2023 KAIST 메이커톤

자동관리 모듈형 스마트팜

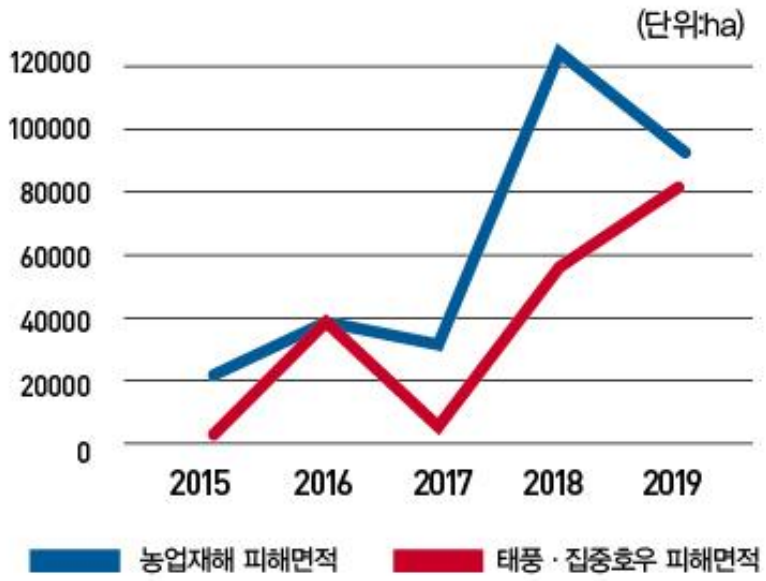
경민호

기계공학과 이민호
기계공학과 남경민

목차

1. 문제 정의

연도별 이상기후로 인한 농업 재해 피해 현황

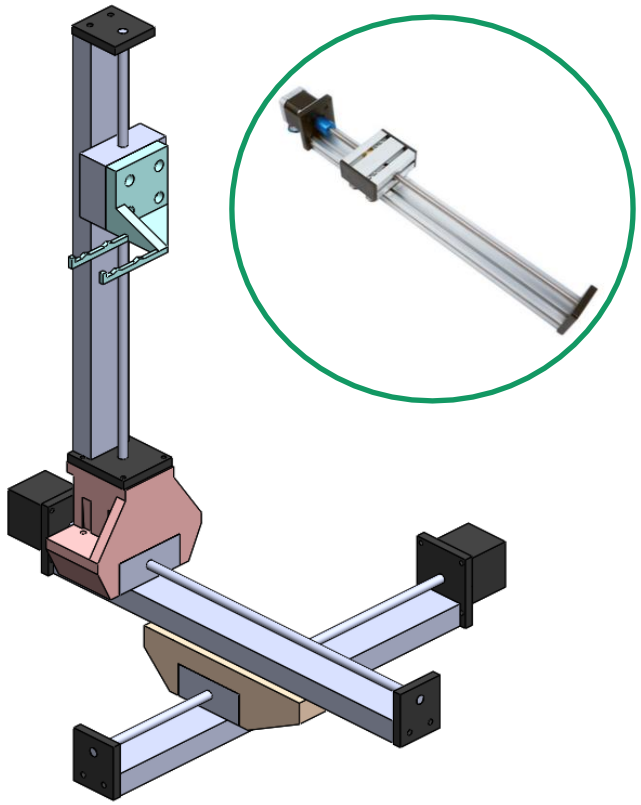


스마트팜

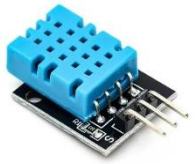
- 공간 활용도
