

Mechatronics Final Report: WalkerAssist

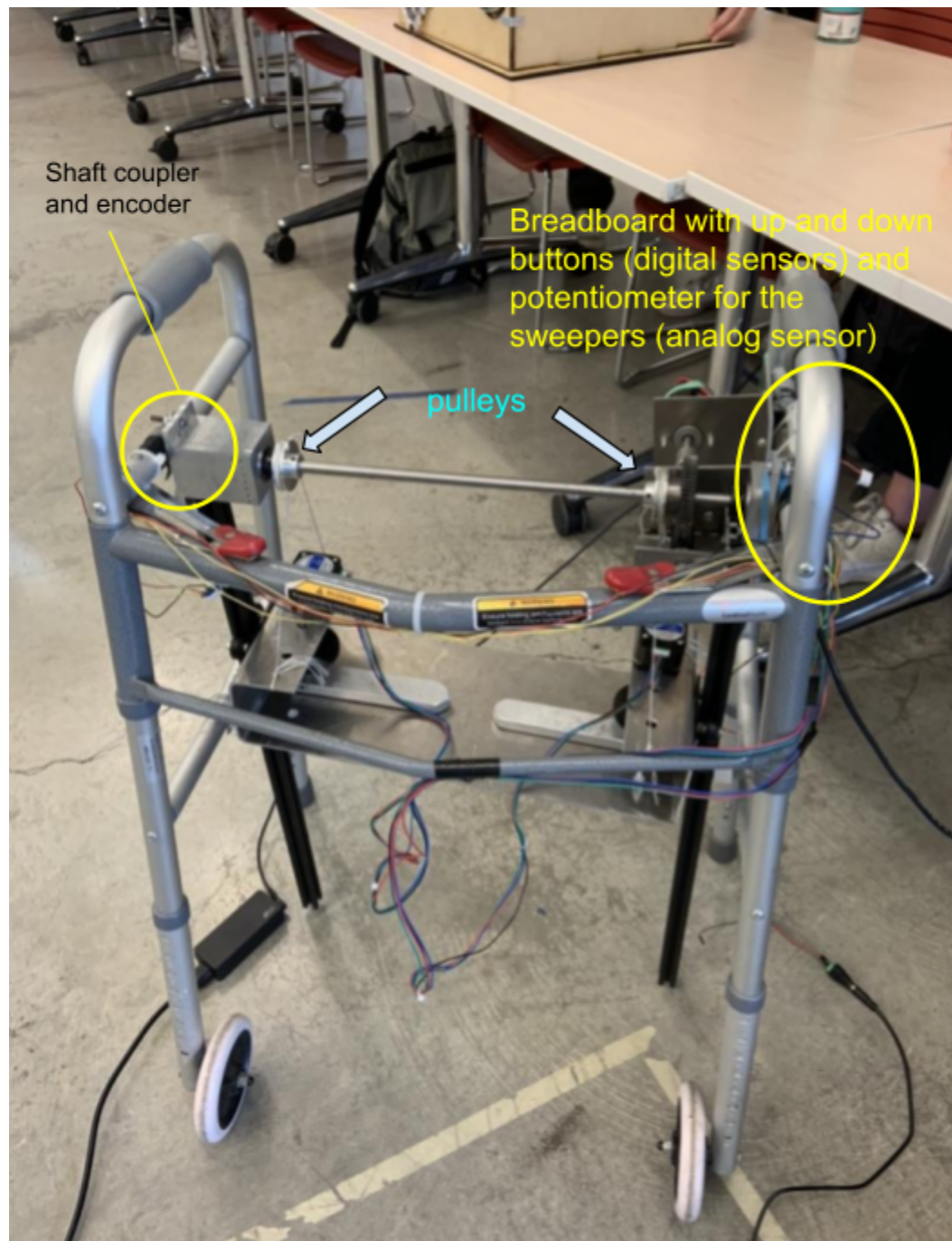
Opportunity:

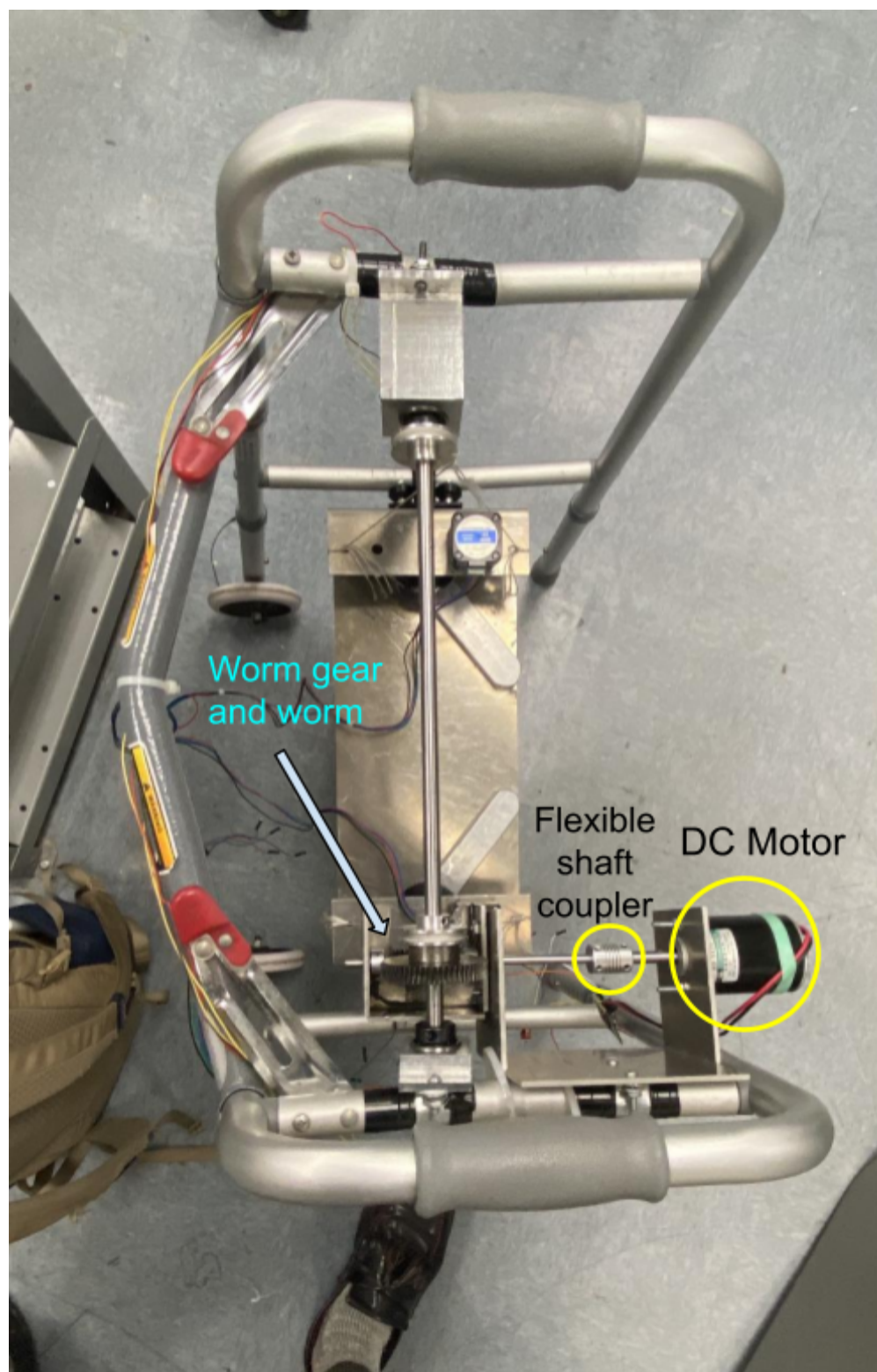
In our current version of the walker, our opportunity is very similar to the original opportunity we outlined in P2 in terms of functionality. We initially viewed this device as something that would seamlessly be integrated into any home. The device would be like a piece of furniture in the home as opposed to a device that is not a part of its natural environment. Ideally our product would have customizable skins / outer finishes to match the desired aesthetics of a home. The opportunity we identified and worked on to solve this semester with WalkerAssist is to increase elderly and disabled people's independence. Our device will allow for elderly people to place objects onto the platform on the walker and have those objects be lowered to the ground in a controlled manner and also be deposited onto the ground via the sweepers on the platform. The same would be true in reverse as well. The platform, via the sweepers, would be able to pull (or sweep) objects onto the platform and then raise them to a height for the user to have to bend minimally to pick up the object. This device would allow elderly and disabled people to be able to do more things around the house instead of relying on the aid of another person to help them.

Comparison:

In our initial ideation, we envisioned that this device would be integrated into the home, modular, and also have the platform and sweepers being able to move desired objects up, down, off and on. We will address each of these in detail. In regard to the integration into the users home, we saw the device being able to be used as a table and have the aesthetic features to be considered a piece of furniture. Our final prototype did not have any sort of aesthetic appeal due to our main focus on the functionality being correct. The aesthetic appeal could be made from laser cut wood that could be painted to hide the electrical components and mechanical mechanisms present in the device. For the modular aspect, we decided that to ensure proper functionality, the modular aspect would not be accounted for in our final design. To make our device modular would be difficult with our current design since we used bolts to attach our device to the walker. If we used some sort of secure clip design, our product could potentially be able to be easily taken on and off of the walker. For the base functionality of movement, we were able to hit our deliverable target entirely. We had a working platform that can lift and lower objects and also working sweepers that could push objects on and off of the platform. Our platform did not reach entirely to the top of the platform due to our design however in the future their design could be modified for more upwards movement ability.

Physical Device Diagram:

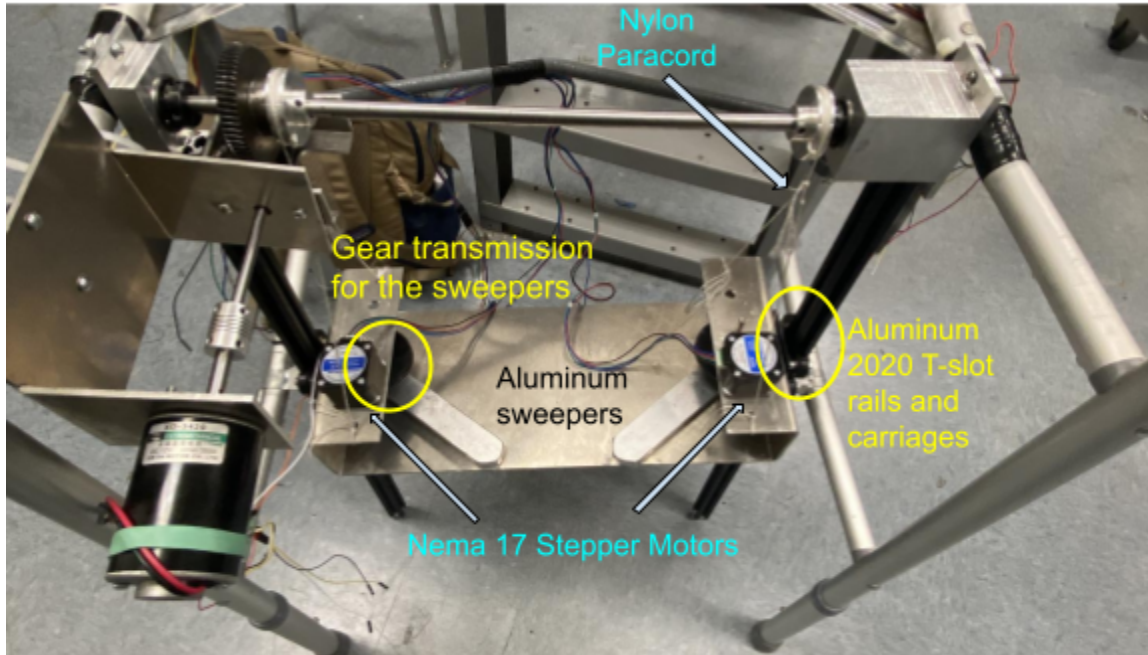




Worm gear
and worm

Flexible
shaft
coupler

DC Motor



Function-Critical Decisions:

