Manual: How to build a Plant Nanny

Guillermo Fernandez, Preston Hong, Dafne Renteria, Sebastian Quiroz

University of California, Berkeley
December 15, 2023

Project Opportunity

Inspired to tackle the pervasive plant death in the CALNERDS STEM student center due to insufficient watering during long breaks; Our team was driven to devise an automatic irrigation system that allows the user to have a plant nanny that supports up to 4 plants while they're away!

1. High Level Strategy

Our system aims to automate plant watering based on soil moisture levels. When the soil's moisture falls below the set threshold, the drivetrain, watering, and passive sensing subsystems collaborate to initiate watering.

While our initial strategy was quite ambitious we must admit; we were able to realize the core systems to achieve the main required functionalities. Despite initial ambitious plans, we streamlined the implementation by using a single sensor instead of four and chose to leverage gravity to drive the water flow. For example, to support 4 plants (4x sensors), it would require us to implement a complex rule set such that the system could handle a situation where 2+ plants require water (prioritization). Since the implementation of each required complex ideology and coding beyond our group's capabilities within the deadline of the project we only implemented 1.

Additionally, another deviation from the original strategy was the design choice was the leverage of gravity to push the flow of water instead of an additional motorized pump—this reduced both design complexity and energy consumption. However, this consideration required us to limit our water reservoir so the motor could still drive the loads because the reservoir had to be elevated above the solenoid (higher than the plant). Ultimately, we accepted this compromise as we could still meet the project requirements but also save money on materials.

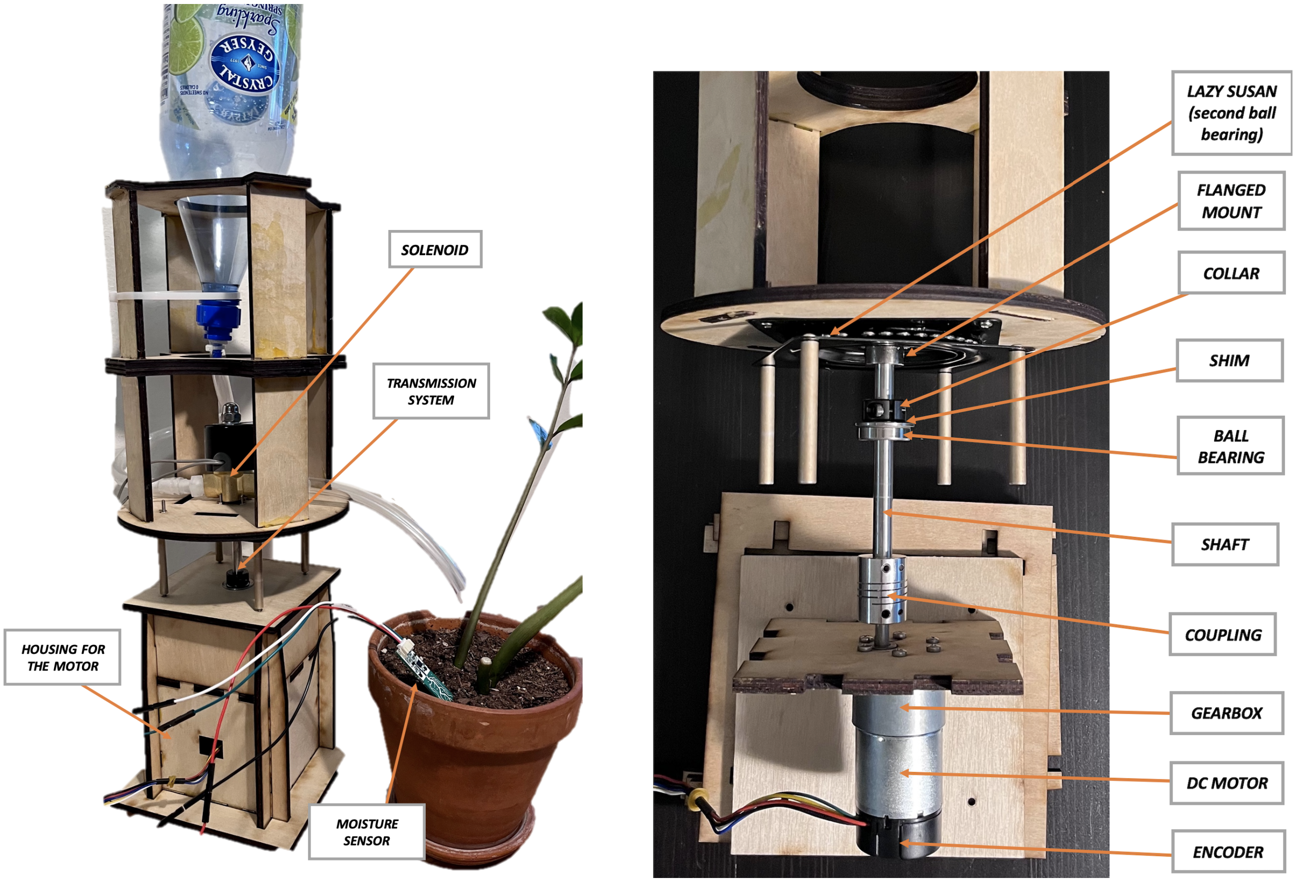
2. Integrated Physical Device

The main components of the system include the motor, solenoid, sensors, motor housing and reservoir (figure 1). To achieve this, we have designed a wooden housing that will serve as a support for most of the system elements, as well as protect the system's transmission and, most importantly, the motor.

