

자동관리 모듈형 스마트팜

경민호

기계공학과 이민호
기계공학과 남경민

목차

1

문제
정의

2

기능
정의

3

설계

4

HW
제작

5

SW
제작

6

결과

1. 문제 정의

연도별 이상기후로 인한 농업 재해 피해 현황



스마트팜



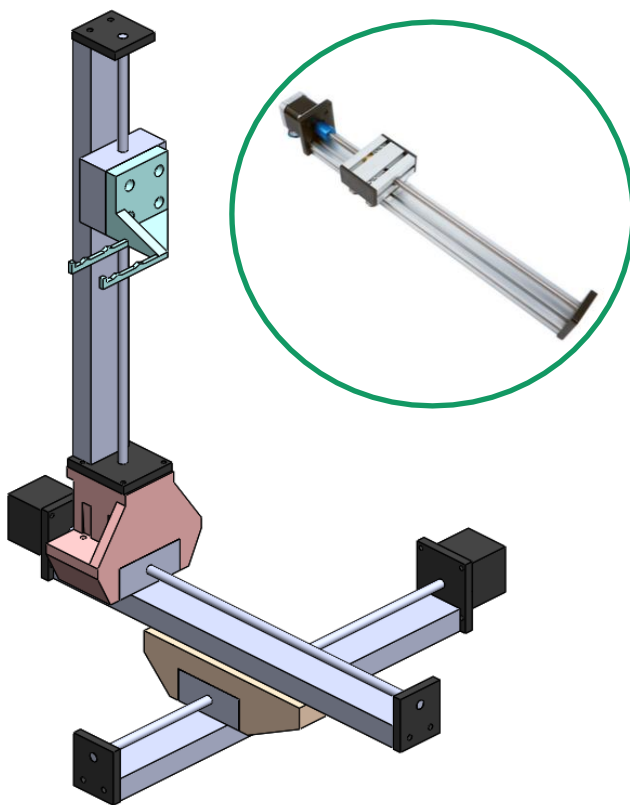
공간 활용도

생산성

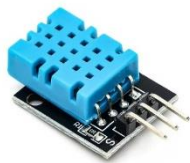
식품 접근성

2. 기능 정의

기능 1 자동 재배



기능 2 센서를 이용한 환경 모니터링 및 유지



온습도



조도



토양 수분



바람

