학사학위논문

자율주행 응급차를 활용한 병원이송 시스템

- Unity를 이용한 자율주행 병원이송시스템 -

지도교수 박경모

경희대학교
생체의공학과

정서윤 2020104123

2024년 12월

자율주행 응급차를 활용한 병원이송 시스템

- Unity를 이용한 자율주행 병원이송시스템 -

지도교수 박경모

이 논문을 학사 학위논문으로 제출함

경희대학교
생체의공학과

정서윤

2024년 12월

정서윤 의 학사학위 논문을 인준함

지도교수 박경모

경희대학교

2024년 12월

목차

1장 서론

1.1. 한국 응급의료 시스템의 문제점

한국의 응급의료 시스템은 오랜 기간 동안 발전해 왔음에도 불구하고, 여전히 여러 가지 구조적 문제를 안고 있다. 현재 한국은 응급의료 시스템의 중심축을 담당할 '컨트롤타워'가 부재한 상태로, 현장에서 임기응변으로 대응해야 한다. 그 결과, 환자 배분과 병원 수용 관리가 체계적으로 이루어지지 않는다. 특히, 응급환자 발생 시 병원 이송 과정에서의 비효율성과 응급실 과밀화 문제가 심각하다. 실제 사례로 2023년 대구에서 발생한 '중증외상 환자 표류 사건'은 한국 응급의료 시스템의 구조적 한계를 여실히 보여준다. 이 사건에서는 구급대가 8개 병원에 수용 가능 여부를 문의했지만, 환자가 적절한 병원에 도착하지 못해 치료가 지연되었고 결국 사망에 이르렀다. 사실상 거리를 달리는 구급차 안에서 구급대원이 직접 전화를 돌려가며 환자를 수용해줄 병원을 찾다 보니 효과적으로 환자를 배분하기는 불가능하다. 결국 이 사건은 병원 수용 여부 확인을 위해 구급대원이 수십 통의 전화를 돌려야 하는 비효율적 구조와, 중앙화된 관리 시스템의 부재에서 비롯된 문제다.

또한, 중증 환자와 경증 환자가 응급실을 혼재해 사용하며, 대형병원 응급실이 과잉 포화되는 현상도 빈번하다. 중앙응급의료센터의 보고서에 따르면, 대형병원을 찾는 응급환자 중 약 47.6%는 경증 환자에 해당한다. 이러한 현상은 응급실 내 의료진의 효율성을 저하시킬 뿐 아니라, 진정한 응급환자들이 적시에 치료받을 수 있는 가능성을 감소시킨다. 또한, 구급대원이 이송하는 환자의 중증도를 판단했더라도 환자가 대형병원을 가겠다고 하면 거부하기가 어려운 구조다. 위와 같은 시스템으로는 실제로 대형병원에서 치료받아야 할 중증 긴급 환자를 제대로 이송하기 어렵다.

응급차의 높은 교통사고율 또한 크게 대두되는 문제점이다. 2016년 소방청 통계에 따르면, 구급차 1,352대 중 329건의 교통사고가 발생하여 24.3%의 사고율을 기록했다. 이는 같은 해 국내 등록 자동차 교통사고 발생률인 0.8%에 비해 약 30배 높은 수준으로, 구급차가 일반 차량보다 훨씬 높은 사고 위험에 노출되어 있음을 보여준다. 교통사고 발생 시 긴급환자(15%),응급환자(28.3%), 잠재 응급환자(33.9%)가 포함된 이송이 지연되어 골든타임을 놓칠 가능성이 크다. 구급차 교통사고의 50%는 병원 이송 중에 발생하며, 이로 인해 환자의 생존율 저하와 의료비 증가 같은 부정적 영향을 초래한다.

교통사고 발생 환경의 주요 특성은 다음과 같다. 사고 장소는 일반 도로에서 60.2%로 가장 높고, 사고 시간은 오전 9시~오후 6시 사이가 47.4%로 가장 많다. 계절은 겨울에 32.1%로 사고 빈도가 가장 높고 요일로는 목요일(29.1%)과 금요일(18.9%)에 사고가 가장 많이 발생한다.

위와 같은 교통사고의 주요 원인으로는 첫 번째, 운전자의 부주의와 피로가 있다. 차량 소통이 원활한 상황에서도 전체 사고의 53.1%가 발생하고, 소방 경력 6년~10년 이하의 운전자가 사고 빈도가 가장 높은 것으로 파악됐다.

두 번째로, 기존 연구에 따르면 운전자의 감정 조절 부족 및 차량 속도 증가가 원인으로 크게

대두되었다. 응급 상황에서 골든타임을 놓칠 수도 있다는 생각에 차량을 급격하게 운전하거나 일반차량을 간격을 준수하지 않은채 앞서나가는 경우에 사고가 많이 발생해 오히려 응급 치료가 지체되는 상황이 많이 발생한다.
세번째로, 시야 확보 문제가 있다. 날씨가 맑은 날 사고 비율이 71.9%로 높으며, 흐린 날에는 시야 확보가 어려워 사고 경험 비율이 더 증가한다.

1.2. 독일 병원 이송 시스템의 효율성

독일의 병원 이송 시스템은 중앙구조관리국(Zentrale Koordinierungsstelle)을 중심으로 높은 효율성을 자랑한다. 이 시스템은 응급환자가 발생하면 병원의 병상 및 의료진 상황, 환자의 중증도를 실시간으로 파악하여 적절한 병원을 사전에 준비시킨다. 구체적으로, 독일 서부 귀터슬로시 중앙구조관리국 상황실에 들어서면 중앙에 설치된 대형 화면으로 심장마비나 외상, 신종 코로나바이러스 감염증(코로나19)을 비롯한 각 질환별로 어느 병원에 현재 이를 치료할 의료진이 있는지를 한눈에 보여준다. 이 화면에는 응급환자들이 탄 구급차가 어느 병원으로 가고 있는지 동선이 뜨고, 심지어 상황실 아래 여러대의 구급차들 중 어떤 구급차가 현재 수리 중인지도 알 수 있다.
응급환자 발생시 이송 과정은 다음과 같다. 응급환자가 발생하면 응급환자의 가족이 112로 전화를 걸고 바로 중앙구조관리국 상황실로 연결된다. 직원은 환자의 상태와 위치 등을 묻고 응급처치법을 조언하며 안심시킨다. 그사이 응급현장에 구급차가 도착한다. 응급구조사가 현장에서 보낸 환자 정보를 토대로 중증인지, 경증인지를 파악한다. 중앙구조관리국 상황실 직원은 응급구조사가 업데이트하는 환자의 상태를 보면서 인근 병원 병상 현황과 의료진 근무 여부를 확인해 '최적의 병원'으로 이송시킨다. 일단 구급차를 탄 환자는 어느 병원으로 갈지, 응급실에 갈지 등을 선택할 수 없고 중앙구조관리국의 안내에 따라야 한다. 또한, 병원과의 유기적인 협력도 이뤄진다. 환자를 어느 병원, 어느 의사에게 보낼지 결정하고 병원에 이를 공유하면 환자가 병원에 도착하기 최소 10분 전에는 응급처치 및 치료 준비를 끝낸다. 이후 병원에 도착하면 미리 마련해놓은 응급실 병상으로 이동한 후 의료진이 곧바로 치료한다.
독일 귀터슬로 중앙구조관리국 사례를 살펴보면, 지역 내 37만 명의 주민을 대상으로 연간 약 360건의 중증 응급환자 이송을 관리한다. 환자가 발생하면, 현장에서 구급대원이 중앙구조관리국에 정보를 전달하며, 중증 환자는 대형병원으로, 경증 환자는 소형 병원으로 이송되도록 명확히 구분된다. 엄격한 환자 분류로 응급실은 붐비지 않았고, 중증환자가 먼저 치료를 받을 수 있다. 환자를 보냈다면 독일 병원은 반드시 환자에게 1차 응급처치를 해야 하는 의무가 있다. 이러한 시스템은 응급실 과밀화를 방지하고, 중증 환자가 신속하게 적절한 치료를 받을 수 있도록 한다. 중앙화되고 체계적인 시스템 덕분에 독일에서는 '응급실 뺑뺑이'라는 개념 자체가 존재하지 않는다.

1.3 중앙 병원 이송 시스템의 필요성

한국 응급의료 시스템의 여러 문제점은 중앙화된 병원 이송 시스템 부재에서 비롯된다. 독일의 효율적인 병원 이송 사례는 한국 응급의료 체계가 나아가야 할 방향성을 제시한다. 독일은 중

앙구조관리국을 통해 병원과 구급차, 그리고 환자 정보를 실시간으로 연결하여 효율적으로 관리한다. 병원의 수용 가능 병상 수, 의료진 상태, 환자 중증도를 기반으로 가장 적합한 병원을 선택하고 이송 전 병원에서 준비를 완료하도록 하는 선제적인 구조를 갖추고 있다.
구체적으로, 중앙화 시스템을 통해 환자의 중증도와 병원의 상태를 실시간으로 확인하고, 경증 환자는 소형 병원으로, 중증 환자는 대형 병원으로 배분할 수 있다. 이러한 분류는 대형 병원의 응급실 과밀화를 방지하고, 의료진이 진정한 응급 환자에게 집중할 수 있는 환경을 제공한다. 엄격한 환자 분류가 이루어지면 응급실이 환자로 넘쳐나는 상황을 예방할 수 있으며, 응급 환자가 적시에 치료받을 확률이 높아진다.
또한, 현재 한국에서는 구급대원이 직접 전화를 돌려 병원 수용 여부를 확인하고 있지만, 중앙 시스템에서는 이를 자동화하여 적합한 병원을 즉시 결정한다. 구급차 내에서 발생하는 혼란을 줄이고, 환자가 병원에 도착하기 전 의료진이 준비할 수 있는 시간을 제공한다. 병상 정보와 의료진 상태를 실시간으로 파악하여 구급차가 병원에 도착하기 전 치료 준비를 완료한다. 이로 인해 골든타임 내 치료가 가능하며, 환자는 도착 즉시 응급 치료를 받을 수 있어 생존율이 증가한다.

한국의 응급차 사고율과 발생 원인을 비추어보아 기계적이고 체계적인 이송이 필요하다. 자율주행 응급차는 응급 운전에서의 어려움을 다음과 같이 해결할 수 있다.
자율주행 기술은 운전자 개입 없이 정밀한 차량 제어를 가능하게 하며, 운전자의 피로와 부주의로 발생하는 인간적 오류를 최소화한다. 골든타임을 확보하려는 운전자의 감정적 운전은 속도 증가와 차량 간격 미준수로 이어져 사고를 유발한다. 자율주행 응급차는 차량 간격과 안전 속도를 기계적으로 유지하며, 효율적인 이송을 가능하게 한다.
또한, 비, 안개, 눈 등으로 인한 시야 제한은 사고 발생률을 높인다. 자율주행 기술은 LiDAR, 레이더, 카메라 등 센서를 활용해 날씨 조건과 무관하게 정확한 거리 측정과 장애물 감지가 가능하다. 이는 특히 야간이나 겨울철 응급차 사고율을 크게 감소시킬 수 있다.

