Tess Callan, Jorell Gotamco, Tachpol (Pond) Posaphiwat

Professor Hannah Stuart

ME102B

12 December 2021

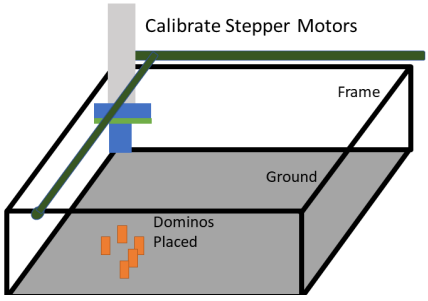
The Domino Placer Manual

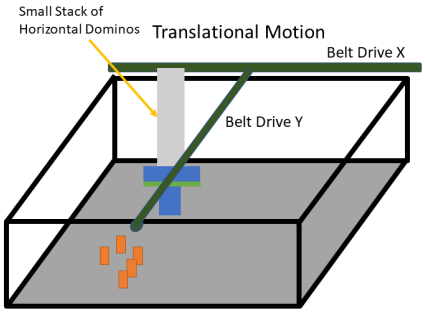
Opportunity

There are a lot of applications for a mechanical actuator to perform repetitive tasks, and our project seeks to demonstrate that more complex repetitive tasks can also be performed. The purpose of this domino placing project is to demonstrate the potential of this process: a seemingly difficult task for a human hand can be automated by a sophisticated mechanical actuator. Although the domino placer is not exactly a useful machine, all it takes is replacing the placer with another end effector like a robot arm or printhead to create a useful 3-axis machine.

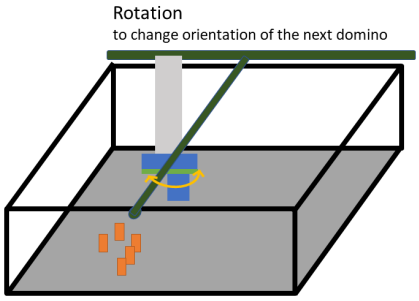
High Level Strategy

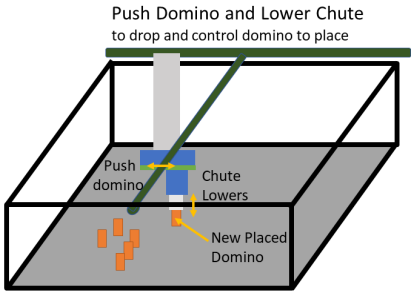
The automatic placing is confined to a frame of 2.5 ft by 2 ft. Two belt drives allow for X and Y translational motion of the placer. First, the stepper motors driving the belts are calibrated by moving to two limit switches at the corner of the frame. This location is defined as origin, allowing for accurate placement of the dominos.



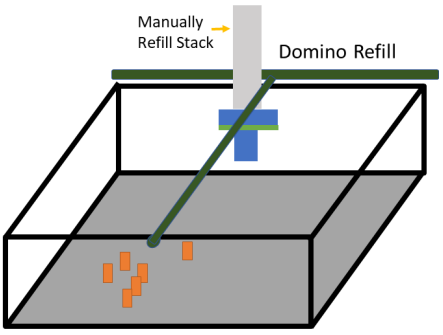


Next, the placer moves to the first domino location. The domino is then rotated using a servo motor and a rotating plate. The domino is then pushed into a lowered chute to ensure the domino is placed upright.



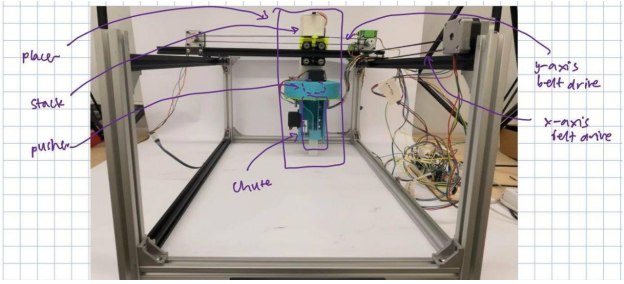


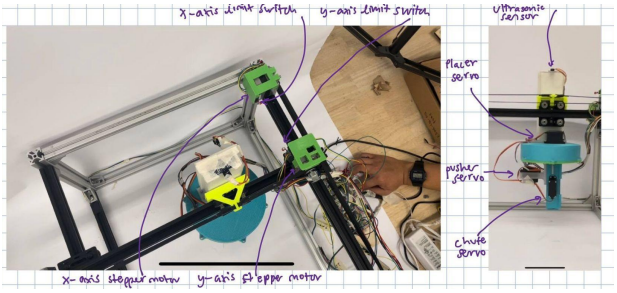
If the dominos run out, the machine goes to a refill state. The stack moves to the frame's side and waits for dominos to be added. This is determined by the distance measured by an ultrasonic sensor at the top of the stack. Once refilled, the user pushes a button to continue placing. In the event of an error, the emergency stop can be pressed to stop the machine.



We were able to reach our desired specifications. The machine places dominos from a stack at desired locations that allow the dominos to hit each other when one is pushed over in the end. Ideally, domino placing would go faster. This could be done by shortening the timer between placing steps. The chute needs a longer time to operate because the domino bounces upon dropping. We found that the chute needs to be down for two seconds to hold the domino while the rotation and pushing can happen in one second.

Final Product





Calculations

Belt Drive Pre-tension

- Stepper motor torque: $\tau_{stepper}$ = 20 N·cm
- Pulley diameter: d_{pulley} = 10 cm

$$F_{pre} = 2(\frac{\tau_{stepper}}{d_{pulley}})$$
$$F_{pre} = 4N$$

Belt Drive Friction

- Carriage and placer mass: $_{carriage}$ = 500 g = 0.5 kg
- Pulley radius: r_{pulley} = 5 cm = 0.05 m
- Coefficient of Friction between carriage and frame: μ = 0.2

$$\tau = r_{pulley}F_{friction} = r_{pulley}m_{carriage}g\mu$$
$$\tau = (0.05\text{m})\,(0.5\text{kg})\,(9.81\text{m/s}^2)\,(0.2)$$
$$\tau = 0.005981N \cdot m$$

The torque required by the stepper motor to move the belt drive with the carriage is 0.005981 $N \cdot m$, which is significantly lower than the 0.2 $N \cdot m$ that the stepper motor is rated for.

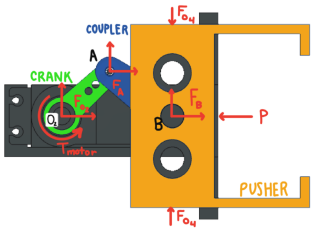
Rotation

- Weight of a domino stack at its maximum: m_{stack} = 1lb = 0.453592 kg
- Effective radius of domino location on rotating plate: r_{domino} = 0.5132 in = 0.013035 m
- Coefficient of Friction between domino and gear(PLA): μ = 0.4

$$\tau = r_{domino}F_{friction} = r_{domino}m_{stack}g\mu$$
$$\tau = (0.013035\text{m})\,(0.453594\text{kg})\,(9.81\text{m/s}^2)\,(0.4)$$
$$\tau = 0.0232N \cdot m$$

The stall torque of the servo motor is 12kg·cm at 6V and 13kg·cm at 7.2V. We are running the motor (within operation voltage) at 5V. This means the stall torque may be near 9 kg·cm, which converts to 0.8826 N·m. This is larger than the expected torque of 0.0232 N·m. Even with any unexpected torque due to friction in the ball bearings on the bottom of the rotating plate or bumps on the top of the rotating plate from imperfections of a 3d print, this motor should be able to handle the torque. These expected imperfections are why we oversized the motors.

