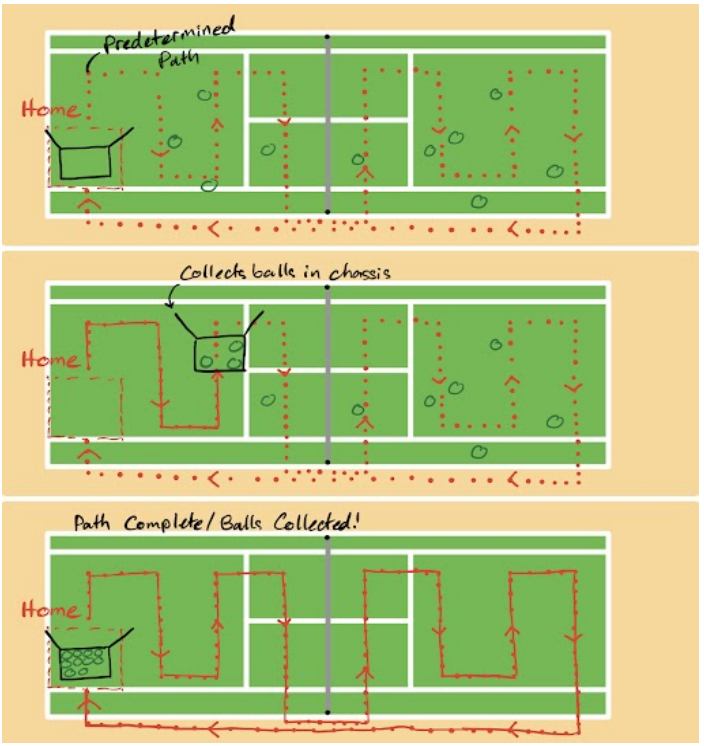
Roll-On-You-Bot

Mark Theis[1], Elizabeth Gammariello[1], Han Nguyen[1]

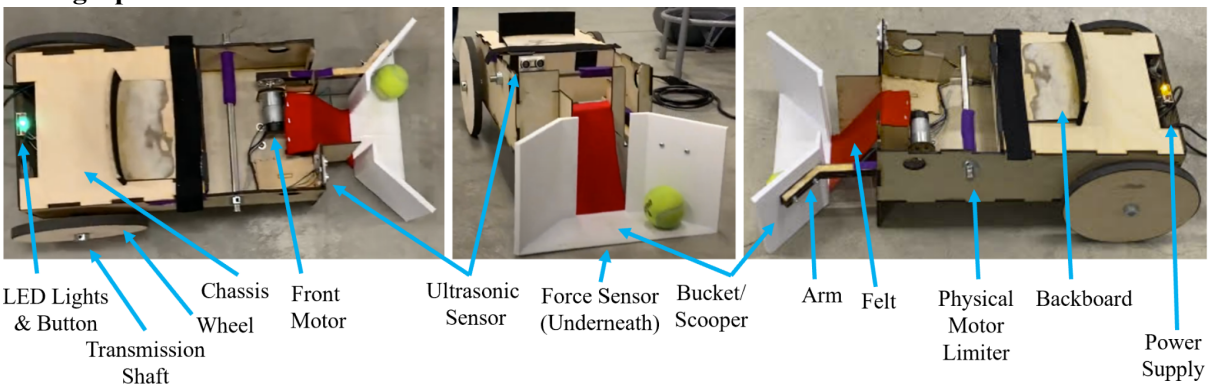
Opportunity:
Ask any tennis player, and they will likely share one of the few dislikes of the game is picking up tennis balls during practice or a match. These 66 mm diameter bright yellow objectives roll away easily and take time to bend down and grab. This led us to our project's opportunity: How can we make picking up tennis balls off of a tennis court easier for the player using mechatronics design? We designed Roll-On-You-Bot, a wheel driven robot that can maneuver on a tennis court, scoop up tennis balls, and store them so that they may be collected by the player.

High Level Strategy:
The Roll-On-You-Bot (or ROY-Bot) drives around tennis courts or concrete surfaces. The ROY-Bot begins in an idle state. Once the button is pressed, the robot enters the collecting state during which it moves forward at full speed for 1 second and then comes to a gradual stop. If the robot scoops up a ball, it will detect the added weight in the front bucket and enter the full state. At this point, the robot waits for the user to push a button so it can empty the ball into the chassis. If the robot did not scoop up a ball, it will wait for the player to push the button so it can move forward again. If, while the robot is moving, it comes too close to an obstacle, such as a wall, net, or player, it will stop moving and wait for the user to physically reset the robot and move it to a different location.

