



동아대학교
공과대학
전자공학과


캡스톤 디자인

종합설계 결과보고서

프로젝트 명	재난 지역 탐사용 무선 지상 드론
팀 명	팀 DCS

Version	1.0
Date	2021-DEC-09

팀원	성명	학번	연락처
	김태욱		
	윤주흔		
	최재원		
	최지훈		
	허준석		

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

CONFIDENTIALITY/SECURITY WARNING

이 문서에 포함되어 있는 정보는 동아대학교 공과대학 전자공학과 개설 교과목 캡스톤 디자인 수강 학생 중 프로젝트 “재난 지역 탐사용 무선 지상 드론” 을 수행하는 “팀 DCS”의 팀원들의 자산입니다. 동아대학교 전자공학과 및 “팀 DCS”의 팀원들의 서면 허락없이 사용되거나, 재가공 될 수 없습니다.

문서 정보 / 수정 내역

Filename	결과보고서 팀 DCS.docx
원안작성자	윤주흔, 김태욱, 최재원, 최지훈, 허준석
수정작업자	윤주흔

수정날짜	대표수정자	Revision	추가/수정항목	내 용
2021-12-07	윤주흔	0.8	최초 작성	초안 작성
2021-12-07	김태욱	0.8	최초 작성	담당부분 작성
2021-12-07	최재원	0.8	최초 작성	담당부분 작성
2021-12-07	최지훈	0.8	최초 작성	담당부분 작성
2021-12-07	허준석	0.8	최초 작성	담당부분 작성
2021-12-08	윤주흔	0.9	수정 작업	내용 수정 및 보완
2021-12-09	윤주흔	1.0	수정 작업	최종 정리

 <div> 동아대학교 전자공학과 캡스톤 디자인 </div>	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

목 차

1	연구 배경 및 목적.....	4
2	프로젝트 목표.....	5
2.1	프로젝트 목표	5
2.2	개발 조건	5
3	작품 구성.....	6
3.1	작품의 형상	6
3.2	작품의 대략적인 구성	7
3.3	구동부	8
3.3.1	DC 모터.....	8
3.3.2	모터 드라이버	9
3.3.3	기어 박스	10
3.3.4	구동부 동작 코드	12
3.4	센서부	14
3.4.1	초음파 센서	14
3.4.2	온습도 센서	15
3.4.3	센서부 동작 코드	16
3.5	시계부	23
3.5.1	파이 카메라	23
3.5.2	서보 모터	25
3.5.3	서보 모터 동작 코드	27
3.6	전원부	28
3.6.1	18650 리튬 이온 충전지	29
3.6.2	AAA규격 알카라인 전지	30
3.6.3	시판 보조배터리	31
3.7	제어부	32
3.7.1	라즈베리파이 4 B+	32
3.7.2	라즈베리파이 제어 코드.....	33
3.8	통신부	47
3.8.1	라즈베리파이 제어 코드.....	47
3.8.2	라즈베리파이 제어 코드.....	47
3.8.3	라즈베리파이 제어 코드.....	51
4	역할 분담 및 개발 일정	72
4.1	역할 분담	73
4.2	개발 일정	74
5	기대 효과.....	75

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

1 연구 배경 및 목적

현재 인류는 눈부신 과학의 발전으로 과거 여지껏 없던 안전하고 풍요로운 사회를 이루어 냈습니다. 하지만 그럼에도 실수에 의한, 인재에 의한, 자연에 의한 피해는 계속해서 발생하고 있습니다.

잘 알려져 있는, 테러로 인한 미국의 세계무역 센터 붕괴 사건이나 한국의 부실 공사로 인한 삼풍백화점 붕괴 사고, 일본의 옴 진리교의 도쿄 지하철 사린 가스 살포 사건, 중국의 텐진 항 폭발 사고 등의 사건/사고 들은 한 가지 공통점을 가지고 있습니다. 재난지역의 피해자가 구조를 바라고 있는 상황에서 구조 대원의 즉각 투입에 한계가 발생하여 피해자의 위치나 처한 상황 등 구조에 필요한 정보의 수집에 어려움이 발생한다는 것입니다.

이러한 사고는 우리 주변에서 흔하게 발생하고 있습니다. 당장 올해 6월 9일 광주에서 철거중이던 5층 건물이 붕괴하여 시내버스를 덮치며 사람들이 매몰된 사고가 있^고 같은 달 24일 미국 플로리다주 마이애미에서도 주민들이 거주하던 아파트가 한밤중 붕괴하여 99명이 사실상 매몰되는 사고가 발생하였습니다. 해당 사고에서 실제로 침대 매트리스 아래에서 10살 소년이 구조되는 등 여러 생존자들이 매몰된 상태에서 구조되^는습니다.

본 과제의 드론은 구조 대원이 투입되기 어려운 재난 지역에 즉각적으로 대신 투입되어 신속하고 유연한 대응을 위해 피해자의 탐색과 재난 지역의 환경 정보를 수집하는 것을 목표로 합니다.

재난을 사전에 막는 것이 가장 좋은 방법이겠지만 아무리 대비를 하여도 재난을 완전히 방지한다는 것은 불가능에 가깝습니다. 따라서 재난이 발생한 후의 일도 대비를 하여야 하기에 해당 과제를 선정하게 되^는습니다.

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

2 프로젝트 목표

2.1 프로젝트 목표

저희 팀 DCS의 드론은 재난 지역에 구조 대원을 대신하여 투입되어 현장의 상황을 파악하고 피해자를 조기에 발견하는 것을 목표로 하고 있습니다. 따라서 다음 조건을 목표로 개발을 수행하였습니다.

2.2 개발 조건

조건 1. 본 드론의 모든 동작은 원격으로 수행되어야 한다.

조건 2. 본 드론은 동작 중 외부 전원을 필요로 하지 않고 드론 자체에서 해결되어야 한다.

조건 3. 본 드론은 실시간 영상을 전송할 수 있어야 한다.

조건 4. 본 드론은 주변의 온도와 습도를 파악하고 해당 정보를 무선으로 전송할 수 있어야 한다.

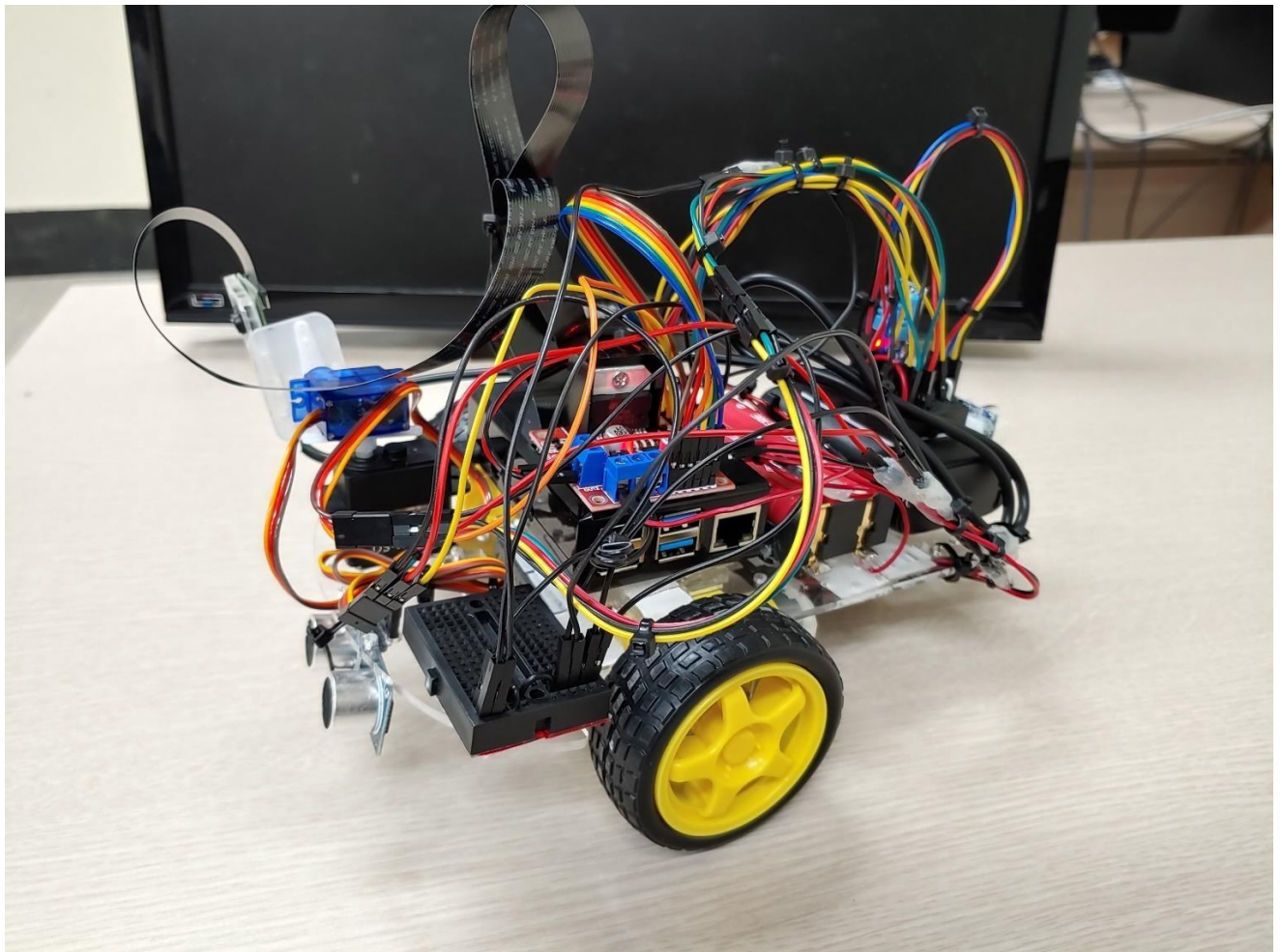
조건 5. 본 드론의 전면과 후면에는 초음파 센서가 장착되어 장애물을 파악할 수 있어야 한다.

조건 6. 본 드론은 불의의 사고로 통신이 제한되어 조종이 불가능한 상황에도 독립적으로 귀환할 수 있어야 한다.

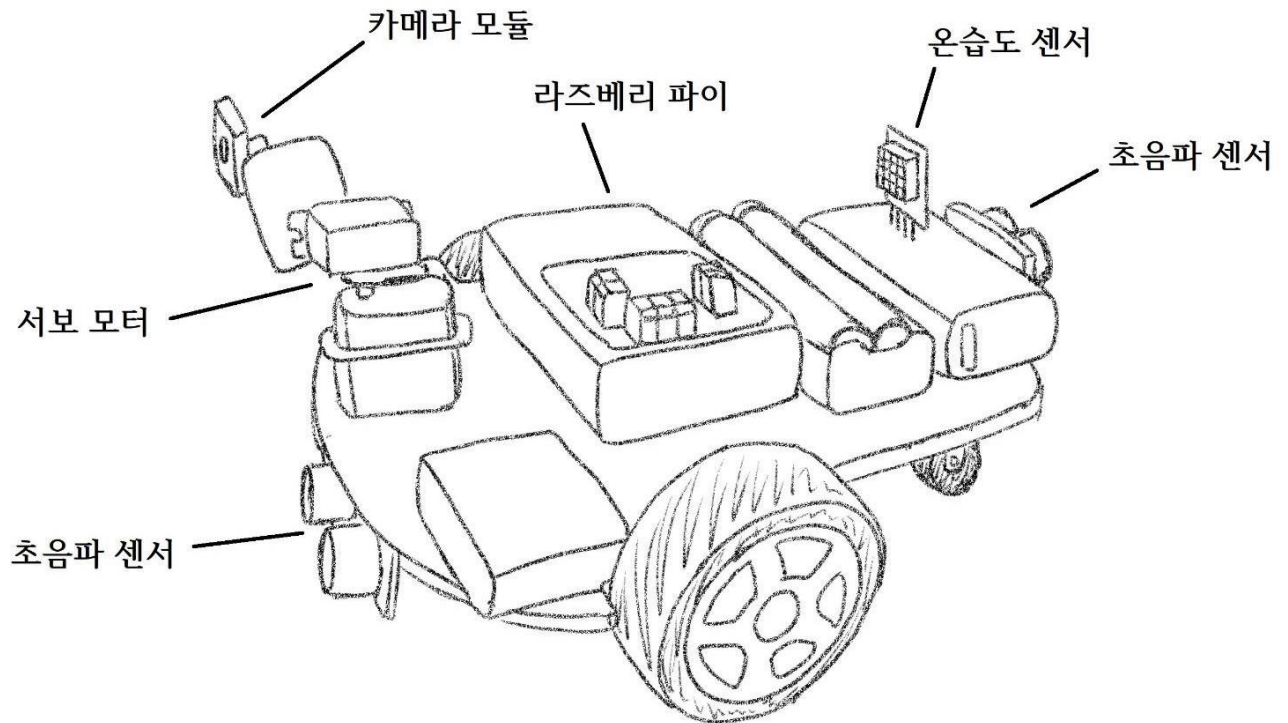
 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

3 작품 구성

3.1 작품의 형상



 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09



3.2 작품의 대략적인 구성

저희 팀이 제작한 드론은 라즈베리 파이로 제어되며 라즈베리 파이의 Access Point 기능을 이용하여 신호를 발생시켜 PC와 연결됩니다. PC에는 C#언어를 사용하여 작성된 드론과 통신하는 전용 프로그램이 설치되어 있습니다. 드론과의 통신은 TCP와 UDP를 이용하며 해당 프로그램을 통해 드론을 조종하며 실시간 화상과 UI를 통해 각 종 센서값을 표시합니다.

드론은 수평으로 배치되어 있는 독립적인 동력을 가진 두 개의 바퀴와 그 두 바퀴와 삼각형 꼴을 이루는 한 개의 동력이 없는 베어링이 장착되어 회전이 가능한 자세 유지용 보조 바퀴로 구성되어 있습니다. 수평으로 배치되어 있는 두 개의 바퀴가 같은 방향으로 구동되어 전진과 후진을 하고 서로 다른 방향으로 구동되어 좌, 우로 선회합니다. 드론의 주행 방향에는 카메라가 장착되어 시야를 확보하는데 해당 카메라는 지표면과 수평하게 장착된 300° 서보 모터와 수직으로 구동하게 장착된 180° 서보 모터에 장착되어 좌우 시야각 약 300°와 위아래 시야각 약 180°를 확보하였습니다.

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

드론의 주행 방향 기준 침단의 하단에는 실시간으로 거리를 측정하는 초음파 센서가 부착되어 드론의 카메라가 전방을 주시하지 않고 있을 때 드론의 전방에 무엇인가 접근하는 것을 알려줄 뿐 아니라 드론의 전고보다 높아 극복하기 힘든 장애물을 미리 파악할 수 있게 해 줍니다. 드론의 후방에도 전방과 마찬가지로 대칭되게 초음파 센서가 부착되어 있어 후방에서 접근하는 물체를 탐지할 수 있습니다. 각 센서는 실시간으로 거리값을 표시해 줄 뿐 아니라 거리가 10cm이하로 측정될 때 드론 조종 프로그램의 UI 색이 변하여 효과적으로 상황을 전달해 줍니다.

드론의 상단에는 온습도 센서가 장착되어 온도와 습도를 측정하여 실시간으로 드론 주변의 온습도값을 표시해 줄 뿐 아니라 온도가 50° 이상이거나 습도가 80% 이상이 면 드론 조종 프로그램의 UI 색이 변하여 효과적으로 상황을 전달해 줍니다.

또한 실시간 이미지 프로세싱을 지원하는 오픈소스 라이브러리인 OpenCV의 딥러닝 기능을 이용해 영상을 분석하여 사물의 종류와 사람을 구분하는 YOLO v4의 MS COCO 데이터 세트를 사용하여 사람을 포함해 80가지의 사물을 구분해 냅니다.

실시간으로 드론이 전송해 주는 영상에 등장하는 사물을 인식해서 표시하고 만일 사람이 인식되면 눈에 띄는 UI로 표시하게 디자인하여 시인성 효과를 높였습니다.

저희 팀이 제작한 드론은 크게 구동부 / 센서부 / 시계부 / 전원부 / 제어 / 통신 총 여섯 부분으로 나눌 수 있습니다.

3.3 구동부

드론의 구동부는 DC모터와 모터드라이버 그리고 기어박스로 구성됩니다. DC모터는 드론의 동력원으로 사용되어 주행을 담당합니다. 모터드라이버는 DC모터를 원하는 방향과 속도로 제어해주는 기능을 합니다. 기어박스는 DC모터에 장착되어 기어비를 변환시켜 회전수를 줄이는 대신 높은 토크를 구현합니다.