The tube that supplies water to irrigate the plants will always start from a predetermined position and will remain in that state (start) until the sensor detects that the humidity of a plant is below the preset value. At this point, the motor will rotate. Once it reaches this position and the motor stops, the solenoid is activated opening the valve to allow water to flow through the tube. The system is designed to water each plant for a specific time, adjustable according to the user's preferences and conditioned by factors such as the type of plant, its size, the weather, the climate, etc.

Our irrigation system has a meticulously designed drive system, which features a seamless integration of components such as the Lazy Susan, collar, shim, ball bearing, shaft, coupling, flanged mount and a single compact unit comprising the Gearbox, DC motor and encoder.

At the heart of our drive system is the integrated unit of gearbox, DC motor and encoder. This compact assembly serves as the power source and control center of the system. To achieve the transmission of the movement from the motor shaft to the rotating platform, it will be necessary to use a shaft of considerable dimensions. This will allow us to reach a suitable height for the solenoid, from which we will be able to adjust the watering of the plants, while ensuring a solid, safe and efficient transmission system. For the connection of both shafts, we will use a coupling, whose main function is to guarantee the transmission of the movement and to absorb the vibrations in the union between these two elements.



(a) General picture with the main elements of the system

(b) Elements of the transmission system

Figure 1: Integrated Physical Device

The collar and shim, strategically positioned next to each other, provide stability and alignment, ensuring that the rotational movement remains consistent. In addition, the ball bearings contribute significantly to the stability of the system by reducing friction, which in turn contributes to the longevity of the components and the system as a whole. This structured and well-coordinated approach to motion transmission is essential to ensure the efficient and long-lasting operation of our automated irrigation system. Within the sequence of elements that make up our drive system, the Lazy Susan is the last component. This element is considered as our second ball bearing. When a shaft is supported by two bearings, it experiences improved resistance against bending and deflection. This design choice helps distribute the load more evenly along the shaft, preventing excessive flexing that could occur with only one bearing. Also, the Lazy Susan enables smooth and controlled movement, allowing the system to precisely direct the flow of water.

3. Function Critical Designs and Calculations

3.1 Motor Calculations

The mass of the rotating disk and framing is 1.3kg. The mass of 2L of water is 2kg. Using these numbers, we can solve for our moment of inertia:

$$I = 0.052161186\ kgm^2 \text{ (From CAD)}$$

Our goal is to travel $\frac{\pi}{2}$ radians in 2 seconds. To achieve this, we solved equation 1, setting t=2s.

$$\frac{\pi}{2} = \alpha * t^2 \tag{1}$$

The angular acceleration needed is: $\alpha_{goal} = 0.393 rad/s^2$.