The first function critical decision was the choice to use a worm gear transmission for our dc motor lift mechanism. Our standard motor would not be able to drive the main shaft on its own while simultaneously driving the platform, so we chose this transmission due to its high torque and lack of backdrivability.

Lifting Motor

Assume the shaft has radius $r = 3mm$.

What force do we need to lift?

The platform is $\approx 5lbs$ and a payload of 10 lbs \rightarrow 15 lbs to lift.

Gear transmission ratio between the worm and worm gear is 60:1.

Assume factor of safety of 2.

$$\begin{aligned}\tau_{in} * 60 &= F.O.S. * \tau_{lifting} \\ &= 2 * 3mm * 15lbs \\ &= 90mm \cdot lbs\end{aligned}$$

$$\tau_{in} = \frac{6}{4}mm \cdot lbs = 0.00668N \cdot m$$

Another critical design decision was our bearing sizing. The calculations are detailed below, but basically we wanted to make sure we properly sized our bearings for the amount of load that would be on them.

Assume l is 6 in.

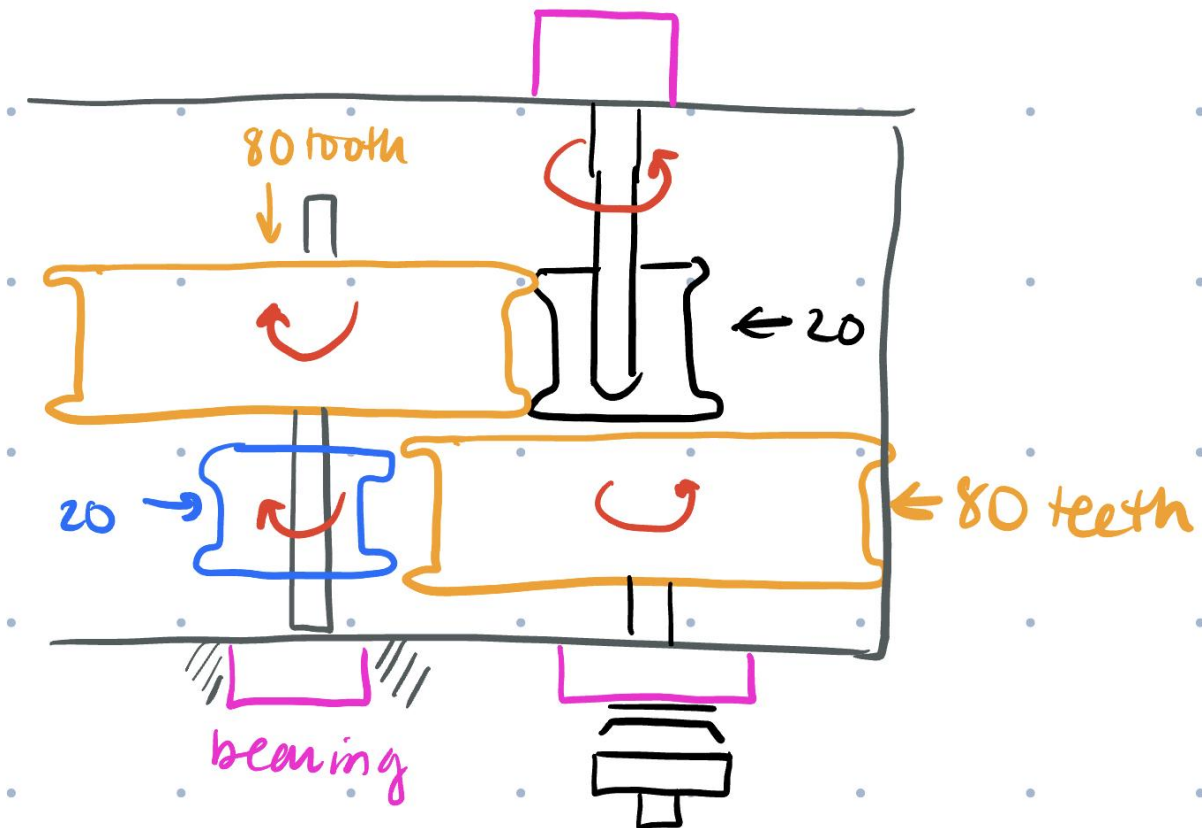
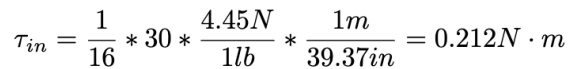
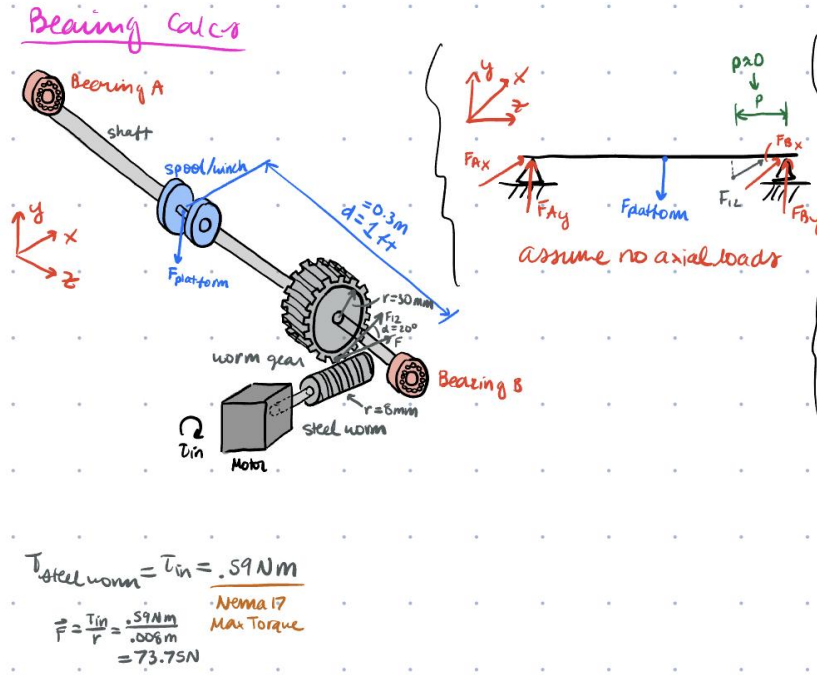