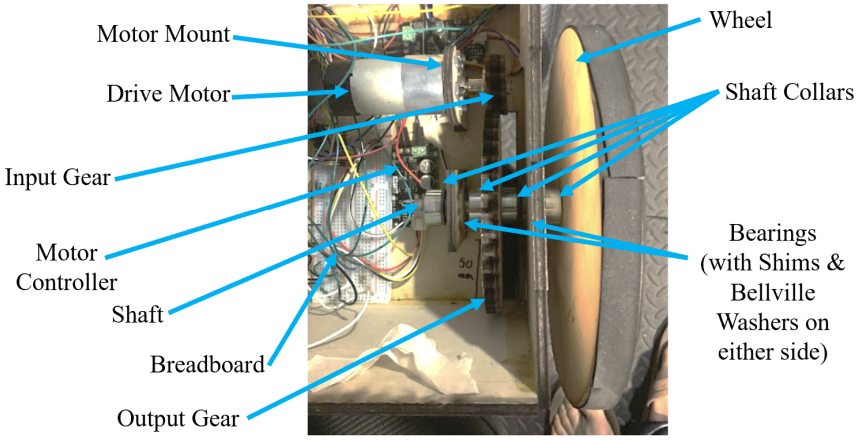


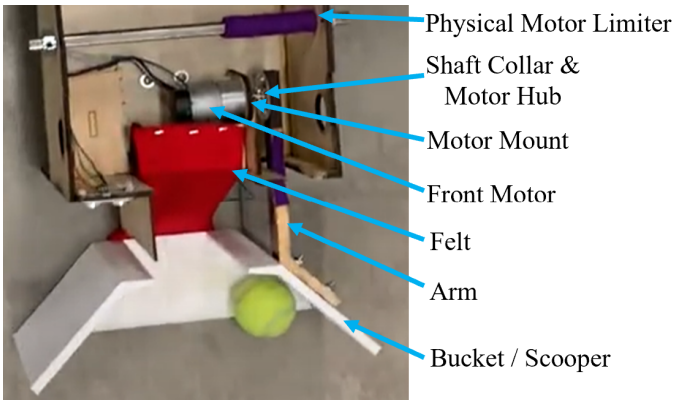
Initially, ROY-Bot was going to detect the position of nearby balls, drive to them, and collect them. Our team quickly realized, however, that this was unreasonable to implement in our prototype due to the cost of sensors, such as a lidar. Moreover, the software challenges, while an interesting feat, were beyond the scope of the course. This is a potent area of investigation for future iterations of ROY-Bot. Another change was to the ball emptying mechanism. Originally, we planned to use a rack and pinion to lift up the collector bucket and dump balls into the chassis. However, based on advice from mentors and team discussion, we modified this to an arm that a motor rotates to lift the bucket.

Photographs:



[1] ME102B at UC Berkeley, {markt, eagammariello, hanhn} @berkeley.edu





At the front of the robot, the white 3D printed bucket scoops up a tennis ball. With the ball inside, a force sensor under the bucket notifies the microcontroller. There is also an HC-SR04 ultrasonic distance sensor that measures frontal clearance. The white bucket lifts up due to actuation from the motor and arm, depositing the ball into the chassis. Sometimes, the ball goes above the top of the robot, so a small backboard helps guide it into the storage compartment. At the rear of the robot are three status LEDs (red, yellow and green) and a push button for control. Power cables protrude from the rear which supply the motors and microcontroller. Inside, a breadboard routes the sensors and motor controllers to the Arduino Uno microcontroller and the power supplies. The two drive motors attach to 38 mm gears for the 4:1 transmission to the wheels. The corresponding 152 mm output gears attach to the wheel shafts. The wheels themselves are secured on the exterior of the chassis. One enhancement that we expect could improve the performance of the drivetrain would be the addition of locating features to the motor mounts. This would align them optimally on both sides of the robot to their respective transmissions.

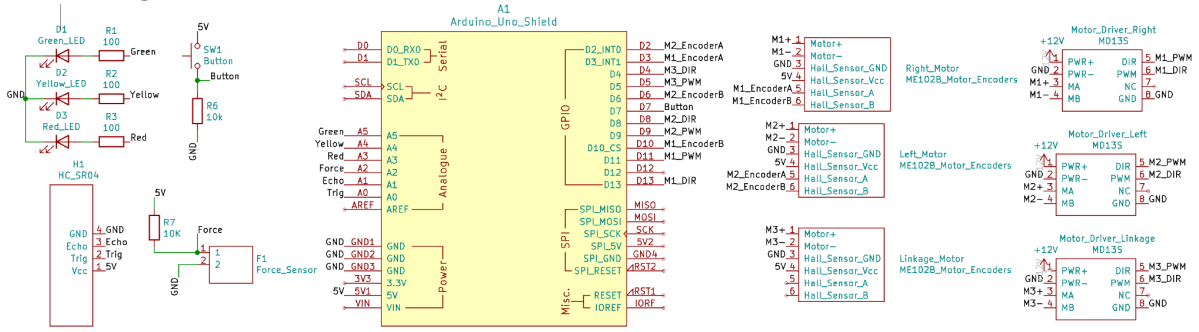
Function-Critical Decisions:

When designing the drive train, we separated the motors from the wheels by means of a gear box transmission. This provided two advantages: firstly, it increased the gear ratio by 4:1, giving us more torque to drive our chassis, and secondly, it rested the weight of the robot on larger shaft and bearing pairs that are rated for higher radial loads. However, there was still the force from the meshing gears to consider. We determined the maximum radial force on the motor shaft as follows:

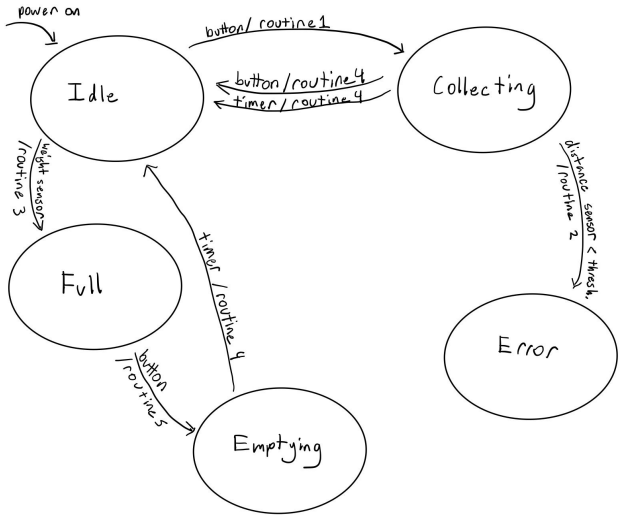
$$\tau_{max} = 0.63\,N \cdot m,\; F_{max} = \frac{\tau_{max}}{R} = 33.0\,N,\; F_{max,rad} = F_{max} \cdot sin(20°) = 11.29\,N$$

The input gear has a 38 mm diameter, and with the maximum torque available from the motors in our configuration, the highest radial force was 11.29 N, equivalent to 1.15 kg. We contacted the manufacturer and they specified they rate their motors for 2kg of radial load. With this advice, we found separating the motor from the input gear by a second shaft and flexible shaft coupler was unnecessary. Moreover, this led us to a similar approach with the front motor connected to the ball scooper bucket, so we connected the scooper arm directly to the motor in this setup.