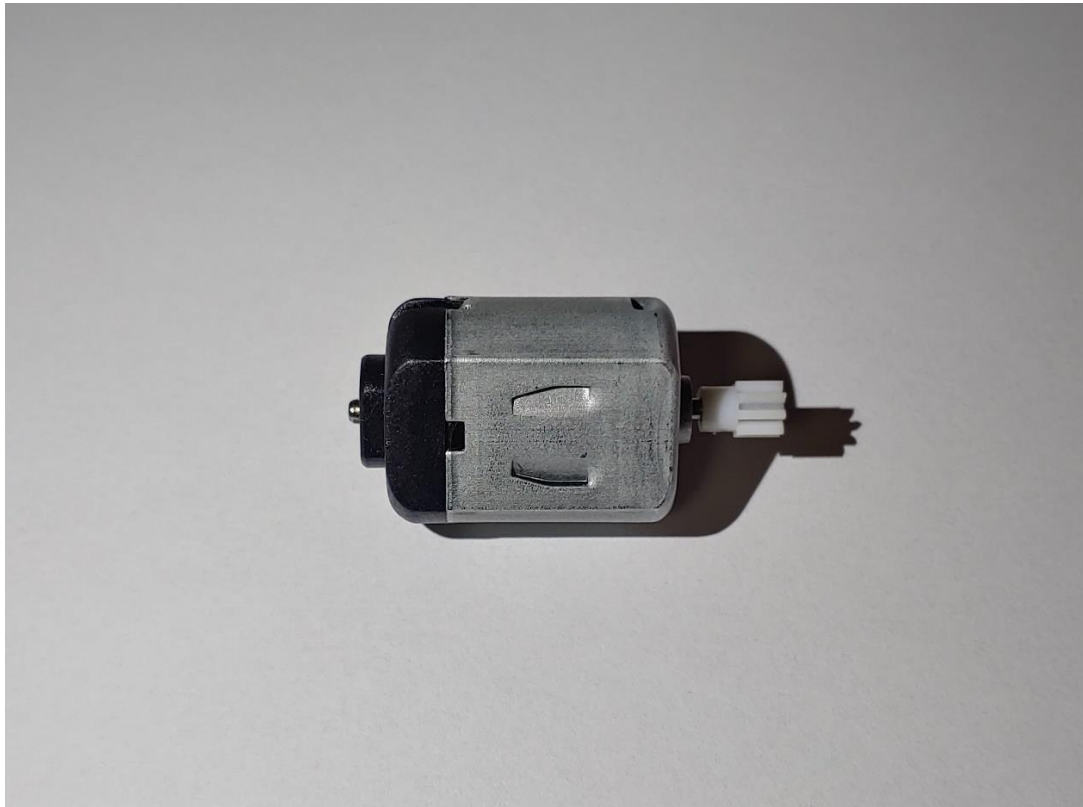
저희 팀에서는 위와 같은 구성으로 구동부를 구현하였으며 목표한 바대로 자유롭게 구동이 가능하였습니다.

3.3.1 DC모터

DC모터는 고정자로 영구자석을 사용하고 회전자로 코일을 사용하여 구성된 장치로, 인

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

가되는 전류의 방향을 전환함으로써 자력의 반발과 흡인력으로 회전력을 생성시킵니다. DC 모터는 기동 토크가 크고 공급되는 전압과 전류의 크기 / 세기에 따라 회전특성과 토크가 직선적으로 비례하여 사용하기 편리하고 출력 효율이 양호한 장점을 가집니다. 저희 팀에서는 작동범위 전압 3.0V ~ 6.0V를 가지고 전류 1.1A 의 DC 모터를 기어박스에 연결하여 사용합니다. 아래는 실제로 사용한 DC모터의 사진입니다.



<DC모터>

3.3.2 모터드라이버

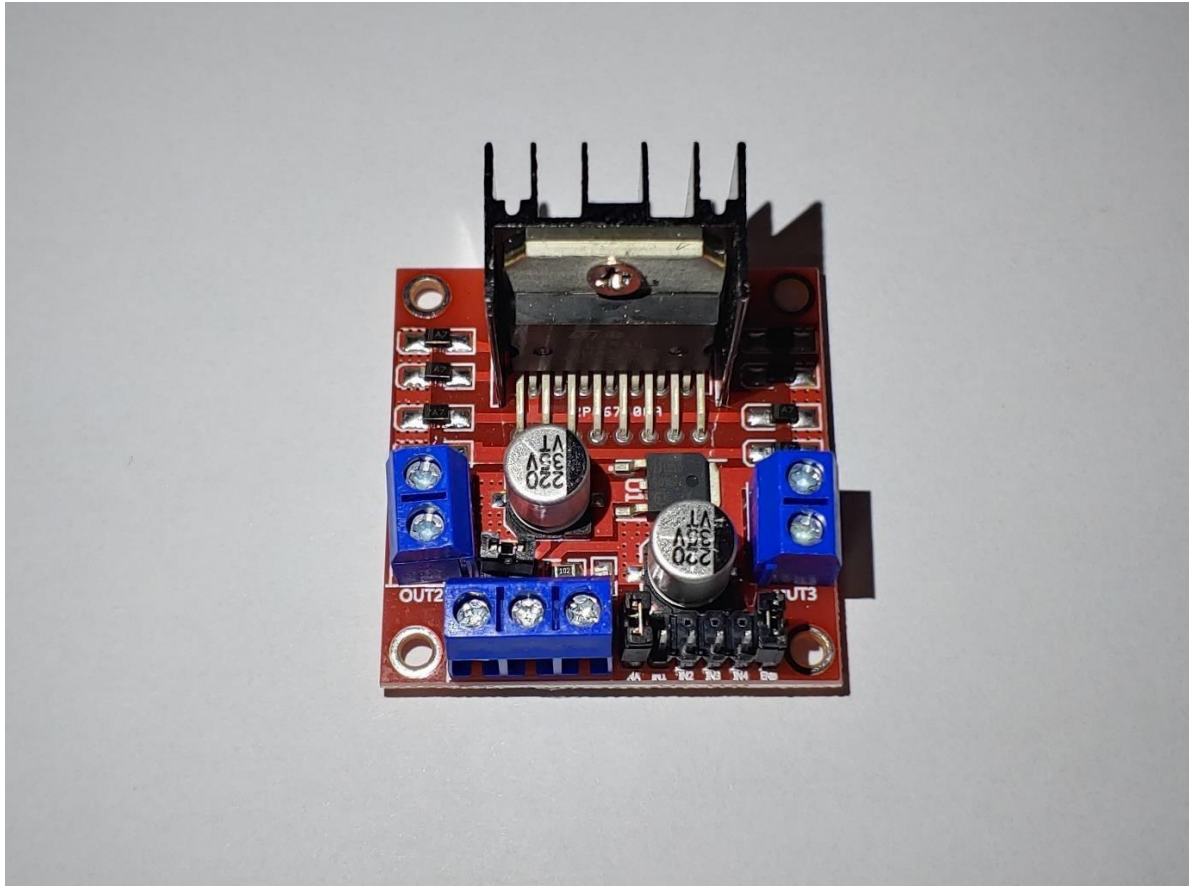
모터 드라이버는 DC모터의 속도와 회전방향을 제어할 수 있는 회로 장치입니다.

DC 모터는 2개의 연결선을 가지며 VCC/GND가 어느 쪽에 연결되는가에 따라 정회전 혹은 역회전이 가능합니다. 또한 DC모터는 보통 인가되는 전압 크기에 따라 회전속도가 비례적으로 변화합니다. 모터 드라이버에서 모터 방향은 각 채널의 드라이버에 HIGH 또는 LOW 신호를 보내 제어됩니다.

IN 1, 2, 3, 4 핀을 통해 방향이 제어되고 PWM 출력을 모터드라이버의 ENA, ENB 입력핀에 연결하여 회전수가 제어됩니다. 저희 팀에서는 최대 46V, 연속 2A의 출력을 낼 수 있는 25W 등급의 L298n 칩이 사용된 H 브리지 모터드라이버를 사용합니다. 아래는 실제로 사

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

용한 모터 드라이버의 사진입니다.



<L298n 모터드라이버>

3.3.3 기어박스

기어박스(감속기)는 DC모터에 부착되어 헬리컬 나사를 이용해 내장된 기어의 이빨 수의 차이를 통해 DC모터의 회전 토크를 강화시키는데 사용됩니다. 기어비가 클수록 회전수는 적어지지만 회전 토크는 강해집니다. 제품에 사용할 기어박스의 후보군으로 각각 1 : 48, 1 : 220, 1 : 288 의 기어비를 가진 세 종류의 기어박스가 정해졌습니다.



MOD EL	VOLTAGE		NO LOAD		AT MAXIMUM EFFICIENCY				STALL			
	OPERATING RANGE	NOMI	SPEE	CURRE	SPEE	CURRE	TORQUE		OUTP	TORQUE	CURRE	
		NA	D	NT	D	NT			UT		NT	
		V	r/mi n	A	r/mi n	A	N. m	Kg.c m	w	N.m	Kg.c m	A
NP01 D -48	3.0 - 9.0	3.0	110	0.12	85	0.20	0.0 1	0.11	0.13	0.02 7	0.27	1.45

<1 : 48 기어비 기어박스 데이터 시트>

MOD EL	VOLTAGE		NO LOAD		AT MAXIMUM EFFICIENCY				STALL			
	OPERATING RA NGE	NOMI	SPEE	CURRE	SPEE	CURRE	TORQUE	OUTP UT	TORQUE	CURRE		
		NA	D	NT	D	NT						
		V	r/mi n	A	r/mi n	A					N. Kg.c m m	N. Kg.c m m
NP01 S -220	3.0 - 6.0	3.0	50	0.25	34	0.45	0.07	0.71	0.52	0.24	2.4	1.1

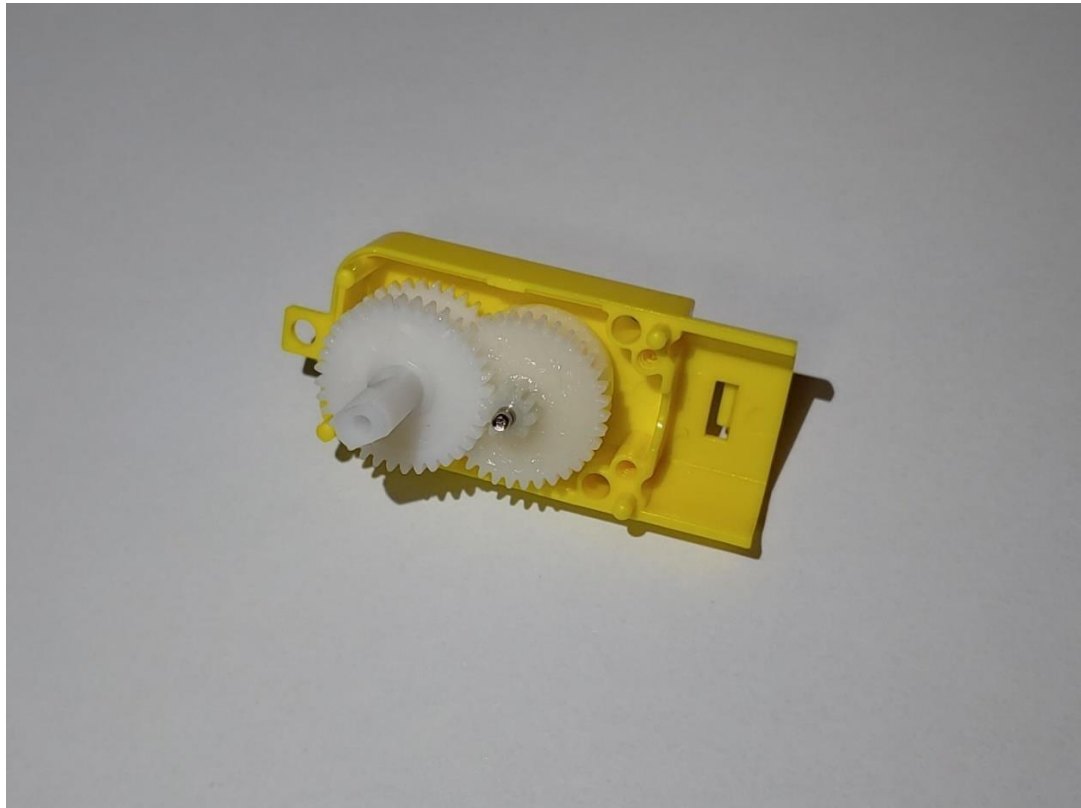
<1 : 220 기어비 기어박스 데이터 시트>

MODEL	VOLTAGE		NO LOAD		AT MAXIMUM EFFICIENCY				STALL			
	OPERATING RANGE	NOMINAL V	SPEED r/min	CURRENT A	SPEED r/min	CURRENT A	TORQUE		OUTPUT W	TORQUE		CURRENT A
							N. m	Kg.cm		N. m	Kg.cm	
NP02 D-288	3.0 - 12.0	6	21	0.10	19	0.17	0.05	0.47	0.18	0.18	1.8	0.6

<1 : 288 기어비 기어박스 데이터 시트>

저희 팀에서는 드론의 무게가 약 1kg정도 되는 비교적 무거운 무게를 가지고 있고 목적의 특성상 빠른 속도보다는 높은 토크를 우선시 해야 한다고 결론을 내고 2.4Kg.cm의 토크를 낼 수 있는 1 : 220 의 기어비를 가지는 기어박스를 채택하여 사용하였습니다. 아래는 실제로 사용한 기어박스의 사진입니다.

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09



<1 : 220 기어박스>

3.3.4 구동부 동작 코드

DC모터 동작 코드	
<pre># -*- coding: utf8 -*- import RPi.GPIO as GPIO #### 모터 상태 STOP = 0 FORWARD = 1 BACKWARD = 2 class DCMotor: global STOP, FORWARD, BACKWARD def _init_(self, EN, INA, INB): self.EN = EN self.INA = INA</pre>	



```
self.INB = INB
```

```
GPIO.setup(self.EN, GPIO.OUT, initial=GPIO.LOW)
```

```
GPIO.setup(self.INA, GPIO.OUT, initial=GPIO.LOW)
```

```
GPIO.setup(self.INB, GPIO.OUT, initial=GPIO.LOW)
```

```
# 100khz 로 PWM 동작 시킴
```

```
self.motor=GPIO.PWM(self.EN, 100)
```

```
self.motor.start(0)
```

```
def _setMotorContorl(self, PWM, INA, INB, speed, stat):
```

```
#모터 속도 제어 PWM
```

```
self.motor.ChangeDutyCycle(speed)
```

```
if stat == FORWARD:
```

```
    GPIO.output(INA, GPIO.HIGH)
```

```
    GPIO.output(INB, GPIO.LOW)
```

```
elif stat == BACKWARD:
```

```
    GPIO.output(INA, GPIO.LOW)
```

```
    GPIO.output(INB, GPIO.HIGH)
```

```
elif stat == STOP:
```

```
    GPIO.output(INA, GPIO.LOW)
```

```
    GPIO.output(INB, GPIO.LOW)
```

```
# 모터 제어함수 간단하게 사용하기 위해 한번더 래핑(감쌈)
```

```
def setMotor(self, speed, stat):
```

```
    self._setMotorContorl(self.EN, self.INA, self.INB, speed, stat)
```

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

3.4 센서부

드론의 센서부는 초음파 센서와 온습도 센서로 구성됩니다. 초음파 센서는 드론 전방 하단과 드론의 후방에 부착됩니다. 온습도 센서는 드론의 상단에 장착됩니다.

전방 하단에 배치된 초음파 센서는 드론의 전고로 극복하기 힘든 장애물을 판단할 수 있게 해 줍니다. 후방에 배치된 초음파 센서는 드론의 사각지대에 장애물이 존재하는지 유무를 파악할 수 있게 해 줍니다.

온습도 센서는 드론 주변의 온도와 습도를 파악할 수 있게 해 주어 현장 내부의 상황은 물론 드론이 동작할 수 있는 한계 환경을 미리 파악하여 드론의 생존성을 높일 수 있습니다.

저희 팀에서는 위와 같이 센서부를 구성하였고 목표한 바대로 초음파 센서의 동작과 온습도 센서의 동작이 오차 범위 내에서 정상적으로 작동하는 것을 확인하였습니다.

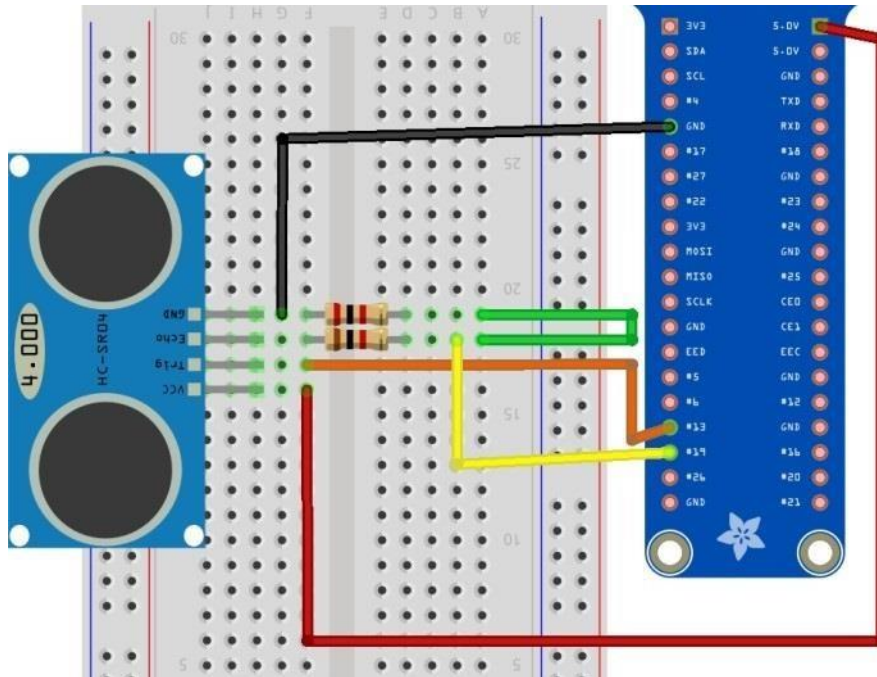
3.4.1 초음파 센서

초음파는 인간이 들을 수 있는 소리의 주파수 영역보다 더 높은 주파수를 가진 사운드 펄스를 말합니다. 초음파 센서의 Trigger부에서 전방으로 높은 주파수 사운드 펄스를 짧게 여러 번 발사합니다. 이 사운드 펄스는 음속으로 전파되어 어떠한 물체에 충돌한 뒤 다시 초음파 센서로 반사되어 돌아옵니다. 이 반사된 사운드 펄스를 초음파 센서의 echo부에서 수신하여 이에 대한 시간차를 기반으로 거리를 산출해 냅니다.