Pusher



- Weight of a domino stack at its maximum: $m_{stack} = 1\text{lb} = 0.453592$ kg
- Coefficient of Friction between domino and gear(PLA): $\mu = 0.4$
- Length of pusher crank = 0.55 in = 0.014 m
- Length of pusher coupler = 1 in = 0.254 m

Crank: $F_{Ax} + F_{O2x} = 0$, $F_{Ay} + F_{O2y} = 0$, and $-T_{motor} - F_{Ax}r_2sin(\theta_2) - F_{Ay}r_2cos(\theta_2) = 0$
Coupler: $-F_{Ax} - F_{Bx} = 0$, $-F_{Ay} - F_{By} = 0$, and $-F_{By}r_3cos(\theta_3) - F_{Bx}r_3sin(\theta_3) = 0$
Pusher: $F_{Bx} = P$ and $F_{By} = F_{O4}$

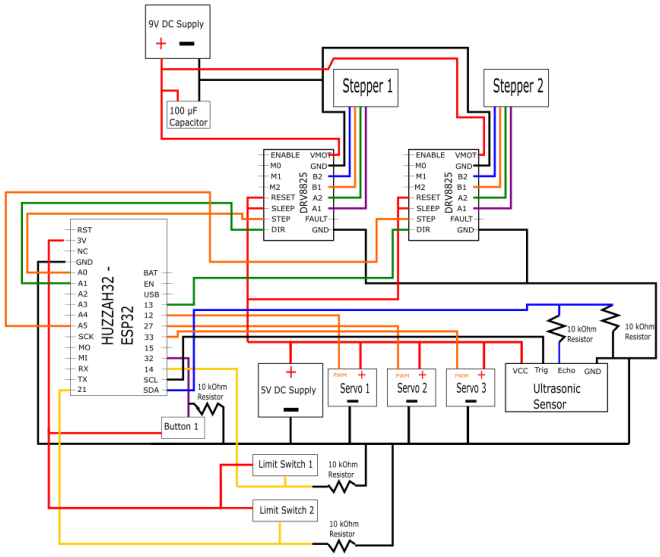
By using the system of static equilibrium equations above for each link in the pusher mechanism, the torque at the motor can be calculated given the force P, which is equal to $m_{stack}g\mu = 0.3238N$. The motor torque required to push a domino from the stack was evaluated to 0.0248 N·m, which is significantly lower than the 0.8826 N·m the motor is rated for.

Chute

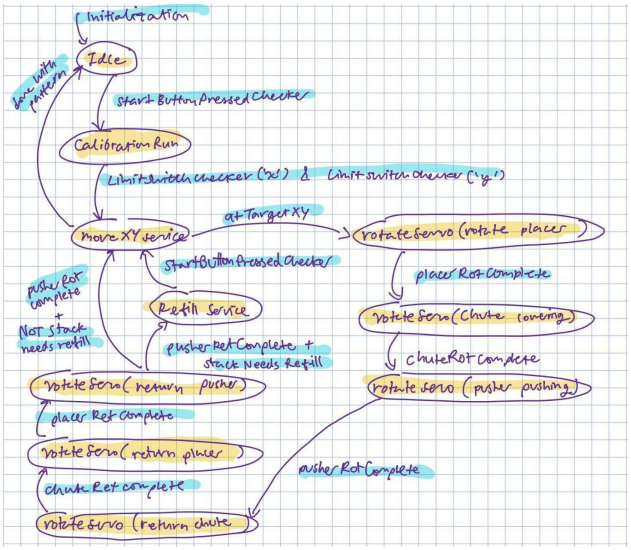
- Weight of a chute: $m_{chute} = 0.0482$ kg
- Length of pusher crank = 0.55 in = 0.014 m
- Length of pusher coupler = 1 in = 0.254 m

Using the same method as for the pusher linkage, the motor torque required to raise and lower was evaluated to 0.0066 Nm, which is significantly lower than the 0.8826 Nm the motor is rated for.

Circuit Diagram



State Transition Diagram



Reflection

During the course of this class, we were able to move the project along nicely by starting early on the complicated mechanical components and breaking the projects up into smaller components. This gave each of us more flexibility for when we had time to work on the project, especially during the middle of the semester project work. Also, being in the same lab session gave us the ability to have a set time to work as a group throughout the semester. While we started the design and construction of the project early, we started the written deliverables near the deadlines. Working earlier on the reports would have allowed us to spend more time on the details and get feedback earlier.

Appendix

Future Work

Design Changes

The most important change needed to be made is to the pusher and chute mechanisms. While the slider-crank mechanisms worked for the prototype, they are too unreliable for a final product. For example, the rail on the lowering chute is currently just a bolt sliding in a 3D printed slot. This system allows for a lot of slack and tends to jam when not aligned properly. A well made design would use a system like linear rails to ensure smooth travel. The slider-crank mechanism controlled by a servo could be replaced with a linear actuator and linear rails, which would jam much less often.

Another change to be made is to the belt drive assembly. In the current setup, the stepper motors are cantilevered, which puts undue stress on the motor. By separating the motor from the belt drive and connecting to the belt drive pulley by an intermediate gear, the motor can avoid the belt tension forces, while still transmitting the forces.

Manufacturing Changes

The belt drive setup could use a little more sturdy of an assembly. The belts are currently being taped together to connect to the carriages. Using zip ties or staples would strengthen this connection and prevent any slipping.

The bottom of the frame would also benefit from a floor being added. This would allow a flat surface for the dominos to be placed on. Feet could also be added to the frame to allow leveling.

Cleaning the wiring and creating an enclosure for the electronics would greatly improve the aesthetics of the machine.

Software Changes

A major addition to the project would be to create a program that could take a user submitted line drawing and turn it into a domino pattern for the placer to recreate. This program was excluded from the project in order to focus on the mechanical operation of the machine, but this would be a great addition to the project that would greatly increase the user experience.

