ME 102B - Final Report

Window Cleaning Robot for Highrise Glass Buildings

Braeden Swidenbank, Kyle Pedersen

University of California, Berkeley, December 12, 2021

Opportunity

Our selected opportunity is to provide a way to clean windows without the requirement of a human. We see this as a way to improve worker safety and productivity.

High Level Strategy

Initially, we envisioned our cleaner to be suspended at each corner by four cables. This rope would be winched in and out by four separate gearmotors allowing the robot to move. The cleaner would push against the window by manipulating its center of gravity. In the end, because of cost and difficult kinematics we opted for a cartesian system with linear motion in the x axis (along the building) and a winch to raise and lower the cleaner in the y axis (up and down the building). Additionally, we used powered fans to press the cleaner against the window instead of a weight shift mechanism.

Assembly and Subsystems

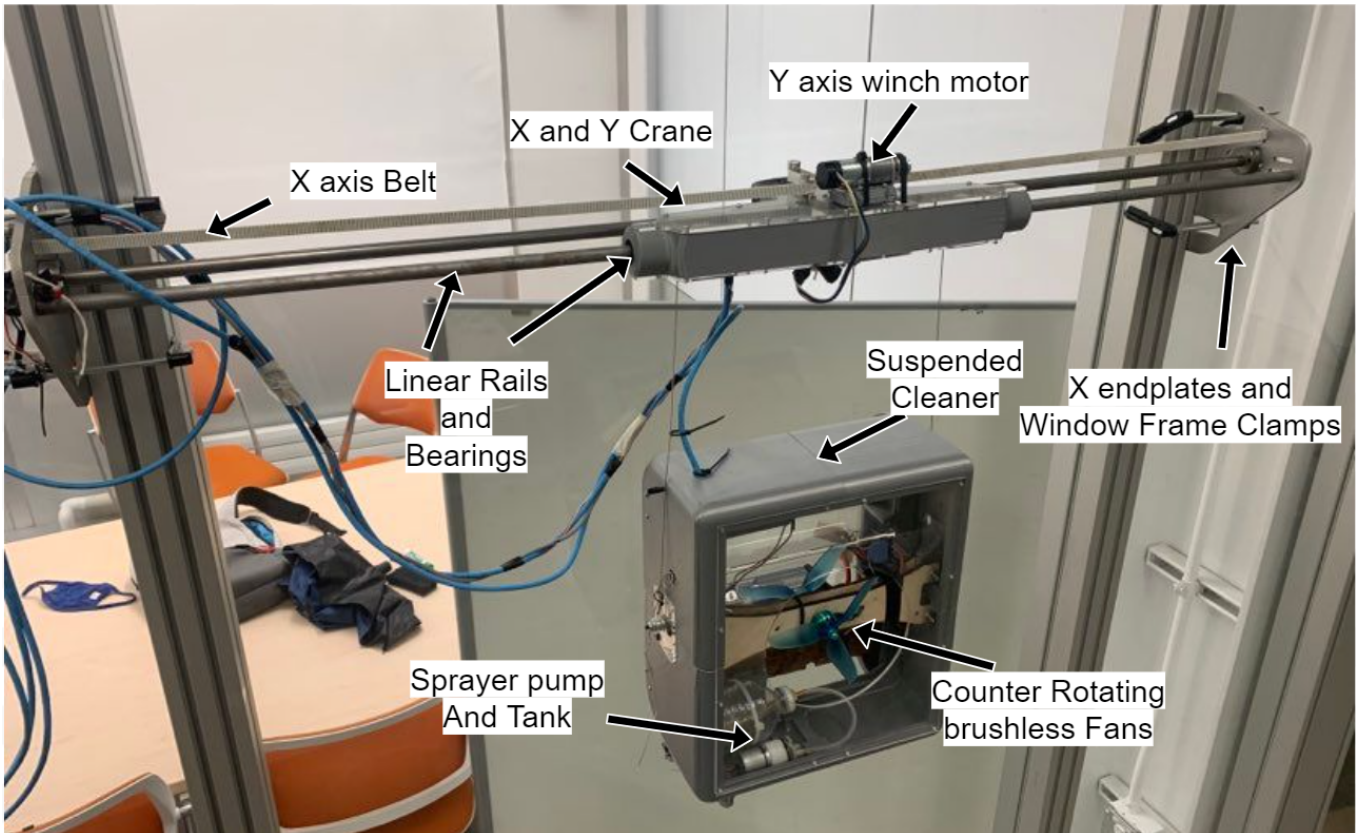

Figure 1. Integrated assembly clamped to a window in Jacobs Hall 310.