신호가 되돌아 올때까지 걸리는 시간 t 는 다음 공식으로 산출해 낼 수 있습니다.

$$t(s) = 2 * L(\text{물체와의 거리 } m) / V_s(\text{음속})$$

저희 팀에서는 거리 최소 2cm, 최대 450cm, 각도 최대 15°를 측정 가능한 초음파 센서인 HC-SR04 모델을 사용합니다. 또한 HC-SR04 초음파 센서는 출력값을 5v로 출력해 주게 되는데 라즈베리 파이의 입력 GPIO 단자는 최대 3.3V까지 입력이 허용되며 그 값을 초과하는 전압이 입력되면 손상의 위험이 있습니다. 따라서 5V를 3.3V 근처로 전압을 낮춰주는 회로를 구성하여야 합니다. 아래는 해당 회로의 회로도입니다.



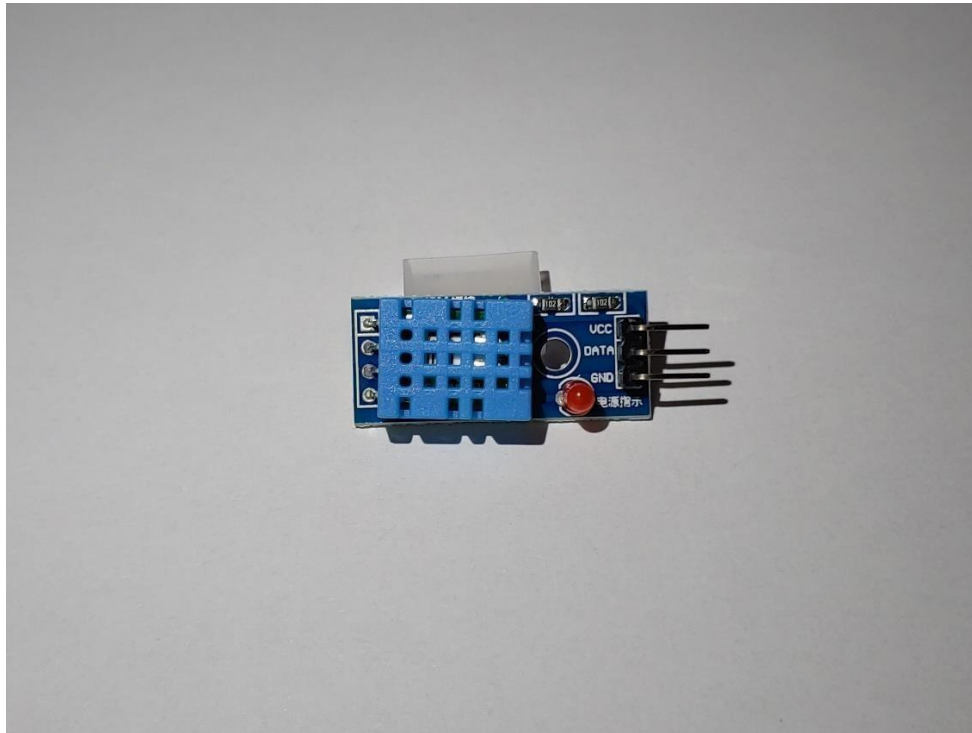
<초음파 센서 전압 강하 회로>

3.4.2 온습도 센서

온습도 센서는 현재 대기 중의 온도와 습도를 측정할 수 있는 센서입니다. 온도센서는 온도에 따라 물질의 저항값이 변하는 소재를 사용하여 저항값의 변화를 감지해 온도를 측정합니다. 습도센서는 습도에 따라 두 전극 사이의 저항값이 변하는 특성을 이용하여 공기중의 습도 변화를 알아내고, 표면에 부착된 수분의 양에 의해 전극의 전도 도에 변화가 일어나게 되면 이를 감지해서 습도를 측정합니다.

저희 팀에서는 습도 측정 범위 최소 20%RH, 최대 90%RH, 오차 $\pm 5\%RH$, 온도 측정 최소 $0^{\circ}C$, 최대 $50^{\circ}C$, 오차 $\pm 2^{\circ}C$ 를 가진 온습도 센서인 DHT-11 모델을 사용합니다. 아래는 실제로 사용한 온습도 센서의 사진입니다.

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09



<DHT-11 온습도 센서>

3.4.3 센서부 동작 코드

초음파 센서 동작 코드	
<pre># -*- coding: utf8 -*- import RPi.GPIO as GPIO import time from multiprocessing import Process, Value #### 초음파 센서 #거리 타임 아웃 용 MAX_DISTANCE_CM = 300 MAX_DURATION_TIMEOUT = (MAX_DISTANCE_CM * 2 * 29.1) #17460 # 17460us = 300cm class Ultrasonic: def __init__(self, TriggerPin, EchoPin): self.distance = Value('d', 0) self.TrigerPin = TriggerPin self.EchoPin = EchoPin</pre>	



```
GPIO.setup(self.EchoPin, GPIO.IN)
GPIO.setup(self.TrigerPin, GPIO.OUT, initial=GPIO.LOW)

p1 = Process(target=self.MeasureDistance, args=(self.distance, ))
p1.daemon = True
p1.start()

def MeasureDistance(self, distance):
    while True:
        self.fail = False
        time.sleep(0.1)

        # 트리거를 10us 동안 High 했다가 Low로 함.
        GPIO.output(self.TrigerPin, GPIO.HIGH)
        time.sleep(0.00001)
        GPIO.output(self.TrigerPin, GPIO.LOW)

        # ECHO로 신호가 들어 올때까지 대기
        self.timeout = time.time()
        while GPIO.input(self.EchoPin) == GPIO.LOW:
            #들어왔으면 시작 시간을 변수에 저장
            self.pulse_start = time.time()
            if ((self.pulse_start - self.timeout)*1000000) >= MAX_DURATION_TIMEOUT:
                #171206 중간에 통신 안되는 문제 개선용
                self.fail = True
                break

        if self.fail:
            continue

        #ECHO로 인식 종료 시점까지 대기
        while GPIO.input(self.EchoPin) == GPIO.HIGH:
            #종료 시간 변수에 저장
            self.pulse_end = time.time()
            if ((self.pulse_end - self.pulse_start)*1000000) >= MAX_DURATION_TIMEOUT:
                #171206 중간에 통신 안되는 문제 개선용
                self.fail = True
                break

        if self.fail:
```



continue

```
#인식 시작부터 종료까지의 차가 바로 거리 인식 시간
self.pulse_duration = (self.pulse_end - self.pulse_start) * 1000000
```

```
# 시간을 cm로 환산
self.distance = (self.pulse_duration/2)/29.1
distance.value = int(self.distance)
```

```
def getDistance(self):
    return self.distance.value
```

온습도 센서 동작 코드

```
# -*- coding: utf8 -*-
import RPi.GPIO as GPIO
import time

class DHT11Result:
    'DHT11 sensor result returned by DHT11.read() method'

    ERR_NO_ERROR = 0
    ERR_MISSING_DATA = 1
    ERR_CRC = 2

    error_code = ERR_NO_ERROR
    temperature = -1
    humidity = -1

    def __init__(self, error_code, temperature, humidity):
        self.error_code = error_code
        self.temperature = temperature
        self.humidity = humidity

    def is_valid(self):
        return self.error_code == DHT11Result.ERR_NO_ERROR

class DHT11:
    'DHT11 sensor reader class for Raspberry'
```



```
__pin = 0

def __init__(self, pin):
    self.__pin = pin

def read(self):
    GPIO.setup(self.__pin, GPIO.OUT)

    # send initial high
    self._send_and_sleep(GPIO.HIGH, 0.05)

    # pull down to low
    self._send_and_sleep(GPIO.LOW, 0.02)

    # change to input using pull up
    GPIO.setup(self.__pin, GPIO.IN, GPIO.PUD_UP)

    # collect data into an array
    data = self._collect_input()

    # parse lengths of all data pull up periods
    pull_up_lengths = self._parse_data_pull_up_lengths(data)

    # if bit count mismatch, return error (4 byte data + 1 byte checksum)
    if len(pull_up_lengths) != 40:
        return DHT11Result(DHT11Result.ERR_MISSING_DATA, 0, 0)

    # calculate bits from lengths of the pull up periods
    bits = self._calculate_bits(pull_up_lengths)

    # we have the bits, calculate bytes
    the_bytes = self._bits_to_bytes(bits)

    # calculate checksum and check
    checksum = self._calculate_checksum(the_bytes)
    if the_bytes[4] != checksum:
        return DHT11Result(DHT11Result.ERR_CRC, 0, 0)

    # ok, we have valid data
```



```
# The meaning of the return sensor values
# the_bytes[0]: humidity int
# the_bytes[1]: humidity decimal
# the_bytes[2]: temperature int
# the_bytes[3]: temperature decimal

temperature = the_bytes[2] + float(the_bytes[3]) / 10
humidity = the_bytes[0] + float(the_bytes[1]) / 10

return DHT11Result(DHT11Result.ERR_NO_ERROR, temperature, humidity)

def __send_and_sleep(self, output, sleep):
    GPIO.output(self.__pin, output)
    time.sleep(sleep)

def __collect_input(self):
    # collect the data while unchanged found
    unchanged_count = 0

    # this is used to determine where is the end of the data
    max_unchanged_count = 100

    last = -1
    data = []
    while True:
        current = GPIO.input(self.__pin)
        data.append(current)
        if last != current:
            unchanged_count = 0
            last = current
        else:
            unchanged_count += 1
            if unchanged_count > max_unchanged_count:
                break

    return data

def __parse_data_pull_up_lengths(self, data):
    STATE_INIT_PULL_DOWN = 1
    STATE_INIT_PULL_UP = 2
```



```
STATE_DATA_FIRST_PULL_DOWN = 3
STATE_DATA_PULL_UP = 4
STATE_DATA_PULL_DOWN = 5

state = STATE_INIT_PULL_DOWN

lengths = [] # will contain the lengths of data pull up periods
current_length = 0 # will contain the length of the previous period

for i in range(len(data)):

    current = data[i]
    current_length += 1

    if state == STATE_INIT_PULL_DOWN:
        if current == GPIO.LOW:
            # ok, we got the initial pull down
            state = STATE_INIT_PULL_UP
            continue
        else:
            continue
    if state == STATE_INIT_PULL_UP:
        if current == GPIO.HIGH:
            # ok, we got the initial pull up
            state = STATE_DATA_FIRST_PULL_DOWN
            continue
        else:
            continue
    if state == STATE_DATA_FIRST_PULL_DOWN:
        if current == GPIO.LOW:
            # we have the initial pull down, the next will be the data pull up
            state = STATE_DATA_PULL_UP
            continue
        else:
            continue
    if state == STATE_DATA_PULL_UP:
        if current == GPIO.HIGH:
            # data pulled up, the length of this pull up will determine whether it is 0 or 1
            current_length = 0
            state = STATE_DATA_PULL_DOWN
```



```
        continue
    else:
        continue
    if state == STATE_DATA_PULL_DOWN:
        if current == GPIO.LOW:
            # pulled down, we store the length of the previous pull up period
            lengths.append(current_length)
            state = STATE_DATA_PULL_UP
            continue
        else:
            continue

    return lengths

def __calculate_bits(self, pull_up_lengths):
    # find shortest and longest period
    shortest_pull_up = 1000
    longest_pull_up = 0

    for i in range(0, len(pull_up_lengths)):
        length = pull_up_lengths[i]
        if length < shortest_pull_up:
            shortest_pull_up = length
        if length > longest_pull_up:
            longest_pull_up = length

    # use the halfway to determine whether the period it is long or short
    halfway = shortest_pull_up + (longest_pull_up - shortest_pull_up) / 2
    bits = []

    for i in range(0, len(pull_up_lengths)):
        bit = False
        if pull_up_lengths[i] > halfway:
            bit = True
        bits.append(bit)

    return bits

def __bits_to_bytes(self, bits):
    the_bytes = []
```



```
byte = 0

for i in range(0, len(bits)):
    byte = byte << 1
    if (bits[i]):
        byte = byte | 1
    else:
        byte = byte | 0
    if ((i + 1) % 8 == 0):
        the_bytes.append(byte)
        byte = 0

return the_bytes

def __calculate_checksum(self, the_bytes):
    return the_bytes[0] + the_bytes[1] + the_bytes[2] + the_bytes[3] & 255
```

3.5 시계부

드론의 시계부는 파이 카메라와 두 종류의 서보 모터로 이루어집니다. 파이 카메라는 라즈베리 파이와 호환되는 전용 카메라입니다. 서보 모터에 부착되어 시야를 확보합니다. 서보 모터는 180° 와 300°의 구동 범위를 가진 두 종류를 사용합니다.

비교적 큰 크기와 넓은 가동범위를 가진 300° 서보 모터가 드론의 전방에 시야와 수평하게 장착되어 서보 모터가 가동할 경우 좌우로 시야 방향이 조절됩니다. 비교적 작은 크기와 좁은 가동범위를 가진 180° 서보 모터는 파이 카메라를 장착한 채로 300° 서보 모터 위에 수직으로 장착됩니다. 따라서 180° 서보 모터가 가동할 경우 상하로 시야 방향이 조절됩니다.

저희 팀에서는 위와 같이 시계부를 구현하였으며 드론 차체의 물리적 한계로 실제 각 서보 모터의 300°와 180°의 가동 범위를 온전히 확보하지는 못하였지만 주행에 필요한 전방의 시야 확보는 충분히 가능 할 정도의 시야 범위를 구현하였으며 실시간 영상 전송과 영상 분석을 성공시켰습니다.

3.5.1 파이 카메라

파이 카메라(Pi Camera)는 라즈베리 파이 전용으로 맞춤 설계된 카메라 모듈입니다. 라즈베리 파이 보드 상단에 있는 15p FFC 소켓에 리본케이블로 연결됩니다. 파이

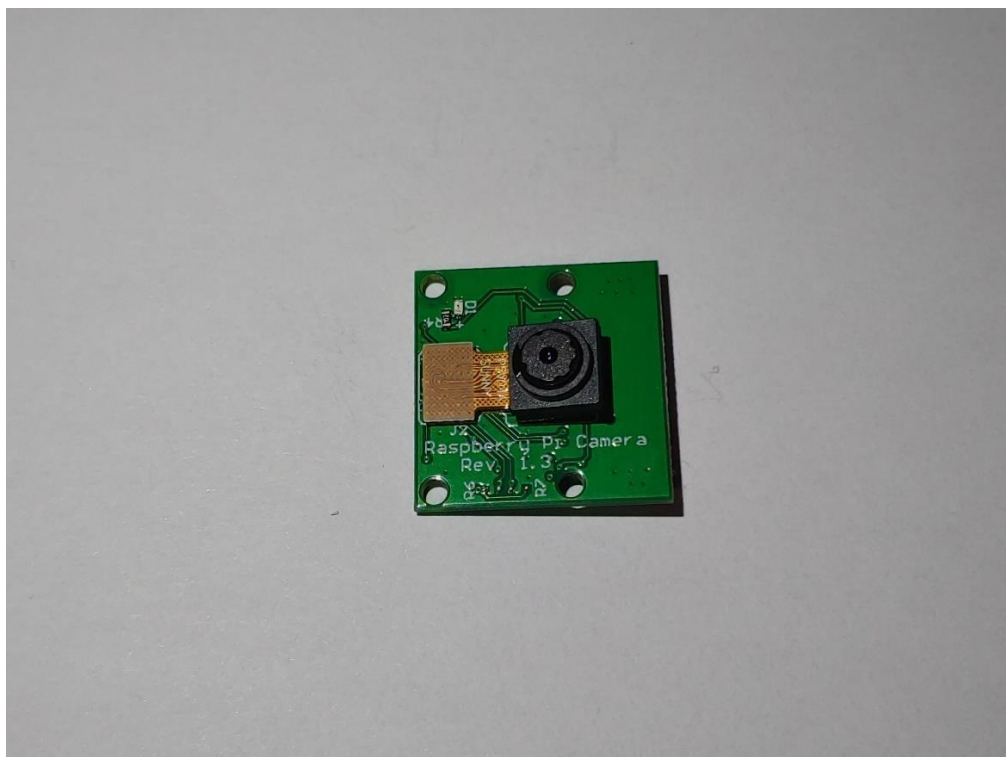
 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

카메라는 특별히 전용 설계된 CSI 인터페이스가 사용되는데, 이 CSI 버스는 픽셀 데이터만 전송하지만 매우 높은 데이터 전송 속도를 제공합니다.

또한 파이 카메라의 영상 전송을 위해 mjpg-streamer를 사용합니다. mjpg-streamer는 CPU와 RAM이 매우 제한된 리소스를 가진 임베디드 장치를 위해 개발되었습니다. 하나 이상의 입력 플러그인의 JPEG 프레임을 여러 출력 플러그인으로 복사하는 응용 프로그램입니다. 웹캠에서 IP기반 네트워크를 통해 MJPG stream을 수신할 수 있는 여러 소프트웨어를 뷰어로 사용할 수 있습니다.