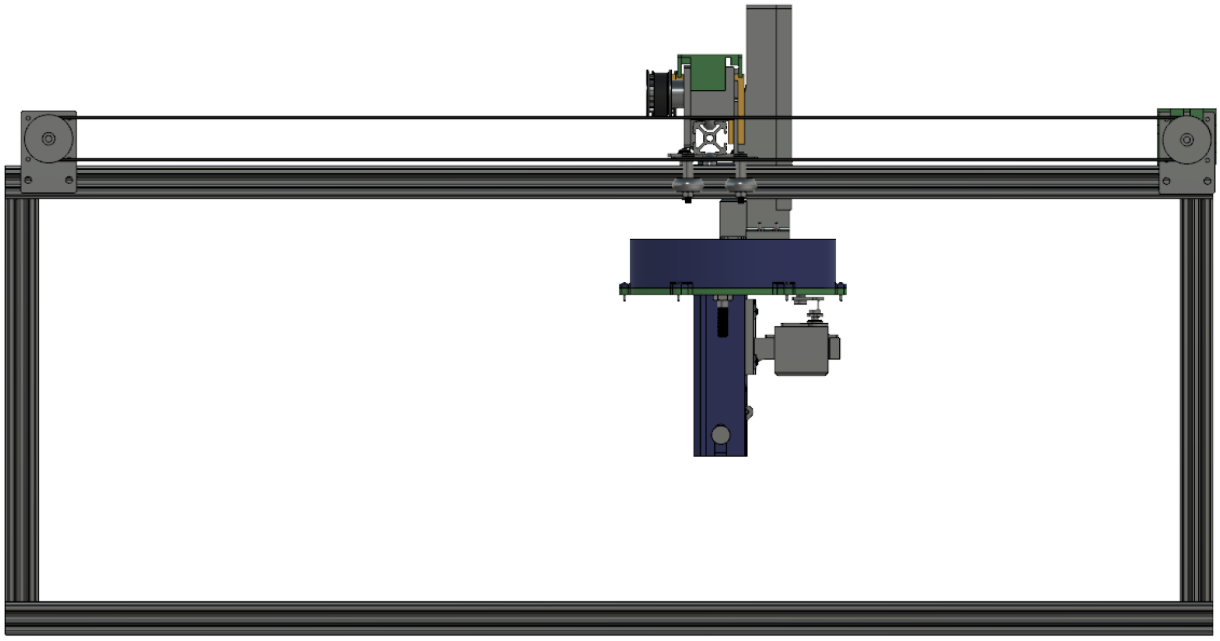
Bill of Materials

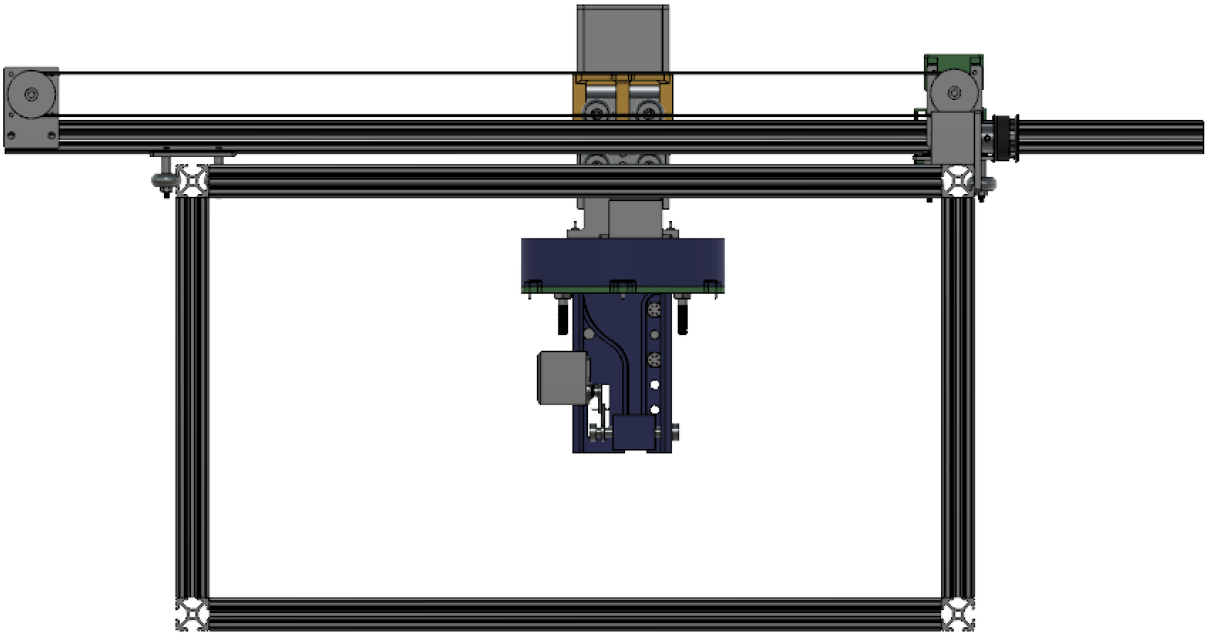
Note: All of the bolts were previously owned or purchased from the hardware store. Links to mcmaster show possible sources for the item in the future.

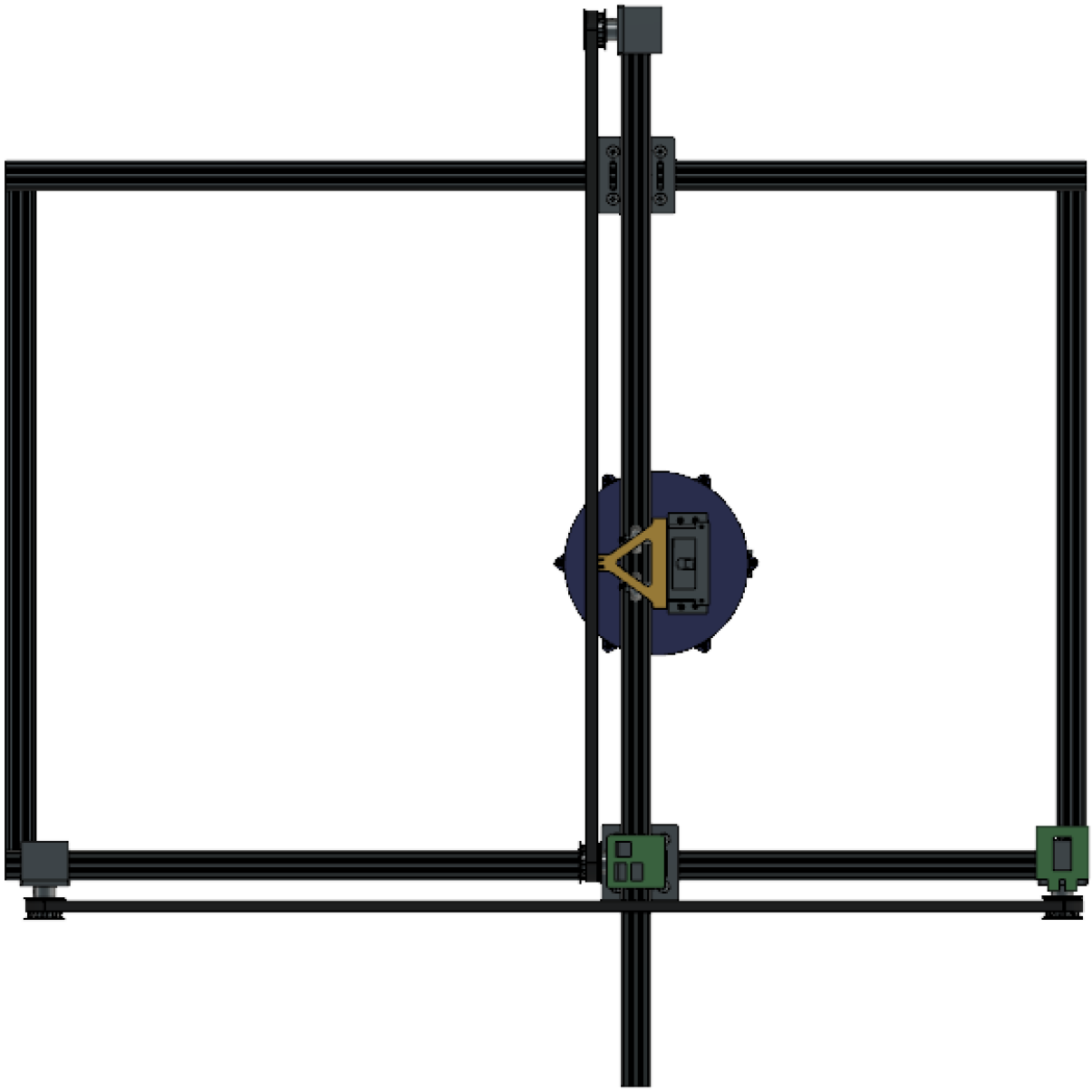
Item	Quantity	Price/Item	Source
Dominos	55	$ 9.97	Amazon
80/20 Aluminum	five 2ft and four 1ft pieces	N/A	lab (borrowed from Tom)
Set of 2020 Aluminum T-slot rails	1	$ 31.99	Amazon
2020 Aluminum T-slot rail	1	$ 13.95	Amazon
Set of 10 brackets and bolts for 80/20	1	$ 10.47	Amazon
Carriage (Gantry support with bearing slide for 20 x 20 mm extrude)	3	$ 11.99	Amazon
∅1/8" hardened steel bearing balls	1	$ 5.80	Amazon
Roller lever actuated limit switch (Pack of 10)	1	$ 4.99	Amazon
Belt Drive Kit (belt, driving pulleys, and driven pulleys)	1	$ 13.99	Amazon
Mini Servo with Position Feedback	1	$ 19.95	Amazon