Circuit Diagram:



State Machine Diagram:



Routine 1 (enter collecting):
- LED: Green
- Drivetrain motor: on
- Ball emptying linkage motor: off

Routine 2 (enter error):
- LED: Red (flashing)
- Drivetrain motor: off
- Ball emptying linkage motor: off

Routine 3 (enter full):
- LED: Red
- Drivetrain motor: off
- Ball emptying linkage motor: off

Routine 4 (enter idle):
- LED: Yellow
- Drivetrain motor: off
- Ball emptying linkage motor: off

Routine 5 (enter emptying):
- LED: Yellow (flashing)
- Drivetrain motor: off
- Ball emptying linkage motor: on

The above state machine diagram represents the final version of our state machine used during the showcase. All states have remained the same during the project. There were some changes to the transitions. These include: removing front distance checking during the emptying state, and checking for button presses during the drive state to change to the idle state if pressed.

Reflection:

Before manufacturing, our group made sure that we had an understandable and functionable state machine diagram. This helped the software and electrical aspects of the project come together well. Having a functioning and detailed CAD also made the manufacturing and ordering of components organized and smooth. However, because our group was so focused on the theoretical aspects of the project, we did not start manufacturing and integration until late in the semester. Our advice to future ME102B students would be to begin manufacturing and testing subsystems as soon as possible. Specific areas of improvement we would investigate with more time and resources would be position control of the front motor, which would require a larger microcontroller, and an improved ball detection mechanism for the front bucket.

Appendix

Bill of Materials

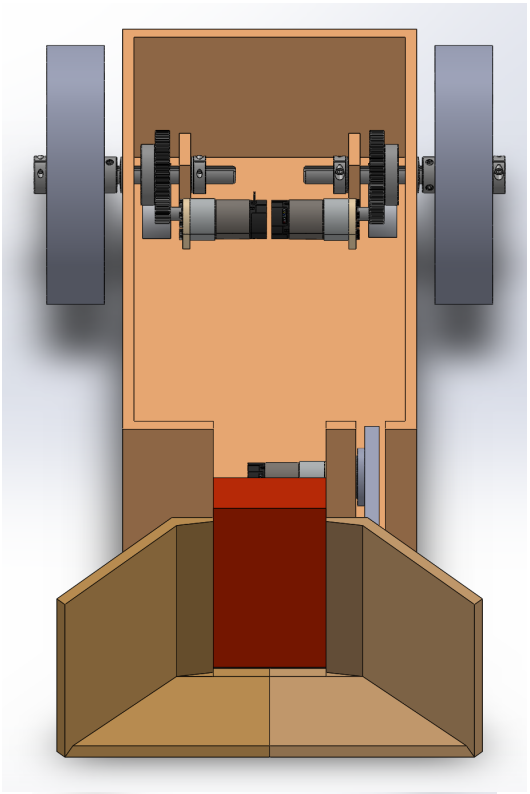
Item	Notes	Quantity	Cost Each	Total Cost
¼ Inch Plywood 24” x 48” Sheet ● Chassis ● Gears ● Wheels ● Arm ● Motor hubs	Purchased from the Jacobs Hall Material Store	4	$13.32	$53.28
Pololu 30:1 Metal Gearmotor 37Dx68L mm 12V with 64 CPR Encoder	Borrowed from Hesse Hall	3	$0	$0
Ultrasonic Sensor	Provided by ME102B MicroKit	1	$0	$0
Pressure Sensor	Purchased from Amazon	1 pack of 2	$11.09	$11.09
½ Inch Shaft Collars	Purchased from Amazon	3 packs of 4	$10.99	$32.97
½ Inch Shaft Shims	Purchased from McMaster-Carr	1 pack of 10	$4.75	$4.75
½ Inch Shaft Disk Spring	Purchased from McMaster-Carr	1 pack of 12	$8.93	$8.93
6mm Shaft Flange Coupler	Purchased from Amazon	1 pack of 4	$7.99	$7.99
½ Inch Metal Rod ● Transmission shafts ● Arm motor limiter	Machined into a D-shaped transmission shaft	4 ft.	$1.50	$6
M3 x 10 mm Screws	Provided by Jacobs Hall hardware supply	18	$0	$0
M4 x 25mm Screws	Provided by Jacobs Hall hardware supply	6	$0	$0
M3 Nuts	Provided by Jacobs Hall hardware supply	12	$0	$0
M4 Nuts	Provided by Jacobs Hall hardware supply	6	$0	$0

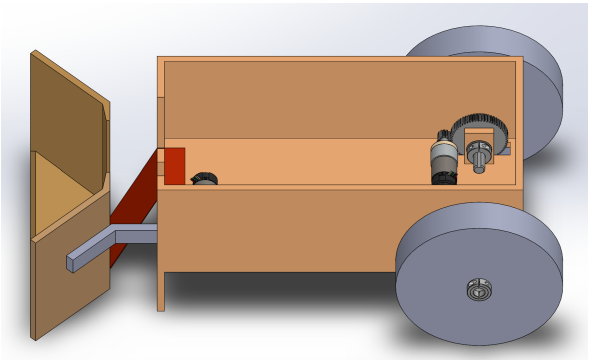
M3 Washers	Provided by Jacobs Hall hardware supply	12	$0	$0
M4 Washers	Provided by Jacobs Hall hardware supply	6	$0	$0
Arduino Uno Mictrocontroller	Purchased from Amazon	1	$19.99	$19.99
3D Printer PLA Plastic • Front Scooper	PLA used from Jacobs Hall Ultimaker printers	500 g	$0	$0
2” Swivel Caster Wheels	Purchased on Amazon	Pack of 4	$24.23	$24.23
Tennis Ball(s)	Borrowed from group members	2	$0	$0
Felt 10 x 10 inch piece	Provided by Jacobs Hall hardware supply	2	$0	$0
Total Cost:				$169.23