2장 개요

한국의 응급의료 시스템 문제를 해결하기 위해, 독일의 병원 이송 시스템 사례를 참고한 새로운 시스템 설계가 요구된다. 이에 따라, Unity 엔진을 활용한 가상 병원 이송 시스템을 제안한다. 가상 환경은 실제 의료 현장을 그대로 재현하며, 다양한 시나리오에서 시스템의 효율성을 평가할 수 있는 유용한 도구다. 본 연구에서는 Unity를 활용하여 가상 도시 맵과 병원 이송 알고리즘 및 자율주행 응급차를 설계함으로써 한국 응급의료 시스템의 구조적 문제를 해결하기 위한 새로운 방안을 제시하고자 한다.

병원 이송 시스템은 다음과 같은 주요 요소를 포함한다.
1. Unity 기반의 가상 도시 맵 구현
다양한 도시 환경을 재현하기 위해 에셋을 활용하여, 도시와 교통망을 모델링한다. 병원, 응급차, 도로 네트워크를 포함한 가상 환경을 구성하며, NavMesh를 생성하여 차량 이동 및 경로 탐색을 지원한다.
2. 효율적인 병원 선택 알고리즘
환자의 중증도와 증상 유형, 병원의 전문의 보유 여부, 혼잡도를 고려하여 최적의 병원을 선택한다. 환자가 도착하기 전 병원에 사전 준비를 완료하도록 병원 정보를 실시간으로 공유한다.
3. 구급차 할당 및 경로 최적화:
가장 가까운 구급차를 신속히 배치하고, NavMesh를 이용해 환자와 병원 간의 최단 경로를 계산한다. 구급차의 상태와 위치를 실시간으로 추적하여 효율적인 대기 목록 관리를 수행한다.
4. 대기 목록 관리 및 시각화:
제한된 구급차 수로 인해 발생하는 대기 시간을 최소화하기 위해, 환자의 대기 목록을 관리하고 예상 대기 시간을 실시간으로 제공한다. 환자의 대기 상태를 시각적으로 표현하여 환자, 구급차, 병원의 상황을 효율적으로 관리한다.

자율주행 응급차는 다음과 같은 시스템으로 작동한다.
1. 차선 인식 시스템
OpenCV for Unity를 활용하여 차선 검출 시스템을 구축한다. 먼저 차량 전방 카메라로 주기적인 2D 이미지 캡처 한 후 그레이스케일 변환과 관심 영역(ROI) 설정을 통해 불필요한 영역을 제거한다. 이후, 엣지검출과 허프 변환으로 이미지에서 선형 특징 감지 한다. 검출된 선분을 분석하고 선처리를 진행하여 필요한 선들인 중앙선, 대시라인, 통행 불가선만 분류할 수 있도록 한다. 검출된 선분을 Unity의 3D 월드 좌표계로 변환하여 차선을 Lane1과 Lane2로 정의한다. 이를 기반으로 차량의 위치를 차선안에 위치하도록 조정한다. 또한, 차량이 중앙선을 넘어가거나 지정된 차선에서 벗어날 경우, 차선 경계선과의 좌표 비교를 통해 즉각적으로 위치를 교정한다.
2. 충돌 회피 알고리즘
차량과 구급차 간의 동적 거리 계산을 통해 충돌 방지하며 응급차와 차량 기준 회피 알고리즘

을 이용하여 차량의 위치를 조정한다.
응급차 기준 회피 알고리즘은 응급차의 앞・옆・앞차의 앞 차량 상태를 분석하여, 차선 변경 또는 속도 조절을 수행한다. 필요한 경우, 응급차 또는 일반 차량을 이동시키거나 속도를 증가시켜 교통 흐름을 원활히 유지한다.
일반 차량 기준 회피 알고리즘은 일반 차량이 앞 차량과의 거리 및 옆차선 상태를 기반으로 최적 경로를 탐색한다. 옆차선이 비어 있을 경우 차선을 변경하고, 그렇지 않으면 기존 차선을 유지한다.

본 연구에서 제안하는 시스템은 응급 의료 현장의 효율성을 높이고, 환자의 생존율을 향상시키는데 기여할 것으로 기대된다. 기존 현장에 새로운 디지털 기술을 도입하고, 최적화된 응급 서비스 모델을 제시함으로써, 독일과 같이 선진화된 의료 체계를 수립할 수 있을 것이다.

3장 본론

3.1 병원 이송 시스템

3.1.1 Unity 도시 맵 설정

그림1. Unity 도시 맵

가상 도시 맵은 이상적인 도시 구조와 교통 시스템을 재현하는 데 강력한 도구로 작용한다.

1) 에셋 선택 및 활용

① 에셋 선택

SimplePoly City - Low Poly Assets: 이 에셋 팩은 도시 환경을 구성하는 데 필요한 다양한 3D 모델과 오브젝트를 제공한다. 주요 요소는 다음과 같다.

-Props: 교통 신호등, 가로등, 벤치 등 도시 환경의 소품을 포함한다.

-Natures: 나무, 풀 등 자연물을 포함하여 도시의 녹지 공간을 표현한다.

-Buildings: 주거용, 상업용, 공공건물 등 다양한 건물 유형을 제공한다.

-Vehicles: 자동차, 버스, 트럭 등 다양한 차량을 포함한다.

Low Poly Road Pack: 도로 네트워크를 구성하기 위한 에셋으로, 다양한 형태의 도로와 교차로를 제공한다.

-Freeway: 일반 차선으로 사용되며, 직선 도로를 표현한다.

-Freeway Join Crossway: 3차 교차로를 구현한다.

-Freeway Join Freeway: 4차 교차로를 구현한다.

②오브젝트의 태그 설정

각 오브젝트에 적절한 태그를 부여하여 태그 기반 그룹 관리와 상호 작용을 용이하게 한다.

-Vehicles: 일반 차량은 'vehicle' 태그를 부여한다.

-Ambulance: 응급차는 'vehicle'에서 별도로 분리하여 'ambulance' 태그를 부여한다. 이를 통해 응급차의 특수한 동작과 경로 설정이 가능하다.

-Buildings: 건물은 'building' 태그를 사용한다.

-Hospital: 병원은 'building'에서 별도로 분리하여 'hospital' 태그를 부여한다. 병원 위치 정보를 활용하여 응급차가 최적 병원으로 이동하게 한다.

-Props: 교통 신호등 등 소품은 'prop' 태그를 사용한다.

-Person: 사람은 새로운 3D 오브젝트인 실린더(cylinder)를 사용하여 표현하고 'person' 태그를 부여한다.

③사람의 시각화

사람 오브젝트는 간단한 형태의 실린더를 사용하여 표현한다. 보행자 요소를 간단하게 표현하기 위해 실린더로 구성한다. 각 사람 오브젝트는 랜덤한 위치에 배치되며, 추후 동적 움직임을 수행한다.

④가상 도시 맵의 주요 구성 요소

최종적으로 가상 도시 맵은 다음과 같은 중요 요소들로 구성된다.

응급차(Ambulance): 'ambulance' 태그를 가진 응급차 3대를 배치한다. 병원과 사람 오브젝트 사이를 이동하며, 응급 상황 시나리오를 시뮬레이션한다.

병원(Hospital): 'hospital' 태그를 가진 병원 3곳을 맵 내에 분배하여 배치한다. 응급차의 목적지로 사용되며, 이송 시스템의 핵심 요소이다.

사람(Person): 'person' 태그를 가진 사람 오브젝트 10명을 맵 내에 배치한다. 이들은 응급 상황의 발생 지점으로 사용된다.

도로 네트워크: 'Low Poly Road Pack'의 에셋을 활용하여 현실적인 도로망을 구축한다. 일반 차선과 다양한 형태의 교차로를 포함하여 차량의 이동 경로를 다양화한다.

2) Navmesh 생성

도로 환경을 구축하기 위해 Navmesh를 생성한다. 도로 즉, freeway를 포함하는 게임오브젝트는 walkable로, 이를 제외한 게임오브젝트들(props, buildings, persons 등)은 not walkable로 구성한 뒤 bake한다.

Navmesh로 bake 된 영역은 차량들이 이동할 수 있는 부분으로 Navmesh에 포함된 Setdestination, SetPath등과 같은 기능 등을 수행할 때 이동할 수 있는 부분 내에서 최적 경로를 계산하고 판단하도록 한다.

3.1.2 병원 이송 알고리즘

효율적인 병원 이송 시스템은 응급 상황에서 환자의 생명을 구하는 데 핵심적인 역할을 한다. 본 절에서는 Unity 엔진을 활용하여 구현한 병원 이송 시스템의 구조와 동작 과정을 상세하게 설명한다. 환자 요청 생성부터 병원 선택 기준, 구급차 할당 및 출동, 대기 목록 관리까지 총 네 단계로 구성된다.

1) 환자 요청 생성

응급 상황의 현실적인 시뮬레이션을 위해 환자 요청은 무작위로 생성한다. 포아송 분포를 이용하여 환자 요청의 발생 간격을 모델링한다. 위 시스템에서는 환자 요청은 평균 분당 5개의 요청이 발생하도록 설정하고 다음 요청까지의 시간을 샘플링했다. 환자 요청 시간은 랜덤하게 정할 수 있으며 시뮬레이션의 효율성을 위해 5분으로 설정했다. 이후 해당 시간만큼 대기한 뒤 새로운 요청을 생성하고 이를 반복하는 과정을 수행한다.
새로운 요청이 생성될 때에는 'person' 태그의 오브젝트 중에서 이미 요청되지 않은 환자를 대상으로 해 중복 요청을 방지하고 안정성을 확보한다.
요청이 생성될 때마다 환자의 증상 유형과 중증도 또한 무작위로 할당한다. 이때, 환자의 특성은 환자 객체의 속성으로 저장된다.

증상유형	신경학적, 심혈관계, 중독 및 대사장애, 외과적, 출혈, 소와과적, 정신과적
중증도	1,2,3,4,5

이러한 증상 유형들은 응급의료에 관한 법률에서 정의한 응급실에서 필수적으로 다루어야 하는 8가지 주요 진료 과목에 해당한다. 진료 과목과 중증도를 통해 환자와 병원 의료진의 실시간 매칭을 구현하여 현실성과 정확성을 높였다.

2) 병원 선택 기준

구급차는 환자를 태운 후 적합한 병원을 선택하여 이동한다. 병원 선택은 환자의 증상 유형과 중증도, 병원의 전문의 보유 현황, 수용 가능 여부, 그리고 혼잡도(congestion level)를 종합적으로 고려한다.