Diagram for reference



Bearing Calculations

$$F_{platform} = 15 \text{ lbs} = 67 \text{ N}$$

$$F_{12} = 73.75 \text{ N}$$

$$\sum F_x = 0 = F_{Ax} + F_{Bx} + F_{12}$$

$$\sum F_y = 0 = F_{Ay} + F_{By} - F_{platform}$$

$$\begin{aligned} \sum M_B = 0 &= (-2d\hat{z}) \times (F_{Ax}\hat{x} + F_{Ay}\hat{y}) + (-d\hat{z}) \times (-F_{platform}\hat{y}) + (\cancel{p}\hat{z} - r\hat{y}) \times (F_{12}\hat{x}) \\ &= -2dF_{Ax}\hat{y} + 2dF_{Ay}\hat{x} - dF_{platform}\hat{x} + r\hat{z} \end{aligned}$$

$$\hat{x} \cdot (-2dF_{Ax}\hat{y} - dF_{platform}\hat{x}) = 0$$

$$F_{Ay} = F_{platform}/2$$

$$\Rightarrow F_{By} = F_{platform}/2$$

$$\hat{y} \cdot (-2dF_{Ax}\hat{y}) = 0$$

$$F_{Ax} = 0$$

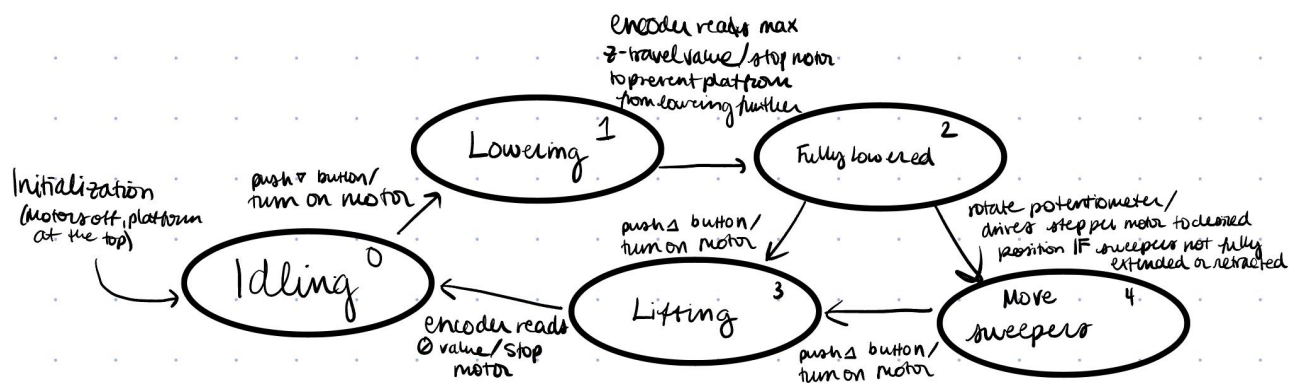
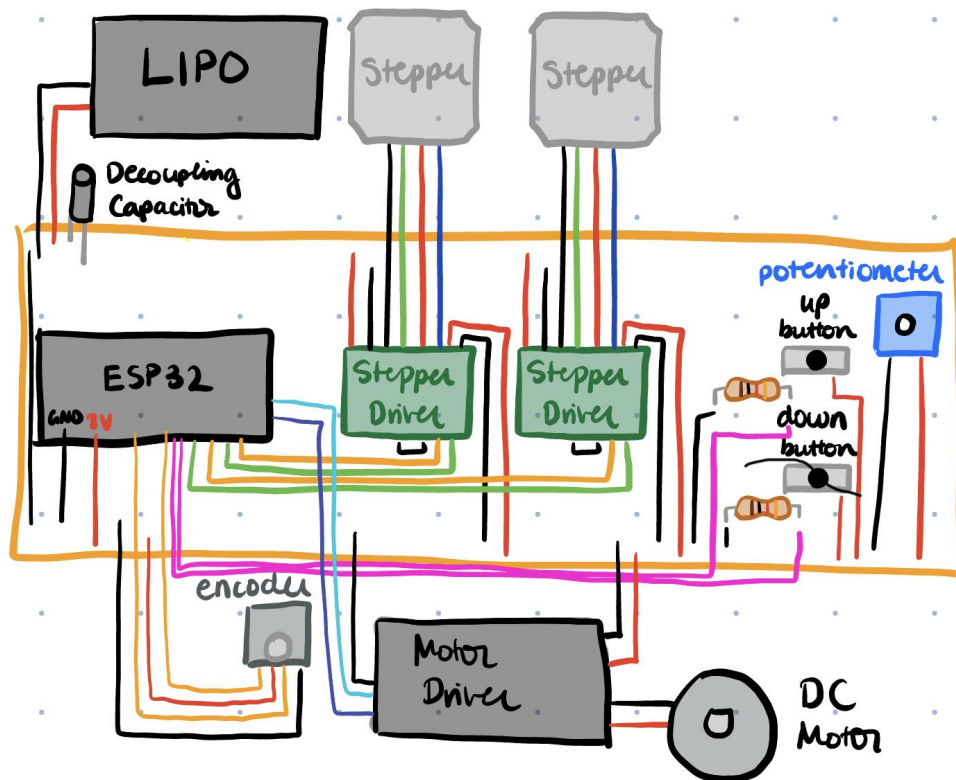
$$\Rightarrow F_{Bx} = 0$$