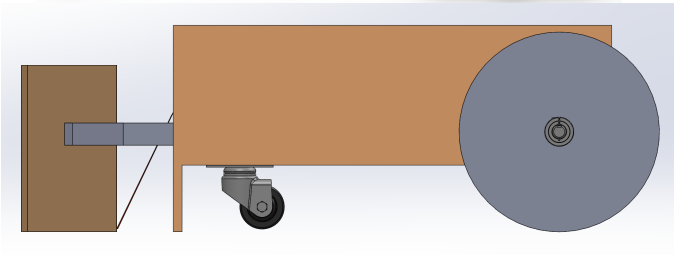
CAD Images

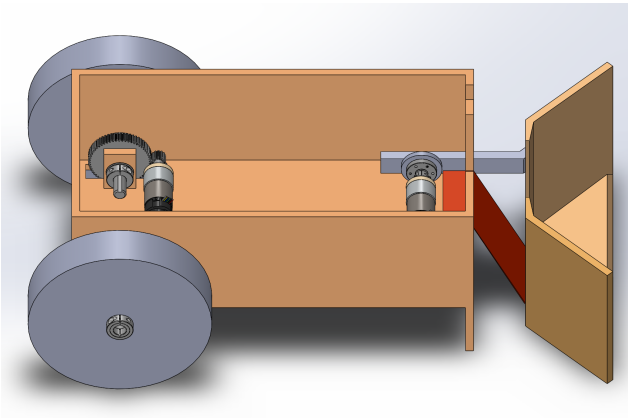
The chassis cover and motor / transmission protector were not included in the CAD for ease of viewing.

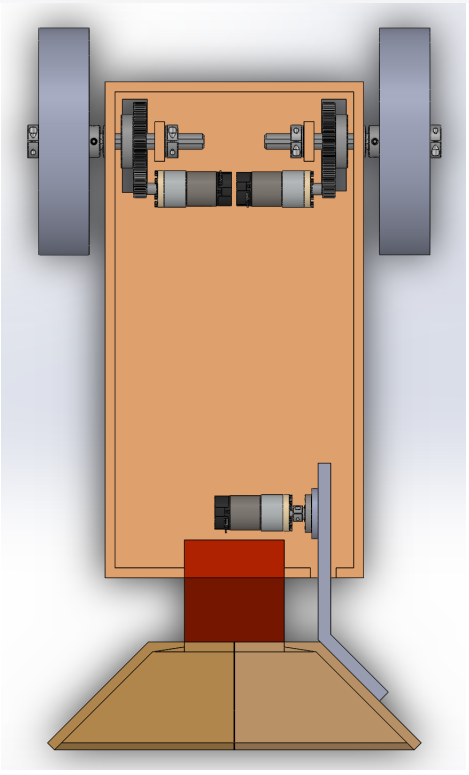
Shown below are the front, side, and top views of the robot.



