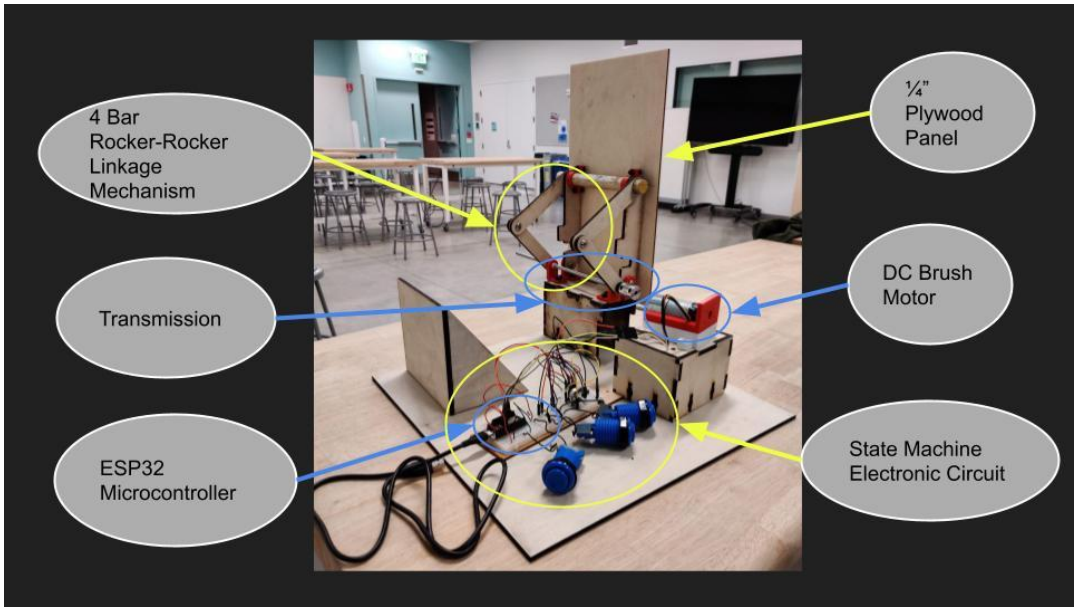
Bear Fold

ME102B - 2021| Bernardo Fichman Lacerda & Thomas Ragsac

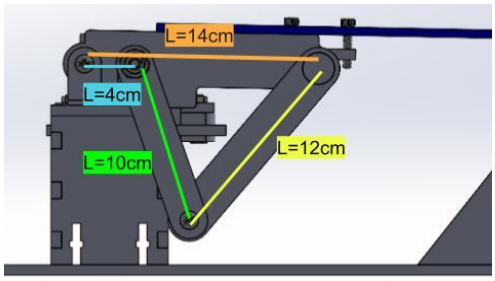


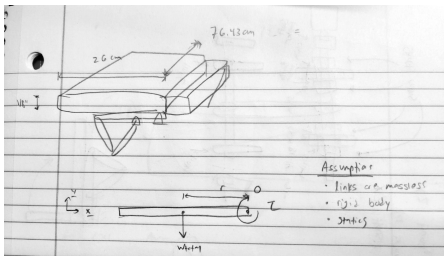
Bear Fold was a clothes folding machine created to make house chores easier and more fun. The original goal of our device was designed to have three different buttons for distinct clothing pieces: shirts, pants, and shorts. Each button would activate an unique sequence of rotations involving the three motors of the system, and each sequence was going to occur according to the type of cloth being folded. To achieve the folding motion, each motor is connected to a d-shaft to drive a double-crank four-bar linkage. The shaft would rotate a light weight panel where the cloth is going to be strategically positioned on. The amount of rotation done by each motor would be determined by a current sensor which commands the panel to come back to the initial position after achieving a maximum current. As a safety feature, if the current sensor detects excessive pressure applied to the panel, all motors would stop rotating and user input would be blocked until the excessive pressure was removed. A potentiometer connected to the system would modify how much torque would be applied by the motors, changing the speed of the clothes folding process and the strength of each folding crease.

Due to time and budget constraints, the project was reduced to one panel and one DC brush motor, to clearly demonstrate all the mechanical and electrical aspects of the project. In order to simulate the multiple patterns of folding, the panel would perform different movements for each button press. Since the system was reduced to one component, the outer housing was not assembled as intended. In addition, the current would have been measured using INA219 High Side DC Current Sensors, but due to difficulties in making the sensor interact with the ESP32, current sensing was not implemented on the project. In addition, the state diagram was not able to be fully integrated into the machine. We touch on this later in the document.

Function Critical Decisions

One of our concerns was having a transmission to exert a sufficient torque to lift the panels, while at the same time rotating the panels quick enough to prevent the clothes from falling or sliding off the panel. We decided to incorporate a linkage into the transmission. Linkages could provide mechanical advantage we required and could be customized to achieve the motion we desired. The best feasible mechanism achieved was a double-crank four bar linkage to achieve full rotation of a panel. It was created following Garchoff's rule with the smallest linkages fixed to the housing. The folding panels were laid on the 14cm linkages, connected by tight fit dowels and a d-shaft.





The motor we were given by Tom Clark from the Hesse Machine Shop was a *dfrobot Metal DC Geared Motor w/Encoder - 12V 251RPM.* The motor specifications are listed in the appendix section. Luckily, the torque requirement we needed was less than the 18 kg*cm stall torque. The approximated torque required to move the biggest panel was calculated as shown below:

Assumptions:

- *Linkages are massless, rigid body, statics*

$$\rho_{panel} = \rho_{birch\ plywood} = 700\ kg/m^3$$

Calculating Total Weight

$$w_{total} = w_{panel} + w_{clothes}$$

$$m_{panel} = \rho_{panel} * V_{panel}$$

$$m_{panel} = (700\ kg/m^3) * V_{Panel}$$

$$V_{panel} = 1/4\ inch * 76.43cm * 26cm$$

$$m_{panel} = 0.4417\ kg$$

$$w_{total} = 0.4417kg * 9.8m/s^2 = 4.3326\ N$$

Moment Balance

$$\Sigma M_o = 0 = -T_{required} + r * (w_{total})$$

$$T_{required} = r * (w_{total})$$

$$= (0.5 * width_{panel}) * (w_{total})$$

$$= (0.5 * 26 * 10^{-2}) * (4.3326N)$$

$$T_{required} = 0.5632\ N * m$$

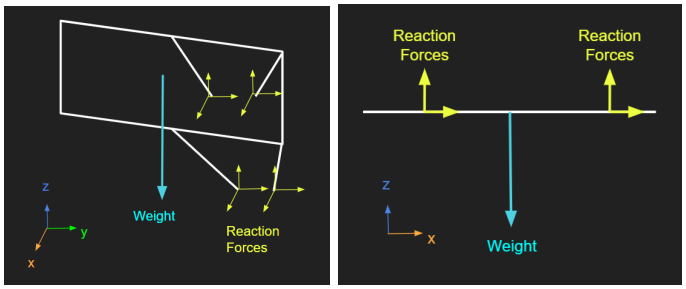
$$T_{required} = 5.74\ kg * cm$$

Deciding Transmission Parts

The motor was decided to be directly attached to the linkages via a d-shaft. The components in our transmission can be found in the cross section view of the CAD in the bottom of the document. The shaft coupler is used to couple the movement between the motor and the transmission system. A transmission is used to prevent excessive loading on the motor bearings internal of the motor. Two shaft collars are used to prevent axial movement of parts. We knew there would be no radial loading, so using friction would be sufficient. Nylon sleeve bearings are put in one of our 3D printed brackets. The choice of a sleeve bearing

was to allow smooth movement of the d-shaft, to transmit load to the transmission instead of the motor. More significantly, for savings since sleeve bearings do not require belleville washers or washers as they do not have ball bearings inside like ball bearings.

