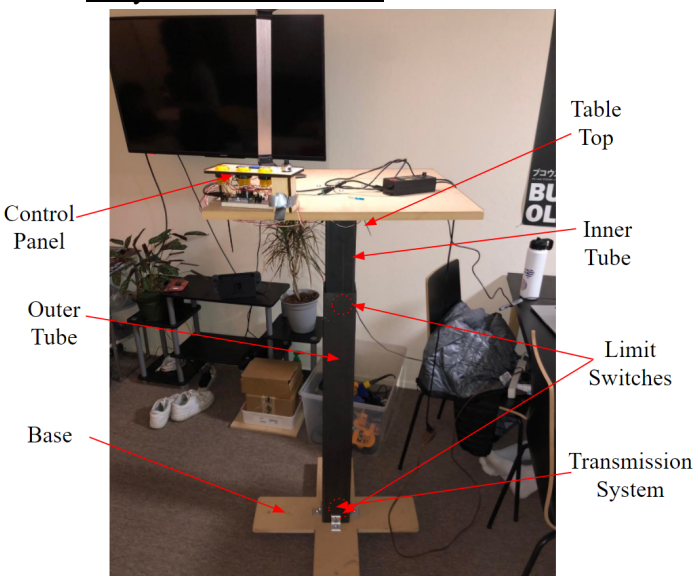
Group 9

Fall 2021 – Project Assignment #4
Final Report

Ben Liao, Alexis Wei, Allison Yuan, Atlas Yuen

For students, teachers, and workers alike, long hours of working can lead to bad working habits. This, in turn, can cause a decrease in productivity, performance, and even result in physical strain on the body, thereby creating an overall bad work experience. Our team chose to address this problem as we are students who deal with this issue every day: studying, doing homework, or even using our computers are all done sitting at a table for countless hours. The purpose of our product is to help people lead healthier work habits and lifestyles. Adjustable standing desks currently exist, but they are often very plain and lack customization abilities. We automated this process to make it easier to find the optimal working position for the user and to promote standing up after a prolonged period of being seated.

Fully Assembled Device:



Transmission System:

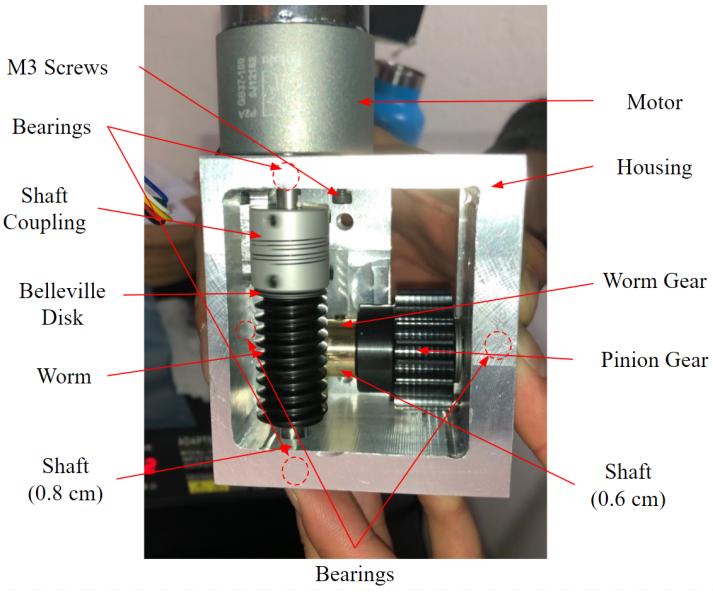
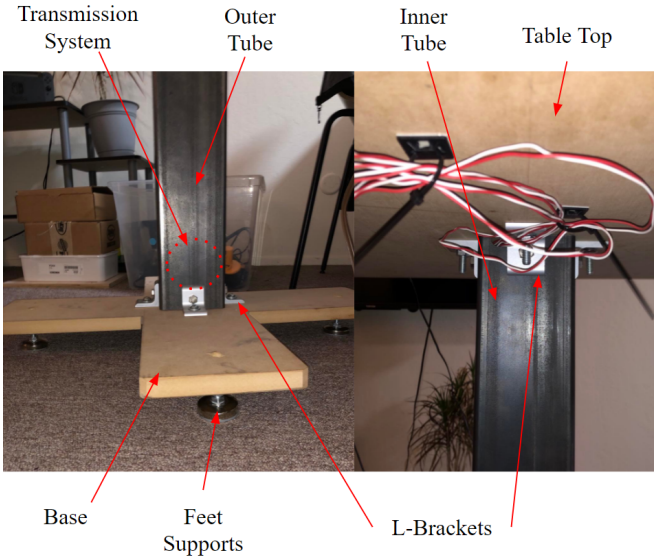
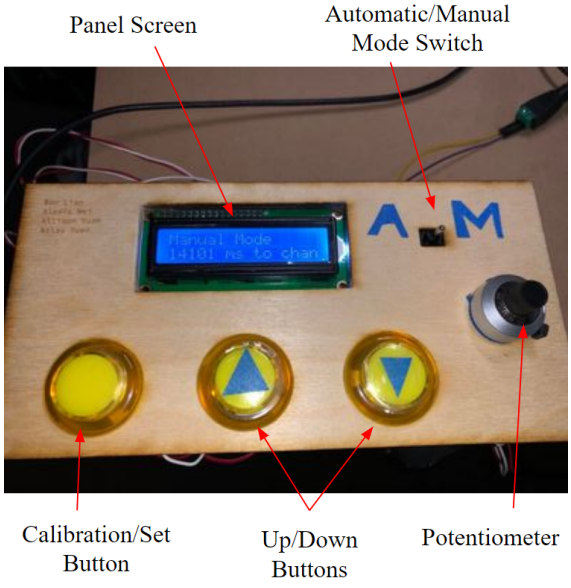


Table Base/Top Joints:



Control Panel:



Group 9

From our initial project proposal of our high-level strategy and system-level design to our final design, we made numerous changes for us to create a successful device. Our high-level strategy was to create an adjustable standing desk that takes the inputs of an individual's height and automatically adjusts to the optimal seated or standing position for the best posture for the individual (based on 5th-95th percentile female/male dimensions), which we did end up executing. However, we initially planned to offer modular add-ons such as an alarm, PDF scanner, or task reminder which we did not add to this device after the suggestions from the GSI and professor.
For our system-level design from our initial project proposal, we changed important aspects of the design such as attaching the rack to the outer tube instead of the inner tube and attaching the motor and transmission to the inner tube instead of the outer tube. Furthermore, we changed the shape of the tube to be a square shape instead of a circular shape to more easily mount the housing of the transmission system to our tube. We also changed the length of the shaft and created 2 small shafts to fit our system.
Our main function critical decisions were all with regard to the heavy-duty transmission system, such as deciding on the motor to use, or the number of teeth on the worm gear. We wanted our transmission to be able to lift 300 lbs, including the table top and inner tube. We accomplished this by choosing a high-torque motor driving a 20:1 speed-reducing transmission, then performing computations to guarantee that the theoretical lifting force of our table met our design specifications.

Calculations

Compute Lifting Force of Mechanism

$$\tau_{recommended\ max\ for\ motor} = 100kg * mm = 10kg * cm \approx 100N * cm = 1N * m.$$

This torque is transferred to the worm, so the worm has a torque of 1 N*m. We know that the worm to worm gear speed (gear) ratio is 20:1, so there is 20 N*m torque on the worm gear. The worm gear to pinion gear's gear ratio is 1:1, so there is 20 N*m torque on the pinion gear. Finally, the pinion gear's radius = 15 mm = 0.015 m. Therefore, with the motor running at maximum recommended torque, force applied from pinion to rack vertically is computed as follows:

$$\tau = F * r \rightarrow F = \frac{\tau}{r} = \frac{20N*m}{0.015m} = \mathbf{1333.3\ N}\ (\approx 136\ \text{kg} = 300\ \text{lbs})$$

Assuming a conservative 75% efficiency for the worm gear (given the slow-spinning worm transmission at 20:1 gear ratio), we achieve our design goal of generating lifting force of 1000N.