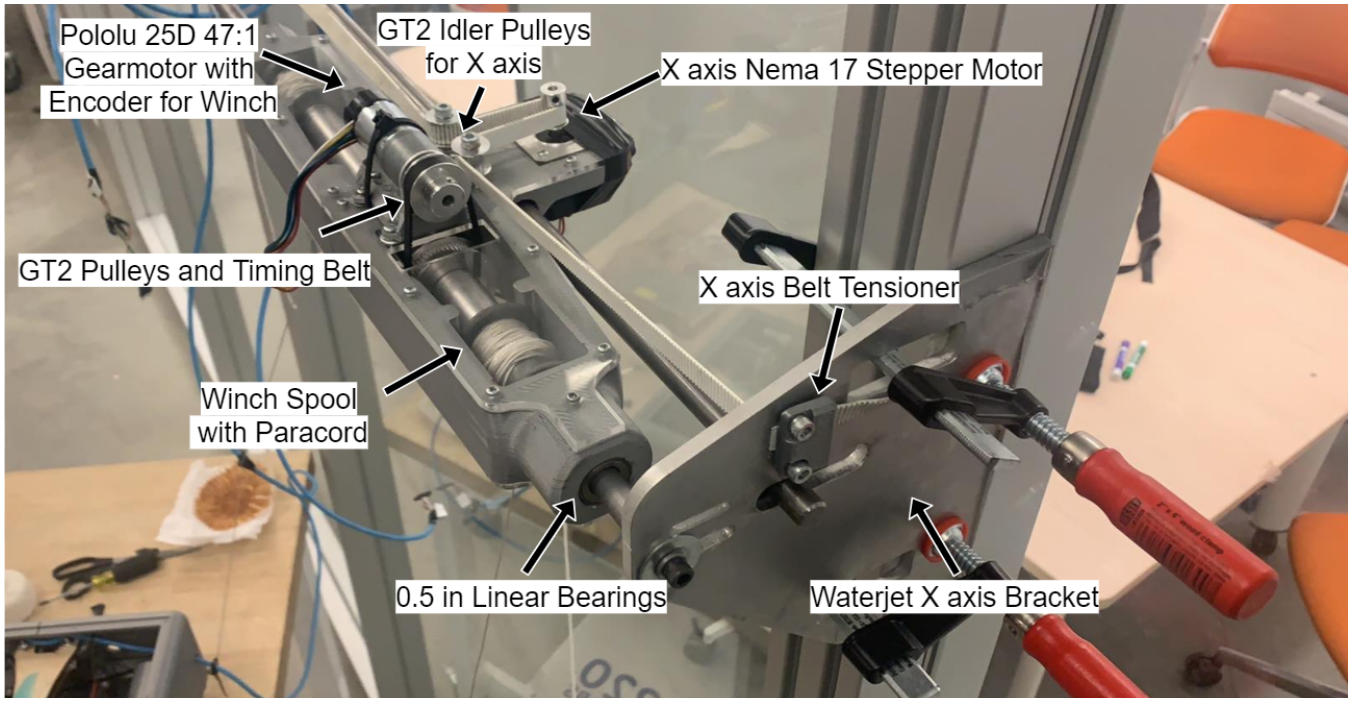

Figure 2. X linear motion, y axis winch, and window mounting

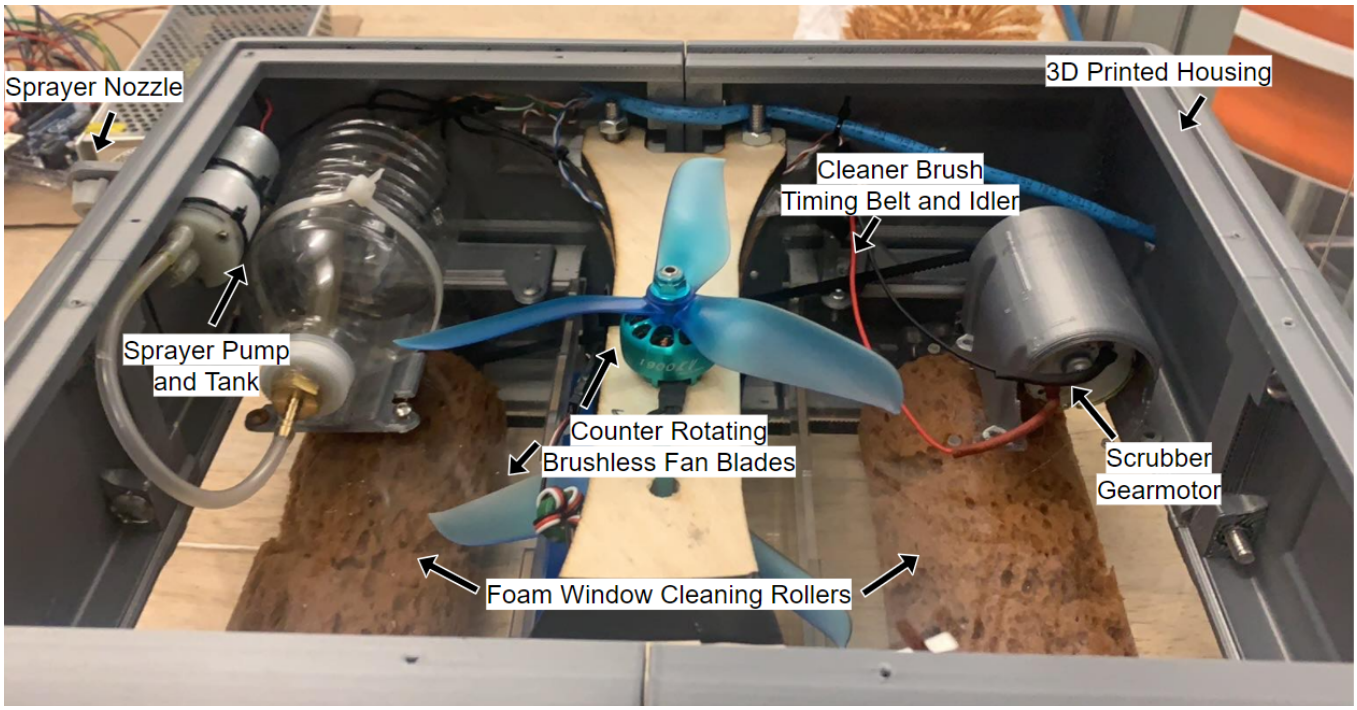

Figure 3. Sprayer pump and tank, Counter rotating brushless fans, cleaner pads and drivetrain.

Function-Critical Elements

The most crucial element in this project was the gearmotor winch that lifts the cleaner up and down. To ensure that the motor would not stall we calculated the required torque with equation 1. The weight of the cleaning unit was estimated to be 3 kg and a safety factor of 2 was utilized in the calculation:

$$T = r * F \quad (1)$$

$$T = (0.025\ m) * (3\ kg * 9.8\ m/s)$$

$$T = 0.75\ N * 2$$

Our results indicated that a gearmotor with 16 kg*cm torque would meet the requirements. A 47:1 Pololu 25D was chosen. A Nema 17 with 3.2 kg*cm torque output was chosen for the x axis due to friction being the only substantial force in the x direction. For the scrubbing motor RPM was the most important factor (for cleaning purposes). A 12v 1000 RPM motor was chosen and geared down 2:1 to achieve a cleaner speed of 500 RPM (Ideal for our application).

