ME 102B Final Deliverable

Team 10
Ollie the Trash Robot
Upasana Dilip, Linda Krellner, Audrey Young, Tyler Vo
December 2024

1. Opportunity & Strategy

We sought to solve a problem in our community regarding public health and safety. In many urban areas and places with high pedestrian traffic, trash laying all over the environment is often an issue. Our opportunity is an effort to tackle this issue and keep our communities cleaner without endangering human workers. To combat this, we made Ollie, a trash pickup device that rolls around, picks up trash, and deposits it in a cart. Our device's high level strategy was using three DC motors, two for the drivetrain and one for the arm assembly, two servo motors, and an ultrasonic sensor to sense trash. Two DC motors actuate the wheels for it to drive around, and once it senses trash via an ultrasonic sensor, it scoops the trash via pin-machine like doors into a 3-D printed dustpan. Then another DC motor actuates the arm mechanism, and lifts whole dustpan with trash into the air until the trash slides into the cart. We didn't have exact specifications in terms of speed but we did want the arm to be at a high enough angle for trash to slide in, which turned out to be four 50 degree turns on the motor.

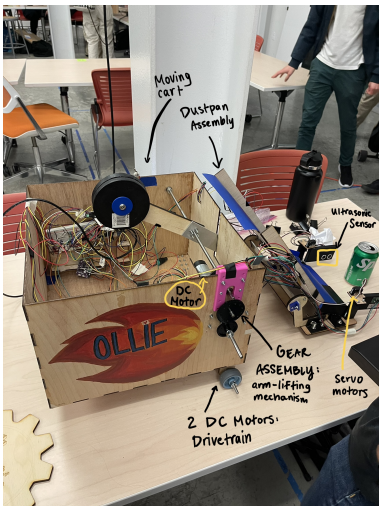


Figure 1: Final Robot at Showcase

2. Function Critical Decisions & Calculations

There are many material and design decisions we had to make for the robot. The main function critical decisions we had to make were for the actuation of the doors, the arm lifting actuation, and the driving. The equations used were the following:

$$\tau = r \times F \tag{1}$$

$$R = T_2/T_1 = \omega_1/\omega_2 = r_2/r_1 \tag{2}$$

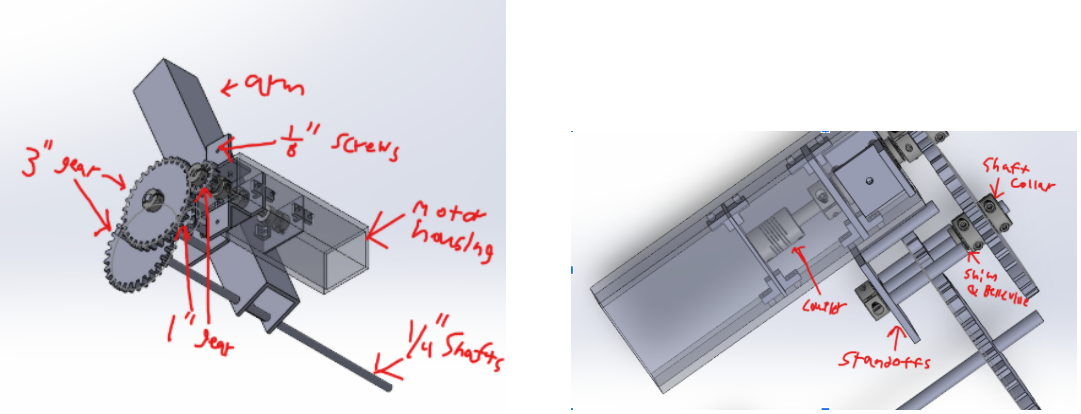
$$\sigma = My/I \tag{3}$$

$$\Sigma F = ma \tag{4}$$

One function critical calculation was the motor sizing for the door actuation. The motor was sized for possible friction between brushes on the door and ground, as well as friction between the ground and the aluminum cans. Our final design did not include brushes on the doors, but the required stall torque calculated using (1) was 14.938 oz-in. The servo motors actuating the doors had a stall torque of 18.09 oz-in, satisfying the torque requirements.

The function critical calculations for the lifting arm involved motor sizing (1), shaft sizing (3), bearing sizing (4), and gear ratio (2) calculations. A motor actuating a set of gears provided a slower, higher torque output to a shaft that would raise and lower the arm, shown in (2a) and (2b). The amount of torque needed to hold up the weight of the dustpan assembly (1) when the arm is parallel to the ground was used to size the motor. Since this is when the moment is at its max, the maximum required stall torque is 1280 oz-in. Using a 12V motor with a gearbox and a 5.33:1 gear ratio (2) gave us a total stall torque of 3330.89 oz-in. The arm motor connects to the shaft using a flexible shaft coupler, included in the motor housing shown in (2b). It is supported by two bearings, and is assembled with shims and Belleville washers. We calculated that the bearing must be able to support a maximum force of 4 pounds using (4).