Compute Force Experienced by Bearings

Forces are applied to the shaft at four points: the left bearing (closest to the pinion gear), the pinion gear (force applied by rack to the pinion gear), the worm gear (force applied by the worm on the worm gear), and the right bearing (closest to worm gear). We wish to find the reaction forces at the left and right bearings, which means we must find the radial and tangential forces acting on the gears at maximum load and torque, followed by a static analysis on the system. The tangential force on the pinion gear is the maximum lifting load, calculated previously as ≈ 1333 N. The corresponding radial force on the pinion gear (which has a 20° pressure angle) is:

$$1333 * \tan(20°) = 485\ \text{N}.$$

The radial force on the worm gear is given by: $F_{rw} = [F_{tw} \sin \Phi n] \div [(\cos \Phi n \sin \lambda) + \mu \cos \lambda]$, where F_{tw} is the tangential force on the worm, Φ is the pressure angle, λ is the lead angle, and μ is the coefficient of friction between worm and worm gear. An estimate for the coefficient of friction between the worm and worm gear for our system is $\mu = 0.03$ (from low speeds & speed ratio, high pressure angle, lubrication). The lead angle is given by the following: $\lambda = \tan^{-1}(L / \pi d) = \tan^{-1}(z_w p_x / \pi d_w)$, where $z_w = 1$ is the number of thread starts, p_x is the circular pitch of the worm gear, and d_w =16 mm is the pitch diameter of the worm. The circular pitch of the worm gear is given by the following: $p_x = (\pi d_g) / z_g$, where d_g is the pitch diameter

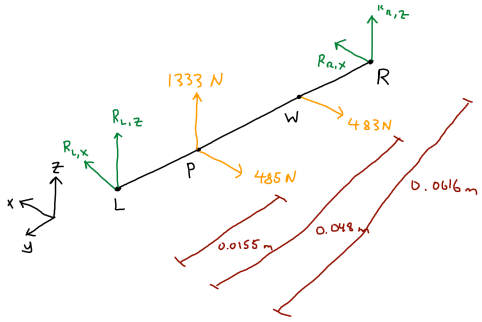
of the worm gear (20 mm) and z_g is the number of teeth on the worm gear (20). Thus the circular pitch is p_x = (π * 20 mm) / 20 = 3.142 mm. Plugging back to find the lead angle, we have:

$$\lambda = \tan^{-1}((1 * 3.142 \text{ mm}) / (\pi * 16 \text{ mm})) = 3.577°.$$

Finally, plugging back to find the radial force, we have:

$$F_{rw} = [(125 \text{ N})\sin(20°)] \div [(\cos(20°)\sin(3.577°)) + 0.03\cos(3.577°)] = 483 \text{ N}$$

Now we consider the following free body diagram of the shaft and labeling, we solve for the reaction forces in the x and z directions at the right and left bearings. Point P is the location of the pinion gear on the shaft; point W is the location of the worm gear on the shaft. Writing out the equations, we have:



$\sum F_z = R_{L,z} + R_{R,z} + 1333 = 0$

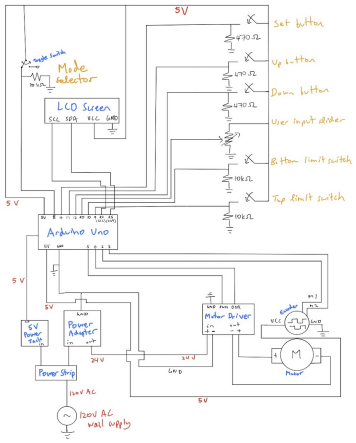
$\sum F_x = R_{L,x} + R_{R,x} - 485 - 483 = 0$

$\sum M_{L,z} = -0.0155(485) - 0.048(483) + 0.0616R_{R,x} = 0$

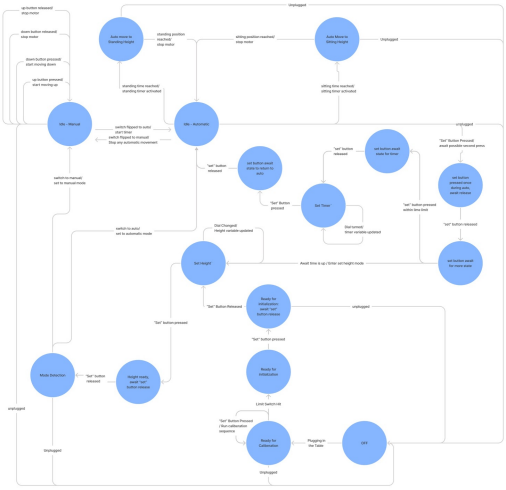
$\sum M_{L,x} = -0.0155(1333.3) - 0.0616R_{R,z} = 0$

Solving, we have $R_{L,z}$ = 997.8 N, $R_{R,z}$ = 335.5 N, $R_{L,x}$ = 469.6 N, $R_{R,x}$ = 498.4 N. This gives us a magnitude of $|\mathbf{R_R}|$ **= 600.8 N ≈ 135 lbs** and $|\mathbf{R_L}|$ **= 1102.8 N ≈ 248 lbs**. These are less than the rated maximum load of 370 lbs on the bearings we purchased, showing that the bearings can handle maximum loading conditions.

Circuit Diagram*:



State Diagram*:



*Larger versions of these pictures are put in appendix for readability

Reflection for the Class

The team brainstormed ideas and discussed solutions together vigorously, enabling us to catch potential flaws in each others' designs before the final manufacturing of the assembly. Making sure that the solution was reviewed and made sense to each member of the team helped us be successful in achieving a final design that required minimal alterations. However, we did not allocate enough time for materials to arrive, and when some items were backordered, it left us scrambling to find replacements from other manufacturers. In the future, we would definitely leave at least 3 weeks for this process and would strongly recommend future students to take this into consideration. During this project, we were maybe too conservative with material strength and used more material than necessary. Although this was not a point of failure, learning more about this in advance would have saved us some amount of material costs.

Group 9

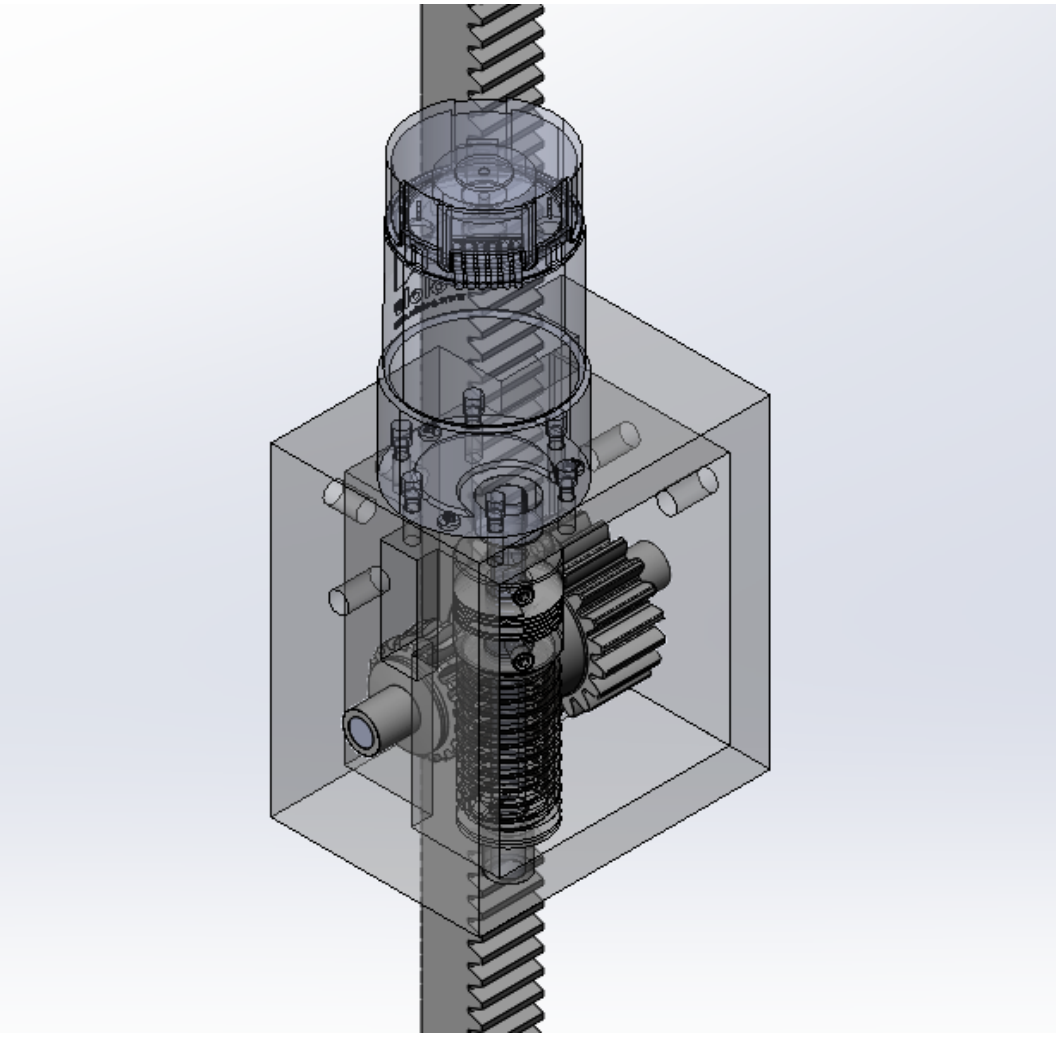
Appendix

Bill of Materials

https://docs.google.com/spreadsheets/d/1tdtZyfz99KjiVQWYW79l_5LbjZedNkZaBb2w3WFj6Ug/edit?usp=sharing