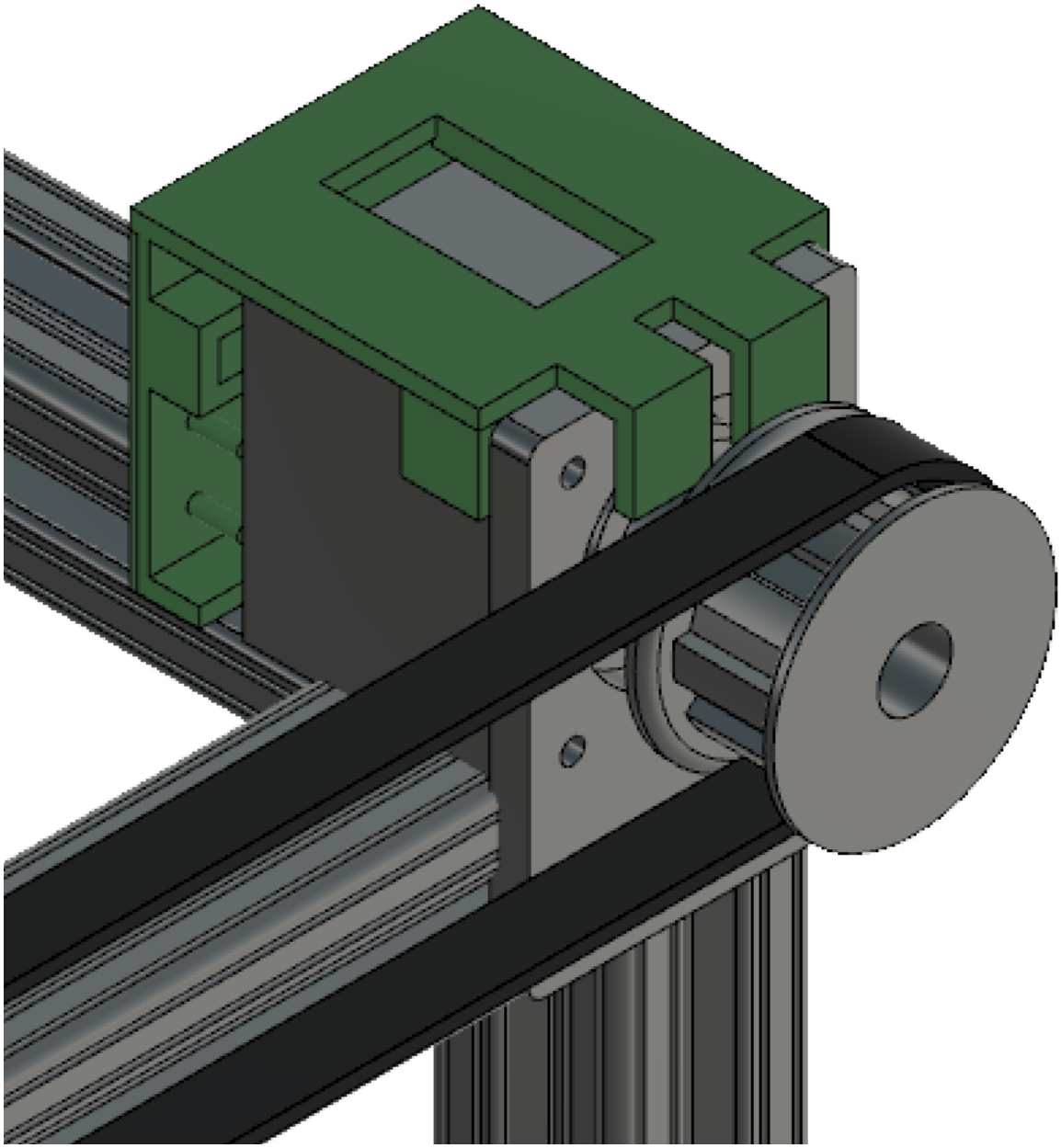
M5 Nut to attach stack to carriage	4	N/A	Mcmaster
M5 16 mm bolt to mount stack	4	N/A	Mcmaster
3-48x1/2" bolt to connect rotating housing	6	$ 0.55	Mcmaster
3-48x3/4" bolt to attach rotation to stack	4	$ 0.50	Mcmaster
0-80X1/4" bolt to mount rotating servo	6	$ 0.25	Mcmaster
0-80 Nut	4	$ 0.33	Mcmaster
0-80 washer	4	$ 0.40	Mcmaster
3-48 washer	10	N/A	Mcmaster
3-48 nut	10	$ 0.59	Mcmaster
5/16x2" bolts	4	$ 7.08 per 10	Mcmaster
5/16 nut	4	$ 6.38 per 100	Mcmaster
Permatex 24200 Threadlocker Blue	6 ml	$ 7.54	Amazon
Stepper Motors	4	N/A	lab (borrowed from Tom)
ESP32 Huzzah32	1	N/A	lab kit
Breadboard	1	N/A	lab kit
1/16" Acrylic Sheet	8in x 24in	N/A	Jacobs Hall
PETG 1.75mm filament	1kg spool	$ 21.99	Amazon
PLA 1.75mm filament	1kg spool	N/A	Jacobs Hall
10 kOhm Resistors	5	N/A	lab kit
Electrical Wires	50 ft	N/A	Jacobs Hall
Button	1	N/A	lab kit
Power supply adapter connector	2	N/A	lab kit
Power source 5 V	1	N/A	lab kit
Power source 12 V	1	N/A	lab