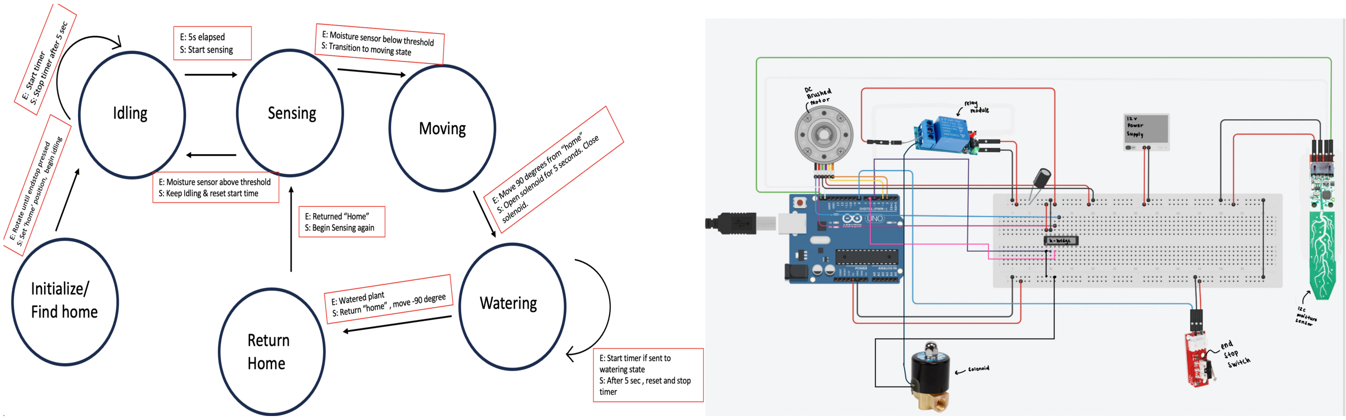
$$\tau = I\alpha \tag{2}$$

Using equation 2 and $\tau_{stall} = 0.18kgm$ of the motor included in our kit, we then determined our maximum angular acceleration, $\alpha_{stall} = 3.45rad/s^2$. Following the rule of thumb of 60%, we set $\alpha_{max} = 2rad/s^2$. This is well above our goal angular acceleration: $\alpha_{max} = 2rad/s^2 > \alpha_{goal} = 0.393rad/s^2$.

3.2 Bearings and Coupling

Our flanged ball bearing has radial load capacities of 330 lbs dynamic and 120 lbs static with a max rated speed of 45000rpm. This is well above any load we will experience. The 4" lazy susan bearing has a load capacity of 300 lbs, which is also above any load we will experience. Since we have a direct transmission, a flexible coupling was chosen to compensate for any misalignment.

4. State Diagram and Circuit Diagram



(a) General picture with the main elements of the system

(b) Elements of the transmission system

Figure 2: Integrated Physical Device

5. Reflection

Fabricating an idea on paper to something tangible and real is not an easy task. One of the most critical strategies for a successful project is good communication. You must also keep in mind that you will iterate your design multiple times and most of the issues with your design will only be seen once fabricated, so start early. Fabrication can be time-consuming and costly depending on the manufacturing method, so be diligent and do it in a timely manner. Take into account things like tolerances, fits, sizing, strength and stability of design, wiring etc. Lastly, when it comes to the code, make sure you allow ample time for debugging. For our team in particular the code was the most difficult part.

6. Appendix

Appendix A: Bill of Materials

Table 1: Motor Drivetrain Components

Component Name	Vendor + Website	Quantity	Cost	Notes
12V 40 RPM Motor w encoder	DFRobot	1	$0.00	Already have
6mm to 8mm Shaft Aluminum Coupling	Amazon	1	$8.49	
1/4"D, D-Profile Rotary Shaft, 4"L	McMaster	1	$0.00	Already have
4" lazy susan turntable	Amazon	1	$4.99	
M3 3mm (L) philips screw (motor mounting)	McMaster	6	$0.00	Already have
M4 16mm (L) Philips Screw	McMaster	16	$0.00	Already have
M4 Screw nut	McMaster	8	$0.00	Already have
1/4" (D inner) 5/8" (D outer) Flanged Bearing	McMaster	1	$6.42	
1/4" (D inner) Shim	McMaster	1	$13.94	
8mm (D) Shaft Collar	Ruland	1	$0.00	Already have
Belleville Disc Spring for 1/4" Shaft Diameter	McMaster	2	$2.33	
Mechanical Limit Switch	Amazon	1	$6.00	
8” Flange Mount	Amazon / CAD	1	$0.00	Already have
Laser Cut Housing	Jacobs Machine Shop	24	$11.00	
D:5mm, 5cm (L) Standoff	Tom	4	$0.00	
		Total Cost:	$53.17	

Table 2: Irrigation Components

Component Name	Vendor + Website	Quantity	Cost	Notes
12V DC Solenoid Valve 1/4"	Amazon	1	$15.69	
Arduino Uno	Arduino	1	$0.00	Already have
12V DC Power Supply	Tom	1	$0.00	Tom Lent
1/4” id to 1/4” mip plastic hose adapter	Amazon	2	$6.99	Already have
12V Relay Module	Zero	1	$0.00	Tom Lent
10FT 1/4" Vinyl tubing	Amazon	1	$7.90	
Bottle Cap Adapter 1/4" tubing	Amazon	1	$9.54	
1.25L Plastic Bottle	Safeway	1	$1.00	
		Total Cost:	$41.12	

Table 3: Soil Sensor Components

Component Name	**Vendor + Website**	**Quantity**	**Cost**	**Notes**
Soil Sensor	Adafruit	4	$62.09	
Cable - Male	found in kit	4	-	
Cable - Female	found in kit	4	-	
Stemma Cable	found in kit	4	-	
Jumper Wires / Misc Wires	Tom	Many	$0.00	Already have (kit)
DRV8833 Dual Motor Driver Carrier	Pololu	1	$0.00	Already have (kit)
100 µF Capacitor	Tom	1	$0.00	Already have (kit)
Breadboard	Tom	1	$0.00	Already have (kit)
		Total Cost:	$62.09	

Table 4: Total Project Cost

Description	**Cost**
Motor Drivetrain	$53.17
Irrigation	$41.12
Soil Sensor	$62.09
Total Project Cost	$156.38