①병원 속성의 랜덤 설정 및 UI 표시

전문의 수	0~5
최대수용인원	5~10
혼잡도	'Low', 'High'

각 병원은 신경외과, 심장학, 일반내과, 외과, 소아과, 정신과, 산부인과 등 다양한 분야의 전문의를 보유한다. 시스템 시작 시, 각 병원의 전문의 수를 0에서 5 사이의 정수로 랜덤하게 설정한다. 병원의 최대 수용 인원(capacity)은 5에서 10 사이의 값으로 랜덤하게 설정하며, 처음 수용인원은 0으로 설정한다. 병원의 혼잡도는 'Low' 또는 'High'로 랜덤하게 설정하며 이는 병원의 현재 운영 상태를 나타낸다.

각 병원은 자신의 전문의 수, 최대 수용 인원, 현재 수용 인원, 혼잡도 등의 정보를 병원 오브젝트 위에 텍스트 형태로 표시한다. CreateUI 함수를 통해 구현되며, 병원 객체의 UpdateUI 메서드를 통해 실시간으로 갱신된다.

그림2. Hospital UI

②전문의 보유 여부 확인

각 병원은 다양한 전문의를 보유하고 있으며, 이는 병원 객체의 속성으로 정의되어 있다. 환자의 증상 유형과 중증도에 따라 필요한 전문의 수를 결정하고, 해당 전문의를 충분히 보유한 병원을 후보로 선정한다.

중증도에 따라 환자를 치료하기 위해 필요한 전문의의 수는 다음과 같다.

중증도	필요 전문의 수
1	5
2	4
3	3
4	2
5	1

예를 들어, 중증도가 2인 환자는 해당 증상 분야의 전문의가 최소 4명 이상 있는 병원에서 치료받을 수 있다.

환자 요청시 환자의 증상과 환자의 중증도를 기반으로 필요한 전문의와 그 숫자를 도출한다. 병원이 해당 전문의를 충분히 보유하고 있는지 확인하고 조건을 만족하는 병원들을 후보로 선정한다.

③수용 가능 여부 확인

병원의 현재 수용 인원이 최대 수용 인원(capacity)을 초과하지 않는지 확인한다. 병원의 현재 입원 환자 수를 확인하여 병원의 최대 수용 인원보다 적은 병원일 경우 수용가능하므로 위 병원들을 후보로 선정한다.

만약 ②과 ③ 의 후보 병원이 같고 유일하다면 해당 병원을 besthospital로 선정한다. 후보 병원이 여러개일 경우, 거리와 혼잡도를 2차로 결정하여 선택한다.

④거리 및 혼잡도 기반 최적 병원 선정

후보 병원 중에서 구급차의 현재 위치와의 거리와 병원의 혼잡도를 합산하여 최적의 병원을

선택한다. 먼저 전문의 보유 여부와 수용 가능 여부를 모두 만족하는 병원들의 리스트를 만든다. 이후, 구급차의 현재 위치와 각 병원의 위치 간의 거리를 계산하여 순위를 지정한다. 또, 병원의 혼잡도를 수치화하여 'Low'가 'High'보다 높은 순위로 설정한다. 최종적으로 거리의 순위와 혼잡도의 순위를 합친 점수를 기준으로 병원들을 오름차순으로 정렬한다. 이때 합친 종합 점수가 가장 낮은 병원을 선택한다. 가장 낮은 종합 점수가 동일한 병원이 여러개일 경우, 랜덤으로 병원 한 개를 선택하도록 한다.
이 과정은 구급차의 이동 시간을 최소화하면서도 병원의 혼잡도를 고려하여 환자가 신속하고 원활한 치료를 받을 수 있도록 돕는다.

⑤병원 선택 실패 처리
모든 병원이 해당 환자를 수용할 수 없는 경우, 시스템은 이를 로그에 기록하고 환자에게 적절한 안내를 제공한다. 환자를 수용할 수 없는 경우는 최적 병원을 판단할 수 없는 경우로 전문의의 부족과 수용 능력 초과가 이에 해당된다. 수용할 수 없는 사유가 결정되면 환자의 UI를 통해 거부 사율를 전달하며 시스템 로그는 해당 내용을 기록하여 추후 분석에 활용한다.

3) 구급차 할당 및 출동
환자 요청이 생성되면 시스템은 즉시 이용 가능한 구급차를 탐색하여 할당하고, 구급차는 환자에게 출동한다.

①구급차 선택 로직
가장 가까운 구급차부터 시작하여 이용 가능 여부를 확인한다. 만약 가장 가까운 구급차가 이미 할당되었거나 사용 중인 경우, 다음으로 가까운 구급차를 검사한다. 이러한 과정을 반복하여 이용 가능한 구급차를 찾는다.
시스템은 FindClosestAvailableAmbulance함수를 통해 이미 할당되었거나 환자를 태우고 있는 구급차를 제외한 구급차를 순회하며, 환자와의 거리를 계산한다. 거리 계산은 Vector3.Distance 함수를 사용하여 구급차와 환자 간의 직선 거리를 측정하는 방식이다.
모든 구급차가 이미 사용 중이라면, 환자는 대기 목록(Queue)에 추가된다. 이는 requestQueue로 구현되어 있으며, 구급차가 이용 가능해질 때까지 환자의 요청이 대기하게 된다.

②구급차 할당 및 이동 시작
선택된 구급차는 해당 환자에게 할당되고, 환자의 위치로 이동을 시작한다. 이때, 세 개의 주요 플래그가 구급차 이동에 영향을 미친다.
isMoving 플래그는 구급차의 이동 상태를 나타내는 플래그로, 이동 중에는 true로 설정된다. 이동이 완료되면 false로 변경하여 다음 동작이 가능하도록 한다.
hasPerson 플래그는 구급차에 환자가 탑승했는지 여부를 나타내며. 환자를 탑승하면 true로 설정되고, 병원에 도착하여 환자를 내리면 false로 초기화된다.
assignedPerson 및 destinationHospital는 현재 할당된 환자와 목적지 병원을 저장하는 변수로, 구급차가 해당 위치로 이동할 수 있게 한다.

구급차는 환자 요청시 구급차 객체인 AssignToPerson 메서드를 호출한다. 이 메서드는 구급차의 assignedPerson 속성에 환자의 Transform을 저장하고, isMoving 플래그를 설정하여 이동 상태를 나타낸다.
환자 위치로의 이동은 MoveToPersonAndPickup 코루틴을 통해 이루어진다. NavMeshAgent를 활용하여 구급차가 환자의 위치로 이동하도록 경로를 계산한다. 이때 환자의 위치는 인도로 NavMesh에서 이동할 수 없는 영역(Not Walkable)이다. 따라서, FindClosestWalkablePosition 함수를 사용하여 환자의 위치 주변에서 가장 가까운 Walkable 지점을 찾아 구급차를 이 지점으로 이동시킨다. 이동 경로를 설정할 때에는 navMeshAgent.ResetPath()를 호출하여 이전 경로를 초기화하고 이동하기 위해 NavMeshAgent.SetDestination 메서드를 사용하여 구급차의 목적지를 설정한다. 이동 중에는 navMeshAgent.remainingDistance와 navMeshAgent.stoppingDistance를 비교하여 도착 여부를 판단한다.
구급차가 할당되면 환자의 UI 텍스트는 'Ambulance 도착 대기 중'으로 업데이트되어 환자 객체의 uiText 속성을 통해 관리된다. 또한, 구급차의 UI 텍스트를 통해 현재 위치, 목표 위치, 남은 거리를 실시간으로 표시한다.

③환자의 구급차 탑승

환자는 구급차가 도착한 Walkable 지점으로 이동하기 위해 MovePersonToAmbulance 코루틴을 수행한다. 현실적인 이동 애니메이션을 구현하지는 않았지만, 환자의 Transform.position을 구급차가 도착한 위치로 변경하여 이동한 것으로 간주한다. 이때, 약간의 대기 시간을 두어(예: WaitForSeconds(1f)) 환자가 이동하는 시간을 시뮬레이션한다. 이후, 환자의 Transform을 구급차의 Transform의 자식으로 설정하여 구급차와 함께 이동하도록 한다. 환자의 localPosition은 (0, 0, 0)으로 설정하여 구급차 내의 좌석에 적절하게 배치한다.
이후, 구급차의 hasPerson 플래그를 true로 설정하고, 환자의 UI 텍스트를 'Ambulance 탑승 중'으로 업데이트한다.

④구급차의 병원 이동 시작

환자를 탑승한 구급차는 이제 목적지 병원으로 이동하며 MoveToHospital 코루틴을 수행한다.
구급차와 환자는 부모-자식 관계로 연결되어 있으므로, 구급차의 이동에 따라 환자도 함께 이동한다. 이는 Unity의 Transform 계층 구조를 활용한 것으로, 구급차의 Transform이 이동하면 자식인 환자의 Transform도 상대적으로 이동한다.

⑤병원 도착

구급차는 선택된 병원으로 환자를 이송하며, 도착 후 환자를 입원시키는 MoveToHospital 코루틴을 수행한다.
구급차는 destinationHospital 속성에 저장된 병원의 위치를 목적지로 설정한다. 이때, 병원의 위치가 NavMesh 상에서 Not Walkable 영역에 있으므로 이를 해결하기 위해 병원 위치 주변에서 가장 가까운 Walkable 지점을 찾아 실제 목적지로 설정한다. 이때, FindClosestWalkable Position함수를 이용해 주변의 Walkable 지점을 탐색한다. 이후, 구급차의 NavMeshAgent는 설

정된 목적지까지의 경로를 계산하고 이동을 시작한다. 이동경로를 설정할때는 이전 경로의 잔여 데이터를 제거하기 위해 navMeshAgent.ResetPath()를 호출한다.
이동 중 구급차와 환자의 UI 텍스트는 지속적으로 업데이트되어 현재 상태를 표시한다. 예를 들어, “Ambulance1: Hospital2로 이동 중” 과 같은 메시지를 표시한다. 또한, 구급차의 위치와 남은 거리를 실시간으로 모니터링하여 이동 상태를 추적한다.
이후, navMeshAgent.remainingDistance와 navMeshAgent.stoppingDistance를 비교하여 구급차가 목적지에 도착했는지 판단한다. 도착 시 navMeshAgent.isStopped를 true로 설정하여 구급차의 이동을 중지한다. 또한, hasPerson 속성을 false로 설정하여 환자를 더 이상 태우고 있지 않음을 표시한다.
환자를 내려준 구급차는 ssignedPerson과 destinationHospital과 같은 속성을 null로 초기화하여 다음 환자 이송을 준비하는 대기 상태로 전환된다. 구급차의 UI 텍스트를 “Ambulance1: 대기 중“으로 업데이트하고 AmbulanceSystem 클래스의 AssignAmbulanceAfterWaiting 메서드를 호출하여 대기 목록에 있는 환자 요청이 있는지 확인한다.
대기 중인 환자가 있다면, 해당 환자에게 구급차를 할당하고 이송 절차를 다시 시작한다.