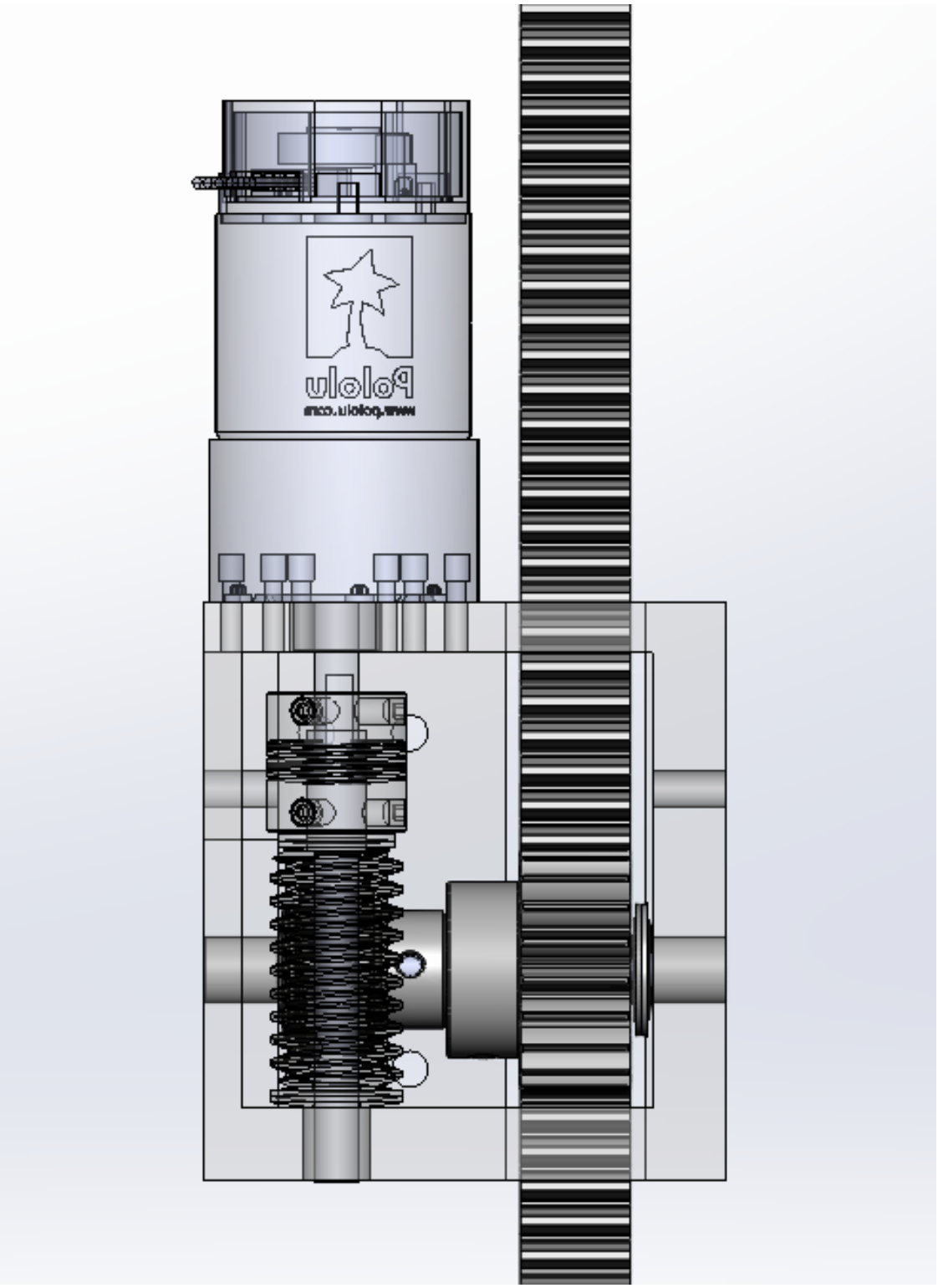
CAD of Transmission System

Isometric View



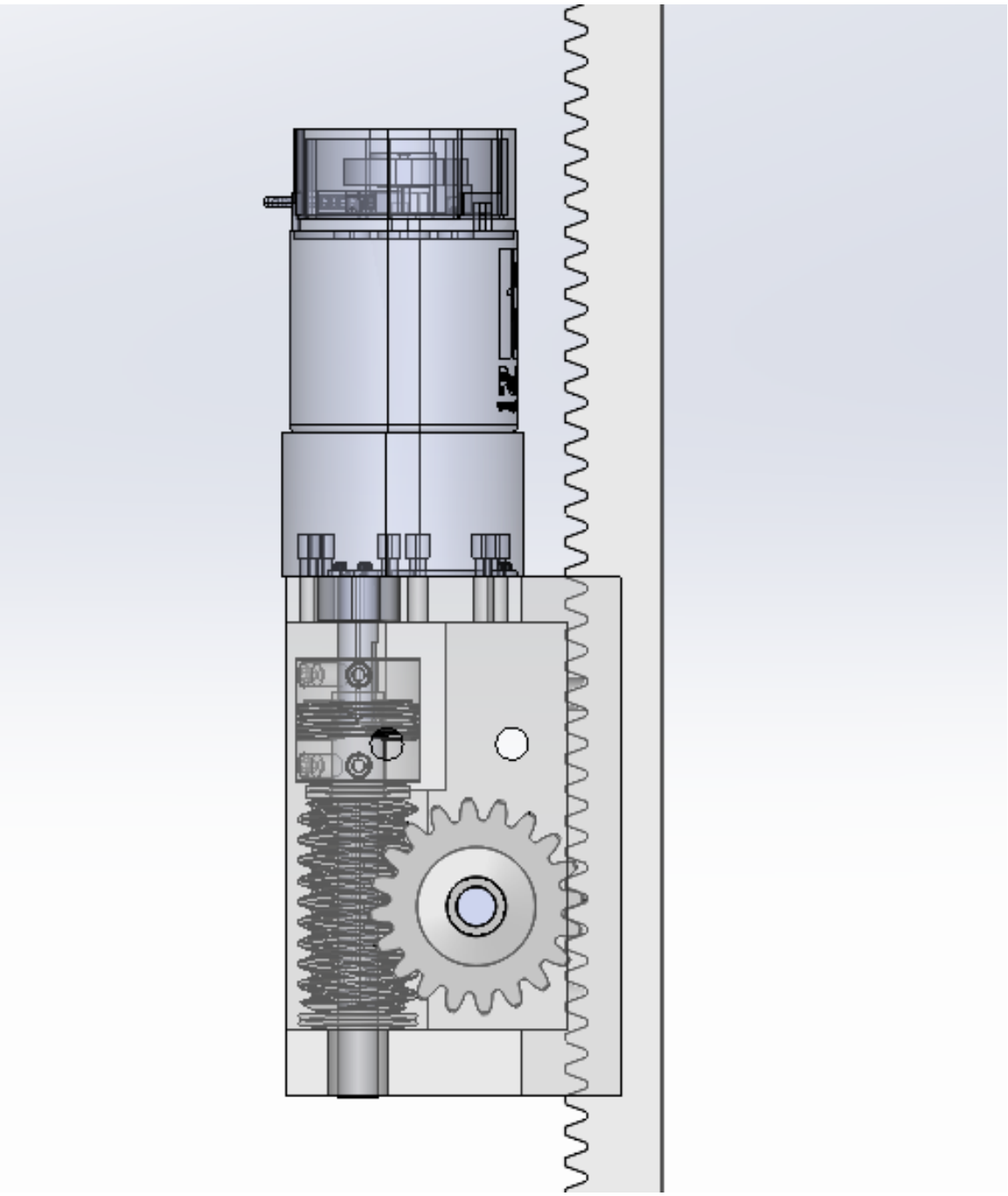
Group 9

Front View



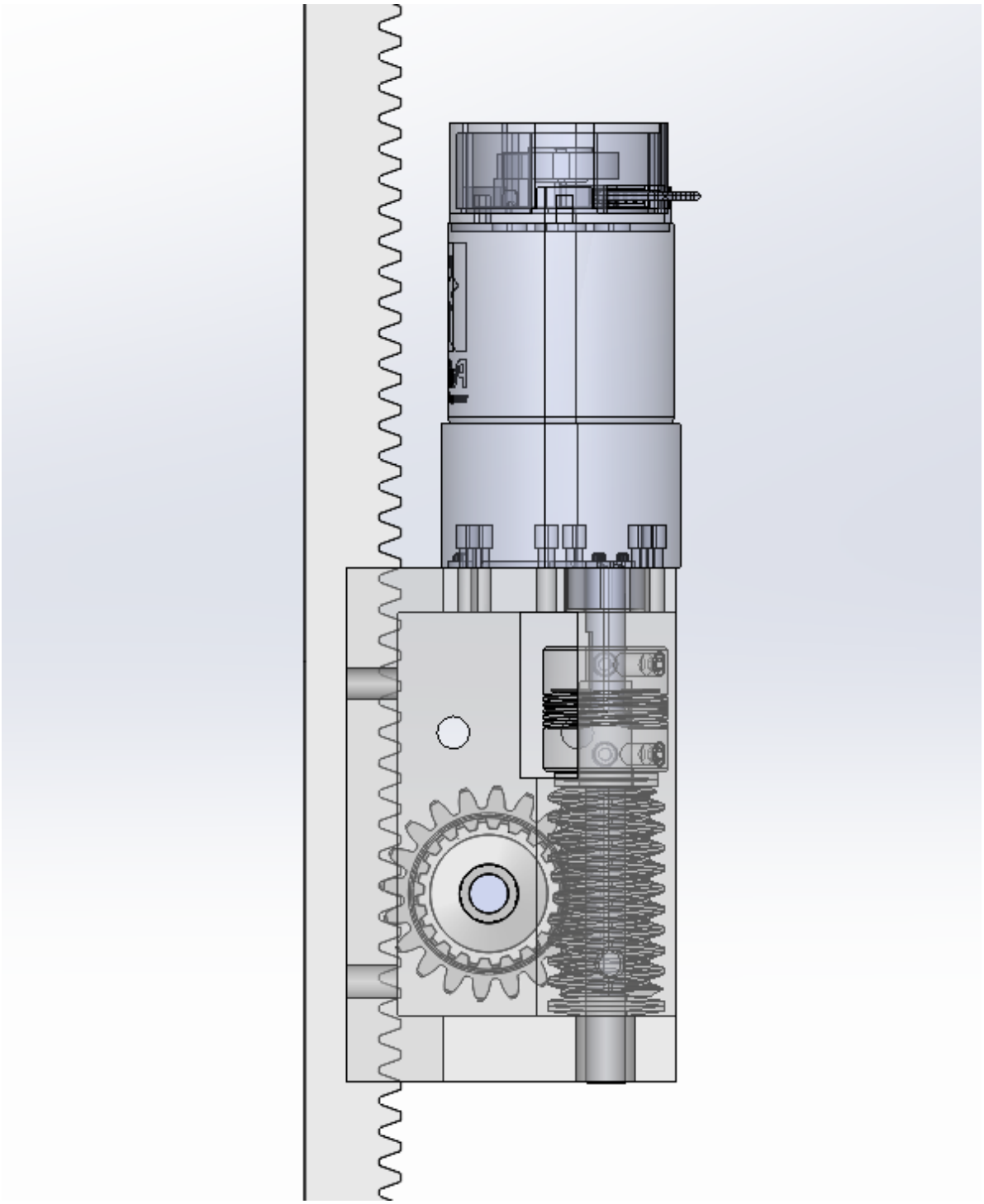
