
27
28 //Servos
29 const int numOfServos = 8;
30 const int s_Tool_Rotate = 0; //360d //TR
31 const int s_Tool_Height = 1; //TH
32 const uint8_t s_Tool_Lock = 2; //TL
33 const uint8_t s_QuickSwapHotend_Lock = 3; //QL
34 const uint8_t s_ToolHolder_Rotate = 4; //360d //HR
35 const uint8_t s_Cutter_Rotate = 5; //CR
36 const uint8_t s_Cutter_Action = 6; //CA
37 const uint8_t s_WasteCup_Action = 7; //WA
38
39 int servos[numOfServos][3]; //pin #, max angle, current angle
40 String servos_names[numOfServos] = {"Tool_Rotate", "Tool_UpDown", "Tool_LockUnlock",
  "Cutter_Rotate", "Cutter_CutOpen", "WasteCup_DumpFill", "ToolHolder_Rotate",
  "ExtruderHotend_LockUnlock"};
41
42 //LCD
43 LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
44 unsigned long tepTimer = 0;
45 bool setupComplete = false;
46 int buttonPress; // variable to store the value coming
  from the analog pin
47 int potValue = 0;
48 int potMinValue = 600; //15
49 int potMaxValue = 1000;
50 int potRange = 400; //1008
51 int inServoTuning = false;
52 int potCentered = false;
53 int toolDegrees = 0;
54 int pulselength = 0;
55 int currentPotPosition = 0;
56 int currentAngle = 0;
57 int moveToAngle = 0;
58 bool servoSetModeEnabled = false;
59 bool buttonPressed = false;
60 float anglePercentOfMax = 0.0;
61 int potMatchValue = 0;
62 int maxAngle = 180.0; //360.0
63 int currentServoEditing = 0;
64 bool loopStarted = false;
65
66 //swap tools process

```

```

67  const int eeps_ServoNumber = 0;
68  const int eeps_Degrees = 1;
69  const int eeps_MsDelayPerDegreeMoved = 2;
70  const int eeps_MsDelayAfterCommandSent = 3;
71  const int eeps_StepType = 4; //eeps_ButtonCheck = 4;
72  const int eeps_IsIncludedInMMU = 5;
73
74  const int eeRegularStep = 0; //eeButtonCheck_No = 0;
75  const int eeButtonCheck_Empty = 1;
76  const int eeButtonCheck_HoldingTool = 2;
77  const int eeExtrude = 3;
78  const int eeRetract = 4;
79  const int eeToolHolderPrepRotate = 5;
80  const int eeAddHalfDegreePrecision = 6;
81  const int eeToolHolderPrepUNrotate = 7;
82
83  const int eeToolHolderPrepRotate_Degrees = 3;
84  const int eeToolHolderPrepUNrotate_Degrees = 3; //4; //1; //2; //3; //3; //1; //3; //6;
    //3;
85
86
87  const int numOfProcessSteps_LoadTool = 21; //***change this if adding or removing
    process steps
88  const int numOfProcessSteps_UnloadTool = 30; //***change this if adding or removing
    process steps
89  bool executeNextProcessStep = false;
90  int currentStepOfProcess = 0;
91  int ProcessSteps_LoadTool[numOfProcessSteps_LoadTool][5]; //servo number, degrees,
    msDelayPerDegree, msDelayAfterCommandSent, buttonCheck
92  int ProcessSteps_UnloadTool[numOfProcessSteps_UnloadTool][6]; //servo number, degrees,
    msDelayPerDegree, msDelayAfterCommandSent, buttonCheck, IsIncludedInMMU
93  int ps_currentServo = 0;
94  int ps_targetAngle = 0;
95  int ps_msDelayPerDegreeMoved = 0;
96  int ps_msDelayAfterCommandSent = 0;
97  int numberOfStepsToProcess = 0;
98  int msDelayAfterCommandSent_Buffer = 100; //50 //in milliseconds //extra ms delay
99
100
101
102
103
104
105  //menu
106  //menus enums
107  const int eeOnce = 0;
108  const int eeEndless = 1;
109  const int eeLoadTool = 0;
110  const int eeUnloadTool = 1;
111  const int eeAutomaticProcess = 0;
112  const int eeManualProcess = 1;
113
114  const int mm_menu_OnceEndless = 0;
115  const int mm_menu_LoadUnload = 1;
116  const int mm_menu_AutoManual = 2;
117  const int mm_menu_Process = 3;
118
119  bool automaticExecuteProcessSteps = false;
120  int menu_currentLevel = 0;
121  int menu_OnceEndless_position = 0; //for the Once/Endless menu
122  int menu_LoadUnload_position = 0; //for the load/unload menu
123  int menu_AutoManual_position = 0; //for the auto/manual menu
124  const int menu_OnceEndless_TotalPositions = 2;
125  const int menu_LoadUnload_TotalPositions = 2;
126  const int menu_AutoManual_TotatlPositions = 2;
127  String menu_OnceEndless_Name[menu_OnceEndless_TotalPositions] = {"Once", "Endless"};
128  String menu_LoadUnload_Names[menu_LoadUnload_TotalPositions] = {"Load", "Unload"};
129  String menu_AutoManual_Names[menu_AutoManual_TotatlPositions] = {"Auto", "Manual"};
130  bool Selected_LoadUnload = false;

```

```

131 bool Selected_Load = false;
132 bool Selected_Unload = false;
133
134
135 //error state button press
136 const bool ErrorCheckingEnabled = false;
137 const int CheckButton_Pin = 3; //0; //digital pin zero(0)
138 const int eeePauseAtRotation = 95;
139 bool InErrorState = false;
140 int CurrentProcessType = eeLoadTool;//load/unload
141 int ErrorOnProcessStep = 0;
142
143 //tool holder rotation and selection
144 bool firstPositionCommandGiven = false;
145 const int servoMinAngle = 0;
146 float pos_Tool_Holder_FirstTool = 15; //11.7; //11; //12; //2;
147 bool toolIsLoaded = false;
148 int CurrentTool = 0;
149 int eeth_maxTool = 24;
150
151 int pos_Tool_Lock_Locked;
152 int pos_Cutter_Rotate_Stowed;
153 bool LockToolPartWayThru = false;
154 int numMsUntilLock = 50; //100; //200; //10ms per degree currently
155
156 void printWithParity(String message) {
157     Serial.println(message + String(checkParity(message)));
158 }
159
160 void updateLCD(String message1, String message2) {
161     lcd.clear();
162     lcd.setCursor(0, 0);
163     lcd.print(message1);
164     lcd.setCursor(0, 1);
165     lcd.print(message2);
166 }
167
168 void updateLCD_line1(String message) {
169     lcd.clear();
170     lcd.setCursor(0, 0);
171     lcd.print(message);
172 }
173
174 void updateLCD_line2(String message) {
175     lcd.clear();
176     lcd.setCursor(0, 1);
177     lcd.print(message);
178 }
179
180
181 //**** Holder Rotate (HR) ****
182 //Servo 4
183 int Adjustment_HolderRotate = 0;
184
185 void ToolHolder_AlignToThisTool(int SelectThisTool)
186 {
187     int localPulseLength = 0;
188     int msDelayPerToolPostionToCompleteMovement = 50;
189     int msDelayPadding = 50;
190     int msDelayUntilRotationComplete = 0; //total ms to delay for the current tool holder
rotation
191 float degreesPerTool = 14.4; //14.72; //computed angle is 14.4d per spline calced as 25T
splines 360/25=14.4 //14.72; //last 14.80 //this works for 25T's but barely: 14.85;
//19:14.95 slightly too much;
192 float degreesPositionOfSelectedTool = (float)SelectThisTool * degreesPerTool;
193
194 //apply adjustment from EEPROM
195     pos_Tool_Holder_FirstTool = pos_Tool_Holder_FirstTool + Adjustment_HolderRotate;
196

```