Calculations can be done for how much force will be transmitted on the sleeve bearings. We can reduce it from a 3D problem to a 2D problem.



Assumptions:

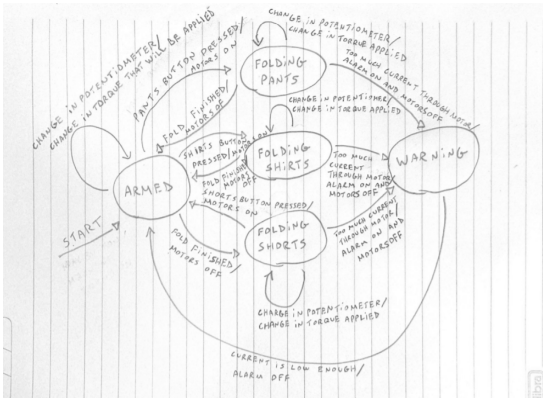
- *2D problem, linkages are massless, rigid body, statics*

$\Sigma Fx = 0$ $\Sigma Fy = 0 = F_{A1} + F_{A2} - W$ $\Sigma M_o = 0 = -\frac{L}{2} * W + L * F_{A2}$	$W\frac{L}{2} = L * F_{A2}$ $F_{A2} = \frac{W}{2}$ $0 = F_{A1} + F_{A2} - W$ $0 = F_{A1} + \frac{W}{2} - W$ $F_{A1} = \frac{W}{2}$

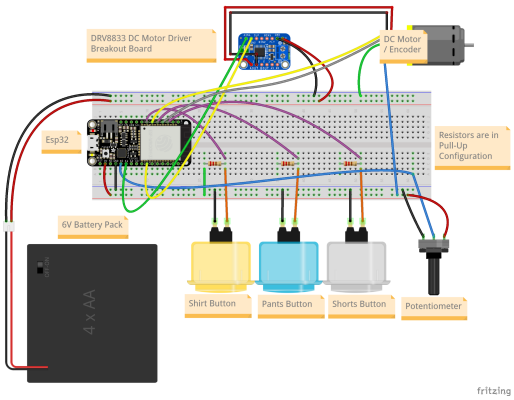
Coupling Mechanical Parts

Our d-shaft had to be constrained to the laser cut linkages. We did not want to use glue, so d shaped hole with a -0.08in from nominal dimensions was laser cut out from our linkages to create the tight, interference fit we needed. With a loose fit, the shaft would slip.

State Transition Diagram



Circuit Diagram



State Machine

We were able to program the buttons to activate changing modes from armed to Folding Shirts, Folding Pants, and Folding Shorts. We achieved this by creating a button isr() function that would then trigger a ___buttonPressCheck function to be true. We were able to activate a change in a potentiometer by implementing a checkPotentiomter() event checker and service routine.

We were not able to implement the folding of the motor. We were able to incorporate the motor rotating once in the clothes folding modes and we implemented PI control, yet incorporating both was not completed in time. The plan was to have the motor be position controlled using the encoder attached to the motor. The motor has a gear ratio of 43.8:1 and a full revolution of the output shaft equates to a 700 count. We planned to have the motor move 90 degrees, or 700/4 counts with position control. First, with the panel closed, the encoder would reset the count. Then the PI control would make the motor rotate up to 700 counts and oscillate at the open panel/clothes folded position. This would have happened until a timer finishes. Once the timer finishes the count would reset to 0 and then the motor would rotate to a -700 count and oscillate at the close panel/clothes placement position until a timer finishes. This sequence would keep happening until the number of folds per folding mode needed was complete.

We were also not able to completely incorporate current sensing to activate a warning mode. We had trouble having the ESP32 communicate with an I2C device if the ESP32 was flashed with Arduino. We were able to establish communication with an Arduino. After asking the GSI for a solution, we learned the solution is configuring the pins in the setup(). Unfortunately, we ran out of time.

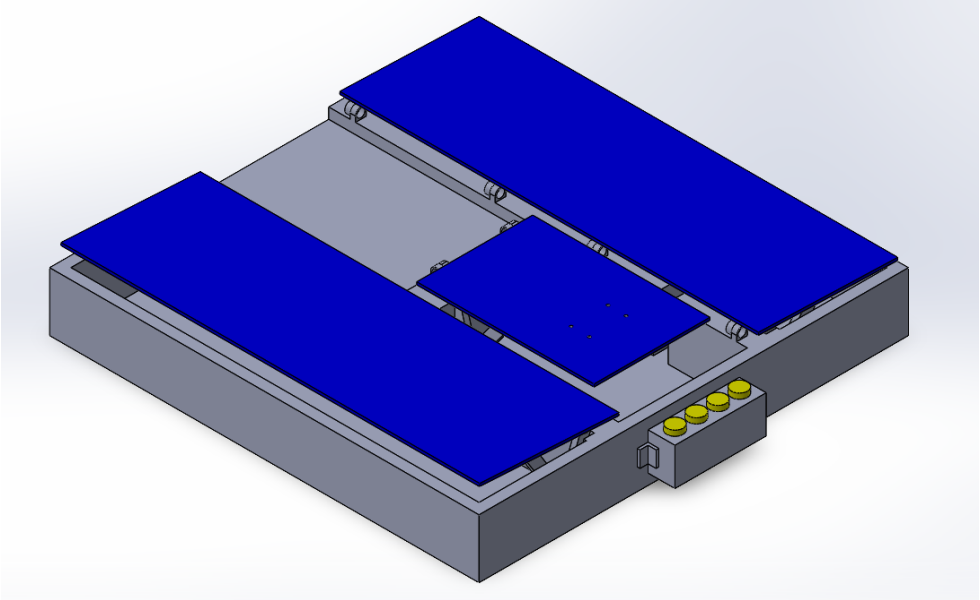
Reflections

During the project creation and development, a couple strategies were well implemented. Open communication and regular weekly meetings throughout the whole semester allowed us to work at a steady pace and prevented any group member from being overloaded or end up doing all the work by himself. In addition, the equal say on decisions and the sense of partnership created a healthy and pleasant working environment. Another successful strategy implemented was the constant collaboration and communication with people who had more knowledge and design experience than us. By seeking help from the professor, the GSIs, Tom Clark and Adam Hutz, the design specialist from the Jacobs lab, we were able to learn a lot and find solutions to our difficulties and save time.

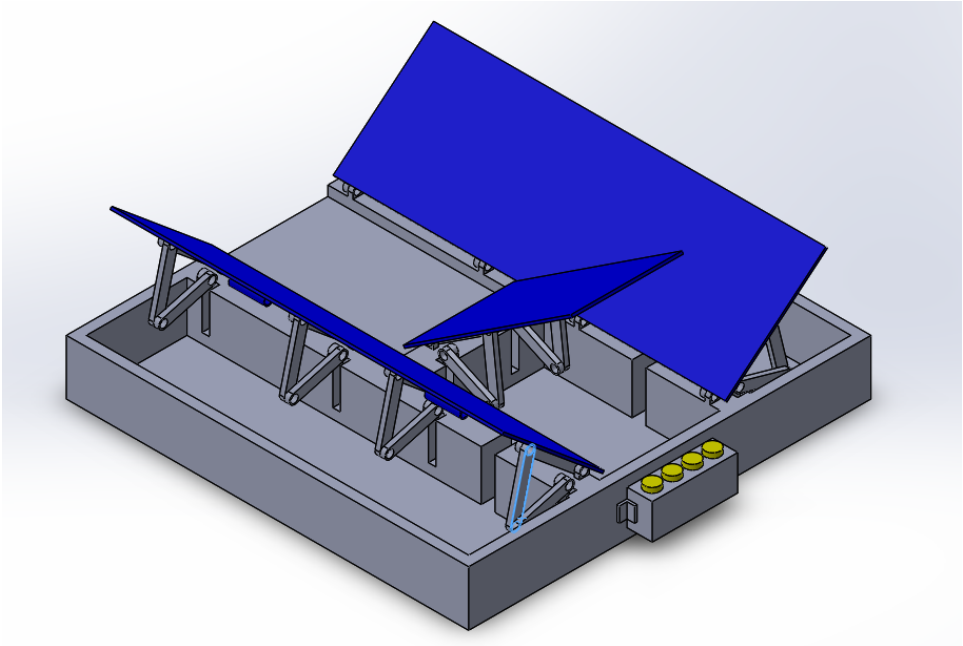
One of the things that we would do differently was the time management of the process; we spent too much time trying to design a mechanism, and that delayed the rest of the project considerably. We started the project with unrealistic ambitions, without knowing that even machines that seem simple can be complex during the creation process. At first, we conceptualized what the mechanism would do. When trying to make the mechanism work realistically with all necessary parts in CAD, it turned out to be very difficult. Even with help from those with design experience, designing a mechanism delayed our ability to order parts on time, prototype on time, and implement the state diagram with a working mechanical system on time. In addition, when implementing our CAD design and code for other aspects, we found we had not considered important details that we would have solved sooner if we ordered parts sooner. i.e. having all linkages attached rigidly and not interfere with each other required a new linkage design. Or constricting a d-shaft to the linkages required a tight tolerance interference fit, a fit found by iterative prototyping. Finally, manufacturing our whole physical product from scratch was time consuming and expensive. Buying elements ready or semi ready would have been a great alternative in the long run.