The shaft connected to the motor was sized using bending analysis and a FOS of 2, which gave a minimum radius of 0.05". For ease of manufacturability, we went with a 1/4" diameter for all arm shafts. Given the diameter of the shaft, the ball bearings we found had a maximum static load of 30lbs and maximum dynamic load of 80lbs. The drivetrain involved similar calculations of shaft sizing, motor sizing, and bearing sizing, shown in (3).



(a) Labeled Image of the Gear System for the Arm Lifting Mechanism.

(b) Top View of Gear System for the Arm Lifting Mechanism.

Figure 2: Gear system for the arm lifting subsection

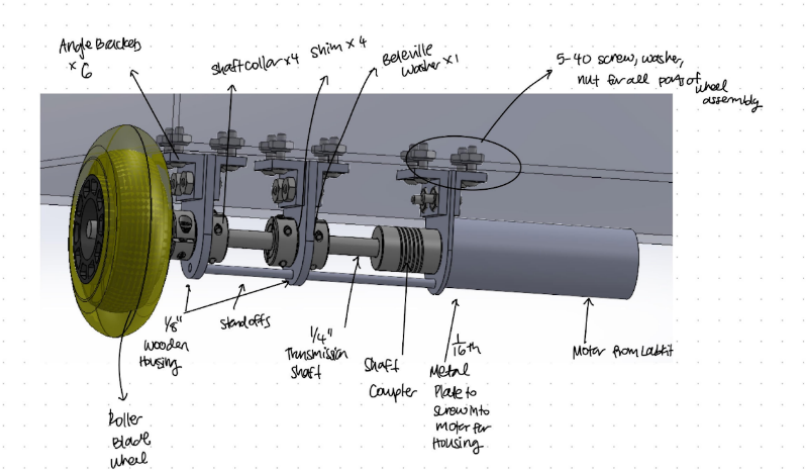
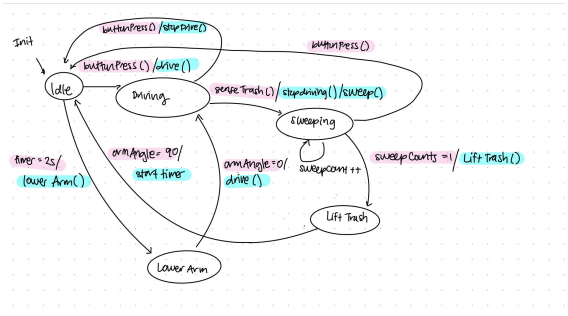
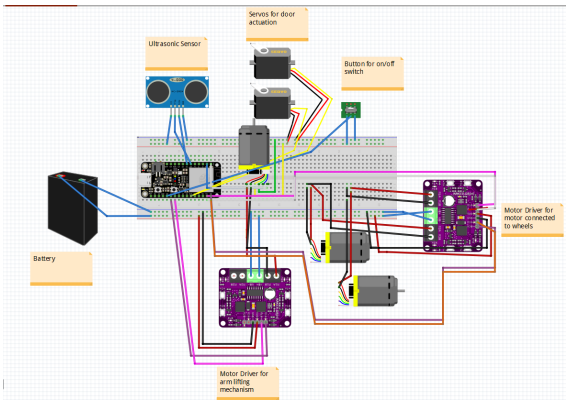


Figure 3: Driving Mechanism Motor Housing.

3. Circuit Diagram and State Machine



(a) Final state machine used at functionality check and showcase.



(b) Circuit Diagram used at functionality check and showcase

4. Reflection

Our group found that frequent communication with the Jacobs staff was invaluable. They provided key insights into machining techniques and material selection, which significantly improved our assembly process. Their expertise saved us time and reduced errors in our build. One thing we would change is creating a more detailed mechanical assembly plan before starting. This would have streamlined the process and avoided the issues caused by misaligned components. We could also have considered reinforcing the cart itself as the thin walls introduced a significant amount of play in the system.

5. Appendix

Please find our BOM here:

Part Name	Link	Vendor	Unit Cost	Quantity	Total Price	Num Per Container	Num Used	Notes
Chassis and Arm Assembly								
Ultrasonic Sensor	https://microkit.berkeley.edu/dc-gearmotor/							
Long range ultrasonic sensor								
Button								
Large Motor	12V Metal DC Geared Motor with Encoder (43.8:1, 251RPM, 18Kg. cm)ming-belt-pulleys/material~plastic-1/	Microkit	$0.00	3	$0.00	1	3	Comes in berkeley microkit, 18kg*cm stall torque, motor for arm movement and wheels, will be oversized for both arms and wheels
Small Motor	https://microkit.berkeley.edu/dc-gea	Microkit	$0.00	1	$0.00	1	1	Comes in old berkeley microkit, 1.3 kg*cm stall torque, motor for sweeper movement
1/8" screw	Product code: 92125A214	McMaster		23				Needed for motor housing and stand-off for complex gear
Ball Bearing	https://www.servocity.com/1-4-id-x-	ServoCity	$2.00	6	$11.97	2	6	McMaster, needed for supports in wheel subassembly and arm gear train subassembly
Shim for Ball Bearing	https://www.mcmaster.com/97022A	McMaster	$0.86	20	$17.26	10	12	McMaster, 1/4" shim needed for spacing between ball bearing inner race and collar/washer
Shaft Collar for 1/4" shaft	https://www.servocity.com/2920-ser	ServoCity	$2.50	6	$14.97	2	6	McMaster, collars to hold bearings together
Beleville Washer for 1/4" Shaft	https://www.mcmaster.com/94065K	McMaster	$5.00	10	$5.00	10	3	MCMaster beleville washers to add preload
Rollerblade Wheels	https://www.amazon.com/ProTec-W	Amazon	$9.00	1	$9.00	2	2	Amazon roller blade wheels, 6mm hole in bearing, will need to turn down shaft by 0.35mm
Swivel Wheel	https://www.amazon.com/Caster-W	Amazon	$12.95	4	$12.95	4	1	Amazon swivel wheel to attach to back of chassis, will need to extend length somehow
1/4"x6" D shaft	https://www.mcmaster.com/8632T1	McMaster	$7.52	3	$22.56	1	3	mcmaster d shaft for wheels and arm motor housing
32P, 16T, 0.250" (1/4) Bore, Plain Bore Gear (Delrin)	https://www.servocity.com/32p-16t-(	ServoCity	$3.59	1	$3.59	1	1	
32P, 48T, 0.250" (1/4) Bore, Plain Bore Gear (Delrin)	https://www.servocity.com/32p-48t-(	ServoCity	$5.19	1	$5.19	1		
32P, 64T, 0.250" (1/4) Bore, Plain Bore Gear (Delrin)	https://www.servocity.com/32p-64t-(	ServoCity	$5.99	2	$11.98	1	1	
Shaft Coupler	https://www.servocity.com/6mm-to-(	ServoCity	$5.99	3	$17.97	1	3	servocity shaft couplers, perfectly fit <3
Angle Bracket (Large)	https://www.mcmaster.com/1556A2	McMaster	$1.06	2	$2.12	1	2	McMaster angle brackets for holding the motor box up
Small Angle Bracket	https://www.amazon.com/Extra-Hea	Amazon	$6.95	20	$6.95	20	12	Amazon small angle brackets to hold up housing for wheel assembly and housing for arm motor
~~1/16" Almunim~~	~~https://www.mcmaster.com/8975K2~~	~~McMaster~~	~~$0.00~~	~~1~~	~~$0.00~~	~~1~~	~~1~~	~~To cut 3 housing walls to attach the motor~~
1/8" Plywood	https://store.jacobshall.org/products	Jacobs Hall Mate	$10.27	1	$10.27	1	1	1/8" thickness plywood to cut the wooded chassis box and elements from motor housings(s)
5-40 Screw	https://www.mcmaster.com/92949A	McMaster	$5.39	1	$5.39	100	72	5-40 screw for all housing and attachments in chassis
5-40 Washer	https://www.mcmaster.com/92141A	McMaster	$1.43	1	$1.43	1	72	5-40 washer for all housing and attachments in chassis
5-40 Nut	https://www.mcmaster.com/90480A	McMaster	$4.19	1	$4.19	1	72	5-40 nut for all housing and attachments in chassis

Part Name	Link	Vendor	Unit Cost	Quantity	Total Price	Num Per Container	Num Used	Notes
Flanged Coupler	https://www.amazon.com/SDTC-Te	Amazon	$7.49	1	$7.49	1	1	device that will transmit torque between the main arm shaft to the flat plate that is the main arm
M3 Bolt	https://www.mcmaster.com/92095A	Ace Hardware	$8.00	1	$8.00	100	1	the link for for m3 bolts on mcmaster, price should be comparable or less
M3 Nuts	https://www.acehardware.com/depa	Ace Hardware	$7.99	1	$7.99	100	1	the link is for M4 nuts but we actually need M3 nuts, could not find the right link, price should be comparable
1/4" Diameter 18 inch length rotary shaft 12L14 Carbon Steel	https://www.mcmaster.com/1327K1	McMaster	$22.13	1	$22.13			
1/4" Diameter 3 inch length rotary shaft 12L14 Carbon Steel	https://www.mcmaster.com/1327K3		$5.13	1	$5.13			
For Dustpan Gear Assembly								
1/4"-20 Screw	screws | McMaster-Carr	McMaster	$10.43	10	$10.43	75	10	
1/4"-20 Nuts	Medium-Strength Steel Hex Nut, Gr	McMaster	$7.08	10	$7.08	100	10	
5-40" Screw 1/2"	18-8 Stainless Steel Pan Head Slot	McMaster	$0.00	8	$0.00	100	8	HAVE IN BOM ALREADY
5-40 Nuts	Low-Strength Steel Hex Nut, Zinc-P	McMaster	$0.00	4	$0.00	100	1	HAVE IN BOM ALREADY
1/8" Alum	Multipurpose 6061 Aluminum Sheet	McMaster	$29.30	1	$29.30	1	1	dustpan base, 6x12", arms 9"x"2.5", 10.25"x1", servo plate 1.7"x1.46"
1/8" alum 5052	trying to scrounge							2 2"x2" angle bracket, 2 1"x4" servo holders
1/8" Door Axles	Rotary Shaft, 12L14 Carbon Steel,	McMaster	$3.17	2	$6.34	1	2	for doors, lowkey ace has em tho cheap
New Dustpan Gear Assembly								
motor	https://www.pololu.com/product/281	Polou	$5.25	1	$5.25	1	1	new servo motor to open doors
nut for motor	https://www.mcmaster.com/91828A	McMaster	$8.15	4	$8.15	100	4	
screw for motor	PN: 92000A019 https://www.mcmas	McMaster	$6.14	4	$6.14	100	4	
Miscellaneous								
~~Button (that stays~~								
Parts Total					$286.22			
					$130.19			
					$32.55			