```

197     servos[s_ToolHolder_Rotate][eeCurrentAngle] = degreesPositionOfSelectedTool +
pos_Tool_Holder_FirstTool;
198
199     localPulseLength = fMap(degreesPositionOfSelectedTool + pos_Tool_Holder_FirstTool,
servoMinAngle, servos[s_ToolHolder_Rotate][eeMaxAngle], servo_pwm_min, servo_pwm_max
);
200
201     pwm.setPWM(servos[s_ToolHolder_Rotate][eePinNum], 0, localPulseLength);
202
203     msDelayUntilRotationComplete = abs(CurrentTool - SelectThisTool) *
msDelayPerToolPostionToCompleteMovement + msDelayPadding;
204
205     delay(msDelayUntilRotationComplete);
206
207     CurrentTool = SelectThisTool;
208 }
209
210 int fMap(float desiredAngle, int MinAngle, int MaxAngle, int minPWM, int maxPWM)
211 {
212     int angleRange = MaxAngle - MinAngle;
213     int pwmRange = maxPWM - minPWM;
214
215     float desiredAnglePercentOfRange = desiredAngle / angleRange;
216     float pwmByDesiredAngle = float(pwmRange) * float(desiredAnglePercentOfRange) + float(
minPWM);
217
218     return pwmByDesiredAngle;
219 }
220
221 void setup()
222 {
223     //position variables
224     //Servo 0
225     //**** Tool Rotate (TR) ****
226     int Adjustment_Tool_Rotate = 0; //1; //-1; //0; //1; //2; //changed out servo all
should be off by same angle over last servo
227     //next line is starting first 1st position
228     int pos_Tool_Rotate_ButtingTheToolToTheLeftOfNext = 104 + Adjustment_Tool_Rotate;
//103 //102; //104; //103;
229     int pos_Tool_Rotate_LeftOfToolInHolder = 98 + Adjustment_Tool_Rotate; //97 //98;
//99; //101; //101 to try and deal with the single nozzle load failure //100; //102;
//101; //103; //95; //SetupMode
230     int pos_Tool_Rotate_UnderToolHolder_ConnectWithNozzleCollar = 96 +
Adjustment_Tool_Rotate; //95 //80, 85; //90; //83; //76 = 11
231     int pos_Tool_Rotate_UnderToolHolder_CenteredUnderCurrentTool = 98 +
Adjustment_Tool_Rotate; //97 //98, 97; //95;
232     int pos_Tool_Rotate_UnderToolHolder_ConnectWithNozzleCollar_NoPressure =
pos_Tool_Rotate_UnderToolHolder_CenteredUnderCurrentTool; //95; //97; //98; //96;
//91; //86 = 5
233     int pos_Tool_Rotate_ReleaseFromHotendUnderToolHolder = 109 + Adjustment_Tool_Rotate;
//108 //106 //104;
234     int pos_Tool_Rotate_BetweenBothNozzles =
pos_Tool_Rotate_ReleaseFromHotendUnderToolHolder - 7 + Adjustment_Tool_Rotate;
235     int pos_Tool_Rotate_ButtonToolCheck = 75 + Adjustment_Tool_Rotate; //74, 72, 75, 70
//72; //74; //75; //68;
236     int pos_Tool_Rotate_UnderExtruder_JerkConnectWithNozzleCollar = 281 +
Adjustment_Tool_Rotate; //280 //275, 282; //283; //284; //265; //270; //274;
237     int pos_Tool_Rotate_UnderExtruder_ConnectWithNozzleCollar = 286 +
Adjustment_Tool_Rotate; //285 //284; 286; //Why did this change????!??? 285; //283;
//284; //283; // 282; //283; //284; //285; //283; //287; //285; //278;
238     int pos_Tool_Rotate_UnderExtruder_JerkReleaseFromNozzleCollar = 293 +
Adjustment_Tool_Rotate; //292 //293 291; // 310; //305; //297;
239     int pos_Tool_Rotate_UnderExtruder_ReleasedFromNozzleCollar = 291 +
Adjustment_Tool_Rotate; //290 //291, 293 291; //285;
240     int pos_Tool_Rotate_WaitingForUnloadCommand = 148 + Adjustment_Tool_Rotate; //147
//140;
241     int pos_Tool_Rotate_PastWasteCup = 259 + Adjustment_Tool_Rotate; //258 //251;
242
243     //**** Tool Height (TH) ****

```

```

244 //Servo 1
245 int Adjustment_Tool_Height = 0; //4; //14; //1 //-1; increasing this will lower all
up/down moves. decreasing this will raise all up/down moves
246 //next line is starting first 1st position
247 int pos_Tool_Height_LowestLevel = 129 + Adjustment_Tool_Height; //125 //121 //Servo
change 126; // below 126 it causes the servo to stall. 127; //126; //119; //117
248 int pos_Tool_Height_ButtingTheToolToTheLeftOfNext = 44 + Adjustment_Tool_Height;
//40 //37, Servo change 42; //we want to be at the height of the hex on the right
nozzle //39; //40; //41;
249 int pos_Tool_Height_NozzleCollarLevel = 37 + Adjustment_Tool_Height; //33 //30,
Servo change 35; //38; //36; //29;
250 int pos_Tool_Height_ToolLoweredButStillInHolder = 59 + Adjustment_Tool_Height; //54
//Servo change 49; //42;
251 int pos_Tool_Height_ToolFullyInsertedInHolder = 31 + Adjustment_Tool_Height; //27
//25, 22, Servo change 27; //29; //20; //13;
252 int pos_Tool_Height_ToolFullyInsertedInHolder_NoPressure = 38 +
Adjustment_Tool_Height; //34 //35, 32, Servo change 37; //36; //40; //29; //SetupMode
253 int pos_Tool_Height_ToolLoweredButStillInExtruder = 52 + Adjustment_Tool_Height;
//48 //Servo change 53; //53 moves up until the hotend taper is past the edge of
the inner bore of the heater block //56; //49;
254 int pos_Tool_Height_ToolFullyInsertedIntoExtruder = 41 + Adjustment_Tool_Height;
//37 //39, 40, 39, 33, Servo change 38; //40; //42; //35; //28;
255 int pos_Tool_Height_ToolFullyInsertedIntoExtruder_ScrappingHotendMildPressure = 44 +
Adjustment_Tool_Height; //40 //Servo change 43; //42; //41; //42; //40; //42;
//clunking? maybe 41?
256 int pos_Tool_Height_ToolFullyInsertedIntoExtruder_NoPressure = 42 +
Adjustment_Tool_Height; //38 //39; 38; //Servo change 43; //44; //42; // 45; // 42;
//35;
257 int pos_Tool_Height_ToolLowered_CuttingHeight = 122 + Adjustment_Tool_Height; //118
//117 //125 //113; //111, 108, 109; 107; //higher number moves down away from cutter
leaving longer filament strand sticking out. //108; //Servo change 112; //works with
new cutting sequence //110; //108; at 108 there was a single instance of cutting the
heatbreak copper and it was ruined. //107; //99;
258 int pos_Tool_Height_ToolLowered_BelowCutterJaws = 116 + Adjustment_Tool_Height;
//112 //Servo change 117; //110;
259
260 //**** Tool Lock (TL) (micro 280d servo) ****
261 //Servo 2
262 int Adjustment_Tool_Lock = 0;
263 //next line is starting first 1st position
264 int pos_Tool_Lock_Unlocked = 195 + Adjustment_Tool_Lock; //180
265 pos_Tool_Lock_Locked = 112 + Adjustment_Tool_Lock; //8 //7; //8; //standard
move172degrees. 8; //now precision move +.5 //9; //8; //9; //8; //9; 13;
266
267 //**** QuickSwap- Hotend Lock (QL) ****
268 //Servo 3
269 int Adjustment_QuickSwapHotend_Lock = 0;
270 //next line is 1st starting first 1st position
271 int pos_QuickSwapHotend_Lock_Locked = 70 + Adjustment_QuickSwapHotend_Lock; //0
272 int pos_QuickSwapHotend_Lock_Unlocked = 104 + Adjustment_QuickSwapHotend_Lock; //34
//32; //33; //34; //35; //29;
273
274 //**** Cutter Rotate (CR) ****
275 //Servo 5
276 int Adjustment_Cutter_Rotate = 0;
277 //next line is starting first 1st position
278 pos_Cutter_Rotate_Stowed = 27 + Adjustment_Cutter_Rotate; //25; //1; //must be greater
than 0. 0 causes major jittering. Something about 25 works better than 26. 26 had
lots of jitter.
279 int pos_Cutter_Rotate_Cutting = 123 + Adjustment_Cutter_Rotate; //122; //125; //126,
124, 122; //121; //122; //121; //122; //120; //124; //126; //127; //128; //get closer
but lower the cutting height a little //126; //127; 127 is too close //126; //or
maybe 127? //121; //99;
280
281 //**** Cutter Action (CA) ****
282 //Servo 6
283 int Adjustment_Cutter_Action = 0;
284 //next line is starting first 1st position
285 int pos_Cutter_Action_Open = 175 + Adjustment_Cutter_Action; //160

```

```

286     int pos_Cutter_Action_Cut = 15 + Adjustment_Cutter_Action; //0 //20; //40; //6; //7;
287     //21;
288     //**** Waste Cup Action (WA) (micro 280d servo) ****
289     //Servo 7
290     int Adjustment_WasteCup_Action = 0;
291     //next line is starting first 1st position
292     int pos_WasteCup_Action_Fill = 110 + Adjustment_WasteCup_Action; //0
293     int pos_WasteCup_Action_Dump = 99 + Adjustment_WasteCup_Action; //107
294
295
296
297
298     Serial.begin(9600);
299     // Serial.println("Swapper3D Start!");
300
301
302     //this ensures that the serial buffer is empty
303     //so that no aberant tool changes are performed
304     ClearSerialBuffer();
305
306
307     lcd.begin(16, 2);
308     lcd.setCursor(0,0);
309     lcd.print("Ready to Swap!");
310     lcd.setCursor(0,1);
311     lcd.print("Empty");
312
313     pwm.begin();
314     pwm.setOscillatorFrequency(27000000);
315     pwm.setPWMFreq(300); // Digital servos run at 300Hz updates
316
317
318     //initialize the pin for the end effector insert full/empty checks
319     // pinMode(CheckButton_Pin, INPUT_PULLUP);
320     pinMode(A3, INPUT_PULLUP);
321
322     delay(10);
323
324
325     //pin #, max angle, start angle, current angle
326     servos[s_Tool_Rotate][eePinNum] = 15;
327     servos[s_Tool_Rotate][eeMaxAngle] = 360;
328     servos[s_Tool_Rotate][eeCurrentAngle] = pos_Tool_Rotate_ButtingTheToolToTheLeftOfNext
329     ;
330     servos[s_Tool_Height][eePinNum] = 14; //using this pwm servo port on the servo
331     shield causes random bytes on the serial lines
332     servos[s_Tool_Height][eeMaxAngle] = 180;
333     servos[s_Tool_Height][eeCurrentAngle] = pos_Tool_Height_LowestLevel;
334     servos[s_Tool_Lock][eePinNum] = 13;
335     servos[s_Tool_Lock][eeMaxAngle] = 280; //micro
336     servos[s_Tool_Lock][eeCurrentAngle] = pos_Tool_Lock_Unlocked;
337     servos[s_QuickSwapHotend_Lock][eePinNum] = 12; //s_QuickSwapHotend_Lock
338     servos[s_QuickSwapHotend_Lock][eeMaxAngle] = 180; //
339     servos[s_QuickSwapHotend_Lock][eeCurrentAngle] = pos_QuickSwapHotend_Lock_Locked; //
340     servos[s_ToolHolder_Rotate][eePinNum] = 11; //s_ToolHolder_Rotate
341     servos[s_ToolHolder_Rotate][eeMaxAngle] = 360; //
342     servos[s_ToolHolder_Rotate][eeCurrentAngle] = pos_Tool_Holder_FirstTool; //
343     servos[s_Cutter_Rotate][eePinNum] = 10; //s_Cutter_Rotate
344     servos[s_Cutter_Rotate][eeMaxAngle] = 180; //
345     servos[s_Cutter_Rotate][eeCurrentAngle] = pos_Cutter_Rotate_Stowed; //
346     servos[s_Cutter_Action][eePinNum] = 9; //s_Cutter_Action
347     servos[s_Cutter_Action][eeMaxAngle] = 180; //
348     servos[s_Cutter_Action][eeCurrentAngle] = pos_Cutter_Action_Open; //
349     servos[s_WasteCup_Action][eePinNum] = 8; //s_WasteCup_Action
350     servos[s_WasteCup_Action][eeMaxAngle] = 280; //micro
351     servos[s_WasteCup_Action][eeCurrentAngle] = pos_WasteCup_Action_Fill; //

```