기능 3 사용자 인터페이스



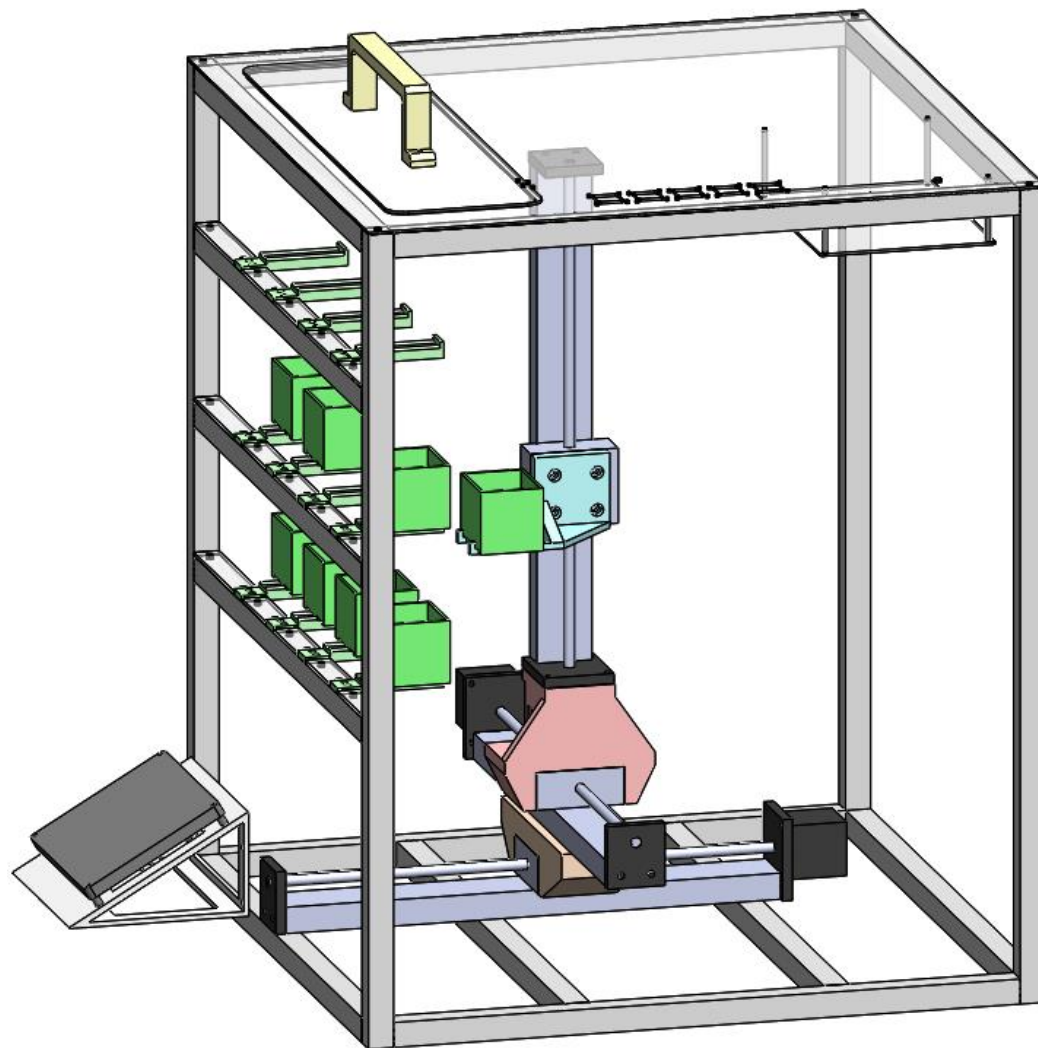
3. 설계

프레임

- 프로파일
- 아크릴
- 화분 거치대
- 화분

구동부

- LCD
- LCD 거치대

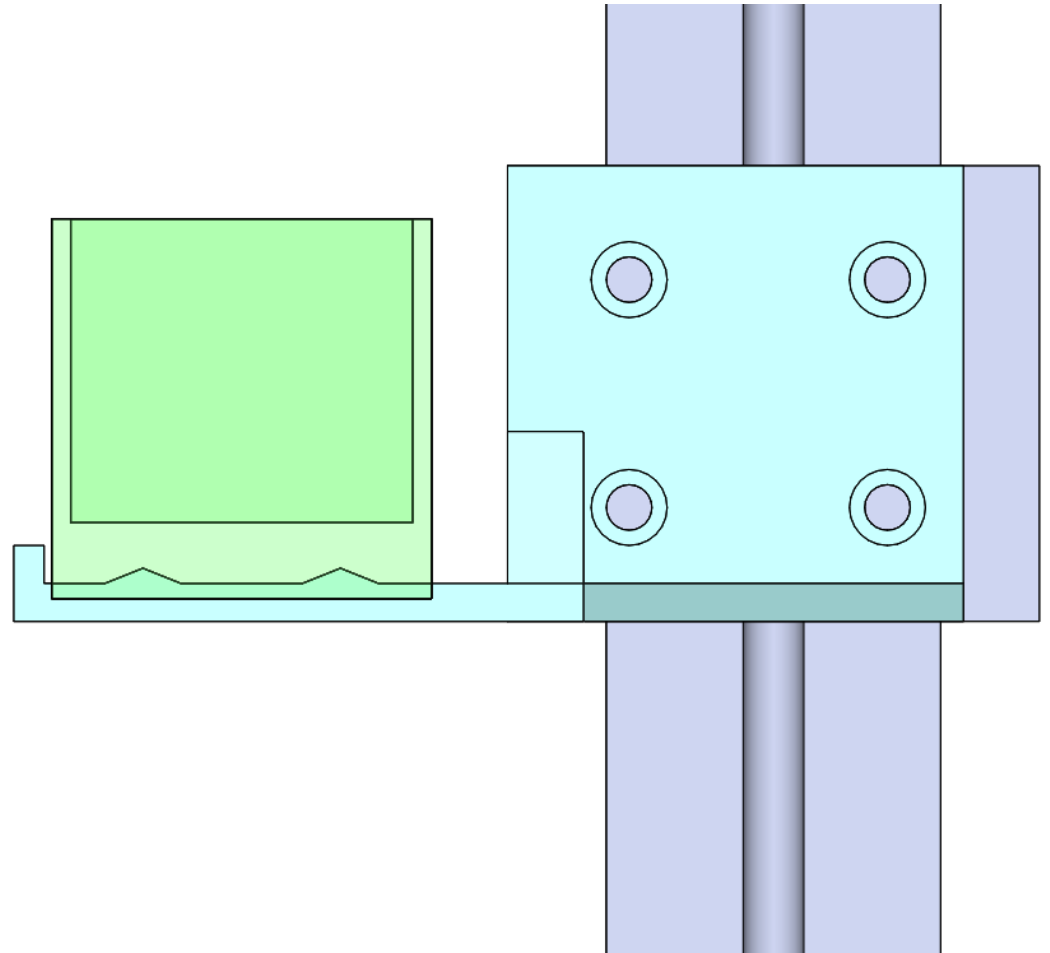
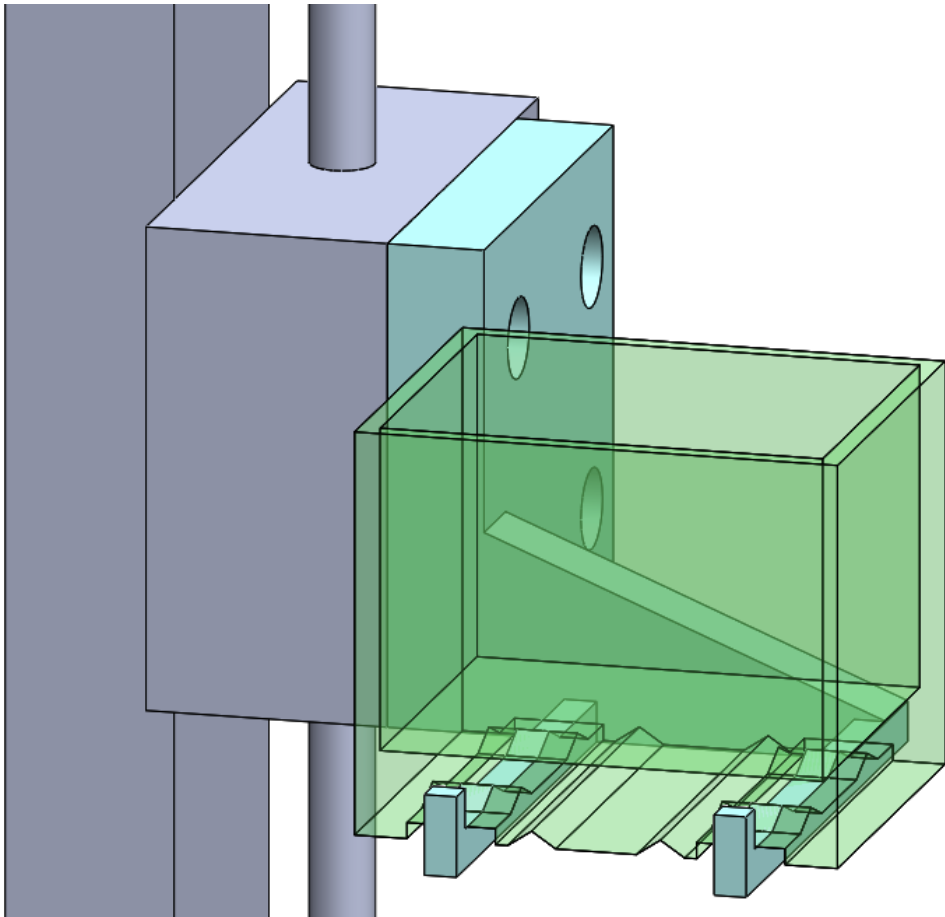


구동부

- 모터 연결부 1
- 모터 연결부 2
- 그리퍼

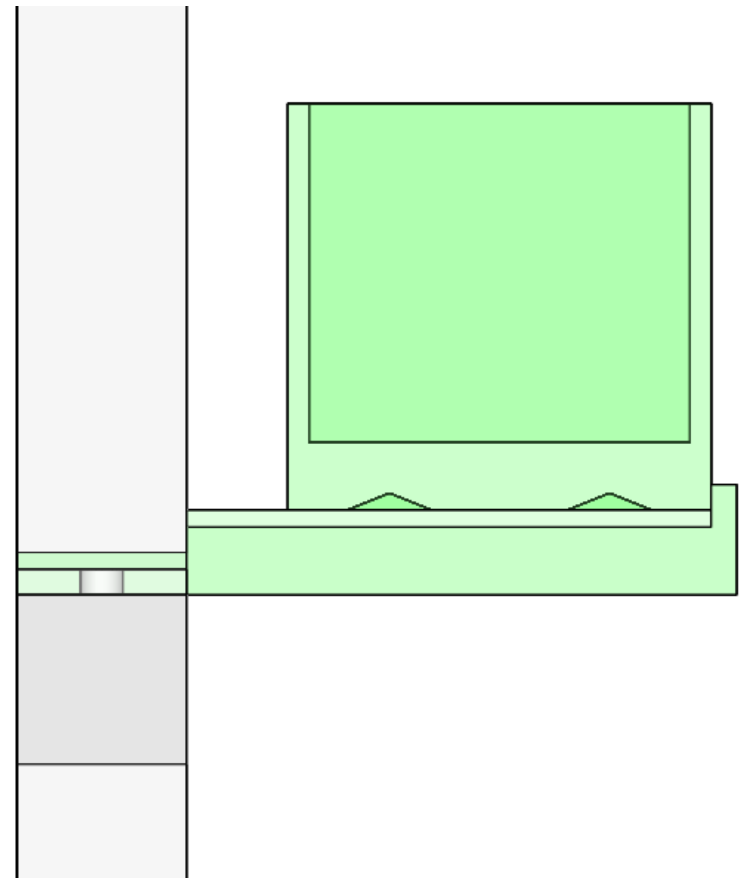
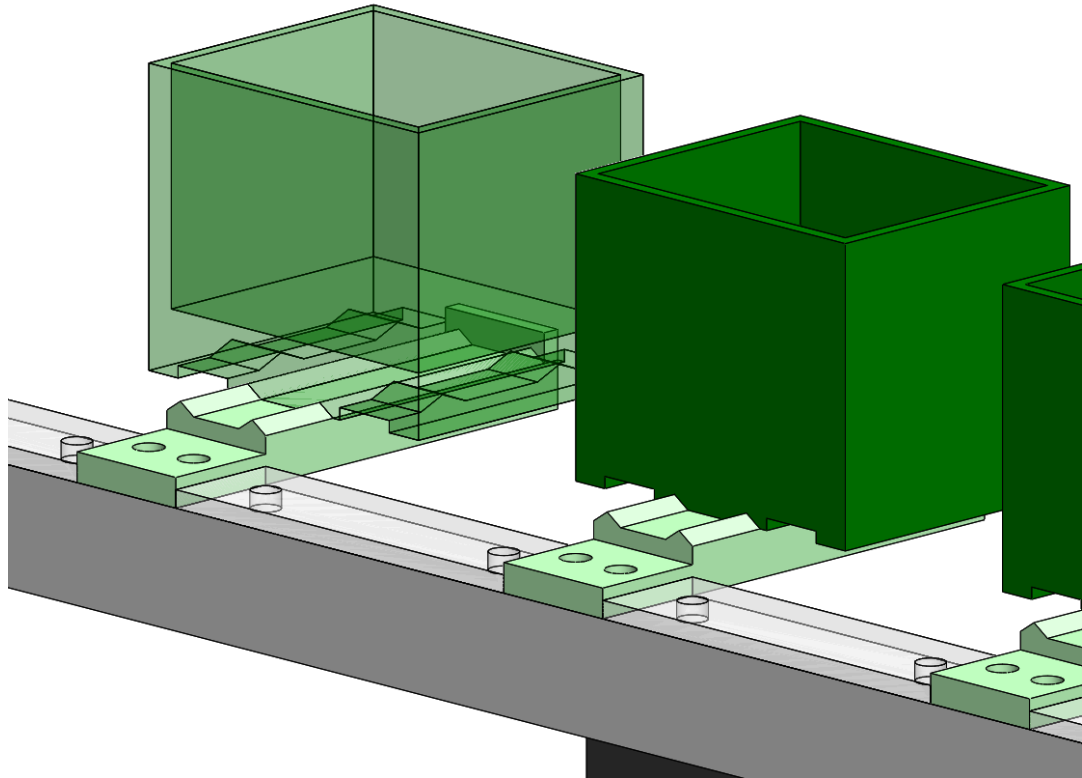
3. 설계

