

CAMERA VISION / EDGE AI / ODOMETRY MAPPING

암흑 및 재난 환경 극복을 위한 자율주행 기기

범용 카메라 영상 기반 딥러닝 판단과 실시간 삼각함수 오도메트리 2D 매핑 시스템

01 / INPUT

Camera Vision

암흑 실시간 감지

02 / INFERENCE

Edge AI BPU

딥러닝 경로 판별

03 / FINAL CONTROL

Odometry Engine

자율 회전 & 제어

01 / REAL-TIME TRAJECTORY

TRACKING



X: +132.5 cm / Y: +88.0 cm

X_OBSTACLE / DETECTED

CAPTURE

RealVNC Camera Stream

CLASSIFY

CNN-Based Edge AI

VALIDATE

Cross-Validation
Optimization

OUTPUT

Autonomous Avoidance Drive
2D Foot&Marker Map

INDEX

목차

- 01 APPLICATION SERVICE ARCHITECTURE
- 02 PROJECT IMPLEMENTATION PHOTOS DEVELOPMENT RESOURCES
- 03 DEVELOPMENT RESOURCES
- 04 CORE FEATURES & OPERATIONAL MECHANISM
- 05 PROBLEM DEFINITION & CORE IDEA
- 06 FUTURE ROADMAP
- 07 MOTIVATION & GOAL
- 08 PLANNING / DEVELOPMENT / TESTING
- 09 STACK

응용 서비스 구성도

재난 환경 탐색 → 비전/AI 추론 → 자율주행 제어 → 2D 매핑 및 구호 확장 파이프라인



결과물 구현 사진

트랙 → 기기주행 → 주행종료 후 최종 맵핑 데이터 (데모영상 참고)