```

352
353
354 for(int i; i < 8; i++)
355 {
356     pulselength = map(servos[i][eeCurrentAngle], 0, servos[i][eeMaxAngle],
357                       servo_pwm_min, servo_pwm_max);
358     pwm.setPWM(servos[i][eePinNum], 0, pulselength);
359     delay(100);
360 }
361
362 //align to the first tool
363 ToolHolder_AlignToThisTool(0);
364
365 //store process steps
366 ProcessSteps_LoadTool[0][eeps_ServoNumber] = s_Tool_Height;
367 ProcessSteps_LoadTool[1][eeps_ServoNumber] = s_Tool_Rotate;
368 ProcessSteps_LoadTool[2][eeps_ServoNumber] = s_Tool_Height;
369 ProcessSteps_LoadTool[3][eeps_ServoNumber] = s_Tool_Rotate;
370 ProcessSteps_LoadTool[4][eeps_ServoNumber] = s_Tool_Rotate;
371 ProcessSteps_LoadTool[5][eeps_ServoNumber] = s_Tool_Height;
372 ProcessSteps_LoadTool[6][eeps_ServoNumber] = s_Tool_Lock;
373 ProcessSteps_LoadTool[7][eeps_ServoNumber] = s_Tool_Height;
374 ProcessSteps_LoadTool[8][eeps_ServoNumber] = s_Tool_Rotate;
375 ProcessSteps_LoadTool[9][eeps_ServoNumber] = s_Tool_Rotate;
376 ProcessSteps_LoadTool[10][eeps_ServoNumber] = s_QuickSwapHotend_Lock;
377 ProcessSteps_LoadTool[11][eeps_ServoNumber] = s_Tool_Height;
378 ProcessSteps_LoadTool[12][eeps_ServoNumber] = s_Tool_Lock;
379 ProcessSteps_LoadTool[13][eeps_ServoNumber] = s_Tool_Height;
380 ProcessSteps_LoadTool[14][eeps_ServoNumber] = s_QuickSwapHotend_Lock;
381 ProcessSteps_LoadTool[15][eeps_ServoNumber] = s_Tool_Height;
382 ProcessSteps_LoadTool[16][eeps_ServoNumber] = s_Tool_Rotate;
383 ProcessSteps_LoadTool[17][eeps_ServoNumber] = s_Tool_Rotate;
384 ProcessSteps_LoadTool[18][eeps_ServoNumber] = s_Tool_Height;
385 ProcessSteps_LoadTool[19][eeps_ServoNumber] = s_Tool_Rotate;
386 ProcessSteps_LoadTool[20][eeps_ServoNumber] = s_Tool_Rotate;
387
388 ProcessSteps_LoadTool[0][eeps_Degrees] =
389 pos_Tool_Height_ButtingTheToolToTheLeftOfNext;
390 ProcessSteps_LoadTool[1][eeps_Degrees] = pos_Tool_Rotate_LeftOfToolInHolder;
391 ProcessSteps_LoadTool[2][eeps_Degrees] = pos_Tool_Height_NozzleCollarLevel;
392 ProcessSteps_LoadTool[3][eeps_Degrees] =
393 pos_Tool_Rotate_UnderToolHolder_ConnectWithNozzleCollar;
394 ProcessSteps_LoadTool[4][eeps_Degrees] =
395 pos_Tool_Rotate_UnderToolHolder_ConnectWithNozzleCollar_NoPressure;
396 ProcessSteps_LoadTool[5][eeps_Degrees] = pos_Tool_Height_ToolLoweredButStillInHolder;
397 ProcessSteps_LoadTool[6][eeps_Degrees] = pos_Tool_Lock_Locked;
398 ProcessSteps_LoadTool[7][eeps_Degrees] = pos_Tool_Height_LowestLevel;
399 ProcessSteps_LoadTool[8][eeps_Degrees] = pos_Tool_Rotate_ButtonToolCheck;
400 ProcessSteps_LoadTool[9][eeps_Degrees] =
401 pos_Tool_Rotate_UnderExtruder_ConnectWithNozzleCollar;
402 ProcessSteps_LoadTool[10][eeps_Degrees] = pos_QuickSwapHotend_Lock_Unlocked;
403 ProcessSteps_LoadTool[11][eeps_Degrees] =
404 pos_Tool_Height_ToolLoweredButStillInExtruder;
405 ProcessSteps_LoadTool[12][eeps_Degrees] = pos_Tool_Lock_Unlocked;
406 ProcessSteps_LoadTool[13][eeps_Degrees] =
407 pos_Tool_Height_ToolFullyInsertedIntoExtruder;
408 ProcessSteps_LoadTool[14][eeps_Degrees] = pos_QuickSwapHotend_Lock_Locked;
409 ProcessSteps_LoadTool[15][eeps_Degrees] =
410 pos_Tool_Height_ToolFullyInsertedIntoExtruder_NoPressure;
411 ProcessSteps_LoadTool[16][eeps_Degrees] =
412 pos_Tool_Rotate_UnderExtruder_JerkReleaseFromNozzleCollar;
413 ProcessSteps_LoadTool[17][eeps_Degrees] =
414 pos_Tool_Rotate_UnderExtruder_ReleasedFromNozzleCollar;
415 ProcessSteps_LoadTool[18][eeps_Degrees] = pos_Tool_Height_LowestLevel;
416 ProcessSteps_LoadTool[19][eeps_Degrees] = pos_Tool_Rotate_ButtonToolCheck;
417 ProcessSteps_LoadTool[20][eeps_Degrees] = pos_Tool_Rotate_WaitingForUnloadCommand;

```