- 생산성
- 식품 접근성

2. 기능 정의

기능 1 자동 재배



기능 2 센서를 이용한 환경 모니터링 및 유지



온습도

조도

토양 수분

바람

기능 3 사용자 인터페이스

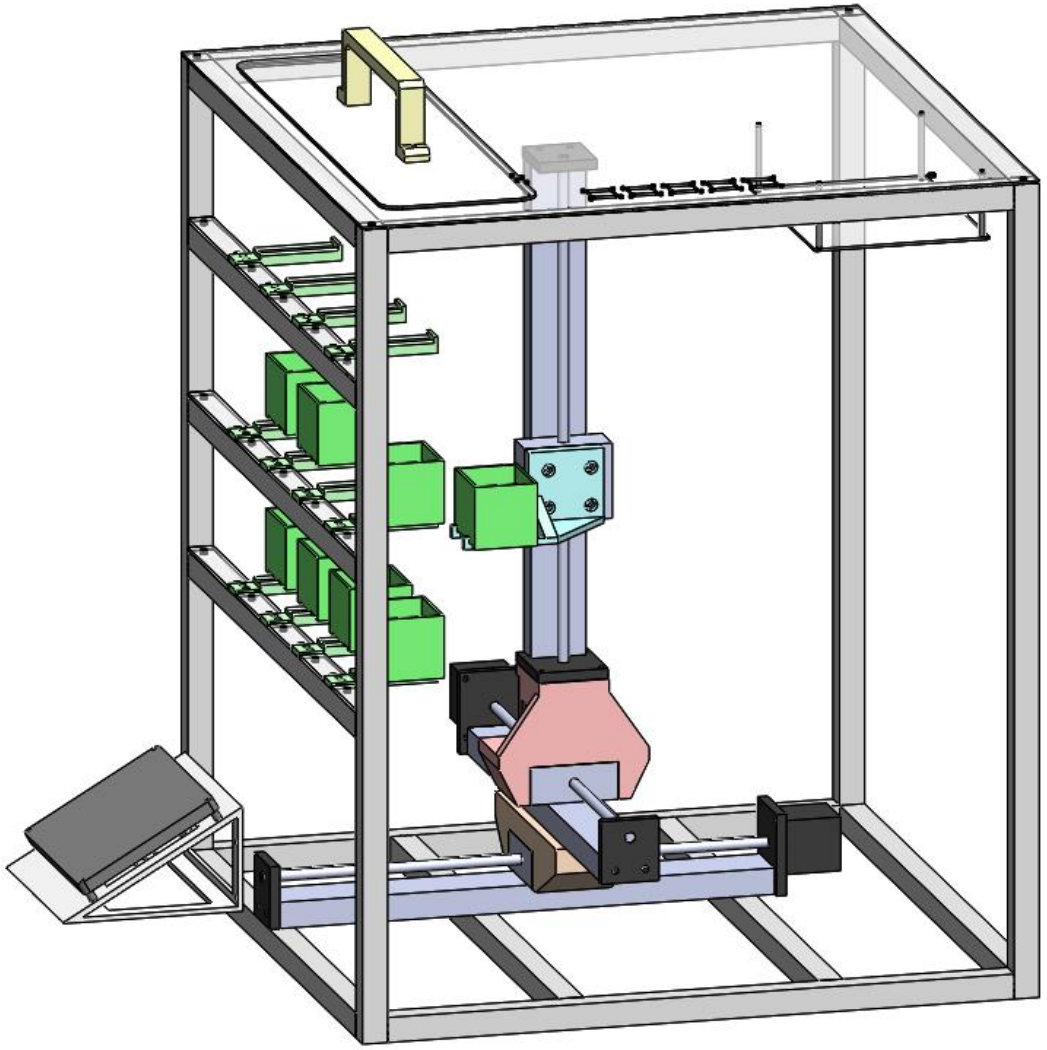
3. 설계

프레임

- 프로파일
- 아크릴
- 화분 거치대
- 화분

구동부

- LCD
- LCD 거치대

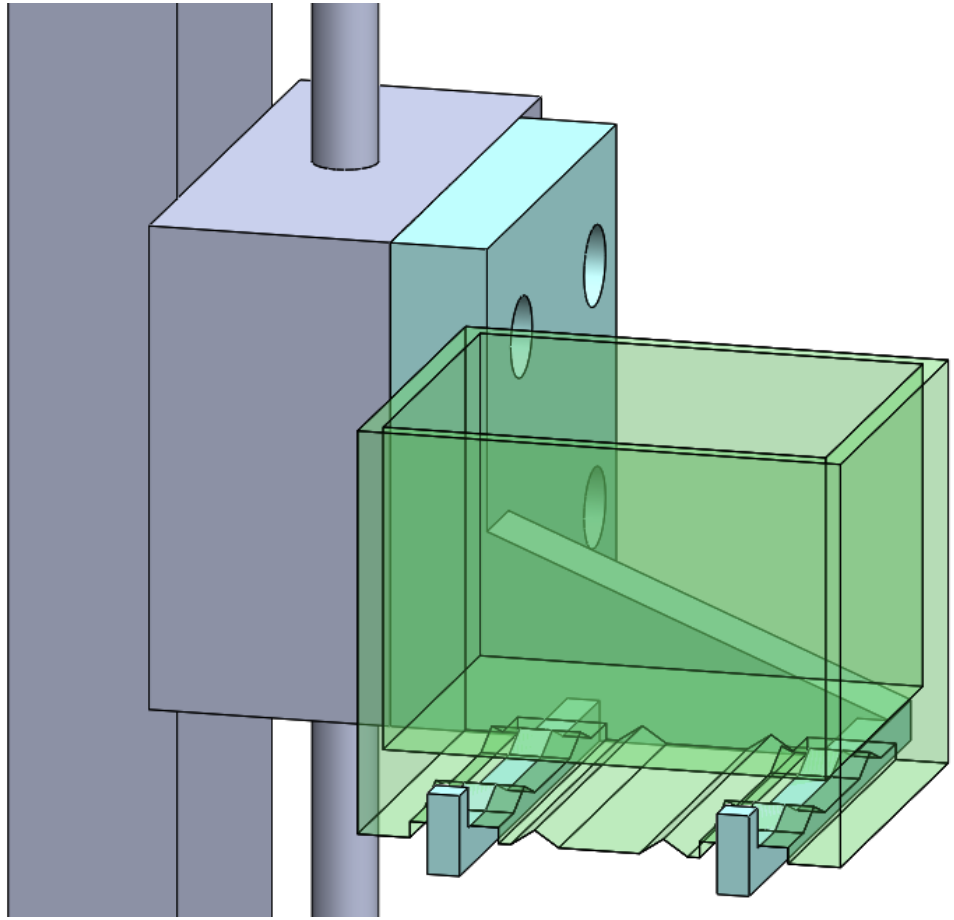