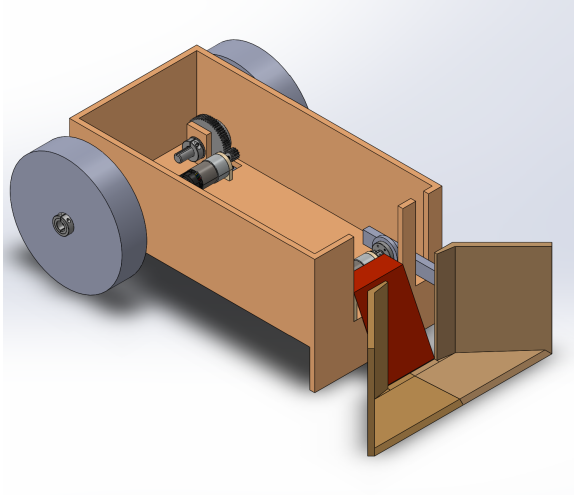


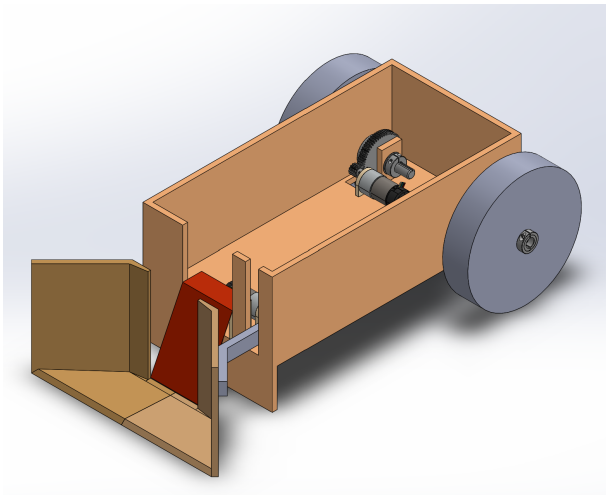




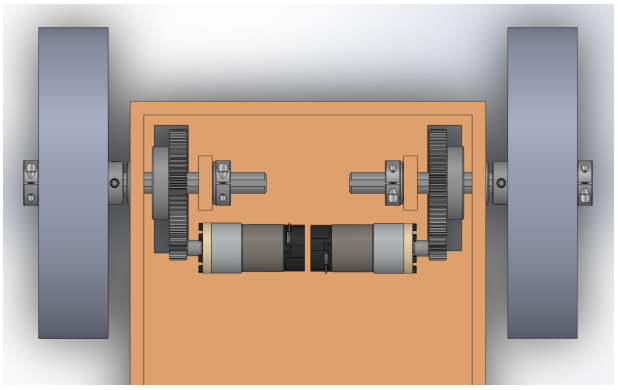


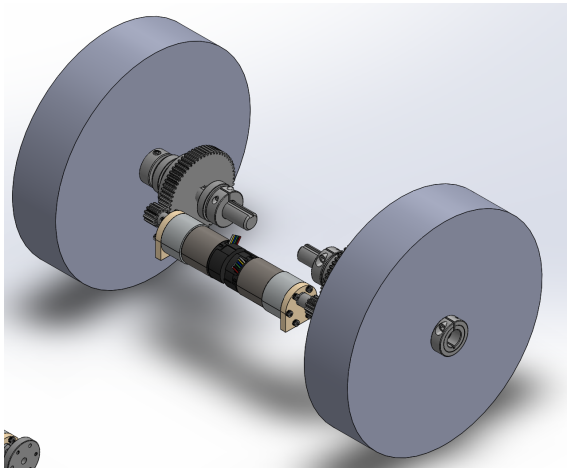
Shown below are the isometric views of the robot.



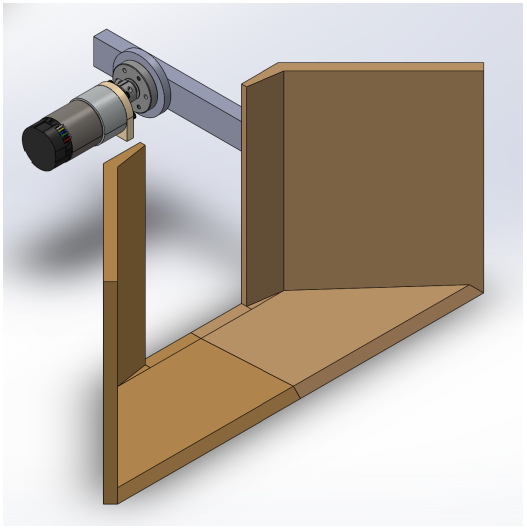


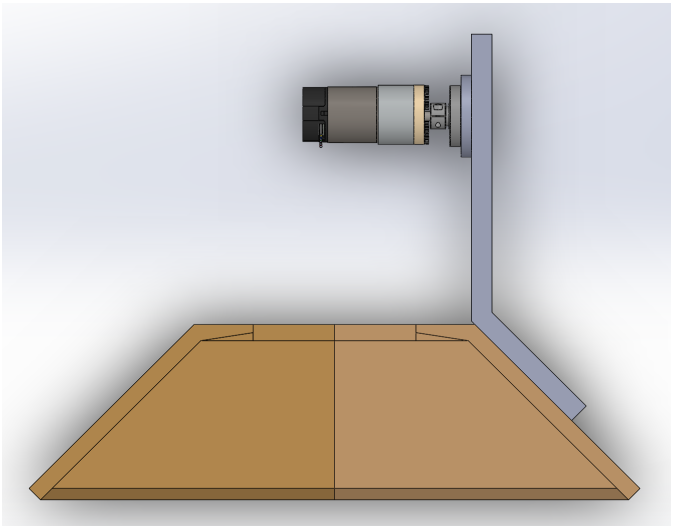
Shown below are views of the transmission mechanisms.

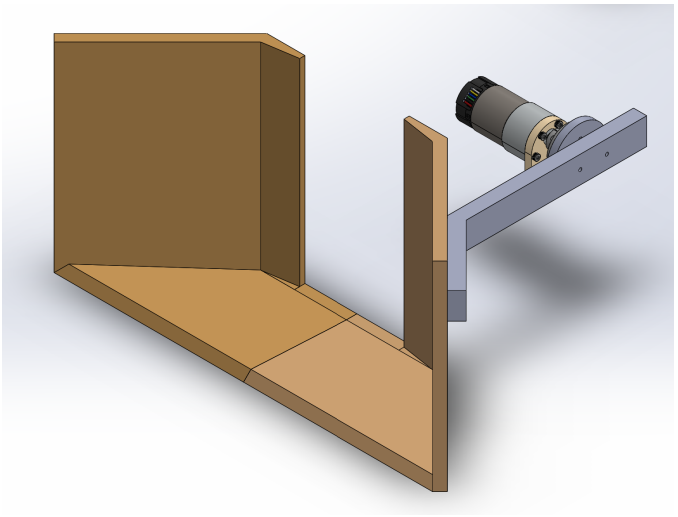




Shown below are views of the arm mechanism.