화분 - 그리퍼 연결 부분



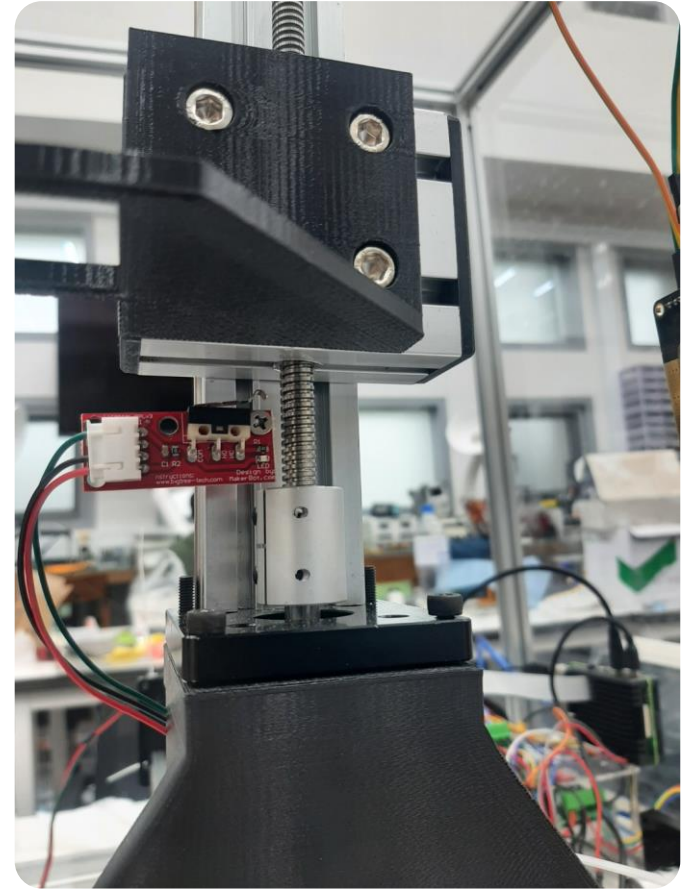
3. 설계

화분 거치대 - 화분 연결 부분



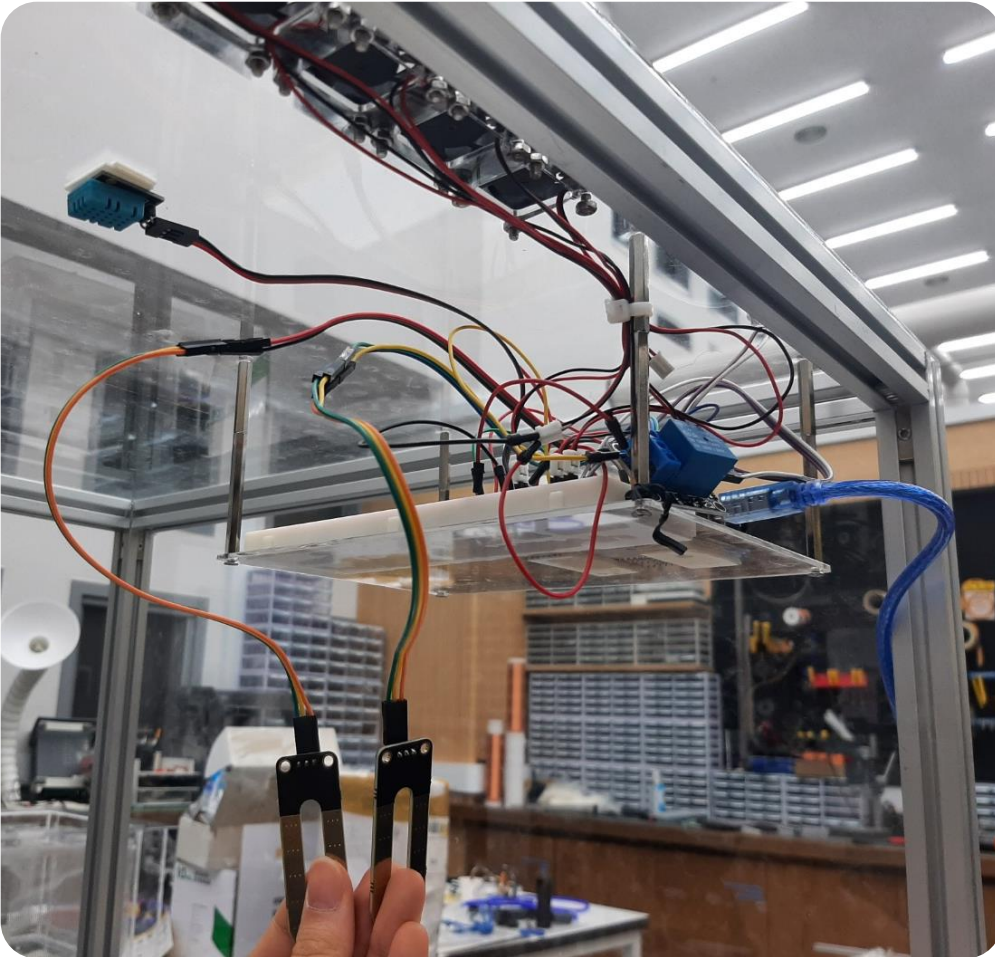
4. HW 제작

1. 프레임 및 구동부 제작

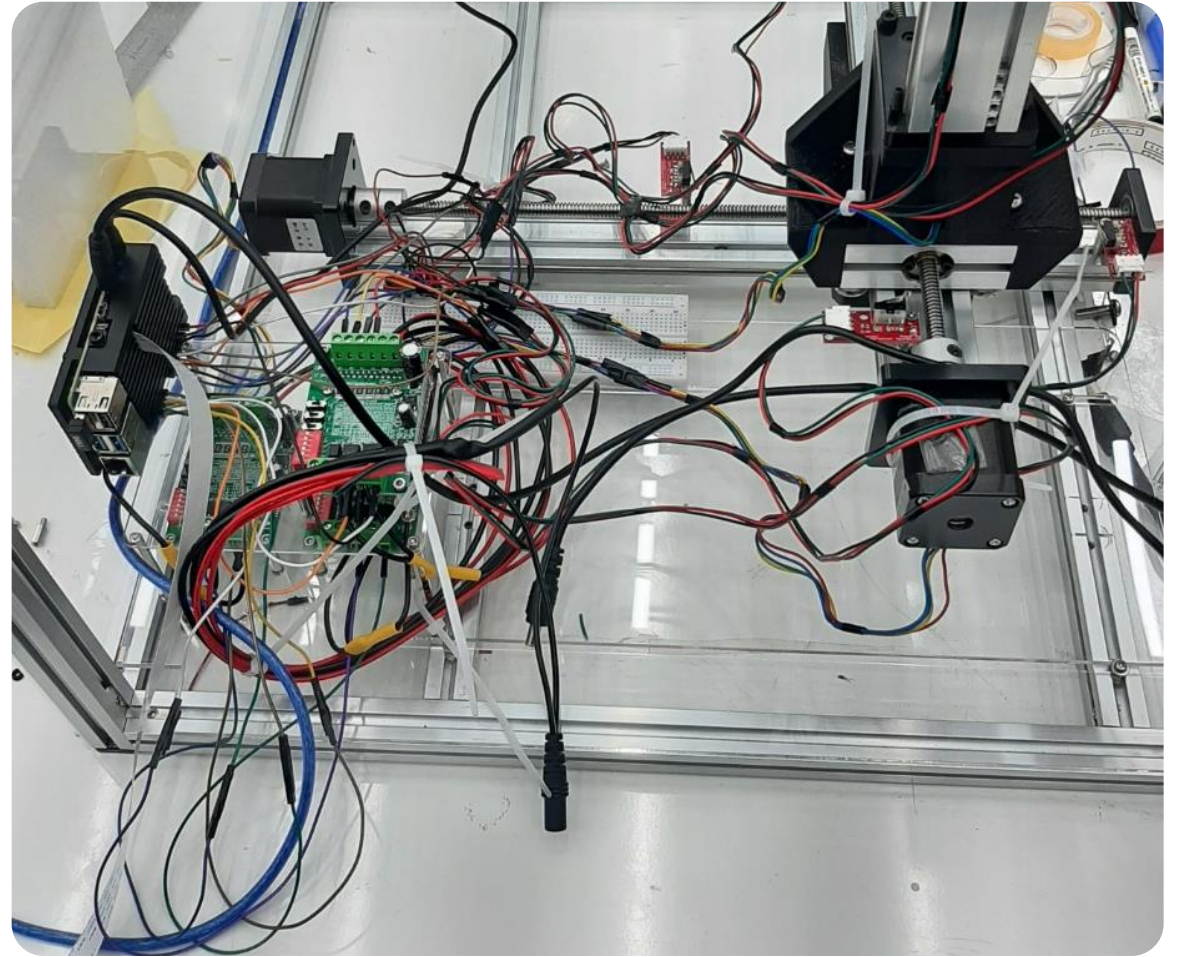


4. HW 제작

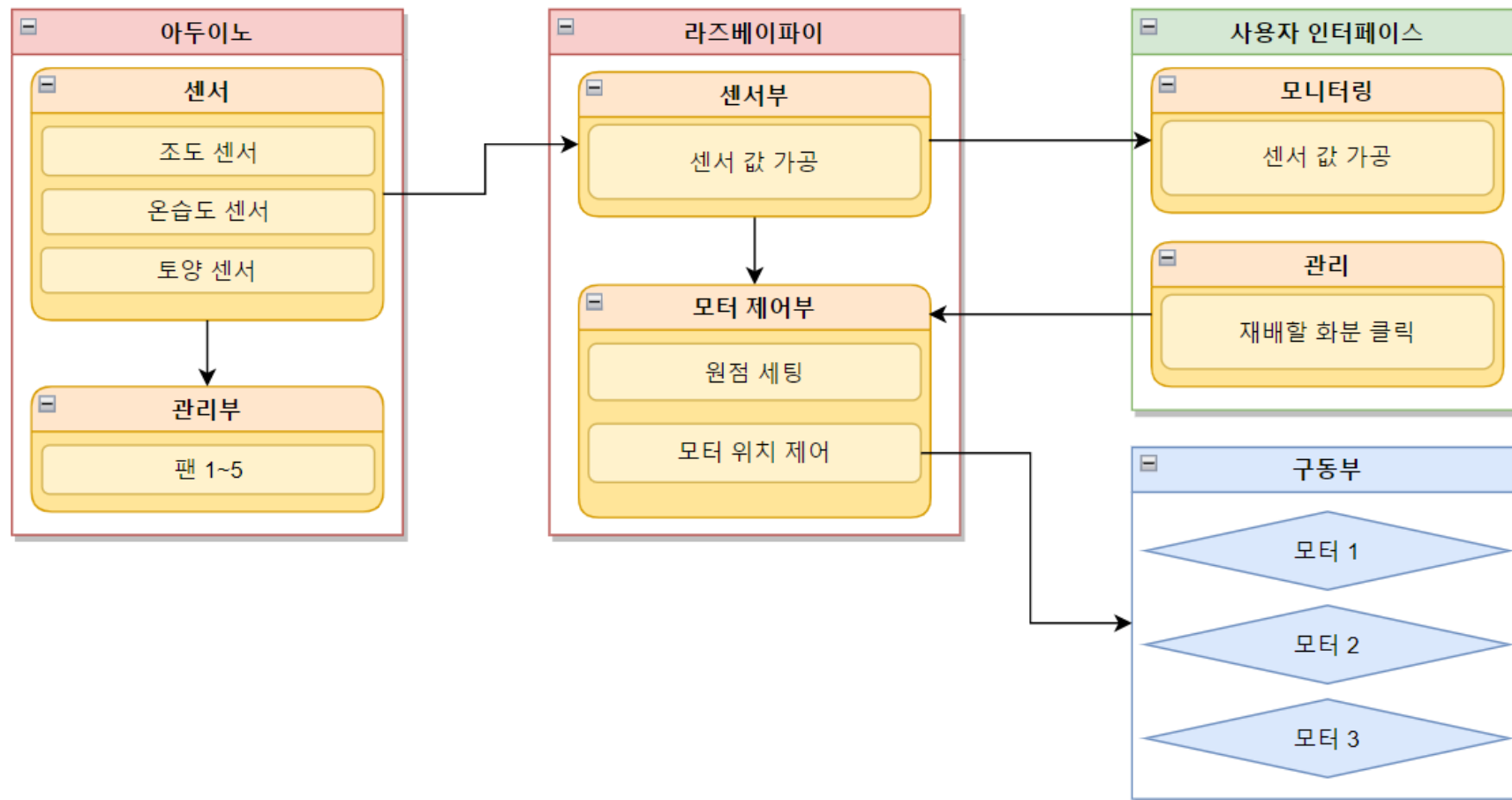
2. 아두이노 회로 제작 및 결합



3. 라즈베리 파이 회로 제작 및 결합

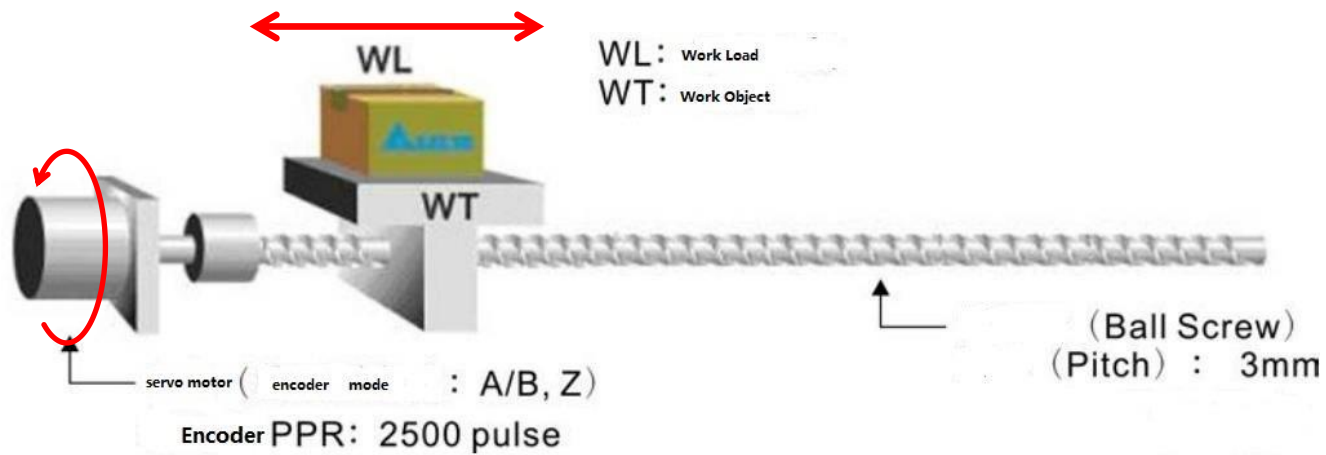


5. SW 제작



5. SW 제작 - 모터 제어부

회전수 기반 위치 제어



5. SW 제작 - 모터 제어부

코딩

```
1 import RPi.GPIO as GPIO
2 import time
3 from enum import Enum
4 import numpy as np
5 import signal
6 import sys
7
8 MOTOR_X_CLK_PIN=13
9 MOTOR_X_CLK_PIN=6
10 MOTOR_Y_CLK_PIN=26
11 MOTOR_Y_CLK_PIN=19
12 MOTOR_Z_CLK_PIN=21
13 MOTOR_Z_CLK_PIN=20
14
15 END_SWITCH_X1=22
16 END_SWITCH_X2=23
17 END_SWITCH_Y1=24
18 END_SWITCH_Y2=25
19 END_SWITCH_Z1=27
20 END_SWITCH_Z2=17
21
22 X_LEN=24786
23 Y_LEN=14882
24 Z_LEN=26762
25
26 X_UNIT=9600
27 Z_UNIT=10200
28
29 X_OFFSET=4200
30 Y_OFFSET=6500
31 Z_OFFSET=3000
32
33 Y_IN_DIST=6400
34 Y_OUT_DIST=-7000
35 Z_UP_DIST=3500
36
37 OUT=GPIO.OUT
38 IN=GPIO.IN
39
40 ROTATION_T=0.001
41 STOP_T=0.001
42 ORIGIN=4000
43 GRID=(3,4)
44
45 class Motor(Enum):
46     X = 0
47     Y = 1
48     Z = 2
49
50 class Dir(Enum):
51     CW = True #GPIO.HIGH
52     CCW = False #GPIO.LOW
53
54 class SmartFarmControl():
55     def __init__(self):
56         self.xpos=0
57         self.ypos=0
58         self.zpos=0
59         self.xdir=True
60         self.ydir=True
61         self.zdir=True
62
63         self.xlen=X_LEN
64         self.ylen=Y_LEN
65         self.zlen=Z_LEN
66
67         self.emergencyStop=False
68
69         self.modes=["initialization" for i in range(3)]
70         self.setPinMode()
71
72         # self.initializing_end_to_end()
73
74         ***
75
76         An initialization function
77
78         Parameters
79         ~~~~~
80         origin: origin of passenger
81
82         dest: destination of passenger
83
84         state: one of WAIT, ONBOARD, ARRIVAL
85         ~~~~~
86         4200
87
88         ***
89
90     def setPinMode(self):
91         GPIO.setmode(GPIO.BCM)
92         GPIO.setup(MOTOR_X_CLK_PIN,OUT)
93         # GPIO.setmode(GPIO.BCM)
94         GPIO.setup(MOTOR_X_CLK_PIN,OUT)
95
96         # GPIO.setmode(GPIO.BCM)
97         GPIO.setup(MOTOR_Y_CLK_PIN,OUT)
98         # GPIO.setmode(GPIO.BCM)
99         GPIO.setup(MOTOR_Z_CLK_PIN,OUT)
100
101         # GPIO.setmode(GPIO.BCM)
102         GPIO.setup(MOTOR_Z_CLK_PIN,OUT)
103         # GPIO.setmode(GPIO.BCM)
104         GPIO.setup(MOTOR_Z_CLK_PIN,OUT)
105
106         GPIO.setup(END_SWITCH_X1, IN, pull_up_down = GPIO.PUD_UP)
107         GPIO.setup(END_SWITCH_X2, IN, pull_up_down = GPIO.PUD_UP)
108         GPIO.add_event_detect(END_SWITCH_X1, GPIO.FALLING, callback=self.switchX1Pressed,bouncetime=300)
109         GPIO.add_event_detect(END_SWITCH_X2, GPIO.FALLING, callback=self.switchX2Pressed,bouncetime=300)
110
111         GPIO.setup(END_SWITCH_Y1, IN, pull_up_down = GPIO.PUD_UP)
112         GPIO.setup(END_SWITCH_Y2, IN, pull_up_down = GPIO.PUD_UP)