Circuit Diagram and State Diagram

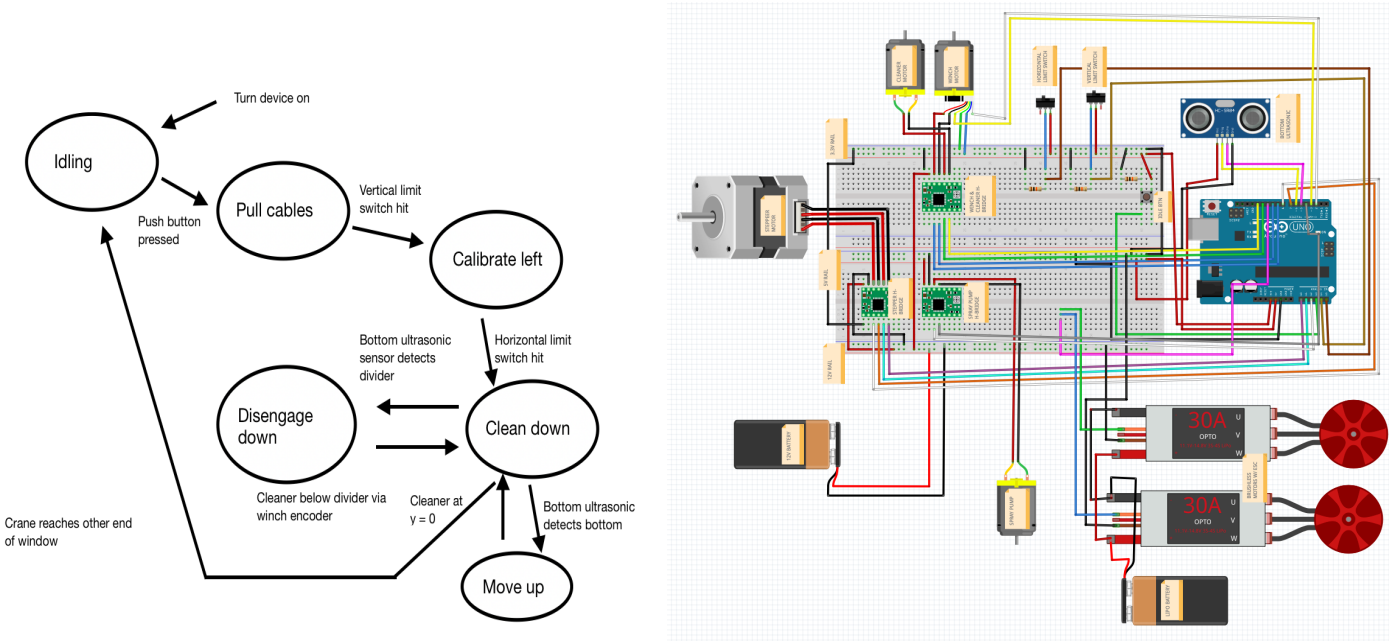

Figure 4. State transition diagram and wiring diagram. NOTE: The circuit diagram includes the water sprayer. In reality, the sprayer pump stopped working (from running dry) and could not be demonstrated. A larger wiring diagram is in Appendix D.

Conclusion

A strategy that worked well is isolating certain subsystems and repeatedly testing them on their own so that when the entire device was integrated, more of the focus could be on assessing the performance of the device as a whole. While this did work—we reliably had each subsystem working well—we integrated and tested the entire device a bit too late, and overlooked certain aspects of its mechanical performance. An example is the weight distribution issue of the cleaner. We spent so much time building, designing, and testing each individual subsystem, that when we fully integrated everything, we found that the cleaner had too much weight on its left side. Had we known this earlier, we could have moved around certain components.

Appendix:

A) Bill of Materials

Crane Components

Part	Description	QTY.
47:1 Gear motor 25D 12V with Encoder	Winch motor inside Crane	1
40T GT2 Pulley 4mm Bore	Winch motor drive train	1
60T Gt2 Pulley 7/8in Bore	Winch motor drive train	1
200mm GT2 Timing Belt 5mm Width	Winch motor drive train	1
15mm x 21mm x 4mm Bearing	Winch torque bar support	2
7/8 O.D 5/8 I.D x 11.5 in Alu Tube	Torque bar stock	1
0.5 in Linear Bearings	Crane linear motion	3
Laser Cut Acrylic Top and Bottom Plates	Crane frame	2
3D Printed Crane Frame PLA	Crane frame	2
Nema 17 Stepper Motor	X linear motion	1
20T GT2 Pulley 5mm Bore	X linear motion	1
GT2 Idler Pulley 10mm Width	X linear motion	2
GT2 Belt 10mm Width x 55 in Long	X linear motion	1
0.5 in x 48 in Steel Rod	X linear motion guides	2
Clamping Bracket 0.25 in Alu Plate	Mounts crane to window (water jet)	2
3in Clamps	Connects window bot to window	4
Belt Tensioning Brackets	3D printed brackets for x belt tension	2
0.5 in Lock Collars	Keeps linear rail constrained	2
Winch Rope Spools	3D printed spools for winch	2
End Stop Switches	Trigger switches for X and Y	2
1.18mm Paracord	Winch Rope (Ft)	40