병원에 도착하면 환자는 구급차에서 내려 병원 오브젝트로 이동한다. 먼저, 환자의 Transform을 구급차에서 분리하여 parent-child 관계를 해제한다. 그 다음, 환자의 위치를 병원의 위치로 설정하여 병원에 도착했음을 시각적으로 표시한다. 이는 assignedPerson.position = destinationHospital.hospitalTransform.position으로 위치를 변경함으로써 구현된다.
마지막으로, 환자의 UI 텍스트를 “Person1: Hospital2 도착“과 같이 업데이트하여 도착 사실을 알린다.
병원은 입원 상태 처리를 위해 병원의 AdmitPerson 메서드를 호출하여 currentOccupancy를 1 증가시킨다. 새롭게 병원의 상태(예: 현재 수용 인원, 혼잡도 등)를 업데이트하고, UpdateUI 메서드를 통해 UI에 반영한다.

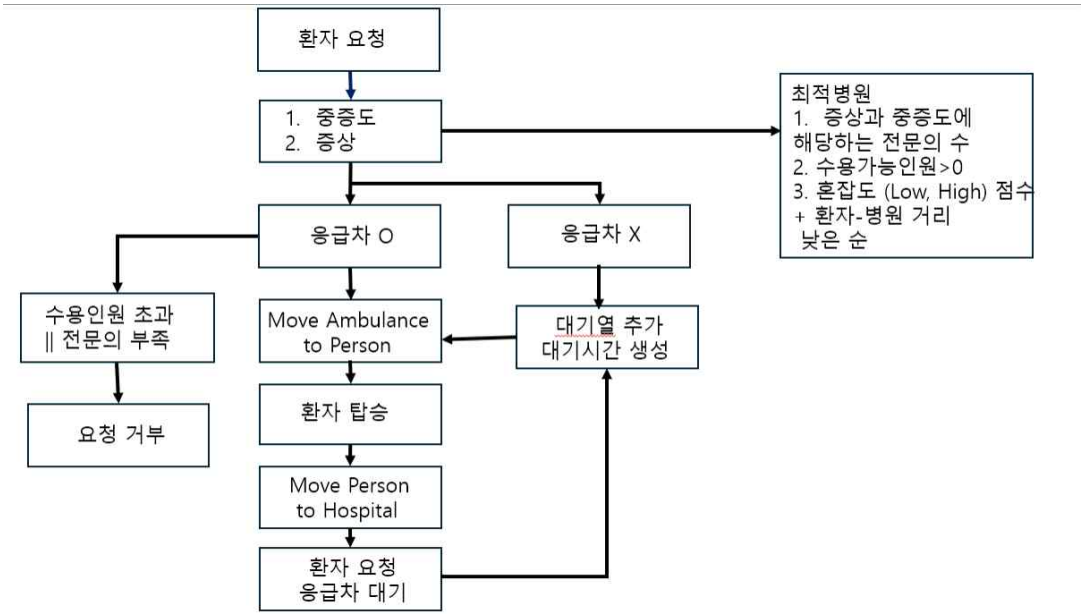

그림3. 병원 이송 시스템 다이어그램

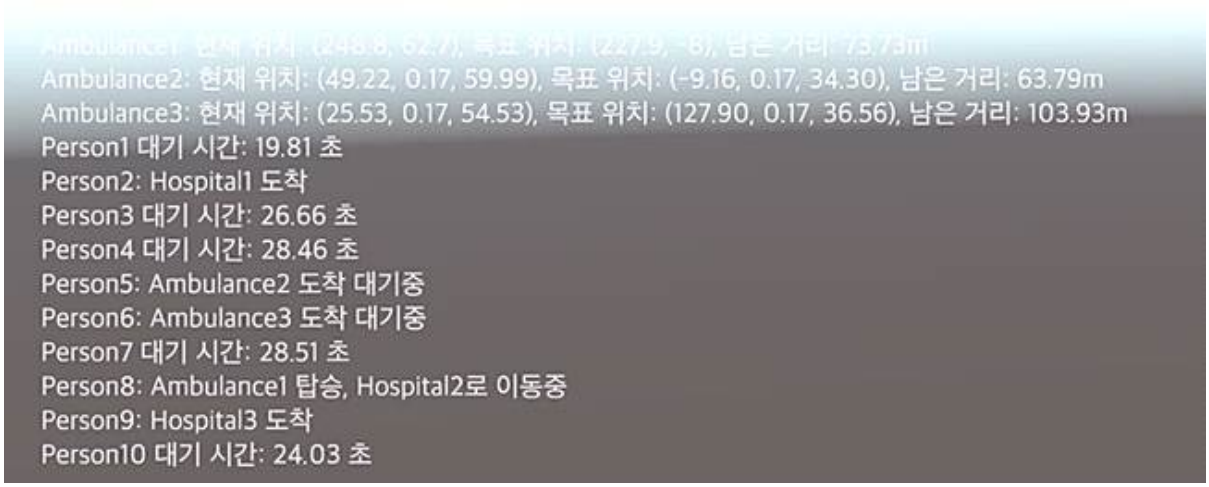

그림4. 병원 이송 시스템 UI

4) 대기 목록 관리

응급 상황에서 구급차의 수가 제한되어 모든 요청을 즉시 처리할 수 없는 경우가 발생할 수 있다. 이때, 새로운 환자 요청을 대기 목록(Queue)에 추가하고 대기시간을 생성하도록 하는 대기 목록 관리를 이용한다. 대기 목록 관리는 환자들의 대기 시간을 예측하고, 구급차가 이용 가능해질 때 효율적으로 재할당하는 것을 목표로 한다.

①대기 시간 계산

대기 중인 환자에 대해서는 현재 운행 중인 구급차가 언제 이용 가능해질지 예상하여 대기 시간을 계산한다. 구체적으로, 구급차의 현재 상태와 이동 경로, 속도를 고려하여 예상 대기 시간을 계산한다.

구급차의 현재 위치에서 예정된 목적지까지의 경로 길이를 계산한다. 예정된 목적지는 병원, 다음 환자의 위치가 될 수 있다. CalculatePathLength 함수는 먼저 NavMesh에서 이동 가능한 Walkable 영역 내의 목적지에서 가장 가까운 지점을 찾는다. 이후, NavMesh.CalculatePath 함수를 사용하여 시작점과 도착점 사이의 경로를 계산한다. 경로를 계산할때에는 모든 코너(corner) 간의 거리를 합산하여 총 경로 길이를 산출한다. 이때 경로가 유효하지 않을 경우 경로 계산에 실패했다는 에러 메시지를 남기고 walkable한 지점을 다시 산출하여 경로를 재계산한다.
경로를 계산한 후, 대기 시간 예측을 위해 경로 탐색 경우를 두가지로 나눈다. hasPerson 플래그를 활용하여 false일 경우 첫 번째 경우로 true일 경우 두 번째 경우로 나눈다.
1. 구급차가 환자를 태우러 가는 중인 경우: 구급차의 현재 위치에서 할당된 환자의 위치까지의 거리 + 환자의 위치에서 목적지 병원까지의 거리 + 병원에서 대기 중인 환자의 위치까지의 거리
2. 구급차가 환자를 태우고 병원으로 이동 중인 경우: 구급차의 현재 위치에서 목적지 병원까지의 거리 + 병원에서 대기 중인 환자의 위치까지의 거리

각 경우에서 계산된 총 경로 길이를 구급차의 이동 속도로 나누어 예상 시간을 산출한다.
예를 들어, 총 경로 길이가 10km이고 구급차의 속도가 50km/h라면, 예상 대기 시간은 12분이 된다. 예상 대기 시간은 모든 운행 중인 구급차 중 가장 짧은 시간을 지닌 구급차로 선택하여 해당 환자의 대기시간으로 설정한다.

②대기 시간 UI 업데이트

대기 중인 환자의 머리 위에는 대기 시간을 표시하는 UI가 일정 간격으로 업데이트 된다. 구체적으로, 환자가 대기 목록에 추가될 때, CreateWaitingTimeUI 함수를 통해 환자 머리 위에 FloatText 컴포넌트를 생성한다. 이후, UpdateWaitingTimeForPerson 메서드를 주기적으로 호출하여 대기 시간을 업데이트한다. 구급차의 이동 상태에 따라 실시간으로 대기 시간이 달라지므로 시스템의 Update 함수 내에서 타이머를 사용하여 일정 간격(예: 1초)에 한 번씩 대기 시간을 갱신한다. 갱신 할 때에는 각 대기 중인 환자에 대해 CalculateCurrentWaitingTime 메서드를 호출하여 현재 대기 시간을 재계산한다.
계산된 대기 시간을 환자의 uiText와 머리 위의 FloatText에 업데이트하여 사용자에게 실시간으로 정보를 제공한다. 만약 대기 시간이 0이 되거나 구급차가 이용 가능해진 경우, 해당 환자는 대기 목록에서 제거된다. 이후, 환자의 UI는 대기 시간 정보를 제거하고, 구급차가 할당되었음을 표시한다.

③구급차 이용 가능 시 재할당

구급차가 운행을 마치고 대기 상태가 되면, 시스템은 대기 목록에서 환자를 deque하여 즉시 구급차를 할당하고 출동시킨다. 대기 목록은 Queue 자료구조를 사용하여 FIFO(First-In, First-Out) 방식으로 환자 요청을 관리한다. 초기화된 구급차는 자동으로 재할당되는 AssignAmbulanceAfterWaiting 메서드가 호출되어 대기 목록에서 다음 환자를 가져온다. 가져온 환자에 대해 새로이 구급차를 할당하고, 구급차 출동 절차를 따른다.

구급차의 상태 변화(예: 운행 완료, 새로운 환자 할당 등)는 이벤트로 감지되어 대기 시간 계산에 반영되는데 이때, 여러 구급차와 환자가 동시에 상태 변화를 일으킬 수 있으므로, 대기 시간 계산과 UI 업데이트는 스레드 세이프(thread-safe)한 방식으로 처리한다.
또한, 대기 시간 계산은 계산량이 많을 수 있으므로, 업데이트 주기를 적절히 설정하여 성능을 최적화한다. 불필요한 계산을 줄이기 위해 대기 목록에 변화가 있을 때만 대기 시간을 재계산하도록 로직을 구성한다.

3.2 차선인식

도로 주행에서 차선 인식은 차량의 주행 안정성과 자율 주행의 핵심 요소 중 하나이다. 본 절에서는 Unity 엔진과 OpenCV for Unity를 활용하여 구현한 차선 인식 시스템의 구조와 동작 과정을 상세하게 설명한다. 이 시스템은 카메라 캡처부터 그레이스케일 변환(Gray scale), 관심 영역(Region of Interest, ROI) 생성, 엣지 검출(Edge Detection)에 이르는 과정을 포함하며, 각 단계에서 사용된 원리와 알고리즘을 공식과 함께 설명한다.

1) OpenCV for Unity를 이용한 차선 인식 시스템 구현

OpenCV는 컴퓨터 비전 및 이미지 처리에 널리 사용되는 오픈소스 라이브러리이며, OpenCV for Unity는 Unity 엔진에서 활용할 수 있도록 포팅한 플러그인이다. 이를 통해 Unity 환경에서 강력한 이미지 처리 기능을 구현할 수 있다.