The next function critical design decision was the use of the v-slot rails and carriage. While our pulley could drive the platform up as the result of our digital input buttons, we had no way to lock the x and y-axes. With this implementation, the platform would be mounted to these carriages, locking it strictly in the z-axis. There

were no real calculations involved in this decision other than the measurements involved to align the rails with the assembly above and the platform itself.

The next function-critical design decision was the dc motor mount. We needed a way to mount the motor to the walker frame itself, so we elected to bend sheet metal and create several locating features to mount and align the motor with the worm gear transmission. The critical step in this process was switching to a flexible shaft coupling for the motor, allowing for some slop in the bending process. There were no major calculations in this process other than those that went into the hole positioning, as much of the alignment and mounting was made to be adjustable so we could position it by hand on our walker.

Circuit Diagram and State Transition Diagram:



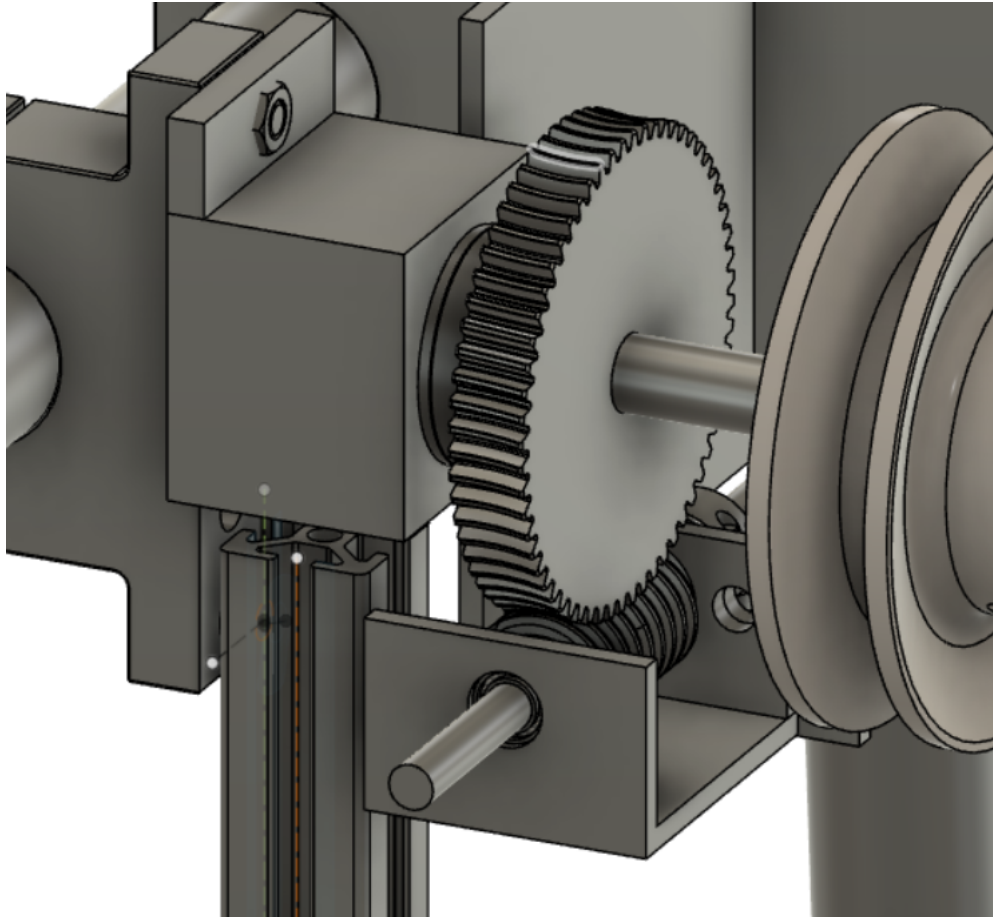
Reflection:

WalkerAssist was a momentous challenge from the inception of the project, and at this point in its progress we can be fully confident that it was a successful prototype. Though challenges arose from the complexity due to augmenting an existing device that was not intended to be augmented, our team was able to come up with adequate solutions that will serve as a backbone for all of our future work on this project. The electronics also serve as a strong base for future iterations, allowing for even more accessibility within the home for WalkerAssist's users.