Group 9

Right View



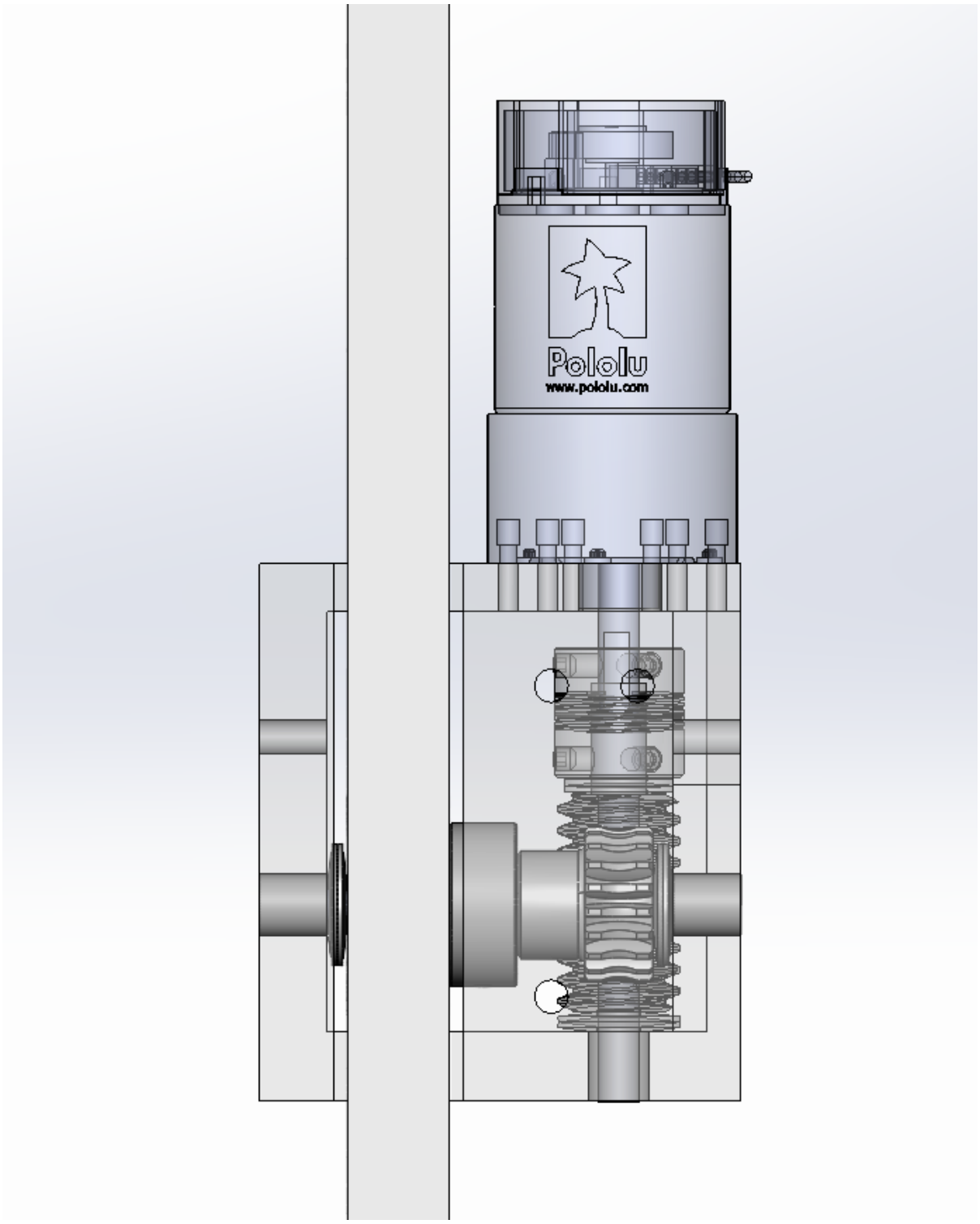
Group 9

Left View



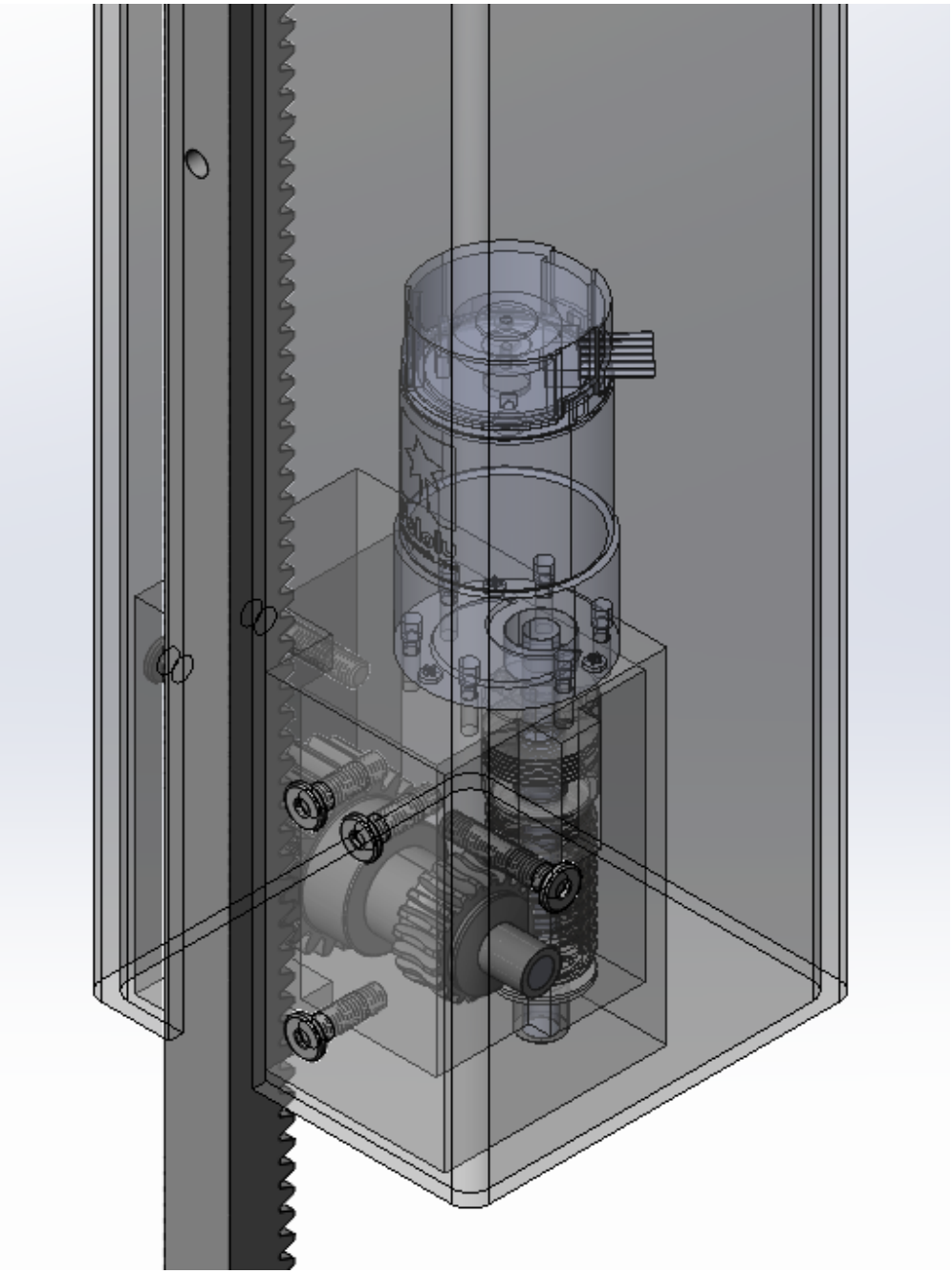
Group 9

Back View



