

```

1 #include <tistdtypes.h>
2 #include <coecsl.h>
3 #include "user_includes.h"
4 #include "math.h"
5
6 // These two offsets are only used in the main file user_CRSSRobot.c You just need
  to create them here and find the correct offset and then these offset will adjust
  the encoder readings
7 float offset_Enc2_rad = -0.415; //-0.37;
8 float offset_Enc3_rad = 0.233; //0.27;
9
10 // Your global variables.
11 float px = 0; // x coordinate of end effector
12 float py = 0; // y coordinate of end effector
13 float pz = 0; // z coordinate of end effector
14 float theta1m_IK = 0; // theta1 motor from inverse kinematics
15 float theta2m_IK = 0; // theta2 motor from inverse kinematics
16 float theta3m_IK = 0; // theta3 motor from inverse kinematics
17 long mycount = 0;
18
19 // Variables for Omega calculations
20 float Theta1_old = 0;
21 float Omega1_old1 = 0;
22 float Omega1_old2 = 0;
23 float Omega1 = 0;
24
25 float Theta2_old = 0;
26 float Omega2_old1 = 0;
27 float Omega2_old2 = 0;
28 float Omega2 = 0;
29
30 float Theta3_old = 0;
31 float Omega3_old1 = 0;
32 float Omega3_old2 = 0;
33 float Omega3 = 0;
34
35 //Forward Kinematics
36 float x, y, z;
37 // Variables for Velocity calculations
38 float x_old = 0;
39 float vx_old1 = 0;
40 float vx_old2 = 0;
41 float vx = 0;
42
43 float y_old = 0;
44 float vy_old1 = 0;
45 float vy_old2 = 0;
46 float vy = 0;
47
48 float z_old = 0;
49 float vz_old1 = 0;
50 float vz_old2 = 0;
51 float vz = 0;
52
53 // Constants
54 float dt = 0.001;
55 float threshold1 = 0.01;
56 float threshold2 = 0.06;
57 float threshold3 = 0.02;

```

```

58
59 float theta_dotdot = 0.0;
60
61 float x_desired = 0;
62 float y_desired = 0;
63 float z_desired = 0;
64
65 //Without Mass
66 float p1 = 0.0300;
67 float p2 = 0.0128;
68 float p3 = 0.0076;
69 float p4 = 0.0753;
70 float p5 = 0.0298;
71
72 float a_theta2;
73 float a_theta3;
74
75 float sintheta2;
76 float costheta2;
77 float sintheta3;
78 float costheta3;
79 float g = 9.81;
80
81 float J1 = 0.0167;
82 float J2 = 0.03;
83 float J3 = 0.0128;
84
85 //Minimum velocity, positive and negative viscous coefficients, and positive and
    negative coulomb (static) coefficients for each joint
86 float Viscouspos1=0.17, Viscouspos2=0.23, Viscouspos3=0.1922;
87 float Viscousneg1=0.17, Viscousneg2=0.287, Viscousneg3=0.2132;
88 float Coulombpos1=0.40, Coulombpos2=0.40, Coulombpos3=0.40;
89 float Coulombneg1=-0.35, Coulombneg2=-0.40, Coulombneg3=-0.50;
90 float slope1 = 3.6;
91 float slope2 = 3.6;
92 float slope3 = 3.6;
93 float ff = 1;
94
95 float a0,a1,a2,a3;
96 float tau1cur,tau2cur,tau3cur;
97
98 float desired_1, desired_2, desired_3;
99 float desired_dot_1, desired_dot_2, desired_dot_3;
100 float desired_doubledot_1, desired_doubledot_2, desired_doubledot_3;
101
102 //sin and cos of the three joints
103 float cosq1 = 0;
104 float sinq1 = 0;
105 float cosq2 = 0;
106 float sinq2 = 0;
107 float cosq3 = 0;
108 float sinq3 = 0;
109 //Jacobian Transpose
110 float JT_11 = 0;
111 float JT_12 = 0;
112 float JT_13 = 0;
113 float JT_21 = 0;
114 float JT_22 = 0;
115 float JT_23 = 0;
116 float JT_31 = 0;

```

```

117 float JT_32 = 0;
118 float JT_33 = 0;
119 //sin and cos used for rotation