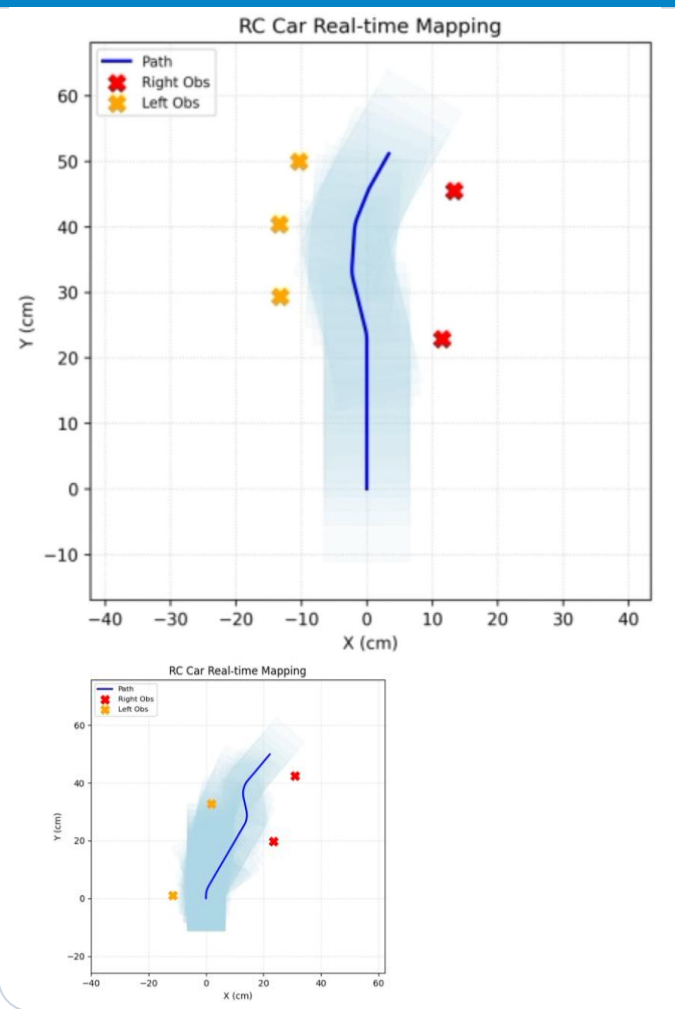
트랙 / 가로2m 세로3m 암막 텐트와 내부 장애물



기기 / 가로13cm 세로22cm 자율주행 기반 모터제어 기기



맵핑 데이터 / 실시간 장애물 회피경로와 장애물 표시마커



개발자료

재난 환경 → 실시간 비전기반 AI 추론 → 자율주행 제어 → 2D 맵핑

전체 시스템 흐름도



개발자료

제한된 예산 → 암흑 환경내 안정적인 자율주행 기기와 2D 매핑

기술 스택 선정 이유

하드웨어

Raspberry Pi	카메라 수집·AI 추론·모터 제어를 한 보드 내 처리
RC 차체 + DC모터 + L293D	13×22cm 소형으로 잔해 사이주행, 방향 제어
18650 + HW-688	독립 전원으로 모터 부하에도 오작동 방지
OV5647 (Fisheye)	광각 화각으로 좌·우 장애물을 동시에 포착

개발도구

Raspberry Pi OS	Linux 기반, 하드웨어 호환성과 안정성 우수
VS Code / Thonny	원격 개발과 현장 디버깅을 모두 지원
Tkinter / TkAgg	대시보드 GUI를 별도 설치 없이 구성
runpy / sys 로더	주행 코드를 수정하지 않고 동적으로 불러와 확장

소프트웨어

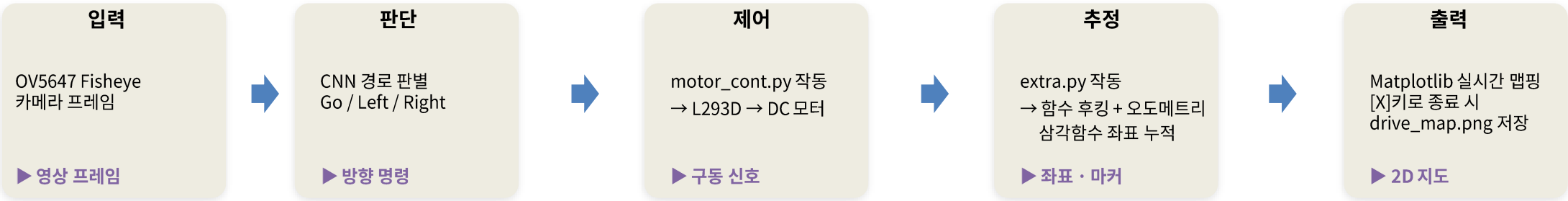
Python 3	비전·AI·제어·매핑을 한 언어로 빠르게 구현
NumPy 벡터 연산	좌표 변환을 일괄 계산해 실시간 연산 부하 감소
Matplotlib (TkAgg)	지도를 실시간 렌더링하고 PNG로 바로 저장
motor_cont	모터 제어 전용 라이브러리로 주행 로직과 분리

주요기술

위치 추정 (Odometry)	삼각함수 누적만으로 추가 센서 없이 위치 계산
멀티스레딩	GUI와 주행 제어를 동시에 실행해 지연 방지
함수 후킹 (Hooking)	모터 제어를 감싸 회전 순간에 마킹을 주입
동적 매핑	주행 궤적과 장애물 위치를 자동 계산해 누적

→프레임당 딥러닝 기반 판단 → 구동부 → 위치추정 → 실시간 맵핑

실시간 데이터 흐름



매 프레임 반복 · 멀티스레딩 : 주행 제어 스레드 // GUI (지도 렌더링)스레드가 동시에 실행

① 위치 추정

회전·직진 명령을 후킹해
이동 정보를 가로채 기록
삼각함수(cos·sin)로
x, y, 방향을 누적
13×22cm 면적을
하늘색 잔상 도포 → 유효 통로

② 장애물 마킹

우측 감지
→ 좌회전이 시작되는 **찰나**
우측면 5cm 바깥 **빨간 X**
좌측 감지
→ 우회전, **주황 X**
통로 (안전)와 장애물 (위험)구역