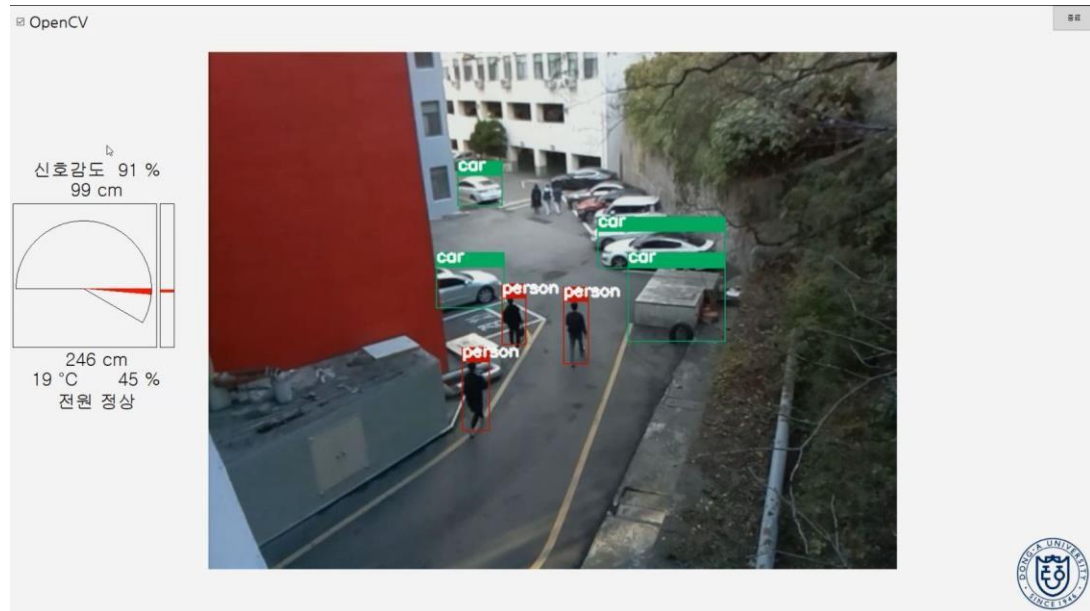
저희 팀에서는 화소 5MP, 해상도 2952 x 1944, 이미지 센서 크기 1/4 inch, 화각 72도를 가진 YR-020 모델을 사용합니다.

또한 mjpg-streamer에서 스트리밍 해 준 영상을 자체 제작한 DCS 프로그램에서 http를 이용해 재생합니다. 스트리밍되는 해당 영상을 실시간 이미지 프로세싱을 지원하는 오픈소스 라이브러리인 OpenCV의 딥러닝 기능을 이용해 영상을 분석하여 사람을 포함한 최대 80가지의 사물을 구분하는 YOLO v4 라이브러리를 사용하여 영상에 등장하는 사물과 사람을 표시합니다. 아래는 실제 사용한 파이 카메라의 사진과 분석된 영상의 이미지입니다.



<YR-020 라즈베리 파이 호환 카메라>

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09



<Open CV를 이용한 영상 분석>

3.5.2 서보 모터

서보 모터는 위치를 제어하도록 만들어진 모터입니다. 서보 모터의 내부에는 DC모터, 위치를 측정하는 가변 저항, 증폭기를 통한 위치 제어 회로가 포함되어 있습니다. 서보 모터는 DC모터에 비하여 작동 각도를 미세하게 조절 가능한 것이 특징이며 물체의 위치, 방위 자세, 회전 속도 등을 제어하는데 주로 사용됩니다. 서보 모터는 모터 드라이버가 내장되어 있어 DC모터보다 가격이 비싸지만 구성과 사용이 비교적 간단한 특징이 있습니다.

저희 팀에서는 동작 전압 4.8V, 회전속도 0.1sec/60degree, 토크 1.8kg·cm, 최대 동작 범위 180°를 가진 SG-90 서보 모터와 동작 전압 4.8V, 토크 1.6kg·cm, 최대 동작 범위 300°±10°를 가진 DS-R005 서보 모터를 사용합니다.

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09



<SG-90 서보 모터>



<DS-R005 서보 모터>

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

3.5.3 서보 모터 동작 코드

서보 모터 동작 코드
<pre> # -*- coding: utf8 -*- import RPi.GPIO as GPIO import threading, time SG_SERVO_MAX_DUTY = 11 # 서보의 최대(180도) 위치의 주기 SG_SERVO_MIN_DUTY = 3 # 서보의 최소(0도) 위치의 주기 DR_SERVO_MAX_DUTY = 12 # 서보의 최대(300도) 위치의 주기 DR_SERVO_MIN_DUTY = 2 # 서보의 최소(0도) 위치의 주기 class ServoMotor: global SERVO_MAX_DUTY, SERVO_MIN_DUTY def __init__(self, PIN, TYPE): self.PIN = PIN self.TYPE = TYPE self.TIMER = 0.0 GPIO.setup(self.PIN, GPIO.OUT) if self.TYPE == 0: self.servo = GPIO.PWM(self.PIN, 50) # 180도 50HZ else: self.servo = GPIO.PWM(self.PIN, 50) # 300도 self.servo.start(0) stimer = threading.Thread(target=self. deAttached, args=()) stimer.daemon = True stimer.start() def deAttached(self): while True: if time.time() - self.TIMER > 0.5: GPIO.setup(self.PIN, GPIO.IN) time.sleep(0.5) def Set(self, degree): GPIO.setup(self.PIN, GPIO.OUT) </pre>

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

```

if self.TYPE == 0:
    # 각도는 180도를 넘을 수 없다.
    if degree > 180:
        degree = 180

    # 각도(degree)를 duty로 변경한다.
    duty = SG_SERVO_MIN_DUTY+(degree*(SG_SERVO_MAX_DUTY-
    SG_SERVO_MIN_DUTY)/180.0)
    # duty 값 출력
    print("Degree: {} to {}(Duty)".format(degree, duty))

    # 변경된 duty값을 서보 pwm에 적용
    self.servo.ChangeDutyCycle(duty)

elif self.TYPE == 1:
    if degree > 300:
        degree = 300

    duty = DR_SERVO_MIN_DUTY+(degree*(DR_SERVO_MAX_DUTY-
    DR_SERVO_MIN_DUTY)/300.0)
    print("Degree: {} to {}(Duty)".format(degree, duty))
    self.servo.ChangeDutyCycle(duty)

self.TIMER = time.time()

```

3.6 전원부

드론의 전원부는 18650 리튬 이온 충전지와 AAA규격 알카라인 전지 그리고 QC 3.0을 지원하는 시판 보조배터리로 구성됩니다. 18650 리튬 이온 충전지는 드론의 구동계인 DC 모터에 사용되고 AAA 알카라인 전지는 서보 모터의 구동에 사용되며 시판 보조배터리는 라즈베리 파이의 구동에 사용됩니다.

저희 팀은 위와 같은 구성으로 드론의 전원부를 구성하여 전력 과다/부족 없이 안정적으로 드론을 구동시키는 것을 성공하였습니다.

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

3.6.1 18650 충전지

드론의 구동을 담당하는 DC모터에는 전원으로 18650 규격을 가진 리튬 이온 충전지 가 사용됩니다. DC모터의 출력 토크는 입력되는 전압과 전류에 의하여 결정됩니다. 저희 팀의 드론 무게가 무거운 바 DC모터 작동 최대의 전압과 전류를 공급해 주고 장기간 사용의 편의성을 위하여 통상적으로 사용되는 AA 규격 알카라인 전지보다 더 안정적이고 강한 전류를 공급해주며 무게도 더 가벼운 18650 규격의 3.7V 2600mAh 리튬 이온 충전지를 사용하게 되었습니다. 해당 충전지를 직렬로 2개 연결하여 출력 전압 7.4V와 출력 전류 2.6A 를 구성하고, DC모터에 병렬로 연결해 주어 전압 7.4V와 전류 1.3A를 각 DC모터에 입력해 줍니다. 저희 팀에서는 출력 전압 3.7V, 충전용량 2600mAh를 가진 HY-2600 모델을 사용합니다. 아래는 실제 사용한 18650 충전지의 사진입니다.

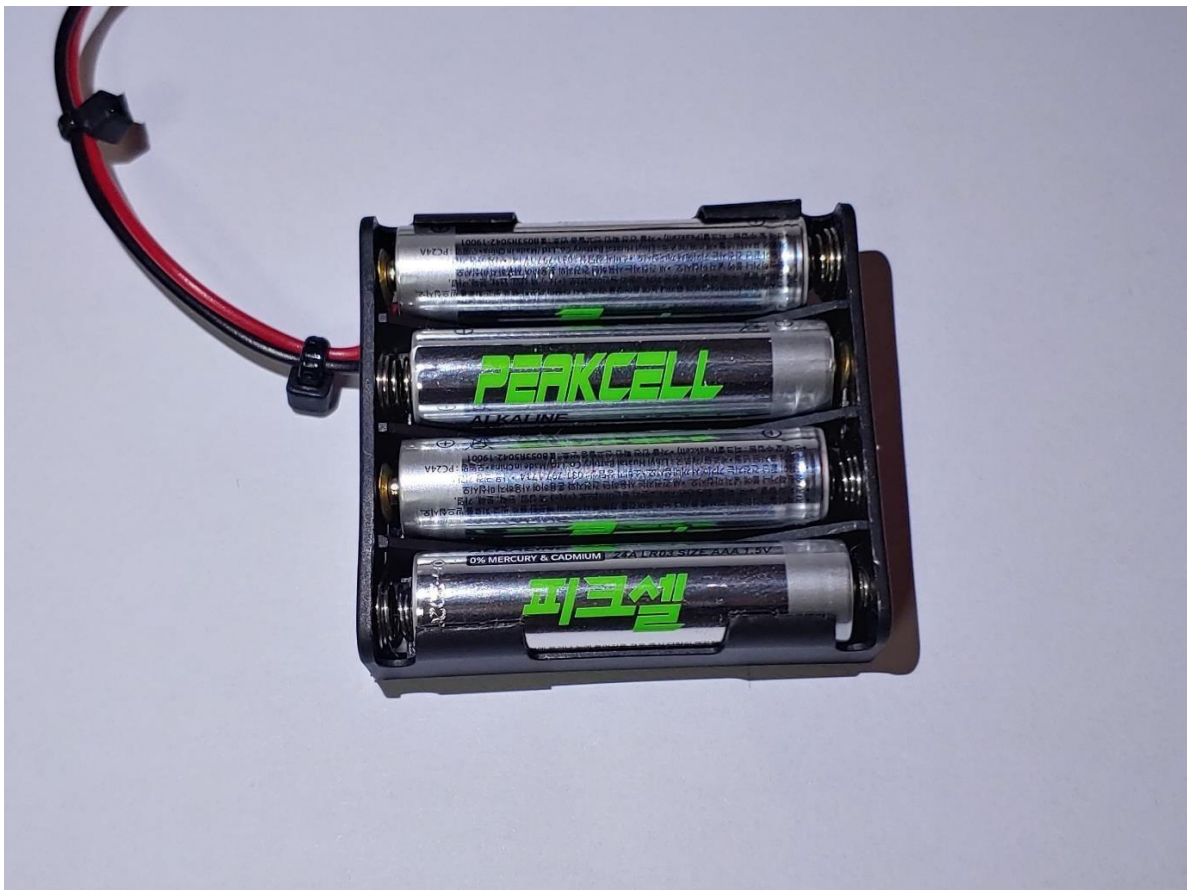


< HY-2600 모델 18650 충전지 >

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

3.6.2 AAA규격 알카라인 전지

카메라의 구동을 담당하는 서보 모터에는 전원으로 AAA 규격을 가진 알카라인 전지가 사용됩니다. 서보 모터는 DC모터에 비하여 비교적 부담이 적기에 드론의 부품 배치와 하중을 고려하여 일반적으로 사용하는 AAA 규격의 1.5V 알카라인 전지를 사용하게 되었습니다. 해당 전지 4개를 직렬로 연결하여 6V를 구성하여 각 서보모터에 별도의 전원으로 연결해 주게 됩니다. 저희 팀에서는 예산을 고려하여 피크셀 알카라인 AAA 모델을 사용합니다. 아래는 실제 사용한 AAA 전지의 모습입니다.



<시판용 AAA 전지>

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

3.6.3 시판 보조배터리

드론의 제어를 총괄하는 라즈베리 파이의 전원으로는 QC3.0을 지원하는 시판 보조 배터리가 사용됩니다. 저희 팀에서 사용하는 라즈베리 파이 4 B+ (이하 라즈베리 파이) 모델은 안정적으로 작동하기 위해서 전압 5V와 전류 3A 이상의 전원 공급을 필요로 합니다. 라즈베리 파이 전용 UPS는 드론에 이용하기에 너무 크고 무거우며 비용도 예산을 초과합니다. 또한 별도로 정류 회로와 리튬 이온 배터리를 이용하여 전원을 구성 하기에는 비용이 너무 많이 드는 문제가 있습니다. 따라서 다른 방법을 찾을 수 밖에 없습니다.

흔히 사용하는 시판 고속 충전기에는 PD(Power Delivery)또는 QC(Quick Charge)기능이 탑재되어 있습니다. 해당 기술은 5V 3A 이상의 전원을 공급해 주는 기술입니다. PD와 QC는 전원을 공급받는 기기에 각각 해당 기술이 적용된 제어 칩이 존재하여 전원을 공급하는 기기와 공급받는 기기의 서로간에 통신을 통해 적절한 전원을 공급하여 줍니다. 라즈베리 파이의 전원부에는 PD와 QC의 제어 칩이 존재하지 않습니다. PD는 쌍방간에 통신이 불가능한 경우 기본으로 설정된 전압을 출력합니다. 만약 기본으로 설정된 전압이 5V를 초과한다면 라즈베리 파이의 손상 가능성이 있습니다. QC는 쌍방간 통신이 불가능한 경우 무조건 5V를 출력합니다. 따라서 3A이상 전송이 가능한 케이블을 이용하면 QC를 통하여 라즈베리 파이에 충분한 전원을 공급해 줄 수 있습니다.

저희 팀에서는 크기와 무게, 예산을 고려하여 10000mAh 용량을 가진 스카이 필 X11 미니 보조 배터리를 사용합니다.

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

3.7 제어부

3.7.1 라즈베리파이 4 B+

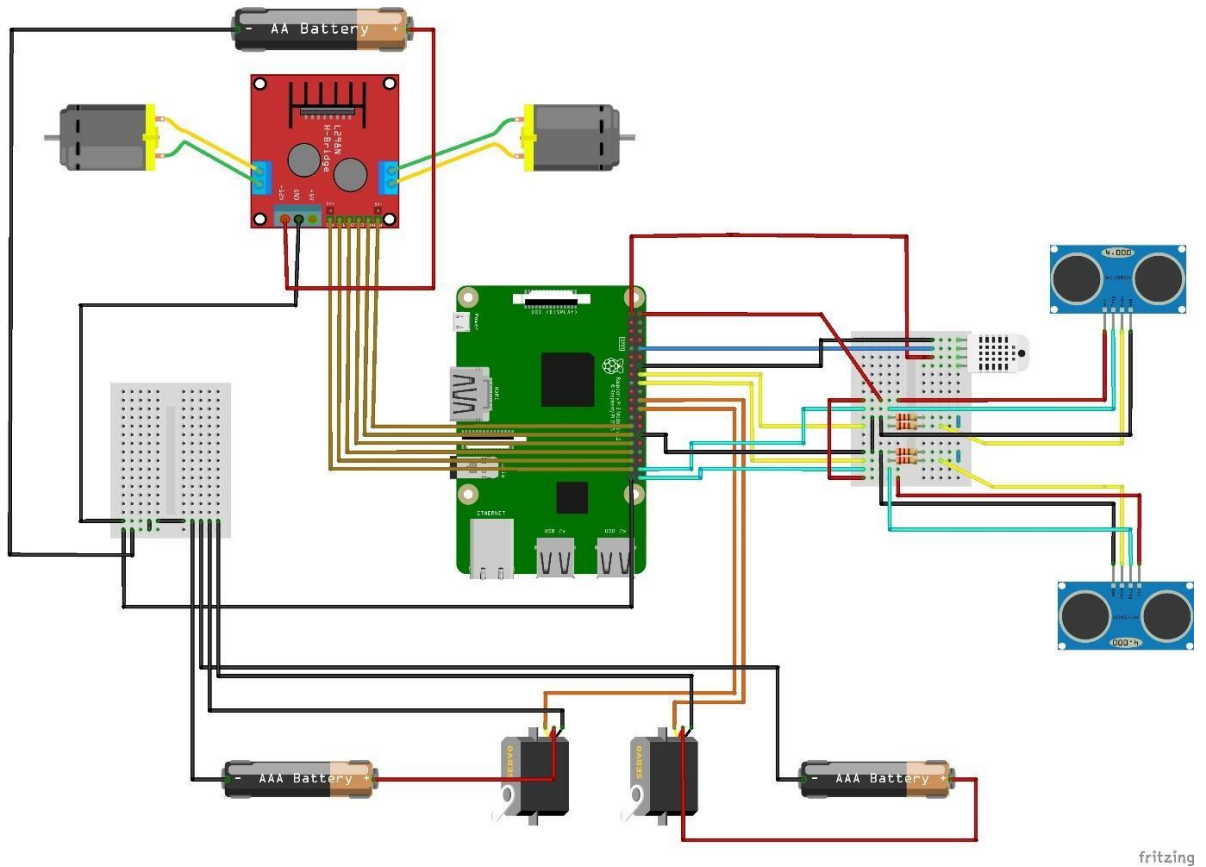
드론의 제어에는 라즈베리 파이 4 B+ (이하 라즈베리 파이) 모델을 사용합니다. 라즈베리 파이(Raspberry Pi)는 영국의 라즈베리 파이 재단이 학교와 개발도상국에서 기초 컴퓨터 과학의 교육을 증진시키기 위하여 개발된 신용카드 크기의 싱글 보드 컴퓨터입니다. 우수한 성능과 저렴한 가격에 리눅스를 포함한 편리한 개발 환경을 가지고 있습니다. 블루투스 및 와이파이 모듈을 내장하고 전용 디스플레이와 전용 카메라 모듈을 비롯하여 USB 입력, HDMI 입력을 지원하고 GPIO핀을 통하여 신호를 출력합니다.

저희 팀에서는 라즈베리 파이에서 작성된 Python 코드로 DC모터, 서보모터, 초음파 센서, 온습도 센서, 파이카메라를 제어하며 라즈베리 파이의 AP기능을 이용해 PC와 통신하여 직접 작성한 프로그램을 통해 드론을 제어합니다. 아래는 실제 사용한 라즈베리 파이 4 B+ 모델의 사진과 앞서 설명한 하드웨어 회로도의 이미지입니다.