```

411 ProcessSteps_LoadTool[0][eeps_MsDelayPerDegreeMoved] = 0;
412 ProcessSteps_LoadTool[1][eeps_MsDelayPerDegreeMoved] = 0;
413 ProcessSteps_LoadTool[2][eeps_MsDelayPerDegreeMoved] = 0;
414 ProcessSteps_LoadTool[3][eeps_MsDelayPerDegreeMoved] = 0;
415 ProcessSteps_LoadTool[4][eeps_MsDelayPerDegreeMoved] = 0;
416 ProcessSteps_LoadTool[5][eeps_MsDelayPerDegreeMoved] = 0;
417 ProcessSteps_LoadTool[6][eeps_MsDelayPerDegreeMoved] = 0;
418 ProcessSteps_LoadTool[7][eeps_MsDelayPerDegreeMoved] = 0;
419 ProcessSteps_LoadTool[8][eeps_MsDelayPerDegreeMoved] = 0;
420 ProcessSteps_LoadTool[9][eeps_MsDelayPerDegreeMoved] = 0;
421 ProcessSteps_LoadTool[10][eeps_MsDelayPerDegreeMoved] = 0;
422 ProcessSteps_LoadTool[11][eeps_MsDelayPerDegreeMoved] = 20; //60;//6;
423 ProcessSteps_LoadTool[12][eeps_MsDelayPerDegreeMoved] = 0;
424 ProcessSteps_LoadTool[13][eeps_MsDelayPerDegreeMoved] = 0;
425 ProcessSteps_LoadTool[14][eeps_MsDelayPerDegreeMoved] = 0;
426 ProcessSteps_LoadTool[15][eeps_MsDelayPerDegreeMoved] = 0;
427 ProcessSteps_LoadTool[16][eeps_MsDelayPerDegreeMoved] = 0;
428 ProcessSteps_LoadTool[17][eeps_MsDelayPerDegreeMoved] = 0;
429 ProcessSteps_LoadTool[18][eeps_MsDelayPerDegreeMoved] = 0;
430 ProcessSteps_LoadTool[19][eeps_MsDelayPerDegreeMoved] = 0;
431 ProcessSteps_LoadTool[20][eeps_MsDelayPerDegreeMoved] = 0;
432
433 ProcessSteps_LoadTool[0][eeps_MsDelayAfterCommandSent] = 450; //160;
434 ProcessSteps_LoadTool[1][eeps_MsDelayAfterCommandSent] = 250; //150; //160;
435 ProcessSteps_LoadTool[2][eeps_MsDelayAfterCommandSent] = 250; //150; //160;
436 ProcessSteps_LoadTool[3][eeps_MsDelayAfterCommandSent] = 350; //150; //80;
437 ProcessSteps_LoadTool[4][eeps_MsDelayAfterCommandSent] = 100;
438 ProcessSteps_LoadTool[5][eeps_MsDelayAfterCommandSent] = 150;
439 ProcessSteps_LoadTool[6][eeps_MsDelayAfterCommandSent] = 30; //130;
440 ProcessSteps_LoadTool[7][eeps_MsDelayAfterCommandSent] = 120;
441 ProcessSteps_LoadTool[8][eeps_MsDelayAfterCommandSent] = 300; //200; //90;
442 ProcessSteps_LoadTool[9][eeps_MsDelayAfterCommandSent] = 500; //260; //rotate to
under extruder
443 ProcessSteps_LoadTool[10][eeps_MsDelayAfterCommandSent] = 300; //110; //give the
hotend time to stabilize before moving up to heaterblock bore
444 ProcessSteps_LoadTool[11][eeps_MsDelayAfterCommandSent] = 300; //220;
445 ProcessSteps_LoadTool[12][eeps_MsDelayAfterCommandSent] = 130; //cannot unlock
without hitting the cooling shroud. Must have delay. //30; //130;
446 ProcessSteps_LoadTool[13][eeps_MsDelayAfterCommandSent] = 230; //130; //fully
inserted into extruder
447 ProcessSteps_LoadTool[14][eeps_MsDelayAfterCommandSent] = 300; //200; //110; //Lock
the hotend into extruder
448 ProcessSteps_LoadTool[15][eeps_MsDelayAfterCommandSent] = 90;
449 ProcessSteps_LoadTool[16][eeps_MsDelayAfterCommandSent] = 200; //110; //jerk nozzle
release from hotend collar
450 ProcessSteps_LoadTool[17][eeps_MsDelayAfterCommandSent] = 90;
451 ProcessSteps_LoadTool[18][eeps_MsDelayAfterCommandSent] = 180;
452 ProcessSteps_LoadTool[19][eeps_MsDelayAfterCommandSent] = 1000; //700;
453 ProcessSteps_LoadTool[20][eeps_MsDelayAfterCommandSent] = 160;
454
455 ProcessSteps_LoadTool[0][eeps_StepType] = eeRegularStep;
456 ProcessSteps_LoadTool[1][eeps_StepType] = eeRegularStep;
457 ProcessSteps_LoadTool[2][eeps_StepType] = eeRegularStep;
458 ProcessSteps_LoadTool[3][eeps_StepType] = eeToolHolderPrepUNrotate; //eeRegularStep;
//rotate connect with nozzle collar
459 ProcessSteps_LoadTool[4][eeps_StepType] = eeRegularStep;
460 ProcessSteps_LoadTool[5][eeps_StepType] = eeRegularStep;
461 ProcessSteps_LoadTool[6][eeps_StepType] = eeRegularStep; //eeAddHalfDegreePrecision;
//eeRegularStep;
462 ProcessSteps_LoadTool[7][eeps_StepType] = eeRegularStep;
463 ProcessSteps_LoadTool[8][eeps_StepType] = eeButtonCheck_HoldingTool;
464 ProcessSteps_LoadTool[9][eeps_StepType] = eeRegularStep; //eeAddHalfDegreePrecision;
//eeRegularStep; //rotate centered under heater block bore
465 ProcessSteps_LoadTool[10][eeps_StepType] = eeRegularStep;
466 ProcessSteps_LoadTool[11][eeps_StepType] = eeRegularStep;
467 ProcessSteps_LoadTool[12][eeps_StepType] = eeRegularStep;
468 ProcessSteps_LoadTool[13][eeps_StepType] = eeRegularStep;
469 ProcessSteps_LoadTool[14][eeps_StepType] = eeRegularStep;
470 ProcessSteps_LoadTool[15][eeps_StepType] = eeRegularStep;

```



```

471 ProcessSteps_LoadTool[16][eeps_StepType] = eeRegularStep;
472 ProcessSteps_LoadTool[17][eeps_StepType] = eeRegularStep;
473 ProcessSteps_LoadTool[18][eeps_StepType] = eeRegularStep;
474 ProcessSteps_LoadTool[19][eeps_StepType] = eeButtonCheck_Empty;
475 ProcessSteps_LoadTool[20][eeps_StepType] = eeRegularStep;
476
477
478 ProcessSteps_UnloadTool[0][eeps_ServoNumber] = s_Tool_Rotate;
479 ProcessSteps_UnloadTool[1][eeps_ServoNumber] = s_Tool_Height;
480 ProcessSteps_UnloadTool[2][eeps_ServoNumber] = s_Tool_Rotate;
481 ProcessSteps_UnloadTool[3][eeps_ServoNumber] = s_Tool_Rotate;
482 ProcessSteps_UnloadTool[4][eeps_ServoNumber] = s_QuickSwapHotend_Lock;
483 ProcessSteps_UnloadTool[5][eeps_ServoNumber] = s_Tool_Height; //down to cutting
height
484 ProcessSteps_UnloadTool[6][eeps_ServoNumber] = s_Tool_Lock;
485 ProcessSteps_UnloadTool[7][eeps_ServoNumber] = s_QuickSwapHotend_Lock;
486 ProcessSteps_UnloadTool[8][eeps_ServoNumber] = s_Cutter_Rotate;
487 ProcessSteps_UnloadTool[9][eeps_ServoNumber] = s_Cutter_Action;
488 ProcessSteps_UnloadTool[10][eeps_ServoNumber] = s_Cutter_Action; //here
489 ProcessSteps_UnloadTool[11][eeps_ServoNumber] = s_Tool_Height;
490 ProcessSteps_UnloadTool[12][eeps_ServoNumber] = s_Tool_Rotate;
491 ProcessSteps_UnloadTool[13][eeps_ServoNumber] = s_Cutter_Action;
492 ProcessSteps_UnloadTool[14][eeps_ServoNumber] = s_Cutter_Action;
493 ProcessSteps_UnloadTool[15][eeps_ServoNumber] = s_Cutter_Rotate;
494 ProcessSteps_UnloadTool[16][eeps_ServoNumber] = s_Tool_Height; //lowest height
495 ProcessSteps_UnloadTool[17][eeps_ServoNumber] = s_Tool_Rotate; //to check button
496 ProcessSteps_UnloadTool[18][eeps_ServoNumber] = s_Tool_Rotate;
497 ProcessSteps_UnloadTool[19][eeps_ServoNumber] = s_Tool_Height;
498 ProcessSteps_UnloadTool[20][eeps_ServoNumber] = s_Tool_Lock;
499 ProcessSteps_UnloadTool[21][eeps_ServoNumber] = s_Tool_Height;
500 ProcessSteps_UnloadTool[22][eeps_ServoNumber] = s_Tool_Height;
501 ProcessSteps_UnloadTool[23][eeps_ServoNumber] = s_Tool_Rotate;
502 ProcessSteps_UnloadTool[24][eeps_ServoNumber] = s_Tool_Rotate;
503 ProcessSteps_UnloadTool[25][eeps_ServoNumber] = s_Tool_Height;
504 ProcessSteps_UnloadTool[26][eeps_ServoNumber] = s_Tool_Rotate;
505 ProcessSteps_UnloadTool[27][eeps_ServoNumber] = s_Tool_Rotate;
506 ProcessSteps_UnloadTool[28][eeps_ServoNumber] = s_WasteCup_Action;
507 ProcessSteps_UnloadTool[29][eeps_ServoNumber] = s_WasteCup_Action;
508
509 ProcessSteps_UnloadTool[0][eeps_Degrees] =
pos_Tool_Rotate_UnderExtruder_ReleasedFromNozzleCollar;
510 ProcessSteps_UnloadTool[1][eeps_Degrees] =
pos_Tool_Height_ToolFullyInsertedIntoExtruder_NoPressure;
//pos_Tool_Height_ToolFullyInsertedIntoExtruder_ScrappingHotendMildPressure;
//pos_Tool_Height_ToolFullyInsertedIntoExtruder_ScrappingHotendMildPressure;
511 ProcessSteps_UnloadTool[2][eeps_Degrees] =
pos_Tool_Rotate_UnderExtruder_JerkConnectWithNozzleCollar;
512 ProcessSteps_UnloadTool[3][eeps_Degrees] =
pos_Tool_Rotate_UnderExtruder_ConnectWithNozzleCollar;
513 ProcessSteps_UnloadTool[4][eeps_Degrees] = pos_QuickSwapHotend_Lock_Unlocked;
514 ProcessSteps_UnloadTool[5][eeps_Degrees] = pos_Tool_Height_ToolLowered_CuttingHeight;
515 ProcessSteps_UnloadTool[6][eeps_Degrees] = pos_Tool_Lock_Locked;
516 ProcessSteps_UnloadTool[7][eeps_Degrees] = pos_QuickSwapHotend_Lock_Locked;
517 ProcessSteps_UnloadTool[8][eeps_Degrees] = pos_Cutter_Rotate_Cutting;
518 ProcessSteps_UnloadTool[9][eeps_Degrees] = pos_Cutter_Action_Cut;
519 ProcessSteps_UnloadTool[10][eeps_Degrees] = pos_Cutter_Action_Open; //here
520 ProcessSteps_UnloadTool[11][eeps_Degrees] =
pos_Tool_Height_ToolLowered_BelowCutterJaws;
521 ProcessSteps_UnloadTool[12][eeps_Degrees] = pos_Tool_Rotate_PastWasteCup;
522 ProcessSteps_UnloadTool[13][eeps_Degrees] = pos_Cutter_Action_Cut;
523 ProcessSteps_UnloadTool[14][eeps_Degrees] = pos_Cutter_Action_Open;
524 ProcessSteps_UnloadTool[15][eeps_Degrees] = pos_Cutter_Rotate_Stowed;
525 ProcessSteps_UnloadTool[16][eeps_Degrees] = pos_Tool_Height_LowestLevel;
526 ProcessSteps_UnloadTool[17][eeps_Degrees] = pos_Tool_Rotate_ButtonToolCheck;
//should have tool
527 ProcessSteps_UnloadTool[18][eeps_Degrees] =
pos_Tool_Rotate_UnderToolHolder_CenteredUnderCurrentTool; //ready to lift into
position
528 ProcessSteps_UnloadTool[19][eeps_Degrees] =

```