차선 인식 시스템은 차량에 부착된 카메라로부터 영상을 캡처하고, 이미지 처리 알고리즘을 적용하여 차선을 검출한다. 주요 과정은 다음과 같다.

1. 카메라 2D 이미지 캡처
2. 그레이스케일 변환
3. 관심 영역(ROI) 설정
4. 엣지 검출
5. 허프변환
6. 전처리
7. 3D 변환

2) 카메라 2D 이미지 캡처

차량의 전방에 부착된 카메라를 통해 주기적으로 도로 이미지를 캡처한다. 이 카메라는 차량의 이동과 회전에 따라 함께 움직이며, 실시간으로 도로 상황을 촬영한다. 카메라는 초당 1프레임의 간격으로 이미지를 캡처하는데 차량의 속도와 처리 능력에 따라 조정될 수 있다. 캡쳐한 이미지는 2D로 이미지상 좌표를 활용하여 차선 인식을 실행한뒤 3D 좌표로 변환하는 과정을 수행한다. 구체적으로, RenderTexture로부터 픽셀 데이터를 읽어와 Texture2D 형식의 이미지로 저장하며 Unity의 ReadPixels 함수를 활용하여 구현한다.

그림5. Texture2D 이미지

3) 그레이스케일 변환

그레이스케일 이미지는 각 픽셀이 밝기 정보만을 포함하므로, 컬러 이미지보다 데이터의 양이 적다. 이에 따라 연산 속도가 증가하고 메모리 사용량이 감소된다. 또한, 엣지 검출과 같은 이미지 처리 알고리즘은 밝기 변화에 기반하므로 그레이스케일 이미지에서 더 효과적으로 동작한다. 따라서, 캡쳐한 컬러 이미지를 그레이스케일로 변환해 처리 속도를 향상시키고, 불필요한 색상 정보를 제거한다.

그레이스케일 변환은 컬러 이미지의 RGB 값을 하나의 명암 값으로 변환하는 과정이다. 이는 다음과 같은 공식을 따른다.

$$Y = 0.299R + 0.587R + 0.114B$$

여기서 Y는 그레이스케일 값이며, R, G, B는 각각의 색상 채널의 값이다. 이 가중치는 인간의 눈이 각 색상에 대해 느끼는 밝기의 민감도를 반영한다.

그레일스케일 변환은 먼저 Unity의 Texture2D 형식의 이미지를 OpenCV의 Mat 형식으로 바꾼다. Unity의 Texture2D 이미지는 Unity 엔진에서 사용하는 텍스처 이미지 형식이다. 이를 OpenCV의 Mat 형식으로 변환하여 OpenCV의 이미지 처리 함수들을 사용할 수 있게 한다.
그다음, OpenCV의 색상 변환 함수인 cvtColor를 사용하여 BGR(Blue, Green, Red) 형식의 이미지를 그레이스케일로 변환한다. 이때, 각 픽셀에 대해 위의 그레이스케일 공식이 적용되어, RGB 값이 단일 밝기 값 Y로 변환된다. 변환된 그레이스케일 이미지를 새로운 Mat 객체에 저장하여 이후의 처리에 사용되도록 한다.

그림6. 그레이스케일 변환 이미지

4) 관심 영역(ROI) 설정

도로 이미지에는 차선 외에도 건물, 가로등, 교통 신호등 등 다양한 객체가 포함된다. 이러한 불필요한 영역을 제외하고 차선이 있을 것으로 예상되는 부분만을 관심영역으로 처리하면 계산량을 줄이고 정확도를 향상시킬 수 있다.

ROI를 설정하기 위해 이미지와 동일한 크기의 검은색 마스크를 생성한다. 이때, 마스크는 초기값이 0인 1채널 이미지로 검은색과 흰색을 이용하여 관심영역을 구분짓는다. 이후, 차선이 위치할 것으로 예상되는 영역을 지정한다. 특정 좌표를 꼭짓점으로 하여 두 꼭짓점을 이은 선분을 이어 관심영역으로 설정한다. 예를 들어, 이미지 하단의 좌우 끝과 중앙 상단 두 부분을 연결하여 사다리꼴 형태의 영역을 만든다.

```
MatOfPoint roiPolygon = newMatOfPoint(
  newPoint(0, imgMat.height()/2+300),
  newPoint(imgMat.width() - 250, imgMat.height()),
  newPoint(imgMat.width() / 2100, imgMat.height() / 2100),
  newPoint(350, imgMat.height() / 2100)
);
```

관심 영역을 설정한 뒤, OpenCV의 fillPoly 함수를 사용하여 정의한 다각형 영역을 마스크에 흰색으로 채운다. 이로써 관심 영역은 흰색(값이 255)으로, 나머지 영역은 검은색(값이 0)으로 표시된다.

ROI 작용은 OpenCV의 bitwise_and 함수를 사용하여 마스크와 그레이스케일 이미지를 비트 단위 AND 연산한 것으로 구현된다. 마스크의 흰색 영역에 해당하는 부분만 유지되고, 검은색 영역은 제거된다.

$$MaskedImage(x,y) = GrayscaleImage(x,y) \wedge Mask(x,y)$$

5) 엣지 검출

ROI 영역에서 차선을 검출하기 위해 엣지 검출을 시행한다. 차선은 이미지에서 명확한 밝기 변화로 나타나는 선형 구조이다. 엣지 검출은 이러한 밝기 변화가 큰 부분을 찾아내어 차선 후보를 추출한다. 특히, Canny 엣지 검출은 이미지에서 가장자리를 찾는 데 널리 사용되는 알고리즘으로, 다음과 같은 단계로 이루어진다.

①노이즈 제거

이미지를 부드럽게 만들어 노이즈를 감소시키기 위해 가우시안 블러를 적용한다. 가우시안 블러는 이미지의 각 픽셀 주변의 값들을 가우시안 분포에 따라 가중 평균하여 부드러운 이미지를 생성한다. 이를 통해 작은 노이즈나 불규칙성이 제거되어 엣지 검출의 정확도를 높일 수 있다.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

여기서 G(x,y)는 가우시안 커널이며, σ는 표준 편차, x와 y는 커널 내의 좌표이다.

이미지 I에 가우시안 블러를 적용하여 노이즈를 감소시킨 과정은 다음과 같다.

$$I_{blur}(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} G(i,j) * I(x+i, y+j)$$

여기서 k는 커널의 크기와 관련된 값이다. 예를 들어, 커널 크기가 5×5이면 k=2이다.

②그라디언트 계산

노이즈가 제거된 이미지에서 밝기 변화량인 그라디언트를 계산한다. 이는 Sobel 연산자를 사용하여 수평 및 수직 방향의 밝기 변화를 구하는 방식으로 이루어진다. 각 픽셀에서 수평 방향과 수직 방향의 그라디언트를 계산하고, 엣지의 강도와 방향을 얻는다.

$$G_x = I_{blur} * S_x \, , G_y = I_{blur} * S_y \, ,$$

여기서 Gx는 수평 방향 그라디언트이고 Gy는 수직 방향 그라디언트이다. 계산 ∗는 컨볼루션 연산을 의미하고, Sobel 연산자인 Sx 와 Sy를 이용하여 커널을 계산한다.

수평 방향 커널 Sx:

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

수직 방향 커널 Sy:

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

그라디언트 크기 G와 방향 θ 계산

$$G = \sqrt{G_x^2 + G_y^2}\ , \theta = \arctan\left(\frac{G_x}{G_y}\right)$$

③비최대 억제(Non-maximum Suppression)

엣지의 정확한 위치를 찾기 위해 그라디언트 방향을 따라 인접한 픽셀들과 비교하여 국소 최대값이 아닌 픽셀을 제거한다. 각 픽셀에서 엣지의 방향을 기준으로 주변 픽셀들과 비교하여, 현재 픽셀이 국소 최대값인 경우에만 엣지로 유지하고 그렇지 않으면 제거한다. 결과적으로, 엣지의 두께를 1픽셀로 얇게 만들어 정확한 위치를 파악할 수 있게 한다.

구체적으로, 두 개의 임계값을 설정하여 강한 엣지와 약한 엣지를 구분하는 이력 임계값 처리(Hysteresis Thresholding)를 한다. 엣지 강도가 상한 임계값보다 큰 픽셀은 강한 엣지로 간주하고, 하한 임계값과 상한 임계값 사이에 있는 픽셀은 약한 엣지로 간주한다. 약한 엣지 중에서 강한 엣지와 연결되어 있는 픽셀만을 최종 엣지로 인정하고, 그렇지 않은 픽셀은 제거한다. 이를 통해 노이즈로 인한 거짓 엣지를 제거하고 실제 엣지만을 남길 수 있다.

상한 임계값 T_{high} 와 하한 임계값 T_{low}설정

$T_{low} = 0.4 * T_{high}$로 설정한다. T_{high}= 200이면 T_{low}=80이 된다.

강한 엣지와 약한 엣지 결정

강한 엣지: $G \geq T_{high}$

약한 엣지: $T_{low} \leq G < T_{high}$

비엣지: $G < T_{low}$

약한 엣지가 강한 엣지와 8-연결성을 통해 연결되어 있으면 엣지로 간주하고, 그렇지 않으면 제거한다.

최종적으로 다음과 같은 코드를 활용한다.

Imgproc.GaussianBlur(maskedImage, edges, new Size(5, 5), 0);

Imgproc.Canny(edges, edges, 100, 200);

그 결과, 차선과 같은 선형 구조가 강조된 이진 엣지 이미지를 얻는다.