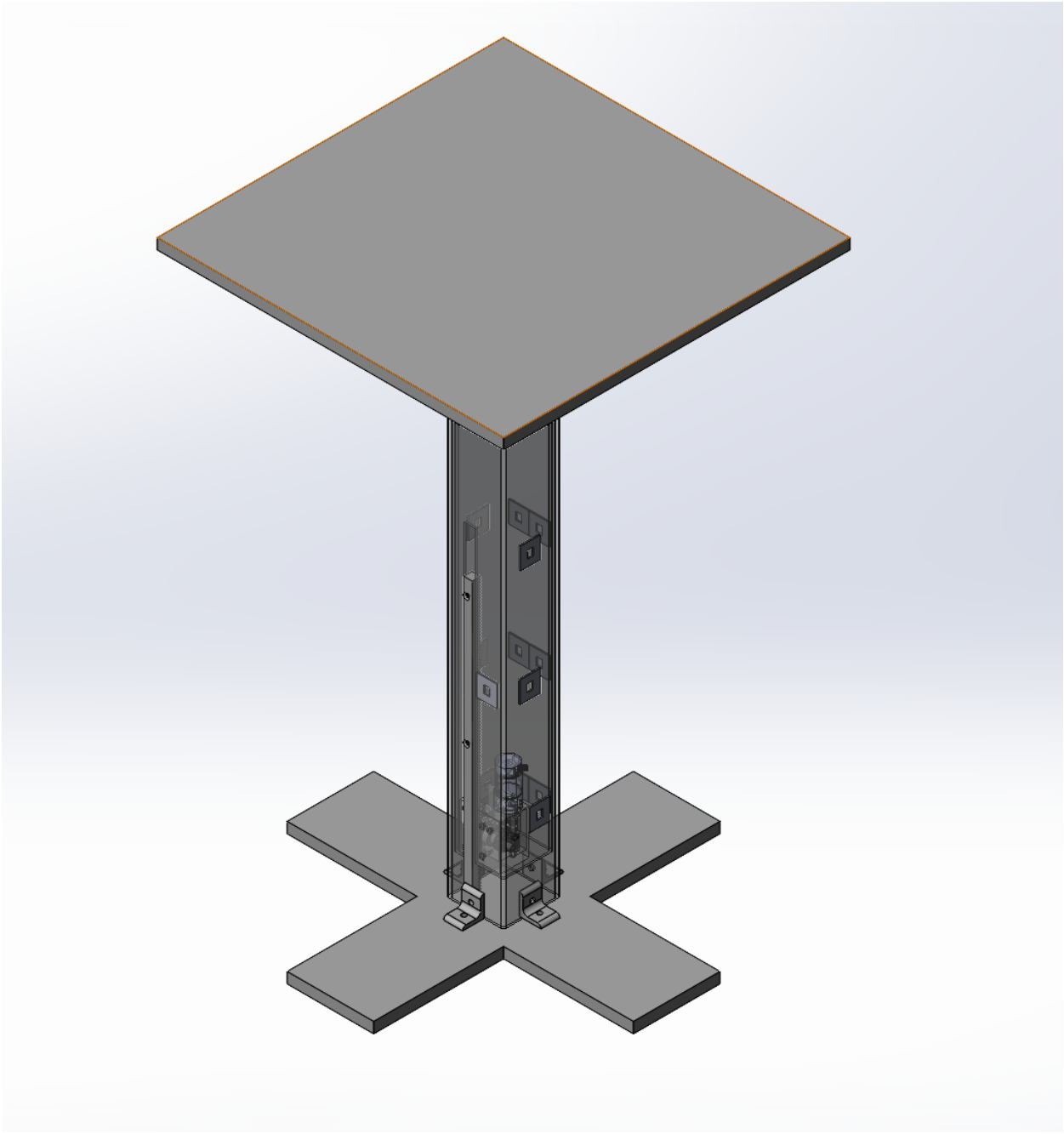
Group 9

Transmission system mounted into inner tube



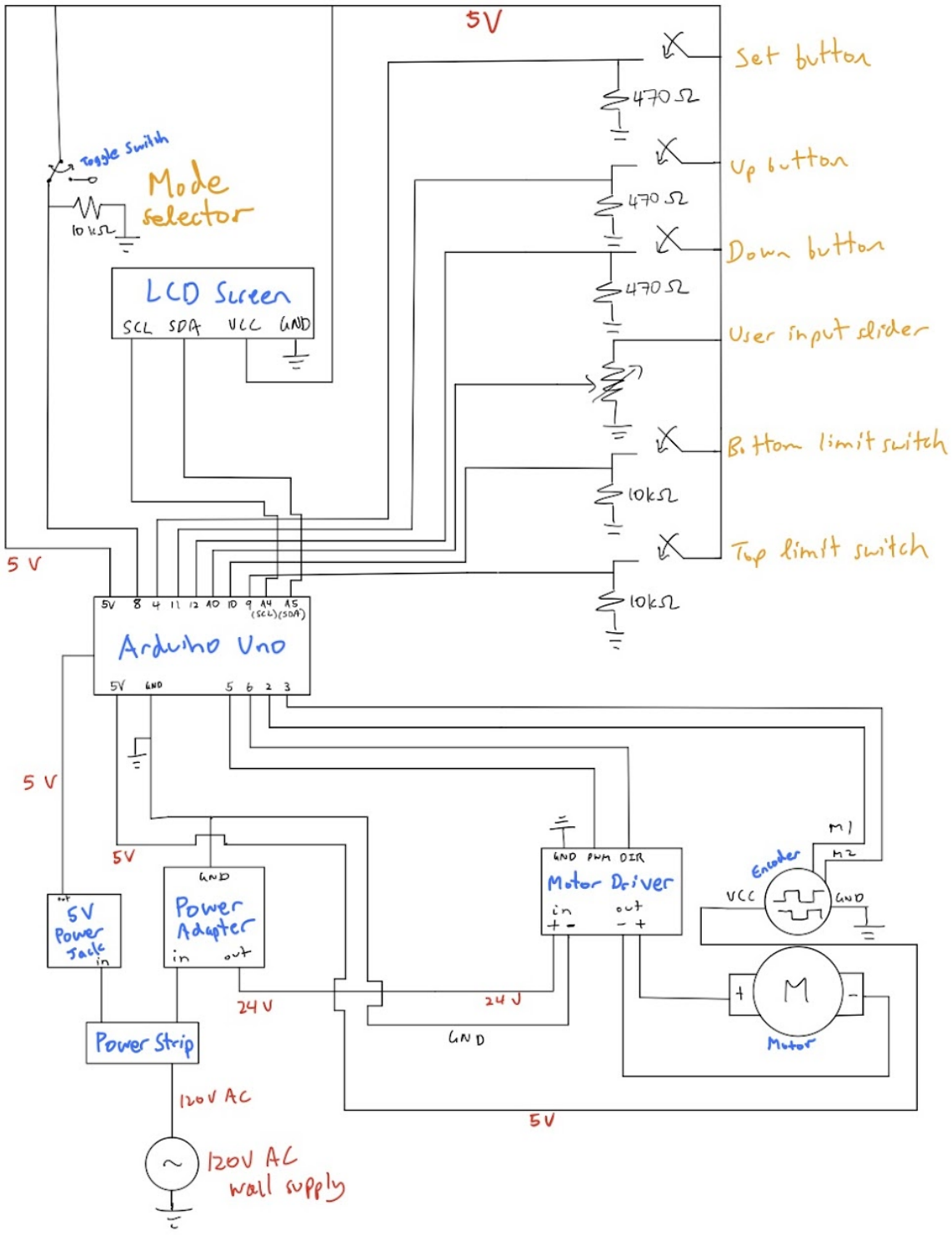
Group 9

Final CAD Assembly