Cleaner Components

Part	Description	QTY.
Grearisan 12v 1000rpm Motor	Drives cleaner brushes	1
20T GT2 Pulley 0.25 in Bore	Cleaner drive pulley	1
40T GT2 Pulley 0.25 in Bore	Cleaner drivetrain pulley	2
610 mm GT2 Belt 5mm Width	Cleaner drivetrain	1
Paint Roller Frame	Mounts for cleaner brushes	2
0.25 in threaded rods 12in long	Cleaner brush axles	2
0.25 in x 5/8 in x .25 in Bearings	Cleaner motor bearings	4
Acrylic Panels	Structure and water control	6
3D printed structure PLA	Main cleaner structure	4
9in Paint Frame 1 in nap	Rotating cleaning pads	2
Mini Water Bottle	Sprayer tank	1
1/8 in Pipe Nipple	Sprayer Tank	1
5v Pump and Nozzle	Sprayer Tank	1
5mm Idler Pulley	String Attachment Pulleys	2
3s 5000 MAH Battery	Propeller power	1
1900 kv Brushless Motors	Propeller powertrain	2
6 in Propellers	CCW and CW propellers for thrust	2
Cat 5 Ethernet	Sensor and power conduit (ft)	40
Ultrasonic Sensor	Senses obstacles and the floor	1
12v 5a Power Supply	Power for the entire system	1

B) CAD

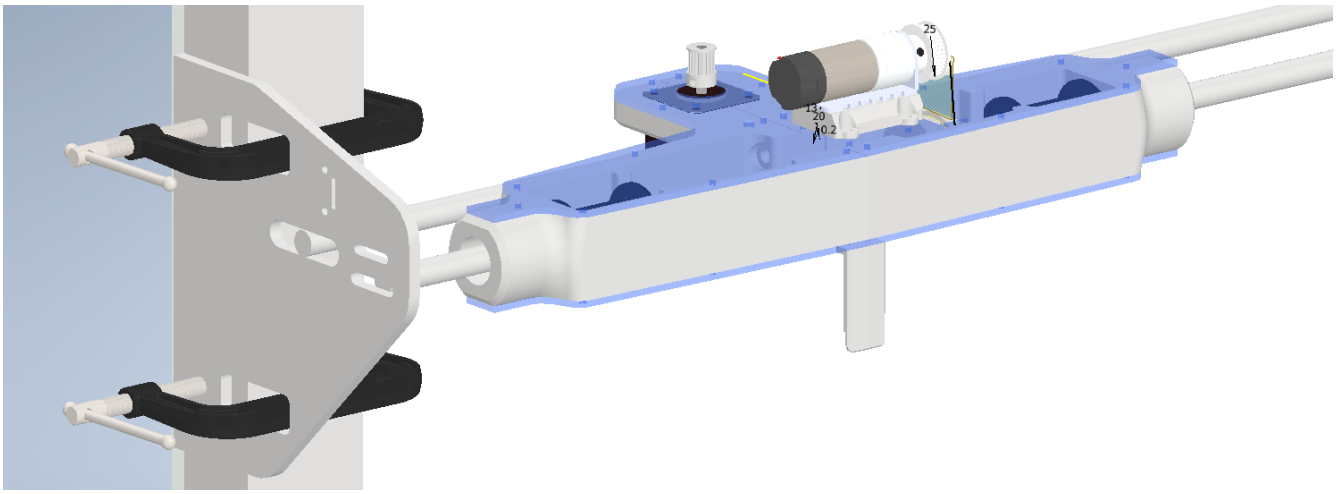

Figure 1b. X axis and winch crane clamped to a window frame modeled after an actual window in Jacobs hall room 310.

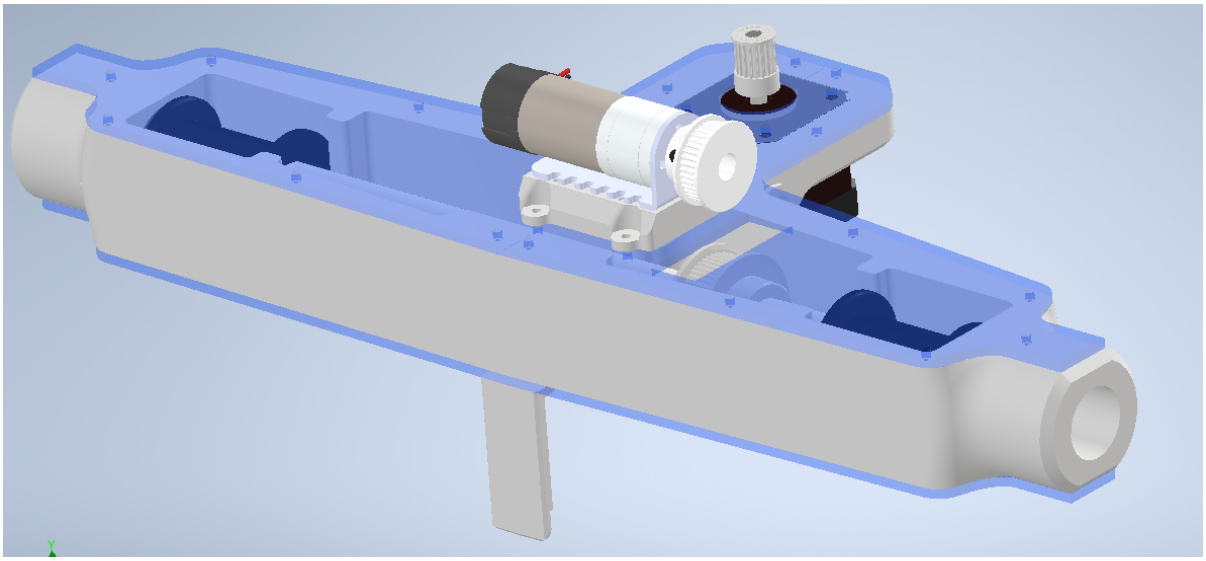

Figure 2b. Winch crane assembly highlighting gear motor drive train (Note: 200mm GT2 timing belt is not pictured). Winch spools are colored in black. Stepper motor and pulley are visible in the back (Note: 10mm GT2 belt is not pictured)