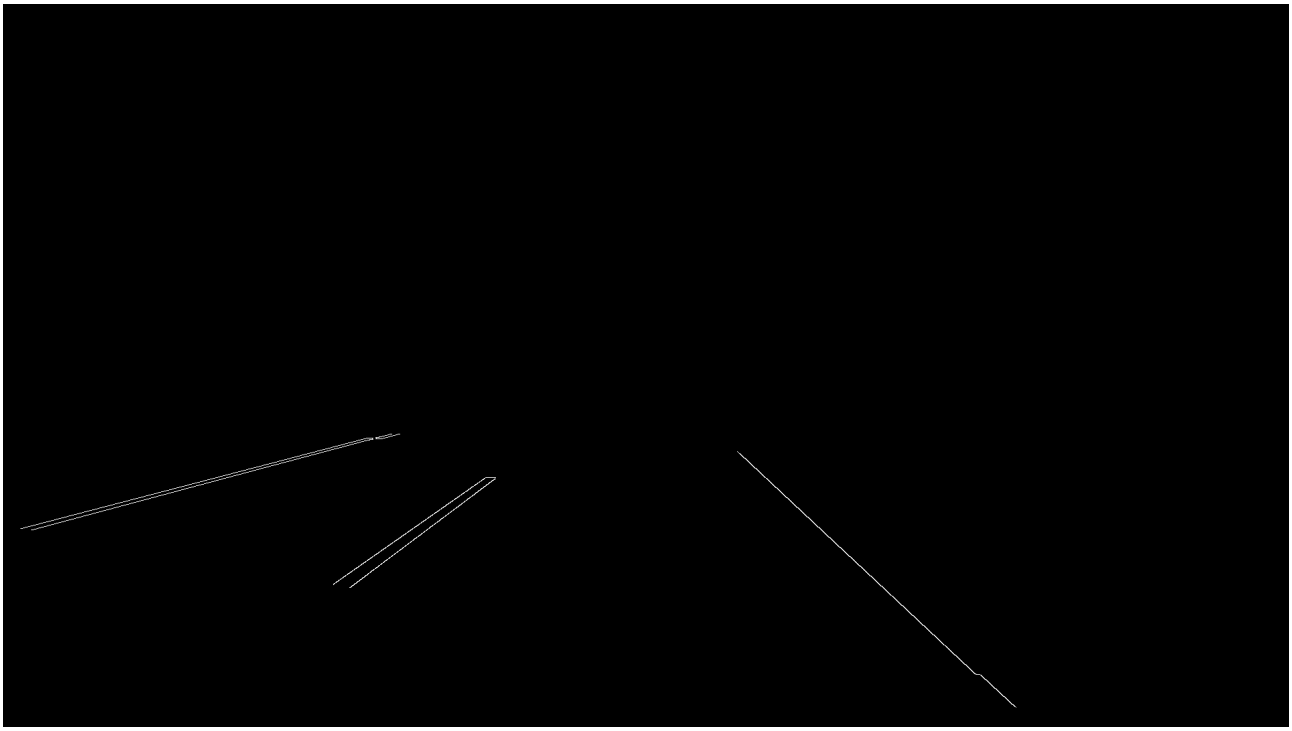

그림7. ROI 설정 후 엣지 검출 이미지

6) 허프 변환

허프 변환은 이미지에서 직선을 검출하는 강력한 방법으로, 엣지 이미지에서 연속적인 선형 패턴을 찾아낸다. 특히 차선 인식에서 허프 변환은 도로의 차선을 구성하는 직선을 검출하는 강력한 도구로써 작용한다.

허프 변환은 이미지의 픽셀 공간에서 선들을 선형 방정식으로 변환하여 직선을 찾는다.

직선의 일반적인 방정식은 다음과 같다.

$$y = mx + b$$

위와 같이 일반적인 방정식은 기울기 m이 무한대가 되는 수직선의 경우를 처리하기 어렵기 때문에 차선인식에서는 직선을 극좌표로 표현한 공식을 사용한다.

$$\rho = x\cos\theta + y\sin\theta$$

여기서 ρ는 원점에서 직선까지의 수직 거리, θ는 직선의 법선 벡터와 x축 사이의 각도이다.

엣지 이미지의 각 흰색 픽셀에 대해 가능한 모든 θ 값을 대입하여 해당하는 ρ 값을 계산하고, (ρ, θ) 공간에서 누적 히스토그램을 만든다. 누적값이 큰 (ρ, θ) 조합은 이미지에서 직선이 존재한다는 것을 의미한다.

허프 변환의 파라미터는 다음과 같이 설정된다.
거리 해상도 (ρ): 누적 히스토그램의 거리 축의 해상도. 일반적으로 1로 설정하여 모든 픽셀 단위 거리를 고려한다.
각도 해상도 (θ): 각도 축의 해상도. π/180으로 설정하여 1도 단위로 각도를 고려한다.
임계값 (threshold): 누적 히스토그램에서 직선으로 인정되기 위한 최소 투표수이다.
최소 선 길이 (minLineLength): 검출할 선의 최소 길이이다.
최대 선 간격 (maxLineGap): 선분 사이의 최대 허용 간격으로, 이 간격 이내에 있는 선분들은 하나의 선으로 간주된다.

실제로, 엣지 검출 결과 이미지를 입력으로 하여 허프 변환을 수행한 과정은 다음과 같다.
Imgproc.HoughLinesP(edges, lines, 1, Mathf.PI / 180, 100, 150, 30);
거리 해상도: 1 픽셀, 각도 해상도: 1도 (π/180라디안), 임계값: 100 , 최소 선 길이: 150 픽셀 최대 선 간격: 30 픽셀
이후 허프 변환 결과로 얻은 선분들을 lines객체에 각각 저장한다.
검출된 선분의 개수를 확인하고, 선분이 검출되지 않았을 경우에는 차선 검출 프로세스를 중단한다.

7) 선처리

차선 인식 시스템의 성능을 향상시키기 위해서는 이미지 처리 과정이 매우 중요하다.
이 단계에서는 허프 변환으로 검출된 선들을 linesegment 객체로 변환하여 각도를 기준으로 불필요한 노이즈를 제거하고 선을 병합한다. 이후 각도를 기준으로 그룹화하여 중앙선, 대시라인, 통행불가선을 분류한다.

①선분 정보 추출 및 필터링
검출된 각 선분의 시작점(x1,y1)과 끝점 좌표(x2,y2)를 추출하여 LineSegment 객체를 생성한다. 이후, 각 선분에 해당하는 길이와 각도를 계산한다. ROI로 설정된 영역에서 각도가 30보다 작은 양수의 경우, 차선과 수직인 선들이므로 위 경우를 제거한 선들을 allLines라는 새로운 리스트에 추가한다.

②선분 병합
허프 변환을 통해 검출된 선분들은 종종 짧고 끊겨 있거나 중복되는 경우가 많다. 이러한 선분들을 하나의 선으로 병합하고, 도로의 차선에 해당하는 선으로 분류하기 위해 각도와 거리를 이용해 분류한다.
allLines 리스트에 추가된 선들 중에서 선분의 각도 차이가 1도이하이거나 선분의 시작점의 거리 차이가 50 이하, 끝점의 거리 차이가 50 이하는 선분들을 한 선분으로 병합한다. 이렇게 병합된 선들은 mergedLines에 추가한 뒤 mergedLines에 포함된 선들과 병합되지 않고 allLines에 남아있는 선들을 합쳐서 allFinalLines라는 최종 선분 리스트를 구성한다.

③선분 분류
allFinalLines 리스트에 있는 선분들을 분류하기 위해 ProcessandClassifyLanes 함수를 이용한다. 이 함수는 중앙선, 대시라인, 통행불가선 으로 분류하여 도로의 차선을 정확히 인식하도록 한다.
ProcessandClassifyLanes 함수는 각도를 기반으로 그룹화한 뒤 각 그룹에서 가장 빈도수가 많은 선분들을 다시 그룹화하고 그중 가장 긴 선분을 그 그룹의 대표 선분으로 선택한다.
그룹으로 계속 묶는 까닭은 허프 변환된 선분들이 비슷한 각도로 여러 선분으로 나누어져서 연속적으로 그룹화 하지 않으면 같은 선으로 인식하지 않기 때문이다.
따라서, 각도 차이가 20도 이내인 선분들을 먼저 그룹으로 묶고 각 그룹에서 각도 차이가 2도 이내인 선분들을 세부 그룹으로 또 묶는다. 마지막으로 세부 그룹 중 각도의 빈도수가 가장 많은 그룹을 묶어 추출한다.

각도의 빈도수가 가장 많은 그룹으로 묶었을 때 최종적으로 3개의 그룹이 추출되어야 한다.
3개보다 적은 그룹이 추출되면 일반 도로에 있지 않다고 가정하고 선들을 추출하지 않는다.
반대로, 3개보다 많은 그룹이 추출되면 부정확한 선들이 추출된 가능성이 있다고 판단하고 그룹들 중 각도 차이가 적은 두 그룹을 한 그룹으로 묶는다.

그다음 대표선분들로 추출된 선들은 각도 순으로 정렬한다. 위 이미지에서는 각도가 가장 작은 선이 중앙에 위치한 선으로 나머지 두 선들은 길이로 분류한다.
Lane 1. 중앙에 위치한 선이 나머지 두 선보다 작을 경우, 중앙에 위치한 선을 대시라인으로 정의하고 나머지 두 라인 중 가장 긴 라인을 통행 불가선으로, 그다음 긴 라인을 중앙선으로 정의한다.
Lane 2. 중앙에 위치한 선이 나머지 두 선보다 클 경우, 중앙에 위치한 선을 중앙선으로 정의하고 나머지 두 라인을 대시라인 1과 2로 정의한다.

분류된 선들은 알아보기 쉽게 각기 다른 색상으로 그린다. 위 이미지에서는 중앙선을 초록색으로, 대시라인은 빨간색으로, 통행불가선은 파란색으로 나타내었다.
허프변환과 선처리 과정을 거친 최종 이미지는 다음과 같다.

그림8. Lane1 이미지

그림9. Lane2 이미지

8) 3D 변환

이미지 좌표계의 선분을 3D 월드 좌표계로 변환하여 실제 환경에서의 위치를 파악한다.

①픽셀 좌표를 뷰포트 좌표로 변환

픽셀 좌표 (x,y)를 이미지의 너비와 높이로 나누어 0에서 1 사이의 뷰포트 좌표로 정규화한다.

$$normalized X = \frac{x}{\text{이미지 너비}}$$

$$normalized Y = \frac{y}{\text{이미지 높이}}$$

②카메라에서 도로까지의 거리 계산

카메라 위치에서 아래 방향으로 레이를 발사하는 레이캐스팅을 수행한다. 이를 통해 도로와의 충돌 여부를 확인하고, 충돌 지점까지의 거리를 계산한다.

③뷰포트 좌표를 월드 좌표로 변환

Unity의 ViewportToWorldPoint 함수를 사용하여 뷰포트 좌표와 도로까지의 거리를 기반으로 3D 월드 좌표로 변환한다.

worldPoint=Camera.ViewportToWorldPoint((1−normalizedX,0.5,normalizedY×distanceToRoad))

전체적인 과정은 그림 8과 같이 나타낼 수 있다.