Appendix B: CAD

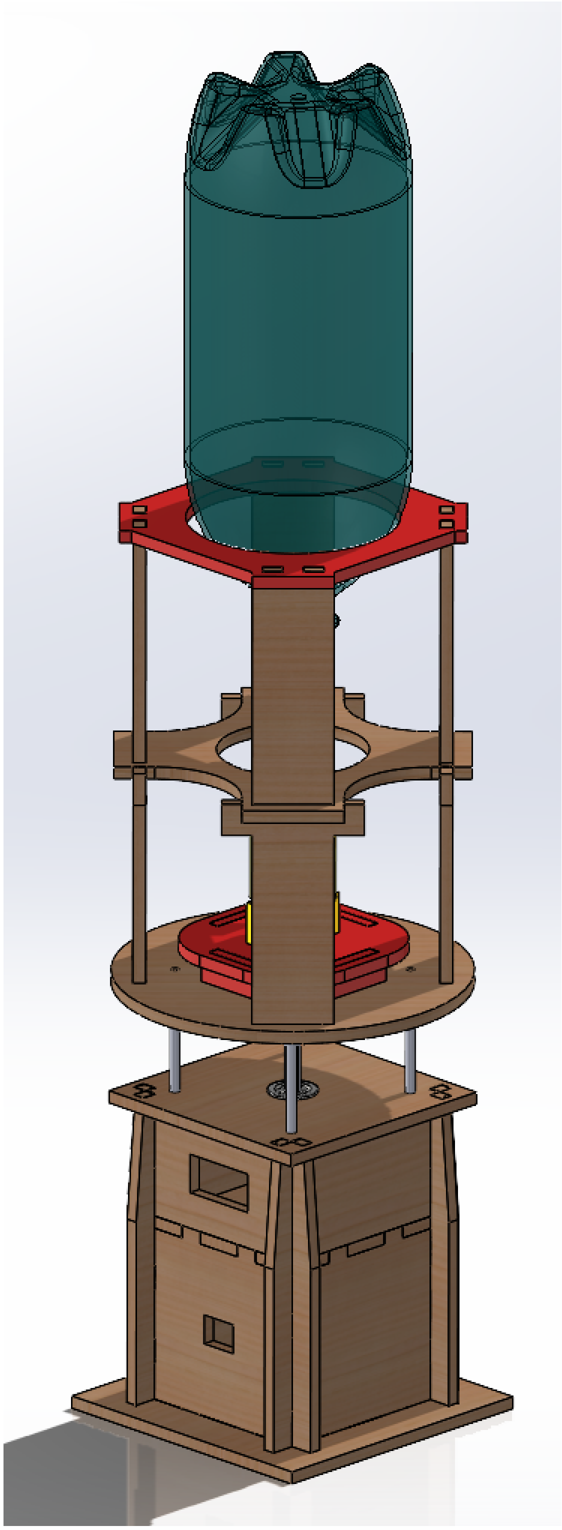


Figure 3: Isometric View of the system structure

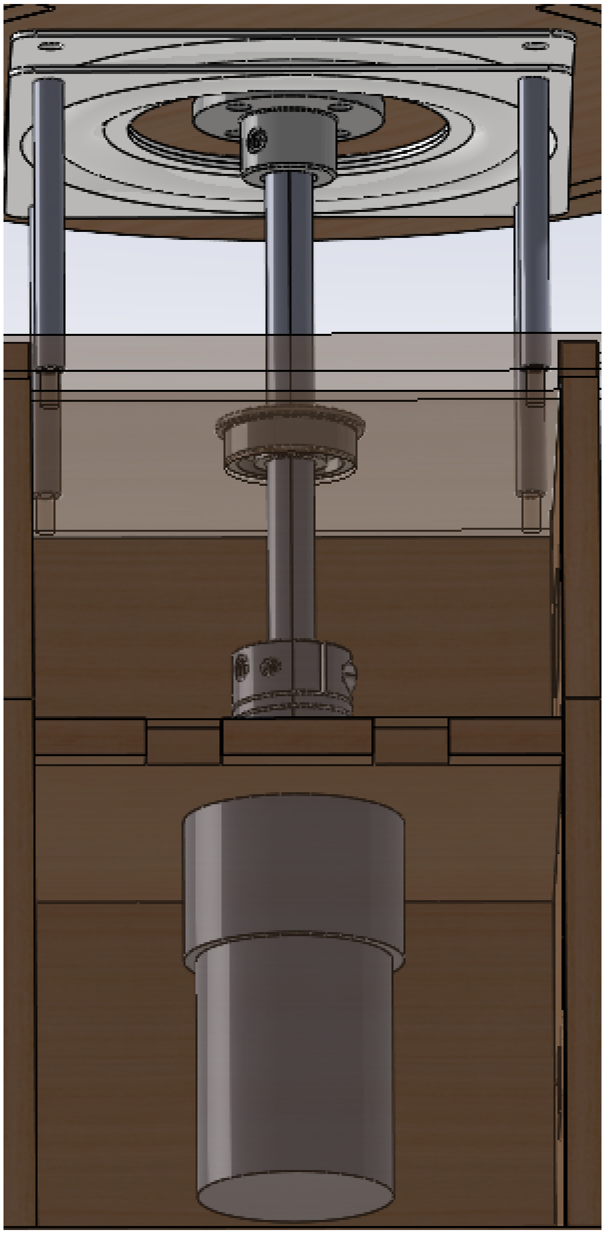


Figure 4: Isometric View of the transmission system

Appendix C: Code

```
#include <Wire.h>
#include "Adafruit_seesaw.h"

#define relayPin 4 // Relay control pin

//Define pin numbers for motor
#define DIR1 6
#define PWM1 5

//Define pin numbmers for encoder
#define encoderPinA 2
#define encoderPinB 3
#define endstop 46
https://www.overleaf.com/project/6566ea9dce4bd0de292a028a

Adafruit_seesaw moistureSensor1;
uint16_t moisture1;

const int relayEnable = 2;
const int thresholdMax = 800;
const unsigned long solenoidDuration = 5000; // Solenoid open duration in
    milliseconds
const unsigned long waitDuration = 30000;     // Wait duration in milliseconds

//Variable for encoder counts
volatile long encoderCount = 0;
//Variables for PID Control
long previousTime = 0;
float ePrevious = 0;
float eIntegral = 0;
int endstopvalue = 0;
int direction = 0;
int direction2 = 0 ;
int sendHome = 0;
int reverse = 0;
int goPlant = 0;
int out =0;
int home = 0;
void handleEncoder() {
     if (digitalRead(encoderPinA) > digitalRead(encoderPinB)){
    encoderCount++;
     }
    else{
    encoderCount--;
    }
}
enum State {
  IDLE = 0,
  SENSING,
  MOVING,
  WATERING,
  RETURNHOME,
};

State currentState = IDLE;
unsigned long startTime = 0;
unsigned long aboveThresholdStartTime1 = 0;
unsigned long solenoidStartTime = 0; // Track the solenoid start time
const unsigned long sensingDuration = 5000; // 5 seconds in milliseconds