5.1 Code

```
#include <Arduino.h>
#include <ESP32Encoder.h>
#include <ESP32Servo.h>

//Define constants ----------------------------------------------
#define BIN 34  // pin used for switch triggering
#define PWM1 17 //pin used for motor driving
#define PWM2 21 //pin 2 used for motor driving
#define armControl_1 25 //pin used for arm control motor PWM
#define armControl_2 26 //2nd pin used for arm control motor PWM
const int trigPin = 33; //ultrasonic pin
const int echoPin = 27; //ultrasonic pin
//const int SERVO_PIN = 5; //servo motor pin
const int SERVO_PIN_left = 4; //servo motor pin
const int SERVO_PIN_right = 5; //second servo motor pin
//Setup variables -----------------------------------------------
//double second = 0;  // Method 2
int state = 0;     //State initialized to 0
int sweepCount = 0; //used to keep track of number of time the doors
    sweep trash into the dustpan
//Setup interrupt variables -------------------------------------
volatile bool interruptCounter = false;  // check timer interrupt for
    encoder
volatile bool ultraInterrupt = false; //timer interrupt for ultrasonic
    sensor
volatile bool buttonIsPressed = false;  // event checker for
volatile bool armTimeInterrupt = false; //timer interrupt for raising
    and lowering the arm
volatile bool servoInterrupt = false; //timer interrupt for servo
volatile bool lifting = false;
volatile int count = 0;
int totalInterrupts;                    // counts the number of
    triggering of the alarm
//timer declarations
ESP32Encoder encoder;
hw_timer_t* timer = NULL;
hw_timer_t* timerUltra = NULL;
hw_timer_t* timerArm = NULL;
hw_timer_t* timerServo = NULL;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMuxUltra1 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMuxArm1 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMuxServo1 = portMUX_INITIALIZER_UNLOCKED;


// To define the states of servos
```

```
enum DoorState{
  DOOR_OPEN,
  DOOR_CLOSED,
};
//vars for servo and ultrasonic sensor
bool inActivePeriod = false;
bool driving = false;
long duration = 0;
int distance = 0;
//ADDED from servo interrupt code
DoorState currentState = DOOR_OPEN;
Servo doorServo; //creates an instance of the Servo class and doorServo
    is a single servo motor that the code can control
Servo doorServo2; //servo for door2

//Define open/closed angles for servos, different because they are
   oriented in mirrored way
const int open_angleRight = 170;
const int open_angleLeft = 0;
const int closed_angle_right = 80;
// worked but went too inward at -10
const int closed_angle_left = 80;
const int SWEEP_LIMIT = 1; // Number of sweeps during active period
float currentTime = 0;
float oldTime = 0;
bool trash = false;
//Variables for angle control
int thetaMax = 1400; //encoder ticks per revolution on DC motor
int liftDes = 0;//6*thetaMax/4; //0 is defined as the angle when arm is
    lifted
int lowerDes = 1450; //lowering and raising behavior are different so
   the motor lowers "less"
int theta = 1545; //5*thetaMax/4;//1925; //initialized to 7*90 degrees,
    will travel this many encoder ticks when raising
int error = 0;
int lastError = 0;
int sumError = 0;
int Kp = 10;
int Kd = 0;//12;
int D = 0;
//int Ki = 0.5;

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

```
const int NOMLPWMLVOLTAGE = 100;
//Initialization the three timers ------------------------------------------
void IRAM_ATTR onTime() {
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount();
  encoder.clearCount();
  interruptCounter = true;  // the function to be called when timer
      interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}
void IRAM_ATTR onTimeUltra() {
  portENTER_CRITICAL_ISR(&timerMuxUltra1);
  ultraInterrupt = true;  // the function to be called when timer
      interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMuxUltra1);
}
void IRAM_ATTR onTimeArm() {
  portENTER_CRITICAL_ISR(&timerMuxArm1);
  armTimeInterrupt = true;  // the function to be called when timer
      interrupt is triggered
  //timerRestart(timerArm);
  //timerStop(timerArm);
  portEXIT_CRITICAL_ISR(&timerMuxArm1);
}
void IRAM_ATTR onTimeServo() {
  portENTER_CRITICAL_ISR(&timerMuxServo1);
  servoInterrupt = true;  // the function to be called when timer
      interrupt is triggered
  //timerRestart(timerArm);
  //timerStop(timerArm);
  portEXIT_CRITICAL_ISR(&timerMuxServo1);
}
//initializes and begins timers
void TimerInterruptInit() {
  // Method 1
  // 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.
  // 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, 1000000, true);         // sets how many
      tics will you count to trigger the interrupt
  // timerAlarmEnable(timer); // Enables timer