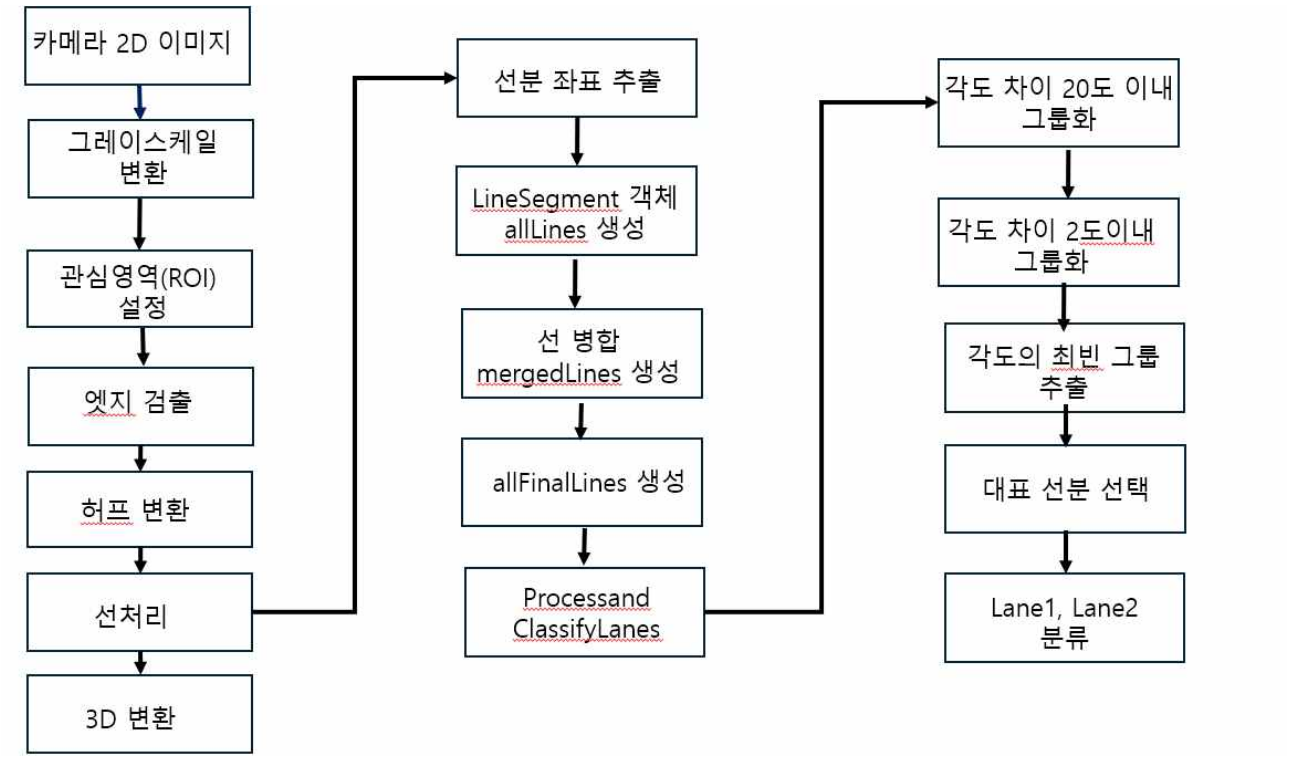

그림10. 차선 인식 다이어그램

변환한 좌표는 중앙선, 대시라인, 통행불가선의 시작점과 끝점의 좌표를 나타낼 수 있다. 이 좌표로 Lane1과 Lane2를 정의하고 모든 차량은 Lane1과 Lane2에 위치하도록 한다. 만약 Lane1과 Lane2에 위치하지 않는다고 파악되면 Lane1과 Lane2에 위치하도록 차량의 위치를 조정해 준다.
MoveVehicleToLane 함수는 먼저 중앙선을 밟고 있을때와 아닐때를 나눈다. 모든 차량은 중앙선을 넘어가면 안되므로 중앙선의 좌표와 0.1f거리 이내에 있으면 중앙선에 있다고 판단하고 Lane2에 위치하도록 한다. 중앙선에 위치하지 않은 차량은 Lane2와 Lane1중 하나에 있다고 가정하고 각 라인에 제대로 위치해 있는지를 확인한다. 차량이 대시라인 좌표와 0.1f거리 이내에 있으면 대시라인에 있다고 판단하고 Lane2와 Lane1중 하나를 골라 위치하도록 한다.
구체적으로, Lane2에 위치하기 위해서는 차량의 왼쪽좌표가 중앙선 좌표보다 크고 차량의 오른쪽 좌표가 대시라인 좌표보다 작도록 한다. 반면, Lane1에 위치하기 위해서는 차량의 왼쪽좌표가 대시라인 좌표보다 크고 차량의 오른쪽 좌표가 통행불가선 좌표보다 작도록 한다.

3.3 충돌 회피 알고리즘

충돌 회피 알고리즘은 차량과 앰뷸런스의 움직임을 관리하여 교통 흐름을 안전하게 하는 것을 목표로 한다. 차량간의 거리와 차량들의 위치 배열을 고려하여 속도를 조정하거나 차선을 변경하도록 한다.
충돌 회피는 크게 두 부류로 나뉘는데 첫 번째는 응급차 기준이고 두 번째는 일반 차량 기준이다.

3.3.1 응급차 기준 충돌 회피 알고리즘

응급차 기준 충돌 회피는 3가지 경우로 나눌 수 있다.
응급차 앞차가 있거나 없는 경우, 응급차 옆차가 있거나 없는 경우, 응급차 앞차의 앞에 차가 있거나 없는 경우이다.
세부적으로 다음과 같은 과정을 거친다.
1.응급차 앞에 차가 없다면 응급차는 같은 차선을 유지하며 속도를 증가시킨다.
2.응급차 앞에 차가 있다면 옆차가 있는지 확인한다.
3.옆차가 없다면 응급차가 옆차선으로 이동하고 옆차가 있다면 응급차 앞차의 앞을 확인한다.
4.응급차 앞차의 앞에 차가 있다면 원래의 차선과 속도를 유지한다.
5.응급차 앞차의 앞에 차가 없다면 앞차의 속도를 증가시켜서 옆차선으로 이동하도록 하고 응급차는 같은 차선을 유지하며 속도를 증가시킨다.

①응급차 앞차 판단
응급차 앞에 차가 있는지 없는지 확인하는 BlockingVehicle 함수를 실행한다. BlockingVehicle 함수는 unity에서 제공하는 raycast로 탐지한다. 차량의 정중앙 앞에서 전방으로 raycast를 발사하여 20m내에 다른 차량이 있는지를 확인한다.
응급차 앞에 차가 없다면 응급차는 같은 차선을 유지하며 속도를 증가시킨다.

SetVehicleSpeed는 응급차의 속도를 증가시키거나 감소시키는데 위 경우 차량의 속도를 기본속도(5f)보다 증가시킨 7f로 설정한다.

②응급차 옆차 판단
응급차 앞에 차가 있다면 응급차 옆차를 확인하는 AdjacentVehicle 함수를 실행한다. AdjacentVehicle 함수는 차량의 현재 차선을 기준으로 작동한다. 이 함수는 unity에서 제공하는 hit Colliders의 Physics.OverlaSphere를 이용하여 차량의 위치를 중심으로 구 형태의 탐지 영역을 생성한다. 반지름 20미터내의 영역을 120도의 시야각으로 탐지하도록 설정한다. 부채꼴 형태의 탐지영역으로 설정하면 응급차 차량의 정 옆에 위치하지 않아도 옆에 있다고 판단할 수 있어 보다 정확한 판단을 할 수 있다.
차량이 Lane1에 있다면 차량의 왼쪽 기준으로 설정한 탐지영역으로 차량을 탐지하고 차량이 Lane2에 있다면 차량의 오른쪽 기준으로 설정한 탐지영역으로 차량을 탐지하도록 한다.
응급차 옆에 차가 없다면 응급차는 ChangeLane함수를 이용하여 옆차선으로 이동한다. ChangeLane 함수 또한 차량의 현재 차선을 기준으로 작동하는데 현재 Lane1에 있다면 Lane2로, 현재 Lane2에 있다면 Lane1로 이동시킨다. Lane1로 또는 Lane2로 바꾸는 과정은 3.2 차선 인식에서 정의하고 분류한 기준에 따른다.

③응급차 앞차의 앞 판단
응급차 옆에 차가 있다면 응급차 앞차의 앞을 확인하는 BlockingVehicle을 수행한다. 응급차 앞차를 판단할때와 같은 함수로 이번에는 응급차 앞차를 현재 차량으로 놓고 BlockingVehicle을 수행한다.
응급차 앞차의 앞에 차량이 있을 경우 응급차는 특별한 행동을 수행하지 않고 원래의 속도와 차선을 유지한다.
응급차 앞차의 앞에 차량이 없을 경우 응금차 앞차의 속도를 증가시키는 SetVehicleSpeed를 수행한뒤 옆차선으로 이동하는 ChangeLane을 수행한다. 이때, 속도를 증가시켜서 이동했음에도 응급차 옆차와 부딪힐 수 있으므로 옆차선으로 이동한뒤 응급차 옆차와의 거리를 측정해 부딪히지 않을 만큼의 속도를 증가시킨다. Adjust 함수는 이동한 응급차 앞차의 정중앙뒤 와 응급차 옆차의 정중앙 앞을 3DVector를 이용해 측정한뒤 20미터 이내에 있다고 판단하면 20미터 이내를 벗어날때까지 이동한 응급차 앞차의 속도를 증가시킨다.

그림11. 응급차 충돌 회피(차선 변경)

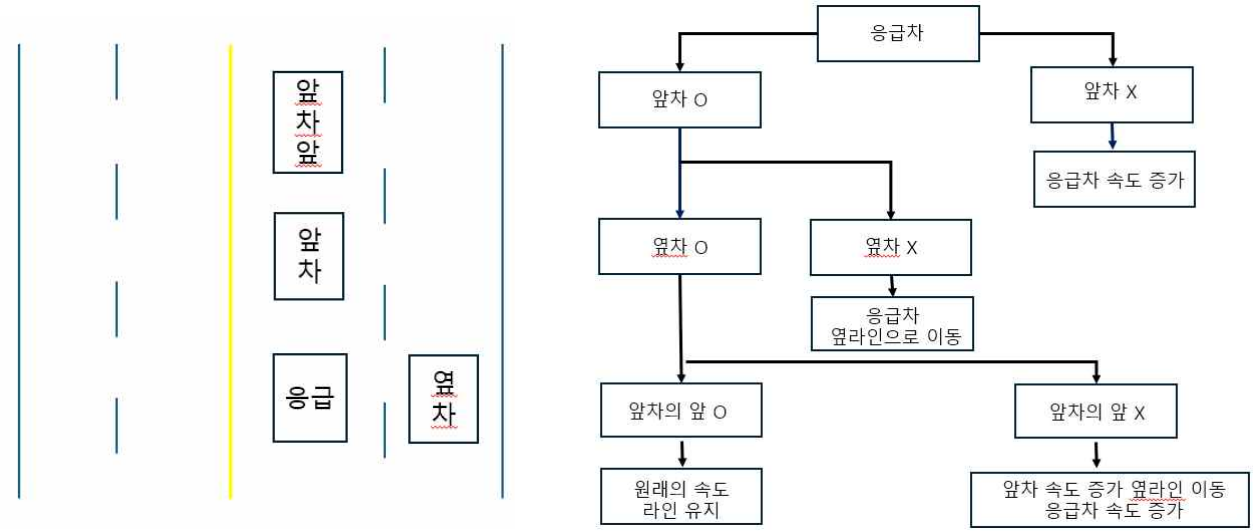

그림12. 응급차 기준 충돌 회피 다이어그램

3.3.2 일반 차량 기준 충돌 회피 알고리즘

일반 차량 기준 충돌 회피는 2가지 경우로 나눌 수 있다.
일반 차량 앞차가 있거나, 일반 차량 옆차가 있거나 없는 경우 없는 경우로 나뉜다.
세부적으로 다음과 같은 과정을 거친다.
1.일반 차량 앞에 차가 없다면 현재의 차선과 속도를 유지한다.
2.일반 차량 앞에 차가 있다면 옆차가 있는지 확인한다.
3.옆차가 없다면 일반차량은 옆차선으로 이동한다.
4.옆차가 있다면 현재의 차선과 속도를 유지한다.