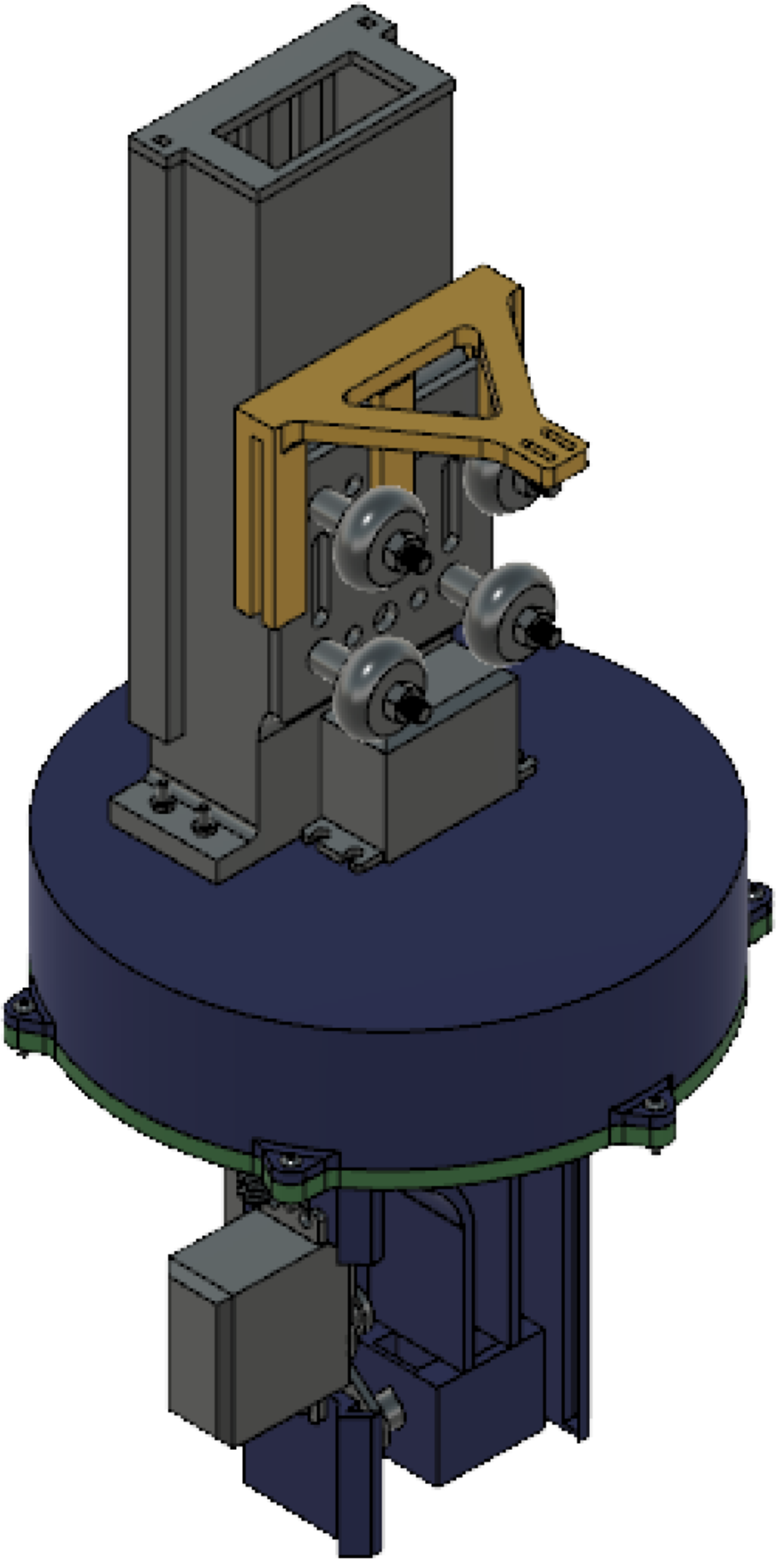
o Images of the CAD, showing mechanical transmission elements