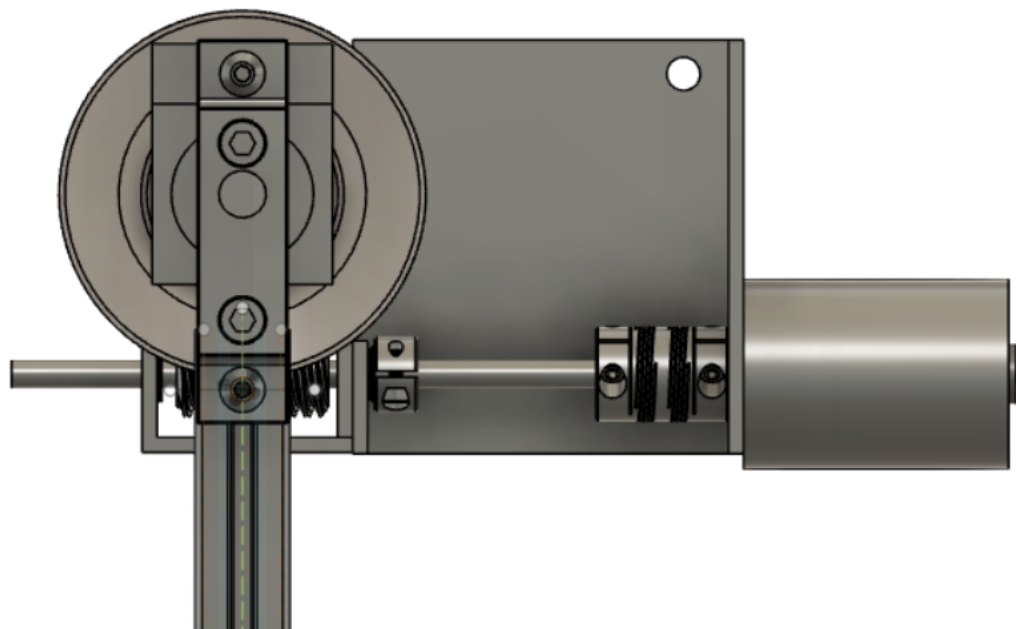
CAD Images



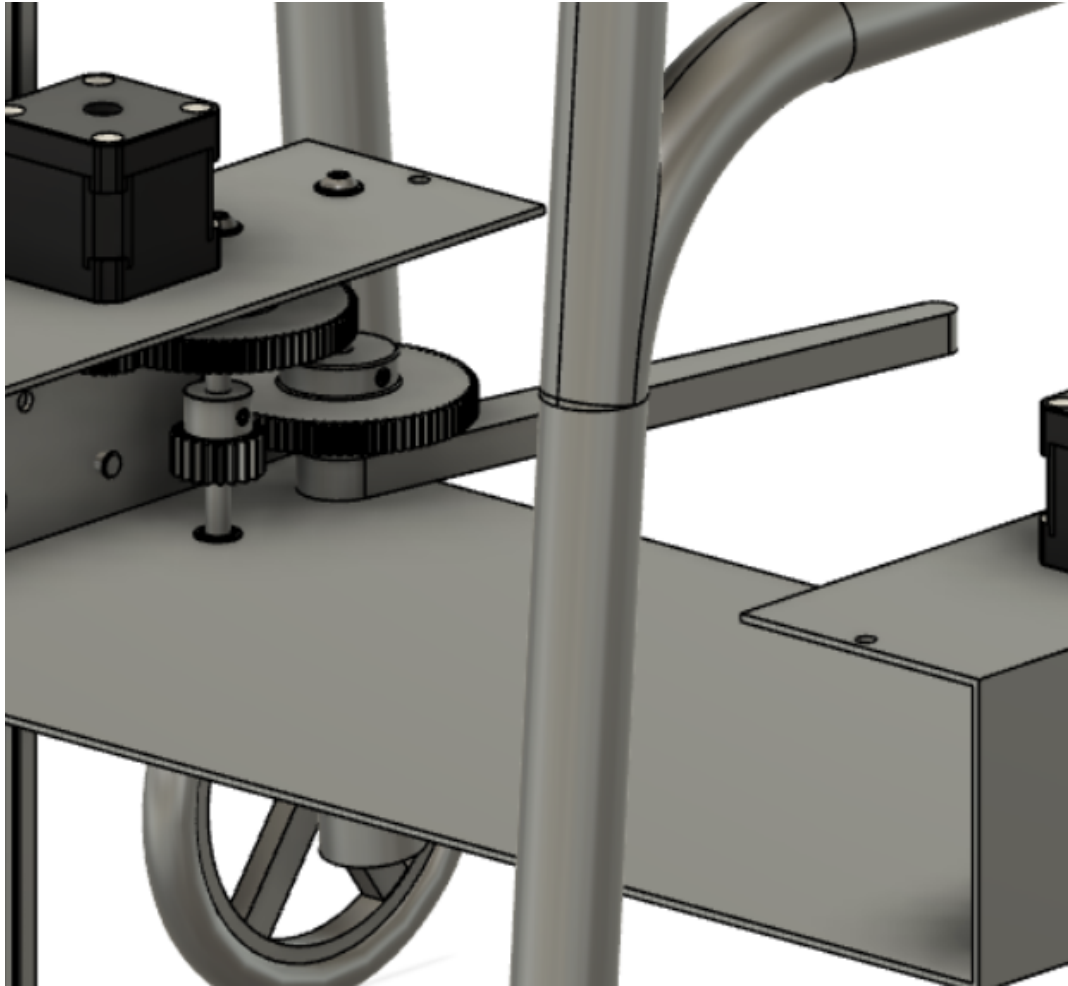
Isometric



Worm Gear Transmission



Side view of DC motor housing

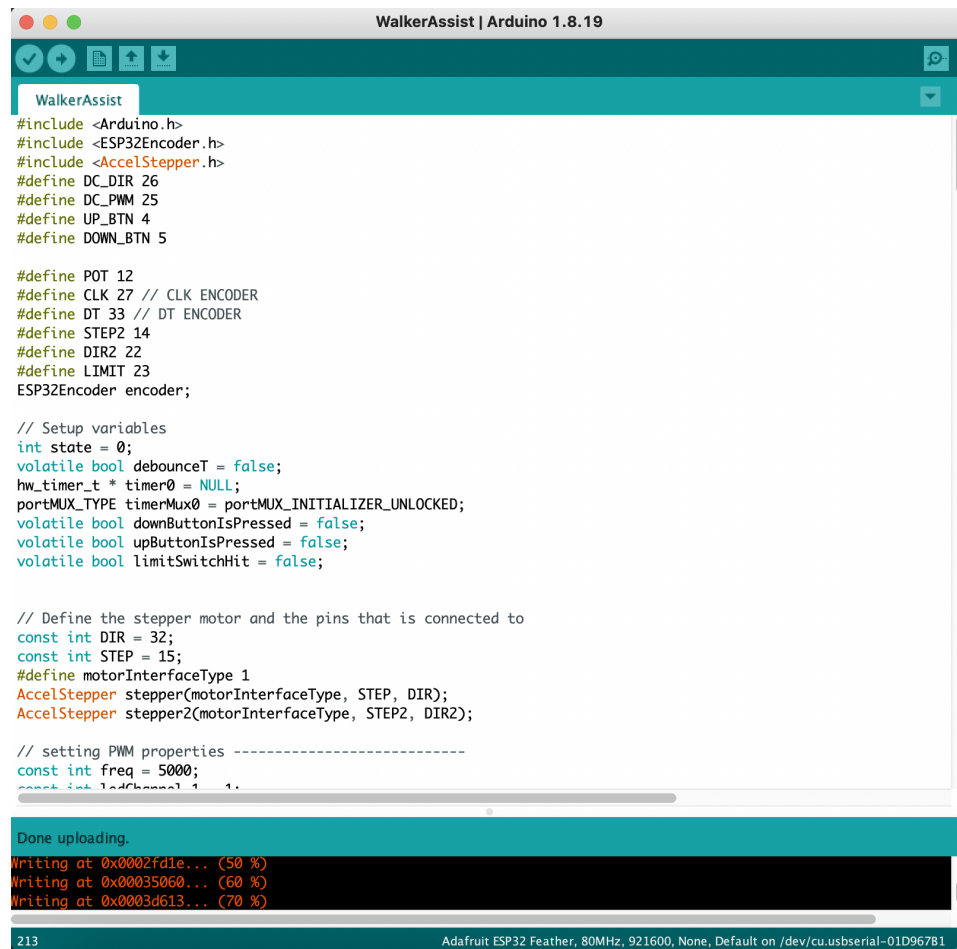


Platform Stepper Transmission

Bill of Materials:

<https://docs.google.com/spreadsheets/d/1E1pIgQ9XqCPWG0skxiDehvkyoKR7dqIrluxeeteut18/edit?usp=sharing>

Code:



```
WalkerAssist | Arduino 1.8.19

#include <Arduino.h>
#include <ESP32Encoder.h>
#include <AccelStepper.h>
#define DC_DIR 26
#define DC_PWM 25
#define UP_BTN 4
#define DOWN_BTN 5

#define POT 12
#define CLK 27 // CLK ENCODER
#define DT 33 // DT ENCODER
#define STEP2 14
#define DIR2 22
#define LIMIT 23
ESP32Encoder encoder;

// Setup variables
int state = 0;
volatile bool debounceT = false;
hw_timer_t * timer0 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
volatile bool downButtonIsPressed = false;
volatile bool upButtonIsPressed = false;
volatile bool limitSwitchHit = false;

// Define the stepper motor and the pins that is connected to
const int DIR = 32;
const int STEP = 15;
#define motorInterfaceType 1
AccelStepper stepper(motorInterfaceType, STEP, DIR);
AccelStepper stepper2(motorInterfaceType, STEP2, DIR2);

// setting PWM properties -----
const int freq = 5000;
const int ledChannel = 1;

Done uploading.
Writing at 0x0002fd1e... (50 %)
Writing at 0x00035060... (60 %)
Writing at 0x0003d613... (70 %)

213 Adafruit ESP32 Feather, 80MHz, 921600, None, Default on /dev/cu.usbserial-01D967B1
```

```
WalkerAssist
// setting PWM properties -----
const int freq = 5000;
const int ledChannel_1 = 1;
const int resolution = 8;
int MAX_PWM_VOLTAGE = 250;
int motor_PWM;
int potValue = 0;
int prevPotValue = 0;
int stepValue = 0;
int potResolution = 2048;
const int maxSweepingPosition = -215;
const int minSweepingPosition = 80;
const int maxZ_Travel = 420;
//200 / (10 * PI) * 1000; //Counts per rev * (1 rev / 10pi mm) * (1000mm travel) --> counts
int long encoderZ_Position = 0;
int sweepingPosition = 0;

//ISRs
void IRAM_ATTR onTime(){
  portENTER_CRITICAL_ISR(&timerMux0);
  debounceT = true;
  portEXIT_CRITICAL_ISR(&timerMux0);
  timerStop(timer0);
}

void IRAM_ATTR upButtonISR(){
  portENTER_CRITICAL_ISR(&timerMux0);
  upButtonIsPressed = true;
  timerStart(timer0);
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR downButtonISR(){
  portENTER_CRITICAL_ISR(&timerMux0);