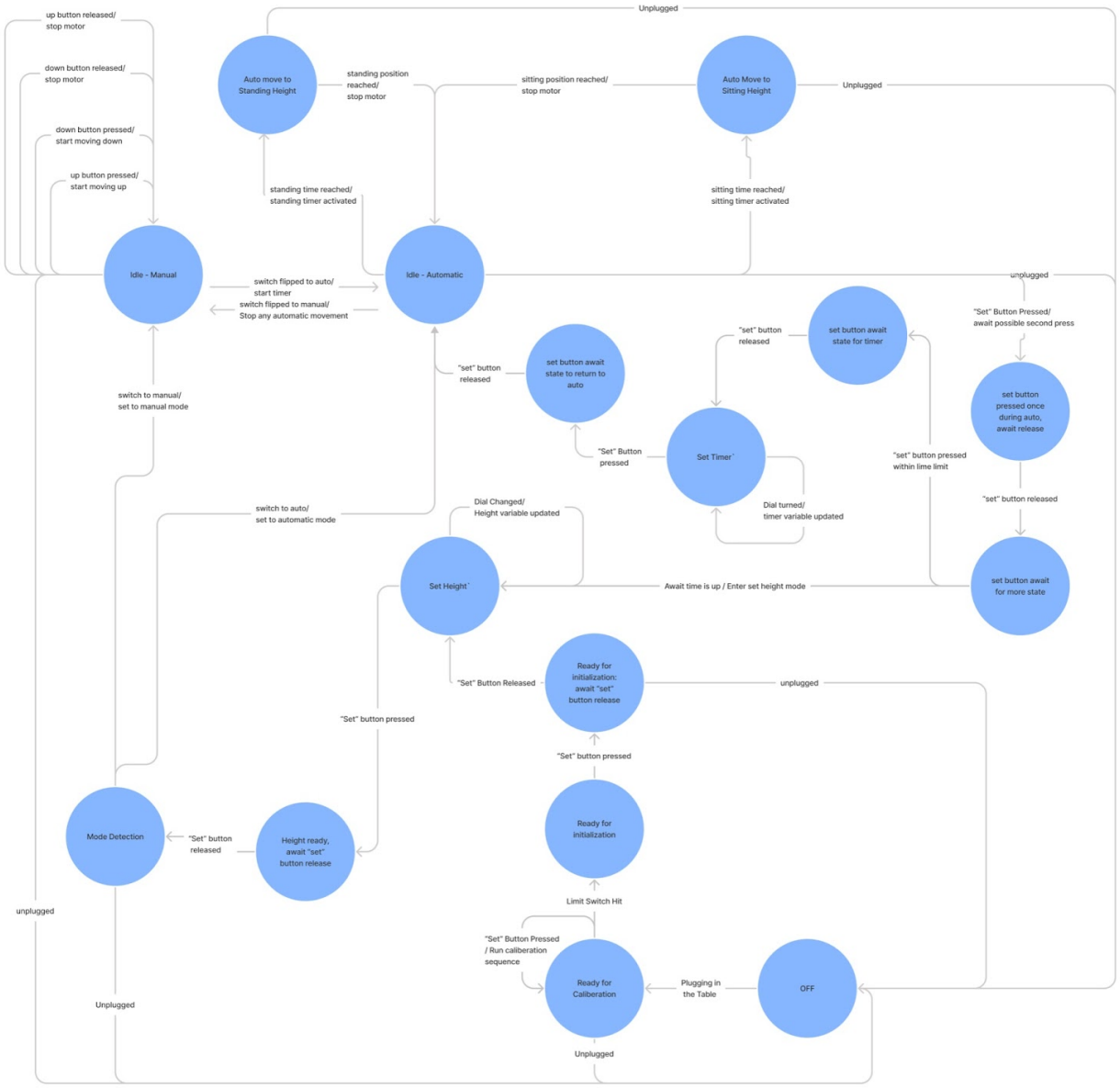
Group 9

Circuit Diagram



Group 9

State Diagram



Link to State Diagram:
https://www.figma.com/file/Q5hgkG41WCWkfkwWFHURdQ/State-Diagram?node-id=0%3A1