①일반 차량 앞차 판단
일반차량 앞에 차가 있는지 없는지 확인하는 BlockingVehicle 함수를 실행한다. 이 함수는 응급차 앞에 차가 있는지 판단할때와 같은 함수로 같은 동작을 수행한다.
일반 차량 앞에 차가 없다면 특별한 행동을 수행하지 않고 현재의 차선과 속도를 유지한다.

②일반차량 옆차 판단
일반 차량 앞에 차가 있다면 옆차가 있는지 확인하는 AdjacentVehicle 함수를 실행한다.
마찬가지로, 응급차 옆에 차가 있는지 판단할때와 같은 함수로 같은 동작을 수행한다.
옆차가 없다면 일반차량은 ChangeLane 함수를 이용하여 옆차선으로 이동한다.
옆차가 있다면 특별한 행동을 수행하지 않고 현재의 차선과 속도를 유지한다.

4장 결론 및 고찰

본 연구는 한국 응급의료 시스템의 구조적 문제를 해결하기 위해 Unity 기반의 가상 병원 이송 시스템을 설계하고 구현했다. 해당 시스템은 독일 병원 이송 시스템의 사례를 참고하여, 효율적인 병원 선택 알고리즘과 자율주행 응급차, 그리고 충돌 회피 알고리즘을 포함한 통합 솔루션을 제안했다. 이를 통해 응급상황에서 환자의 생존율을 높이고, 병원 및 구급차의 자원을 최적화하는 방안을 제시했다.

가상 도시 맵을 구현하여 응급차, 병원, 사람 등 다양한 시뮬레이션 요소를 통합했다. NavMesh를 이용해 차량의 이동 가능 구역과 불가능 구역을 정의하여 현실적인 주행 환경을 재현했다. 이러한 가상 환경은 의료 시스템 개선을 위한 효과적인 실험 플랫폼으로 활용될 수 있다.
병원 이송 알고리즘은 환자의 요청부터 병원 선택, 구급차의 환자 이송, 대기시간을 관리하며 병원 자원의 효율적인 배분을 가능하게 했다. 환자의 특성을 고려한 병원 선택 알고리즘을 구현하고 거리 비교를 통한 구급차의 최소화된 이동으로 골든타임을 지킬 수 있도록 했다. 이후 요청한 환자들의 대기시간을 업데이트하고 경로를 재조정해 혼잡도를 줄이고 체계적인 이동을 구현했다.

OpenCV for Unity를 활용하여 구현한 차선 인식 시스템은 도로 주행 안정성을 확보하고, 차량의 위치를 실시간으로 조정하여 안전한 주행을 가능하게 했다. 그레이스케일 변환, 엣지 검출, 허프 변환과 선처리를 통해 중앙선, 대시라인, 통행 불가선을 정확히 검출하였으며 차량의 주행 경로를 효과적으로 관리했다.
응급차와 일반 차량의 이동을 관리하기 위한 충돌 회피 알고리즘은 교통 흐름을 유지하며, 긴급 상황에서 차량 간 충돌을 방지하는 데 중요한 역할을 한다. 특히, 응급차는 주변 차량의 위치를 실시간으로 탐지하여 최적의 경로를 선택하고, 속도와 차선을 유연하게 조정함으로써 안전하면서도 가능한 빠른 이동을 가능케 했다.

본 연구에서 제안한 시스템은 가상 환경을 통한 다양한 의료 시나리오를 테스트하고, 정책이나 기술 개선의 효과를 검증하는 데 유용한 도구로 활용될 수 있으며, 이는 응급 의료 체계의 디지털화를 가속화하는 데 기여할 것이다. 더불어, 본 연구에서 개발된 자율주행 응급차와 충돌 회피 알고리즘은 향후 자율주행 차량의 안전성과 효율성을 개선하는 데 중요한 기술적 기반을 제공한다. 또한 응급차뿐만 아니라 소방차, 경찰차 등 다른 긴급 차량의 이동 시스템 개발에도 응용 가능하다.

한편, 본 연구는 Unity 기반의 가상 환경에서 구현된 병원 이송 시스템과 알고리즘 설계에 중점을 두었으나, 실제 환경에서의 모든 상황을 포괄적으로 반영하기에는 한계가 있다. 가상 환경은 도로 주행 및 응급 상황을 재현하는 데 유용하지만, 실제 환경에서 발생할 수 있는 다양한 변수들을 완전히 구현하지는 못한다.

예를 들어, 본 연구에서 가상 환경은 환자의 증상 유형과 중증도에 따라 8가지 주요 의료진과의 매칭을 비교적 단순화된 방식으로 구현했다. 그러나 실제 환경에서는 환자의 증상이 단일 유형으로 분류되지 않고, 여러 증상이 복합적으로 나타나는 경우가 흔하다. 이러한 복합 증상은 의료진의 판단을 더욱 어렵게 만들며, 필요한 전문의를 정확히 파악하고 최적의 병원을 선택하는 과정이 훨씬 더 복잡할 것이다. 또한, 실제 병원에서는 의료진의 실시간 진료 가능 여부, 병실 수용 상태, 특정 진료 과목의 우선순위 등 다양한 요인이 추가적으로 고려되어야 한다.

더 나아가, 현실 세계에서 환자, 응급차, 병원의 수는 가상 환경에 비해 훨씬 많고, 이로 인해 병원 이송 시스템이 판단하고 계산해야 할 요소들이 급격히 증가한다. 예를 들어, 동일한 지역에서 여러 환자의 요청이 동시다발적으로 발생하거나, 응급차와 병원의 수용이 제한적인 경우, 시스템은 보다 정교한 최적화 알고리즘과 병렬 처리 능력을 요구받는다. 이러한 상황에서는 단순한 거리와 혼잡도 기반의 병원 선택 기준으로는 충분하지 않으며, 우선순위와 긴급도를 종합적으로 판단하는 고급 의사결정 모델이 필요하다.

또한, 모든 병원과 응급차 정보를 통합적으로 관리하는 중앙 시스템의 구축은 기술적, 조직적 어려움을 동반한다. 각 병원은 자체적인 운영 시스템과 데이터베이스를 보유하고 있으며, 이를 하나의 중앙 시스템과 실시간으로 연동하는 것은 기술적 통합성뿐만 아니라 병원 간의 협력과 정보 공유 체계를 전제로 한다. 특히, 개인정보보호와 같은 법적 제약사항도 고려해야 하며, 데이터의 신뢰성과 보안성을 보장하는 시스템 아키텍처 설계가 필수적이다.

본 연구에서 설계한 차선 인식 알고리즘과 충돌 회피 알고리즘은 이상적인 도로 환경에 적합하도록 구현했다. 하지만, 실제 도로 환경은 단순히 직선과 곡선으로 이루어진 차선뿐만 아니라, 차선이 희미하거나 파손된 구간, 차선이 명확하지 않은 비포장 도로나 좁은 도로와 같은 복잡한 상황을 포함한다.

또한, 충돌 회피 알고리즘은 단순히 차량 간의 거리나 차선 변경 조건만 고려했지만, 실제 환경에서는 교차로, 신호등, 보행자, 자전거, 그리고 도로 위의 갑작스러운 장애물 등 다양한 요소들이 존재한다. 구체적으로, 교차로에서의 차량 우선순위 결정, 신호등 기반의 움직임 제어, 그리고 보행자의 불규칙한 이동을 감지하고 대응하는 기능이 필요하다. 이러한 요소들은 자율주행 차량이 도로에서의 의사 결정을 더 복잡하게 만들며, 본 연구에서 구현한 알고리즘보다 더 높은 수준의 정교함과 복잡성을 요구한다.

추가적으로, 가상 환경은 실제 도로에서 발생할 수 있는 사고 상황을 완벽히 재현하기 어렵다. 예를 들어, 차량 간의 예상치 못한 충돌 현실 세계에서 자율주행 응급차의 안전성과 효과성에 중요한 영향을 미친다.

현실 환경에서 본 연구의 알고리즘과 시스템을 적용하기 위해서는, 가상 환경에서 사용된 도구들보다 훨씬 더 정교한 기술이 요구된다. 고정밀 지도(HD Maps), 다중 센서 융합 기술(LiDAR, Radar, Camera 등), 그리고 강화된 인공지능 기반 의사 결정 알고리즘은 현실 세계에서의 복잡한 상황을 처리하기 위해 필수적이다. 아울러, 실제 차량 데이터를 활용한 시뮬레이션과 필드 테스트를 통해 시스템의 안정성과 정확성을 검증하는 작업이 중요하다. 이러한 작업은 단순히 알고리즘의 기능적 구현뿐만 아니라, 도로교통법 준수 여부와 긴급 상황 대응 능력에 대한 평가를 포함해야 한다.

마지막으로, 실제 환경에서의 적용을 위해서는 안전성 테스트가 다방면에서 수행되어야 한다. 이는 차량이 다양한 시나리오에서 안정적으로 작동할 수 있는지 확인하는 과정을 포함한다. 예를 들어, 차량의 급정거, 급가속, 차선 병합, 그리고 차량 간 통신을 기반으로 한 협조 운전 등이 현실적인 시나리오로 테스트되어야 한다. 또한, 이러한 시스템이 도로 위의 다른 사용자(예: 보행자, 자전거, 일반 차량)와 상호작용할 때 안전성과 신뢰성을 유지할 수 있는지 평가하는 것도 중요하다.

따라서, 본 연구는 Unity 기반의 가상 환경을 활용하여 응급 의료 시스템의 구조적 문제를 해결하기 위한 가능성을 탐구한 데 의의가 있으나, 실제 환경에서 적용 가능성을 높이기 위해서는 앞서 언급한 복잡한 변수들을 고려한 확장 연구와 정밀한 테스트가 필요하다. 이러한 노력은 궁극적으로 한국 응급 의료 체계의 디지털 전환과 선진화를 실현하며, 글로벌 표준화된 응급 이송 시스템의 구축에 기여할 수 있을 것이다.

5장 참고문헌

Malteser Hilfsdienst, Medical rescue sevices in Germany, ETSC Round Table, May 3rd 2018

이정호1・신동민2, 119 구급대 구급차 교통사고 현황 분석 1경기도 용인소방서, 2한국교통대학교 응급구조학과 , 한국응급구조학회지 제 22 권 제 1 호, 35~ 47 (2018. 04)

NICK DAVIS, Autonomous Driving with Unity and BMW: 240 Million Virtual Kilometers , Unity Technologies, August 8th 2020

Mainak Kumar Das, Self-Driving Ambulance for Emergency Application, 2021 5th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech), September 24th – 26th 2021