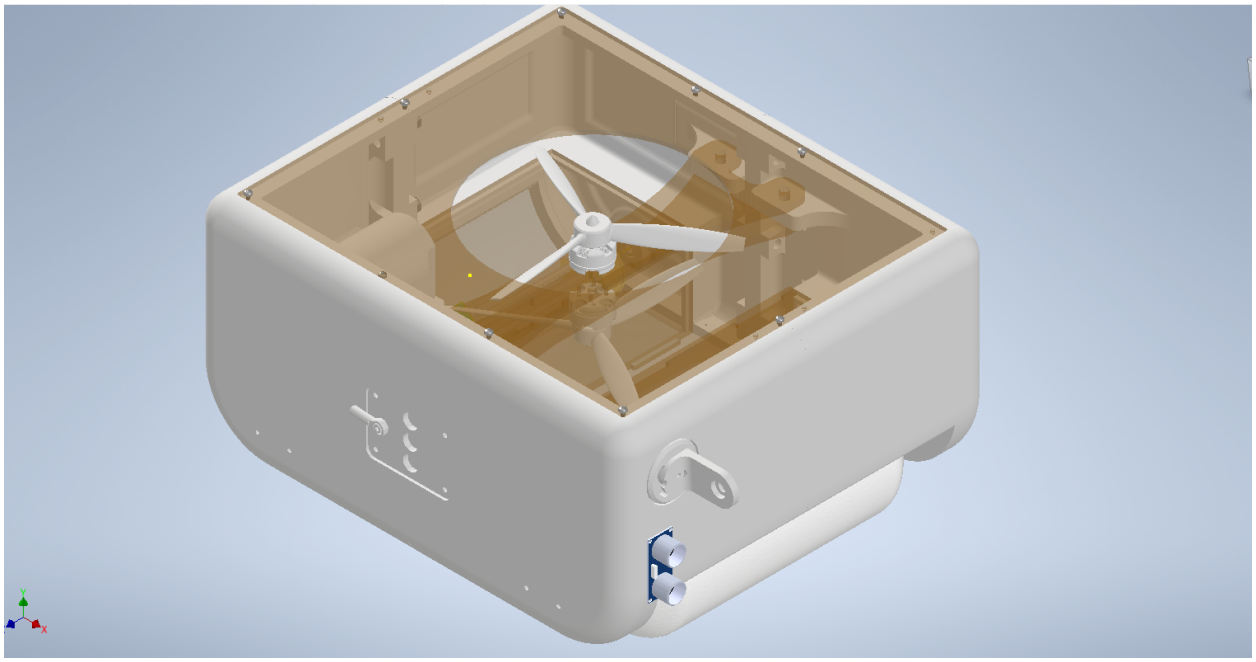

Figure 3b. Cleaner assembly Isometric view highlighting acrylic and 3D printed construction. Adjustable nozzle holder and the ultrasonic sensor are visible.

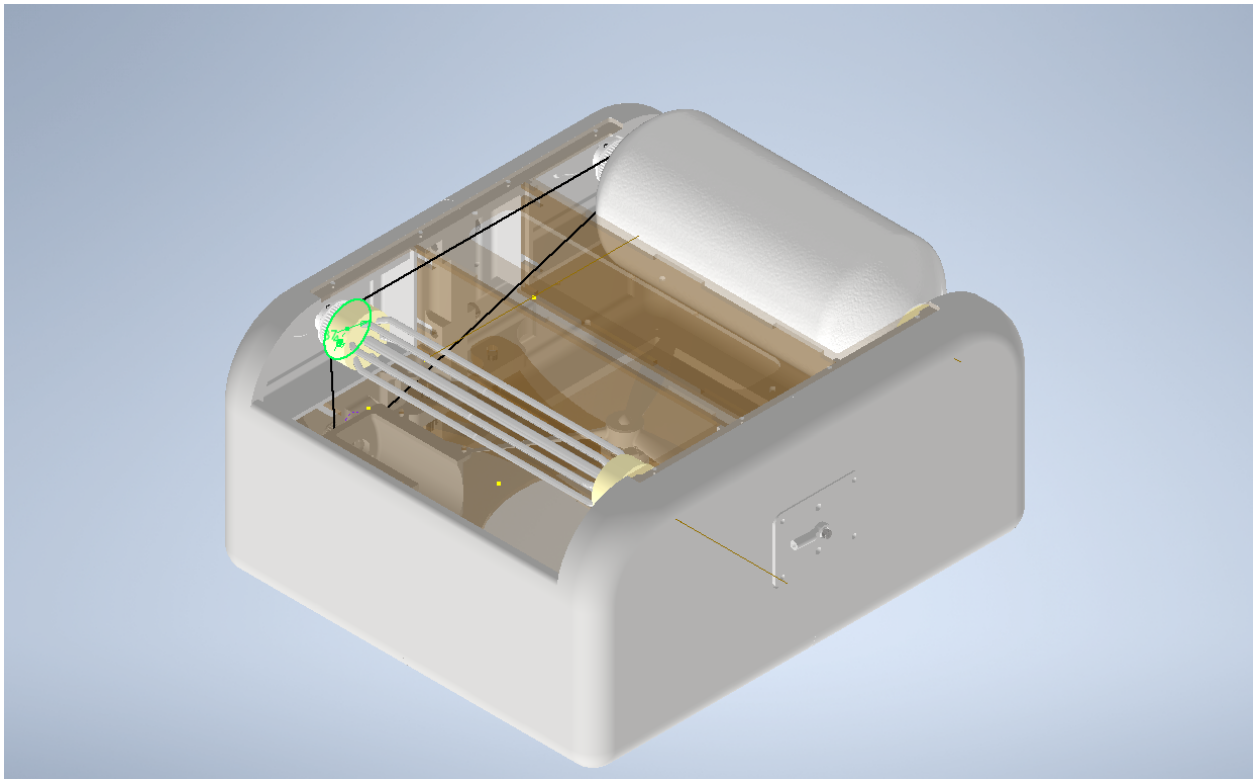

Figure 4b. Cleaner assembly from beneath with one scrubbing pad removed showing paint roller frame. The cleaner drivetrain is visible with the 610 mm timing belt denoted by the black sketch lines.