113         GPIO.add_event_detect(END_SWITCH_Y1, GPIO.FALLING, callback=self.switchY1Pressed,bouncetime=300)
114         GPIO.add_event_detect(END_SWITCH_Y2, GPIO.FALLING, callback=self.switchY2Pressed,bouncetime=300)
115
116         GPIO.setup(END_SWITCH_Z1, IN, pull_up_down = GPIO.PUD_UP)
117         GPIO.setup(END_SWITCH_Z2, IN, pull_up_down = GPIO.PUD_UP)
118         GPIO.add_event_detect(END_SWITCH_Z1, GPIO.FALLING, callback=self.switchZ1Pressed,bouncetime=300)
119         GPIO.add_event_detect(END_SWITCH_Z2, GPIO.FALLING, callback=self.switchZ2Pressed,bouncetime=300)
120
121     def getMotorNum(self,motor):
122         if motor==Motor.X: return 0
123         if motor==Motor.Y: return 1
124         if motor==Motor.Z: return 2
125
126     def checkMode(self):
127         for mode in self.modes:
128             if mode=="initialization":
129                 return False
130             return True
131
132     def checkMotorMode(self,motor):
133         num=self.getMotorNum(motor)
134         if self.modes[num]=="initialization":
135             return False
136         else:
137             return True
138
139     GPIO.output(MOTOR_X_CLK_PIN,dir)
140
141     elif motor==Motor.Y:
142         self.ydir=dir
143         GPIO.output(MOTOR_Y_CLK_PIN,not dir)
144
145     elif motor==Motor.Z:
146         self.zdir=dir
147         GPIO.output(MOTOR_Z_CLK_PIN,not dir)
148
149     def setMotorsDir(self,distances):
150         if distances[0]>0:
151             self.setMotorRotationDir(Motor.X,True)
152         else:
153             self.setMotorRotationDir(Motor.X,False)
154         if distances[1]>0:
155             self.setMotorRotationDir(Motor.Y,True)
156         else:
157             self.setMotorRotationDir(Motor.Y,False)
158         if distances[2]>0:
159             self.setMotorRotationDir(Motor.Z,True)
160         else:
161             self.setMotorRotationDir(Motor.Z,False)
162
163     def setMotorsRotationDir(self,motors,dir):
164         for motor in motors:
165             self.setMotorRotationDir(motor,dir)
166
167     def setDirection(self,dir):
168         if dir is True:
169             return 1
170         else:
171             return -1
172
173     def moveMotors(self,motors):
174         if motors is None:
175             return
176         for motor in motors:
177             if motor==Motor.X:
178                 self.xpos+=self.setDirection(self.xdir)
179                 GPIO.output(MOTOR_X_CLK_PIN,True)
180             elif motor==Motor.Y:
181                 self.ypos+=self.setDirection(self.ydir)
182                 GPIO.output(MOTOR_Y_CLK_PIN,True)
183             elif motor==Motor.Z:
184                 self.zpos+=self.setDirection(self.zdir)
185                 GPIO.output(MOTOR_Z_CLK_PIN,True)
186             time.sleep(ROTATION_T)
187         for motor in motors:
188             if motor==Motor.X:
189                 GPIO.output(MOTOR_X_CLK_PIN,False)
190             elif motor==Motor.Y:
191                 GPIO.output(MOTOR_Y_CLK_PIN,False)
192             elif motor==Motor.Z:
193                 GPIO.output(MOTOR_Z_CLK_PIN,False)
194             time.sleep(STOP_T)
195
196     def movableMotors(self,distances):
197         motors=[]
198         if distances[0]!=0:
199             motors.append(Motor.X)
200         if distances[1]!=0:
201             motors.append(Motor.Y)
202         if distances[2]!=0:
203             motors.append(Motor.Z)
204         return motors
205
206     def updateDistance(self,distances):
207
208         return distances
209
210     def moveMotorsDistance(self,distances):
211         distances=np.array(distances)
212         distances=distances.astype(int)
213         # print(distances)
214         if distances[0]==0:
215             if distances[0]+self.xpos>self.xlen:
216                 distances[0]=self.xlen-self.xpos-1
217             else:
218                 if distances[0]+self.xpos<0:
219                     distances[0]=-self.xpos+1
220         if distances[1]==0:
221             if distances[1]+self.ypos>self.ylen:
222                 distances[1]=self.ylen-self.ypos-1
223             else:
224                 if distances[1]+self.ypos<0:
225                     distances[1]=-self.ypos+1
226         if distances[2]==0:
227             if distances[2]+self.zpos>self.zlen:
228                 distances[2]=self.zlen-self.zpos-1
229             else:
230                 if distances[2]+self.zpos<0:
231                     distances[2]=-self.zpos+1
232         maxdist=max(abs(distances))
233         # print(distances)
234         self.setMotorsDir(distances)
235         for i in range(maxdist):
236             # print(self.movableMotors(distances))
237             self.moveMotors(self.movableMotors(distances))
238             distances=self.updateDistance(distances)
239
240     def moveMotorsToCoords(self,coords):
241         distances=self.calculateCoordDistance(coords)
242         self.moveMotorsDistance(distances)
243
244     def moveMotorsToOrigin(self):
245         # print(self.xpos)
246         # print(self.ypos)
247         # print(self.zpos)
248         distances=[ORIGIN-self.xpos,ORIGIN-self.ypos,ORIGIN-self.zpos]
249         # print(distances)
250         self.moveMotorsDistance(distances)
251
252     def moveMotorsOrigDest(self,orig,dest):
253         self.moveMotorsToCoords(orig)
254         self.moveMotorsDistance([0,0,Z_UP_DIST])#Z UP
255         time.sleep(0.1)
256         self.moveMotorsDistance([0,Y_IN_DIST,0])#Y IN
257         time.sleep(0.1)
258         self.moveMotorsDistance([0,0,Z_UP_DIST])#Z UP
259         time.sleep(0.1)
260         self.moveMotorsDistance([0,Y_OUT_DIST,0])#Y IN
261         time.sleep(0.1)
262         self.moveMotorsToCoords(dest)
263         self.moveMotorsDistance([0,0,Z_UP_DIST])#Z UP
264         time.sleep(0.1)
265         self.moveMotorsDistance([0,Y_IN_DIST,0])#Y IN
266         time.sleep(0.1)
267         self.moveMotorsDistance([0,0,Z_UP_DIST])#Z UP
268         time.sleep(0.1)