Images of CAD

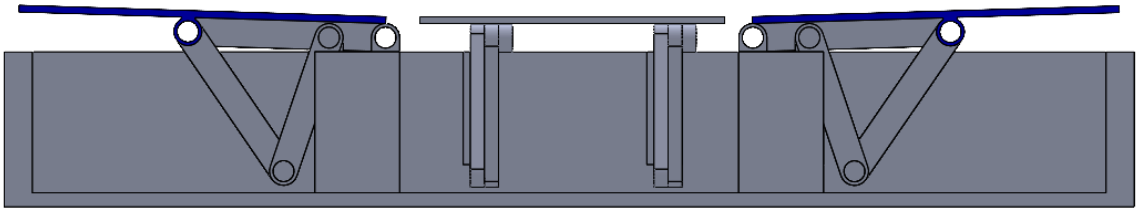
Full Concept CAD



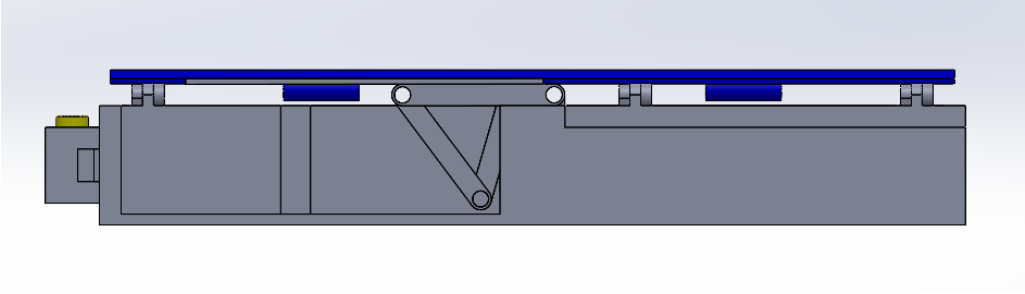
All Panels, Closed



All 3 Panels, Open Showing Mechanisms

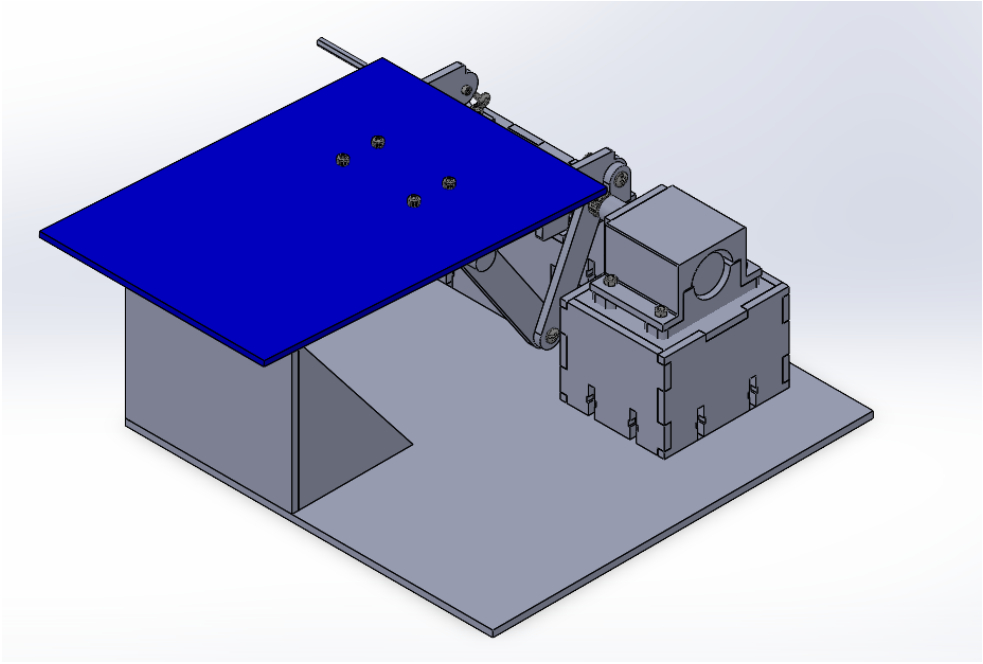


Front Cross Sectional View

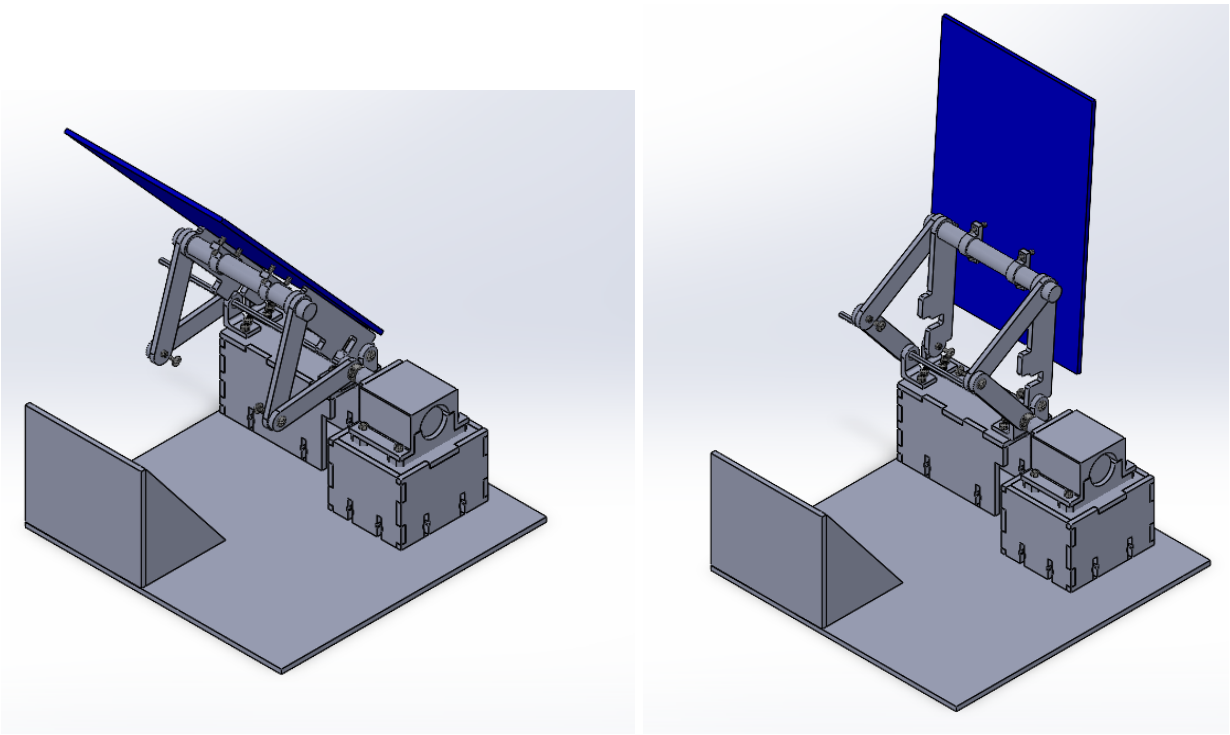


Side Cross Sectional View