```

```
void setup() {
  Serial.begin(115200);
  pinMode (DIR1, OUTPUT);
  pinMode (PWM1, OUTPUT);
  pinMode (encoderPinA, INPUT);
  pinMode (encoderPinB, INPUT);
  pinMode (endstop, INPUT);
  pinMode(relayPin, OUTPUT); // Initialize relay pin as OUTPUT
  //Endstop
  //Interrupt for encoder
  attachInterrupt(digitalPinToInterrupt (encoderPinA), handleEncoder, RISING);

  if (!moistureSensor1.begin(0x36)) {
    Serial.println("ERROR! Moisture sensor 1 not found");
    while (1)
      ;
  }
}

void moveMotor(int dirPin, int pwmPin, float u, int direction){
  if (direction == 1){
    direction2 = 0;
    float speed = 45;
    digitalWrite(dirPin, direction2);
    analogWrite (pwmPin, speed);
  }

  if (direction == 0){
    direction2 = -1;
    float speed = -45;
    digitalWrite(dirPin, direction2);
    analogWrite (pwmPin, speed);
    digitalWrite(0, 0);
    analogWrite (0, 0);
  }

  return 0;
}
void moveBackHome() {
  const int targetAngle = -90; // Target angle in degrees (negative for moving in the opposite direction)
  const float countsPerRevolution = 360; // Number of encoder counts in one revolution
  const int encoderCounts = abs(targetAngle) / 360.0 * countsPerRevolution; // Calculate counts for -90 degrees

  float kp = 2; // Update these values based on your tuning requirements
  float kd = 0.1;
  float ki = 0.01;
  int direction = 0; // Assuming negative direction for a -90-degree rotation

  long initialEncoderCount = encoderCount; // Record the initial encoder count
  long targetEncoderCount = initialEncoderCount - encoderCounts; // Calculate target encoder count

  while (encoderCount > targetEncoderCount) {
    int error = encoderCount - targetEncoderCount;
    float u = pidController(targetEncoderCount, kp, kd, ki); // Compute PID control signal

    // Adjust motor movement based on the PID output
    moveMotor(DIR1, PWM1, u, direction);
```

```
  }

  // Stop the motor after reaching the desired angle
  moveMotor(DIR1, PWM1, 0, 0);
}

float pidController(int target, float kp, float kd, float ki) {
  //Measure the time elapsed since the last iteration
  long currentTime = micros();
  float deltaT = ((float)(currentTime - previousTime)) / 1.0e6;

  //Compute the error, derivative, and integral
  int e = encoderCount - target;
  float eDerivative = (e - ePrevious) / deltaT;
  eIntegral = eIntegral + e * deltaT;

  //Compute the PID control signal
  float u = (kp * e) + (kd * eDerivative) + (ki * eIntegral);

  //Update variables for the next iteration
  previousTime = currentTime;
  ePrevious = e;

  return u;
}

// Function to move the motor approximately 90 degrees
void move90Degrees() {
  const int targetAngle = 90; // Target angle in degrees
  const float countsPerRevolution = 360; // Number of encoder counts in one
   revolution
  const int encoderCounts = targetAngle / 360.0 * countsPerRevolution; //
   Calculate counts for 90 degrees

  float kp = 2; // Update these values based on your tuning requirements
  float kd = 0.1;
  float ki = 0.01;
  int direction = 1; // Assuming positive direction for a 90-degree rotation

  long initialEncoderCount = encoderCount; // Record the initial encoder count
  long targetEncoderCount = initialEncoderCount + encoderCounts; // Calculate
   target encoder count

  while (encoderCount < targetEncoderCount) {
    int error = targetEncoderCount - encoderCount;
    float u = pidController(targetEncoderCount, kp, kd, ki); // Compute PID
   control signal

    // Adjust motor movement based on the PID output
    moveMotor(DIR1, PWM1, u, direction);
  }

  // Stop the motor after reaching the desired angle
  moveMotor(DIR1, PWM1, 0, 0);
}

void openSolenoid() {
  Serial.println("Opening solenoid");
  digitalWrite(relayPin, LOW); // Activate relay to open the solenoid
  solenoidStartTime = millis(); // Record the start time of the solenoid
   operation
}
```

```
void closeSolenoid() {
  digitalWrite(relayPin, HIGH); // Deactivate relay to close the solenoid
}

void loop() {

  endstopvalue = digitalRead(endstop);
  //let motor run until endstopper is hit
    while (sendHome == 0){
    direction2 = 0;
    float speed = 45;
    digitalWrite(DIR1, direction2);
    analogWrite (PWM1, speed);
    endstopvalue = digitalRead(endstop);
  switch (currentState) {
    case IDLE:
      Serial.println("IDLE: doing nothing");
      moisture1 = moistureSensor1.touchRead(0);
      Serial.print("Moisture 1 Humidity:");
      Serial.println(moisture1);

      if (millis() - startTime >= sensingDuration) {
        currentState = SENSING;
        startTime = millis(); // Reset the start time for the next state
        Serial.println("Transitioning to SENSING state");
        aboveThresholdStartTime1 = 0; // Reset the timer for above-threshold duration
      }
      break;

    case SENSING:
      Serial.println("SENSING Moisture 1 ");
      moisture1 = moistureSensor1.touchRead(0);
      Serial.print("Moisture 1 Humidity:");
      Serial.println(moisture1);

      if (moisture1 < thresholdMax) {
        currentState = MOVING;
        Serial.println("Transitioning to MOVING state");
      } else {
        aboveThresholdStartTime1 = 0; // Reset the timer if humidity is above threshold
        currentState = IDLE;
        startTime = millis(); // Reset the start time for the next IDLE state
        Serial.println("Transitioning back to IDLE");
      }
      break;

    case MOVING:
      Serial.println("MOVING");
      move90Degrees(); // Call the function to move the motor approximately 90 degrees
      currentState = WATERING; // Move to the WATERING state after motor movement
      break;

    case WATERING:
      Serial.println("WATERING");
      if (millis() - solenoidStartTime < solenoidDuration) {
      Serial.println("Opening solenoid");
      openSolenoid(); // Open the solenoid
```