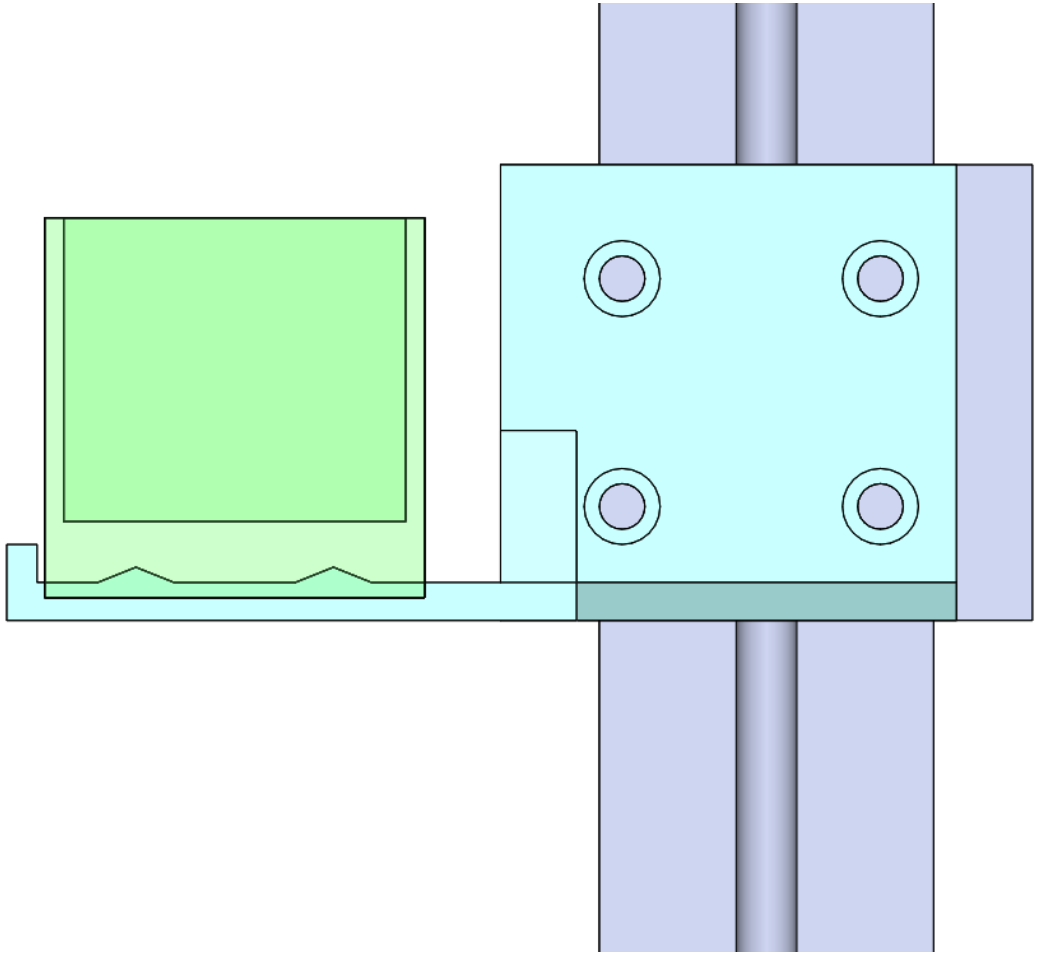
구동부

- 모터 연결부 1
- 모터 연결부 2
- 그리퍼

3. 설계

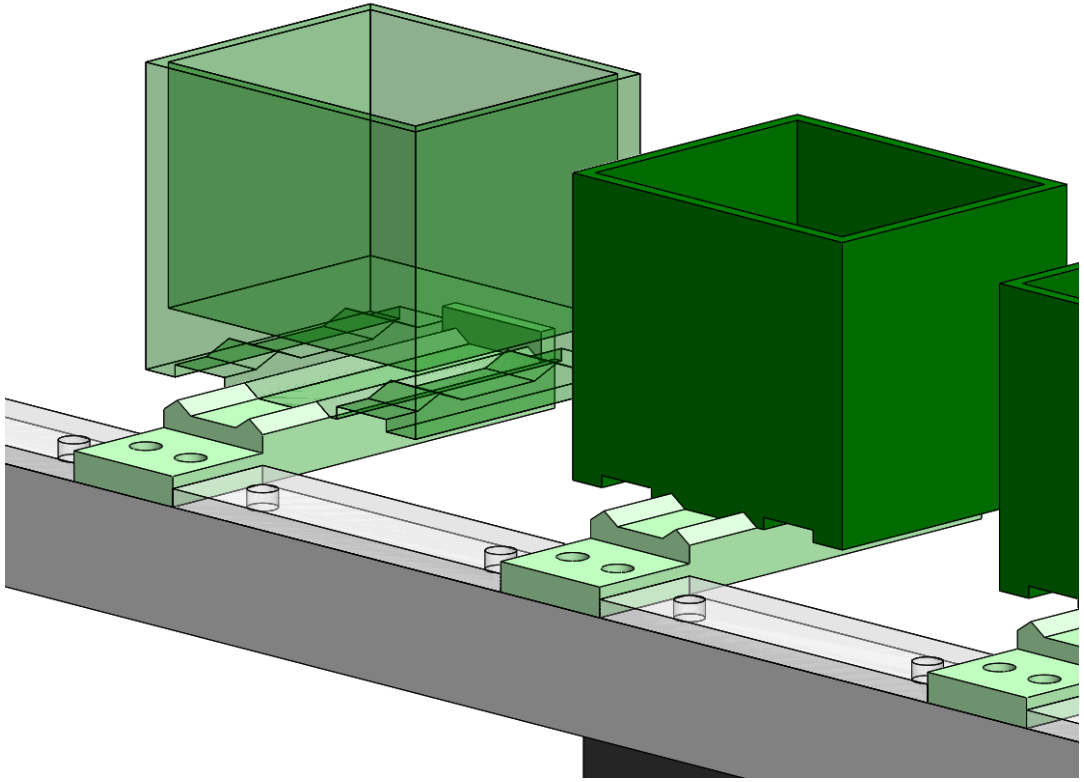
화분 - 그리퍼 연결 부분

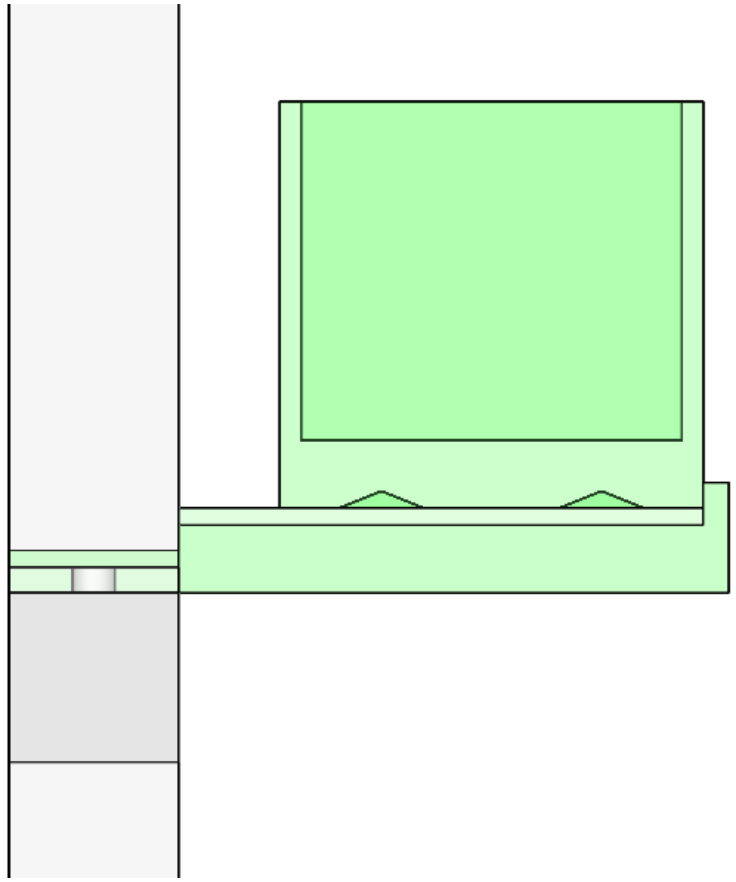




3. 설계

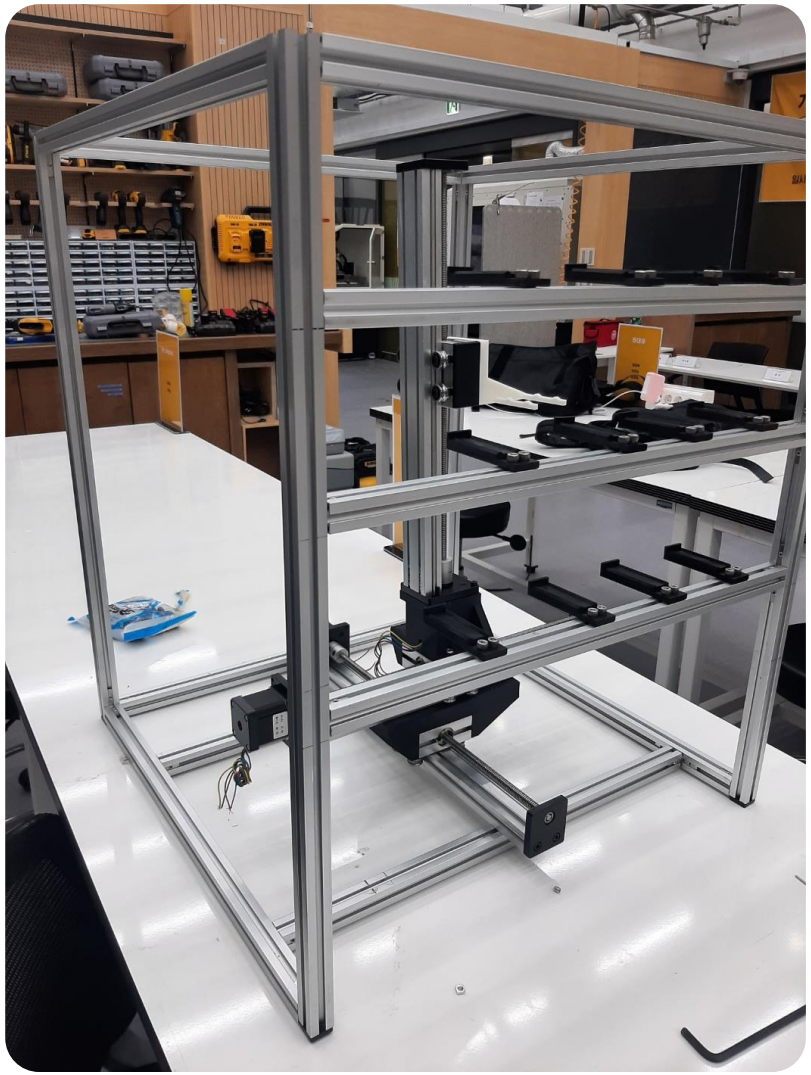