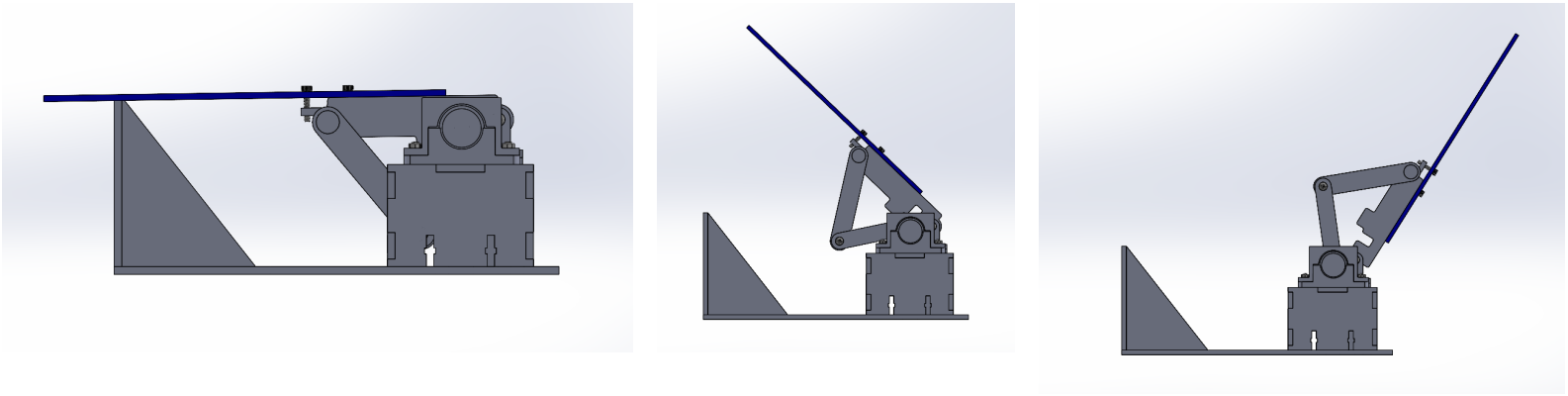
Fabrication CAD



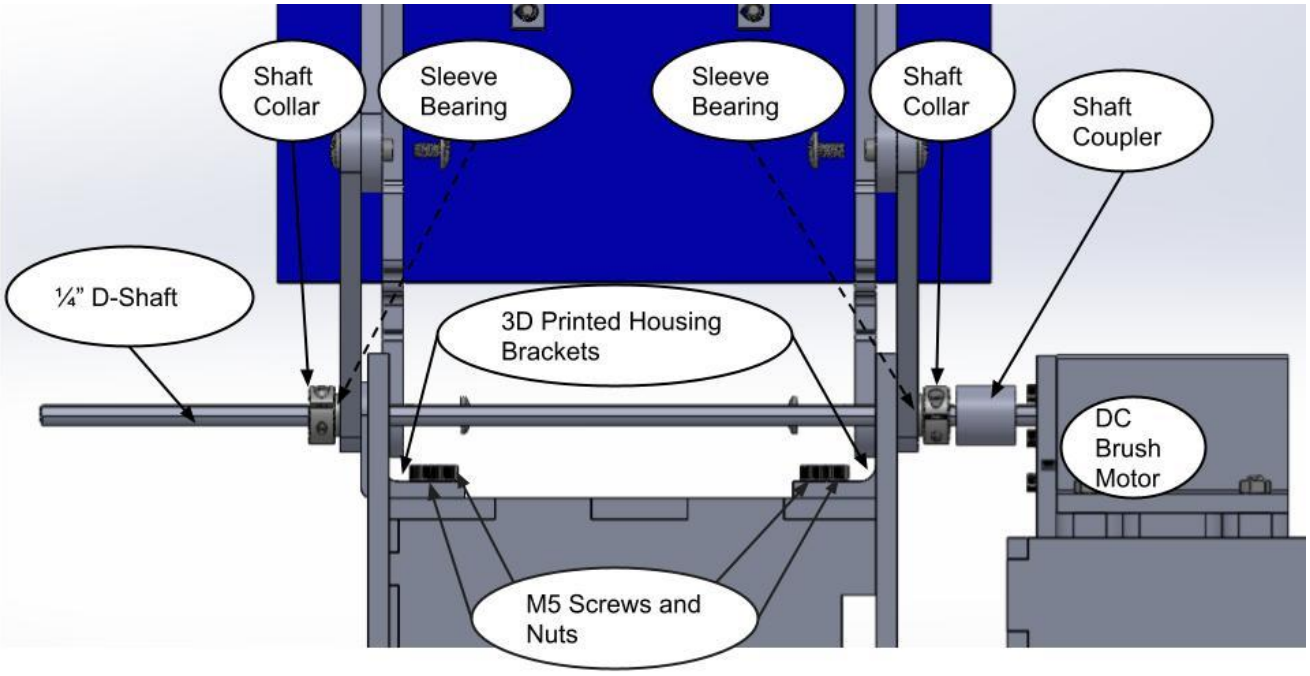
Isometric View, Panel, Linkages, Motor Island, Transmission, Panel Closed



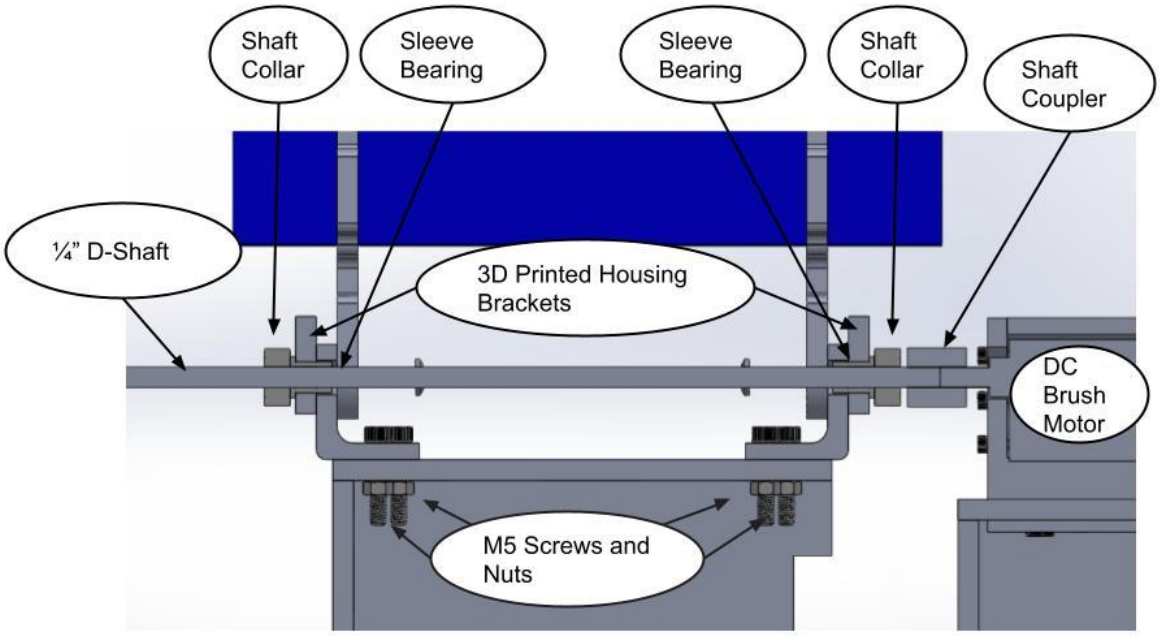
Isometric View, Panel, Linkages, Motor Island, Transmission, Panel Opening