<라즈베리파이 4 B+ 모델>

 <div> <div>동아대학교</div> <div>전자공학과</div> <div>캡스톤 디자인</div> </div>	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09



<작품의 하드웨어 회로도>

3.7.2 라즈베리 파이 제어 코드

라즈베리 파이 제어 코드
<pre>#!/usr/bin/env python # -*- coding: utf8 -*- import RPi.GPIO as GPIO import time import threading import os import sys, signal from pyserver.network import * # https://github.com/juhgiyo/pyserver import ultrasonic, dcMotor, servoMotor, dht11</pre>



```
##### GLOBAL VARIABLE #####
""
    DC모터, 서보모터, 초음파센서(공통 Trigger, 개별 Echo), 온습도센서의 Pin 번호를 구성하는
    변수들
""

#### PIN 설정
# DC모터

MotorPinA_EN = 26
MotorPinA_IN1 = 19
MotorPinA_IN2 = 13

MotorPinB_EN = 0
MotorPinB_IN1 = 6
MotorPinB_IN2 = 5

# 서보모터
VServoMotorPin = 25 # 180도
HServoMotorPin = 8 # 300도

# 초음파센서 Echo가 추가되면 us_prevDis 수정
TrigerPin1 = 20
EchoPin1 = 23
TrigerPin2 = 21
EchoPin2 = 24

# 온습도 센서
TempPin = 15

""

    UDP_LastRecvTime: UDP 수신에서 모터 제어가 이뤄진 마지막 시간
    위 변수로 주기적으로 받아지는 UDP 제어 신호가 어던 문제가 발생하여 수신이 이뤄지지
    않는 지 판단 가능

    ReturningMode: 정해진 시간동안 UDP 수신이 정상적으로 이뤄지지 않으면 자동 복귀 모드로
    전환되는 변수들

    Waypoints: 모터의 "좌, 우, 앞, 뒤"를 몇 초동안 움직였는 지 기록하는 변수 (주기적으로
    저장되는 것이 아닌 이동 방향이 변동될 때 마다 저장됨)
""
```



```
#### 기타
UDP_LastRecvTime = 0.0 # UDP 마지막으로 받은 시간
ReturningMode = False # 드론 복귀 모드
Waypoints = [] # 복귀를 위한 Waypoint 변수 선언 [ [이동방향, 시간], ... ]

##### GPIO initialization #####
"""
    DCLeft, DCRRight에 dcmotor.py에 있는 DCMotor Class를 생성해서 DC모터를 구동하기 위한
    기본적인 초기화를 함
    ultrasonic_trigger은 ultrasonic.py에 있는 Class를 사용하며 공통으로 묶어진 Trigger로 사용해
    동시에 Ping을 보냄
    ultrasonic은 ultrasonic.py의 Ultrasonic class 생성을 한다. 또, Echo Pin을 사용하여 ISR
    방식으로 거리 측정을 함
    servoMotor은 servoMotor.py의 서보모터 제어를 위한 Calss 생성 및 초기화
    temperature dht11.py의 DHT11 제어를 위한 Class 생성 및 초기화
"""

##### 초기 설정
print('GPIO Setup')
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# DC모터
DCLeft = dcMotor.DCMotor(MotorPinA_EN, MotorPinA_IN1, MotorPinA_IN2)
DCRight = dcMotor.DCMotor(MotorPinB_EN, MotorPinB_IN1, MotorPinB_IN2)

# 초음파
ultrasonic1 = ultrasonic.Ultrasonic(TrigerPin1, EchoPin1)
ultrasonic2 = ultrasonic.Ultrasonic(TrigerPin2, EchoPin2)

# 서보모터
servoMotor1 = servoMotor.ServoMotor(VServoMotorPin, 0) # 180도
servoMotor2 = servoMotor.ServoMotor(HServoMotorPin, 1) # 300도

# DHT11 온습도계
temperature = dht11.DHT11(TempPin)

##### DCMotor 제어
"""
    DCMotor_Set은 UDP 신호가 들어올때 호출되거나 복귀 모드를 사용할 때 호출하기 위해
    별도로 함수를 선언함

```



DCMotor_Set('F') 를 사용하면 앞으로 전진함

"""

```
def DCMotor_Set(data):
```

```
    if data == 'S':
```

```
        DCLeft.setMotor(100, dcMotor.STOP)
```

```
        DCRight.setMotor(100, dcMotor.STOP)
```

```
    elif data == 'F':
```

```
        DCLeft.setMotor(100, dcMotor.FORWARD)
```

```
        DCRight.setMotor(100, dcMotor.FORWARD)
```

```
    elif data == 'B':
```

```
        DCLeft.setMotor(100, dcMotor.BACKWARD)
```

```
        DCRight.setMotor(100, dcMotor.BACKWARD)
```

```
    elif data == 'L':
```

```
        DCLeft.setMotor(100, dcMotor.FORWARD)
```

```
        DCRight.setMotor(100, dcMotor.BACKWARD)
```

```
    elif data == 'R':
```

```
        DCLeft.setMotor(100, dcMotor.BACKWARD)
```

```
        DCRight.setMotor(100, dcMotor.FORWARD)
```

```
    elif data == 'FR':
```

```
        DCLeft.setMotor(20, dcMotor.FORWARD)
```

```
        DCRight.setMotor(100, dcMotor.FORWARD)
```

```
    elif data == 'FL':
```

```
        DCLeft.setMotor(100, dcMotor.FORWARD)
```

```
        DCRight.setMotor(20, dcMotor.FORWARD)
```

```
    elif data == 'BR':
```

```
        DCLeft.setMotor(20, dcMotor.BACKWARD)
```

```
        DCRight.setMotor(100, dcMotor.BACKWARD)
```

```
    elif data == 'BL':
```

```
        DCLeft.setMotor(100, dcMotor.BACKWARD)
```

```
        DCRight.setMotor(20, dcMotor.BACKWARD)
```

"""

```
    elif data == 'FL':
```

```
        DCLeft.setMotor(100, dcMotor.STOP)
```

```
        DCRight.setMotor(100, dcMotor.FORWARD)
```

```
    elif data == 'FR':
```

```
        DCLeft.setMotor(100, dcMotor.FORWARD)
```

```
        DCRight.setMotor(100, dcMotor.STOP)
```

```
    elif data == 'BL':
```

```
        DCLeft.setMotor(100, dcMotor.STOP)
```

```
        DCRight.setMotor(100, dcMotor.BACKWARD)
```



```
elif data == 'BR':
    DCLeft.setMotor(100, dcMotor.BACKWARD)
    DCRight.setMotor(100, dcMotor.STOP)
    """"

##### UDP Recevied Process #####
""""

prev_udp_rcv_order는 현재가 아닌 이전에 수행한 DC모터 제어 (예로 'F', 'B'와 같은 데이터가
저장됨)
prev_udp_rcv_time는 이전에 수행했던 명령이 언제 시작했는 지 기억하기 위한 변수들

Waypoint_Append(currentTime):
    UDP 데이터에서 새로운 방향의 데이터가 들어오거나 UDP가 신호가 일정 이상 들어오지
    않는 경우
    이전에 기억하고 있는 prev_udp_rcv_order(현재가 아닌 이전 명령)과 현재 시간에서
    이전에 수행을 시작한 시간을 뺀 시간(즉, 해당 명령을 동작한 시간[예로 2초동안 앞으로])
    두 변수를 tmpArray에 임시로 배열을 지정한다. [앞으로, 2초]
    Waypoints에 맨 끝에 tmpArray를 넣어준다 [[정지, 10초], [앞으로, 2초]]

UDP_Recevied_Process(server, ip, port, data)
    HEY라는 UDP 데이터를 받으면 OK라 답변해 통신에 이상 없음을 확인해준다
    복귀 모드를 종료하고 이전의 명령, 이전 명령이 시작 시간, 이전 이동을 기억한
Waypoints를 초기화 해준다.

[! 삭제될 수 있는 사안] 만약 어떠한 문제로 잠시 동안 UDP 제어가 받아지지 않다가
받아지면
복귀 모드를 종료하고 이전의 명령, 이전 명령이 시작 시간, 이전 이동을 기억한
Waypoints를 초기화 해준다.

현재 시간을 기록하고
DC모터 제어 신호(앞, 뒤, 왼, 오 등...)가 확인되면 DC모터를 제어하고
만약 위 제어 신호가 아니라면 현재 시간 기록을 삭제한다.

현재 시간 기록이 존재한다면 현재 명령이 이전 명령과 다른지 확인하여 다르다면
Waypoint를 기록한다
""""

prev_udp_rcv_order = 0
prev_udp_rcv_time = 0.0

def Waypoint_Append(currentTime):
```



```
global Waypoints, prev_udp_rcv_order, prev_udp_rcv_time
tmpGetTick = currentTime - prev_udp_rcv_time # 현재 명령 시간에서 이전 명령 시간을
빼서 이전에 이동한 시간 구하기
tmpArray = [prev_udp_rcv_order, tmpGetTick]
Waypoints.append(tmpArray)
prev_udp_rcv_order = 0

def UDP_Receivied_Process(server, ip, port, data):
    global UDP_LastRecvTime, ReturningMode, Waypoints, prev_udp_rcv_order, prev_udp_rcv_time
    if data == 'HEY': # 초기 세팅
        server.send(ip, port, 'OK'.encode())
        ReturningMode = False
        prev_udp_rcv_order = 0
        prev_udp_rcv_time = 0.0
        if len(Waypoints) > 0:
            del Waypoints[:]
    else:
        if ReturningMode == True:
            ReturningMode = False
            prev_udp_rcv_order = 0
            prev_udp_rcv_time = 0.0
            if len(Waypoints) > 0:
                del Waypoints[:]

        GetTick = time.time()
        if (data == 'S' or data == 'F' or data == 'B' or data == 'L' or data == 'R' or
            data == 'FL' or data == 'FR' or data == 'BL' or data == 'BR'):
            DCMotor_Set(data)

        UDP_LastRecvTime = GetTick
        if prev_udp_rcv_order != data: # 이전 명령과 현재 명령어 다를 경우 기록
            if prev_udp_rcv_order != 0: # 맨 끝난 명령만 기억하므로 현재 명령은 무시
                Waypoint_Append(GetTick)
            prev_udp_rcv_order = data
            prev_udp_rcv_time = GetTick

##### TCP Receivied/Close Process #####
"""

TCP_Receivied_Process(sock, data)

서보모터 제어는 "V|123", "H|250" 과 같이 수직이면 "V", 수평이면 "H"이며 그 뒤의
```



각도와 같이 데이터가 받아진다
받은 데이터를 V와 123, H와 250으로 따로 따로 나누고
나누어진 값으로 수평, 수직 서보모터를 제어한다

"""

```
TCP_Buffer = ''
def TCP_Receved_Process(sock, data):
    global TCP_Buffer
    TCP_Buffer += data
    if '\n' in data: # 데이터의 끝이라면
        tmp_data = data.split('|') # | 기준으로 명령어를 구분

        for lines in tmp_data:
            if lines == 'REQUEST':
                clearSensorData() # 이전에 전송되던 데이터 값 초기화
            elif 'V:' in lines: # 분할된 문자열 중 V:가 포함되다면
                servo_order = int(lines[2:]) # V|100이면 100만 가져옴
                if (30 <= servo_order) and (servo_order <= 180): # 제어 범위를 넘으면 제어하지
                    않도록
                    servoMotor1.Set(servo_order)
            elif 'H:' in lines: # 분할된 문자열 중 H:가 포함되다면
                servo_order = int(lines[2:]) # H|300이면 300만 가져옴
                if (30 <= servo_order) and (servo_order <= 270): # 제어 범위를 넘으면 제어하지
                    않도록
                    servoMotor2.Set(servo_order)

def TCP_Close_Process():
    #global dcMotor.FORWARD, dcMotor.BACKWARD, dcMotor.STOP
    print('Close TCP')

##### UDP Handler #####
# def on_started(self, server):
# def on_stopped(self, server):
# def on_received(self, server, addr, data):
# def on_sent(self, server, status, data):
class EchoUdpHanlder(IUdpCallback):
    def init (self):
        pass

# called when UDP server is started
```



```
def on_started(self, server):
    print('UDP Server is started')

# called when UDP server is stopped
def on_stopped(self, server):
    print('UDP Server is stopped')

# called when new packet arrived
def on_received(self, server, addr, data):
    #print('UDP Received : ' + str(data) + '(' + data.decode() + ') ' + ' From ' + addr[0] + ':' +
    str(addr[1]))
    UDP_Receieved_Process(server, addr[0], addr[1], data.decode())
    #print('Received : ' + str(data.decode())) # Python3

# called when packet is sent
def on_sent(self, server, status, data):
    print('UDP Send (' + str(status) + '): ' + str(data))

##### TCP Handler #####
# def on_newconnection(self, sock, err):
# def on_disconnect(self, sock):
# def on_received(self, sock, data):
# def on_sent(self, sock, status, data):
class EchoSocketHandler(ITcpSocketCallback):
    def init (self):
        pass

# called when new client connected to the server
def on_newconnection(self, sock, err):
    print('New connection made')

# called when the client disconnected
def on_disconnect(self, sock):
    print('Client disconnected')
    TCP_Close_Process()

# called when new packet arrived from the client
def on_received(self, sock, data):
    #host, port = sock.socket.getpeername() # erasersetMotor
    #print('TCP Received : ' + str(data) + ' From ' + host + ':' + str(port))
```




```
print('TCP Received : ' + str(data.decode())) # Python3
#sock.send(data)
TCP_Receivied_Process(sock, data.decode())

# called when packet is sent
def on_sent(self, sock, status, data):
    print('TCP Send (' + str(status) + '): ' + str(data))

# def on_started(self, server):
# def on_accepted(self, server, sock)
# def on_stopped(self, server):
class EchoServerHandler(ITcpServerCallback):
    def __init__(self):
        pass

    # called when server finished the initialization and started listening
    def on_started(self, server):
        print('TCP server is started')

    # called when new client accepted
    def on_accepted(self, server, sock):
        print('New socket is accepted')

    # called when the server is stopped listening and disconnect all the clients
    def on_stopped(self, server):
        print('TCP server is stopped')

# def on_accept(self, server, addr):
# def get_socket_callback(self):
class EchoAcceptorHandler(IAcceptor):
    def __init__(self):
        self.handler=EchoSocketHandler()

    # called when new client connected
    # must return boolean : True to accept the connection otherwise reject
    def on_accept(self, server, addr):
        return True # accept always

    # called when the server accepted the client and ask for the handler
    # Must return ITcpSocketCallback object
```



```
def get_socket_callback(self):
    return self.handler

##### Threading #####

"""
UDP_LastRecvTime = 0.0 # UDP 마지막으로 받은 시간
ReturningMode = False # 드론 복귀 모드
Waypoints = [] # 복귀를 위한 Waypoint 변수 선언 [ [이동방향, 시간], ... ]

LostCheckThread()
    저장된 Waypoints가 없으면 루프를 다시 돈다

    복귀모드가 꺼져있고 현재시간과 마지막으로 UDP 수신받은 시간을 빼서 250ms이상이면
    모터를 정지하고 마지막으로 수행한 모터제어 명령을 Waypoints에 기록한다

    복귀모드가 꺼져있고 현재시간과 마지막으로 UDP 수신받은 시간을 빼서 30초 이상이면
    복귀모드를 수행한다

    복귀모드가 수행 중이면 반복문을 수행한다
        ReturningMode가 여전히 True인지 Waypoints에 저장된 기록들이 남아있는 지 확인
        후 루프를 수행한다
        Waypoints.pop()을 수행하여 맨 마지막으로 저장된 값을 가져오고 Waypoints의 맨
        마지막에 저장된 값을 지워준다
        runDCMotor에 Waypoints의 맨 마지막 값이 저장되어 있는 데, 수행한 명령을 반대로
        바꿔준다
        Delay를 사용하여 해당 명령이 수행한 시간동안 루프를 대기한다 (만약, 새로운 UDP
        명령이 들어오면 모터 제어를 해당 루프 내에서 계속 바꾸는 것이 아니라 제어에
        영향 없어 보임)
"""

def LostCheckThread():
    global ReturningMode, UDP_LastRecvTime, Waypoints
    while True:
        time.sleep(0.001) # 1ms

        if len(Waypoints) == 0:
            continue

        if ReturningMode == False and (time.time() - UDP_LastRecvTime > 5.0): # 복귀 모드가
            꺼져있고 5초 동안 UDP 데이터가 없으면
```



```
ReturningMode = True
```

```
elif ReturningMode == False and (time.time() - UDP_LastRecvTime > 0.25): # 복귀 모드가  
꺼져있고 250ms 동안 UDP 데이터가 없으면
```

```
if prev_udp_rcv_order != 0: # 맨 끝난 명령만 기억하므로 현재 명령은 무시
```

```
    DCMotor_Set('S') # 모터 정지
```

```
    Waypoint_Append(time.time())
```

```
elif ReturningMode == True:
```

```
    while ReturningMode == True and len(Waypoints) > 0: # 수정필요
```

```
        runDCMotor = Waypoints.pop()
```

```
        if runDCMotor[0] == 'F':
```

```
            runDCMotor[0] = 'B'
```

```
        elif runDCMotor[0] == 'B':
```

```
            runDCMotor[0] = 'F'
```

```
        elif runDCMotor[0] == 'L':
```

```
            runDCMotor[0] = 'R'
```

```
        elif runDCMotor[0] == 'R':
```

```
            runDCMotor[0] = 'L'
```

```
        elif runDCMotor[0] == 'FL':
```

```
            runDCMotor[0] = 'BL'
```

```
        elif runDCMotor[0] == 'FR':
```

```
            runDCMotor[0] = 'BR'
```

```
        elif runDCMotor[0] == 'BL':
```

```
            runDCMotor[0] = 'FL'
```

```
        elif runDCMotor[0] == 'BR':
```

```
            runDCMotor[0] = 'FR'
```

```
        DCMotor_Set(runDCMotor[0])
```

```
        time.sleep(runDCMotor[1])
```

```
.....
```

```
MainThread()
```

```
    초음파센서 값과 온습도계 값, 전력상태를 확인하고 send_data라는 하나의 변수로 묶어서  
    보내준다
```

```
    초음파센서 값은 50ms마다 확인하여 컴퓨터로 보내준 거리 값과 1이상 차이냐  
    전송된다
```

```
    온습도계 값은 1s마다 확인하여 컴퓨터로 보내준 거리 값과 1이상 차이냐 전송된다
```