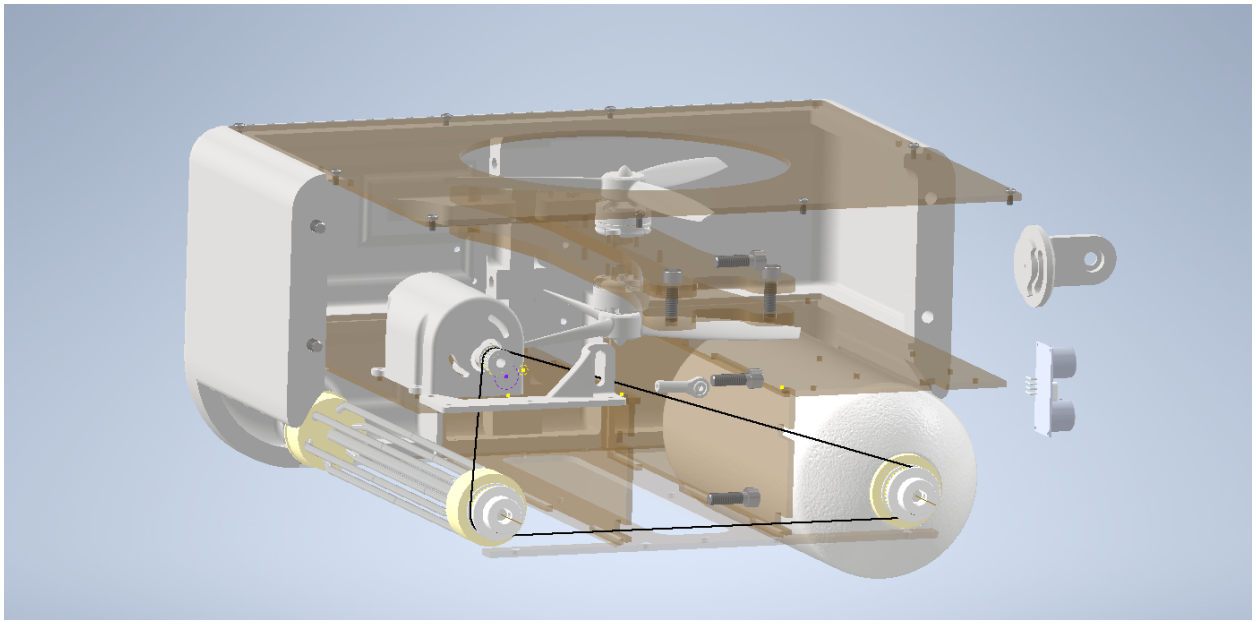

Figure 5b. With part of the structure removed the cleaner drivetrain is fully visible. Notice the slotted feature designed to allow for belt tensioning with an idler pulley. Additionally, the counter rotating brushless fan blades can be seen.

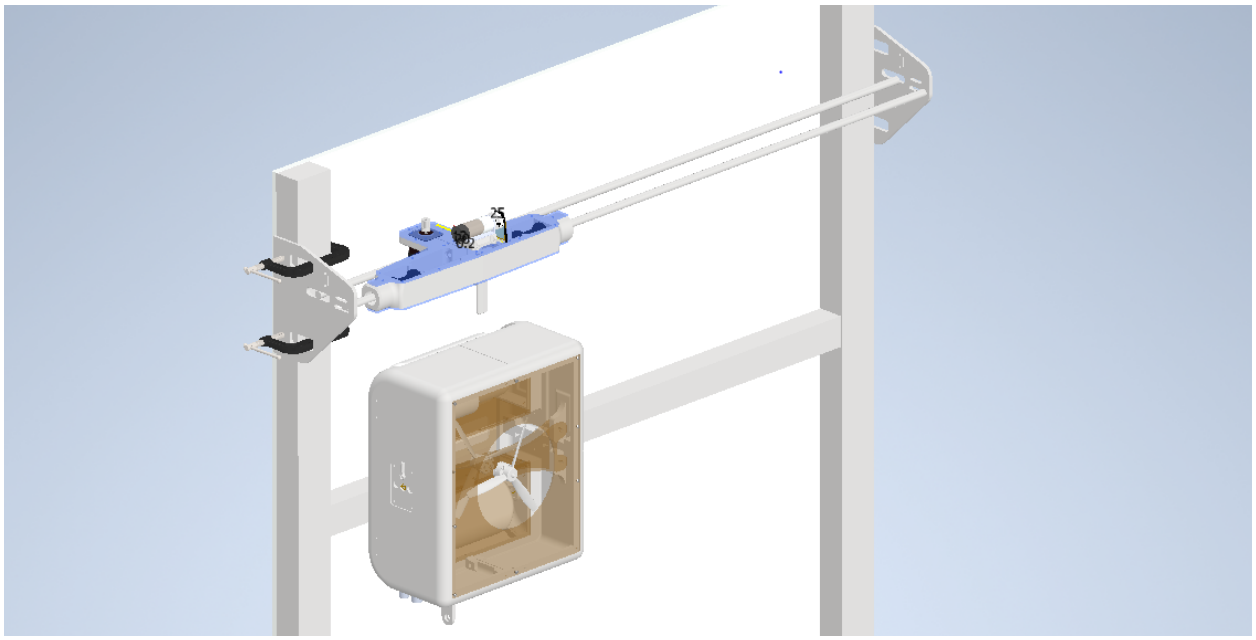

Figure 6b. Crane and cleaner assemblies are constrained to the window frame. Belts and strings are not pictured.