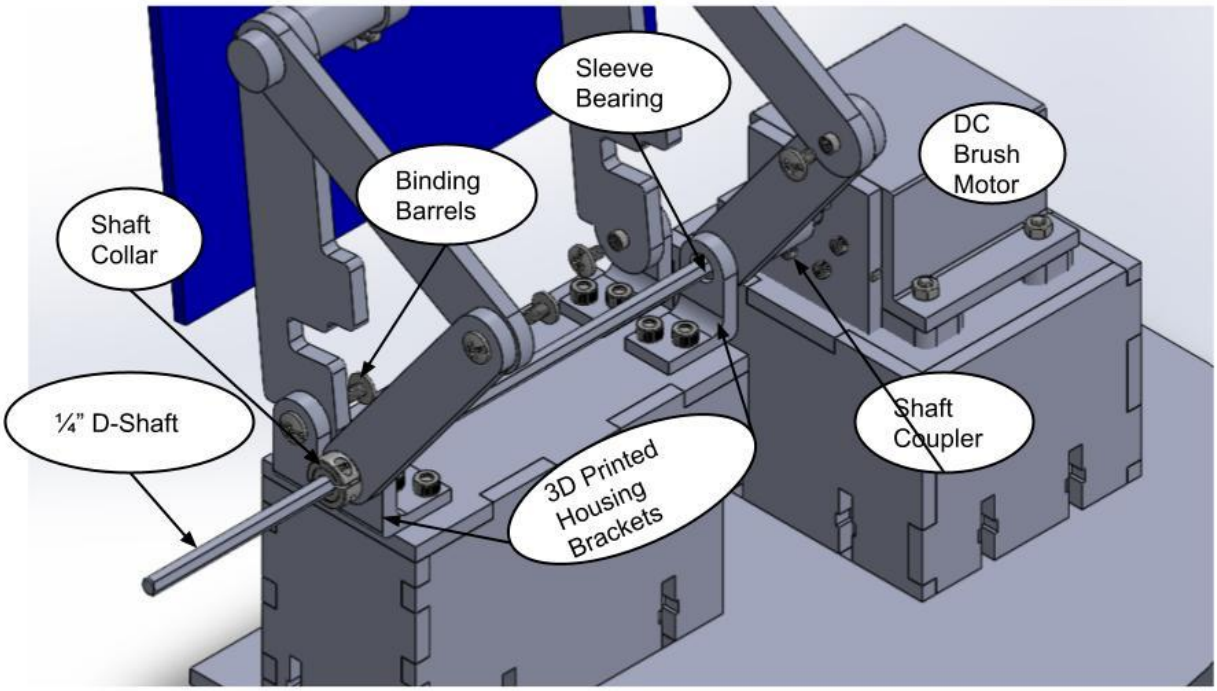
Side View, Linkages, Motor Island, Transmission, Panel Opening