Microcontroller Firmware

```
// import libraries
#include <Encoder.h>
#include <HCSR04.h> // https://github.com/Martinsos/arduino-lib-hc-sr04
#include "TimerInterrupt.h"
// Specify GPIO Mapping
// led
#define led_green A5
#define led_yellow A4
#define led_red A3
// hc_sr04
#define hc_echo A1
#define hc_trig A0
// button
#define button 7
// force_sensor
#define fsensor A2
// motors
#define mdriver_1_dir 13
#define mdriver_1_pwm 11
#define mdriver_1_enc1 3
#define mdriver_1_enc2 10
#define mdriver_2_dir 8
#define mdriver_2_pwm 9
#define mdriver_2_enc1 2
#define mdriver_2_enc2 6
#define mdriver_3_dir 4
#define mdriver_3_pwm 5
// constants
#define STATE_IDLE 0
#define STATE_COLLECTING 1
#define STATE_FULL 2
#define STATE_EMPTYING 3
#define STATE_ERROR 4
#define MIN_DISTANCE 45       // cm
#define FORCE_SENSOR_THRESH 570 // 2.75V / 3.3V * 1023
#define ERROR_MAX 5
#define DRIVE 1
// PWM properties
#define MAX_PWM_VOLTAGE 255
#define NOM_PWM_VOLTAGE 150
#define OMEGA_DES_DRIVE 2200
// Timers
#define USE_TIMER_1 true
#define USE_TIMER_2 true
```

```
#define TIMER1_INTERVAL_MS 50L
#define TIMER2_INTERVAL_MS 100L
#define STATE_INTERVAL_MS 20
#define LOGIC_INTERVAL_US 1000
#define LED_INTERVAL_MS 50
#define DRIVE_TIME_MS 2600
#define SLOW_DOWN_INTERVAL_MS 100
#define SLOW_DOWN_START_MS 1000
#define LINKAGE_TIME_MS 5000
#define BUTTON_DEBOUNCE_TIMER 500
#define DRIVE_FEEDBACK_TIMER 100
#define LINKAGE_PWM_0 210
#define LINKAGE_TIME_1 400
#define LINKAGE_PWM_1 125
#define LINKAGE_TIME_2 640
#define LINKAGE_PWM_2 80 // -1
#define LINKAGE_TIME_21 700
#define LINKAGE_PWM_21 110 // -1
#define LINKAGE_TIME_3 725
#define LINKAGE_PWM_3 2
#define LINKAGE_TIME_4 4000 // going back
#define LINKAGE_PWM_4 120   // -1
#define LINKAGE_TIME_5 4300
#define LINKAGE_PWM_5 70
#define LINKAGE_TIME_6 4400
#define LINKAGE_PWM_6 70
#define LINKAGE_TIME_61 4500
#define LINKAGE_PWM_61 100
#define LINKAGE_TIME_62 4850
#define LINKAGE_PWM_62 30
#define FULL_TIME_MS 5000

// encoder objects
Encoder encDriveLeft(mdriver_1_enc1, mdriver_1_enc2);
Encoder encDriveRight(mdriver_2_enc1, mdriver_2_enc2);
// HC-SR04 distance sensor object
UltraSonicDistanceSensor distanceSensor(hc_trig, hc_echo);

// global variables
int global_state;
volatile bool buttonIsPressed;
volatile float force_sensor_reading;
volatile float force_sensor_reading_acc;
volatile float hc_distance_acc; // cm
volatile float hc_distance;   // cm
volatile bool drive_counter;
```

```
bool driving;
bool emptying;
volatile bool led_on;
long positionLeft;
long positionRight;
bool led_red_flashing;
bool led_yellow_flashing;
long drive_time;
long led_time;
long loop_time;
volatile long current_time_US;
volatile long current_time_MS;
bool button_timer_active;
long button_timer;
long sensor_time;
long drive_time2;
int error_sum_right;
int error_sum_left;
long full_timer;
long slow_down_time;
int omega_des_local;

// Initialization

void setup()
{
  // set global variable initial value
  global_state = STATE_IDLE; // default state is IDLE
  buttonIsPressed = false;
  force_sensor_reading = 0;
  force_sensor_reading_acc = 0;
  drive_counter = false;
  hc_distance = 0;
  hc_distance_acc = 0;
  driving = false;
  emptying = false;
  led_on = false;
  led_red_flashing = false;
  led_yellow_flashing = false;
  positionLeft = 0;
  positionRight = 0;
  drive_time = 0;
  led_time = 0;
  loop_time = 0;
  current_time_US = 0;
  current_time_MS = 0;
```

```
  button_timer_active = false;
  button_timer = 0;
  sensor_time = 0;
  drive_time2 = 0;
  error_sum_right = 0;
  error_sum_left = 0;
  full_timer = 0;
  slow_down_time = 0;
  omega_des_local = OMEGA_DES_DRIVE;

  // assign pins
  pinMode(button, INPUT);
  pinMode(led_green, OUTPUT);
  pinMode(led_yellow, OUTPUT);
  pinMode(led_red, OUTPUT);
  pinMode(fsensor, INPUT);
  pinMode(mdriver_1_dir, OUTPUT);
  pinMode(mdriver_1_pwm, OUTPUT);
  pinMode(mdriver_2_dir, OUTPUT);
  pinMode(mdriver_2_pwm, OUTPUT);
  pinMode(mdriver_3_dir, OUTPUT);
  pinMode(mdriver_3_pwm, OUTPUT);
  pinMode(mdriver_1_enc1, INPUT);
  pinMode(mdriver_1_enc2, INPUT);
  pinMode(mdriver_2_enc1, INPUT);
  pinMode(mdriver_2_enc2, INPUT);
  pinMode(button, INPUT);

  // set the status LED to yellow
  setYellowLED();

  // Zero encoder counters
  encDriveLeft.write(0);
  encDriveRight.write(0);

  // start the serial connection
  Serial.begin(115200);
}

void loop()
{
  current_time_US = micros();
  current_time_MS = millis();
  if (current_time_US - loop_time >= LOGIC_INTERVAL_US)
  { // only run every LOGIC_INTERVAL_US (us)
```

```
// control flashing LEDs
if (current_time_MS - led_time >= LED_INTERVAL_MS)
{ // turn on and off every LED_INTERVAL_MS
if (led_yellow_flashing)
{
if (led_on)
{
      digitalWrite(led_yellow, LOW);
      led_on = false;
}
else
{
      digitalWrite(led_yellow, HIGH);
      led_on = true;
}
}
else if (led_red_flashing)
{
if (led_on)
{
      digitalWrite(led_red, LOW);
      led_on = false;
}
else
{
      digitalWrite(led_red, HIGH);
      led_on = true;
}
}
led_time = current_time_MS;
}

// control state machine
if (current_time_MS - sensor_time >= STATE_INTERVAL_MS)
{                 // every STATE_INTERVAL_MS
refresh_sensors(); // get latest sensor data
switch (global_state)
{
case STATE_IDLE:
// LED should be yellow, all motors are turned off