화분 거치대 - 화분 연결 부분

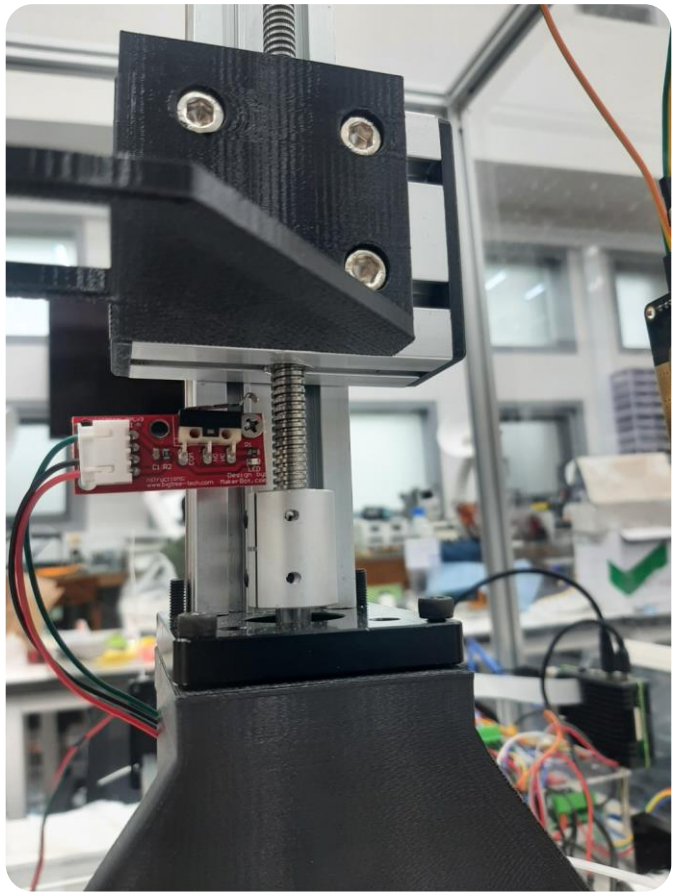




4. HW 제작

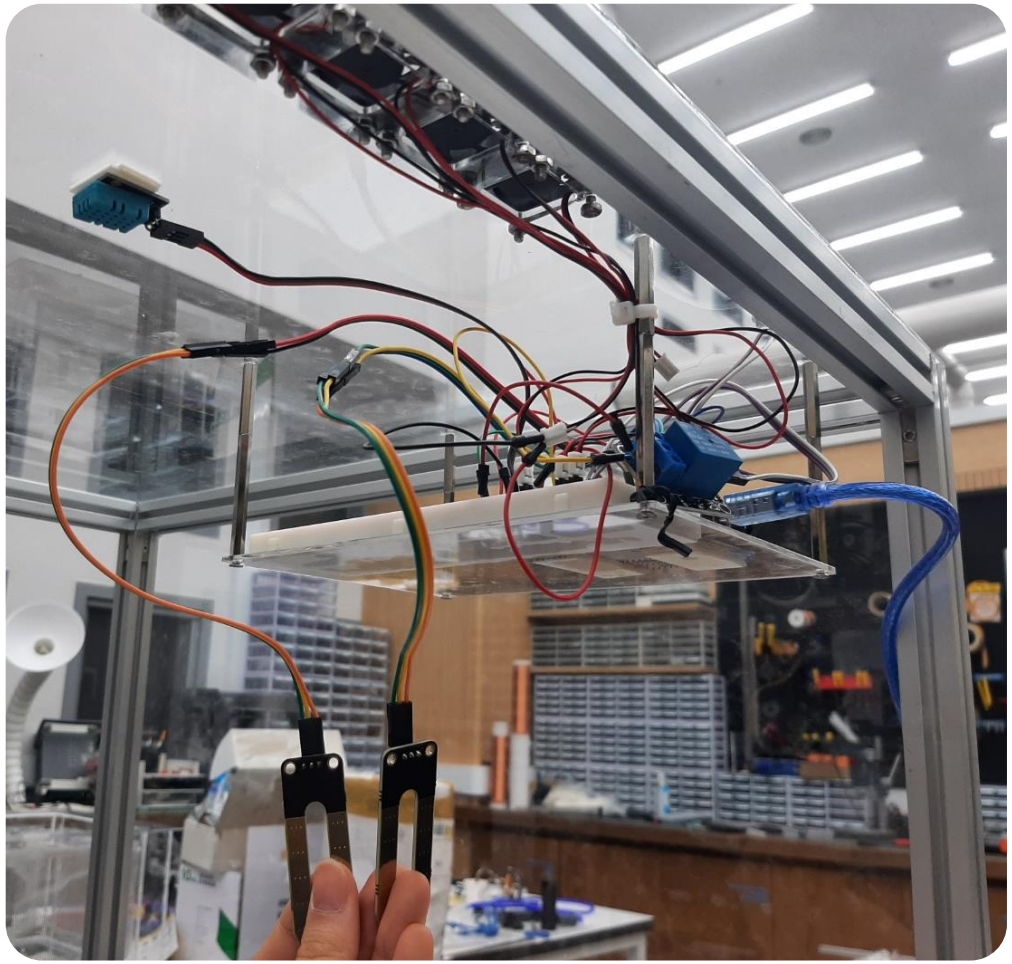
1. 프레임 및 구동부 제작



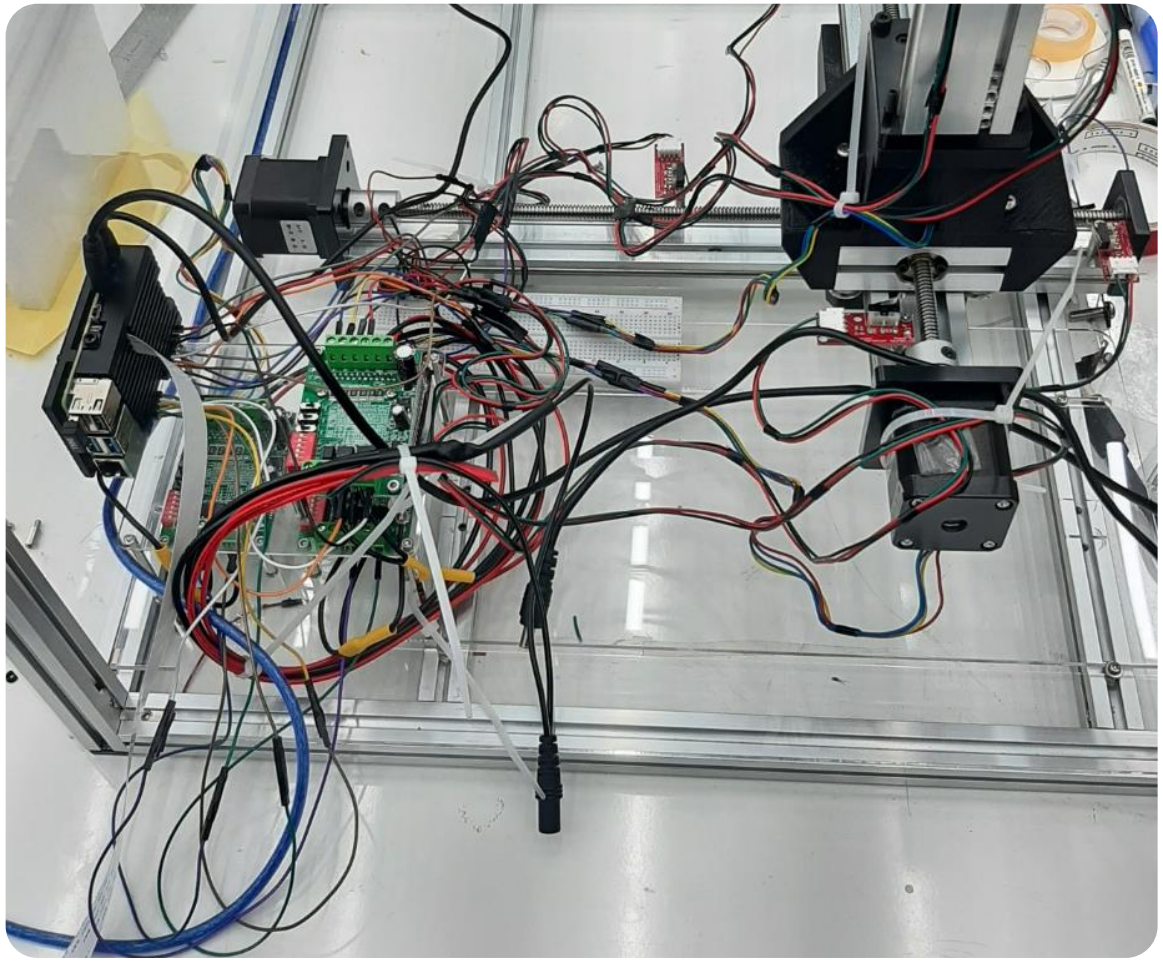


4. HW 제작

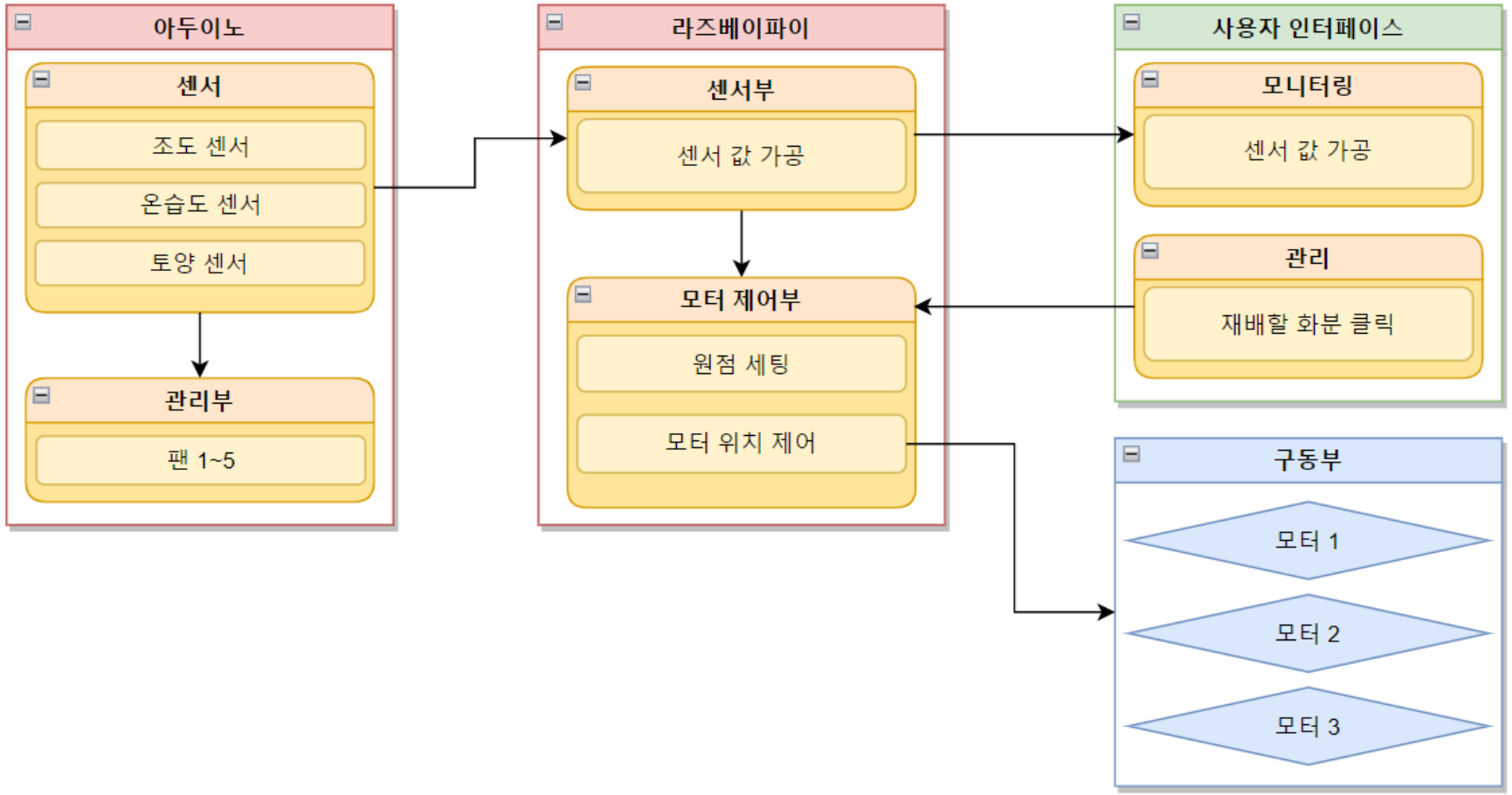
