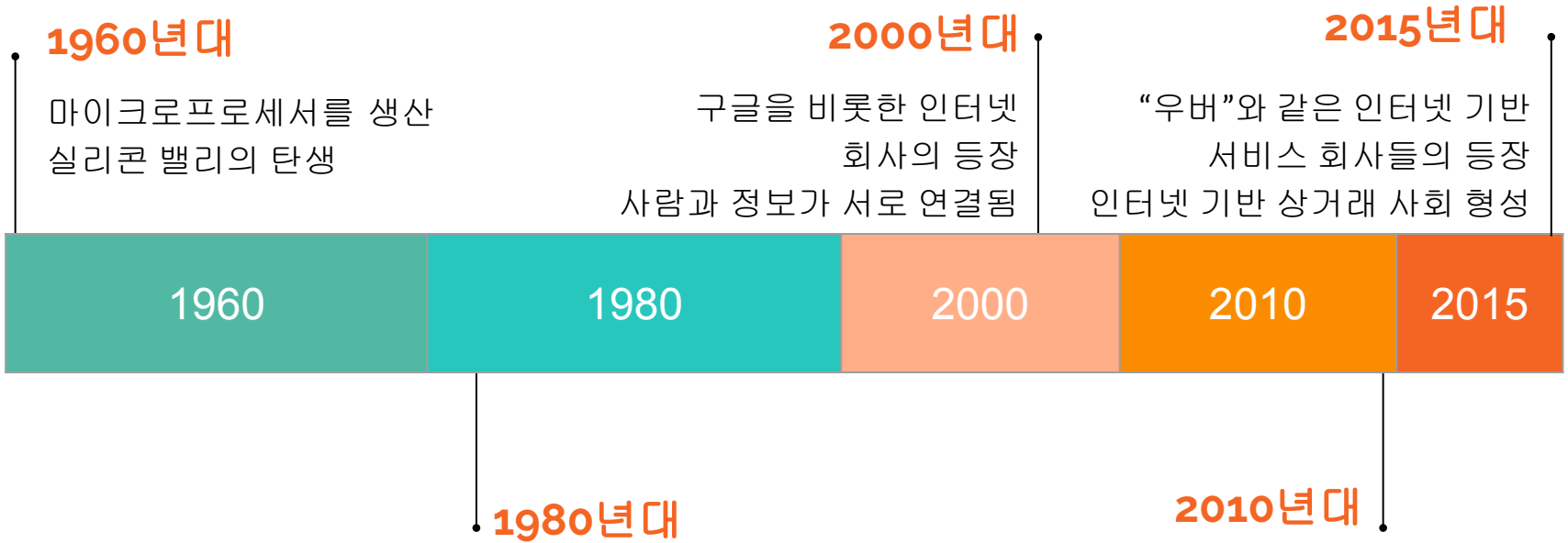
자율 주행 자동차 만들기

1장 1.1~1.3

정보통신공학과 2016013156 강창수

1. 목차

자율 주행 개요


1960년대
마이크로프로세서를 생산
실리콘 밸리의 탄생
2000년대
구글을 비롯한 인터넷
회사의 등장
사람과 정보가 서로 연결됨
2015년대
“우버”와 같은 인터넷 기반
서비스 회사들의 등장
인터넷 기반 상거래 사회 형성
1960
1980
2000
2010
2015
1980년대
마이크로소프트와 애플이
GUI를 도입
PC보급이 실현
2010년대
소셜 네트워크 회사의 등장
오프라인 세상과
인터넷의 경계가 무너짐

자율 주행 기술의 개요

- **알고리즘:** 센서로 수집한 raw(원본, 미가공) 데이터로부터 의미 있는 정보를 추출해 주변 환경을 파악하고 다음 동작을 결정한다.

- **클라이언트 시스템:** OS와 하드웨어 플랫폼으로 구성된 클라이언트 시스템이다. 실시간 및 신뢰성 요구사항을 만족하도록 여러 알고리즘을 서로 엮는다.

  ex) 카메라에서 60Hz 주기로 생성하는 데이터를 제때 처리하려면 클라이언트 시스템의 프로세싱 파이프라인에서 가장 긴 잡업도 최대 16ms 이내에 완료해야 함

- **클라우드 플랫폼:** 자율주행차에 필요한 오프라인 컴퓨팅 및 스토리지 기능을 제공한다.