if (check_weight())
{ // if there is a ball, go to state full
      global_state = STATE_FULL;
      routine3();
}
```

```
if (DRIVE && buttonPressEvent())
{ // if the button has been pressed, start driving
      global_state = STATE_COLLECTING;
      routine1();
}

break;

case STATE_COLLECTING:
// LED should be green, linkage motor is off, drive motors on
if (check_distance())
{ // if there is an object in front of the robot, go to error
      global_state = STATE_ERROR;
      routine2();
}

if (buttonPressEvent())
{ // if the button is pressed, go to idle
      global_state = STATE_IDLE;
      routine4();
}

drive_routine(); // driving

break;

case STATE_FULL:
// LED should be red, all motors are turned off

if (buttonPressEvent())
{ // if the buttin is pressed, start emptying the front scooper
      global_state = STATE_EMPTYING;
      routine5();
}
break;

case STATE_EMPTYING:
// LED should be blinking yellow, linkage motor on, drive off

empty_routine(); // emptying

break;

case STATE_ERROR:
// LED should be blinking red, all motors off
```

```
      // only can leave this state with a system reboot
      break;

      default:
      //Serial.println("SM_ERROR");
      global_state = STATE_ERROR;
      routine2();
      break;
      }
      sensor_time = current_time_MS;
      }
      loop_time = current_time_US;
  }
}

// Event Checkers

// check if the button has been pressed and debounce the signal
bool buttonPressEvent()
{
  if (button_timer_active)
  {
      if (current_time_MS - button_timer >= BUTTON_DEBOUNCE_TIMER)
      {
      buttonIsPressed = false;
      button_timer_active = false;
      }
      return false;
  }
  if (buttonIsPressed)
  {
      button_timer = current_time_MS;
      button_timer_active = true;
      return true;
  }
  return false;
}

// update distance, force, and button sensor values
void refresh_sensors()
{
  hc_distance = distanceSensor.measureDistanceCm();
  force_sensor_reading = analogRead(fsensor);
  int buttonState = digitalRead(button);
  if (buttonState == HIGH)
  {
```

```
      buttonIsPressed = true;
  }
}

// check if there is a ball in the front load bucket
bool check_weight()
{
  if (current_time_MS - full_timer >= FULL_TIME_MS)
  {
      if (force_sensor_reading < FORCE_SENSOR_THRESH)
      {
      return true;
      }
  }
  return false;
}

// check if front distance is too short
bool check_distance()
{
  if (hc_distance < MIN_DISTANCE && hc_distance >= 0)
  {
      return true;
  }
  else
  {
      return false;
  }
}

// Event Service Response

// Routines
void routine1() // start collecting state
{
  Serial.println("Running routine 1");
  setGreenLED();
  stopLinkageMotor();
  startDriveMotors();
}

void routine2() // enter error state
{
  Serial.println("Running routine 2");
  setRedLEDflashing();
  stopDriveMotors();
```

```
  stopLinkageMotor();
}

void routine3() // enter full state
{
  Serial.println("Running routine 3");
  setRedLED();
  stopDriveMotors();
  stopLinkageMotor();
}

void routine4() // enter idle state
{
  Serial.println("Running routine 4");
  setYellowLED();
  stopDriveMotors();
  stopLinkageMotor();
}

void routine5() // enter emptying state
{
  Serial.println("Running routine 5");
  setYellowLEDflashing();
  stopDriveMotors();
  startLinkageMotor();
}

// Subroutines

// control driving
void drive_routine()
{
  if (driving)
  {
      if (current_time_MS - drive_time >= DRIVE_TIME_MS)
      { // after DRIVE_TIME_MS, stop driving and go to idle
      global_state = STATE_IDLE;
      routine4();
      }
      else
      {
      if (current_time_MS - drive_time >= SLOW_DOWN_START_MS &&
current_time_MS - slow_down_time >= SLOW_DOWN_INTERVAL_MS &&
omega_des_local > 0)
      { // after SLOW_DOWN_START_MS milliseconds, start slowing down
      Serial.println("slowing down slowing down slowing down");
```

```
      Serial.println("omega_des: " + String(omega_des_local));
      omega_des_local -= 100;
      slow_down_time = current_time_MS;
      }
      if (current_time_MS - drive_time2 >= DRIVE_FEEDBACK_TIMER)
      { // every DRIVE_FEEDBACK_TIMER (100 ms), perform PI control on
drive motors

      int real_count_right = -encDriveRight.read();
      int real_count_left = -encDriveLeft.read();

      float Kp = 0.5;
      float Ki = 0.6;
      int error_right = omega_des_local - real_count_right;
      int error_left = omega_des_local - real_count_left;
      int Dr = Kp * error_right + Ki * error_sum_right;
      Serial.println("Dr: " + String(Dr));
      int Dl = Kp * error_left + Ki * error_sum_left;
      Serial.println("Dl: " + String(Dl));
      Serial.println("omega: " + String(real_count_right));
      error_sum_right += error_right;
      error_sum_left += error_left;
      //Ensure that the error doesn't get too high
      if (error_sum_right > ERROR_MAX)
      {
            error_sum_right = ERROR_MAX;
      }
      else if (error_sum_right < -ERROR_MAX)
      {
            error_sum_right = -ERROR_MAX;
      }
      if (error_sum_left > ERROR_MAX)
      {
            error_sum_left = ERROR_MAX;
      }
      else if (error_sum_left < -ERROR_MAX)
      {
            error_sum_left = -ERROR_MAX;
      }

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