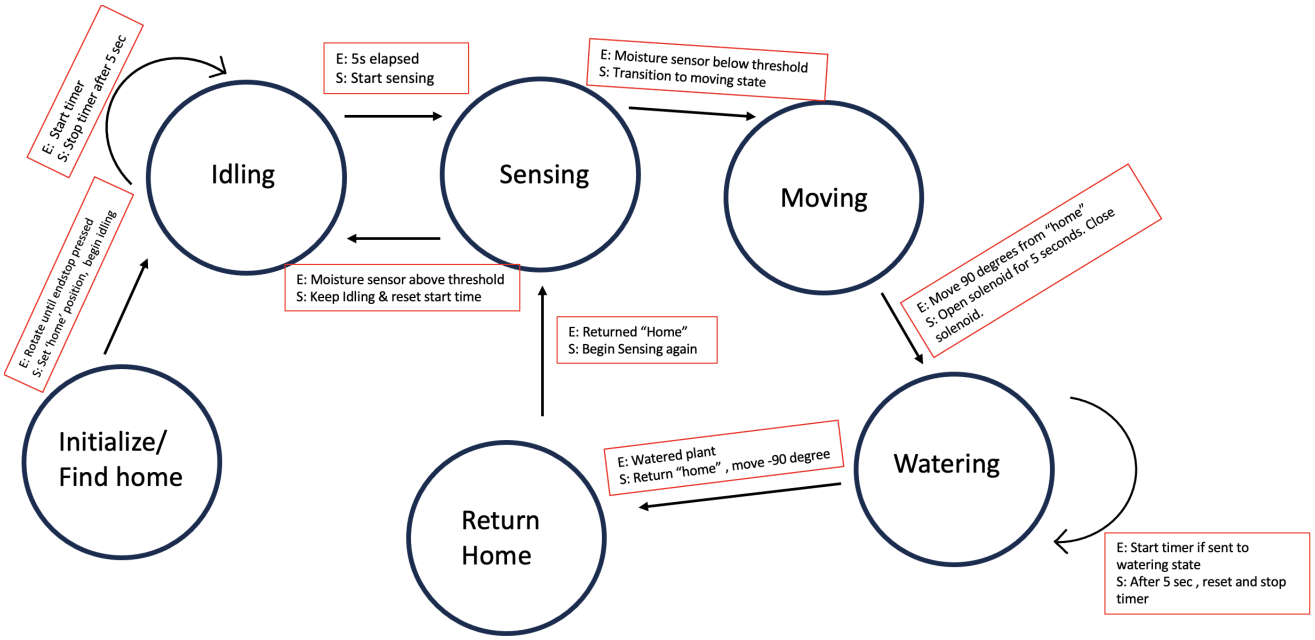
```
      } else {
    if (moisture1 < thresholdMax) {
      closeSolenoid(); // Close the solenoid after the set duration only if
   moisture is below threshold
      currentState = RETURNHOME;
      Serial.println("Plant watered");
      currentState = RETURNHOME;
  }
  }
      break;

    case RETURNHOME:
      Serial.println("RETURNING HOME");
      moveBackHome(); // Call the function to move back home or approximately
   -90 degrees
      currentState = SENSING; // Move to the SENSING state after returning
   home
      startTime = millis(); // Reset the start time for the next IDLE state
      Serial.println("Returning to IDLE state");
      break;
  delay(2000);
  }
}
}
```