  // Method 2
  timer = timerBegin(1000000);            // Set timer frequency to 1Mhz
```

```
  timerAttachInterrupt(timer, &onTime);  // Attach onTimer function to
      our timer.
  // Set alarm to call onTime function every second (value in
      microseconds).
  // Repeat the alarm (third parameter) with unlimited count = 0 (
      fourth parameter).
  timerAlarm(timer, 100000, true, 0);

  timerUltra = timerBegin(1000000);
  timerAttachInterrupt(timerUltra, &onTimeUltra);
  timerAlarm(timerUltra, 100000, true, 0);

  timerArm = timerBegin(1000000);
  timerAttachInterrupt(timerArm, &onTimeArm);
  timerAlarm(timerArm, 2000000, true, 0);

  timerServo = timerBegin(1000000);
  timerAttachInterrupt(timerServo, &onTimeServo);
  timerAlarm(timerServo, 10000000, true, 0);
}
//interrupt service for switch/button
void IRAM_ATTR isr(){
  if(buttonIsPressed == false)
  {
    buttonIsPressed = true;
  }
  else{
    buttonIsPressed = false;
  }

}
void setup() {
  // put your setup code here, to run once:
  Serial.println(interruptCounter);
  Serial.begin(115200);
  TimerInterruptInit();
  pinMode(BIN, INPUT);                  // configures the specified pin
      to behave either as an input or an output
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT);
  attachInterrupt(BIN, isr, RISING);
  //attachInterrupt(BIN, isr, CHANGE);
  Serial.println(interruptCounter);
  ESP32Encoder::useInternalWeakPullResistors = puType::up;  // Enable
      the weak pull up resistors
  encoder.attachHalfQuad(22, 23);                           // Attache
      pins for use as encoder pins
  encoder.setCount(0);
```

```
  ledcAttach(PWM_1, freq, resolution);
  ledcAttach(PWM_2, freq, resolution);
  ledcAttach(armControl_1, freq, resolution);
  ledcAttach(armControl_2, freq, resolution);
  timerStop(timerArm);
  timerStart(timerServo);
  doorServo.attach(SERVO_PIN_left); //attaches the servo output to a
     pin
  doorServo.write(open_angleRight); //Start with doors open

  doorServo2.attach(SERVO_PIN_right); //attaches the second servo
     output to a pin
  doorServo2.write(open_angleLeft); //start with second door open
  Serial.println("heello??");
}

void loop() {
  // put your main code here, to run repeatedly:
  switch(state)
  {
    case 0:
      if(sweepCount ==SWEEP_LIMIT){ //state machine enters idle while
         holding the garbage slide up for 2 seconds
        //Serial.println(timerReadSeconds(timerArm));
        Serial.print(theta);
        if(armTimeInterrupt)
        {
          portENTER_CRITICAL(&timerMuxArm1);
          armTimeInterrupt= false;
          portEXIT_CRITICAL(&timerMuxArm1);
          timerRestart(timerArm);
          timerStop(timerArm);
          Serial.println("stop hold");
          state = 3;
        }
      }
      if(buttonIsPressed && sweepCount ==0){ //state machine is
         initialized into idle state
        buttonIsPressed = false;
        Serial.println("button pressed"); //prints button pressed
        //sweep(DOOR_OPEN);
        startDriving(); //start driving service is called
        state = 1;
      }
    break;
    case 1:
      //Serial.println(buttonIsPressed);
      if(buttonIsPressed)
```

```
  {
    buttonIsPressed = false;
    if(driving)
    {
      Serial.println("button");
      stopDriving();
      state = 0;
    }
  }
  if(trashSensed()) //event checker to see if ultrasonic sensor has
      been triggered // was trashSensed() and servoInterrupt ==
     true
  {
    Serial.println("trash sensed");
    servoInterrupt = false;
    //timerRestart(timerServo);
    //timerStop(timerServo);
    stopDriving(); // service to stop drivin
    sweep(DOOR_OPEN); //service to sweep
    state = 2;
    trash = false;
  }
  if(trashSensed() == false) //continue driving
  {
    //Serial.println("continue driving");
    startDriving();
    if(trash == false){
      oldTime = timerReadMillis(timerServo);
    }

  }
break;
case 2:
{
  //Serial.println(buttonIsPressed);
  if(buttonIsPressed)
  {
    buttonIsPressed = false;
    //if(sweepCount <SWEEP_LIMIT)
    //{
    Serial.println("button");
    stopDriving();
    ledcWrite(armControl_1, 0);
    ledcWrite(armControl_2, 0);
    sweep(DOOR_OPEN);
    sweepCount = 0;
    state = 0;
    //}
```

```
    }
    if(sweepCount < SWEEP_LIMIT) //event checker to see when sweeping
        has finished
    {
      servoInterrupt = false;
      if (currentState == DOOR_OPEN) {
      sweep(DOOR_CLOSED);
      } else {
      sweep(DOOR_OPEN);
      }
      sweepCount++;
      Serial.print("Sweep Count: ");
      Serial.println(sweepCount);
      //delay(1000);
    }
      //delay(1000);