파워상태 값은 500ms마다 확인하여 전력에 이상이 생기면 값을 보내주며 정상일 때는 별도로 보내주지 않는다

send_data에 데이터가 존재하면 TCP 소켓에 있는 유저 처음 한 사람에게만 보내준다

"""

초기 파워 상태 구하기

power_lastStatus = 300

Ultrasonic 기록된 거리 값

Ultrasonic TCP로 전송된 거리 값

us_currentDis = [0, 0]

us_prevDis = [0, 0]

온습도계 보내진 값

th_prevData = [0, 0]

TCP sender Buffer

clearSesorData(): 이전에 보낸 데이터 값을 초기화해 다시 센서 값을 보내줌

def clearSensorData():

global power_lastStatus, us_prevDis, th_prevData

power_lastStatus = 300

us_prevDis = [0, 0]

th_prevData = [0, 0]

def MainThread():

global power_lastStatus, us_currentDis, us_prevDis, th_prevData

us_lastPoll = 0.0

dht_lastPoll = 0.0

power_lastPoll = 0.0

while True:

currentTime = time.time()

send_data = "

if (currentTime - us_lastPoll) >= 0.05: # 50ms

us_lastPoll = currentTime

us_currentDis[0] = ultrasonic1.getDistance()

us_currentDis[1] = ultrasonic2.getDistance()

#print('us1:' + str(us_currentDis[0]) + ' us2:' + str(us_currentDis[1]))

us_send_lastPoll = currentTime

for i in range(len(us_currentDis)):

if us_currentDis[i] != 0 and abs(us_prevDis[i] - us_currentDis[i]) >= 1: # 현재 값과
이전에 보내진 값이 1이상 차이날 경우



```

        us_prevDis[i] = us_currentDis[i]
        send_data += 'US' + str(i) + ':' + str(int(us_currentDis[i])) + '|'

    if (currentTime - dht_lastPoll) >= 1 : # 1sec
        dht_lastPoll = currentTime
        result = temperature.read()
        tmp_reulst = [ int(result.temperature), int(result.humidity) ]
        #print('Temmp:' + str(result.temperature) + ' Humid:' + str(result.humidity))
        if result.is_valid():
            for i in range(len(th_prevData)):
                if tmp_reulst[i] > 0 and abs(th_prevData[i] - tmp_reulst[i]) >= 1: # 현재 값과
                    이전에 보내진 값이 1이상 차이날 경우
                        th_prevData[i] = tmp_reulst[i]
                        send_data += 'TH' + str(i) + ':' + str(tmp_reulst[i]) + '|'

    if (currentTime - power_lastPoll) >= 0.5: # 500ms
        power_lastPoll = currentTime
        stream = os.popen('cat /sys/devices/platform/leds/leds/led1/brightness')
        output = int(stream.read())
        if output != power_lastStatus:
            power_lastStatus = output
            send_data += 'POWER:' + str(power_lastStatus) + '|'

    if len(send_data) > 0:
        for socket in tcp_server.get_socket_list():
            #data_length = len(send_data)
            #send_data = '$' + str(data_length) + '|' + send_data
            send_data += '\n'
            socket.send(send_data.encode())
            break

    #time.sleep(0.001) # 1ms

##### Main #####
"""
    UDP 서버를 50001포트에서 활성화한다.
    TCP 서버를 50002포트에서 활성화한다.
    MainThread, LostCheckThread 함수를 multi threading으로 사용한다.
"""

```



```
port = 50001
handler = EchoUdpHanlder()
bind_addr = ""
udp_server = AsyncUDP(port, handler, bind_addr)

port = 50002
acceptor = EchoAcceptorHandler()
server_handler = EchoServerHandler()
bind_addr = ""
no_delay = True # If True, Nagle's algorithm is not used, otherwise use Nagle's Algorithm
tcp_server = AsyncTcpServer(port, server_handler, acceptor, bind_addr, no_delay)

t1 = threading.Thread(target=MainThread, args=())
t1.daemon = True
t1.start()

t2 = threading.Thread(target=LostCheckThread, args=())
t2.daemon = True
t2.start()

"""
while True:
    try:
        time.sleep(5)
    except KeyboardInterrupt:
        print('Program Interrupt')
        GPIO.cleanup()
        sys.exit(0)
GPIO.cleanup()
"""

def signal_handler(sig, frame):
    GPIO.cleanup()
    sys.exit(0)

if name == 'main ':
    _signal.signal(signal.SIGINT, signal_handler)
    signal.pause()
```

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

3.8 통신부

3.8.1 통신부

드론의 통신부는 라즈베리 파이와 제어용 PC간의 무선 통신과 C#으로 작성된 PC의 드론 조종 프로그램과 부가 기능을 다룹니다. 드론의 통신에는 라즈베리 파이의 AP를 이용하여 UDP와 TCP 방식을 사용합니다.

UDP 방식은 수신된 데이터를 Checksum 으로 확인 후 데이터 손실을 발견, 전송 중 LOSS로 인해 정상 전송되지 않아도 재전송하지 않습니다. TCP 방식은 수신된 데이터가 손실되었다면 송신자에게 데이터를 다시 보내달라는 요청을 하여 정상적으로 데이터를 수신할 수 있습니다. 또한 전송 중 LOSS가 발생 시 수신자가 정상적으로 받았다는 신호를 일정 시간 동안 전송하지 않으면 발신자가 다시 데이터를 전송해 줍니다.

저희 팀은 각 전송 방식의 특성을 고려하여 DC모터 제어에는 UDP통신을, 센서 데이터와 서보모터 제어에는 TCP 통신을 사용합니다. 저희 팀에서는 해당 통신을 구현하는 프로그램을 팀 자체에서 직접 작성하여 C#을 이용하여 PC에서의 프로그램에 Python을 이용하여 라즈베리 파이의 제어에 사용합니다.

드론의 조작과 영상 전송 이외에도 자체적인 기능으로 PC와 드론간에 비정상적으로 연결이 끊기면 자동으로 복귀하는 기능, 미확인 물체가 접근하여 초음파 센서에 일정 이하의 값이 입력되면 경고를 해 주는 기능, 드론이 작동하기에 환경이 열악하여 온습도 센서가 일정 이상의 값을 보낼 경우 경고를 해 주는 기능, 라즈베리 파이에 전원이 정상적으로 공급되지 않을 경우 경고를 해 주는 기능을 포함하고 있습니다.

3.8.2 작성한 코드의 대략적인 설명

[C# DongaDCS 프로그램]

UDP 통신은 50001번 포트로 통신합니다. / TCP 통신은 50002번 포트로 통신합니다.

1. 실행 시 "라즈베리 파이의 아이피" 를 입력하여 연결 버튼을 누르면 입력된 아이피 로 UDP 통신으로 "HEY"라는 문자를 보내 서버의 상태를 확인합니다. 서버에서는 "HEY" 라는 문자를 받으면 "OK"라 회신해줍니다.

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

2. "OK" 라는 UDP 메시지를 받으면 라즈베리 파이와 연결이 가능하다 판단하여 메인 프로그램을 실행합니다. 라즈베리 파이와 TCP 연결, UI 초기화, 키보드 입력 감지를 시작합니다.
3. TCP 연결이 성공하면 라즈베리 파이의 센서 값을 요청하고, 카메라에 부착된 수직, 수평 서보 모터의 각도를 초기화하는 TCP 메시지를 보냅니다. 그리고 파이 카메라 서버를 프로그램에서 크로미움 기반 브라우저를 통해 카메라를 보여줍니다.

타이머1 (10ms)

- 라즈베리 파이에서 수신된 센서 값(초음파1, 초음파2, 온도계, 습도계, 전원상태)을 표시하기 위해 글자를 갱신합니다. 그와 더불어 정해진 조건에 해당하면 글자 색을 변경하여 경고합니다.
- C# 프로그램이 현재 활성화 상태인지 판단하여 활성화 상태라면 "W,A,S,D", "방향 키 위, 아래, 왼쪽, 오른쪽"을 감지하여 방향키 위, 아래일 경우 TCP로 전송하여 변경할 수직 서보 모터 각도를 증가, 감소합니다. 방향키 왼쪽, 오른쪽일 경우 TCP로 전송하여 변경할 수평 서보 모터 각도를 변경합니다. 이때 서보 모터의 각도 범위를 벗어날 경우 최대 혹은 최소 값으로 변경하여 이상이 없도록 합니다.

타이머2 (100ms)

라즈베리 파이로 서보 모터 각도 변경을 요청한 값을 UI에서 표시하기 위해 100ms마다 갱신합니다.

스레드1 (100ms)

- W, A, S, D 각 조합에 맞는 DC 모터 제어 신호를 생성하여 UDP 통신으로 라즈베리 파이에게 전송합니다. 전송할 DC 모터 제어 신호가 존재하지 않다면 "정지" 제어 신호를 보내줍니다.
- 변경할 수직, 수평 서보 모터 값이 변경될 경우 TCP 통신으로 서보 모터 제어 신호를 보냅니다.

TCP 수신 동작

- 수신된 데이터의 끝이 "NULL" 값이 아니라면 버퍼에 추가하여 "NULL" 값을 기다립니다.
- 데이터의 끝인 "NULL"이 확인되면 수신된 데이터의 명령들을 구분하기 위한 "|" 문자를 기준으로 나누어 줍니다.
- "|" 기준을 나누어 준 문자열을 확인하여 "POWER", "TH", "US" 각각 "전원", "온습도", "초음파거리"로 판단하여 현재 저장된 센서 값들을 갱신해줍니다.

[Python Raspberry Pi]

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

DC모터와 초음파센서, 서보 모터, 온습도계의 객체를 생성하고 초기화 합니다.
50001포트를 UDP 서버로 생성합니다. / 50002포트를 TCP 서버로 생성합니다.
스레드 MainThread를 생성합니다. / 스레드 LostCheckThread를 생성합니다.

UDP 메세지 수신

1. "HEY" 메세지를 수신하면 복귀 모드와 관련된 데이터를 초기화하고 "OK"라 송신 합니다.
2. "HEY" 메세지가 아니라면 복귀 모드 수행 중이라면 복귀 모드를 종료하고 초기화 합니다. 그리고 DC 모터 제어 신호인지 확인 후 DC 모터를 작동합니다. 이전에 작동한 모터 제어 신호가 있으면 제어가 실행된 시간과 제어 신호를 복귀 모드 데이터에 기록합니다.

TCP 메세지 수신

1. 메세지의 끝에 "NULL" 값이 확인된다면 데이터의 끝이라 판단하여 처리를 시작합니다.
2. "|" 문자를 기준으로 명령어를 나눕니다.
나누어진 문자가 "REQUEST"면 이전에 전송한 센서 값을 초기화하여 다시 보내도 록만듭니다.
"V:yyy"라면 수직 서보 모터를 yyy각도로 변경해줍니다.
"H:xxx"라면 수평 서보 모터를 xxx각도로 변경해줍니다.

LostCheckThread (복귀 모드 수행을 위한 스레드)

1. 복귀 모드 수행 중이지 않고 마지막으로 수신된 UDP 데이터가 250ms동안 없다면 DC 모터를 멈추고 마지막으로 수행한 명령을 복귀 데이터에 저장합니다.
2. 복귀 모드 데이터가 존재하고 마지막으로 수신된 UDP 데이터가 5초 동안 없다면 복귀 모드를 수행합니다.
3. 복귀 데이터에 저장된 최근 명령을 반대로 수행하고 수행한 시간 동안 동작함, 최근에서 오래된 순으로 동작하며 전진이라면 후진으로 동작하게 합니다.

MainThread

(센서 값을 확인하여 전송된 값과 비교하여 값이 다르다면 데이터를 전송합니다.)

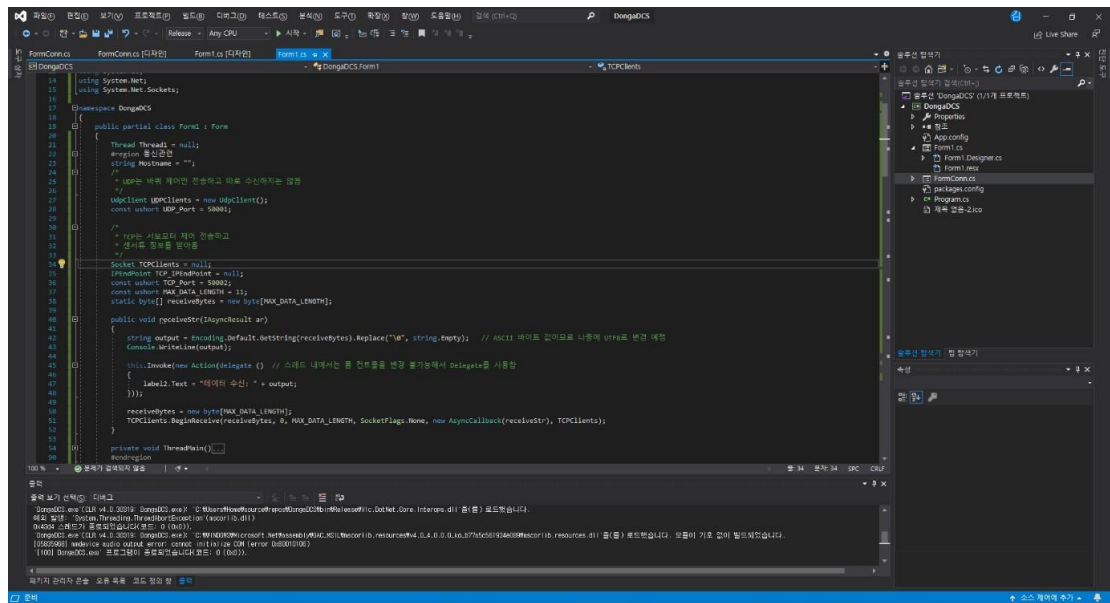
1. 초음파센서로 거리를 측정한지 50ms가 지났다면 초음파1, 초음파2의 거리를 측정합니다. 측정된 거리가 이전에 클라이언트로 전송된 거리 값과 1이상 차이난면 "US(초음파 센서 번호):거리값" 형식으로 전송 버퍼에 추가합니다.
2. 온습도 센서가 마지막으로 작동한 지 1초 이상이라면 온습도 센서값을 확인합니다. 현재 온습도 센서값이 이전에 클라이언트로 전송된 센서 값과 1 이상



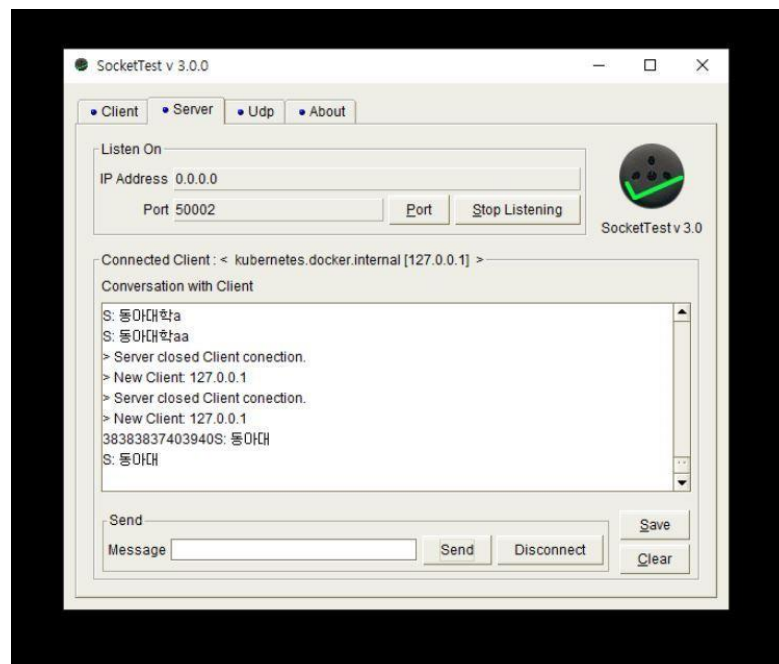
차이나면 "TH(0:온도, 1:습도):센서값]" 형식으로 전송 버퍼에 추가합니다.

3. 500ms마다 라즈베리 파이의 전원 이상 LED 값을 확인하여 전원 이상 유무를 판단합니다. 이전에 보낸 상태 값과 다르면 TCP 전송 버퍼에 "POWER:상태값]" 형태로 추가합니다.

4. 전송할 버퍼가 존재한다면 끝에 "NULL" 값을 붙여 TCP 전송을 합니다.