120 float cosz = 0;
121 float sinz = 0;
122 float cosx = 0;
123 float sinx = 0;
124 float cosy = 0;
125 float siny = 0;
126 //Rotational matrix
127 float R11 = 0;
128 float R12 = 0;
129 float R13 = 0;
130 float R21 = 0;
131 float R22 = 0;
132 float R23 = 0;
133 float R31 = 0;
134 float R32 = 0;
135 float R33 = 0;
136 //Transpose of the rotational matrix
137 float RT11 = 0;
138 float RT12 = 0;
139 float RT13 = 0;
140 float RT21 = 0;
141 float RT22 = 0;
142 float RT23 = 0;
143 float RT31 = 0;
144 float RT32 = 0;
145 float RT33 = 0;
146
147 float thetax = 0;
148 float thetay = 0;
149 float thetaz = 60;
150 //Task Space PD Gains
151 float kpx = 0.5;
152 float kpy = 0.5;
153 float kpz = 0.5;
154 float kdx = 0.025;
155 float kdy = 0.025;
156 float kdz = 0.025;
157
158 //Gravity Compensation in Z direction
159 float fzcnd = 0;
160 float kt = 6;
161 float grav_comp = 5;
162
163 // Gains in N-frame
164 float Kpx_n = 1.3;
165 float Kpy_n = 1.3;
166 float Kpz_n = 1.3;
167 float Kdx_n = 0.03;
168 float Kdy_n = 0.03;
169 float Kdz_n = 0.025;
170
171 //Final Project Variables
172 int mode = 2; //Mode = 1: Impedance Control; Mode = 2: Task Space PD control; Mode =
173 3: Impedance Control 2
174
175 //Desired Trajectory
176 float xd, yd, zd;

```

```

176 float xd_dot = 0;
177 float yd_dot = 0;
178 float zd_dot = 0;
179
180 float a = 0;
181 float b = 0;
182 float c = 0;
183 float d = 0;
184 float t = 0.0;
185
186 float t_total = 0;
187 float deltax = 0;
188 float deltax = 0;
189 float deltaz = 0;
190
191 typedef struct point_tag {
192     float x; // x position
193     float y; // y position
194     float z; // z position
195     float thz; // rotation about z
196     int mode; //Modes for controller
197 } point;
198
199 #define XYZSTIFF 1
200 #define ZSTIFF 2
201 #define XZSTIFF 3
202
203 #define NUM_POINTS 24
204 point point_array[NUM_POINTS] = {
205     { 6.5, 0, 17, 0, XYZSTIFF}, //0 point 0
206     { 10, 0, 20, 0, XYZSTIFF}, //1 point 0 Home Position
207     {7.88, 9.03, 18.86, 0, XYZSTIFF}, //2 point 1
208     {1.57, 13.98, 8, 0, XYZSTIFF}, //3 point 2
209     {1.57, 13.98, 7.1, 0, XYZSTIFF}, //4 point 3 peg hole
210     {1.57, 13.98, 5.00, 0, ZSTIFF}, //5 point 4 x and y weak peg hole
211     {1.57, 13.98, 12, 0, ZSTIFF}, //6 point 5 x and y weak peg hole
212     {7.64, 9.64, 12.34, 0, XYZSTIFF}, //7 point 6 peg hole not used
213     {8.97, 5.11, 13.64, 0, XYZSTIFF}, //8 point 7 avoid obstacle above egg
214     {9, 3.68, 9.7, 0, XYZSTIFF}, //9 point 8 avoid obstacle not used
215     {9.14, 0.82, 9.7, 0, XYZSTIFF}, //10 point 9 avoid obstacle not used
216     {15.33, 3.90, 8.11, 0, XYZSTIFF}, //11 point 10 begin zig zag (Start Point)
217     {16.22, 2.50, 8.11, PI/4, XZSTIFF}, //12 point 11 zig zag Before 1st curve
218     {15.63, 1.80, 8.11, -PI/4, XZSTIFF}, //13 point 12 zig zag After 1st curve
219     {13.04, 2.06, 8.11, PI/4, XZSTIFF}, //14 point 13 zig zag
220     {12.70, 1.11, 8.11, 0, XYZSTIFF}, //15 point 14 up
221     {15.21, -1.98, 8.11, 0, XYZSTIFF}, //16 point 15 top egg
222     {15.21, -1.98, 10.64, 0, XYZSTIFF}, //17 point 16 push egg
223     {9.73, 5.52, 12.64, 0, XYZSTIFF}, //18 point 17 push egg