      //if (sweepCount == SWEEP_LIMIT) { //EVENT CHECKER to check if
          sweep count is the max sweep count
      //sweepCount = 0;
      //Serial.println("resetting sweep period");
      //delay(1000);
      //state = 1;
    if(sweepCount ==SWEEP_LIMIT && theta <= liftDes+25 && theta>=
       liftDes-25){
      Serial.println("TIMER STARTED");
      ledcWrite(armControl_1, LOW);
      ledcWrite(armControl_2, LOW);
      timerStart(timerArm);
      sweep(DOOR_OPEN);
      armTimeInterrupt = false;
      state = 0;
    }
    else if(sweepCount == SWEEP_LIMIT){
      LiftTrash(); //service to lift the grabage slide to put trash
          in cart/receptacle
      //timerStop(timerArm); gioglih;loj;l
      //timerRestart(timerArm);
      //timerStart(timerArm);
      //armTimeInterrupt = false;
      //state = 0;
    }

}
break;
case 3:
{
  sweepCount = 0;
```

```
      if(buttonIsPressed)
      {
        buttonIsPressed = false;
        ledcWrite(armControl_1, 0);
        ledcWrite(armControl_2, 0);
        Serial.println("button");
        state = 0;

      }
      if(theta <= lowerDes +25 && theta >= lowerDes-25)
      {
        state = 1;
        ledcWrite(armControl_1, 0);
        ledcWrite(armControl_2, 0);
        Serial.print("start driving");
        //testing somethibng sweep(DOOR_OPEN);
        theta = 7*(thetaMax/4); //reset angle to be 0, is this gonna
            work?
        //lowerArm(); //service to lower arm
      }
      else{
        lowerArm(); //service to lower arm
        //sweep(DOOR_OPEN);
      }
      //state = 1;
    }
    break;
    // case 4:
    // {
    //   if(angle = 0)
    //   {
    //     startDrive();
    //     state = 0;
    //   }
    // }
  }

}

//Event Checkers

bool CheckForButtonPress(){
  //if(buttonIsPressed == true && DEBOUNCINGflag == false){

  }

bool trashSensed(){
  digitalWrite(trigPin, HIGH);
```

```
  if(ultraInterrupt) {
    portENTER_CRITICAL(&timerMuxUltra1);
    ultraInterrupt = false;  // reset interruptCounter flag to false
    portEXIT_CRITICAL(&timerMuxUltra1);

    // Sets the trigPin on HIGH state for 10 micro seconds
    //digitalWrite(trigPin, HIGH);
    //delay()
    digitalWrite(trigPin, LOW);
    // Reads the echoPin, returns the sound wave travel time in
       microseconds
    duration = pulseIn(echoPin, HIGH);
    // Calculating the distance
    distance = duration * 0.034 / 2;
    // Prints the distance on the Serial Monitor
    Serial.print("Distance:");
    Serial.println(distance);
    if(distance <= 10)
    {
      Serial.println("should start sweeping");
      trash = true;
      //timerStop(timerServo);
      //timerStart(timerServo);
      currentTime = timerReadMillis(timerServo);
      Serial.println("current time");
      Serial.print(currentTime);
      Serial.print("old time");
      Serial.print(oldTime);
      if(currentTime >= oldTime + 550){
        return true;
      }
    }
  }
  return false;
}
// int armAngle()
// {
//   return motorAngle;
// }


//Event Services
void startDriving(){
  if(interruptCounter){
    portENTER_CRITICAL(&timerMux1);
    interruptCounter = false;
    int omegaSpeed = count;
    Serial.print("driving");
```

```
    Serial.println(omegaSpeed);
    portEXIT_CRITICAL(&timerMux1);
    ledcWrite(PWM_1, LOW);
    ledcWrite(PWM_2, 100);
    driving = true;
  }

  //D = controllerCommand();
  //drive(D);
}

void stopDriving(){
  //if(interruptCounter){
  portENTER_CRITICAL(&timerMux1);
  interruptCounter = false;
  int omegaSpeed = count;
  Serial.print("stop driving ");
  Serial.println(omegaSpeed);
  portEXIT_CRITICAL(&timerMux1);
  ledcWrite(PWM_1, LOW);
  ledcWrite(PWM_2, LOW);
  driving = false;
  //}
}

void sweep(DoorState newState){
  //ADDED from servo interrupt code
  currentState = newState;
  if (newState == DOOR_OPEN){
    Serial.println("Doors are now open");
    doorServo.write(open_angleRight);
    doorServo2.write(open_angleLeft);
  } else if (newState == DOOR_CLOSED) {
    Serial.println("Doors are now closed");
    doorServo.write(closed_angle_right);
    doorServo2.write(closed_angle_left);
  }
}

void LiftTrash()
{
  if(interruptCounter){
    portENTER_CRITICAL(&timerMux1);
    interruptCounter = false;
    int omegaSpeed = count;
    theta+=count;
    error = 0-theta;//int(6*thetaMax/4)-theta;
    sumError = sumError+error/10;
```