C) Code

NOTE: On the morning of the demo, our Arduino Mega short-circuited and failed to operate. As such, we had to eliminate some of the interrupts we used for the limit switches. Since the Uno only has two interrupt pins, these were used for the most important interrupt we needed, which was the winch's motor encoder.

```
//STATE
volatile int state;
//RELAY CALIBRATION
volatile bool relayCal = false;
volatile int motorSteps = 0;

volatile int startCleanDown;

//IDLE BUTTON
const int idleButton = A3;

//CONDITION CONSTANTS
int maxMotorSteps = 2;
int maxVerticalDistanceTraveled = 15000;
int maxDisengageDistanceTraveled = 5000;
float ultrasonicThreshold = 10.0;

//VERTICAL LIMIT SWITCH
volatile bool verticalLimitSwitchPress;
const int verticalLimitSwitch = 18;

//HORIZONTAL LIMIT SWITCH
volatile bool horizontalLimitSwitchPress;
const int horizontalLimitSwitch = 19;

//BOTTOM ULTRASONIC SENSOR
#define SOUND_SPEED 0.034
long duration;
float distanceCm;
const int bottomTrigPin = 6;
const int bottomEchoPin = 5;

// POLULU WINCH MOTOR
#define MotFwd  9  // Motor Forward pin
#define MotRev  10 // Motor Reverse pin
int encoderPin1 = 2; //Encoder Output 'A' must connected with intreput pin of arduino.
int encoderPin2 = 3; //Encoder Otput 'B' must connected with intreput pin of arduino.
```

```
volatile int lastEncoded = 0; // Here updated value of encoder store.
volatile long encoderValue = 0; // Raw encoder value from interrupt
const int MAX_PWM_VOLTAGE = 255;
float omegaDes = 20.0;
int omegaSpeed = 0;
int error = 0;
int D = 0;
int errorsum = 0;
int encoderLast = 0;
float w; //angular velocity
int delta_encoder;
int dt;
int PWM = 255;

volatile long startEncoderVal;
long deltaEncoder;

// STEPPER MOTOR
#include <Stepper.h>
#define STEPS 200
Stepper stepper(STEPS, A0, A1, 7, 8);

// MOTOR & CLEANER PINS
const int dcMotorRelayPin1 = 12;
const int dcMotorRelayPin2 = 13;
const int sprayerRelayPin1 = 4;
const int sprayerRelayPin2 = A2;

// BRUSHLESS MOTORS
#include <Servo.h>
Servo myservo;
int throttle = 1700;


void setup() {
  Serial.begin(115200);
  state = 8; //dummy variable to go to default idling state

  //IDLE BUTTON (NO NEED FOR INTERRUPT)
  pinMode(idleButton, INPUT);

  //VERTICAL LIMIT SWITCH
  pinMode(verticalLimitSwitch, INPUT_PULLUP);

  //HORIZONTAL LIMIT SWITCH
  pinMode(horizontalLimitSwitch, INPUT_PULLUP);

  //BOTTOM ULTRASONIC SENSOR
  pinMode(bottomTrigPin, OUTPUT);
  pinMode(bottomEchoPin, INPUT);

  //POLULU WINCH MOTOR
  pinMode(MotFwd, OUTPUT);
  pinMode(MotRev, OUTPUT);
  pinMode(encoderPin1, INPUT_PULLUP);
  pinMode(encoderPin2, INPUT_PULLUP);
  digitalWrite(encoderPin1, PWM); //turn pullup resistor on
  digitalWrite(encoderPin2, PWM); //turn pullup resistor on
  //call updateEncoder() when any high/low changed seen
  //on interrupt 0 (pin 2), or interrupt 1 (pin 3)
  attachInterrupt(0, updateEncoder, CHANGE);
  attachInterrupt(1, updateEncoder, CHANGE);

  // STEPPER MOTOR
  stepper.setSpeed(200);

  //CLEANER & DC MOTOR PINS
  pinMode(dcMotorRelayPin1, OUTPUT);
  pinMode(dcMotorRelayPin2, OUTPUT);
  pinMode(sprayerRelayPin1, OUTPUT);
  pinMode(sprayerRelayPin2, OUTPUT);

  // BRUSHLESS MOTORS
  myservo.attach(11);
  delay(1);
  myservo.write(1000);
  delay(5000);

}
```

```
void loop() {
  switch (state) {
    case 1: // PULL CABLES
      if(verticalSwitch()) {
        pullCablesToCalibrateLeft();
      }
      break;
    case 2: // CALIBRATE LEFT
      stepCalibrationWidth();
      if(horizontalSwitch()) {
          calibrateLeftToCleanDown();
        }
      break;
    case 3: //CLEAN DOWN
      turnOnWinchDown(omegaDes); //PI control
      if (bottomUltrasonic() < ultrasonicThreshold) { //less than 5 cm & divider
        if (deltaCleanDown() > maxVerticalDistanceTraveled) { //distance to bottom
          cleanDownToMoveUp();
        } else {
          cleanDownToDisengageDown();
        }
      }
      else if (motorSteps > maxMotorSteps) {
        cleanDownToIdling();
      }
      break;
    case 4: //DISENGAGE DOWN
      Serial.println("DISENGAGE DOWN");
      if (deltaEncoderVal() > maxDisengageDistanceTraveled) { //vertical distance travelled
        disengageDownToCleanDown();
      }
      break;
    case 5: //MOVE UP
      Serial.println("MOVE UP");
      if (verticalSwitch()) {
        moveUpToCleanDown();
      }
      break;
    default: //IDLING
      if (idlingButton()) {
        idleToPullCables();
      }
      break;
  }
}
`
,
//================================================================================
//==================================CHECKER FUNCTIONS=============================
//================================================================================

bool verticalSwitch() {
  return digitalRead(verticalLimitSwitch);
}

bool horizontalSwitch() {
  return digitalRead(horizontalLimitSwitch);
}

bool idlingButton() {
  return digitalRead(idleButton);
}

int deltaEncoderVal() {
  deltaEncoder = encoderValue - startEncoderVal;
  return abs(deltaEncoder);
}

int deltaCleanDown() {
  return abs(encoderValue - startCleanDown);
}


//================================================================================