2. 아두이노 회로 제작 및 결합



3. 라즈베리 파이 회로 제작 및 결합

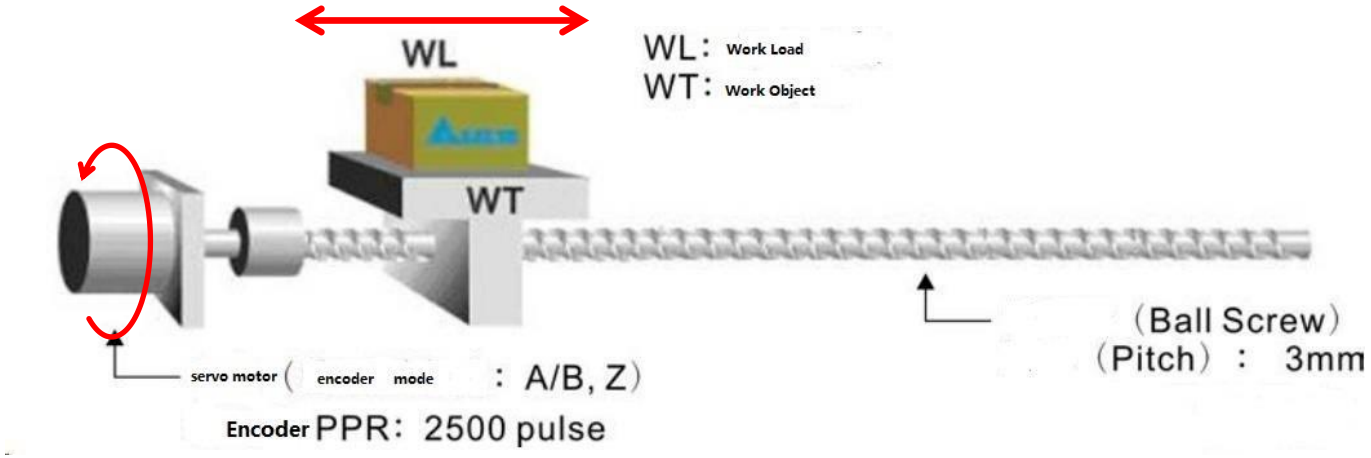


5. SW 제작


아두이노
센서
조도 센서
온습도 센서
토양 센서
관리부
팬 1~5
라즈베이파이
센서부
센서 값 가공
모터 제어부
원점 세팅
모터 위치 제어
사용자 인터페이스
모니터링
센서 값 가공
관리
재배할 화분 클릭
구동부
모터 1
모터 2
모터 3