224     {9.73, 5.52, 11.11, 0, XYZSTIFF}, //19 point 18 top egg
225     {10, 0, 20, 0, XYZSTIFF}, //20 point 19
226     { 6.5, 0, 17, 0, XYZSTIFF}, // point 0 not used
227     { 6.5, 0, 17, 0, XYZSTIFF}, // point 0 not used
228     { 6.5, 0, 17, 0, XYZSTIFF}, // point 0 not used
229 };
230
231
232 #pragma DATA_SECTION(whattoprint, ".my_vars")
233 float whattoprint = 0.0;
234
235 #pragma DATA_SECTION(whatnottoprint, ".my_vars")

```

```

236 float whatnottoprint = 0.0;
237
238 #pragma DATA_SECTION(theta1array, ".my_arrs")
239 float theta1array[100];
240
241 #pragma DATA_SECTION(theta2array, ".my_arrs")
242 float theta2array[100];
243
244 long arrayindex = 0;
245
246 float printtheta1motor = 0;
247 float printtheta2motor = 0;
248 float printtheta3motor = 0;
249
250 // Assign these float to the values you would like to plot in Simulink
251 float Simulink_PlotVar1 = 0;
252 float Simulink_PlotVar2 = 0;
253 float Simulink_PlotVar3 = 0;
254 float Simulink_PlotVar4 = 0;
255
256 void inverseKinematics(float px, float py, float pz) {
257     // Calculate DH angles from end effector position
258     float theta1 = atan2(py, px); // DH theta 1
259     float z = pz - 10;
260     float beta = sqrt(px*px + py*py);
261     float L = sqrt(z*z + beta*beta);
262     float theta3 = acos((L*L - 200)/200); // DH theta 3
263     float theta2 = -theta3/2 - atan2(z, beta); // DH theta 2
264
265     // Convert DH angles to motor angles
266     theta1m_IK = theta1;
267     theta2m_IK = (theta2 + PI/2);
268     theta3m_IK = (theta3 + theta2m_IK - PI/2);
269 }
270
271 // Omega estimation
272 void omega(float theta1motor, float theta2motor, float theta3motor) {
273     Omega1 = (theta1motor - Theta1_old)/0.001;
274     Omega1 = (Omega1 + Omega1_old1 + Omega1_old2)/3.0;
275
276     Theta1_old = theta1motor;
277
278     Omega1_old2 = Omega1_old1;
279     Omega1_old1 = Omega1;
280
281     Omega2 = (theta2motor - Theta2_old)/0.001;
282     Omega2 = (Omega2 + Omega2_old1 + Omega2_old2)/3.0;
283
284     Theta2_old = theta2motor;
285
286     Omega2_old2 = Omega2_old1;
287     Omega2_old1 = Omega2;
288
289     Omega3 = (theta3motor - Theta3_old)/0.001;
290     Omega3 = (Omega3 + Omega3_old1 + Omega3_old2)/3.0;
291
292     Theta3_old = theta3motor;
293
294     Omega3_old2 = Omega3_old1;
295

```

```

296     Omega3_old1 = Omega3;
297 }
298
299 //Velocity Updates
300 void velocity(float x, float y, float z) {
301     vx = (x - x_old)/0.001;
302     vx = (vx + vx_old1 + vx_old2)/3.0;
303
304     x_old = x;
305
306     vx_old2 = vx_old1;
307     vx_old1 = vx;
308
309     vy = (y - y_old)/0.001;
310     vy = (vy + vy_old1 + vy_old2)/3.0;
311
312     y_old = y;
313
314     vy_old2 = vy_old1;
315     vy_old1 = vy;
316
317     vz = (z - z_old)/0.001;
318     vz = (vz + vz_old1 + vz_old2)/3.0;
319
320     z_old = z;
321
322     vz_old2 = vz_old1;
323     vz_old1 = vz;
324 }
325
326 // Generation cubic trajectory
327 //This function is used to calculate the coefficients required to perform a cubic
    trajectory. The cubic function ranges from times t to t_f, and from positions p_0 to
    p_1.
328 //Source: Robot Modeling and Control by Spong, Hutchinson, Vidyasagar. Page 175-176.