```

pos_Tool_Height_ToolLoweredButStillInHolder;
529 ProcessSteps_UnloadTool[20][eeps_Degrees] = pos_Tool_Lock_Unlocked;
530 ProcessSteps_UnloadTool[21][eeps_Degrees] = pos_Tool_Height_ToolFullyInsertedInHolder
;
531 ProcessSteps_UnloadTool[22][eeps_Degrees] =
pos_Tool_Height_ToolFullyInsertedInHolder_NoPressure;
532 ProcessSteps_UnloadTool[23][eeps_Degrees] =
pos_Tool_Rotate_ReleaseFromHotendUnderToolHolder;
533 ProcessSteps_UnloadTool[24][eeps_Degrees] = pos_Tool_Rotate_BetweenBothNozzles;
//pos_Tool_Rotate_LeftOfToolInHolder;
534 ProcessSteps_UnloadTool[25][eeps_Degrees] = pos_Tool_Height_LowestLevel;
535 ProcessSteps_UnloadTool[26][eeps_Degrees] = pos_Tool_Rotate_ButtonToolCheck;
536 ProcessSteps_UnloadTool[27][eeps_Degrees] =
pos_Tool_Rotate_ButtingTheToolToTheLeftOfNext;
537 ProcessSteps_UnloadTool[28][eeps_Degrees] = pos_WasteCup_Action_Dump;
538 ProcessSteps_UnloadTool[29][eeps_Degrees] = pos_WasteCup_Action_Fill;
539
540 ProcessSteps_UnloadTool[0][eeps_IsIncludedInMMU] = true;
541 ProcessSteps_UnloadTool[1][eeps_IsIncludedInMMU] = true;
542 ProcessSteps_UnloadTool[2][eeps_IsIncludedInMMU] = true;
543 ProcessSteps_UnloadTool[3][eeps_IsIncludedInMMU] = true;
544 ProcessSteps_UnloadTool[4][eeps_IsIncludedInMMU] = true;
545 ProcessSteps_UnloadTool[5][eeps_IsIncludedInMMU] = true;
546 ProcessSteps_UnloadTool[6][eeps_IsIncludedInMMU] = false; //needs to be completely
removed it's not used in either...
547 ProcessSteps_UnloadTool[7][eeps_IsIncludedInMMU] = true;
548 ProcessSteps_UnloadTool[8][eeps_IsIncludedInMMU] = true;
549 ProcessSteps_UnloadTool[9][eeps_IsIncludedInMMU] = true;
550 ProcessSteps_UnloadTool[10][eeps_IsIncludedInMMU] = true; //here
551 ProcessSteps_UnloadTool[11][eeps_IsIncludedInMMU] = false; //tool down almost all
the way
552 ProcessSteps_UnloadTool[12][eeps_IsIncludedInMMU] = false; //tool rotate
553 ProcessSteps_UnloadTool[13][eeps_IsIncludedInMMU] = false; //pos_Cutter_Action_Cut;
554 ProcessSteps_UnloadTool[14][eeps_IsIncludedInMMU] = false; //pos_Cutter_Action_Open;
555 ProcessSteps_UnloadTool[15][eeps_IsIncludedInMMU] = true;
556 ProcessSteps_UnloadTool[16][eeps_IsIncludedInMMU] = true;
557 ProcessSteps_UnloadTool[17][eeps_IsIncludedInMMU] = true;
558 ProcessSteps_UnloadTool[18][eeps_IsIncludedInMMU] = true;
559 ProcessSteps_UnloadTool[19][eeps_IsIncludedInMMU] = true;
560 ProcessSteps_UnloadTool[20][eeps_IsIncludedInMMU] = true;
561 ProcessSteps_UnloadTool[21][eeps_IsIncludedInMMU] = true;
562 ProcessSteps_UnloadTool[22][eeps_IsIncludedInMMU] = true;
563 ProcessSteps_UnloadTool[23][eeps_IsIncludedInMMU] = true;
564 ProcessSteps_UnloadTool[24][eeps_IsIncludedInMMU] = true;
565 ProcessSteps_UnloadTool[25][eeps_IsIncludedInMMU] = true;
566 ProcessSteps_UnloadTool[26][eeps_IsIncludedInMMU] = true;
567 ProcessSteps_UnloadTool[27][eeps_IsIncludedInMMU] = true;
568 ProcessSteps_UnloadTool[28][eeps_IsIncludedInMMU] = false;
//pos_WasteCup_Action_Dump;
569 ProcessSteps_UnloadTool[29][eeps_IsIncludedInMMU] = false;
//pos_WasteCup_Action_Fill;
570
571 ProcessSteps_UnloadTool[0][eeps_MsDelayPerDegreeMoved] = 0;
572 ProcessSteps_UnloadTool[1][eeps_MsDelayPerDegreeMoved] = 0;
573 ProcessSteps_UnloadTool[2][eeps_MsDelayPerDegreeMoved] = 0;
574 ProcessSteps_UnloadTool[3][eeps_MsDelayPerDegreeMoved] = 0;
575 ProcessSteps_UnloadTool[4][eeps_MsDelayPerDegreeMoved] = 0;
576 ProcessSteps_UnloadTool[5][eeps_MsDelayPerDegreeMoved] = 10; //lower to cutting
height
577 ProcessSteps_UnloadTool[6][eeps_MsDelayPerDegreeMoved] = 0;
578 ProcessSteps_UnloadTool[7][eeps_MsDelayPerDegreeMoved] = 0;
579 ProcessSteps_UnloadTool[8][eeps_MsDelayPerDegreeMoved] = 6; //this makes the end
position more repeatable than allowing the servo to control it's deceleration //6;
//0; //cutter rotate
580 ProcessSteps_UnloadTool[9][eeps_MsDelayPerDegreeMoved] = 0;
581 ProcessSteps_UnloadTool[10][eeps_MsDelayPerDegreeMoved] = 0; //here
582 ProcessSteps_UnloadTool[11][eeps_MsDelayPerDegreeMoved] = 0;
583 ProcessSteps_UnloadTool[12][eeps_MsDelayPerDegreeMoved] = 0;
584 ProcessSteps_UnloadTool[13][eeps_MsDelayPerDegreeMoved] = 0;

```