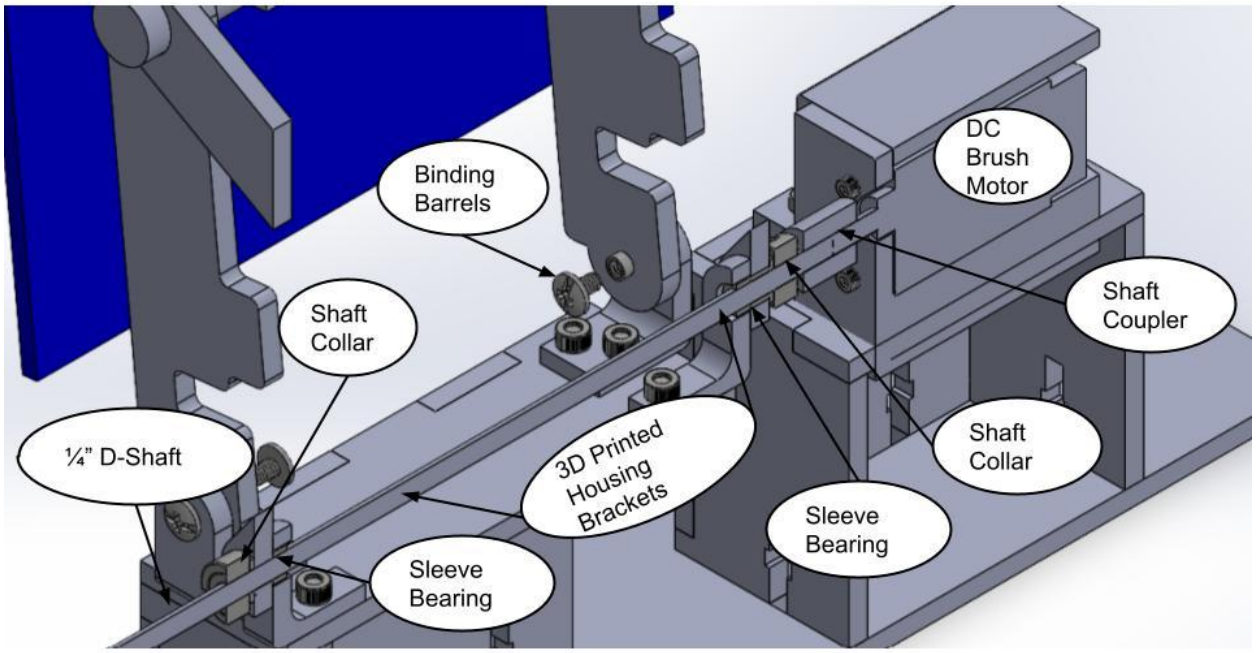
Labeled Front View of Transmission



Labeled Cross-Sectional View of Transmission



Labeled Isometric View of Transmission



Labeled Cross-Sectional Isometric View of Transmission

Screenshot of Code

Initialization Code

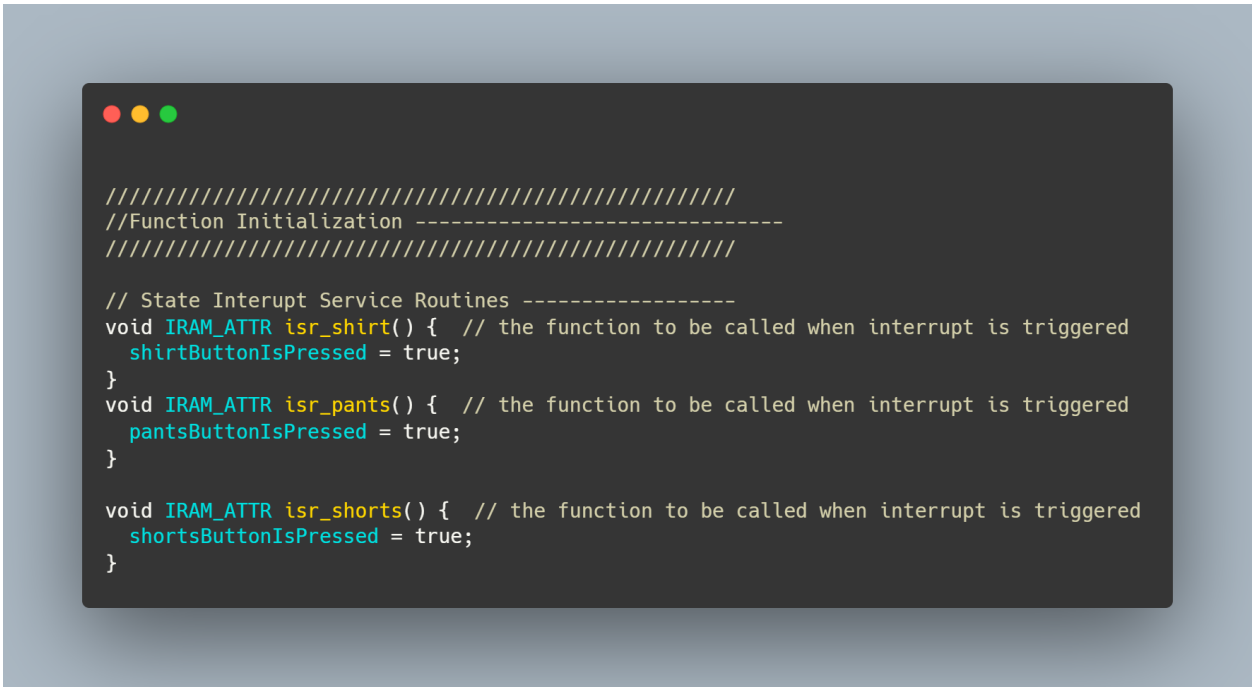
Define Pins

```
#include <Arduino.h>
#include <ESP32Encoder.h>

ESP32Encoder encoder;

//////////////////////////////////////////////////////
//Define constants --------------------------------
//////////////////////////////////////////////////////
#define BTN_SI 27  // Shirt button pin number
#define BTN_P 15  // Pants button pin number
#define BTN_SO 14  // Shorts button pin number
#define POT 26 // Potentiometer

#define BIN_1 25 // Motor Pin
#define BIN_2 4 // Motor Pin
```

Setup Variables

```
////////////////////////////////////////////////////
//Setup variables ---------------------------------
////////////////////////////////////////////////////

//Setup variables for velocity calculations------------
int initialCount = 0;
int finalCount = 0;
int deltaCount = 0;

float dT = 20 * (pow(10, -3)); // milli seconds
float omega = 0;
float numerator = 0;

// Potentiometer Variables --------------------------
int currentPotentiometerVal;
int previousPotentiometerVal;
int threshold = 4095 / 2; //threshold value for on/off
int offset = 25; //hysterisis offset
int upperThreshold = threshold + offset; //ipper Threshold
int lowerThreshold = threshold - offset; //lower Threshold

// Button Variables ---------------------------------
volatile bool shirtButtonIsPressed;
volatile bool pantsButtonIsPressed;
volatile bool shortsButtonIsPressed;

// Timer Variables ----------------------------------

// Timer for Button Bounce
volatile bool timerInterrupt;
hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

// Timers for Motor
volatile bool interruptCounter = false;    // check timer interrupt 1
volatile bool deltaT = false;     // check timer interrupt 2
int totalInterrupts = 0;   // counts the number of triggering of the alarm
hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

// setting PWM properties ---------------------------
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
int MAX_PWM_VOLTAGE = 255;
int currentPWMVoltage = 255;

int currentPWM = 0;
int D = 0;
bool direction = true;
int count;
int maxCount = 700;
int countDes = 42;
int countDesFold = 700 / 4;
int countDesReset = 0;
int Kp = 5;
int Ki = 8;
int error = 0;
int errorSum = 0;

// State Variables ----------------------------------
const int armedMode = 1; //Armed State
const int foldingShirtMode = 2; //Folding shirts State
const int foldingPantsMode = 3; //Folding pants State
const int foldingShortsMode = 4; //Folding shorts State
const int warningMode = 5; //Warning State
volatile int currentState = armedMode;
```

Defining Functions, isr for button press



```
////////////////////////////////////////////////////
//Function Initialization -----------------------------
////////////////////////////////////////////////////

// State Interupt Service Routines -----------------
void IRAM_ATTR isr_shirt() {  // the function to be called when interrupt is triggered
  shirtButtonIsPressed = true;
}
void IRAM_ATTR isr_pants() {  // the function to be called when interrupt is triggered
  pantsButtonIsPressed = true;
}

void IRAM_ATTR isr_shorts() {  // the function to be called when interrupt is triggered
  shortsButtonIsPressed = true;
}
```

Defining Functions, isr for timers

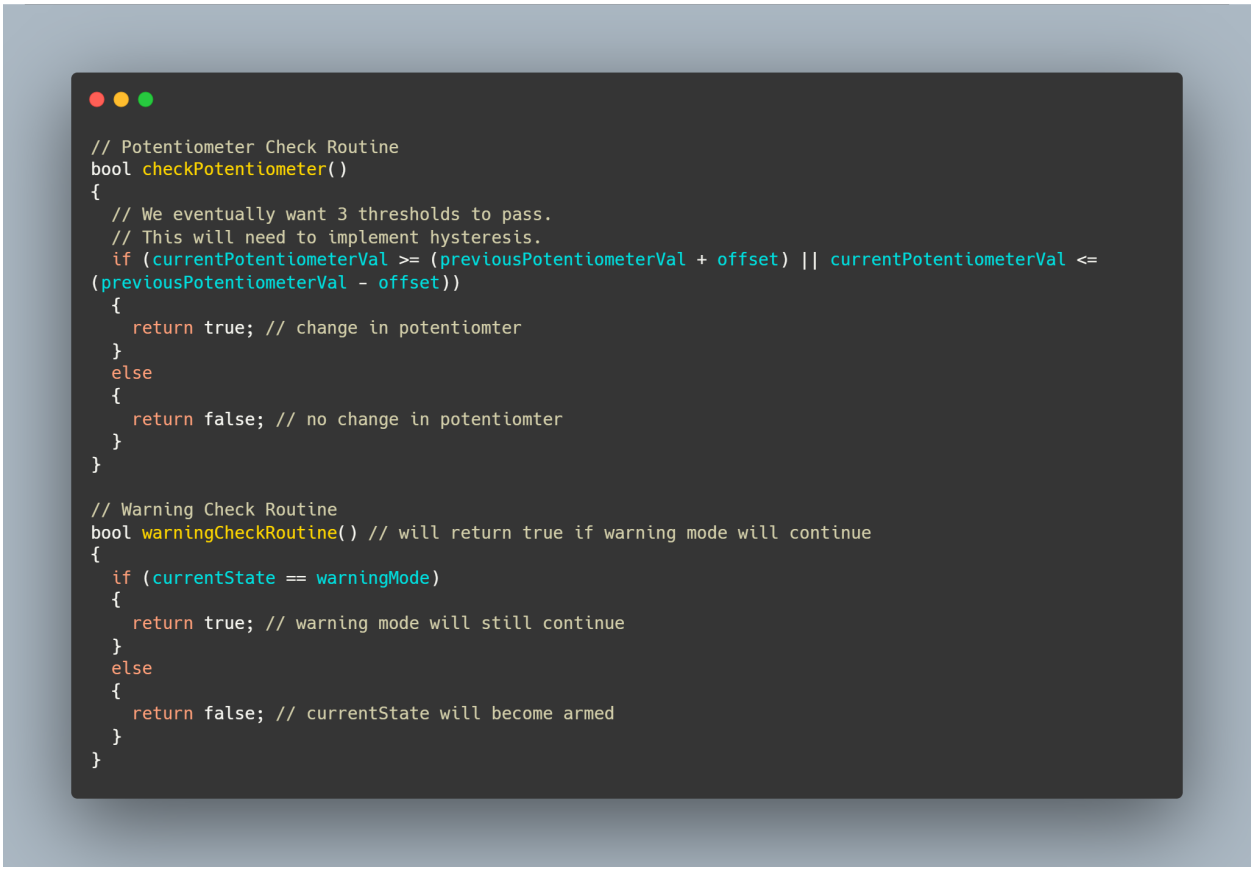
```
// Timer Functions ---------------------------------

//button bounce onTime()
void IRAM_ATTR onTime() // Code is run when timer is finished
{
  timerInterrupt = true; // this activates only when the timer has finished
  portENTER_CRITICAL_ISR(&timerMux);
  portEXIT_CRITICAL_ISR(&timerMux);
  timerStop(timer); // Stops the timer since we only want it to start when the button is pressed

  //Reset Button Press
  shirtButtonIsPressed = false;
  pantsButtonIsPressed = false;
  shortsButtonIsPressed = false;

  currentState = armedMode;
  Serial.println("Button Bounce Timer Finished.");
}

// Timers for motors
void IRAM_ATTR onTime0()
{
  portENTER_CRITICAL_ISR(&timerMux0);
  interruptCounter = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
}

// Timers for motors
void IRAM_ATTR onTime1()
{
  portENTER_CRITICAL_ISR(&timerMux1);
  deltaT = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

// Timer Initialization function for Button Bounce and Motor
void TimerInterruptInit()
{ //The timer simply counts the number of Tic generated by the quartz. With a quartz clocked at 80MHz,
we will have 80,000,000 Tics.

  // Button Bounce Timer Initialization
  timer = timerBegin(0, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 =
1,000,000 tics / sec
  timerAttachInterrupt(timer, &onTime, true);    // sets which function do you want to call when the
interrupt is triggered
  timerAlarmWrite(timer, 10000000, true); //5000, true);         // sets how many tics will you count
to trigger the interrupt

  timerAlarmEnable(timer); // Enables timer

  timerStop(timer); // Stops the timer since we only want it to start when the button is pressed
  timerInterrupt = true; // True means timer can restart. False means timer is still going.


  // Motor Timer Initialization
  timer0 = timerBegin(1, 80, true);  // timer 0, MWDT clock period = 12.5 ns *
TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer0, &onTime0, true); // edge (not level) triggered
  timerAlarmWrite(timer0, 2000000, true); // 2000000 * 1 us = 2 s, autoreload true

  timer1 = timerBegin(2, 80, true);  // timer 1, MWDT clock period = 12.5 ns *
TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
  timerAlarmWrite(timer1, 20000, true); // 20 ms

  // at least enable the timer alarms
  timerAlarmEnable(timer0); // enable
  timerAlarmEnable(timer1); // enable
}
```