③ 시작 · 종료

[Z] 입력: **자율주행 + 맵핑** 동시시작
[X] 입력: 모터 출력 **즉시 차단**
누적 지도를 바탕화면
drive_map.png로 자동 저장

지도 범례

- 파란 중심선 : 이동 궤적
- 하늘색 면적 : 13×22cm 유효 통로
- 빨간 X 마크 : 우측 장애물
- 주황 X 마크 : 좌측 장애물

핵심기능과 작동 방식



핵심기능과 작동 방식

자율주행을 시작하고 종료하는 키 인식

조작 환경

RealVNC 화면에 포커스 클릭 후 키보드 키 입력

[Z] 자율주행 시작 · Line 49-51

manual = 0 → 제어 루프 CNN 판단 시작

[X] 주행 종료 · Line 52-57

주행 플래그를 0으로 → 모터 정지와 지도 저장

keyboard_cont.py

```
49     if key.char.lower() == 'z':  
50         self.manual = 0  
51         print('Autonomous mode ON')  
52     if key.char.lower() == 'x':  
53         self.manual = 1  
54         self.go_flag = 0  
55         self.left_flag=0  
56         self.right_flag=0  
57         print('Autonomous mode OFF')
```

핵심기능과 작동 방식

기존 주행 코드와 매핑 코드 교번

Line 120과 Line 200 함수 교체

원래 drive()를 보관 후 extra_drive()로 교번

Line 198 실제 구동은 동일

계산 후 원래 drive()를 불러 모터 동작 동일

Line 208-213 별도 스레드

지도 창을 띄운 뒤 main.py를 실행

extra.py

```
120     original_drive = motor_cont.drive
121
122     def extra_drive(go_flag, left_flag, right_flag, brake_flag, back_flag):
123         ...
124         original_drive(go_flag, left_flag, right_flag, brake_flag, back_flag)
125
126     motor_cont.drive = extra_drive
127
128     ...
129
130     # Fire up interactive live GUI canvas module inside an isolated thread worker instantly
131     live_ui_thread = threading.Thread(target=init_and_run_live_plot, daemon=True)
132     live_ui_thread.start()
133     print("[Live UI Engine Enabled] Booting autonomous core pipeline...")
134
135     sys.argv = [os.path.join(os.getcwd(), 'main.py')]
136     runpy.run_path(sys.argv[0], run_name='__main__')
```

핵심기능과 작동 방식

카메라 영상을 판단용 입력으로 변환

Line 43 상단 200px 제거

장애물이 보이는 바닥 쪽 영역에 집중

Line 66 64×64 축소

라즈베리파이에서도 빠르게 추론되는 크기로 변환

Line 73 픽셀 값 0~255을 -1,0,1로 정규화

학습 때와 같은 방식으로 각 픽셀 값 0~255를
-1~1 범위로 정규화

주행시 학습 때와 동일 조건으로 이미지 판단

camera_capture.py

```
38     frame = picam2.capture_array()
39     frame = cv2.flip(frame, -1)
40     frame = cv2.resize(frame, (512, 512))
41
42     # TODO: 이미지를 어떻게 바꿔서 처리할까?
43     frame = frame[200:,:] # 위쪽 200 픽셀 제거
44
45     ...
46
47     # 모델에 맞게 이미지 크기 조절
48     frame = cv2.resize(frame, (64, 64))
49
50     # 저장할 이미지 선정 (pred_image)
51     save_image = frame
52
53     # # 학습 전 이미지 전처리
54     frame = np.asarray(frame, dtype=np.float32).reshape(1, 64, 64, 3) #(1,224,224,3)
55     frame = (frame / 127.5) - 1
56
57     # 학습에 사용할 이미지, 저장되는 이미지 선정 (pred_image)
58     self.pred_image = frame
```