```

585 ProcessSteps_UnloadTool[14][eeps_MsDelayPerDegreeMoved] = 0;
586 ProcessSteps_UnloadTool[15][eeps_MsDelayPerDegreeMoved] = 0; //go full speed so that
the tool can be stowed symultaneously //6 slow to keep the servo from dying //0;
//cutter rotate
587 ProcessSteps_UnloadTool[16][eeps_MsDelayPerDegreeMoved] = 0;
588 ProcessSteps_UnloadTool[17][eeps_MsDelayPerDegreeMoved] = 0;
589 ProcessSteps_UnloadTool[18][eeps_MsDelayPerDegreeMoved] = 0;
590 ProcessSteps_UnloadTool[19][eeps_MsDelayPerDegreeMoved] = 0; //60;
591 ProcessSteps_UnloadTool[20][eeps_MsDelayPerDegreeMoved] = 0;
592 ProcessSteps_UnloadTool[21][eeps_MsDelayPerDegreeMoved] = 0;
593 ProcessSteps_UnloadTool[22][eeps_MsDelayPerDegreeMoved] = 0;
594 ProcessSteps_UnloadTool[23][eeps_MsDelayPerDegreeMoved] = 0;
595 ProcessSteps_UnloadTool[24][eeps_MsDelayPerDegreeMoved] = 0;
596 ProcessSteps_UnloadTool[25][eeps_MsDelayPerDegreeMoved] = 0;
597 ProcessSteps_UnloadTool[26][eeps_MsDelayPerDegreeMoved] = 0;
598 ProcessSteps_UnloadTool[27][eeps_MsDelayPerDegreeMoved] = 0;
599 ProcessSteps_UnloadTool[28][eeps_MsDelayPerDegreeMoved] = 0;
600 ProcessSteps_UnloadTool[29][eeps_MsDelayPerDegreeMoved] = 0;
601
602 ProcessSteps_UnloadTool[0][eeps_MsDelayAfterCommandSent] = 550; //350;
603 ProcessSteps_UnloadTool[1][eeps_MsDelayAfterCommandSent] = 630; //430; //330; //230;
//up to nozzle collar level
604 ProcessSteps_UnloadTool[2][eeps_MsDelayAfterCommandSent] = 400; //200; //150; //90;
605 ProcessSteps_UnloadTool[3][eeps_MsDelayAfterCommandSent] = 90;
606 ProcessSteps_UnloadTool[4][eeps_MsDelayAfterCommandSent] = 110;
607 ProcessSteps_UnloadTool[5][eeps_MsDelayAfterCommandSent] = 0;
608 ProcessSteps_UnloadTool[6][eeps_MsDelayAfterCommandSent] = 0; //110; //lock tool.
lock moved to SetServoPosition()
609 ProcessSteps_UnloadTool[7][eeps_MsDelayAfterCommandSent] = 0;
610 ProcessSteps_UnloadTool[8][eeps_MsDelayAfterCommandSent] = 550; //650; //550; //500;
//190; //cutter rotate
611 ProcessSteps_UnloadTool[9][eeps_MsDelayAfterCommandSent] = 600; //550; //370; //130;
//cut
612 ProcessSteps_UnloadTool[10][eeps_MsDelayAfterCommandSent] = 200; //250; //370; //130;
//open //here
613 ProcessSteps_UnloadTool[11][eeps_MsDelayAfterCommandSent] = 0; //50; //uncomment for
Palette
614 ProcessSteps_UnloadTool[12][eeps_MsDelayAfterCommandSent] = 0; //90; //uncomment for
Palette
615 ProcessSteps_UnloadTool[13][eeps_MsDelayAfterCommandSent] = 0; //370; //130; //cut
//uncomment for Palette
616 ProcessSteps_UnloadTool[14][eeps_MsDelayAfterCommandSent] = 0; //370; //130; //open
//uncomment for Palette
617 ProcessSteps_UnloadTool[15][eeps_MsDelayAfterCommandSent] = 0; //cutter rotate
stowed //75; //50; //100; //200; //100; //50; //need slight delay just for the cutter
to rotate a little away from the filament and break the strand //130;s_Cutter_Rotate
no delay needed when stowing the cutter
618 ProcessSteps_UnloadTool[16][eeps_MsDelayAfterCommandSent] = 70;
619 ProcessSteps_UnloadTool[17][eeps_MsDelayAfterCommandSent] = 1000; //700; //200;
//button check should have tool
620 ProcessSteps_UnloadTool[18][eeps_MsDelayAfterCommandSent] = 80;
621 ProcessSteps_UnloadTool[19][eeps_MsDelayAfterCommandSent] = 180;
622 ProcessSteps_UnloadTool[20][eeps_MsDelayAfterCommandSent] = 0; //130;
623 ProcessSteps_UnloadTool[21][eeps_MsDelayAfterCommandSent] = 80;
624 ProcessSteps_UnloadTool[22][eeps_MsDelayAfterCommandSent] = 70;
625 ProcessSteps_UnloadTool[23][eeps_MsDelayAfterCommandSent] = 300; //400; //50;
//release from hotend which is now stowed
626 ProcessSteps_UnloadTool[24][eeps_MsDelayAfterCommandSent] = 50;
627 ProcessSteps_UnloadTool[25][eeps_MsDelayAfterCommandSent] = 350; //150; //lower to
lowest level
628 ProcessSteps_UnloadTool[26][eeps_MsDelayAfterCommandSent] = 400; //300; //180;
button check should be empty
629 ProcessSteps_UnloadTool[27][eeps_MsDelayAfterCommandSent] = 100;
630 ProcessSteps_UnloadTool[28][eeps_MsDelayAfterCommandSent] = 180;
631 ProcessSteps_UnloadTool[29][eeps_MsDelayAfterCommandSent] = 100;
632
633 ProcessSteps_UnloadTool[0][eeps_StepType] = eeRegularStep;
634 ProcessSteps_UnloadTool[1][eeps_StepType] = eeRegularStep;
635 ProcessSteps_UnloadTool[2][eeps_StepType] = eeRegularStep;

```

```

636 ProcessSteps_UnloadTool[3][eeps_StepType] = eeRegularStep;
637 ProcessSteps_UnloadTool[4][eeps_StepType] = eeRegularStep;
638 ProcessSteps_UnloadTool[5][eeps_StepType] = eeExtrude; //lower to cutting height.
    extrude stage 2
639 ProcessSteps_UnloadTool[6][eeps_StepType] = eeAddHalfDegreePrecision;
    //eeRegularStep; //eeAddHalfDegreePrecision; //eeRegularStep; //locking the
    nozzle-hotend into the end effector
640 ProcessSteps_UnloadTool[7][eeps_StepType] = eeRegularStep;
641 ProcessSteps_UnloadTool[8][eeps_StepType] = eeRegularStep;
    //eeAddHalfDegreePrecision; //eeRegularStep; //rotate to cutting position
642 ProcessSteps_UnloadTool[9][eeps_StepType] = eeRegularStep;
643 ProcessSteps_UnloadTool[10][eeps_StepType] = eeRegularStep; //open cutters //here
644 ProcessSteps_UnloadTool[11][eeps_StepType] = eeRegularStep; //eeRetract;Not anymore
    that these steps are only for the palette //after cutters are open. retract
645 ProcessSteps_UnloadTool[12][eeps_StepType] = eeRegularStep;
646 ProcessSteps_UnloadTool[13][eeps_StepType] = eeRegularStep;
647 ProcessSteps_UnloadTool[14][eeps_StepType] = eeRegularStep;
648 ProcessSteps_UnloadTool[15][eeps_StepType] = eeRetract; //eeRegularStep;
649 ProcessSteps_UnloadTool[16][eeps_StepType] = eeRegularStep;
650 ProcessSteps_UnloadTool[17][eeps_StepType] = eeButtonCheck_HoldingTool;
651 ProcessSteps_UnloadTool[18][eeps_StepType] = eeRegularStep;
652 ProcessSteps_UnloadTool[19][eeps_StepType] = eeRegularStep;
653 ProcessSteps_UnloadTool[20][eeps_StepType] = eeRegularStep;
654 ProcessSteps_UnloadTool[21][eeps_StepType] = eeRegularStep;
655 ProcessSteps_UnloadTool[22][eeps_StepType] = eeRegularStep;
656 ProcessSteps_UnloadTool[23][eeps_StepType] = eeToolHolderPrepRotate;
657 ProcessSteps_UnloadTool[24][eeps_StepType] = eeRegularStep;
658 ProcessSteps_UnloadTool[25][eeps_StepType] = eeRegularStep;
659 ProcessSteps_UnloadTool[26][eeps_StepType] = eeButtonCheck_Empty;
660 ProcessSteps_UnloadTool[27][eeps_StepType] = eeRegularStep;
661 ProcessSteps_UnloadTool[28][eeps_StepType] = eeRegularStep;
662 ProcessSteps_UnloadTool[29][eeps_StepType] = eeRegularStep;
663
664
665 ClearSerialBuffer();
666
667 setupComplete = true;
668 loopStarted = true;
669
670 inputString.reserve(15);
671 updateLCD("Ready to Swap!", "Insert: Empty");
672 }
673
674
675 int checkParity(String message) {
676     int count = 0;
677     for (int i = 0; i < message.length(); i++) {
678         int value = message[i];
679         while (value) {
680             count++;
681             value = value & (value - 1);
682         }
683     }
684     return ~count & 1; // returns 1 for odd parity, 0 for even parity
685 }
686
687
688 void loop() {
689     if (stringComplete) {
690         int inputParity = inputString.charAt(inputString.length() - 1) - '0';
691         String inputMessage = inputString.substring(0, inputString.length() - 1);
692         String inputMessage_TextPart = inputMessage;
693         int inputMessage_NumberPart = 0;
694
695         // Find the index where the number starts
696         int numIndex = -1;
697         for (int i = 0; i < inputMessage.length(); i++) {
698             if (isdigit(inputMessage.charAt(i))) {
699                 numIndex = i;

```

```

700         break;
701     }
702 }
703
704 // Extract the text part and number part if a number is found
705 if (numIndex != -1) {
706     inputMessage_TextPart = inputMessage.substring(0, numIndex);
707     inputMessage_NumberPart = inputMessage.substring(numIndex).toInt();
708 }
709
710 Serial.print("number index: ");
711 Serial.println(numIndex);
712 Serial.print("inputMessage: ");
713 Serial.println(inputMessage);
714 Serial.print("inputMessage_TextPart: ");
715 Serial.println(inputMessage_TextPart);
716 Serial.print("inputMessage_NumberPart: ");
717 Serial.println(inputMessage_NumberPart);
718
719 if (checkParity(inputMessage) == inputParity) {
720     if (inputMessage_TextPart == "octoprint") {
721         printWithParity("swapper");
722     } else if (inputMessage_TextPart == "load_insert") {
723         insertNumber = inputMessage_NumberPart;
724         load_insert(insertNumber);
725         printWithParity("ok");
726         delay(3000);
727         updateLCD("Ready to Swap!", "Insert: " + String(insertNumber));
728     } else if (inputMessage_TextPart == "unload_connect") {
729         updateLCD_line1("Connect");
730         unload_connect();
731         printWithParity("ok");
732         delay(3000);
733     } else if (inputMessage_TextPart == "unload_pulldown") {
734         updateLCD_line1("Pulldown");
735         unload_pulldown();
736         printWithParity("ok");
737         delay(3000);
738     } else if (inputMessage_TextPart == "unload_deploycutter") {
739         updateLCD_line1("Deploy cutter");
740         unload_deploycutter();
741         printWithParity("ok");
742         delay(3000);
743     } else if (inputMessage_TextPart == "unload_cut") {
744         updateLCD_line1("Cut");
745         unload_cut();
746         printWithParity("ok");
747         delay(3000);
748     } else if (inputMessage_TextPart == "unload_stowcutter") {
749         updateLCD_line1("Stow cutter");
750         unload_stowcutter();
751         printWithParity("ok");
752         delay(3000);
753     } else if (inputMessage_TextPart == "unload_dumpwaste") {
754         updateLCD_line1("Dump waste");
755         unload_dumpwaste();
756         printWithParity("ok");
757         delay(3000);
758     } else if (inputMessage_TextPart == "unloaded_message") {
759         insertNumber = 0;
760         updateLCD("Ready to Swap!", "Insert: Empty");
761         printWithParity("ok");
762         delay(3000);
763     } else if (inputMessage_TextPart == "swap_message") {
764         if (insertNumber == 0) {
765             updateLCD_line1("Swapping -> " + String(inputMessage_NumberPart));
766         } else {
767             updateLCD_line1("Swapping " + String(insertNumber) + " -> " + String(

```