269         self.moveMotorsDistance([0,-Y_IN_DIST,0])#Y IN
270         time.sleep(0.1)
271         self.moveMotorsToOrigin()
272
273         self.xdir=True
274         # self.xlen=self.counter-self.xlen
275         # print(self.xlen)
276         self.modes[0]="normal"
277         self.setMotorRotationDir(Motor.X,True)
278
279         else:
280             self.xpos=0
281             self.xdir=True
282             self.setMotorRotationDir(Motor.X,True)
283
284         def switchX2Pressed(self,channel):
285             if self.modes[0]=="initialization":
286                 print("x2")
287                 self.xdir=False
288                 # self.xlen=self.counter
289                 self.setMotorRotationDir(Motor.X,False)
290             else:
291                 self.xpos=self.xlen
292                 self.xdir=False
293                 self.setMotorRotationDir(Motor.X,False)
294
295         def switchY1Pressed(self,channel):
296             if self.modes[1]=="initialization":
297                 print("y1")
298                 self.counter[1]=0
299                 self.ypos=0
300                 self.ydir=True
301                 # self.ylen=self.counter-self.ylen
302                 # print(self.ylen)
303                 self.modes[1]="normal"
304                 self.setMotorRotationDir(Motor.Y,True)
305             else:
306                 self.ypos=self.ylen
307                 self.ydir=True
308                 self.setMotorRotationDir(Motor.Y,True)
309
310         def switchY2Pressed(self,channel):
311             if self.modes[1]=="initialization":
312                 print("y2")
313                 self.ydir=False
314                 # self.ylen=self.counter
315                 self.setMotorRotationDir(Motor.Y,False)
316                 # self.emergencyStop=True
317             else:
318                 self.ypos=self.ylen
319                 self.ydir=False
320                 self.setMotorRotationDir(Motor.Y,False)
321
322         def switchZ1Pressed(self,channel):
323             if self.modes[2]=="initialization":
324                 self.counter[2]=0
325                 print("z1")
326                 self.zpos=0
327                 self.zdir=True
328                 # self.zlen=self.counter-self.zlen
329                 # print(self.zlen)
330                 self.modes[2]="normal"
331                 self.setMotorRotationDir(Motor.Z,True)
332             else:
333                 self.zpos=0
334                 self.zdir=True
335                 self.setMotorRotationDir(Motor.Z,True)
336
337         def switchZ2Pressed(self,channel):
338             if self.modes[2]=="initialization":
339                 print("z2")
340                 self.zdir=False
341                 # self.zlen=self.counter
342                 self.setMotorRotationDir(Motor.Z,False)
343             else:
344                 self.zpos=self.zlen
345                 self.zdir=False
346                 self.setMotorRotationDir(Motor.Z,False)
```