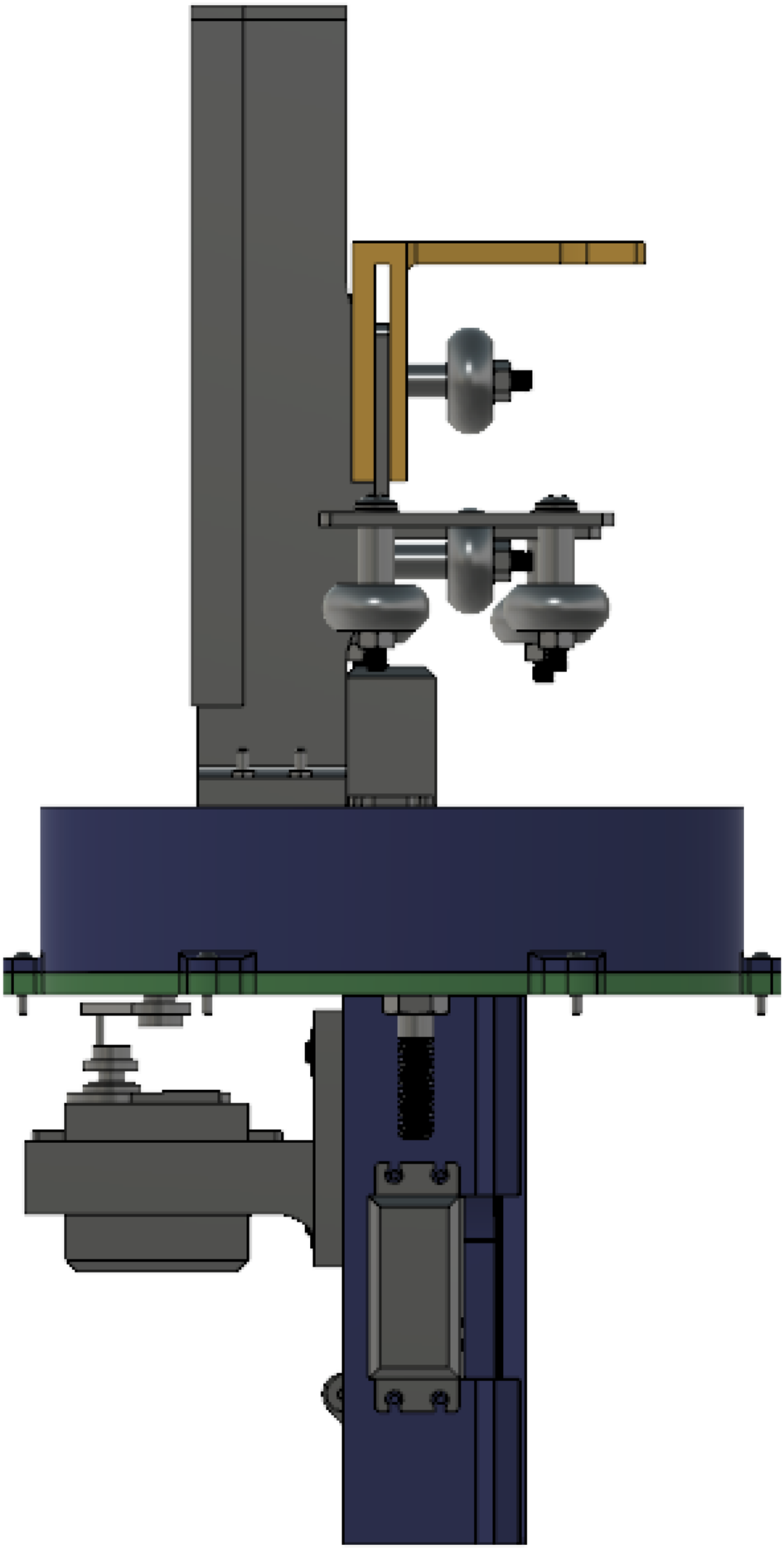


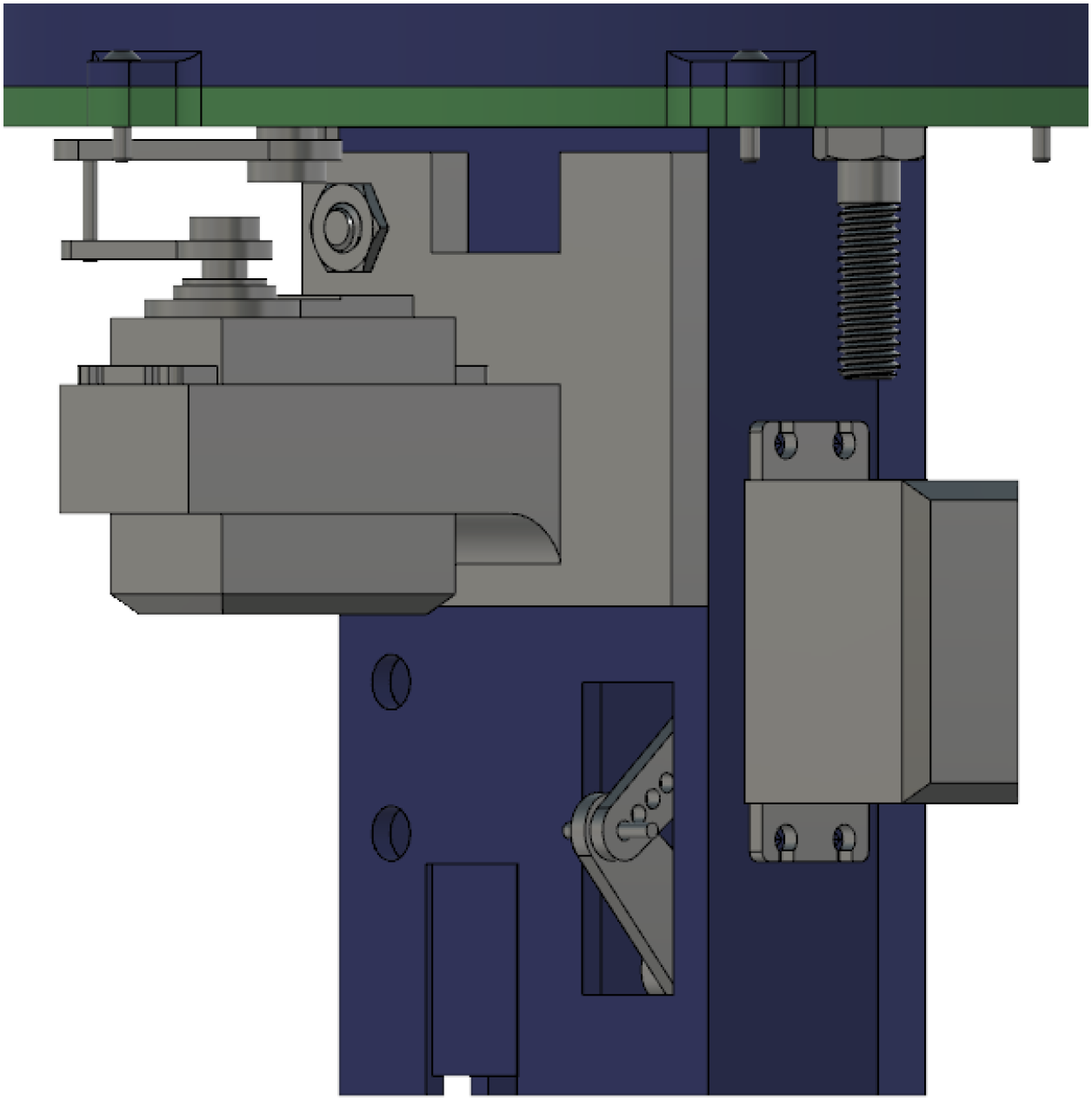


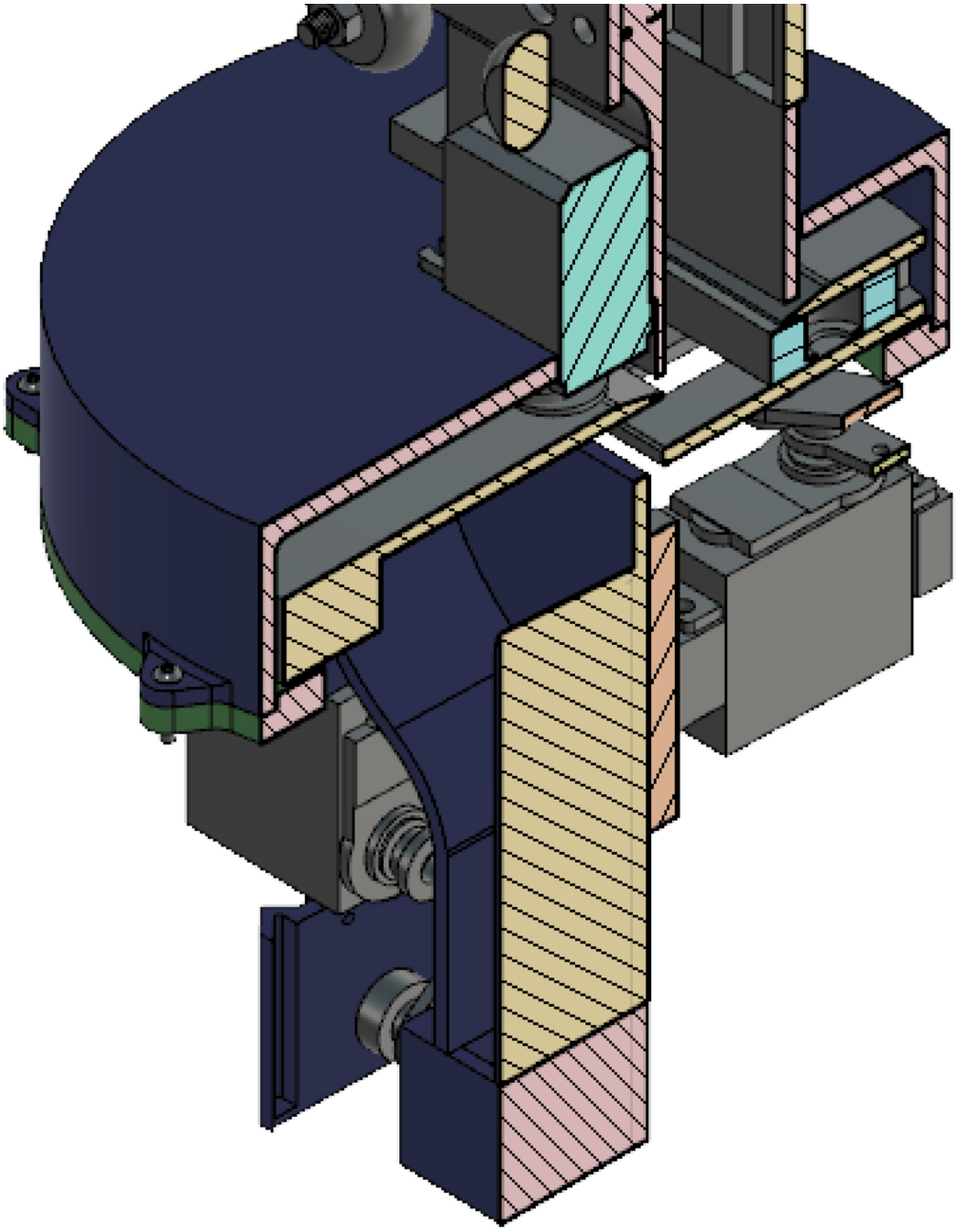












pondposaphiwat / ME102b-Domino-Placer Public

Code Issues Pull requests Actions Projects Wiki Security

main

ME102b-Domino-Placer / ME102B_Full_System.ino

pondposaphiwat cleaned up unused state History

1 contributor

588 lines (468 sloc) | 12.9 KB

```
1   // Libraries for motor control
2   #include <AccelStepper.h>
3   #include <ESP32Servo.h>
4
5   // Pattern import
6   #include "patterns.h"
7
8   #define motorInterfaceType 1
9   #define dirPin 14
10  #define stepPin 32
11  #define dirPin2 15
12  #define stepPin2 33
13
14  #define servoPin1 12
15  #define servoPin2 25    //A1
16  #define servoPin3 26    //A0
17
18  // Switches
19  #define BTN 27
20  #define limitX 39       //A2
21  #define limitY 34       //A3
22
23  // Ultrasonic sensor
24  #define trigPin 22
25  #define echoPin 23
26
27  // Servo timer Init
28  hw_timer_t * timer = NULL;
```

```
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

// Constants
const int refillThreshold = 1000;
const int calibrateMovePosition = -10000;

const float step2inch = 1.5/400; //multiply by this value to go from motor steps to inches
const float inch2step = 400/1.5; //multiply by this value to go from inches to motor steps

const int MAX_X_length = 22; //inches
const int MAX_Y_length = 16; //inches

const int MAX_X_steps = MAX_X_length*inch2step;
const int MAX_Y_steps = MAX_Y_length*inch2step;

const int refillPositionX = 20;
const int refillPositionY = MAX_Y_steps/2;

const int servoHome_chute = 60;
const int servoHome_rotation = 179;
const int servoHome_pusher = 180;

const int servoExtended_chute = 0;
const int servoExtended_pusher = 30;

// Initialize variables (flags)
volatile bool firstMoveCall = true;

volatile bool reachedLimitX = false;
volatile bool reachedLimitY = false;

volatile bool startButtonPressed = false;
bool stackNeedsRefill = false;

//bool placerRotComplete = false;
//bool pusherRotComplete = false;
//bool chuteRotComplete = false;
//bool wholePlacerReturnComplete = false;

// General first, if this doesn't work will do 3 diff timers for 3 servos
bool servoRotComplete = false;

int state = 0;
int ultrasonicDistance;
int currentStep = 0;
int returnStep = 0;

// Pattern declaration
```

```
77  int *x_pattern = pattern_test_x;
78  int *y_pattern = pattern_test_y;
79  int *rot_pattern = pattern_test_rot;
80
81  AccelStepper stepperX = AccelStepper(motorInterfaceType, stepPin, dirPin);
82  AccelStepper stepperY = AccelStepper(motorInterfaceType, stepPin2, dirPin2);
83  Servo myServo_rotation;
84  Servo myServo_pusher;
85  Servo myServo_chute;
86
87  // Limit switch interrupt
88  void IRAM_ATTR isr_limit_x() {  // the function to be called when interrupt is triggered
89    reachedLimitX = true;
90  }
91  void IRAM_ATTR isr_limit_y() {  // the function to be called when interrupt is triggered
92    reachedLimitY = true;
93  }
94
95  // Init Button: back to idle mode immediately
96  void IRAM_ATTR isr_init() {  // the function to be called when interrupt is triggered
97    startButtonPressed = true;
98  }
99
100 // Timer interrupt for timing servo completion
101 void IRAM_ATTR onTime() {
102   portENTER_CRITICAL_ISR(&timerMux);
103   servoRotComplete = true;
104   portEXIT_CRITICAL_ISR(&timerMux);
105 }
106
107 // Timer interrupt hook
108 void TimerInterruptInit() {
109   timer = timerBegin(0, 80, true);
110   timerAttachInterrupt(timer, &onTime, true);
111   timerAlarmWrite(timer, 2000000, true);
112   timerAlarmEnable(timer);
113 }
114
115 void setup() {
116   // put your setup code here, to run once:
117
118   // Interrupt Setup
119   pinMode(BTN, INPUT);
120   pinMode(limitX, INPUT);
121   pinMode(limitY, INPUT);
122   attachInterrupt(BTN, isr_init, RISING);
123   attachInterrupt(limitX, isr_limit_x, FALLING);
124   attachInterrupt(limitY, isr_limit_y, FALLING);
```

```
125
126   // Stepper Setup
127   stepperX.setMaxSpeed(250);
128   stepperX.setAcceleration(30);
129
130   stepperY.setMaxSpeed(250);
131   stepperY.setAcceleration(30);
132
133   // Servo Setup
134   ESP32PWM::allocateTimer(0);
135   ESP32PWM::allocateTimer(1);
136   ESP32PWM::allocateTimer(2);
137   ESP32PWM::allocateTimer(3);
138   myServo_rotation.setPeriodHertz(50);// Standard 50hz servo
139   myServo_rotation.attach(servoPin1, 500, 2500);   // for 90 deg range use 1000 to 2000, for 18
140
141   myServo_pusher.setPeriodHertz(50);// Standard 50hz servo
142   myServo_pusher.attach(servoPin2, 500, 2500);
143   myServo_chute.setPeriodHertz(50);// Standard 50hz servo
144   myServo_chute.attach(servoPin3, 500, 2500);
145
146   // Ultrasonic Sensor Setup
147   pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
148   pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT
149
150   // Timer Setup
151   TimerInterruptInit();
152
153   Serial.begin(9600);
154 }
155
156 void loop() {
157   // State checker
158 //  Serial.print("  Current step: ");
159 //  Serial.print(currentStep);
160   Serial.print("State: ");
161   Serial.println(state);
162
163
164   // put your main code here, to run repeatedly:
165   switch (state) {
166
167     // Idle