```
      {
            Dr = -MAX_PWM_VOLTAGE;
      }
      if (Dl > MAX_PWM_VOLTAGE)
      {
            Dl = MAX_PWM_VOLTAGE;
      }
      else if (Dl < -MAX_PWM_VOLTAGE)
      {
            Dl = -MAX_PWM_VOLTAGE;
      }

      analogWrite(mdriver_1_pwm, Dr);
      analogWrite(mdriver_2_pwm, Dl);

      drive_time2 = current_time_MS;
      encDriveLeft.write(0);
      encDriveRight.write(0);
      }
      }
  }
  else
  {
      analogWrite(mdriver_1_pwm, 0);
      analogWrite(mdriver_2_pwm, 0);
  }
}

void empty_routine()
{
  if (emptying)
  {
      if (current_time_MS - drive_time >= LINKAGE_TIME_MS)
      { // after LINKAGE_TIME_MS, turn off the linkage motor, go to idle
      global_state = STATE_IDLE;
      routine4();
      full_timer = current_time_MS;
      }
      else
      { // raise and then lower front scooper bucket
      if (current_time_MS - drive_time2 >= LINKAGE_TIME_1)
      { // up
      digitalWrite(mdriver_3_dir, HIGH);
      analogWrite(mdriver_3_pwm, LINKAGE_PWM_1);
      }
      if (current_time_MS - drive_time2 >= LINKAGE_TIME_2)
```

```
      { // down
      digitalWrite(mdriver_3_dir, LOW);
      analogWrite(mdriver_3_pwm, LINKAGE_PWM_2);
      }
      if (current_time_MS - drive_time2 >= LINKAGE_TIME_21)
      { // down
      digitalWrite(mdriver_3_dir, LOW);
      analogWrite(mdriver_3_pwm, LINKAGE_PWM_21);
      }
      if (current_time_MS - drive_time2 >= LINKAGE_TIME_3)
      { // up
      digitalWrite(mdriver_3_dir, HIGH);
      analogWrite(mdriver_3_pwm, LINKAGE_PWM_3);
      }
      if (current_time_MS - drive_time2 >= LINKAGE_TIME_4)
      { // down
      digitalWrite(mdriver_3_dir, LOW);
      analogWrite(mdriver_3_pwm, LINKAGE_PWM_4);
      }
      if (current_time_MS - drive_time2 >= LINKAGE_TIME_5)
      { // up
      digitalWrite(mdriver_3_dir, HIGH);
      analogWrite(mdriver_3_pwm, LINKAGE_PWM_5);
      }
      if (current_time_MS - drive_time2 >= LINKAGE_TIME_6)
      { // up
      digitalWrite(mdriver_3_dir, HIGH);
      analogWrite(mdriver_3_pwm, LINKAGE_PWM_6);
      }
      if (current_time_MS - drive_time2 >= LINKAGE_TIME_61)
      { // up
      digitalWrite(mdriver_3_dir, HIGH);
      analogWrite(mdriver_3_pwm, LINKAGE_PWM_61);
      }
      if (current_time_MS - drive_time2 >= LINKAGE_TIME_62)
      { // up
      digitalWrite(mdriver_3_dir, HIGH);
      analogWrite(mdriver_3_pwm, LINKAGE_PWM_62);
      }
      }
  }
  else
  { // make sure linkage motor is off after timer
      analogWrite(mdriver_3_pwm, 0);
  }
}
```

```
// turn on the green LED
void setGreenLED()
{
  digitalWrite(led_green, HIGH);
  digitalWrite(led_yellow, LOW);
  digitalWrite(led_red, LOW);
  led_yellow_flashing = false;
  led_red_flashing = false;
}

// turn on the yellow LED
void setYellowLED()
{
  digitalWrite(led_green, LOW);
  digitalWrite(led_yellow, HIGH);
  digitalWrite(led_red, LOW);
  led_yellow_flashing = false;
  led_red_flashing = false;
}

// turn on the red LED
void setRedLED()
{
  digitalWrite(led_green, LOW);
  digitalWrite(led_yellow, LOW);
  digitalWrite(led_red, HIGH);
  led_yellow_flashing = false;
  led_red_flashing = false;
}

// turn on the yellow LED, flashing
void setYellowLEDflashing()
{
  digitalWrite(led_green, LOW);
  digitalWrite(led_yellow, HIGH);
  digitalWrite(led_red, LOW);
  led_yellow_flashing = true;
  led_red_flashing = false;
  led_on = true;
  led_time = current_time_MS;
}

// turn on the red LED, flashing
void setRedLEDflashing()
{
```

```
  digitalWrite(led_green, LOW);
  digitalWrite(led_yellow, LOW);
  digitalWrite(led_red, HIGH);
  led_yellow_flashing = false;
  led_red_flashing = true;
  led_on = true;
  led_time = current_time_MS;
}

// start driving
void startDriveMotors()
{
  // left motor
  digitalWrite(mdriver_1_dir, LOW);
  analogWrite(mdriver_1_pwm, 0);
  // right motor
  digitalWrite(mdriver_2_dir, HIGH);
  analogWrite(mdriver_2_pwm, 0);

  driving = true;
  drive_time = current_time_MS;
  drive_time2 = current_time_MS;
  encDriveLeft.write(0);
  encDriveRight.write(0);
  error_sum_right = 0;
  error_sum_left = 0;
  omega_des_local = OMEGA_DES_DRIVE;
  slow_down_time = current_time_MS;
}

// stop driving
void stopDriveMotors()
{
  // left motor
  digitalWrite(mdriver_1_dir, LOW);
  analogWrite(mdriver_1_pwm, 0);
  // right motor
  digitalWrite(mdriver_2_dir, LOW);
  analogWrite(mdriver_2_pwm, 0);

  driving = false;
}

// start raising front scooper bucket
void startLinkageMotor()
{
```

```
  digitalWrite(mdriver_3_dir, HIGH);
  analogWrite(mdriver_3_pwm, LINKAGE_PWM_0);

  emptying = true;
  drive_time = current_time_MS;
  drive_time2 = current_time_MS;
}

// turn off the front scooper bucket
void stopLinkageMotor()
{
  digitalWrite(mdriver_3_dir, LOW);
  analogWrite(mdriver_3_pwm, 0);

  emptying = false;
}
```