//================================================================================
//================================================================================
```

```
//=============================================================================================
//==================================SERVICE FUNCTIONS==========================================
//=============================================================================================
void idleToPullCables() {
  state = 1;
  turnOnWinchUp(omegaDes);
  Serial.println("Starting winch motor");
}

void pullCablesToCalibrateLeft() {
  turnOffWinch();
  state = 2;
  Serial.println("Calibrating left");
}

void calibrateLeftToCleanDown() {
  turnOnSprayer();
  delay(20);
  horizontalLimitSwitchPress = false;
  turnOnWinchDown(omegaDes);
  delay(20);
  turnOnDC();
  startCleanDown = encoderValue;
  state = 3;
  Serial.println("Sprayer On");
  Serial.println("Clean down");
  turnOnProps(throttle);
}

void cleanDownToMoveUp() {
    turnOnWinchUp(omegaDes); //winch now goes upwards
    turnOffDC();
    turnOffSprayer();
    turnOffProps();
    Serial.println("Turning Off Brushless Motors");
    state = 5;
}

void cleanDownToDisengageDown() {
  startEncoderVal = encoderValue; //to keep track of winch
  Serial.println("Turning Off Brushless Motors");
  turnOffProps();
  state = 4;
}

void disengageDownToCleanDown() {
  turnOnProps(throttle);
  turnOnDC();
  Serial.println("Restarting Brushless Motors");
  state = 3;
}
```

```
void moveUpToCleanDown() {
  stepColumnWidth();
  motorSteps++;
  turnOnProps(throttle);
  turnOnDC();
  state = 3;
}

void cleanDownToIdling() {
  turnOffDC();
  turnOffSprayer();
  turnOffWinch();
  turnOffProps();
  state = 8; // turn everything off because stepper at end of crane, state is now idling
}

//=============================================================================================
//============================== END SERVICE FUNCTIONS ========================================
//=============================================================================================



//HELPER FUNCTIONS

float bottomUltrasonic() {
  digitalWrite(bottomTrigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(bottomTrigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(bottomTrigPin, LOW);
  duration = pulseIn(bottomEchoPin, HIGH);
  distanceCm = duration * SOUND_SPEED/2;
  return distanceCm;
}

void updateEncoder(){
  int MSB = digitalRead(encoderPin1); //MSB = most significant bit
  int LSB = digitalRead(encoderPin2); //LSB = least significant bit

  int encoded = (MSB << 1) |LSB; //converting the 2 pin value to single number
  int sum  = (lastEncoded << 2) | encoded; //adding it to the previous encoded value

  if(sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011) encoderValue --;
  if(sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000) encoderValue ++;

  lastEncoded = encoded; //store this value for next time

}


void turnOnWinchUp(float omegaDes) {
 delta_encoder = abs(encoderValue - encoderLast);
 encoderLast = encoderValue;
 dt = 100;
 w = delta_encoder / dt;
 Serial.println(w);
 error = omegaDes - w;
 D = 60 * error +  7 * errorsum;
 errorsum = errorsum + error;

  if (D > MAX_PWM_VOLTAGE) {
  D = MAX_PWM_VOLTAGE;
  errorsum = errorsum - error;
  } else if (D < -MAX_PWM_VOLTAGE) {
  D = -MAX_PWM_VOLTAGE;
  errorsum = errorsum - error;
  } else if (D == 0) {
    analogWrite(MotFwd, LOW);
    analogWrite(MotRev, LOW);
  }
  if (D > 0) {
    analogWrite(MotFwd, LOW);
    analogWrite(MotRev, D);
  } else if (D < 0) {
    analogWrite(MotRev, LOW);
    analogWrite(MotFwd, -D);
  }
 Serial.println(encoderValue);
 delay(100);
}
```

```
void turnOnWinchDown(float omegaDes) {
   delta_encoder = encoderValue - encoderLast;
   encoderLast = encoderValue;
   dt = 50;
   w = delta_encoder / dt;
   error = omegaDes - w;
   D = 60 * error +  7 * errorsum;
   errorsum = errorsum + error;

    if (D > MAX_PWM_VOLTAGE) {
    D = MAX_PWM_VOLTAGE;
    errorsum = errorsum - error;
    } else if (D < -MAX_PWM_VOLTAGE) {
    D = -MAX_PWM_VOLTAGE;
    errorsum = errorsum - error;
    } else if (D == 0) {
      analogWrite(MotFwd, LOW);
      analogWrite(MotRev, LOW);
    }
  if (D > 0) {
    analogWrite(MotFwd, D);
    analogWrite(MotRev, LOW);
  } else if (D < 0) {
    analogWrite(MotRev, -D);
    analogWrite(MotFwd, LOW);
  }
   delay(50);
}

void turnOffWinch() {
  digitalWrite(MotFwd, 0);
  digitalWrite(MotRev, 0);
}

void stepColumnWidth() {
  stepper.step(1125); // step size equivalent to 9 in. roller length
}

void stepCalibrationWidth() {
  stepper.step(-10);
}

void turnOnDC() {
  digitalWrite(dcMotorRelayPin1, LOW);
  digitalWrite(dcMotorRelayPin2, HIGH);
}
void turnOffDC() {
  digitalWrite(dcMotorRelayPin1, LOW);
  digitalWrite(dcMotorRelayPin2, LOW);
}


void turnOnSprayer() {
  digitalWrite(sprayerRelayPin1, HIGH);
  digitalWrite(sprayerRelayPin2, LOW);
}

void turnOffSprayer() {
  digitalWrite(sprayerRelayPin1, LOW);
  digitalWrite(sprayerRelayPin2, LOW);
}

void turnOnProps(int throttle) {
  myservo.write(throttle);
}

void turnOffProps() {
  myservo.write(1000);
}
```

D) Wiring Diagram Enlarged