Group 9

Arduino Code

```
#include <PinChangeInterrupt.h>
#include <LiquidCrystal_I2C.h>

// pin ports in the Arduino ----------------------------
#define AENC1 2   // first encoder pin, this is the interrupt pin
#define AENC2 3   // second encoder pin
#define MOTOR_PWM 5
#define MOTOR_DIR 6
#define SWITCH 8
#define LIMIT_TOP 9
#define LIMIT_BOTTOM 10
#define BTN_UP 11
#define BTN_DOWN 12
#define BTN_SET 4
#define DIAL A0

// lcd screen macros and variables ----------------------------
// macros for specifying which line to print to
#define TOP 0
#define BOT 1
// macro for lcd screen refresh rate, in millis
#define DISP_PERIOD 1500

// Constants ----------------------------
const float cm_per_tick = 0.0385;
const int release_thresh = 1500; // ticks before target we should stop moving
//const float defaultLow = 30;
//const float defaultHigh = 30;

byte state = 0; //assumed starting at 0 once plugged in

bool upPressed = false;
bool motorOnForward = false;
bool motorOnBackward = false;

int autoTime = 10000;
bool toSitting = true;
byte userHeight = 170;
char topScreenDisplayBuffer[100];
char botScreenDisplayBuffer[100];
byte sittingHeight = 110;
byte standingHeight = 130;
float currHeight = 0;

// the following variables are unsigned longs because the time, measured in
// milliseconds, will quickly become a bigger number than can be stored in an
int.
```

```
unsigned long lastDebounceTime = 0;  // the last time the output pin was
toggled
unsigned long debounceDelay = 50;    // the debounce time; increase if the
output flickers
unsigned long doublePressWait = 1000;
unsigned long currentSetButtonTime;
unsigned long lastSetButtonPressedTime = 0;
unsigned long setButtonPressedOnceTime = 0;
unsigned long sitStandTimer = 0;
unsigned long currTime = 0;

//Setup interrupt variables ----------------------------
volatile bool upButtonPressed = false;
volatile bool upButtonReleased = false;
volatile bool downButtonPressed = false;
volatile bool downButtonReleased = false;
volatile bool setButtonPressed = false;
volatile bool setButtonPressedTwice = false;
volatile bool setButtonReleased = false;
volatile bool modeSwitchFlipped = false;
volatile bool topLimitSwitchChanged = false;
volatile bool bottomLimitSwitchChanged = false;
volatile bool canMoveUp = true; // 0 is hit, 1 is untouched
volatile bool canMoveDown = true;
volatile byte modeVal = 0;
volatile int lastSetButtonState = LOW;   // the previous reading from the
input pin
// holds encoder values
volatile int32_t enc;

// Setup LCD ----------------------------
LiquidCrystal_I2C lcd(0x27, 16, 2);     // lcd screen object
char *top_str = NULL, *bot_str = NULL;  // string being displayed
int top_len = 0, bot_len = 0;           // length of string displayed
int top_mult = 0, bot_mult = 0;         // where we're currently printing
unsigned long last_update = 0;          // last update time


void upButtonChangedIsr() {
  // HIGH means pressed, LOW means released
  if (digitalRead(BTN_UP) == HIGH) {
    Serial.println("up button pressed");
    upButtonPressed = true;
    upButtonReleased = false;
  } else if (digitalRead(BTN_UP) == LOW) {
    Serial.println("up button released");
    upButtonPressed = false;
    upButtonReleased = true;
  }
```

```
}

void downButtonPressedIsr() {
  if (digitalRead(BTN_DOWN) == HIGH) {
    Serial.println("down button pressed");
    downButtonPressed = true;
    downButtonReleased = false;
  } else if (digitalRead(BTN_DOWN) == LOW) {
    Serial.println("down button released");
    downButtonPressed = false;
    downButtonReleased = true;
  }
}

void setButtonPressedIsr() {
  currentSetButtonTime = millis();
  if ((currentSetButtonTime  - lastDebounceTime) < debounceDelay) {
    return;
  }

  if (digitalRead(BTN_SET) == HIGH) {
    if (lastSetButtonState == LOW && (currentSetButtonTime  -
lastSetButtonPressedTime) < doublePressWait) {
      setButtonPressedTwice = true;
    } else {
      setButtonPressedTwice = false;
    }
    setButtonPressed = true;
    setButtonReleased = false;
    lastSetButtonState = HIGH;
    lastDebounceTime = currentSetButtonTime;
    lastSetButtonPressedTime = currentSetButtonTime;
  } else if (digitalRead(BTN_SET) == LOW) {
    lastSetButtonState = LOW;
    lastDebounceTime = currentSetButtonTime;
    setButtonPressed = false;
    setButtonReleased = true;
  }
}

void switchIsr() {
  modeSwitchFlipped = true;
  Serial.println(digitalRead(SWITCH));
  if (digitalRead(SWITCH) == LOW) {
    modeVal = 0;
  } else if (digitalRead(SWITCH) == HIGH) {
    modeVal = 1;
  }
}
```

```
void topLimitSwitchIrs() {
  topLimitSwitchChanged = true;
  Serial.println(digitalRead(LIMIT_TOP));
  if (digitalRead(LIMIT_TOP) == LOW) {
    canMoveUp = true;
  } else if (digitalRead(LIMIT_TOP) == HIGH) {
    //stopMotor();
    canMoveUp = false;

  }
}

void bottomLimitSwitchIrs() {
  bottomLimitSwitchChanged = true;
  Serial.println(digitalRead(LIMIT_BOTTOM));
  if (digitalRead(LIMIT_BOTTOM) == HIGH) {
    canMoveDown = true;
  } else if (digitalRead(LIMIT_BOTTOM) == LOW) {
    canMoveDown = false;
  }
}


void handle_enc_tick() {
  if (digitalRead(AENC2)) {
    enc++;
  } else {
    enc--;
  }
  currHeight = (enc / 100 )* cm_per_tick ;
}

void setup() {
  // put your setup code here, to run once:

  pinMode(AENC1, INPUT);
  pinMode(AENC2, INPUT);
  pinMode(BTN_UP, INPUT);
  pinMode(BTN_DOWN, INPUT);
  pinMode(BTN_SET, INPUT);
  pinMode(SWITCH, INPUT);
  pinMode(LIMIT_TOP, INPUT_PULLUP);
  pinMode(LIMIT_BOTTOM, INPUT_PULLUP);
  pinMode(DIAL, INPUT);
  pinMode(MOTOR_PWM, OUTPUT);
  pinMode(MOTOR_DIR, OUTPUT);
  attachPCINT(digitalPinToPCINT(BTN_UP), upButtonChangedIsr, CHANGE);
  attachPCINT(digitalPinToPCINT(BTN_DOWN), downButtonPressedIsr, CHANGE);
```

```
  attachPCINT(digitalPinToPCINT(BTN_SET), setButtonPressedIsr, CHANGE);
  attachPCINT(digitalPinToPCINT(LIMIT_TOP), topLimitSwitchIrs, CHANGE);
  attachPCINT(digitalPinToPCINT(LIMIT_BOTTOM), bottomLimitSwitchIrs, CHANGE);
  attachPCINT(digitalPinToPCINT(SWITCH), switchIsr, CHANGE);
  attachPCINT(digitalPinToPCINT(AENC1), handle_enc_tick, RISING); // set up
ISRs for encoder

  enc = 0;

  Serial.begin(115200);
  canMoveUp = digitalRead(LIMIT_TOP);
  canMoveDown = digitalRead(LIMIT_BOTTOM);
  Serial.print("canMoveDown: ");
  Serial.print(canMoveDown);
  setupLCD();
  if (canMoveDown == 0) {
    state = 1;
    printScreen("Press Set button to continue", TOP);
    printScreen("", BOT);
  } else {
    state = 0;
    printScreen("Press the Set button to caliberate", TOP);
  }

  Serial.print("state: ");
  Serial.print(state);
  modeVal = digitalRead(SWITCH);

}

void loop() {
  // put your main code here, to run repeatedly:
  //Serial.println(state);
  if (millis() - last_update > DISP_PERIOD) {
    updateDisplay();
    last_update = millis();
  }
  switch (state) {
    case 0: // Calibration - watch for set button to start
      if (setButtonPressedEvent()) {
        printScreen("Caliberating", TOP);
        setButtonPressed = false;
        Serial.println("starting caliberation");
        rotateMotorBackward();
      }

      if (bottomLimitSwitchChangedEvent()) {
        Serial.print("canMoveDown: ");
        Serial.println(canMoveDown);
```

```
      Serial.print("motorOnBackward: ");
      Serial.println(motorOnBackward);
      if (canMoveDown == false) {
        stopMotor();
        enc = 0; // set encoder to zero
        printScreen("done caliberating", TOP);
        Serial.println("switch to state 1");
        state = 1;
        printScreen("Press Set button to continue", TOP);
        printScreen("", BOT);
        break;
      }
    }
    break;
  case 1: // Read to initialize

    if (setButtonPressedEvent()) {
      Serial.println("switch to state 8");
      state = 8;
    }
    break;
  case 2: // Height Detection
    setupUserHeight();
    if (setButtonPressedEvent()) {
      calculateSittingHeight(userHeight);
      calculateStandingHeight(userHeight);
      //Serial.println("switch to state 9");
      state = 9;
      break;
    }
    break;
  case 3: // Mode Detection
    if (modeVal == 0) {
      //manual
      //Serial.println("switch to state 4");
      state = 4;
      printScreen("Manual Mode", TOP);
    } else if (modeVal == 1) {
      setupAutoTime();
      //Serial.println("switch to state 5");
      state = 5;
      printScreen("Auto Mode", TOP);
    }
    break;
  case 4: // Idle - Manual
    if (modeChanged()) {
      //Serial.println("switch to state 3");
      state = 3;
      break;
```

```
      }

      if (upButtonPressedEvent() && motorOnForward == false && canMoveUp ==
true) {
        //Serial.println("up button is being pressed");
        rotateMotorForward();
      } else if (upButtonReleasedEvent()) {
        //Serial.println("up button is being released");
        stopMotor();
      } else if (downButtonPressedEvent() && motorOnBackward == false &&
canMoveDown == true ) {
        //Serial.println("down button is being pressed");
        rotateMotorBackward();
      } else if (downButtonReleasedEvent()) {
        //Serial.println("down button is being released");
        stopMotor();
      }
      if (topLimitSwitchChangedEvent()) {
        if (canMoveUp == false && motorOnForward == true) {
          stopMotor();
        }
        break;
      }
      if (bottomLimitSwitchChangedEvent()) {
        if (canMoveDown == false && motorOnBackward == true) {
          stopMotor();
        }
        break;
      }
      break;
    case 5:
      currTime = millis();
      sprintf(botScreenDisplayBuffer, "%d ms to change: ", (autoTime -
(currTime - sitStandTimer)));
      printScreen(botScreenDisplayBuffer, BOT);
      if (modeChanged()) {
        //Serial.println("switch to state 3");
        state = 3;
        break;
      }

      if (setButtonPressedEvent()) {
        setButtonPressedOnceTime = millis();
        //Serial.println("switch to state 7");
        state = 7;
        break;
      }

      if ((currTime - sitStandTimer) >= autoTime && motorOnForward == false
```

```
&& motorOnBackward == false) {
        // timer hit to switch
        if (toSitting == true) {
          Serial.println("moving to sitting height");
          printScreen("Move to sitting", TOP);
          if (currHeight <= (sittingHeight - 110)){
            startMotor(1, sittingHeight - 110);
          } else {
            startMotor(0, sittingHeight - 110);
          }
        } else {
          Serial.println("moving to standing height");

          printScreen("Move to standing", TOP);
          if (currHeight >= (standingHeight - 110)){
            startMotor(0, standingHeight - 110);
          } else {
            startMotor(1, standingHeight - 110);
          }
        }
        toSitting = !toSitting;
        sitStandTimer = currTime;
      }
      if (topLimitSwitchChangedEvent()) {
        if (canMoveUp == false && motorOnForward == true) {
          printScreen("Top limit switch hit", BOT);
          stopMotor();
          toSitting = !toSitting;
          sitStandTimer = currTime;
        }
        break;
      }
      if (bottomLimitSwitchChangedEvent()) {
        if (canMoveDown == false && motorOnBackward == true) {
          printScreen("Bottom limit switch hit", BOT);
          stopMotor();
          toSitting = !toSitting;
          sitStandTimer = currTime;
        }
        break;
      }
      break;
    case 6: // timer mode
      setupAutoTime();
//keep reading the dial until the set button is pressed again
      delay(500);
      if (setButtonPressedEvent()) {
        //Serial.println("switch to state 12");
        state = 12;
```

```
        break;
      }
      break;
    case 7: // after push button once, wait for second button push
      if ((millis() - setButtonPressedOnceTime) > doublePressWait) {
        //Serial.println("switch to state 10");
        state = 10;
        break;
      }

      if (setButtonPressedTwiceEvent()) {
        //Serial.println("switch to state 11");
        state = 11;
        break;
      }
      break;
    case 8: // button pressed at state 1, wait for button release to move on
to state 2
      if (setButtonReleasedEvent()) {
        //Serial.println("switch to state 2");
        state = 2;
        lcd.clear();
        printScreen("setup height", TOP);
        printScreen("Press Set Button to submit", BOT);
      }
      break;
    case 9: // button pressed at state 2, wait for button release to move on
to state 3 (mode detection)
      if (setButtonReleasedEvent()) {
        //Serial.println("switch to state 3");
        state = 3;
        printScreen("detect mode", TOP);
      }
      break;
    case 10: // set button pressed one during state 7, wait for release to
move to state 2
      if (setButtonReleasedEvent()) {
        //Serial.println("switch to state 2");
        state = 2;
        printScreen("setup height", TOP);
        printScreen("Press Set Button to submit", BOT);
      }
      break;
    case 11: // set button pressed twice during state 7, wait for release to
move to state 6
      if (setButtonReleasedEvent()) {
        //Serial.println("switch to state 6");
        state = 6;
        printScreen("setup timer", TOP);
```

```
        printScreen("Press Set Button to submit", BOT);
      }
      break;
    case 12: // exit from timer mode to auto (6 to 5)
      if (setButtonReleasedEvent()) {
        //Serial.println("switch to state 5");
        state = 5;
        printScreen("auto mode", TOP);
      }
      break;
  }
}

//Event checkers
bool modeChanged() {
  // switch the mode from automatic to manual, or vice versa
  if (modeSwitchFlipped == true) {
    modeSwitchFlipped = false;
    return true;
  }
  return false;
}

bool upButtonPressedEvent() {
  if (upButtonPressed == true) {
    return true;
  } else {
    return false;
  }
}

bool upButtonReleasedEvent() {
  if (upButtonReleased == true) {
    upButtonReleased = false;
    upButtonPressed = false;
    return true;
  } else {
    return false;
  }
}

bool downButtonPressedEvent() {
  if (downButtonPressed == true) {
    return true;
  } else {
    return false;
  }
}
```

```
bool downButtonReleasedEvent() {
  if (downButtonReleased == true) {
    downButtonReleased = false;
    downButtonPressed = false;
    return true;
  } else {
    return false;
  }
}

bool setButtonPressedEvent() {
  if (setButtonPressed == true) {
    return true;
  }
  return false;
}

bool setButtonPressedTwiceEvent() {
  // somehow figure out how to detect two presses
  if (setButtonPressedTwice == true) {
    return true;
  }
  return false;
}

bool setButtonReleasedEvent() {
  if (setButtonReleased == true) {
    return true;
  }
  return false;
}

bool topLimitSwitchChangedEvent() {
  if (topLimitSwitchChanged == true) {
    Serial.println("top limit switch change detected");
    topLimitSwitchChanged = false;
    return true;
  }
  return false;
}

bool bottomLimitSwitchChangedEvent() {
  if (bottomLimitSwitchChanged == true) {
    Serial.println("bottom limit switch change detected");
    bottomLimitSwitchChanged = false;
    return true;
  }
  return false;
}
```

```
// Active Functions

void startMotor(byte dir, byte height) {
  // direction 1 = forward
  // direction 0 = backwards

  Serial.print("dir");
  Serial.println(dir);
  Serial.print("height");
  Serial.println(height);
  float target_enc_pos = ((float) height / cm_per_tick) * 100;
  Serial.print("target enc val: ");
  Serial.println(target_enc_pos);

  Serial.print("current encoder value: ");
  Serial.println(enc);
  if (dir == 1 && canMoveUp == true) {
    Serial.println("call to move motor forward");
    while (enc < (target_enc_pos - release_thresh)  && modeVal == 1) {
      motorOnForward = true;
      motorOnBackward = false;
      rotateMotorForward();
    }
  } else if (dir == 0 && canMoveDown == true) {
    Serial.println("call to move motor backward");
    while (enc >= (target_enc_pos + release_thresh) && modeVal == 1) {
      motorOnForward = false;
      motorOnBackward = true;
      rotateMotorBackward();
    }
  }
  stopMotor();
}

void stopMotor() {
  Serial.println("motor stop motion");
  // run actual code that would stop the motor
  digitalWrite(MOTOR_PWM, 0);
  printScreen("Table Stopped", BOT);
  motorOnForward = false;
  motorOnBackward = false;
}

void rotateMotorForward() {
  Serial.println("forward");
  printScreen("Moving Up", BOT);
  digitalWrite(MOTOR_DIR, HIGH);
  digitalWrite(MOTOR_PWM, 255);
```

```
  motorOnForward = true;
  motorOnBackward = false;

}

void rotateMotorBackward() {
  Serial.println("backward");
  printScreen("Moving Down", BOT);
  digitalWrite(MOTOR_DIR, LOW);
  digitalWrite(MOTOR_PWM, 255);
  motorOnForward = false;
  motorOnBackward = true;

}

void setupAutoTime() {
  Serial.print("Dial value: ");
  Serial.println(analogRead(DIAL));
  //every time the new timer changes, all previous timer is cancelled

  autoTime = round(analogRead(DIAL) * 100);
  sprintf(topScreenDisplayBuffer, "new time: %d", autoTime);
  printScreen(topScreenDisplayBuffer, TOP);
  sitStandTimer = millis();
  Serial.print("new time being set: ");
  Serial.println(autoTime);
}

void setupUserHeight() {
  Serial.print("Dial value: ");
  Serial.println(analogRead(DIAL));
  userHeight = round(analogRead(DIAL));
  sprintf(topScreenDisplayBuffer, "User height: %d", userHeight);
  printScreen(topScreenDisplayBuffer, TOP);
  if (userHeight > 195) {
    userHeight = 195;
  } else if (userHeight < 170) {
    userHeight = 170;
  }
  Serial.print("user height: ");
  Serial.println(userHeight);
}

void calculateSittingHeight(byte height) {
  sittingHeight = height - 60;
  Serial.print("user sitting height: ");
  Serial.println(sittingHeight);
}
```

```
void calculateStandingHeight(byte height) {
  standingHeight = height - 40;
  Serial.print("user standing height: ");
  Serial.println(standingHeight);
}

// lcd functions ***************************
void printScreen(char *str, byte line) {
  int len = 0;
  while (str[len++] != '\0'); // get length of string

  if (line == TOP) {
    top_str = str;
    top_len = len;
    top_mult = 0;
  } else {
    bot_str = str;
    bot_len = len;
    bot_mult = 0;
  }
}

void updateDisplay() {
  lcd.clear();
  // do top line
  lcd.setCursor(0, TOP);
  if (top_mult >= 0) {
    if (top_len <= 16) {
      lcd.print(top_str);
    } else {
      lcd.print(top_str + 16 * (top_mult++));
      if (16 * top_mult >= top_len) {
        top_mult = 0;
      }
    }
  } else {
    top_mult++;
  }
  // repeat for bottom line
  lcd.setCursor(0, BOT);
  if (bot_mult >= 0) {
    if (bot_len <= 16) {
      lcd.print(bot_str);
    } else {
      lcd.print(bot_str + 16 * (bot_mult++));
      if (16 * bot_mult >= bot_len) {
        bot_mult = 0;
      }
    }
```

```
  } else {
    bot_mult++;
  }
}

void setupLCD() {
  // initialize screen
  lcd.init();
  lcd.clear();
  lcd.backlight();

//  printScreen("Thank you for", TOP);
//  printScreen("using our standing desk", BOT);
}
```