Listing 1: Blink.ino

Appendix D: Extra Images

1. State Diagram



2. Circuit Diagram

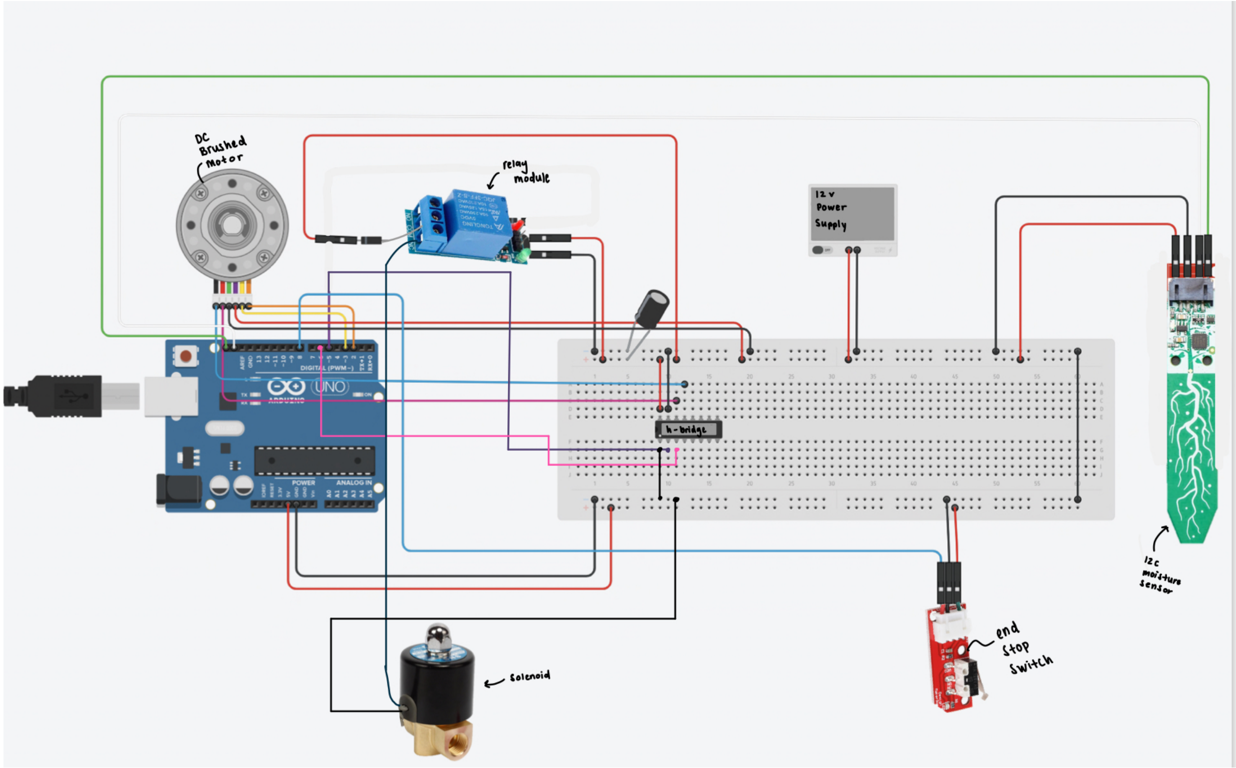


Figure 5: Circuit Diagram