Setup()

```
////////////////////////////////////////////////
//SETUP----------------------------------------
////////////////////////////////////////////////

void setup()
{
  // put your setup code here, to run once:
  pinMode(BTN_SI, INPUT);
  pinMode(BTN_P, INPUT);
  pinMode(BTN_SO, INPUT);
  pinMode(POT, INPUT);
  Serial.begin(115200);

  // Motor Initialization ----------
  ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up resistors
  encoder.attachHalfQuad(32, 33); // Attache pins for use as encoder pins
  encoder.setCount(0);  // set starting count value after attaching
  encoder.clearCount();

  // configure LED PWM functionalitites
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(BIN_1, ledChannel_1);
  ledcAttachPin(BIN_2, ledChannel_2);

  //Interupt Initialization----------

  //ISR Button Initialization
  volatile bool shirtButtonIsPressed = false; // starts false because button is not pressed yet
  volatile bool pantsButtonIsPressed = false; // starts false because button is not pressed yet
  volatile bool shortsButtonIsPressed = false; // starts false because button is not pressed yet
  attachInterrupt(BTN_SI, isr_shirt, RISING);  // set the "BTN" pin as the interrupt pin; call function
named "isr" when the interrupt is triggered; "Rising" means triggering interrupt when the pin goes from
LOW to HIGH
  attachInterrupt(BTN_P, isr_pants, RISING);  // set the "BTN" pin as the interrupt pin; call function
named "isr" when the interrupt is triggered; "Rising" means triggering interrupt when the pin goes from
LOW to HIGH
  attachInterrupt(BTN_SO, isr_shorts, RISING);  // set the "BTN" pin as the interrupt pin; call
function named "isr" when the interrupt is triggered; "Rising" means triggering interrupt when the pin
goes from LOW to HIGH

  //Timer Initialization
  TimerInterruptInit(); // Initiates timer intterupt

  //Potentiometer Initialization
  int currentPotentiometerVal = 0;
  int previousPotentiometerVal = 0;

}
```

```
//Main loop ---------------------------------
void loop()
{
  // Get Potentiometer Values
  previousPotentiometerVal = currentPotentiometerVal;
  currentPotentiometerVal = analogRead(POT);
  currentPWM = map(analogRead(POT), 0, 4095, 0, MAX_PWM_VOLTAGE);

  switch (currentState)
  {
    case armedMode:
      //Reset Motor
      //MotorReset();
      controlPositionReset();

      //Event Checker for Shirt Button Press
      if (CheckForButtonPress_Shirt())
      {
        ButtonResponse_Shirt();

      }

      //Event Checker for Pants Button Press
      if (CheckForButtonPress_Pants())
      {
        ButtonResponse_Pants();
      }

      //Event Checker for Shorts Button Press
      if (CheckForButtonPress_Shorts())
      {
        ButtonResponse_Shorts();
      }

      //Event Checker for Potentiometer Change
      if (checkPotentiometer())
      {
        //changePotentiometerService();
      }
      break;

    case foldingShirtMode:
      shirtMotorRoutine();
      break;

    case foldingPantsMode:
      pantsMotorRoutine();
      break;

    case foldingShortsMode:
      shortsMotorRoutine();
      break;

    case warningMode:
      break;

    default:
      break;
  }
}
```

```
// ------------------------------------------------
// Event Checker & Service Routine Code ---------------
// ------------------------------------------------

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

// Button Check Routine ----------------------------------
bool CheckForButtonPress_Shirt()
{
  // check if the button is pressed
  if (shirtButtonIsPressed == true)
  {
    shirtButtonIsPressed = 0; //reset condition
    // check if the timer is finished
    if (timerInterrupt == true)
    {
      timerInterrupt = false; // reset happens once button is pressed
      return true;
    }
    else
    {
      return false;
    }
  }
  else
  {
    return false;
  }
}

bool CheckForButtonPress_Pants()
{

  // check if the button is pressed
  if (pantsButtonIsPressed == true)
  {
    // check if the timer is finished
    if (timerInterrupt == true)
    {
      timerInterrupt = false; // reset happens once button is pressed
      return true;
    }
    else
    {
      return false;
    }
  }
  else
  {
    return false;
  }
}

bool CheckForButtonPress_Shorts()
{

  // check if the button is pressed
  if (shortsButtonIsPressed == true)
  {
    // check if the timer is finished
    if (timerInterrupt == true)
    {
      timerInterrupt = false; // reset happens once button is pressed
      return true;
    }
    else
    {
      return false;
    }
  }
  else
  {
    return false;
  }
}
```



```
// Potentiometer Check Routine
bool checkPotentiometer()
{
  // We eventually want 3 thresholds to pass.
  // This will need to implement hysteresis.
  if (currentPotentiometerVal >= (previousPotentiometerVal + offset) || currentPotentiometerVal <=
(previousPotentiometerVal - offset))
  {
    return true; // change in potentiomter
  }
  else
  {
    return false; // no change in potentiomter
  }
}

// Warning Check Routine
bool warningCheckRoutine() // will return true if warning mode will continue
{
  if (currentState == warningMode)
  {
    return true; // warning mode will still continue
  }
  else
  {
    return false; // currentState will become armed
  }
}
```

```
////////////////////////////////////////////////////////
// Service Routine Functions -----------------------------
////////////////////////////////////////////////////////

// Button -----------------------------
void ButtonResponse_Shirt()
{
  currentState = foldingShirtMode;
  Serial.print("State: ");
  Serial.println("Shirt");

  //timer restarts only when the button is pressed
  //  & the timer has already finished
  timerRestart(timer);
}

void ButtonResponse_Pants()
{
  currentState = foldingPantsMode;
  Serial.print("State: ");
  Serial.println("Pants");

  //timer restarts only when the button is pressed
  //  & the timer has already finished
  timerRestart(timer);
}

void ButtonResponse_Shorts()
{
  currentState = foldingShortsMode;
  Serial.print("State: ");
  Serial.println("Shorts");

  //timer restarts only when the button is pressed
  //  & the timer has already finished
  timerRestart(timer);
}
```

```
// Motor Service Routines -------------------------

void shirtMotorRoutine()
{
  displayCount();
  controlPositionFold();
}

void pantsMotorRoutine()
{
  displayCount();
  MotorFold();
}

void shortsMotorRoutine()
{
  displayCount();
  MotorFold();
}

void changePotentiometerService()
{
  //D = map(analogRead(POT), 0, 4095, -NOM_PWM_VOLTAGE, NOM_PWM_VOLTAGE);
  // change the torque that will br applied to the motor
}

// Warning Service Routine
void warningServiceRoutine()
{
  Serial.println("Speed,Warning: Too much current through motor");
  // reset all motors to original position
  // turn off all motors
  // activate speaker
}
```

```
// Motor Move Service ----------------
void displayCount()
{
  count = encoder.getCount();
  Serial.println("Count, countDes, error, D, Time, PWM: ");
  Serial.print(count);
  Serial.print(",");
  Serial.print(countDes);
  Serial.print(",");
  Serial.print(error);
  Serial.print(D);
  Serial.print(",");
  Serial.print(timerReadSeconds(timer));
  Serial.print(",");
  Serial.println(currentPWM);
}

void controlPositionFold()
{
  countDes = countDesFold;
  error = countDes + count; // count error

  D = Kp * error + (1) * Ki * errorSum; // includes P and I term
  // the scaling ratio 1/100 is to make
  // Ki a decimal value

  // Make the motor move if there is still error
  if (D >= 0)
  {
    // move CW, D -, Count -
    ledcWrite(ledChannel_2, LOW);
    ledcWrite(ledChannel_1, currentPWM);

  }
  if (D <= 0)
  {
    //move CCW, D +, Count +
    ledcWrite(ledChannel_2, currentPWM);
    ledcWrite(ledChannel_1, LOW);
  }

  if (D == 0)
  {
    ledcWrite(ledChannel_2, LOW);
    ledcWrite(ledChannel_1, LOW);
  }

  //Ensure that you don't go past the maximum possible command
  if (currentPWM > MAX_PWM_VOLTAGE)
  {
    currentPWM = MAX_PWM_VOLTAGE;
  }
  else if (currentPWM < -MAX_PWM_VOLTAGE) {
    currentPWM = -MAX_PWM_VOLTAGE;
  }
}
```