5. SW 제작 - 모터 제어부

회전수 기반 위치 제어



5. SW 제작 - 모터 제어부

코딩

```python
import RPi.GPIO as GPIO
import time
from enum import Enum
import numpy as np
import signal
import sys

MOTOR_X_CW_PIN=13
MOTOR_X_CLK_PIN=6
MOTOR_Y_CW_PIN=26
MOTOR_Y_CLK_PIN=19
MOTOR_Z_CW_PIN=21
MOTOR_Z_CLK_PIN=20

END_SWITCH_X1=22
END_SWITCH_X2=23
END_SWITCH_Y1=24
END_SWITCH_Y2=25
END_SWITCH_Z1=27
END_SWITCH_Z2=17

X_LEN=24786
Y_LEN=14082
Z_LEN=26762

X_UNIT=9600
Z_UNIT=10200

X_OFFSET=4200
Y_OFFSET=6500
Z_OFFSET=3000

Y_IN_DIST=6400
Y_OUT_DIST=-7000
Z_UP_DIST=3500

OUT=GPIO.OUT
IN=GPIO.IN

ROTATION_T=0.001
STOP_T=0.001
ORIGIN=4000
GRID=(3,4)

class Motor(Enum):
    X = 0
    Y = 1
    Z = 2
class Dir(Enum):
    CW = True #GPIO.HIGH
    CCW = False #GPIO.LOW

class SmartFarmControl():
    def __init__(self):
        self.xpos=0
        self.ypos=0
        self.zpos=0
        self.xdir=True
        self.ydir=True
        self.zdir=True

        self.xlen=X_LEN
        self.ylen=Y_LEN
        self.zlen=Z_LEN
        self.emergencyStop=False

        self.modes=["initialization" for i in range(3)]
        self.setPinMode()


        # self.initializing_end_to_end()

        """
        An initialization function

        Parameters
        ----------
        origin: origin of passenger

        dest: destination of passenger

        state: one of WAIT, ONBOARD, ARRIVAL
            4200
        """
    def setPinMode(self):
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(MOTOR_X_CW_PIN,OUT)
        # GPIO.setmode(GPIO.BCM)
        GPIO.setup(MOTOR_X_CLK_PIN,OUT)

        # GPIO.setmode(GPIO.BCM)
        GPIO.setup(MOTOR_Y_CW_PIN,OUT)
        # GPIO.setmode(GPIO.BCM)
        GPIO.setup(MOTOR_Y_CLK_PIN,OUT)

        # GPIO.setmode(GPIO.BCM)
        GPIO.setup(MOTOR_Z_CW_PIN,OUT)
        # GPIO.setmode(GPIO.BCM)
        GPIO.setup(MOTOR_Z_CLK_PIN,OUT)

        GPIO.setup(END_SWITCH_X1, IN, pull_up_down = GPIO.PUD_UP)
        GPIO.setup(END_SWITCH_X2, IN, pull_up_down = GPIO.PUD_UP)
        GPIO.add_event_detect(END_SWITCH_X1, GPIO.FALLING, callback=self.switchX1Pressed,bouncetime=300)
        GPIO.add_event_detect(END_SWITCH_X2, GPIO.FALLING, callback=self.switchX2Pressed,bouncetime=300)

        GPIO.setup(END_SWITCH_Y1, IN, pull_up_down = GPIO.PUD_UP)
        GPIO.setup(END_SWITCH_Y2, IN, pull_up_down = GPIO.PUD_UP)
        GPIO.add_event_detect(END_SWITCH_Y1, GPIO.FALLING, callback=self.switchY1Pressed,bouncetime=300)
        GPIO.add_event_detect(END_SWITCH_Y2, GPIO.FALLING, callback=self.switchY2Pressed,bouncetime=300)

        GPIO.setup(END_SWITCH_Z1, IN, pull_up_down = GPIO.PUD_UP)
        GPIO.setup(END_SWITCH_Z2, IN, pull_up_down = GPIO.PUD_UP)
        GPIO.add_event_detect(END_SWITCH_Z1, GPIO.FALLING, callback=self.switchZ1Pressed,bouncetime=300)
        GPIO.add_event_detect(END_SWITCH_Z2, GPIO.FALLING, callback=self.switchZ2Pressed,bouncetime=300)
    def getMotorNum(self,motor):
        if motor==Motor.X:return 0
        if motor==Motor.Y:return 1
        if motor==Motor.Z:return 2
    def checkMode(self):
        for mode in self.modes:
            if mode=="initialization":
                return False
        return True
    def checkMotorMode(self,motor):
        num=self.getMotorNum(motor)
        if self.modes[num]=="initialization":
            return False
        else:
            GPIO.output(MOTOR_X_CW_PIN,dir)
        elif motor==Motor.Y:
            self.ydir=dir
            GPIO.output(MOTOR_Y_CW_PIN,not dir)
        elif motor==Motor.Z:
            self.zdir=dir
            GPIO.output(MOTOR_Z_CW_PIN,not dir)
    def setMotorsDir(self,distances):
        if distances[0]>=0:
            self.setMotorRotationDir(Motor.X,True)
        else:
            self.setMotorRotationDir(Motor.X,False)
        if distances[1]>=0:
            self.setMotorRotationDir(Motor.Y,True)
        else:
            self.setMotorRotationDir(Motor.Y,False)
        if distances[2]>=0:
            self.setMotorRotationDir(Motor.Z,True)
        else:
            self.setMotorRotationDir(Motor.Z,False)

    def setMotorsRotationDir(self,motors,dir):
        for motor in motors:
            self.setMotorRotationDir(motor,dir)
    def setDirection(self,dir):
        if dir is True:
            return 1
        else:
            return -1

    def moveMotors(self,motors):
        if motors is None:
            return
        for motor in motors:
            if motor==Motor.X:
                self.xpos+=self.setDirection(self.xdir)
                GPIO.output(MOTOR_X_CLK_PIN,True)
            elif motor==Motor.Y:
                self.ypos+=self.setDirection(self.ydir)
                GPIO.output(MOTOR_Y_CLK_PIN,True)
            elif motor==Motor.Z:
                self.zpos+=self.setDirection(self.zdir)
                GPIO.output(MOTOR_Z_CLK_PIN,True)
        time.sleep(ROTATION_T)
        for motor in motors:
            if motor==Motor.X:
                GPIO.output(MOTOR_X_CLK_PIN,False)
            elif motor==Motor.Y:
                GPIO.output(MOTOR_Y_CLK_PIN,False)
            elif motor==Motor.Z:
                GPIO.output(MOTOR_Z_CLK_PIN,False)
        time.sleep(STOP_T)

    def movableMotors(self,distances):
        motors=[]
        if distances[0]!=0:
            motors.append(Motor.X)
        if distances[1]!=0:
            motors.append(Motor.Y)
        if distances[2]!=0:
            motors.append(Motor.Z)
        return motors

    def updateDistance(self,distances):
        return distances
    def moveMotorsDistance(self,distances):
        distances=np.array(distances)
        distances=distances.astype(int)
        # print(distances)
        if distances[0]>=0:
            if distances[0]+self.xpos>=self.xlen:
                distances[0]=self.xlen-self.xpos-1
        else:
            if distances[0]+self.xpos<=0:
                distances[0]=-self.xpos+1
        if distances[1]>=0:
            if distances[1]+self.ypos>=self.ylen:
                distances[1]=self.ylen-self.ypos-1
        else:
            if distances[1]+self.ypos<=0:
                distances[1]=-self.ypos+1
        if distances[2]>=0:
            if distances[2]+self.zpos>=self.zlen:
                distances[2]=self.zlen-self.zpos-1
        else:
            if distances[2]+self.zpos<=0:
                distances[2]=-self.zpos+1
        maxdist=max(abs(distances))
        # print(distances)
        self.setMotorsDir(distances)
        for i in range(maxdist):
            # print(self.movableMotors(distances))
            self.moveMotors(self.movableMotors(distances))
            distances=self.updateDistance(distances)

    def moveMotorsToCoords(self,coords):
        distances=self.calculateCoordDistance(coords)
        self.moveMotorsDistance(distances)

    def moveMotorsToOrigin(self):
        # print(self.xpos)
        # print(self.ypos)
        # print(self.zpos)
        distances=[ORIGIN-self.xpos,ORIGIN-self.ypos,ORIGIN-self.zpos]
        # print(distances)
        self.moveMotorsDistance(distances)

    def moveMotorsOrigDest(self,orig,dest):
        self.moveMotorsToCoords(orig)
        time.sleep(0.1)
        self.moveMotorsDistance([0,Y_IN_DIST,0])#Y IN
        time.sleep(0.1)
        self.moveMotorsDistance([0,0,Z_UP_DIST])#Z UP
        time.sleep(0.1)
        self.moveMotorsDistance([0,Y_OUT_DIST,0])#Y IN
        time.sleep(0.1)      #catch

        self.moveMotorsToCoords(dest)
        self.moveMotorsDistance([0,0,Z_UP_DIST])#Z UP
        time.sleep(0.1)
        self.moveMotorsDistance([0,Y_IN_DIST,0])#Y IN
        time.sleep(0.1)
        self.moveMotorsDistance([0,0,-Z_UP_DIST])#Z UP
        self.moveMotorsDistance([0,-Y_IN_DIST,0])#Y IN

        time.sleep(0.1)
        self.moveMotorsToOrigin()
            self.xdir=True
            # self.xlen=self.counter-self.xlen
            # print(self.xlen)
            self.modes[0]="normal"
            self.setMotorRotationDir(Motor.X,True)
        else:
            self.xpos=0
            self.xdir=True
            self.setMotorRotationDir(Motor.X,True)
    def switchX2Pressed(self,channel):
        if self.modes[0]=="initialization":
            print("x2")
            self.xdir=False
            # self.xlen=self.counter
            self.setMotorRotationDir(Motor.X,False)
        else:
            self.xpos=self.xlen
            self.xdir=False
            self.setMotorRotationDir(Motor.X,False)
    def switchY1Pressed(self,channel):
        if self.modes[1]=="initialization":
            print("y1")
            self.counter[1]=0
            self.ypos=0
            self.ydir=True
            # self.ylen=self.counter-self.ylen
            # print(self.ylen)
            self.modes[1]="normal"
            self.setMotorRotationDir(Motor.Y,True)
        else:
            self.ypos=0
            self.ydir=True
            self.setMotorRotationDir(Motor.Y,True)
    def switchY2Pressed(self,channel):
        if self.modes[1]=="initialization":
            print("y2")
            self.ydir=False
            # self.ylen=self.counter
            self.setMotorRotationDir(Motor.Y,False)
            # self.emergencyStop=True
        else:
            self.ypos=self.ylen
            self.ydir=False
            self.setMotorRotationDir(Motor.Y,False)
    def switchZ1Pressed(self,channel):
        if self.modes[2]=="initialization":
            self.counter[2]=0
            print("z1")
            self.zpos=0
            self.zdir=True
            # self.zlen=self.counter-self.zlen
            # print(self.zlen)
            self.modes[2]="normal"
            self.setMotorRotationDir(Motor.Z,True)
        else:
            self.zpos=0
            self.zdir=True
            self.setMotorRotationDir(Motor.Z,True)
    def switchZ2Pressed(self,channel):
        if self.modes[2]=="initialization":
            print("z2")
            self.zdir=False
            # self.zlen=self.counter
            self.setMotorRotationDir(Motor.Z,False)
```