168     case 0:
169       idleStateService();
170       if (startButtonPressedChecker()) {
171         startPressedAtIdleService();
172       }
```

```
173       break;
174
175     // Calibration nation
176     case 1:
177       // E-Stop
178       if (startButtonPressedChecker()) {
179         eStopService();
180       }
181
182       // Run motors
183       calibrationRun();
184
185       // Calibrate Once
186       if(firstMoveCallChecker())
187         calibrationStateService();
188
189       firstMoveCallFalseService();
190
191       if (LimitSwitchChecker('s')) {
192         limitSwitchesReachedService();
193       } else {
194         if (LimitSwitchChecker('x')) {
195           calibrateStepper(&stepperX);
196         }
197         if (LimitSwitchChecker('y')) {
198           calibrateStepper(&stepperY);
199         }
200       }
201
202       break;
203
204     // Move XY
205     case 2:
206       // E-Stop
207       if (startButtonPressedChecker()) {
208         eStopService();
209       }
210
211       if (doneWithPatternChecker()) {
212         doneWithPatternService();
213       }
214
215       moveXYService();
216       firstMoveCallFalseService();
217
218       if (atTargetXYChecker(&stepperX, &stepperY)) {
219         atTargetXYService();
220       }
```

```
221
222       break;
223
224     ////////////////////////////////////////
225     /////// CASE 3, 4, 5: PLACE DOMINO ///////
226     ////////////////////////////////////////
227
228     // Rotate whole placer
229     case 3:
230       // E-Stop
231       if (startButtonPressedChecker()) {
232         eStopService();
233       }
234
235       rotateServo(&myServo_rotation, rot_pattern[currentStep]);
236
237       if (placerRotCompleteChecker()) {
238         placerRotCompleteService();
239       }
240       break;
241
242     // Lower chute
243     case 4:
244       // E-Stop
245       if (startButtonPressedChecker()) {
246         eStopService();
247       }
248
249       rotateServo(&myServo_chute, servoExtended_chute);
250
251       if (chuteRotCompleteChecker()) {
252         chuteRotCompleteService();
253       }
254       break;
255
256     // Push domino
257     case 5:
258       // E-Stop
259       if (startButtonPressedChecker()) {
260         eStopService();
261       }
262
263       rotateServo(&myServo_pusher, servoExtended_pusher);
264
265       if (pusherRotCompleteChecker()) {
266         pusherRotCompleteService();
267       }
268       break;
```

```
269     ////////////////////////////////////////
270
271     ////////////////////////////////////////////////////
272     ///CASE 8, 9, 10: RETURN PLACER TO ZERO CONFIG STATE ///
273     ////////////////////////////////////////////////////
274
275     // Return chute
276     case 8:
277       // E-Stop
278       if (startButtonPressedChecker()) {
279         eStopService();
280       }
281
282       rotateServo(&myServo_chute, servoHome_chute);
283       if (chuteReturnCompleteChecker()) {
284         chuteReturnCompleteService();
285       }
286       break;
287
288     // Return placer rotation
289     case 9:
290       // E-Stop
291       if (startButtonPressedChecker()) {
292         eStopService();
293       }
294       rotateServo(&myServo_rotation, servoHome_rotation);
295
296       if (placerReturnCompleteChecker()) {
297         placerReturnCompleteService();
298       }
299       break;
300
301     // Retract pusher
302     case 10:
303           // E-Stop
304       if (startButtonPressedChecker()) {
305         eStopService();
306       }
307
308       rotateServo(&myServo_pusher, servoHome_pusher);
309       if (pusherReturnCompleteChecker()) {
310         pusherReturnCompleteService();
311
312         if (stackNeedsRefillChecker()) {
313           stackNeedsRefillService();
314
315         } else {
316           stackDoesNotNeedRefillService();
```

```
317
318         }
319       }
320       break;
321     ////////////////////////////////////////
322
323     // Refill - move to closest y edge
324     case 6:
325       refillService();
326
327       if (startButtonPressedChecker()) {
328         doneRefillService();
329       }
330
331       break;
332 }
333
334 ////////////////////////////////////////////////////////////
335 ////////////////////////////////////////////////////////////
336 ////////////////////////////////////////////////////////////
337
338
339 // Event checkers
340 bool startButtonPressedChecker() {
341   return startButtonPressed;
342 }
343
344 bool stackNeedsRefillChecker() {
345   checkRefill();
346   return stackNeedsRefill;
347 }
348
349 bool motorIsNotRunningChecker(AccelStepper stepperX, AccelStepper stepperY) {
350   return (!stepperX.isRunning() & !stepperY.isRunning());
351 }
352
353 bool placerIsAtEdgeChecker() {
354   // Check if it is in the allowed range
355   if (-1 < stepperY.currentPosition() < 1) {
356     return true;
357   } return false;
358 }
359
360 bool LimitSwitchChecker(char axis) {
361   if (axis == 'x') {
362     return reachedLimitX;
363   } else if (axis == 'y') {
364     return reachedLimitY;
```

```
365     } else if (axis == 's') {
366       return (reachedLimitX && reachedLimitY);
367     }
368   }
369