핵심기능과 작동 방식

CNN 모델의 네 가지 주행 행동 학습 / Go · Left · Right · Brake

초기 학습

38장

클래스당 · 불안정

데이터 보강 후

78장

클래스당 · 안정

Line 63-73 합성곱 4층 + 분류층

64×64 입력 → Go · Left · Right · Brake 확률
특징추출 4층을 거쳐 특징압축

- 선 · 모서리 특징
- 가장 큰 값
- 모서리특징으로 윤곽 특정
- 넓은 범위 패턴파악

분류층

- 압축된 특징을 512개의 숫자 한줄로 변환
- 64개의 판단근거 정리
- 과적합을 막는 Dropout

model.py

```

...
ain, Y_valid = train_test_split(data, labels, test_size=0.2, random_state=42 )
...
61 # Convolutional Neural Network Model
62 model = tf.keras.Sequential()
63 model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(64, 64, 3)))
64 model.add(layers.MaxPooling2D((2, 2)))
65 model.add(layers.Conv2D(128, (3, 3), activation='relu'))
66 model.add(layers.MaxPooling2D((2, 2)))
67 model.add(layers.Conv2D(64, (3, 3), activation='relu'))
68 model.add(layers.MaxPooling2D((2, 2)))
69 model.add(layers.Conv2D(32, (3, 3), activation='relu'))
70 model.add(layers.Flatten())
71 model.add(layers.Dense(64, activation='relu'))
72 model.add(layers.Dropout(0.5))
73 model.add(layers.Dense(4, activation='softmax'))
74
75 # 모델 학습
76 model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
77 history = model.fit(X_train, Y_train, validation_data=(X_valid, Y_valid), epochs=10, batch_size = 128)

```

Line 59 검증 20% · TFLite 변환

데이터의 20%는 학습에 쓰지 않는 검증용으로 실제 판단 능력을 확인
이후 학습된 모델은 TFLite로 변환 후 라즈베리파이에서 가볍게 실행
→ 0.08초마다 다음 행동 판단

핵심기능과 작동 방식

판단을 조향 각도와 모터 출력으로 옮기며 장애물 회피 주행

motor_cont.py · config.py

Line 73-85 행동별 출력값

행동	서보	좌 / 우 출력
직진	83°	14.5 / 14.5
좌회전	60°	20 / 14.5
우회전	105°	14.5 / 20

config.py 일괄 관리

조향 각도와 모터 출력 같은 주요 값을 config.py 한 곳에 넣어 관리, 조정 속도를 더 빠르게 함

```

73     elif go_flag == 1:
74         if left_flag==right_flag: # left right together
75             rpi_servo.set_deg(NEUTRAL_DEG)
76             rightMotor(1,0, GO_OUTPUT)
77             leftMotor(1,0, GO_OUTPUT)
78     elif left_flag == 1: #left
79         rpi_servo.set_deg(LEFT_DEG)
80         rightMotor(1,0, GO_OUTPUT)
81         leftMotor(1,0, TURN_OUTPUT)
82     elif right_flag == 1: #right
83         rpi_servo.set_deg(RIGHT_DEG)
84         rightMotor(1,0, TURN_OUTPUT)
85         leftMotor(1,0, GO_OUTPUT)

```

핵심기능과 작동 방식

시간과 삼각함수로 차량 경로 맵핑

extra.py

Line 158-159 직진

12 cm/s $\times$ dt를 $\cos \cdot \sin$ 으로 x, y에 나눠 가산

Line 161-164 회전

회전시 40 °/s로 방향을 바꿔 속도의 40%로 전진

Line 144-145 시간 보정

한 주기가 0.15초를 넘으면 0.08초로 보정

```
143 dt = current_time - last_time
144 if dt > 0.15:
145     dt = 0.08
146 last_time = current_time
...
155 # Precision trigonometric accumulation odometry engine
156 if active_action == 0:
157     rad = math.radians(car_angle)
158     car_x += CAR_SPEED * dt * math.cos(rad)
159     car_y += CAR_SPEED * dt * math.sin(rad)
160 elif active_action == 1:
161     car_angle += TURN_SPEED * dt
162     rad = math.radians(car_angle)
163     car_x += (CAR_SPEED * 0.4) * dt * math.cos(rad)
164     car_y += (CAR_SPEED * 0.4) * dt * math.sin(rad)
```