5. SW 제작 - 사용자 인터페이스

Pygame을 이용하여 제작

Python Pygame

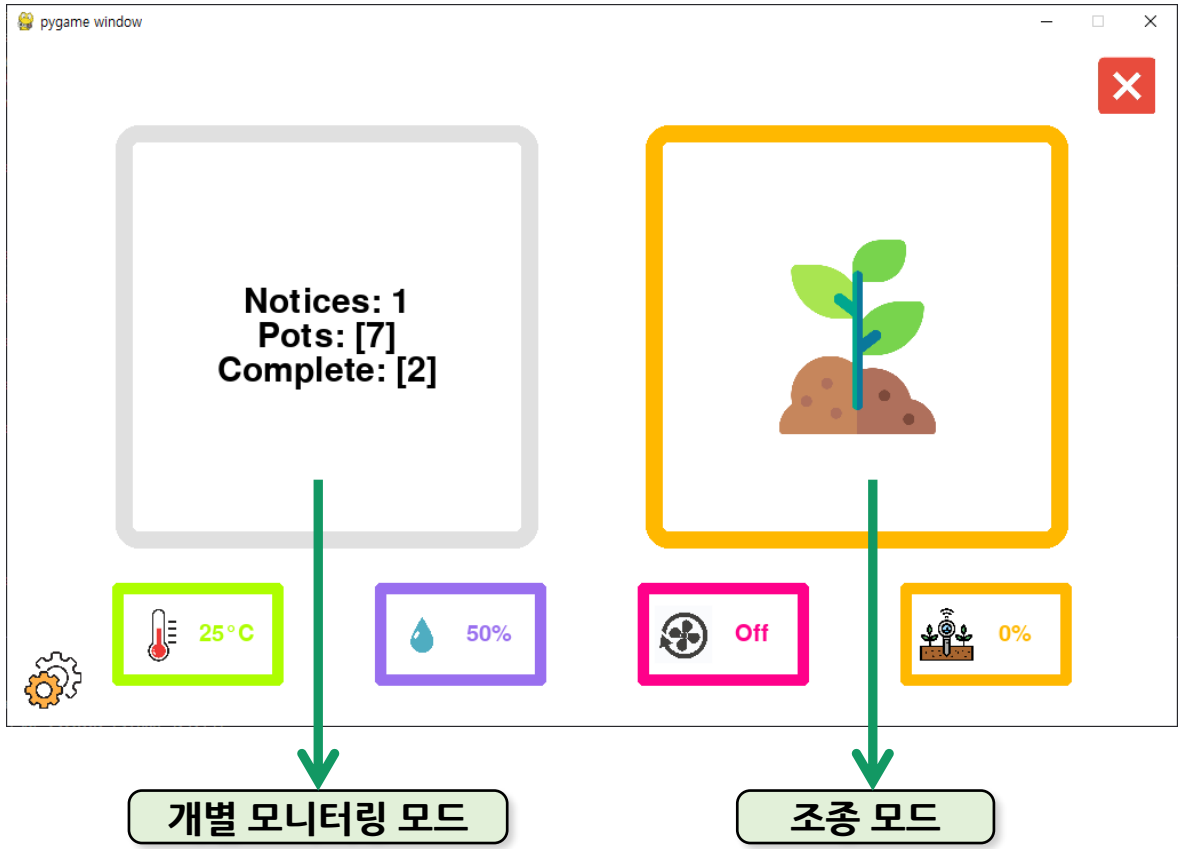
```python
#import RPi.GPIO as GPIO
import time
import signal
import sys
import pygame
import serial
import SmartFarmControl
import numpy as np

TEMP_LIMIT=3.0
HUMID_LIMIT=20.0
SOIL_LIMIT=20.0
INFO_ICON_SIZE=(150,90)
NOTIFICATION_SIZE=(370,370)
POT_GRID=(3,4)

#import SmartFarmControl
class Color():
    black=(0,0,0)
    gray=(224,224,224)
    white=(255,255,255)
    yellow=(255,184,0)
    magenta=(255,0,138)
    cyan=(82,0,255)
    skyblue=(0,240,255)
    green=(173,255,0)

class Info():
    def __init__(self,infos):
        self.critical=infos[0]
        self.temp=infos[1]
        self.status=infos[2]
    def printInfo(self):
        return str(self.critical)+" "+str(self.temp)

class potInfo():
    def __init__(self,potBool,infos):
        self.potBool=potBool
        self.infos=infos
    def updatePotInfo(self,potBool,infos):
        self.potBool=potBool
        self.infos=infos
    def getPotInfo(self):
        if self.infos is None:
            return None
        return self.infos
    def returnPotInfo(self):
        return self.potBool,self.getPotInfo()

class potGridInfo():
    def __init__(self,grid=POT_GRID):
        self.grid=grid
        self.PotInfos=[potInfo(True,Info((False,25,"Good"))) for i in range(grid[0]*grid[1])]
        for i in range(grid[1]):
            self.PotInfos[i].updatePotInfo(False,None)
    def returnPotGridInfo(self,pos):
        return self.PotInfos[pos[0]*self.grid[1]+pos[1]].returnPotInfo()
    def printPotGridInfo(self):
        return "aaaa"
    def updatePotGridInfo(self,orig,dest):
        info=self.PotInfos[orig[0]*self.grid[0]+orig[1]]
        self.PotInfos[dest[0]*self.grid[0]+dest[1]]=potInfo(info.potBool,info.infos)
        self.PotInfos[orig[0]*self.grid[0]+orig[1]].updatePotInfo(False,None)
```

```python
class Button:
    def __init__(self,img_in,pos, img_act, action = None):
        self.width=img_in.get_width()
        self.height=img_in.get_height()
        self.img_in=img_in
        self.img_act=img_act
        self.pos=pos
        self.action=action
    def printScreen(self,display):
        display.blit(self.img_in,(self.pos[0]-self.width/2,self.pos[1]-self.height/2))
    def updateMouseOn(self,display,mouse):
        if self.pos[0] + self.width/2.0 > mouse[0] > self.pos[0]-self.width/2.0 and self.pos[1] + self.height/2.0 > mouse[1] > self.pos[1]-self.height/2.0:
            display.blit(self.img_act,(self.pos[0]-self.img_act.get_width()/2,self.pos[1]-self.img_act.get_height()/2))
        else:
            display.blit(self.img_in,(self.pos[0]-self.width/2,self.pos[1]-self.height/2))
    def updateClick(self,display,mouse):
        if self.pos[0] + self.width/2.0 > mouse[0] > self.pos[0]-self.width/2.0 and self.pos[1] + self.height/2.0 > mouse[1] > self.pos[1]-self.height/2.0:
            display.blit(self.img_act,(self.pos[0]-self.img_act.get_width()/2,self.pos[1]-self.img_act.get_height()/2))
            if self.action is not None:
                time.sleep(0.2)
                self.action()
class infoIcon:
    def __init__(self,img_in,name,info,unit,pos,size,thickness,radius,range):
        color=Color()
        self.img=img_in
        self.name=name
        self.info=info
        self.unit=unit
        self.pos=pos
        self.size=size
        self.thickness=thickness
        self.radius=radius
        self.range=range
        self.color=color.green

    def updateInfo(self,info):
        self.info=info

    def calculateColor(self,upper_bound,normal,lower_bound):
        color=Color()
        if normal is None:
            if self.name=="humid":
                up=np.array(color.cyan)
                down=np.array(color.gray)
            if self.name=="soil":
                up=np.array(color.skyblue)
                down=np.array(color.yellow)
            slope=(up-down)/(upper_bound-lower_bound)
            return down+slope*(self.info-lower_bound)
        else:
            if self.name=="temp":
                up=np.array(color.magenta)
                down=np.array(color.cyan)
                middle=np.array(color.green)
                if self.info>normal:
                    slope=(up-middle)/(upper_bound-normal)
                    return middle+slope*(upper_bound-self.info)
                else:
                    slope=(middle-down)/(normal-lower_bound)
                    return down+slope*(self.info-lower_bound)

    def setColor(self):
        color=Color()
        if self.name=="temp":
```