329
330 float cubic2points(float t, float t_f, float p_0, float p_1){
331     a = 2*(p_0 - p_1)/(t_f*t_f*t_f);
332     b = -3*(p_0 - p_1)/(t_f*t_f);
333     d = p_0;
334     return (a*t*t*t + b*t*t + d);
335 }
336
337 void lab(float theta1motor,float theta2motor,float theta3motor,float *tau1,float
    *tau2,float *tau3, int error) {
338     //Forward Kinematics
339     x = 10*cos(theta1motor)*(cos(theta3motor) + sin(theta2motor));
340     y = 10*sin(theta1motor)*(cos(theta3motor) + sin(theta2motor));
341     z = 10*cos(theta2motor) - 10*sin(theta3motor) + 10;
342
343     //Coefficients for different time steps
344     // trajectory
345     t = (mycount%200000)/1000.;
346
347     float t_0 = 0.75; //Homing
348     float t_1 = t_0 + 1.0; //Align to the Hole
349     float t_2 = t_1 + 1.0; // Going into the Hole
350     float t_3 = t_2 + 1.0; //From inside hole to outside hole
351     float t_4 = t_3 + 1.0; //Outside hole to above egg
352     float t_5 = t_4 + 1.0; //Above egg to entrance of Zigzag

```

```

353 float t_6 = t_5 + 0.65; //1st Straight line from point A to B
354 float t_7 = t_6 + 0.65; //1st Curve from point A to B
355 float t_8 = t_7 + 1.25; //2nd Straight line from point A to B
356 float t_9 = t_8 + 0.55; //2nd Curve from point A to B
357 float t_10 = t_9 + 0.4; //3rd Straight line from point A to exit
358 float t_11 = t_10 + 0.4; //Exit to Up
359 float t_12 = t_11 + 0.4; //Up to top of egg
360 float t_13 = t_12 + 0.8; //PUSH!!!!
361 float t_14 = t_13 + 2.0; //keep pushing!!!!
362 float t_15 = t_14 + 0.7; //After egg go back to home position
363
364 if (t <= t_0){ // Homing
365     xd = cubic2points(t, t_0 , point_array[0].x, point_array[1].x);
366     yd = point_array[0].y;
367     zd = cubic2points(t, t_0 , point_array[0].z, point_array[1].z);
368     mode = 2;
369 }
370 else if (t_0<t && t<=t_1){ //Aligning to the Hole
371     xd = cubic2points(t-t_0, t_1-t_0 , point_array[1].x, point_array[3].x);
372     yd = cubic2points(t-t_0, t_1-t_0 , point_array[1].y, point_array[3].y);
373     zd = cubic2points(t-t_0, t_1-t_0 , point_array[1].z, point_array[3].z);
374     mode = 2;
375 }
376 else if (t_1<t && t<=t_2/2){ //Going into the Hole
377     xd = cubic2points(t-t_1, t_2/2-t_1, point_array[3].x, point_array[5].x);
378     yd = cubic2points(t-t_1, t_2/2-t_1 , point_array[3].y, point_array[5].y);
379     zd = cubic2points(t-t_1, t_2/2-t_1 , point_array[3].z, point_array[5].z);
380     mode = 3;
381 }
382 else if (t_2/2<t && t<=t_2){ //Staying inside Hole
383     xd = cubic2points(t-t_2/2, t_2-t_2/2, point_array[5].x, point_array[5].x);
384     yd = cubic2points(t-t_2/2, t_2-t_2/2 , point_array[5].y, point_array[5].y);
385     zd = cubic2points(t-t_2/2, t_2-t_2/2 , point_array[5].z, point_array[5].z);
386     mode = 3;
387 }
388 else if (t_2<t && t<=t_3){ //From inside hole to outside hole
389     xd = cubic2points(t-t_2, t_3-t_2 , point_array[5].x, point_array[6].x);
390     yd = cubic2points(t-t_2, t_3-t_2 , point_array[5].y, point_array[6].y);
391     zd = cubic2points(t-t_2, t_3-t_2 , point_array[5].z, point_array[6].z);
392     mode = 3;
393 }
394 else if (t_3<t && t<=t_4){ //Outside hole to above egg
395     t_total = t_4 - t_3;
396     deltax = point_array[8].x - point_array[6].x;
397     deltay = point_array[8].y - point_array[6].y;
398     deltaz = point_array[8].z - point_array[6].z;
399     xd = point_array[6].x + deltax*(t-t_3)/t_total;
400     yd = point_array[6].y + deltay*(t-t_3)/t_total;
401     zd = point_array[6].z + deltaz*(t-t_3)/t_total;
402     mode = 3;
403 }