핵심기능과 작동 방식

회전이 시작되는 순간 장애물 마킹

extra.py

Line 176-181 직진 → 좌회전

오른쪽 장애물로 보고 빨간 X 마킹

Line 183-188 직진 → 우회전

왼쪽 장애물로 보고 주황 X 마킹

Line 178 표시 거리

차폭 절반(6.5cm) + 5cm, 진행 방향과 $\pm 90^\circ$

```
174     if incoming_action != active_action:
175         with plot_lock:
176             if active_action == 0 and incoming_action == 1:
177                 rad_obs = math.radians(car_angle - 90)
178                 obs_dist = (CAR_WIDTH / 2.0) + 5.0
179                 obs_x = car_x + obs_dist * math.cos(rad_obs)
180                 obs_y = car_y + obs_dist * math.sin(rad_obs)
181                 right_obstacles.append((obs_x, obs_y))
182
183             elif active_action == 0 and incoming_action == 2:
184                 rad_obs = math.radians(car_angle + 90)
185                 obs_dist = (CAR_WIDTH / 2.0) + 5.0
186                 obs_x = car_x + obs_dist * math.cos(rad_obs)
187                 obs_y = car_y + obs_dist * math.sin(rad_obs)
188                 left_obstacles.append((obs_x, obs_y))
189
190         active_action = incoming_action
```

핵심기능과 작동 방식

차량이 지나간 면적을 누적해 실시간 맵핑

Line 81-86 차체 꼭짓점

차량의 실제 크기 13×22cm 사각형을
현재 방향 각도만큼 회전하여 지나간 면적 그림

Line 88-89 하늘색 누적

반투명하게 겹쳐 확보된 통로 표시

Line 113-115 자동 저장

종료 시 바탕화면에 drive_map.png(300dpi)로
자동저장

extra.py

```

77         rad = math.radians(c_angle)
78         cos_a, sin_a = math.cos(rad), math.sin(rad)
79         f, b, w = CAR_LENGTH/2.0, -CAR_LENGTH/2.0, CAR_WIDTH/2.0
80
81         corners = np.array([
82             [cx + f*cos_a - (-w)*sin_a, cy + f*sin_a + (-w)*cos_a],
83             [cx + f*cos_a - w*sin_a, cy + f*sin_a + w*cos_a],
84             [cx + b*cos_a - w*sin_a, cy + b*sin_a + w*cos_a],
85             [cx + b*cos_a - (-w)*sin_a, cy + b*sin_a + (-w)*cos_a]
86         ])
87
88         poly = Polygon(corners, closed=True, color='lightblue', alpha=0.08, zorder=2)
89         ax.add_patch(poly)
...
113     desktop_path = "/home/pi/Desktop/drive_map.png"
114     try:
115         fig.savefig(desktop_path, dpi=300, bbox_inches='tight')

```

향후계획

실시간 장애물 회피 및 매핑 → 소형화 → 군집주행 메커니즘 → 구호 물자 전달

현재 단계	장비 초소형화	군집 주행 & 병합	구호 물자 전달
• 카메라+딥러닝 자율주행 • 오도메트리 2D 매핑 • 4방위 멀티 탐색 검증 • 암흑 속 유효 통로 확보	• 손바닥 크기로 소형화 • 붕괴 잔해 암흑 진입 • 경량 AI 모델 최적화 • 잔해 속 암흑 추적 능력 확보	• 수십 대 동시 투입 • 분산 네트워킹 데이터 공유 • 2D 지도 실시간 병합 • 도면 없이 골든타임 내 내부파악	• 확인된 유효 통로 활용 • 요구조자에게 의약품 전달 • 가장 빠른 안전 경로 반환 • 골든타임 내 구조 지원

제작동기 및 목표

암흑/재난 사각지대 → 38장/AI 추론 → 78장/ AI 고도화 → 매핑 및 군집 구호 시스템

진입불가 암흑사각지대에서 시행착오와 유효통로를 찾는 기술

저희가 주목한 일상의 사각지대는 건물이 붕괴된 재난 현장, 그중 사람의 손이 닿지 않고 빛도 없는 암흑 공간이었습니다.