```
    int lastDiff = error-lastError;
    D = Kp*error+Kd*lastDiff;//+Ki*sumError;
    Serial.print(theta);
    Serial.print("-");
    Serial.print(error);
    Serial.print("-");
    Serial.println(D);
    lastError = error;
    portEXIT_CRITICAL(&timerMux1);
    if (D > NOM_PWM_VOLTAGE) {
      D = NOM_PWM_VOLTAGE;
      sumError-=error;
    }
    else if (D < -NOM_PWM_VOLTAGE) {
      D = -NOM_PWM_VOLTAGE;
      sumError-=error;
    }
    if (D > 0) {
      ledcWrite(armControl_1, LOW);
      ledcWrite(armControl_2, D);
    } else if (D < 0) {
      ledcWrite(armControl_1, -D);
      ledcWrite(armControl_2, LOW);
    } else {
      ledcWrite(armControl_2, LOW);
      ledcWrite(armControl_1, LOW);
    }
    lifting = true;
  }
}
void lowerArm()
{
  if(interruptCounter){
    portENTER_CRITICAL(&timerMux1);
    interruptCounter = false;
    int omegaSpeed = count;
    theta+=count;
    error = lowerDes-theta;//int(6*thetaMax/4);
    sumError = sumError+error/10;
    int lastDiff = error-lastError;
    int D = Kp*error+Kd*lastDiff;//+Ki*sumError;
    lastError = error;
    Serial.print(theta);
    Serial.print("-");
    Serial.print(error);
    Serial.print("-");
    Serial.println(D);
    portEXIT_CRITICAL(&timerMux1);
```

```
    if (D > NOM_PWM_VOLTAGE) {
      D = NOM_PWM_VOLTAGE;
      sumError-=error;
    }
    else if (D < -NOM_PWM_VOLTAGE) {
      D = -NOM_PWM_VOLTAGE;
      sumError-=error;
    }
    if (D > 0) {
      ledcWrite(armControl_1, LOW);
      ledcWrite(armControl_2, 70);
    } else if (D < 0) {
      ledcWrite(armControl_1, -70);
      ledcWrite(armControl_2, LOW);
    } else {
      ledcWrite(armControl_2, LOW);
      ledcWrite(armControl_1, LOW);
    }
    lifting = true;
  }
}

// //Controller Functions
// int controllerCommand()
// {
//   return 100;
// }

int angleControl(int desiredAngle)
{
  //lab 6 angle control code
  theta+=count;
  error = desiredAngle-theta;
  sumError = sumError+error/10;
  int lastDiff = error-lastError;
  int D = Kp*error*Kd*lastDiff;//+Ki*sumError;
  lastError = error;
  return D;
}
// void drive(int D)
// {
//   if (D > 0) {
//     ledcWrite(BIN_1, LOW);
//     ledcWrite(BIN_2, D);
//   } else if (D < 0) {
//     ledcWrite(BIN_2, LOW);
//     ledcWrite(BIN_1, -D);
//   } else {
```

```
//      ledcWrite(BIN_2, LOW);
//      ledcWrite(BIN_1, LOW);
//    }
// }

//Timer

// void TimerTEMP()
// {
//   if (interruptCounter) {  // interruptCounter will be 'true' when
    timer interrupt is triggered
//      portENTER_CRITICAL(&timerMux);
//      interruptCounter = false;  // reset interruptCounter flag to
    false
//      portEXIT_CRITICAL(&timerMux);

//      totalInterrupts++;

//      Serial.print("totalInterrupts");
//      Serial.println(totalInterrupts);
//      Serial.print("Timer interrupt is triggered in every ");

//      // Method 1
//      // Serial.print(timerAlarmReadSeconds(timer));

//      // Method 2
//      Serial.print(second);

//      Serial.println(" second(s)");