404 else if (t_4<t && t<=t_5){ //Above egg to entrance of Zigzag
405     xd = cubic2points(t-t_4, t_5-t_4 , point_array[8].x, point_array[11].x);
406     yd = cubic2points(t-t_4, t_5-t_4 , point_array[8].y, point_array[11].y);
407     zd = cubic2points(t-t_4, t_5-t_4 , point_array[8].z, point_array[11].z);
408     mode = 2;
409 }
410 else if (t_5<t && t<=t_6){ //1st Straight line from point A to B
411     t_total = t_6 - t_5;
412     deltax = point_array[12].x - point_array[11].x;

```



```

413     deltay = point_array[12].y - point_array[11].y;
414     deltaz = point_array[12].z - point_array[11].z;
415     xd = point_array[11].x + deltax*(t-t_5)/t_total;
416     yd = point_array[11].y + deltay*(t-t_5)/t_total;
417     zd = point_array[11].z + deltaz*(t-t_5)/t_total;
418     mode = 1;
419 }
420 else if (t_6<t && t<=t_7){ //1st Curve from point A to B
421     xd = cubic2points(t-t_6, t_7-t_6 , point_array[12].x, point_array[13].x);
422     yd = cubic2points(t-t_6, t_7-t_6 , point_array[12].y, point_array[13].y);
423     zd = cubic2points(t-t_6, t_7-t_6 , point_array[12].z, point_array[13].z);
424     mode = 1;
425 }
426 else if (t_7<t && t<=t_8){ //2nd Straight line from point A to B
427     t_total = t_8 - t_7;
428     deltax = point_array[14].x - point_array[13].x;
429     deltay = point_array[14].y - point_array[13].y;
430     deltaz = point_array[14].z - point_array[13].z;
431     xd = point_array[13].x + deltax*(t-t_7)/t_total;
432     yd = point_array[13].y + deltay*(t-t_7)/t_total;
433     zd = point_array[13].z + deltaz*(t-t_7)/t_total;
434     mode = 1;
435 }
436 else if (t_8<t && t<=t_9){ //2nd Curve from point A to B
437     xd = cubic2points(t-t_8, t_9-t_8 , point_array[14].x, point_array[15].x);
438     yd = cubic2points(t-t_8, t_9-t_8 , point_array[14].y, point_array[15].y);
439     zd = cubic2points(t-t_8, t_9-t_8 , point_array[14].z, point_array[15].z);
440     mode = 1;
441 }
442 else if (t_9<t && t<=t_10){ //3rd Straight line from point A to exit
443     t_total = t_10 - t_9;
444     deltax = point_array[16].x - point_array[15].x;
445     deltay = point_array[16].y - point_array[15].y;
446     deltaz = point_array[16].z - point_array[15].z;
447     xd = point_array[15].x + deltax*(t-t_9)/t_total;
448     yd = point_array[15].y + deltay*(t-t_9)/t_total;
449     zd = point_array[15].z + deltaz*(t-t_9)/t_total;
450     mode = 1;
451 }
452 else if (t_10<t && t<=t_11){ //Exit to Up
453     t_total = t_11 - t_10;
454     deltax = point_array[17].x - point_array[16].x;
455     deltay = point_array[17].y - point_array[16].y;
456     deltaz = point_array[17].z - point_array[16].z;
457     xd = point_array[16].x + deltax*(t-t_10)/t_total;
458     yd = point_array[16].y + deltay*(t-t_10)/t_total;
459     zd = point_array[16].z + deltaz*(t-t_10)/t_total;
460     mode = 2;
461 }
462 else if (t_11<t && t<=t_12){ //Up to top of egg
463     xd = cubic2points(t-t_11, t_12-t_11 , point_array[17].x, point_array[18].x);
464     yd = cubic2points(t-t_11, t_12-t_11 , point_array[17].y, point_array[18].y);
465     zd = cubic2points(t-t_11, t_12-t_11 , point_array[17].z, point_array[18].z);
466     mode = 2;
467 }
468 else if (t_12<t && t<=t_13){ //PUSH!!!!
469     xd = cubic2points(t-t_12, t_13-t_12 , point_array[18].x, point_array[19].x);
470     yd = cubic2points(t-t_12, t_13-t_12 , point_array[18].y, point_array[19].y);
471     zd = cubic2points(t-t_12, t_13-t_12 , point_array[18].z, point_array[19].z);
472     mode = 3;

```



```

473         }
474
475     else if (t_13<t && t<=t_14){ //keep pushing!!!!