370   // To check whether stepper has reached target position
371   bool stepperRunningChecker(AccelStepper stepper) {
372     return stepper.isRunning();
373   }
374
375   bool placerRotCompleteChecker() {
376     return servoRotComplete;
377   }
378
379   bool chuteRotCompleteChecker() {
380     return servoRotComplete;
381   }
382
383   bool pusherRotCompleteChecker() {
384     return servoRotComplete;
385   }
386
387   bool chuteReturnCompleteChecker() {
388     return servoRotComplete;
389   }
390
391   bool placerReturnCompleteChecker() {
392     return servoRotComplete;
393   }
394
395   bool pusherReturnCompleteChecker() {
396     return servoRotComplete;
397   }
398
399   bool doneWithPatternChecker() {
400     return (currentStep >= sizeof x_pattern);
401   }
402
403   bool atTargetXYChecker(AccelStepper *stepperX, AccelStepper *stepperY) {
404     return (!(stepperX->isRunning()) && !(stepperY->isRunning()));
405   }
406
407   bool firstMoveCallChecker() {
408     return firstMoveCall;
409   }
410
411   // Services
412   void checkRefill() {
```

```
413   int durationSum = 0;
414
415 //  for (int i = 0; i < 10; ++i) {
416 //    digitalWrite(trigPin, LOW);
417 //    delayMicroseconds(2);
418 //    digitalWrite(trigPin, HIGH);
419 //    delayMicroseconds(10);
420 //    digitalWrite(trigPin, LOW);
421 //    int duration = pulseIn(echoPin, HIGH);
422 //    durationSum += duration;
423 //  }
424 //
425 //  float durationAvg = durationSum/10.0;
426 //  Serial.println(durationAvg);
427
428     digitalWrite(trigPin, LOW);
429     delayMicroseconds(2);
430     digitalWrite(trigPin, HIGH);
431     delayMicroseconds(10);
432     digitalWrite(trigPin, LOW);
433     int durationAvg = pulseIn(echoPin, HIGH);
434
435     Serial.print("========== Distance:  ");
436     Serial.println(durationAvg);
437
438   if (durationAvg > refillThreshold) {
439     stackNeedsRefill = true;
440   }
441 }
442
443 void stopStepper(AccelStepper *stepper) {
444   stepper->stop();
445 }
446
447 void calibrateStepper(AccelStepper *stepper) {
448   stepper->stop();
449   stepper->setCurrentPosition(0);
450 }
451
452 void rotateStepper(AccelStepper *stepper, int goal_position) {
453   if(firstMoveCallChecker()) {
454     stepper->moveTo(goal_position);
455   }
456
457   stepper->run();
458 }
459
460 // Check again if blocking is ok, might need timer interrupt to help see if done
```

```
461 void rotateServo(Servo *servo, int goal_angle) {
462   servo->write(goal_angle);
463 }
464
465 void idleStateService() {
466   reachedLimitX = false;
467   reachedLimitY = false;
468 }
469
470 void calibrationRun() {
471   stepperX.run();
472   stepperY.run();
473 }
474 // E-Stop service
475 void eStopService() {
476   reachedLimitX = false;
477   reachedLimitY = false;
478   startButtonPressed = false;
479   firstMoveCall = true;
480   state = 0;
481 }
482
483 void pusherRotCompleteService() {
484   servoRotComplete = false;
485   state = 8;
486   timerRestart(timer);
487 }
488
489 void limitSwitchesReachedService() {
490   reachedLimitX = false;
491   reachedLimitY = false;
492   firstMoveCall = true;
493   calibrateStepper(&stepperX);
494   calibrateStepper(&stepperY);
495   state = 2;
496 }
497
498 void atTargetXYService() {
499   stepperX.stop();
500   stepperY.stop();
501   firstMoveCall = true;
502   servoRotComplete = false;
503   state = 3;
504   timerRestart(timer);
505 }
506
507 void doneWithPatternService() {
508   Serial.println("Done with pattern");
```

```
509   currentStep = 0;
510   state = 0;
511 }
512
513 void calibrationStateService() {
514   stepperX.moveTo(-MAX_X_steps - 10000);
515   stepperY.moveTo(-MAX_Y_steps - 10000);
516 }
517
518 void placerReturnCompleteService() {
519   servoRotComplete = false;
520   state = 10;
521   timerRestart(timer);
522 }
523
524 void placerRotCompleteService() {
525   servoRotComplete = false;
526   state = 4;
527   timerRestart(timer);
528 }
529
530 void doneRefillService() {
531   startButtonPressed = false;
532   firstMoveCall = true;
533   state = 2;
534 }
535
536 void startPressedAtIdleService() {
537   reachedLimitX = false;
538   reachedLimitY = false;
539   startButtonPressed = false;
540   state = 1;
541 }
542
543 void moveXYService() {
544   rotateStepper(&stepperX, x_pattern[currentStep]);
545   rotateStepper(&stepperY, y_pattern[currentStep]);
546 }
547
548 void chuteRotCompleteService() {
549   servoRotComplete = false;
550   state = 5;
551   timerRestart(timer);
552 }
553
554 void chuteReturnCompleteService() {
555   servoRotComplete = false;
556   state = 9;
```

```
557   timerRestart(timer);
558 }
559
560 void pusherReturnCompleteService() {
561   servoRotComplete = false;
562   currentStep += 1;
563 }
564
565 void stackNeedsRefillService() {
566   stackNeedsRefill = false;
567   state = 6;
568 }
569
570 void stackDoesNotNeedRefillService() {
571   state = 2;
572 }
573
574 void errorStateOverService() {
575   startButtonPressed = false;
576   state = 0;
577 }
578
579 void firstMoveCallFalseService() {
580   firstMoveCall = false;
581 }
582
583 void refillService() {
584   rotateStepper(stepperX, refillPositionX);
585   rotateStepper(stepperY, refillPositionY);
586
587   firstMoveCallFalseService();
588 }
```