  새로운 알고리즘을 테스트, HD 맵을 업데이트, 인지와 추적 및 의사결정 모델에 대한 트레이닝 수준을 높이는데 활용

자율 주행 알고리즘

감지(sensing), 인지(perception), 의사결정(decision)을 처리하는 알고리즘

감지: 수집한 미가공 원본 데이터로부터 의미 있는 정보를 추출

ex) GPS/IMU, 라이다(LiDAR), 카메라

인지: 차량의 위치에 대한 위치 측정(localization)을 통해 주변 환경을 파악

ex) 로컬라이제이션, 개체 인지, 개체 추적

의사결정: 목표 지점까지 안전하고 안정적으로 도달하는 데 필요한 동작을 결정

ex) 경로 계획, 동작 예측, 장애물 회피

Sensing

GPS/IMU: GPS와 IMU는 자율주행차의 위치를 파악하는데 필요하다.

GPS: 로컬라이제이션 센서 중에서도 상당히 정확한 편이지만 업데이트 주기가 느림

IMU: 시간의 흐름에 따른 누적 오차로 인해 정확도가 갈수록 떨어지는 대신 업데이트 주기가 매우 짧음

이 둘을 잘 조합하면 차량 위치에 대한 정확한 정보를 실시간(200Hz 이상의 주기)으로 업데이트 할 수 있다.

라이다(LiDAR): 라이다는 매핑, 로컬라이제이션, 장애물 회피 등에 사용된다. 레이저 빔을 표면에 쏴서 반사될 때까지 걸리는 시간을 재는 방식으로 거리를 측정한다.

정확도가 높기 때문에 HD 맵을 생성해서 이동 중인 자동차의 위치를 파악해 추정(로컬라이제이션)한 뒤, 앞에 나타난 장애물을 감지하는 작업에 주로 활용한다.

ex) 벨로다인 6빔 레이저: 10Hz 주기로 회전하면서 측정 값을 초당 130만 번 읽음

카메라: 카메라는 주로 차로 감지, 신호등 감지, 보행자 감지 등과 같은 개체 인지 및 추적 작업에 활용한다. 2018~2019년 기준 개발된 자율 주행차는 안전성을 높이기 위해 1080p 카메라를 8개 이상 장착한다.

레이더와 소나(sonar): 레이더와 소나(초음파 탐지) 시스템은 주로 장애물 회피를 위한 최후의 수단으로 사용한다.

차량 이동 경로 선상에서 가장 가까이 있는 대상까지의 거리, 속도 정보를 얻음

수집된 센서 데이터는 차량의 주변 환경을 파악하는 인지 단계로 전달된다.

Perception

GPS/IMU

“칼만 필터” 기법을 사용해 GPS와 IMU의 장점만 조합하여 로컬라이제이션 결과값을 도출한다.

문제점

- 정확도는 1미터 범위에서만 보장된다.
- GPS 신호는 다중 경로 문제를 갖고 있다.
- 터널같이 하늘이 가려진 곳에서는 GPS 값을 정확히 받을 수 없다

카메라

로컬라이제이션 순서

1) 스테레오 이미지 쌍에 대한 삼각 측량을 통해 시차맵부터 구한다. 이 맵은 각 지점에 대한 깊은 정보를 추출하는 데 활용한다.
2) 연속적으로 들어오는 스테레오 이미지 프레임 사이에서 두드러진 특징을 찾아서 각 프레임에 있는 특징점 사이의 상호 연관 관계를 파악한다.
3) 이렇게 추출한 특징과 기존에 파악한 맵을 비교하는 방식으로 차량의 현재 위치를 알아낼 수 있다.

하지만 비전 기반 위치 측정 기법은 조명 상태에 민감하기 때문에 이 방식에 전적으로 의존할 수 없다.

라이다(LiDAR)

라이다로부터 생성된 포인트 클라우드로 주변 환경에 대한 형태를 기술할 수는 있지만, 포인트 단위로 구별하기는 힘들다. “파티클 필터”를 적용해, 관측된 형태를 기존에 파악된 맵과 비교하는 방식으로 불확실성을 줄인다.