BOM

Bear Fold's Purchase Portfolio								Total (Projected):	$ 152.92
Item Name	Description	Purchase Justification	Serial Number / SKU	Price (ea.)	Quantity	Vendor	Link to Item	Notes	Subtotal
									$ -
Plywood - 1/4" x 12" x 24" (Variable)	Plywood for laser cutting	Reason 1: The exterior that will provi	No Serial Number	$ 1.67	2	Jacobs Institute	https://store.ja	Will be used for both the exterior shell c	$ 3.34
Plywood - 1/4" x 24" x 48"	Plywood for laser cutting	Reason 1: The exterior that will provi	No Serial Number	$ 13.32	2	Jacobs Institute	https://store.ja	Will be used for both the exterior shell c	$ 26.64
Steel Hex Nut, Medium-Strength, Class 8, M5 x 0.8 mm Thread, Packs of 100	Nuts for M5 screws	These are to fasten the m5 screws.	90592A095	$ 1.76	1	McMaster Carr	https://www.mc		$ 1.76
Black-Oxide Alloy Steel Socket Head Screw, M3 x 0.5 mm Thread, 8 mm Long, Pack of 100	M3 Screws	M3 Screws for motor mount	91290A118	$ 8.65	1	McMaster Carr	https://www.mc		$ 8.65
Black-Oxide Alloy Steel Socket Head Screw, M5 x 0.8 mm Thread, 25 mm Long, Packs of 50	M5 screws	M5 screws are used to fasten the later cut panels	91290A282	$ 9.34	1	McMaster Carr	https://www.mc	First purchase of m5 screws	$ 9.34
18-8 Stainless Steel Binding Barrel and Screw, 10-24 Thread Size, for 1/4"-3/8" Material Thickness	1/4"-3/8" Binding Posts	Binding Posts to assemble linkages.	99537A301			McMaster Carr	https://www.mc	First purchase of binding posts. Different size binding posts were later purchased.	$ 33.00
Black-Oxide Alloy Steel Socket Head Screw, M5 x 0.8 mm Thread, 25 mm Long, Packs of 50	M5 screws	Bought because we needed 50 more screws	91290A282	$ 1.65	20	McMaster Carr	https://www.mc	Second purchase of m5 screws. We found out we needed more than the 50 originally purposed screws.	$ 9.34
				$ 9.34	1				
Sealegend Shirt Folding Board Shirt Folder Clothes Folding Board Durable Plastic T-Shirts Clothes L	A flexible plastic panel used to fold	We used this as a model for our des	N/A	$ 9.99	1	Amazon	https://www.ar		$ 9.99
Hardwood Rounds (Common: 3/4 in. x 4 ft.; Actual: 0.75 in. x 0.75 in. x 48 in.)	3/4" x 48" Pine dowels	To connect linkages together	148229	$ 3.79	3	Home Depot	https://www.ho	Second 3/4 dowles purchased	$ 11.37
18-8 Stainless Steel Binding Barrel and Screw, 10-24 Thread Size, for 5/8"-3/4" Material Thickness	5/8-3/4 Binding Posts	Binding Posts to assemble linkages.	99537A304	$ 1.78	4	McMaster Carr	https://www.mc	Second purchase of binding posts. This	$ 7.12
D-Profile Ebony Shaft, 303 Stainless Steel, 1/4" Diameter, 12" Long	1/4" D-shaft L=12"	Connects to our motor and to the r	8632T139	$ 10.14	1	McMaster Carr	https://www.mc		$ 10.14
Light Duty Dry-Running Nylon Sleeve Bearing, 1/2" OD x 3/32" Thick Flange, for 1/4" Shaft Diameter, 1/2" Long	Nylon Dry Bushings	To decrease friction in our system	6389K407	$ 1.21	2	McMaster Carr	https://www.mc		$ 2.42
Clamping Two-Piece Shaft Collar for 1/4" Diameter Shaft, Black-Oxide 12L14 Carbon Steel	Shaft Collars for 1/4" shaft	To prevent axial movement	6436K12	$ 5.11	2	McMaster Carr	https://www.mc		$ 10.22
uxcell 6mm to 6.35mm Shaft Coupling 25mm Length 20mm Diameter Stepper Motor Coupler Al	Shaft Coupler 6mm to 1/4"	Shaft coupler to connect our motor	A16121900ux0381	$ 8.99	1	Amazon	https://www.amazon.com/uxcell-Coupling-Diameter-A		$ 8.99

Appendix

SPECIFICATION

- Gear ratio: 43.8:1
- No-load speed: 251 + 10% RPM
- No-load current: 350 mA
- Start Voltage: 1.0 V
- Stall Torque: 18 Kg.cm
- Stall Current: 7 A
- Insulation resistance: 20 M Ω
- EncoderOperating Voltage: 5 V
- Encoder type: Hall
- Encoder Resolution: 16CPR(motor shaft)/700CPR(gearbox shaft)
- Weight: 205g

A1 dfrobotics Metal DC Geared Motor w/Encoder - 12V 251RPM 18Kg.cm Motor Specifications