```

768     }
769     printWithParity("ok");
770 } else if (inputMessage_TextPart == "wiper_deploy") {
771     updateLCD_line1("Deploy wiper");
772     wiper_deploy();
773     printWithParity("ok");
774     delay(3000);
775     updateLCD_line1("Wiper deployed");
776 } else if (inputMessage_TextPart == "wiper_stow") {
777     updateLCD_line1("Stow wiper");
778     wiper_stow();
779     printWithParity("ok");
780     delay(3000);
781     updateLCD("Ready to Swap!", "Insert: " + insertNumber);
782 } else {
783     // Handle command not found
784     printWithParity("Command not found");
785 }
786 } else {
787     Serial.println("Parity check failed");
788 }
789
790 inputString = "";
791 stringComplete = false;
792 }
793 }
794
795
796
797
798 // function that continuously reads incoming serial data,
799 //appending each character to a string until it encounters a newline character,
800 //which signals the end of a message.
801 void serialEvent() {
802     while (Serial.available()) {
803         char inChar = (char)Serial.read();
804         if (inChar != '\n' && inChar != '\r') {
805             inputString += inChar;
806         }
807         if (inChar == '\n') {
808             stringComplete = true;
809         }
810     }
811 }
812
813 void ClearSerialBuffer()
814 {
815     while (Serial.available())
816     {
817         Serial.read();
818     }
819 }
820
821
822
823 void load_insert(int toolToLoad)
824 {
825
826     toolIsLoaded = true;
827     bool errorResume = false;
828     //return; //js
829
830     currentStepOfProcess = 0;
831     ToolHolder_AlignToThisTool(toolToLoad);
832
833
834     numberOfStepsToProcess = numOfProcessSteps_LoadTool;
835     CurrentProcessType = eeLoadTool;
836

```



```

837 while (currentStepOfProcess < numberOfStepsToProcess)
838 {
839     // RefreshPositionServoInfo(); //moved to the ProcessStep method
840     // SetServoPosition(ps_currentServo, ps_targetAngle,
841     ps_msDelayPerDegreeMoved); //moved to the ProcessStep method
842     // delay(ps_msDelayAfterCommandSent); //moved to the ProcessStep method
843     // Serial.println("process step");
844     ProcessStep(); //***this delays the next step, checks the buttons, and runs
845     special extrudes on the printer
846
847     if(!InErrorState)
848     {
849         currentStepOfProcess++;
850     }
851     //if in error button check
852     //unwind to waiting for user input
853     else
854     {
855         // Serial.println("park the tool actuator.");
856         SetServoPosition(s_Tool_Rotate, 95, 0);
857         //unlock the tool holder
858         pulselength = map(180, 0, servos[s_Tool_Lock][eeMaxAngle], servo_pwm_min,
859         servo_pwm_max);
860         pwm.setPWM(servos[s_Tool_Lock][eePinNum], 0, pulselength);
861         delay(200);
862
863         do
864         {
865             buttonPress = analogRead(0);
866             // Serial.println(buttonPress);
867             }while (buttonPress < 820 || buttonPress > 830);
868
869             //return the tool lock to the pre-error state
870             pulselength = map(servos[s_Tool_Lock][eeCurrentAngle], 0, servos[s_Tool_Lock
871             ][eeMaxAngle], servo_pwm_min, servo_pwm_max);
872             pwm.setPWM(servos[s_Tool_Lock][eePinNum], 0, pulselength);
873             delay(200);
874
875             Serial.println("Error was reset");
876             lcd.clear();
877             lcd.setCursor(0,0);
878             lcd.print("Loading");
879             InErrorState = false;
880         }
881     }
882
883     // toolIsLoaded = true; //should be at the top so that it works when SetupMode
884     currentStepOfProcess = 0;
885
886     lcd.setCursor(0,1);
887     lcd.print("Heating nozzle");
888     //delay(30000); //no delay needed because in PS there is an M109 wait for temp.
889     //25000); //30000); //must delay some time so the nozzle-hotend can heatup
890     lcd.setCursor(0,1);
891     lcd.print("          ");
892
893     ClearSerialBuffer();
894 }
895
896 void unload_connect(){
897 }
898
899 void unload_pullupdown(){
900 }

```

```

901
902 void unload_deploycutter(){
903
904 }
905
906 void unload_cut(){
907
908 }
909
910 void unload_stowcutter(){
911
912 }
913
914 void unload_dumpwaste(){
915
916 }
917
918 void wiper_deploy(){
919
920 }
921
922 void wiper_stow(){
923
924 }
925
926
927 void Unload()
928 {
929     if (!toolIsLoaded) return;
930
931     toolIsLoaded = false;
932     //return; //js
933
934     currentStepOfProcess = 0;
935     numberOfStepsToProcess = numOfProcessSteps_UnloadTool;
936     CurrentProcessType = eeUnloadTool;
937     //already aligned to the current tool. The currentTool is set in
     ToolHolder_AlignToThisTool()
938
939
940     while (currentStepOfProcess < numberOfStepsToProcess)
941     {
942         ProcessStep(); /***this delays the next step, checks the buttons, and runs
         special extrudes on the printer
943
944         if(!InErrorState)
945         {
946             do
947             {
948                 currentStepOfProcess++;
949             } while (ProcessSteps_UnloadTool[currentStepOfProcess][eeps_IsIncludedInMMU]
             == false);
950         }
951         //if in error button check
952         //unwind to waiting for user input
953         else
954         {
955             SetServoPosition(s_Tool_Rotate, 95, 0);
956             //unlock the tool holder
957             pulselength = map(180, 0, servos[s_Tool_Lock][eeMaxAngle], servo_pwm_min,
             servo_pwm_max);
958             pwm.setPWM(servos[s_Tool_Lock][eePinNum], 0, pulselength);
959             delay(200);
960
961             do
962             {
963                 buttonPress = analogRead(0);
964                 // Serial.println(buttonPress);
965             }while (buttonPress < 820 || buttonPress > 830);

```

```

966
967         //return the tool lock to the pre-error state
968         pulselength = map(servos[s_Tool_Lock][eeCurrentAngle], 0, servos[s_Tool_Lock]
        [eeMaxAngle], servo_pwm_min, servo_pwm_max);
969         pwm.setPWM(servos[s_Tool_Lock][eePinNum], 0, pulselength);
970         delay(200);
971
972         Serial.println("Error was reset");
973         lcd.clear();
974         lcd.setCursor(0,0);
975         lcd.print("Unloading");
976         InErrorState = false;
977     }
978 }
979 currentStepOfProcess = 0;
980 }
981
982 void RefreshPositionServoInfo()
983 {
984     switch(CurrentProcessType)
985     {
986         case eeLoadTool: //Load process
987             ps_currentServo = ProcessSteps_LoadTool[currentStepOfProcess][
                eeps_ServoNumber];
988             ps_targetAngle = ProcessSteps_LoadTool[currentStepOfProcess][eeps_Degrees];
989             ps_msDelayPerDegreeMoved = ProcessSteps_LoadTool[currentStepOfProcess][
                eeps_MsDelayPerDegreeMoved];
990             ps_msDelayAfterCommandSent = ProcessSteps_LoadTool[currentStepOfProcess][
                eeps_MsDelayAfterCommandSent] + msDelayAfterCommandSent_Buffer;
991             break;
992         case eeUnloadTool: //unload process
993             ps_currentServo = ProcessSteps_UnloadTool[currentStepOfProcess][
                eeps_ServoNumber];
994             ps_targetAngle = ProcessSteps_UnloadTool[currentStepOfProcess][eeps_Degrees];
995             ps_msDelayPerDegreeMoved = ProcessSteps_UnloadTool[currentStepOfProcess][
                eeps_MsDelayPerDegreeMoved];
996             ps_msDelayAfterCommandSent = ProcessSteps_UnloadTool[currentStepOfProcess][
                eeps_MsDelayAfterCommandSent] + msDelayAfterCommandSent_Buffer;
997             break;
998     }
999 }
1000
1001 bool CheckButton_Pressed()
1002 {
1003     delay(10);
1004
1005     // if(digitalRead(CheckButton_Pin)==1)
1006     if(analogRead(CheckButton_Pin) > 1020)
1007     {
1008         return true;
1009     }
1010     else
1011     {
1012         return false;
1013     }
1014 }
1015
1016 void ProcessStep()
1017 {
1018     int stepType = 0;
1019
1020     switch(CurrentProcessType)
1021     {
1022         case eeLoadTool: //Load process
1023             stepType = ProcessSteps_LoadTool[currentStepOfProcess][eeps_StepType];
1024             break;
1025         case eeUnloadTool: //unload process
1026             stepType = ProcessSteps_UnloadTool[currentStepOfProcess][eeps_StepType];
1027             break;

```