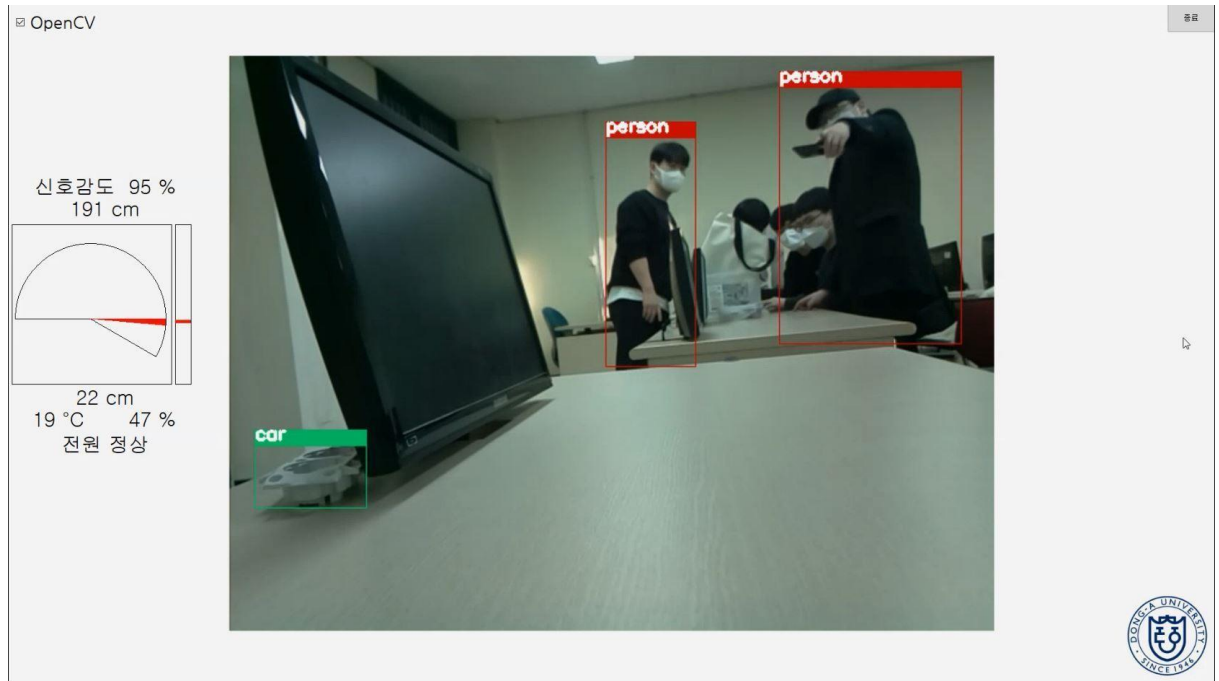


<코드 작성 환경>



<50002 포트 연결 테스트>

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09



<실제 조종 프로그램의 UI>

좌측의 최상단은 Open CV를 켜고 끄는 버튼입니다. 우측의 최상단은 프로그램 종료 버튼입니다.

좌측의 UI는 위에서부터 드론과 조종 PC간의 신호 강도, 전방 초음파센서 거리, 서보모터의 현재 각도 표시, 후방 초음파센서 거리, 온습도 데이터, 라즈베리파이 전원 상태를 표시하고 있습니다.

3.8.3 통신부 코드

통신 접속 확인 코드	
<pre>using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Linq; using System.Text; using System.Threading; using System.Windows.Forms;</pre>	



```
using System.Net;
using System.Net.Sockets;
//UI 예외처리
namespace DongaDCS
{
    public partial class FormConn : Form
    {
        public delegate void FormSendDataHandler(string ServerIP);
        public event FormSendDataHandler DataEvent;

        private UdpClient UDPClients = new UdpClient(); // UDP Client
        string Hostname = "";
        const ushort ServerPort = 50001;           // RemoteServer Port
        bool Success = false;                      // 호스트 확인이 성공적이면

        public FormConn()
        {
            InitializeComponent();
        }

        private void FormConn_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (e.CloseReason != CloseReason.FormOwnerClosing && !Success) // 폼 사용자가 닫을 경우
                this.Owner.Close(); // 폼 소유주까지 닫기
        }

        /*
        아이피를 입력하고 연결 버튼을 클릭하면 수행되는 작동
        해당 아이피 주소로 HEY라는 UDP를 보내주고 수신을 받을 준비함
        */
        private void button1_Click(object sender, EventArgs e)
        {
            Hostname = ipText.Text.Trim(); // TextBox에서 빈칸 제거

            if (Hostname.Length > 0) // 서버 정보가 있다면
            {
                string data = "HEY"; // 보낼 메시지
                byte[] datagram = Encoding.UTF8.GetBytes(data); // 메시지를 바이트화해서 보냄
            }
        }
    }
}
```



```
this.Text = Hostname + "의 상태 확인중...";
try
{
    UDPClients.Send(datagram, datagram.Length, Hostname, ServerPort);    // HEY라는
    단어를 서버로 UDP 전송
    UDPClients.BeginReceive(new AsyncCallback(recv), null); // 전송 후 응답 확인을
    비동기적으로 확인
}
catch (Exception ex)
{
    this.TopMost = false;    // 최상위 폼에서 잠시 내림
    MessageBox.Show(new Form() { WindowState = FormWindowState.Maximized,
    TopMost = true }, ex.Message, "경고", MessageBoxButtons.OK,
    MessageBoxIcon.Exclamation);
    Console.WriteLine(ex.ToString());
    this.TopMost = true;    // 메시지 박스를 클릭 후 원복
}
}

/*
UDP 수신에서 OK라는 답변을 받으면 해당 창을 닫고 제어 프로그램에게 아이피 주소를
전송하고 프로그램을 보여준다
*/
private void recv(IAsyncResult res)
{
    IPEndPoint RemoteIpEndPoint = new IPEndPoint(IPAddress.Parse(Hostname), ServerPort);
    byte[] received = UDPClients.EndReceive(res, ref RemoteIpEndPoint);

    string receiveMSG = Encoding.UTF8.GetString(received); // 수신받은 바이트를 UTF8로
    인코딩
    Console.WriteLine("UDP Recv: {0}", receiveMSG);
    if(receiveMSG.Equals("OK")) // OK면
    {
        this.Invoke(new Action(delegate () // 스레드 내에서는 폼 컨트롤을 변경 불가능해서
        Delegate를 사용함
        {
            Success = true;
            DataEvent(Hostname);    // 부모창으로 이벤트 전송
        }
    }
}
```

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

```

        this.Close();
    });
}
else
    UDPClients.BeginReceive(new AsyncCallback(recv), null); // OK가 아니면 다시 수신 대기
}
}
}
}

```

드론 제어 프로그램 코드	
<pre> using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Linq; using System.Text; using System.Threading; using System.Windows.Forms; using System.Runtime.InteropServices; using System.Diagnostics; using System.Reflection; using System.IO; using System.Net; using System.Net.Sockets; using Emgu.CV.Dnn; using Emgu.CV; using Emgu.CV.Util; using Emgu.CV.Structure; using DarknetYolo; using DarknetYolo.Models; namespace DongaDCS { public partial class Form1 : Form { [DllImport("user32.dll")] </pre>	



```
public static extern IntPtr FindWindow(string lpClassName, string lpWindowName);
[DllImport("user32.dll")]
public static extern IntPtr GetForegroundWindow();

private float vertical_servo = 90.0f;    // 수직 서보 각도
private const float min_v_servo = 30.0f;
private const float max_v_servo = 180.0f;

private float horizon_servo = 60.0f;    // 수평 서보 각도
private const float min_h_servo = 30.0f;
private const float max_h_servo = 240.0f;

private int[] dht11 = new int[] { 0, 0 };    // 온도 습도
private int[] distance = new int[] { 0, 0 };    // 거리
private bool powerError = false;

Thread Thread1 = null, Thread2 = null, Thread3 = null;    // 1: TCP송신, 2: Wifi 신호세기, 3:
OpenCV
#region openCV
string labels = @"Models\coco.names";
string weights = @"Models\yolov4-tiny.weights";
string cfg = @"Models\yolov4-tiny.cfg";
string video = @"Resources\test.mp4";

Image<Bgr, Byte> imgeOriginal;
private void timer1_Tick(object sender, EventArgs e)
{
    imageBox1.Image = imgeOriginal;
}

private void RefreshCamera()
{
    Console.WriteLine("카메라 시작");
    VideoCapture cap = new VideoCapture("http://" + Hostname + ":8090/?action=stream");
    DarknetYOLO model = new DarknetYOLO(labels, weights, cfg, PreferredBackend.Cuda,
    PreferredTarget.Cuda);
    model.NMSThreshold = 0.4f;
    model.ConfidenceThreshold = 0.5f;
    cap.FlipVertical = true;
    cap.FlipHorizontal = true;
```



```
while (true)
{
    Mat frame = new Mat();
    try
    {
        cap.Read(frame);
        //CvInvoke.Resize(frame, frame, new Size(720, 480));
    }
    catch (Exception ex)
    {
        Console.WriteLine("VideoEnded");
        frame = null;
    }
    if (frame == null)
        continue;

    CvInvoke.Rotate(frame, frame, Emgu.CV.CvEnum.RotateFlags.Rotate180);
    if (checkbox_openCV.Checked)
    {
        List<YoloPrediction> results = model.Predict(frame.ToBitmap(), 512, 512);
        try
        {
            foreach (var item in results)
            {
                MCvScalar Color;
                if (item.Label.Contains("person") == true)
                    Color = new MCvScalar(0, 0, 200);
                else
                    Color = new MCvScalar(100, 200, 0);
                CvInvoke.Rectangle(frame, new Rectangle(item.Rectangle.X, item.Rectangle.Y
                    - 13, item.Rectangle.Width, 13), Color, -1);
                CvInvoke.PutText(frame, item.Label, new Point(item.Rectangle.X,
                    item.Rectangle.Y - 5), Emgu.CV.CvEnum.FontFace.HersheySimplex, 0.5, new
                    MCvScalar(255, 255, 255), 2);
                CvInvoke.Rectangle(frame, item.Rectangle, Color, 1);
            }
        }
        catch (Exception ex)
        {
        }
    }
}
```




```
        Console.WriteLine(ex.Message);
    }
}
imageOriginal = frame.ToImage<Bgr, Byte>();

/*
CvInvoke.WaitKey(10);
*/
}
}
#endregion

#region 통신관련
string Hostname = "";
/*
* UDP는 바퀴 제어만 전송하고 따로 수신하지는 않음
*/
UdpClient UDPClients = new UdpClient();
const ushort UDP_Port = 50001;

/*
* TCP는 서보모터 제어 전송하고
* 센서류 정보를 받아옴
*/
Socket TCPClients = null;
IPEndPoint TCP_IPEndPoint = null;
const ushort TCP_Port = 50002;
const ushort MAX_DATA_LENGTH = 1024;
static byte[] receiveBytes = new byte[MAX_DATA_LENGTH];
StringBuilder sb = new StringBuilder();

/*
TCP 수신된 내용을 처리함
수신된 데이터의 종류(초음파, 온습도, 전력)를 분류하여 처리함
*/
public void receiveStr(IAsyncResult ar)
{
    try
    {
        int bytesRead = TCPClients.EndReceive(ar);
```



```
if (bytesRead > 0)
{
    string output = Encoding.Default.GetString(receiveBytes, 0, bytesRead).Replace("\0",
    string.Empty);
    sb.Append(output);
    /*
    if (waitingBuffer == 0 && output.Contains("$"))
    {
        waitingBuffer = Convert.ToInt32(output.Substring(1, output.IndexOf('|') - 1));
    }
    if((waitingBuffer + waitingBuffer.ToString().Length + 2) > sb.ToString().Length &&
    waitingBuffer > 0)
    {
        receiveBytes = new byte[MAX_DATA_LENGTH];
        TCPClients.BeginReceive(receiveBytes, 0, MAX_DATA_LENGTH, SocketFlags.None,
        new AsyncCallback(receiveStr), TCPClients);
        return;
    }
    output = sb.ToString();
    */

    output = sb.ToString();
    if (output[output.Length - 1] == '\n') // 전송된 데이터의 끝이라면
    {
        Console.WriteLine("결과 " + output);

        string[] result = output.Split('|');
        for (int i = 0; i < result.Length; i++)
        {
            if (result[i].Contains("POWER"))
            {
                string[] data = result[i].Split(':');
                if (data.Length == 2)
                {
                    int tmpPowerValue = System.Convert.ToInt32(data[1]);
                    if (tmpPowerValue == 255)
                        powerError = false;
                    else
                        powerError = true;
                    Console.WriteLine("Power:" + data[1]);
                }
            }
        }
    }
}
```



```
        }
    }
    else if (result[i].Contains("TH"))    // 온습도계 일 경우
    {
        string[] data = result[i].Split(':');
        if (data.Length == 2)    // 올바른 값이면 2개
        {
            for (int j = 0; j < 2; j++)
            {
                if (string.Compare(data[0], ("TH" + j).ToString(), true) == 0) //
                거리
                {
                    dht11[j] = System.Convert.ToInt32(data[1]);
                    Console.WriteLine(data[0] + ":" + data[1]);
                }
            }
        }
    }
    else if (result[i].Contains("US")) // 초음파 센서
    {
        string[] data = result[i].Split(':');
        if (data.Length == 2)    // 올바른 값이면 2개
        {
            for (int j = 0; j < distance.Length; j++)
            {
                if (string.Compare(data[0], ("US" + j).ToString(), true) == 0) //
                거리
                {
                    distance[ j] = System.Convert.ToInt32(data[1]);
                    Console.WriteLine("UltraSonic" + j + ":" + data[1]);
                }
            }
        }
    }
    sb = new StringBuilder();
    receiveBytes = new byte[MAX_DATA_LENGTH];
}

TCPClients.BeginReceive(receiveBytes, 0, MAX_DATA_LENGTH, SocketFlags.None, new
```



```
        AsyncCallback(receiveStr, TCPClients);
    }
    catch(Exception ex)
    {
        Console.WriteLine(ex.Message);
        MessageBox.Show(new Form() { WindowState = FormWindowState.Maximized, TopMost
        = true }, ex.Message, "경고", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        this.Invoke(new Action(delegate ()
        {
            GetHostnameForm();
        }));
    }
}

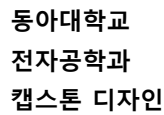
/*
TCP 연결을 성공할 경우 TCP 수신을대기하며
추가적으로 파이카메라를 확인할 수 있는 크로미움기반 웹 브라우저를 생성함
*/
public void ConnectCallback(IAsyncResult ar)
{
    try
    {
        TCPClients.BeginReceive(receiveBytes, 0, 11, SocketFlags.None, new
        AsyncCallback(receiveStr), TCPClients);
        TCP_Send("REQUEST|V:" + (int)vertical_servo + "|H:" + (int)horizon_servo + '\n');

        if (Thread3 != null)
            Thread3.Abort();
        Thread3 = new Thread(new ThreadStart(RefreshCamera)); Thread3.Start();

    }
    catch (Exception ex)
    {
        this.Invoke(new Action(delegate ()
        {
            GetHostnameForm();
        }));
        Console.WriteLine(ex.Message);
    }
}
```



```
/*
    DC모터 제어를 위한 UDP 데이터, 서보모터 제어를 위한 TCP 데이터 전송을 위해 Thread를
    생성함
    WASD 키에 따른 제어 조합을 구하고 UDP로 전송 (WASD키를 안 누르면 'S'가 전송됨)
    서보모터 위치가 이전에 전송한 서보모터 위치가 다를 경우 보내줌
*/
private void TCP_Send(String Data)
{
    SocketAsyncEventArgs args = new SocketAsyncEventArgs();
    byte[] szData = Encoding.UTF8.GetBytes(Data);
    args.SetBuffer(szData, 0, szData.Length);
    TCPClients.SendAsync(args);
}
private void ThreadMain()
{
    double prev_vertical_servo = vertical_servo;
    double prev_horizon_servo = horizon_servo;
    string dataMSG = "";
    while (true)
    {
        try
        {
            if (Hostname.Length > 0)    // 호스트 정보가 있다면
            {
                // UDP
                dataMSG = "S";
                if (KeyStatus[0] && !KeyStatus[1] && KeyStatus[2] && !KeyStatus[3]) // W A
                    dataMSG = "FL";
                else if (!KeyStatus[0] && !KeyStatus[1] && KeyStatus[2] && KeyStatus[3]) // W D
                    dataMSG = "FR";
                else if (KeyStatus[0] && KeyStatus[1] && !KeyStatus[2] && !KeyStatus[3]) // S A
                    dataMSG = "BL";
                else if (!KeyStatus[0] && KeyStatus[1] && !KeyStatus[2] && KeyStatus[3]) // S D
                    dataMSG = "BR";
                else if (KeyStatus[0] && !KeyStatus[1] && !KeyStatus[2] && !KeyStatus[3]) // A
                    dataMSG = "L";
                else if (!KeyStatus[0] && KeyStatus[1] && !KeyStatus[2] && !KeyStatus[3]) // S
                    dataMSG = "B";
                else if (!KeyStatus[0] && !KeyStatus[1] && KeyStatus[2] && !KeyStatus[3]) // W
```