  downButtonIsPressed = true;
```

```
WalkerAssist
void IRAM_ATTR downButtonISR(){
  portENTER_CRITICAL_ISR(&timerMux0);
  downButtonIsPressed = true;
  timerStart(timer0);
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR limitISR(){
  portENTER_CRITICAL_ISR(&timerMux0);
  limitSwitchHit = true;
  timerStart(timer0);
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void setup() {
  // PUSH BUTTONS
  attachInterrupt(UP_BTN, upButtonISR, RISING);
  attachInterrupt(DOWN_BTN, downButtonISR, RISING);

  //STEPPERS
  stepper.setMaxSpeed(1000);
  stepper.setAcceleration(60);
  stepper.setSpeed(200);

  stepper2.setMaxSpeed(1000);
  stepper2.setAcceleration(60);
  stepper2.setSpeed(200);
  stepper2.moveTo(200);

  //LIMIT SWITCH
  pinMode(LIMIT, INPUT);
  attachInterrupt(LIMIT, limitISR, RISING);

  // ENCODER
  encoder.attachHalfQuad ( DT, CLK );
  encoder.setPulse( 0 );
```



```
WalkerAssist
// ENCODER
encoder.attachHalfQuad ( DT, CLK );
encoder.setCount ( 0 );
Serial.begin ( 115200 );

//POTENTIOMETER
pinMode(POT, INPUT);
prevPotValue = analogRead(POT);

// configure LED PWM functionalites
ledcSetup(ledChannel_1, freq, resolution);

// attach the channel to the GPIO to be controlled
ledcAttachPin(DC_PWM, ledChannel_1);
pinMode(DC_DIR, OUTPUT);

// Idling State
state = 0;
stopPulley();

//Initialize Timer
TimerInterruptInit();
}

void loop() {
  delay(500);

  Serial.println("State: ");
  Serial.println(state);

  switch (state) {
    case 0: // IDLING
      if (checkForDownButtonPress()) {
        drivePulleyDown();
        state = 1;
      }
  }
```

```
WalkerAssist
    state = 1;
  }

  break;

  case 1: // LOWERING
    //ENCODER
    encoderZ_Position = encoder.getCount() / 2;
    Serial.println("Encoder value: ");
    Serial.println(encoderZ_Position);
    if (abs(encoderZ_Position) >= maxZ_Travel) {
      stopPulley();
      state = 2;
    }
    break;

  case 2: // FULLY LOWERED
    if (checkForUpButtonPress()) {
      drivePulleyUp();
      state = 3;
    }
    //POTENTIOMETER
    potValue = analogRead(POT);
    if (checkForMoveSweepers()) {
      moveSweepers();
      state = 4;
    }
  }

  break;

  case 3: //LIFTING
    //ENCODER
    encoderZ_Position = encoder.getCount() / 2;
    Serial.println("Encoder value: ");
    Serial.println(encoderZ_Position);
    if (abs(encoderZ_Position) <= 20) {
```