이렇게 측정 값과 맵의 상호 연관성을 파악하는 방식으로 이동 중인 차량의 로컬라이제이션을 처리할 수 있다.

하지만 공기 중에 부유 입자가 많으면 측정 값에 노이즈가 발생한다.

레이더와 소나 시스템으로 생성된 데이터는 긴급한 동작을 수행하는데 사용되기 때문에 가공할 일이 많지 않다.

다음과 같은 문제점들을 보완하기 위해 “센서 융합 프로세스”가 필요하다.

칼만 필터

칼만 필터(Kalman filter)는 잡음이 포함되어 있는 측정치를 바탕으로 선형 역학계의 상태를 추정하는 재귀 필터로, 루돌프 칼만이 개발하였다. 칼만 필터는 컴퓨터 비전, 로봇 공학, 레이다 등의 여러 분야에 사용된다. 칼만 필터는 과거에 수행한 측정값을 바탕으로 현재의 상태 변수의 결합분포를 추정한다.

알고리즘은 예측과 업데이트의 두 단계로 이루어진다. 예측 단계에서는 현재 상태 변수의 값과 정확도를 예측한다. 현재 상태 변수의 값이 실제로 측정된 이후, 업데이트 단계에서는 이전에 추정한 상태 변수를 기반으로 예측한 측정치와 실제 측정치의 차이를 반영해 현재의 상태 변수를 업데이트한다.

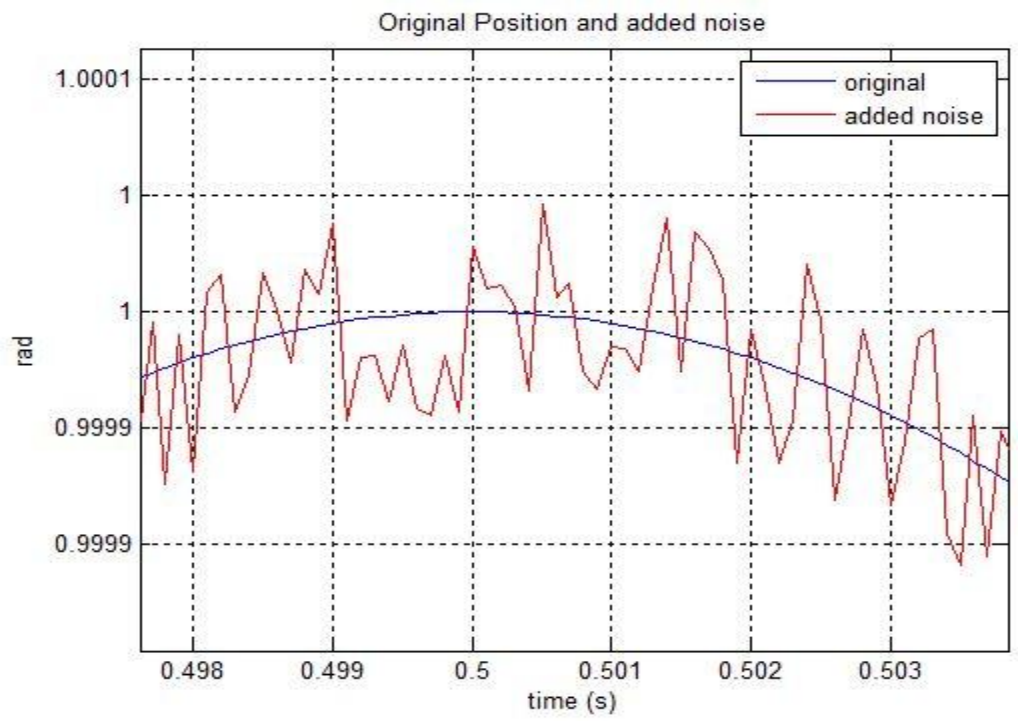
확장 칼만 필터는 비선형 시스템을 기반으로 동작한다.