처음부터 완벽한 정답을 가진 완성형 로봇을 만들겠다는 거창한 목표로 시작하지 않았습니다. 오히려 제한된 예산과 환경 속에서 카메라 비전과 기본적인 오도메트리 엔진만으로 어디까지 구현할 수 있을지, 무모해 보이는 도전을 하였습니다.

실제로 개발 초기에는 단 38장의 데이터만으로 학습된 AI가 갈팡질팡하며 장애물을 인식하지 못하는 수많은 **실패**를 겪었습니다. 하지만 이 시행착오를 AI의 손실률 수치로 기록하며 부족한 데이터를 78장으로 보완하고 매핑 오차를 잡아가는 과정 속에서 기기가 조금씩 **진화**하는 즐거움을 배웠습니다.

비록 지금은 카메라와 딥러닝으로 1대의 기기가 암흑을 헤쳐나가는 대안의 걸음마 단계이지만, 더 나아가 구호 시스템을 향후 완성하고자 합니다.

이 **실패와 진화의 루프**가 반복되며 기기를 손바닥 크기로 초소형화하고, 수십대의 기기가 협력하는 군집 주행 시스템을 형성하고,

붕괴된 재난현장에서 구조대원의 안전을 지키고, 요구조자에게 구호물자를 삼시간에 전달하여

재난 현장의 골든타임을 실질적으로 지켜내는 **구호 인프라의 기초기술**이 될 것이라 확신합니다

기획 / 제작 / 시험 진행과정

암흑환경 트랙보안 → 광각 라이트로 장애물 인식을 향상 → AI 추론 정확도 향상



트랙 재조정 사유

암막 재조정

1차 확정 후 실전 테스트에서 **암막 환경 미흡** 트랙 내구성 문제 발견
빛 차단 및 안정적 테스트를 위해 2차 재조정 실시

기기 재조정 사유

광각 라이트 교체

LED 라이트 사용 시 암막 내 **직진광**으로 인한 **한정적인 학습률 발생**
차량 앞바퀴 보완 및 LED light에서 광각 라이트로 교체

데이터 재입력 사유

정확도 향상

라이트 변경으로 인한 3차 데이터 입력후 소량데이터 로딩으로 인한
판단 정확도 저하 보완 4차에 데이터를 추가 반영하여 AI성능 극대화

GKEML 시스템 스펙



하드웨어 (HW)

메인 보드: 라즈베리 파이 (Raspberry Pi)

차량 프레임: 13cm × 22cm 소형 RC 차체

구동 시스템: DC 구동 모터 & 모터 드라이버 (L293D) & 카메라 OV5647 (Fisheye)

전원 공급: 배터리 (Panasonic 18650) 및 전압 안정화 모듈 (HW-688)



소프트웨어 (SW)

개발 언어: Python 3 기반 알고리즘

실시간 매핑: Matplotlib (TkAgg 렌더링)

행렬/수치 계산: NumPy Vector Operations

모터 제어: motor_cont 커스텀 라이브러리



개발 도구 (Dev Tools)

운영체제: Raspberry Pi OS (Linux 기반)

통합 개발 환경: VS Code / Thonny IDE

디스플레이 엔진: Tkinter / TkAgg GUI

동적 모듈 로더: Python runpy / sys 파이프라인



주요 기술 (Key Tech)

위치 추정 (Odometry): 삼각함수 누적 주행 추적

멀티스레딩: GUI 및 주행 제어 동시 실행

함수 후킹 (Hooking): 모터 제어 래핑 및 추적 주입

동적 매핑: 주행 궤적 및 장애물 자동 계산



감사합니다

Q&A

sonhh7580@naver.com

GKEML Autonomous Rescue System