476         xd = cubic2points(t-t_13, t_14-t_13 , point_array[19].x, point_array[19].x);
477         yd = cubic2points(t-t_13, t_14-t_13 , point_array[19].y, point_array[19].y);
478         zd = cubic2points(t-t_13, t_14-t_13 , point_array[19].z, point_array[19].z);
479         mode = 3;
480     }
481
482     else if (t_14<t && t<=t_15){ //After egg go back to home position
483         xd = cubic2points(t-t_14, t_15-t_14 , point_array[19].x, point_array[20].x);
484         yd = cubic2points(t-t_14, t_15-t_14 , point_array[19].y, point_array[20].y);
485         zd = cubic2points(t-t_14, t_15-t_14 , point_array[19].z, point_array[20].z);
486         mode = 3;
487     }
488     else{
489         xd = point_array[20].x;
490         yd = point_array[20].y;
491         zd = point_array[20].z;
492     }
493     // Calculate/Update omegas/velocities
494     omega(theta1motor, theta2motor, theta3motor);
495     velocity(x, y, z);
496
497     // position error
498     float x_error = xd - x;
499     float y_error = yd - y;
500     float z_error = zd - z;
501     float xd_error = xd_dot - vx;
502     float yd_error = yd_dot - vy;
503     float zd_error = zd_dot - vz;
504
505     if(mode == 1){ //Zigzag mode
506         // Gains in N-frame
507         Kpx_n = 1.0;
508         Kpy_n = 1.0;
509         Kpz_n = 1.3;
510         Kdx_n = 0.02;
511         Kdy_n = 0.02;
512         Kdz_n = 0.025;
513         cosq1 = cos(theta1motor);
514         sinq1 = sin(theta1motor);
515         cosq2 = cos(theta2motor);
516         sinq2 = sin(theta2motor);
517         cosq3 = cos(theta3motor);
518         sinq3 = sin(theta3motor);
519         JT_11 = -10*sinq1*(cosq3 + sinq2);
520         JT_12 = 10*cosq1*(cosq3 + sinq2);
521         JT_13 = 0;
522         JT_21 = 10*cosq1*(cosq2 - sinq3);
523         JT_22 = 10*sinq1*(cosq2 - sinq3);
524         JT_23 = -10*(cosq3 + sinq2);
525         JT_31 = -10*cosq1*sinq3;
526         JT_32 = -10*sinq1*sinq3;
527         JT_33 = -10*cosq3;
528
529         cosz = cos(thetaz);
530         sinz = sin(thetaz);
531         cosx = cos(thetax);
532         sinx = sin(thetax);

```

```

533     cosy = cos(thetay);
534     siny = sin(thetay);
535
536     RT11 = R11 = cosz*cosy-sinz*sinx*siny;
537     RT21 = R12 = -sinz*cosx;
538     RT31 = R13 = cosz*siny+sinz*sinx*cosy;
539     RT12 = R21 = sinz*cosy+cosz*sinx*siny;
540     RT22 = R22 = cosz*cosx;
541     RT32 = R23 = sinz*siny-cosz*sinx*cosy;
542     RT13 = R31 = -cosx*siny;
543     RT23 = R32 = sinx;
544     RT33 = R33 = cosx*cosy;
545
546     *tau1 = (JT_11*R11 + JT_12*R21 + JT_13*R31)*(Kdx_n*R11*xd_error +
Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error +
Kpx_n*R31*z_error) + (JT_11*R12 + JT_12*R22 + JT_13*R32)*(Kdy_n*R12*xd_error +
Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error +
Kpy_n*R32*z_error) + (JT_11*R13 + JT_12*R23 + JT_13*R33)*(Kdz_n*R13*xd_error +
Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error +
Kpz_n*R33*z_error);
547     *tau2 = (JT_21*R11 + JT_22*R21 + JT_23*R31)*(Kdx_n*R11*xd_error +
Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error +
Kpx_n*R31*z_error) + (JT_21*R12 + JT_22*R22 + JT_23*R32)*(Kdy_n*R12*xd_error +
Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error +
Kpy_n*R32*z_error) + (JT_21*R13 + JT_22*R23 + JT_23*R33)*(Kdz_n*R13*xd_error +
Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error +
Kpz_n*R33*z_error);
548     *tau3 = (JT_31*R11 + JT_32*R21 + JT_33*R31)*(Kdx_n*R11*xd_error +
Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error +
Kpx_n*R31*z_error) + (JT_31*R12 + JT_32*R22 + JT_33*R32)*(Kdy_n*R12*xd_error +
Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error +
Kpy_n*R32*z_error) + (JT_31*R13 + JT_32*R23 + JT_33*R33)*(Kdz_n*R13*xd_error +
Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error +
Kpz_n*R33*z_error);
549     }
550     else if(mode == 2){//Task Space PD controller
551
552         float fx = kpx*(xd - x) + kdx*(xd_dot - vx);
553         float fy = kpy*(yd - y) + kdy*(yd_dot - vy);
554         float fz = kpz*(zd - z) + kdz*(zd_dot - vz);
555
556         //Compute Jacobian Transpose
557         // Jacobian Transpose
558         cosq1 = cos(theta1motor);
559         sinq1 = sin(theta1motor);
560         cosq2 = cos(theta2motor);
561         sinq2 = sin(theta2motor);
562         cosq3 = cos(theta3motor);
563         sinq3 = sin(theta3motor);
564         JT_11 = -10*sinq1*(cosq3 + sinq2);
565         JT_12 = 10*cosq1*(cosq3 + sinq2);
566         JT_13 = 0;
567         JT_21 = 10*cosq1*(cosq2 - sinq3);
568         JT_22 = 10*sinq1*(cosq2 - sinq3);
569         JT_23 = -10*(cosq3 + sinq2);
570         JT_31 = -10*cosq1*sinq3;
571         JT_32 = -10*sinq1*sinq3;
572         JT_33 = -10*cosq3;
573         //
574         float x_grav_comp = 0;

```

```

575 float y_grav_comp = 0.0254*JT_23*fzcmd/kt;
576 float z_grav_comp = 0.0254*JT_33*(fzcmd + grav_comp)/kt;
577 *tau1 = JT_11*fx + JT_12*fy + x_grav_comp;
578 *tau2 = JT_21*fx + JT_22*fy + JT_23*fz + y_grav_comp;
579 *tau3 = JT_31*fx + JT_32*fy + JT_33*fz + z_grav_comp;
580
581 }
582
583 else if(mode == 3){//Simple Impedance Control
584     Kpx_n = 1.0;
585     Kpy_n = 1.0;
586     Kpz_n = 1.3;
587     Kdx_n = 0.03;
588     Kdy_n = 0.03;
589     Kdz_n = 0.025;
590     //Compute Jacobian Transpose
591     // Jacobian Transpose
592     cosq1 = cos(theta1motor);
593     sinq1 = sin(theta1motor);
594     cosq2 = cos(theta2motor);
595     sinq2 = sin(theta2motor);
596     cosq3 = cos(theta3motor);
597     sinq3 = sin(theta3motor);
598     JT_11 = -10*sinq1*(cosq3 + sinq2);
599     JT_12 = 10*cosq1*(cosq3 + sinq2);
600     JT_13 = 0;
601     JT_21 = 10*cosq1*(cosq2 - sinq3);
602     JT_22 = 10*sinq1*(cosq2 - sinq3);
603     JT_23 = -10*(cosq3 + sinq2);
604     JT_31 = -10*cosq1*sinq3;
605     JT_32 = -10*sinq1*sinq3;
606     JT_33 = -10*cosq3;
607
608     cosz = cos(thetaz);
609     sinz = sin(thetaz);
610     cosx = cos(thetax);
611     sinx = sin(thetax);
612     cosy = cos(thetay);
613     siny = sin(thetay);
614
615     RT11 = R11 = cosz*cosy-sinz*sinx*siny;
616     RT21 = R12 = -sinz*cosx;
617     RT31 = R13 = cosz*siny+sinz*sinx*cosy;
618     RT12 = R21 = sinz*cosy+cosz*sinx*siny;
619     RT22 = R22 = cosz*cosx;
620     RT32 = R23 = sinz*siny-cosz*sinx*cosy;
621     RT13 = R31 = -cosx*siny;
622     RT23 = R32 = sinx;
623     RT33 = R33 = cosx*cosy;
624
625     *tau1 = (JT_11*R11 + JT_12*R21 + JT_13*R31)*(Kdx_n*R11*xd_error +
Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error +
Kpx_n*R31*z_error) + (JT_11*R12 + JT_12*R22 + JT_13*R32)*(Kdy_n*R12*xd_error +
Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error +
Kpy_n*R32*z_error) + (JT_11*R13 + JT_12*R23 + JT_13*R33)*(Kdz_n*R13*xd_error +
Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error +
Kpz_n*R33*z_error);
626     *tau2 = (JT_21*R11 + JT_22*R21 + JT_23*R31)*(Kdx_n*R11*xd_error +
Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error +
Kpx_n*R31*z_error) + (JT_21*R12 + JT_22*R22 + JT_23*R32)*(Kdy_n*R12*xd_error +

```

```

Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error +
Kpy_n*R32*z_error) + (JT_21*R13 + JT_22*R23 + JT_23*R33)*(Kdz_n*R13*xd_error +
Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error +