5. SW 제작 - 사용자 인터페이스

Pygame을 이용하여 제작



```
1 #import RPi.GPIO as GPIO
2 import time
3 import signal
4 import sys
5 import pygame
6 import serial
7 import SmartFarmControl
8 import numpy as np
9
10 TEMP_LIMIT=3.0
11 HUMID_LIMIT=20.0
12 SOIL_LIMIT=20.0
13 INFO_ICON_SIZE=(150,90)
14 NOTIFICATION_SIZE=(370,370)
15 POT_GRID=(3,4)
```

```
16
17 #import SmartFarmControl
18 class Color():
19     black=(0,0,0)
20     gray=(224,224,224)
21     white=(255,255,255)
22     yellow=(255,184,0)
23     magenta=(255,0,138)
24     cyan=(82,0,255)
25     skyblue=(0,240,255)
26     green=(173,255,0)
```

```
27
28 class Info():
29     def __init__(self,infos):
30         self.critical=infos[0]
31         self.temp=infos[1]
32         self.status=infos[2]
33     def printInfo(self):
34         return str(self.critical)+" "+str(self.temp)
```

```
35
36 class potInfo():
37     def __init__(self,potBool,infos):
38         self.potBool=potBool
39         self.infos=infos
40     def updatePotInfo(self,potBool,infos):
41         self.potBool=potBool
42         self.infos=infos
43     def getPotInfo(self):
44         if self.infos is None:
45             return None
46         return self.infos
47     def returnPotInfo(self):
48         return self.potBool,self.getPotInfo()
```

```
49
50 class potGridInfo():
51     def __init__(self,grid=POT_GRID):
52         self.grid=grid
53         self.potInfos=[potInfo(True,Info(False,25,"Good")) for i in range(grid[0]*grid[1])]
54         for i in range(grid[1]):
55             self.potInfos[i].updatePotInfo(False,None)
56     def returnPotGridInfo(self,pos):
57         return self.potInfos[pos[0]*self.grid[1]+pos[1]].returnPotInfo()
58     def printPotGridInfo(self):
59         return "aaaa"
60     def updatePotGridInfo(self,orig,dest):
61         info=self.potInfos[orig[0]*self.grid[0]+orig[1]]
62         self.potInfos[dest[0]*self.grid[0]+dest[1]]=potInfo(info.potBool,info.infos)
63         self.potInfos[orig[0]*self.grid[0]+orig[1]].updatePotInfo(False,None)
```

```
64
65 class Button:
66     def __init__(self,img_in,pos,img_act,action=None):
67         self.width=img_in.get_width()
68         self.height=img_in.get_height()
69         self.img_in=img_in
70         self.img_act=img_act
71         self.pos=pos
72         self.action=action
73     def printScreen(self,display):
74         display.blit(self.img_in,(self.pos[0]-self.width/2,self.pos[1]-self.height/2))
75     def updateMouseOn(self,display,mouse):
76         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:
77             display.blit(self.img_act,(self.pos[0]-self.img_act.get_width()/2,self.pos[1]-self.img_act.get_height()/2))
78         else:
79             display.blit(self.img_in,(self.pos[0]-self.width/2,self.pos[1]-self.height/2))
80     def updateClick(self,display,mouse):
81         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:
82             display.blit(self.img_act,(self.pos[0]-self.img_act.get_width()/2,self.pos[1]-self.img_act.get_height()/2))
83             if self.action is not None:
84                 time.sleep(0.2)
85                 self.action()
```

```
86
87 class InfoIcon:
88     def __init__(self,img_in,name,info,unit,pos,size,thickness,radius,range):
89         color=Color()
90         self.img_in=img_in
91         self.name=name
92         self.info=info
93         self.unit=unit
94         self.pos=pos
95         self.size=size
96         self.thickness=thickness
97         self.radius=radius
98         self.range=range
99         self.color=color.green
```

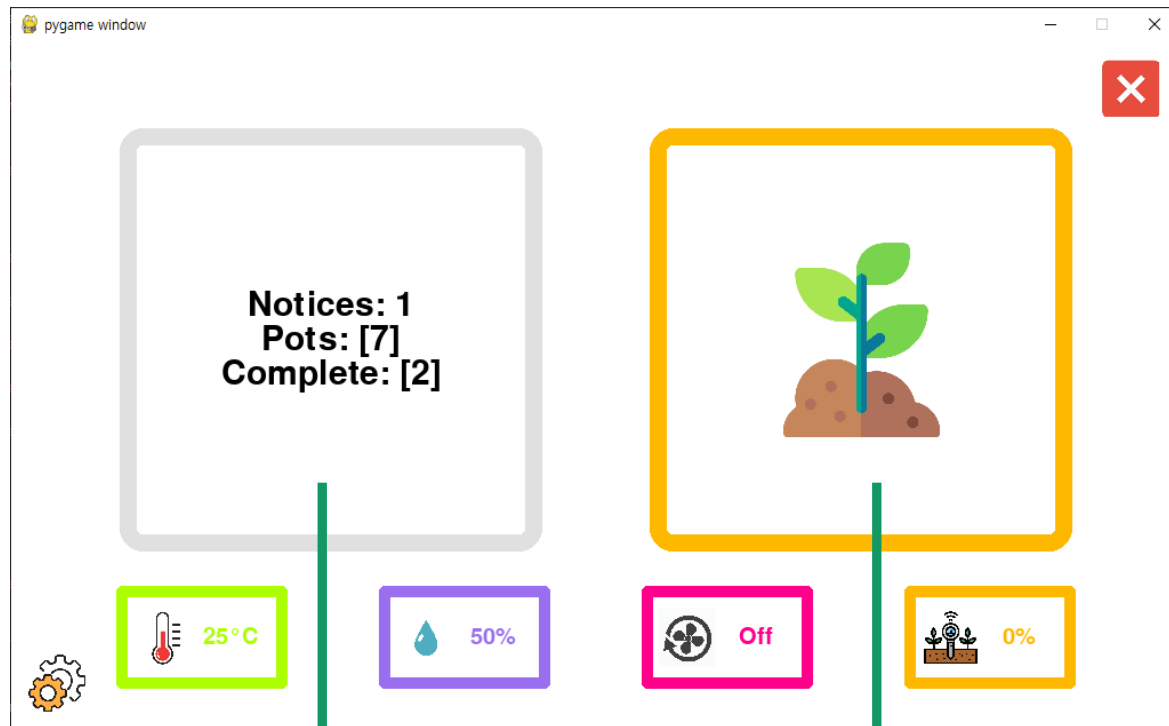
```
100
101     def updateInfo(self,info):
102         self.info=info
103
104     def calculateColor(self,self,upper_bound,normal,lower_bound):
105         color=Color()
106         if normal is None:
107             if self.name=="humid":
108                 up=np.array(color.cyan)
109                 down=np.array(color.gray)
110             if self.name=="soil":
111                 up=np.array(color.skyblue)
112                 down=np.array(color.yellow)
113             slope=(up-down)/(upper_bound-lower_bound)
114             return down+slope*(self.info-lower_bound)
115         else:
116             if self.name=="temp":
117                 up=np.array(color.magenta)
118                 down=np.array(color.cyan)
119                 middle=np.array(color.green)
120             if self.info>normal:
121                 slope=(up-middle)/(upper_bound-normal)
122                 return middle+slope*(upper_bound-self.info)
123             else:
124                 slope=(middle-down)/(normal-lower_bound)
125                 return down+slope*(self.info-lower_bound)
126
127     def setColor(self):
128         color=Color()
129         if self.name=="temp":
```

```
130
131     def printScreen(self,display,info,font):
132         self.updateInfo(info)
133         self.setColor()
134         pygame.draw.rect(
135             display,
136             self.color,
137             pygame.Rect(
138                 (self.pos[0]-self.size[0]/2,self.pos[1]-self.size[1]/2),
139                 self.size,
140             ),
141             self.thickness,
142             self.radius
143         )
144         display.blit(self.img,(self.pos[0]-self.size[0]/2+15,self.pos[1]-self.size[1]/2+20))
145
146         if type(self.info)==bool:
147             if self.info==True:
148                 text=font.render("On"+self.unit,True,self.color)
149                 text_rect=text.get_rect()
150                 text_rect.center=(self.pos[0]+25,self.pos[1])
151             else:
152                 text=font.render("Off"+self.unit,True,self.color)
153                 text_rect=text.get_rect()
154                 text_rect.center=(self.pos[0]+25,self.pos[1])
155         else:
156             if type(self.info) is not str:
157                 text=font.render(str(self.info)+self.unit,True,self.color)
158             else:
159                 text=font.render(self.info+self.unit,True,self.color)
160             text_rect=text.get_rect()
161             text_rect.center=(self.pos[0]+25,self.pos[1])
162         display.blit(text,text_rect)
```

```
163
164 class Notification:
165     def __init__(self,infos,pos,size,thickness,radius,mode,action=None):
166         self.info=infos
167         self.pos=pos
168         self.size=size
169         self.thickness=thickness
170         self.radius=radius
171         self.mode=mode
172         self.action=action
173     def printScreen(self,display,font):
174         color=Color()
175         pygame.draw.rect(
176             display,
177             color.gray,
178             pygame.Rect(
179                 (self.pos[0]-self.size[0]/2,self.pos[1]-self.size[1]/2),
180                 self.size,
181             ),
182             self.thickness,
183             self.radius,
184         )
185         if self.info is None:
186             text=font.render("No Pot Selected",True,color.gray)
187             text_rect=text.get_rect()
188             text_rect.center=self.pos
189             display.blit(text,text_rect)
190         else:
191             temp=self.info.temp
192             status=self.info.status
193             status="Status : "+status
```