```

1028     }
1029
1030
1031     switch(stepType)
1032     {
1033         case eeButtonCheck_Empty:
1034             RefreshPositionServoInfo();
1035             SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
1036             delay(ps_msDelayAfterCommandSent); //delay first, then check button.
1037             //otherwise the button cannot ever be pressed
1038
1039             if(CheckButton_Pressed() && ErrorCheckingEnabled)
1040             {
1041                 Serial.println("ERROR 1");
1042                 lcd.clear();
1043                 lcd.setCursor(0,0);
1044                 lcd.print("ERROR->Not Empty");
1045                 lcd.setCursor(0,1);
1046                 lcd.print("S to retry");
1047                 InErrorState = true;
1048             }
1049             else
1050             {
1051                 if(CheckButton_Pressed())
1052                 {
1053                     Serial.println("Button pressed.");
1054                 }
1055                 else
1056                 {
1057                     Serial.println("Button NOT pressed.");
1058                 }
1059
1060                 InErrorState = false;
1061             }
1062             break;
1063         case eeButtonCheck_HoldingTool:
1064             RefreshPositionServoInfo();
1065             SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
1066             delay(ps_msDelayAfterCommandSent); //delay first, then check button.
1067             //otherwise the button cannot ever be pressed
1068
1069             if(!CheckButton_Pressed() && ErrorCheckingEnabled)
1070             {
1071                 Serial.println("ERROR 2");
1072                 lcd.clear();
1073                 lcd.setCursor(0,0);
1074                 lcd.print("ERROR->Empty");
1075                 lcd.setCursor(0,1);
1076                 lcd.print("S to retry");
1077                 InErrorState = true;
1078             }
1079             else
1080             {
1081                 if(CheckButton_Pressed())
1082                 {
1083                     Serial.println("Button pressed.");
1084                 }
1085                 else
1086                 {
1087                     Serial.println("Button NOT pressed.");
1088                 }
1089
1090                 InErrorState = false;
1091             }
1092             break;
1093         case eeExtrude:
1094             LockToolPartWayThru = true;
1095
1096             // Serial.println("Extrude");

```

```

1095     Serial.write(90);
1096     Serial.write(91);
1097     Serial.write(93);
1098     Serial.write(94);
1099     Serial.write(1); //direction 1=extrude
1100     Serial.write(55); //53); //53 length
1101     Serial.write(66); //5); //80, 75, 72, 70,67, 65 //feedrate
1102
1103     delay(97); //delay before servo movement to allow the extrude to begin on the
        printer
1104
1105     RefreshPositionServoInfo();
1106     SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
1107
1108     delay(ps_msDelayAfterCommandSent);
1109     break;
1110 case eeRetract:
1111     // Serial.println("Retract");
1112     Serial.write(90);
1113     Serial.write(91);
1114     Serial.write(93);
1115     Serial.write(94);
1116     Serial.write(2); //direction 2=retract
1117     Serial.write(70); //56); //pass:56); Fail:53 //retract a little too much,
        then add back after load heat up IF this is a 'same color nozzle size
        switch' otherwise the tool change will restore the difference//53);
        //54);//63); //54); //53); //length
1118     Serial.write(65); //120); //3900//feedrate
1119
1120     RefreshPositionServoInfo();
1121     SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
1122     Serial.print("eeRetract ms delay:");
1123     Serial.println(ps_msDelayPerDegreeMoved);
1124     delay(ps_msDelayAfterCommandSent);
1125     break;
1126 case eeToolHolderPrepRotate://rotate the tool holder slightly to account for the
        pull of the end effector when releasing the nozzle collar
1127     servos[s_ToolHolder_Rotate][eeCurrentAngle] = servos[s_ToolHolder_Rotate][
        eeCurrentAngle] + eeToolHolderPrepRotate_Degrees;
1128     pulselength = map(servos[s_ToolHolder_Rotate][eeCurrentAngle], servoMinAngle,
        servos[s_ToolHolder_Rotate][eeMaxAngle], servo_pwm_min, servo_pwm_max);
1129     pwm.setPWM(servos[s_ToolHolder_Rotate][eePinNum], 0, pulselength);
1130
1131     RefreshPositionServoInfo();
1132     SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
1133     delay(ps_msDelayAfterCommandSent);
1134     break;
1135 case eeToolHolderPrepUNrotate://rotate the tool holder slightly to account for
        the pull of the end effector when releasing the nozzle collar
1136     //UNrotate it
1137     RefreshPositionServoInfo();
1138
1139     SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
        //rotate tool actuator
1140     delay(70); //60); //55); //65); //80); //120); //60); //50); //this delay
        always the tool actuator to begin moving, then the tool holder rotates at
        the same time and when the optimal position is acheived the end effector
        slips onto the nozzles collar.
1141
1142     //Begin rotate TOOL HOLDER
1143     pulselength = map(servos[s_ToolHolder_Rotate][eeCurrentAngle] -
        eeToolHolderPrepUNrotate_Degrees, servoMinAngle, servos[s_ToolHolder_Rotate][
        eeMaxAngle], servo_pwm_min, servo_pwm_max);
1144     pwm.setPWM(servos[s_ToolHolder_Rotate][eePinNum], 0, pulselength);
1145     //End rotate TOOL HOLDER
1146
1147     delay(ps_msDelayAfterCommandSent);
1148
1149     //rotate it back

```

```

1150     pulselength = map(servos[s_ToolHolder_Rotate][eeCurrentAngle], servoMinAngle,
1151                       servos[s_ToolHolder_Rotate][eeMaxAngle], servo_pwm_min, servo_pwm_max);
1152     pwm.setPWM(servos[s_ToolHolder_Rotate][eePinNum], 0, pulselength);
1153     break;
1154 case eeRegularStep:
1155     RefreshPositionServoInfo();
1156     SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
1157     delay(ps_msDelayAfterCommandSent);
1158     break;
1159 case eeAddHalfDegreePrecision:
1160     int precisionPulseLength = 0;
1161     RefreshPositionServoInfo();
1162     precisionPulseLength = fMap((float)ps_targetAngle + (float)0.5, 0, servos[
1163     ps_currentServo][eeMaxAngle], servo_pwm_min, servo_pwm_max);
1164     pwm.setPWM(servos[ps_currentServo][eePinNum], 0, precisionPulseLength);
1165     delay(ps_msDelayAfterCommandSent);
1166     break;
1167 }
1168 }
1169 void SetServoPosition(int ServoNum, int TargetAngle, int msDelay)
1170 {
1171     int currentAngle = servos[ServoNum][eeCurrentAngle];
1172     int angleDifference = TargetAngle - currentAngle;
1173
1174     int msCountedBeforeLock = 0;
1175
1176     //if the msDelay is zero then don't use a loop
1177     if(msDelay == 0)
1178     {
1179         //Serial.println("zero delay");
1180         servos[ServoNum][eeCurrentAngle] = TargetAngle;
1181         pulselength = map(servos[ServoNum][eeCurrentAngle], 0, servos[ServoNum][eeMaxAngle
1182         ], servo_pwm_min, servo_pwm_max);
1183         pwm.setPWM(servos[ServoNum][eePinNum], 0, pulselength);
1184     }
1185     //else use a loop to inject the delay and fake accel
1186     else
1187     {
1188         //Serial.println("ms delay");
1189         if (angleDifference > 0)
1190         {
1191             for(int i = currentAngle; i <= TargetAngle; i++)
1192             {
1193                 servos[ServoNum][eeCurrentAngle] = i;
1194                 pulselength = map(servos[ServoNum][eeCurrentAngle], 0, servos[ServoNum][
1195                 eeMaxAngle], servo_pwm_min, servo_pwm_max);
1196                 pwm.setPWM(servos[ServoNum][eePinNum], 0, pulselength);
1197                 delay(msDelay);
1198
1199                 //deploy the tool lock part way thru the extrude
1200                 if (LockToolPartWayThru)
1201                 {
1202                     msCountedBeforeLock += msDelay;
1203
1204                     //lock the tool
1205                     if(msCountedBeforeLock >= numMsUntilLock)
1206                     {
1207                         LockToolPartWayThru = false;
1208                         servos[s_Tool_Lock][eeCurrentAngle] = pos_Tool_Lock_Locked;
1209                         pulselength = map(servos[s_Tool_Lock][eeCurrentAngle], 0, servos[
1210                         ServoNum][eeMaxAngle], servo_pwm_min, servo_pwm_max);
1211                         pwm.setPWM(servos[s_Tool_Lock][eePinNum], 0, pulselength);
1212                     }
1213                 }
1214             }
1215         }
1216     }
1217 }

```



```
1214     else
1215     {
1216         for(int i = currentAngle; i >= TargetAngle; i--)
1217         {
1218             servos[ServoNum][eeCurrentAngle] = i;
1219             pulselength = map(servos[ServoNum][eeCurrentAngle], 0, servos[ServoNum][
eeMaxAngle], servo_pwm_min, servo_pwm_max);
1220             pwm.setPWM(servos[ServoNum][eePinNum], 0, pulselength);
1221             delay(msDelay);
1222         }
1223     }
1224 }
1225 }
1226
```