Kpz_n*R33*z_error);
627     *tau3 = (JT_31*R11 + JT_32*R21 + JT_33*R31)*(Kdx_n*R11*xd_error +
Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error +
Kpx_n*R31*z_error) + (JT_31*R12 + JT_32*R22 + JT_33*R32)*(Kdy_n*R12*xd_error +
Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error +
Kpy_n*R32*z_error) + (JT_31*R13 + JT_32*R23 + JT_33*R33)*(Kdz_n*R13*xd_error +
Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error +
Kpz_n*R33*z_error);
628 }//
629
630
631 // Friction Compensation
632 //If omega greater than 0.1 (Max Velocity) for joint 1
633 if (Omega1 > 0.1) {
634     *tau1 = *tau1 + 0.6*(Viscouspos1 * Omega1 + Coulombpos1) ;
635 }
636 //If omega greater than -0.1 (Min Velocity) for joint 1
637 else if (Omega1 < -0.1) {
638     *tau1 = *tau1 + 0.6*(Viscousneg1 * Omega1 + Coulombneg1);
639 }
640 //If omega between min and max velocity for joint 1
641 else {
642     *tau1 = *tau1 + 0.6*(slope1*Omega1);
643 }
644
645 //If omega greater than 0.05(Max Velocity) for joint 2
646 if (Omega2 > 0.05) {
647     *tau2 = *tau2 + 0.6*(Viscouspos2 * Omega2 + Coulombpos2);
648 }
649 //If omega greater than -0.05 (Min Velocity) for joint 2
650 else if (Omega2 < -0.05) {
651     *tau2 = *tau2 + 0.6*(Viscousneg2 * Omega2 + Coulombneg2);
652 }
653 //If omega between min and max velocity for joint 2
654 else {
655     *tau2 = *tau2 + 0.6*(slope2*Omega2);
656 }
657 //If omega greater than -0.05 (Max Velocity) for joint 3
658 if (Omega3 > 0.05) {
659     *tau2 = *tau2 + 0.6*(Viscouspos3 * Omega3 + Coulombpos3) ;
660 }
661 //If omega greater than -0.05 (Min Velocity) for joint 3
662 else if (Omega3 < -0.05) {
663     *tau2 = *tau2 + 0.6*(Viscousneg3 * Omega3 + Coulombneg3);
664 }
665 //If omega between min and max velocity for joint 3
666 else {
667     *tau2 = *tau2 + 0.6*(slope3*Omega3);
668 }
669
670
671 //Limits torque to 5N-m
672 if (*tau1 >= 5) {
673     *tau1 = 5;
674 } else if (*tau1 < -5) {
675     *tau1 = -5;
676 }

```

```

677
678     if (*tau2 >= 5) {
679         *tau2 = 5;
680     } else if (*tau2 < -5) {
681         *tau2 = -5;
682     }
683
684     if (*tau3 >= 5) {
685         *tau3 = 5;
686     } else if (*tau3 < -5) {
687         *tau3 = -5;
688     }
689
690
691     // save past states
692     if ((mycount%50)==0) {
693
694         theta1array[arrayindex] = theta1motor;
695         theta2array[arrayindex] = theta2motor;
696
697         if (arrayindex >= 100) {
698             arrayindex = 0;
699         } else {
700             arrayindex++;
701         }
702     }
703
704
705     if ((mycount%500)==0) {
706         if (whattoprint > 0.5) {
707             serial_printf(&SerialA, "I love robotics\n\r");
708         } else {
709             printtheta1motor = theta1motor;
710             printtheta2motor = theta2motor;
711             printtheta3motor = theta3motor;
712             SWI_post(&SWI_printf); //Using a SWI to fix SPI issue from sending too
many floats.
713         }
714
715         GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1; // Blink LED on Control Card
716         GpioDataRegs.GPBTOGGLE.bit.GPIO60 = 1; // Blink LED on Emergency Stop Box
717     }
718
719
720
721     Simulink_PlotVar1 = desired_1; //yellow
722     Simulink_PlotVar2 = theta1motor; //blue
723     Simulink_PlotVar3 = theta2motor; //orange
724     Simulink_PlotVar4 = theta3motor; //green
725     mycount++;
726 }
727
728 void printing(void){
729     // Printing (theta1, theta2, theta3, , py, pz) and then on a new line
(theta1_IK, theta2_IK, theta3_IK)
730     serial_printf(&SerialA, "px: %.2f, py: %.2f, pz: %.2f \n\r", x, y, z);
731 }
732

```