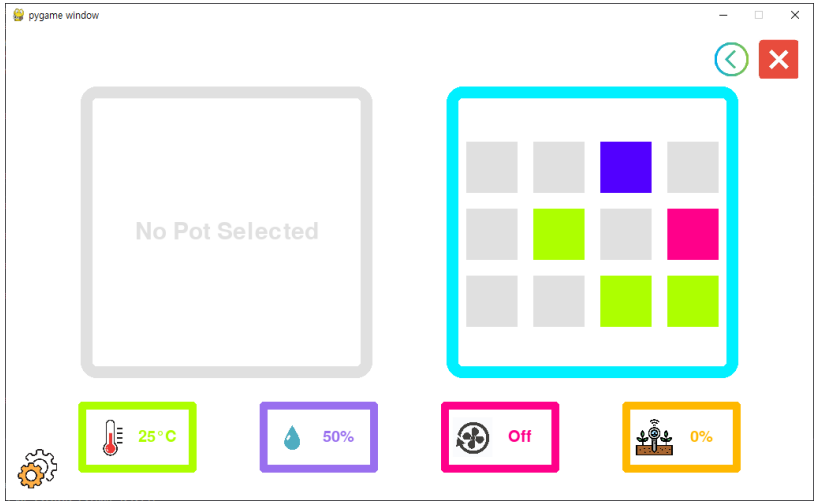
```python
    def printScreen(self,display,info,font):
        self.updateInfo(info)
        self.setColor()
        pygame.draw.rect(
            display,
            self.color,
            pygame.Rect(
                (self.pos[0]-self.size[0]/2,self.pos[1]-self.size[1]/2),
                self.size,
            ),
            self.thickness,
            self.radius
        )
        display.blit(self.img,(self.pos[0]-self.size[0]/2+15,self.pos[1]-self.size[1]/2+20))

        if type(self.info)==bool:
            if self.info==True:
                text=font.render("On"+self.unit,True,self.color)
                text_rect=text.get_rect()
                text_rect.center=(self.pos[0]+25,self.pos[1])
            else:
                text=font.render("Off"+self.unit,True,self.color)
                text_rect=text.get_rect()
                text_rect.center=(self.pos[0]+25,self.pos[1])
        else:
            if type(self.info) is not str:
                text=font.render(str(self.info)+self.unit,True,self.color)
            else:
                text=font.render(self.info+self.unit,True,self.color)
            text_rect=text.get_rect()
            text_rect.center=(self.pos[0]+25,self.pos[1])
        display.blit(text,text_rect)

class Notification:
    def __init__(self,infos,pos,size,thickness,radius,mode, action = None):
        self.info=infos
        self.pos=pos
        self.size=size
        self.thickness=thickness
        self.radius=radius
        self.mode=mode
        self.action=action
    def printScreen(self,display,font):
        color=Color()
        pygame.draw.rect(
            display,
            color.gray,
            pygame.Rect(
                (self.pos[0]-self.size[0]/2,self.pos[1]-self.size[1]/2),
                self.size,
            ),
            self.thickness,
            self.radius,
        )
        if self.info is None:
            text=font.render("No Pot Selected",True,color.gray)
            text_rect=text.get_rect()
            text_rect.center=self.pos
            display.blit(text,text_rect)
        else:
            temp=self.info.temp
            status=self.info.status

            status="Status : "+status
```

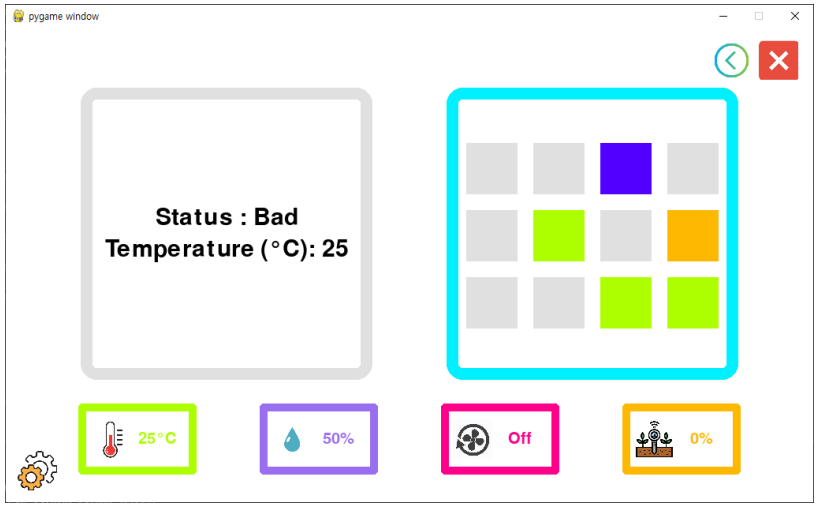
5. SW 제작 - 사용자 인터페이스

메인 화면



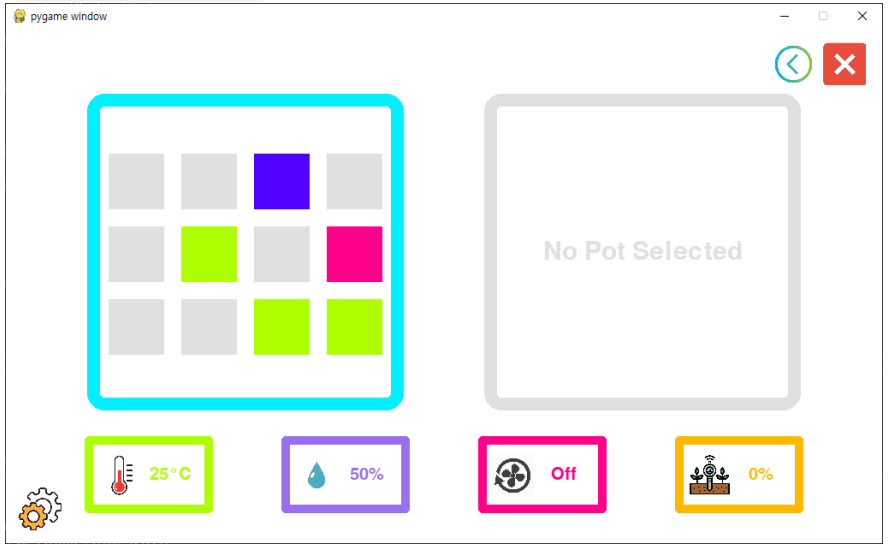
개별 모니터링 모드

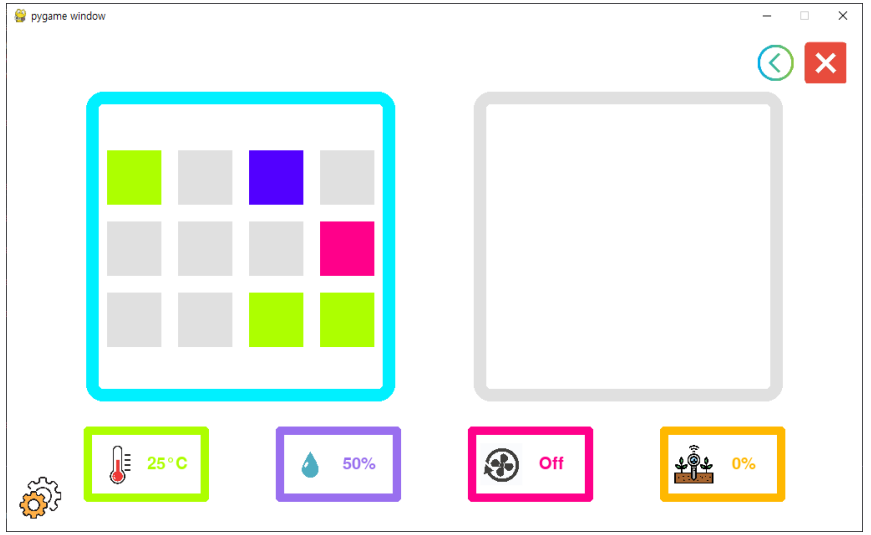


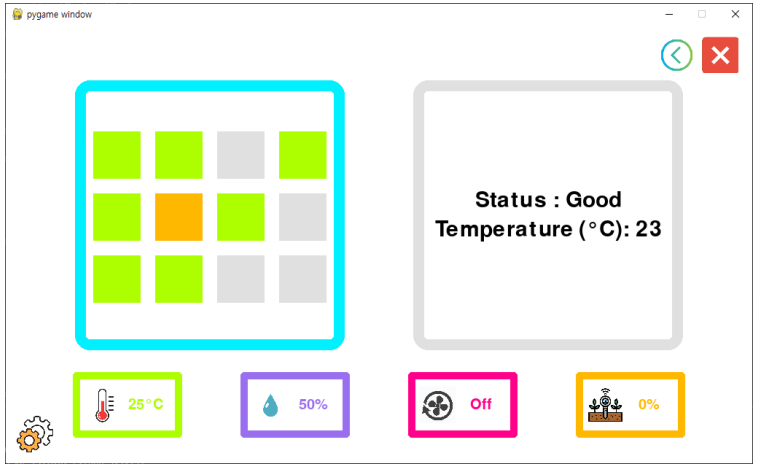


5. SW 제작 - 사용자 인터페이스

조종 모드







6. 결과

장점

1) 모듈화를 통한 부스 크기조절
2) 최소 단위(화분)로 개별 관리
3) 터치형 UI - 사용자 진입장벽 감소, 사용성 개선

개선점

1) 개별 관리 기능 개선 - 센서 자동 탈부착 구조 개발
2) 모니터링 기능 개선 - 비전 활용
3) 환경 유지 기능 개선 - 급수기, 조명