프로젝트 명

재난 지역 탐사용 무선 지상 드론

팀명

팀 DCS

Confidential Restricted

Version 1.0

2021-DEC-09

캡스톤디자인



```
87, // W
68, // D
37, // 방향키 왼쪽
38, // 방향키 위
39, // 방향키 오른쪽
40, // 방향키 아래
};

public static bool IsKeyDown(int KeyCode) // 키 다운 상태인지
{
    return ((GetAsyncKeyState(KeyCode) & 0x8000) != 0) ? true : false;
}

public static bool IsKeyUp(int KeyCode) // 키 떴을 상태인지
{
    return ((GetAsyncKeyState(KeyCode) & 0x8000) == 0) ? true : false;
}

/*
FindWindow로 해당 프로그램이 활성화된 경우에
키보드 입력을 확인하고
방향키를 누를 경우 라즈베리파이에 전송할 서모모터의 각도를 프로그램에서 계산함
*/
private void Timer10_Tick(object sender, EventArgs e)
{
    IntPtr hWnd = FindWindow(null, this.Text);
    IntPtr Foreground = GetForegroundWindow();
    if (!hWnd.Equals(Foreground)) // 이 창이 활성화가 아닌 경우
    {
        for (int i=0; i<MAX_KEYS; i++) // 모든 키 떴을 걸로 변경
            KeyStatus[i] = false;
    }
    else
    {
        for (int i = 0; i < MAX_KEYS; i++)
        {
            if (!KeyStatus[i] && IsKeyDown(KeyList[i])) // 처음으로 KEY DOWN이면 변수
                TRUE하여 누른 상태로 감지
            {
                KeyStatus[i] = true;
            }
        }
    }
}
```



```
    }
    else if (KeyStatus[i] && IsKeyUP(KeyList[i])) // 변수가 TRUE일때 KEY UP이면 막 키를
    떼걸로 감지
    {
        KeyStatus[i] = false;
    }
}

if (KeyStatus[4] && !KeyStatus[5] && !KeyStatus[6] && !KeyStatus[7]) // 왼쪽
    horizon_servo += 1.0f;
else if (!KeyStatus[4] && !KeyStatus[5] && KeyStatus[6] && !KeyStatus[7]) // 오른쪽
    horizon_servo -= 1.0f;
else if (!KeyStatus[4] && KeyStatus[5] && !KeyStatus[6] && !KeyStatus[7]) // 위
    vertical_servo += 2.0f;
else if (!KeyStatus[4] && !KeyStatus[5] && !KeyStatus[6] && KeyStatus[7]) // 아래
    vertical_servo -= 2.0f;

if (horizon_servo < min_h_servo)
    horizon_servo = min_h_servo;
else if (horizon_servo > max_h_servo)
    horizon_servo = max_h_servo;

if (vertical_servo < min_v_servo)
    vertical_servo = min_v_servo;
else if (vertical_servo > max_v_servo)
    vertical_servo = max_v_servo;
}
}
#endregion

#region UI관련
private Panel panel;
private Label[] panelLabel = new Label[6];

const int _X = 0, _Y = 0, _SIZE_X = 250, _SIZE_Y = 250,
        _SERVO_Y = _Y + 75, _PADDING_XY = 4, _RECT_PADDING = 10, _HV_OFFSET = 10,
_V_WIDTHDIV = 10;
private void UI_Initialize()
{
    panel = new Panel();
```




```
panel.Location = new Point(5, this.Height / 2 + 50);
panel.Name = "panel";
panel.Size = new Size(300, 500);
panel.TabIndex = 6;
this.Controls.Add(panel);

for (int i = 0; i < panelLabel.Length; i++)
{
    panelLabel[i] = new Label();
    panelLabel[i].Font = new Font("굴림", 24.0f);
    panelLabel[i].Location = new Point(723, 160);
    panelLabel[i].Size = new Size(panel.Size.Width, 32);
    panelLabel[i].TabIndex = 8;
    panelLabel[i].Text = "";
    panelLabel[i].TextAlign = ContentAlignment.MiddleCenter;
    panel.Controls.Add(panelLabel[i]);
}

/* wifi 신호 표시 */
panelLabel[0].Location = new Point(_X, _Y);

/* 초음파 표시 */
panelLabel[1].Location = new Point(_X, panelLabel[0].Location.Y + 35);
panelLabel[2].Location = new Point(_X, panelLabel[1].Location.Y + _SIZE_Y + 50);

/* 온습도계 표시 */
panelLabel[3].Size = new Size(panel.Size.Width / 2, 30);
panelLabel[3].Location = new Point(_X, panelLabel[2].Location.Y + 35);
panelLabel[4].Size = new Size(panel.Size.Width / 2, 30);
panelLabel[4].Location = new Point(panel.Size.Width / 2, panelLabel[2].Location.Y + 35);

/* 전원 표시 */
panelLabel[5].Location = new Point(_X, panelLabel[4].Location.Y + 40);
}

private void SetLabelWarn(int index, bool warn)
{
    if (index < 0 || index >= panelLabel.Length) return;

    if (warn)
```



```
{
    panelLabel[index].ForeColor = Color.Red;
    panelLabel[index].BackColor = Color.Black;
}
else
{
    panelLabel[index].ForeColor = Color.Black;
    panelLabel[index].BackColor = Color.Transparent;
}
}

private void UI_Render()
{
    Graphics g = panel.CreateGraphics();
    g.Clear(this.BackColor);

    Rectangle rect = new Rectangle(_X, _SERVO_Y, _SIZE_X + _PADDING_XY, _SIZE_Y +
    _PADDING_XY);
    g.DrawRectangle(Pens.Black, rect);

    rect = new Rectangle(_X + _RECT_PADDING/2, _SERVO_Y + 30,
    _SIZE_X - _RECT_PADDING, _SIZE_Y - _RECT_PADDING);
    g.DrawPie(Pens.Black, rect, 180, 210);

    rect = new Rectangle(_X + _RECT_PADDING / 2, _SERVO_Y + 30,
    _SIZE_X - _RECT_PADDING, _SIZE_Y - _RECT_PADDING);
    g.FillPie(Brushes.Green, rect, 0, 5);

    rect = new Rectangle(_X + _RECT_PADDING/2, _SERVO_Y + 30,
    _SIZE_X - _RECT_PADDING, _SIZE_Y - _RECT_PADDING);
    g.FillPie(Brushes.Red, rect, 60 - horizon_servo, 5);

    rect = new Rectangle(_X + _SIZE_X + _HV_OFFSET, _SERVO_Y,
    _SIZE_X / _V_WIDTHDIV, _SIZE_Y + _PADDING_XY);
    g.DrawRectangle(Pens.Black, rect);
    rect = new Rectangle(_X + _SIZE_X + _HV_OFFSET, _SERVO_Y + (int)((_SIZE_Y +
    _PADDING_XY) * (max_v_servo - vertical_servo) / (max_v_servo - min_v_servo)),
    _SIZE_X / _V_WIDTHDIV, 5);
    g.FillRectangle(Brushes.Red, rect);
}
```



```
g.Dispose();
}

int ui_counter = 100;
int wifiSignal = 0;

private void ui_timer10_Tick(object sender, EventArgs e)
{
    int index = 0;

    SetLabelWarn(index, wifiSignal < 20);
    panellLabel[index++].Text = "신호감도 " + wifiSignal.ToString() + " %";

    SetLabelWarn(index, distance[0] < 10);
    panellLabel[index++].Text = distance[0].ToString() + " cm";

    SetLabelWarn(index, distance[1] < 10);
    panellLabel[index++].Text = distance[1].ToString() + " cm";

    SetLabelWarn(index, dht11[0] > 40);
    panellLabel[index++].Text = dht11[0].ToString() + " °C";

    SetLabelWarn(index, dht11[1] > 80);
    panellLabel[index++].Text = dht11[1].ToString() + " %";

    SetLabelWarn(index, powerError);
    panellLabel[index].Text = (powerError ? "전원 이상" : "전원 정상");

    ui_counter++;
    if (ui_counter % 5 == 0) // 50ms마다 작동
    {
        try
        {
            UI_Render();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}
```



```
        ui_counter = 0;
    }
}
#endregion

/* 현재 연결된 WIFI 정보 가져오기 */
private void getWifiStrength()
{
    while (true)
    {
        System.Diagnostics.Process p = new System.Diagnostics.Process();
        p.StartInfo.FileName = "netsh.exe";
        p.StartInfo.Arguments = "wlan show interfaces";
        p.StartInfo.UseShellExecute = false;
        p.StartInfo.RedirectStandardOutput = true;
        p.StartInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
        p.StartInfo.CreateNoWindow = true;
        p.Start();

        string result = p.StandardOutput.ReadToEnd();
        int pos = result.IndexOf("신호");
        if (pos == -1)
        {
            wifiSignal = -1;
            continue;
        }

        string signal = result.Substring(pos);
        signal = signal.Substring(signal.IndexOf(":"));
        signal = signal.Substring(1, signal.IndexOf("\n")).Trim();
        int signal_num = System.Convert.ToInt32(signal.Substring(0, signal.Length - 1));
        wifiSignal = signal_num;
        Thread.Sleep(1000);
    }
}

/*
라즈베리파이의 아이피를 입력하는 창을 띄우는 함수
*/
private void GetHostnameForm()
```



```
{
    Hostname = "";
    Timer10.Enabled = false;
    ui_timer10.Enabled = false;

    this.WindowState = FormWindowState.Minimized; // 폼 최소화
    this.ShowInTaskbar = false; // 작업표시줄 숨기기

    FormConn Connector = new FormConn(); // 서버 정보를 받기 위한 폼을 열음
    Connector.Owner = this; // FormConn의 소유주를 이 폼으로
    Connector.DataEvent += new FormConn.FormSendDataHandler(UpdateEventMethod); //
    delegate 이벤트 전송을 사용하여 자식폼에서 부모폼으로 데이터 전송
    Connector.Show(); // 자식폼 띄우기
    Connector.TopMost = true;

    if (Thread3 != null) // 카메라 갱신 정지
        Thread3.Abort();
}

/*
라즈베리파이 아이피 입력창에서 OK라는 수신을 받으면 해당 함수가 호출되는 데
수신에 성공한 아이피를 받아와 TCP 연결을 시도함
*/
private void UpdateEventMethod(string ServerIP)
{
    Timer10.Enabled = true;
    ui_timer10.Enabled = true;

    this.ShowInTaskbar = true; // 작업표시줄 보이기

    Hostname = ServerIP; // 이벤트로 받은 ServerIP 변수에 대입
    Console.WriteLine("호스트 정보 받음 {0}", Hostname);
    this.WindowState = FormWindowState.Maximized; // 폼 최대화

    TCPClients = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
    // 소켓 생성
    TCP_IPEndPoint = new IPEndPoint(IPAddress.Parse(Hostname), TCP_Port); // 전달받은
    HostIP로 EndPoint 생성

    TCPClients.BeginConnect(TCP_IPEndPoint, new AsyncCallback(ConnectCallback), TCPClients);
}
```



```
SocketAsyncEventArgs args = new SocketAsyncEventArgs();
args.RemoteEndPoint = TCP_IPEndPoint; // 소켓 Args에 Endpoint 대입
TCPClients.ConnectAsync(args); // 소켓 비동기 접속 시도
}

public Form1()
{
    InitializeComponent();
    UI_Initialize();
}

// 라즈베리파이 아이피 주소를 입력하는 창과 통신 전송을 위한 스레드 생성을 함
private void Form1_Load(object sender, EventArgs e)
{
    GetHostnameForm();
    Thread1 = new Thread(new ThreadStart(ThreadMain)); // 통신을 위한 스레드 생성
    Thread1.Start(); // 스레드 시작

    Thread2 = new Thread(new ThreadStart(getWifiStrength)); // WIFI 정보를 얻기 위한 스레드
    생성
    Thread2.Start(); // 스레드 시작

    imageBox1.Location = new Point(350, 0);
    imageBox1.Size = new Size(Screen.PrimaryScreen.Bounds.Width - 700,
    Screen.PrimaryScreen.Bounds.Height);
    imageBox1.SizeMode = PictureBoxSizeMode.Zoom;

    pictureBox_logo.Location = new Point(Screen.PrimaryScreen.Bounds.Width - 150,
    Screen.PrimaryScreen.Bounds.Height - 150);
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    if (Thread1 != null) // 스레드가 있으면
        Thread1.Abort(); // 스레드 강제 종료

    if (Thread2 != null)
        Thread2.Abort();

    if (Thread3 != null)
```



```
Thread3.Abort();

if (TCPClients != null && TCPClients.Connected) // TCP 세션이 열려있고
{
    TCPClients.Disconnect(false);
    //TCPClients.Close(); // 세션 닫아주기
}

private void Form1_Resize(object sender, EventArgs e) // 폼의 크기가 바뀔 경우
{
    if(Hostname == string.Empty) // 호스트 정보가 없을 경우127
    {
        this.WindowState = FormWindowState.Minimized; // 폼 최소화
    }
}

private void Button_Exit_Click(object sender, EventArgs e)
{
    this.Close(); // 프로그램 종료 버튼
}

}
```

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

4 역할 분담 및 개발 일정

4.1 역할 분담

윤주훈 – 조장. 목표 과제의 진행 파악. 라즈베리 파이 제어 코딩.

팀원들이 수행중인 과제 분배 상담과 진행 상황과 앞으로의 일정을 고려하여 목표를 적절하게 조율.

사용되는 모든 부품들의 호환성 검토 및 예산에 맞게 선정 후 구매.

작품이 완성되었을 때의 무게와 여러 상황을 고려하여 작동이 원활하게 이루어 질 수 있도록 DC모터와 기어박스의 선정과 충분한 전원 공급 방법 조사.

라즈베리 파이 환경에서 원하는 대로 DC 모터를 동작시키는 방법을 조사.

하드웨어의 전체적인 제작.

김태욱 - 조원. 스트리밍 및 통신 코딩. 드론 제어 코딩, 하드웨어 제작.

C# 프로그래밍을 통해 무선으로 드론의 모터와 서보 모터를 제어할 수 있는 방법을 구상 및 프로그램 제작.

C# 프로그래밍을 통해 무선으로 드론의 카메라에서 스트리밍 해 주는 영상을 수신할 수 있는 방법을 구상 및 프로그램 제작.

C# 프로그래밍을 통해 무선으로 드론의 각 센서에서 제공해주는 값을 수신할 수 있는 방법을 구상 및 프로그램 제작.

라즈베리 파이 프로그래밍을 위한 기본 세팅과 관련된 라이브러리 조사.

드론에 사용된 각 부품의 동작 코드 호환성 검토 및 조언.

드론에 사용된 각 부품의 하드웨어적 호환성 검토 및 조언.

최재원 – 조원. 라즈베리 파이를 이용한 드론 제어 코딩. 하드웨어 제작.

서보모터의 작동 전압에 따라 토크의 변화를 조사하여 목표하는 구동에 무리가 없을 정도의 적절한 서보모터와 전원부 선정.


서보모터 두 개를 이용하여 카메라와 초음파 센서가 다른 부품과 간섭 없이 3차원으로 기동할 수 있게 하드웨어 설계.

라즈베리파이 환경에서 원하는 대로 서보모터를 동작시키는 방법을 조사.

드론의 하드웨어 설계 구상. 각 부품들의 무게를 고려하여 적절하게 배치해 안정적으로 주행할 수 있게 설계.

최지훈 – 조원. 라즈베리 파이를 이용한 드론 제어 코딩.

라즈베리 파이의 호환 카메라들의 특성과 차이점을 조사하여 예산과 성능을 고려해 적절한 호환 카메라를 선정.

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

적외선 카메라와 일반 카메라의 장단점을 고려하여 어두운 환경에서도 시야를 확보할 수 있는 목표에 부합하는 설계와 부품 선정.


라즈베리 파이 환경에서 원하는 대로 호환카메라를 동작시키고 스트리밍 하는 방법을 조사.

허준석 – 조원. 라즈베리 파이를 이용한 드론 제어 코딩.

초음파센서의 특징을 조사하여 적절한 장소에 초음파 센서를 배치하여 드론의 사각을 보완할 수 있는 디자인을 설계.

드론에 사용된 모든 부품들과 라즈베리 파이의 작동 한계점을 조사하여 제작하는 드론의 작동 한계 환경을 가정하고 온습도 센서를 적절하게 이용하여 경고하는 시스템 설계.

라즈베리 파이 환경에서 원하는 대로 초음파 센서와 온습도 센서를 동작시키는 방법을 조사.

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

4.2 개발 일정

(수행 기간 : 2021년 9월 27일 ~ 2021년 11월 29일)									
구분 \ 과제내용	09/27 ~10/3	10/04 ~10/10	10/11 ~10/17	10/18 ~10/24	10/25 ~11/07	11/08 ~11/14	11/15 ~11/21	11/22 ~마감	
작품 방향성 회의									
부품 선정									
부품 구매									
라즈베리파이 초기설정									
모터, 서보모터 구동									
카메라 스트리밍 테스트									
초음파, 온습도 센서 테스트									
라즈베리 파이와 PC간 통신 테스트									
모터, 서보모터, 초음파 센서, 온습도 센서, 카메라 스트리밍 연동 테스트									
하드웨어 제작									
작품 마무리									

 동아대학교 전자공학과 캡스톤 디자인	결과보고서		
	프로젝트 명	재난 지역 탐사용 무선 지상 드론	
	팀 명	팀 DCS	
	Confidential Restricted	Version 1.0	2021-DEC-09

5 기대 효과

본 드론은 작은 크기를 가지고 있어 구조 대원이 투입되기 곤란한 지형에 투입 가능하고 사람보다 생화학적인 저항력이 높기에 특수한 환경에서도 사람을 대신하여 투입될 수 있습니다. 또한 카메라를 통해 투입된 현장의 상황을 실시간으로 확인할 수 있는 스트리밍 기능을 가지고 있고 카메라의 사각지대는 초음파 센서가 확인하며 온도 센서를 가지고 있어 현장 내부의 대략적인 상황을 파악할 수 있음은 물론 드론의 조종에도 편의를 가지며 이는 드론의 생존성에도 직결됩니다.

또한 사고 지역에 고립된 피해자가 좁은 공간과 어둠 속에서 언제 도착할지 모르는 구조대를 막연하게 기다리는 환경에서 탐사 드론이 자신을 발견한 것을 확인하면 구조대가 자신을 발견했다는 안도감과 구조될 수 있다는 희망을 가지게 되어 특히 어리거나 심약한 피해자의 심리 상태에 도움을 줄 수 있을 것으로 생각됩니다.

본 드론의 시스템을 유지한 채로 형태를 변형하거나 센서를 제거 또는 추가하는 등 기초플랫폼으로 이용할 수 있으며 본 드론에 적용된 무선 통신을 이용한 제어와 자동 복귀 및 영상 전송 기술은 본 과제의 지상 드론 뿐 아니라 비행, 수상 드론에도 적용하여 본 과제의 목표를 넘어 방송 용이나 취미용의 드론은 물론 군사 목적의 드론에도 기초 기술로 응용 가능할 것으로 생각됩니다.

드론의 구동 제어는 무선 청소기나 서빙 로봇 등 모터로 구동하는 로봇 등의 기초 기술로 응용할 수 있을 것으로 생각됩니다. 또한 각종 센서를 이용하여 주변 환경을 파악하는 능력은 드론이나 로봇을 넘어서 다양한 기기의 작동 한계 환경을 인식하여 경고하는 등 여러 분야에서 응용할 수 있을 것으로 생각됩니다.