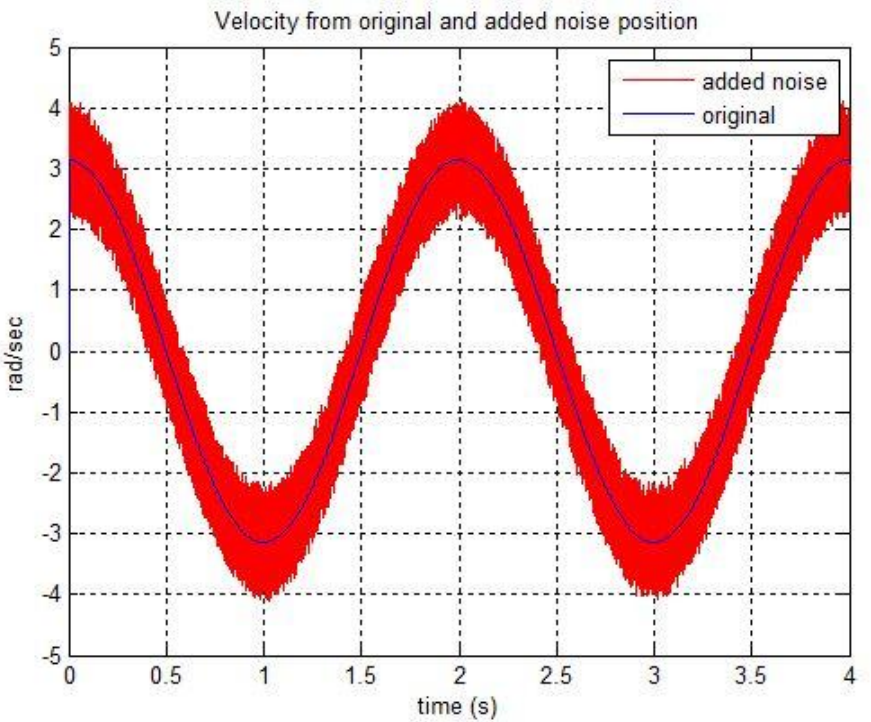
파티클 필터

파티클 필터(particle filter)는 시뮬레이션에 기반을 둔 예측기술의 하나로 계속적인 몬테카를로 방법이라고도 한다. 파티클 필터는 계량경제학에서 중요하게 쓰인다.

파티클 필터의 목적은 연속적으로 들어오는 정보를 오차가 존재하는 관측값만을 가지고 정보를 예측하는 데에 있다.

파티클 필터의 동작원리는 매우 간단하다. 시스템에 적절하게 제안된 확률분포로 임의로 생성된 입력을 여럿 가하여보고 그것들을 종합하여 시스템의 정보를 추측하는 것이다.





칼만 필터는 물체의 측정값에 확률적인 오차가 포함되고, 또한 물체의 특정 시점에서의 상태가 이전 시점의 상태와 선형적인 관계를 가지고 있는 경우 적용이 가능하다.

예를 들어, 레이다 추적의 경우 특정 물체의 위치, 속도, 가속도 등을 측정할 수 있지만 이 측정값에 오차가 포함되어 있을 수 있다. 이 경우, 연속적으로 측정하는 값들을 칼만 필터를 이용해서 해당 물체의 위치를 추정할 수 있다.

파티클 필터의 로컬라이제이션 과정

1. **Random sample(R.S) 생성**: 초기에 맵의 어디에 이동 로봇이 위치하는지에 대한 정보가 아예 없기 때문에 랜덤하게 맵의 군데 군데 입자들을 뿌린다. 이 입자들의 정보는 2D문제에서는 x, y의 좌표 값이 된다. 왜냐면 상태가 위치이기 때문에 좌표 값만으로 위치를 추적 가능하기 때문이다. X_t-1(입자 set)

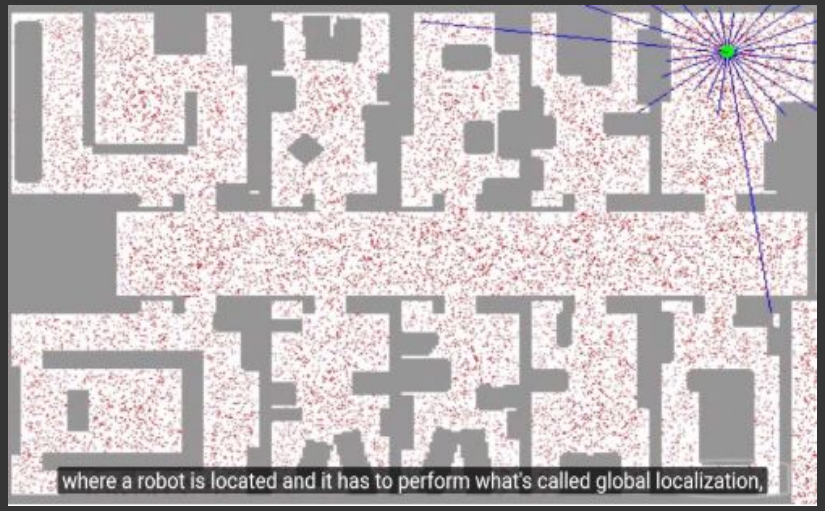
2. **Sampling 과정**: System model(로봇의 action(motion)을 고려한 system model)에 R.S들을 통과 시켜 새로운 위치로 입자들을 이동시킨다. x_t(입자 위치)