//      // Serial.print0000ln(timerRead(timer));
//      if (totalInterrupts % 2 == 0) {
//        // Lights up the LED if the counter is even
//        digitalWrite(LED_PIN, HIGH);
//      } else {
//        // Then swith off
//        digitalWrite(LED_PIN, LOW);
//      }
//    }
//    second = timerReadSeconds(timer);  // Method 2
// }
```

5.2 CAD

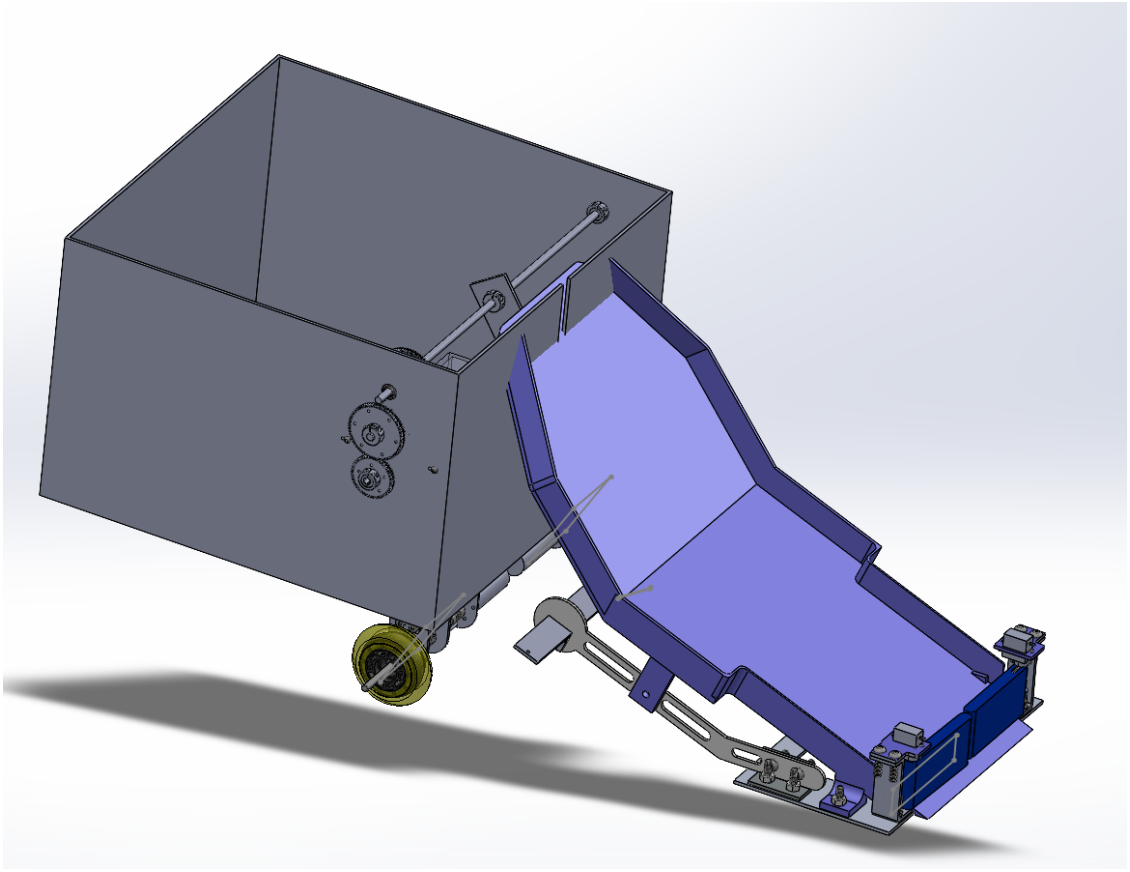
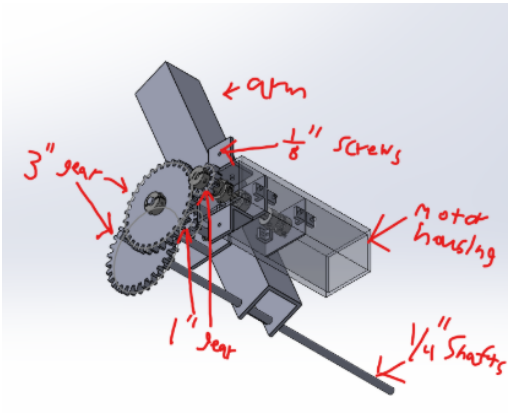
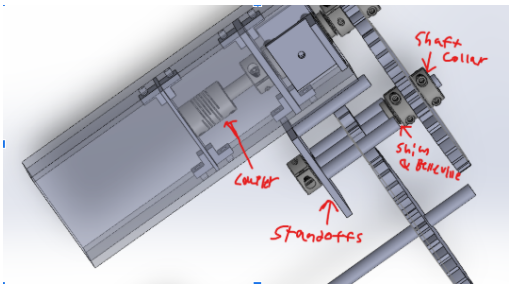


Figure 5: Full CAD, with slight mating mishaps



(a) Labeled Image of the Gear System for the Arm Lifting Mechanism.



(b) Top View of Gear System for the Arm Lifting Mechanism.

Figure 6: Isolated Gear System

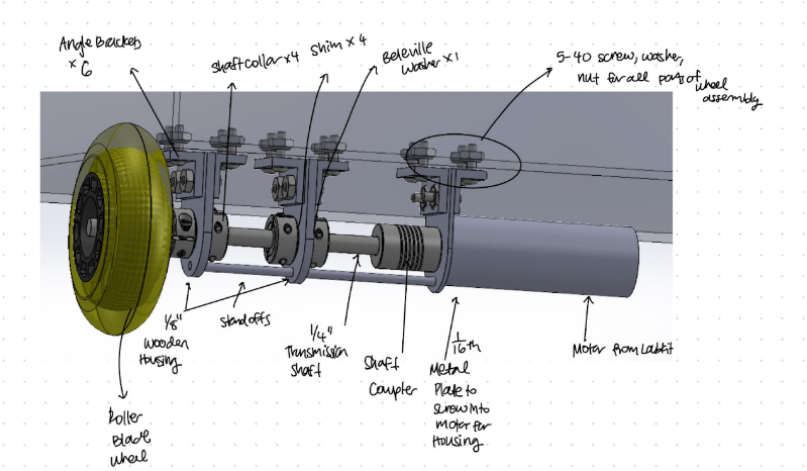


Figure 7: Close up of driving subsystem