5. SW 제작 - 사용자 인터페이스

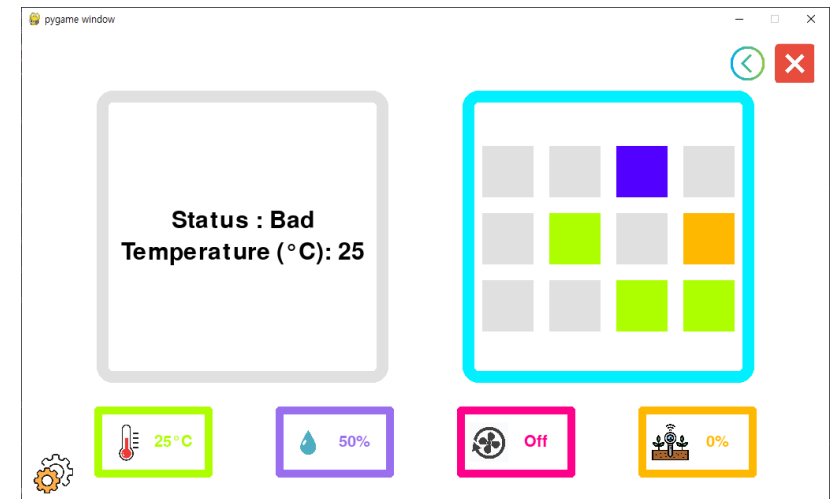
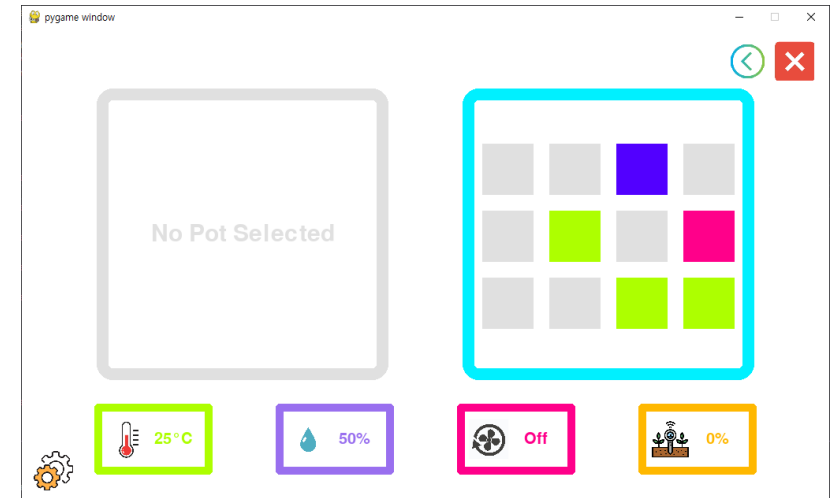
메인 화면



개별 모니터링 모드

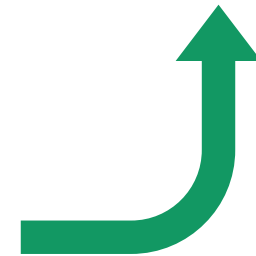
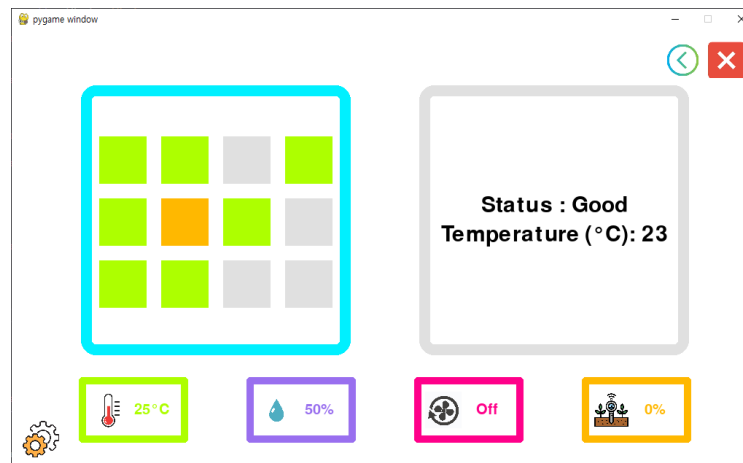
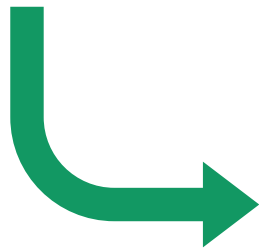
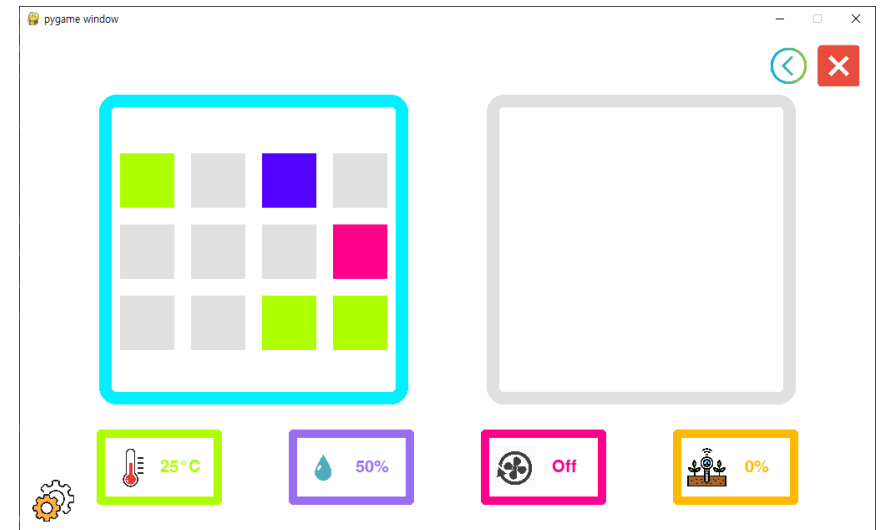
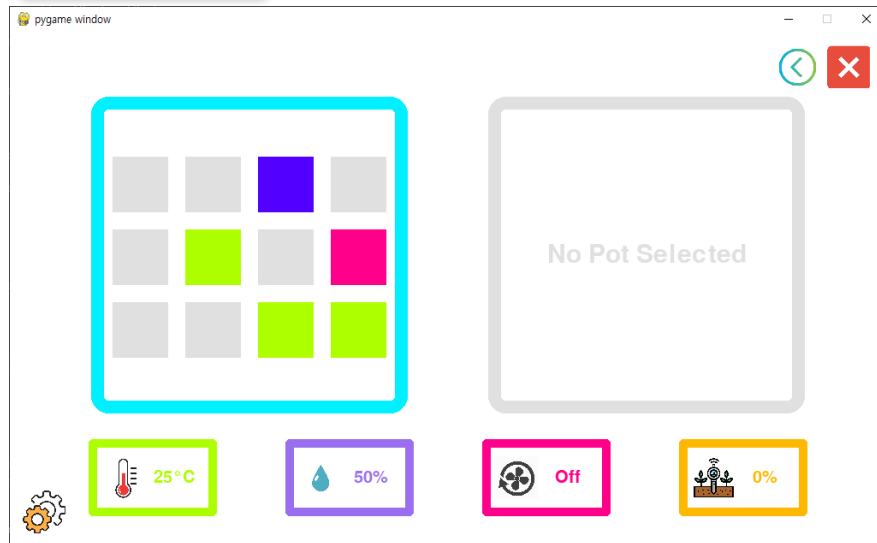
조종 모드

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