WalkerAssist

```
case 3: //LIFTING
//ENCODER
encoderZ_Position = encoder.getCount() / 2;
Serial.println("Encoder value: ");
Serial.println(encoderZ_Position);\
if (abs(encoderZ_Position) <= 20){
  Serial.print("Encoder at zero position");
  stopPulley();
  state = 5;
}

break;

case 4: // MOVE SWEEPERS
potValue = analogRead(POT);
if (checkForMoveSweepers()) {
  moveSweepers();
  state = 4;
}
if (checkForUpButtonPress()) {
  drivePulleyUp();
  setSweepersToZero();
  state = 3;
}

break;

case 5: // HOMING
//encoder.setCount(0);
limitSwitchHit = false;
state = 0;
break;
}
}
```

WalkerAssist

```
// Event Checkers -----

bool checkForUpButtonPress(){
  if (upButtonIsPressed && debounceT){
    debounceTimerReset();
    return true;
  }
  else {
    return false;
  }
}

bool checkForDownButtonPress(){
  if (downButtonIsPressed && debounceT){
    debounceTimerReset();
    return true;
  }
  else {
    return false;
  }
}

bool checkForMoveSweepers(){
  if (abs(potValue - prevPotValue) > potResolution) {
    return true;
  }
  else {
    return false;
  }
}

//Event Services -----
void drivePulleyUp() {
  upButtonIsPressed = false;
}
```

```
//Event Services -----
void drivePulleyUp() {
  upButtonIsPressed = false;
  Serial.println("Drive pulley up");
  ledcWrite(ledChannel_1, MAX_PWM_VOLTAGE);
  digitalWrite(DC_DIR, LOW);
  motor_PWM = MAX_PWM_VOLTAGE;
}

void drivePulleyDown() {
  downButtonIsPressed = false;
  Serial.println("Drive pulley down");
  ledcWrite(ledChannel_1, MAX_PWM_VOLTAGE);
  digitalWrite(DC_DIR, HIGH);
  motor_PWM = MAX_PWM_VOLTAGE;
}

void moveSweepers() {
  Serial.println("Move Sweepers");
  if ((potValue - prevPotValue) > potResolution){
    Serial.println("Pushing off the platform");
    stepper.moveTo(maxSweepingPosition);
    stepper2.moveTo(-maxSweepingPosition);
  }
  if ((potValue - prevPotValue) < -potResolution){
    Serial.println("Sweeping onto the platform");
    stepper.moveTo(minSweepingPosition);
    stepper2.moveTo(-minSweepingPosition);
    while (stepper.distanceToGo() != 0 && stepper2.distanceToGo() != 0) {
      stepper.run();
      stepper2.run();
    }
  }
}
```

```
stepper2.run();
}
prevPotValue = potValue;
}

void setSweepersToZero() {
  Serial.println("Setting sweepers to stowed position");
  stepper.moveTo(0);
  stepper2.moveTo(0);
  while (stepper.distanceToGo() != 0 && stepper2.distanceToGo() != 0) {
    stepper.run();
    stepper2.run();
  }
}

void stopPulley(){
  Serial.println("Stopping pulley");
  limitSwitchHit = false;
  ledcWrite(ledChannel_1, LOW);
  digitalWrite(DC_DIR, LOW);
}

void TimerInterruptInit() { //The timer simply counts the number of Tic generated by the quartz. With a quartz
  timer0 = timerBegin(0, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 = 1,000,000 tics
  timerAttachInterrupt(timer0, &onTime, true); // sets which function do you want to call when the interrupt
  timerAlarmWrite(timer0, 50000, true); // sets how many tics will you count to trigger the interrupt
  timerAlarmEnable(timer0); // Enables timer
  timerStop(timer0); //Pause timer to initialize debounce
}

void debounceTimerReset() {
  portENTER_CRITICAL(&timerMux0);
  debounceT = false;
  portEXIT_CRITICAL(&timerMux0);
  timerStop(timer0);
}
```