3. Flow Calculations Graphs

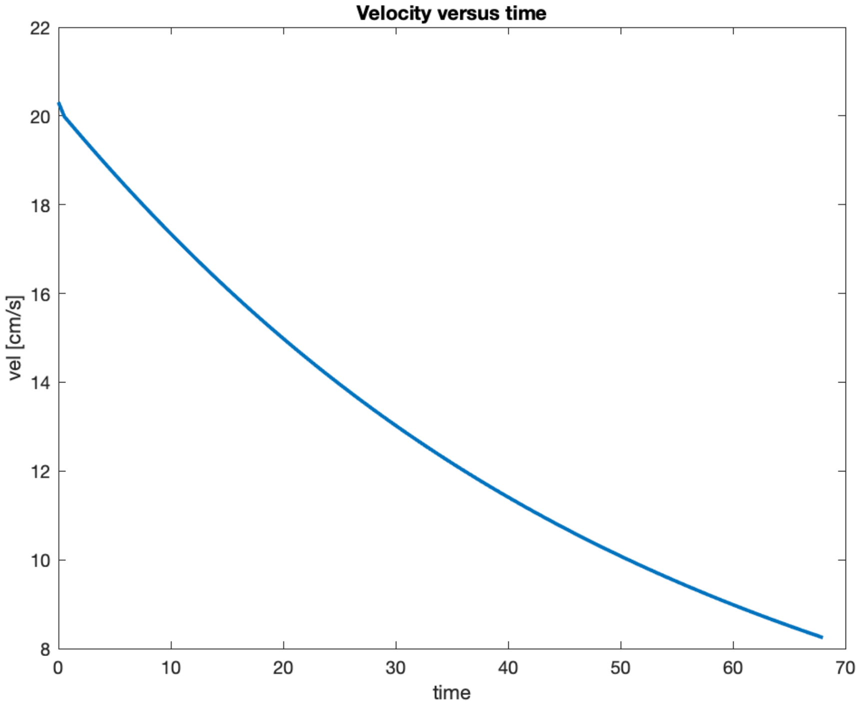


Figure 6: Velocity vs Time

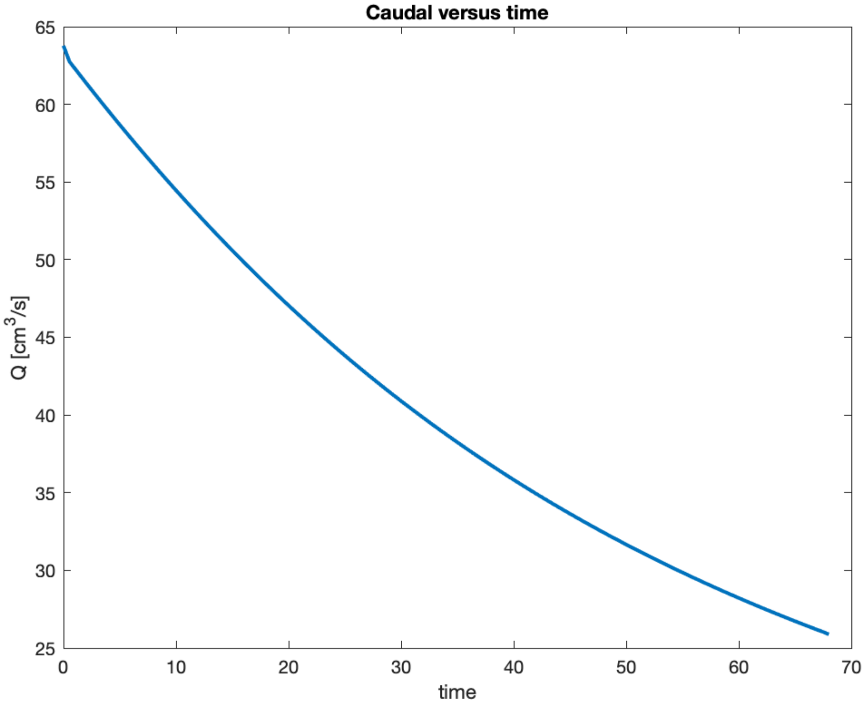


Figure 7: Caudal vs Time