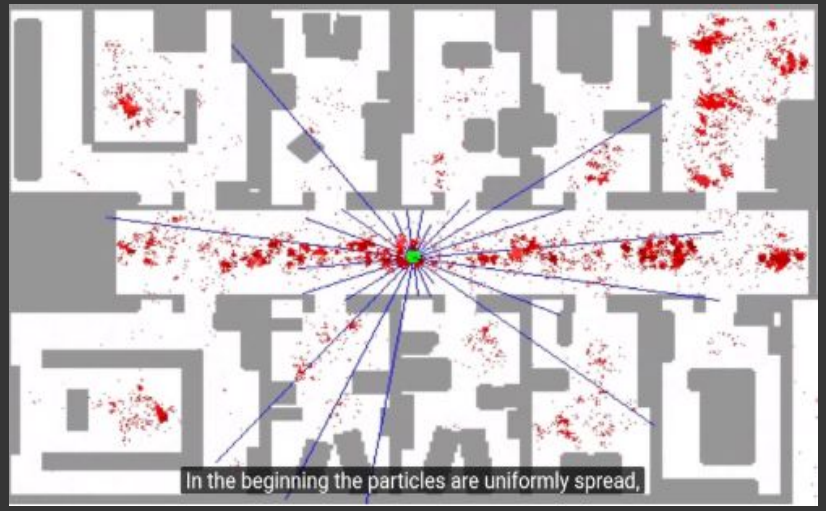
3. **Importance Sampling 과정**: w(x) (각 입자들의 weight) 는 측정 값의 확률에 비례한다. Global localization 문제에서는 측정한다는 의미는 Sonar sensor나 Laser sensor를 이용하여 공간의 형태를 측정한다. 이런 공간의 형태가 전체 맵에서 존재할 확률이 가장 높은 곳에 weight를 더 주게 된다. w_t(입자 weight)

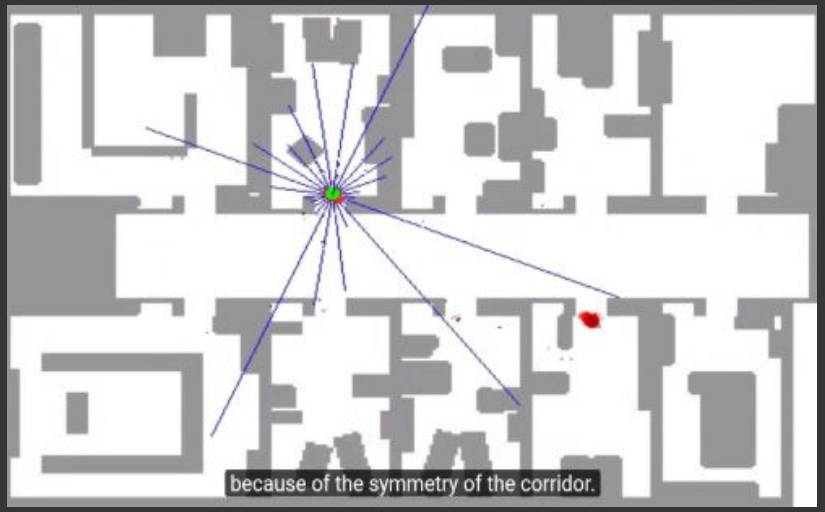
4. 이제 우리는 x_t 와 w_t 를 알게 되었으므로, **X'_t(weighted samples)** 를 알게 된다.

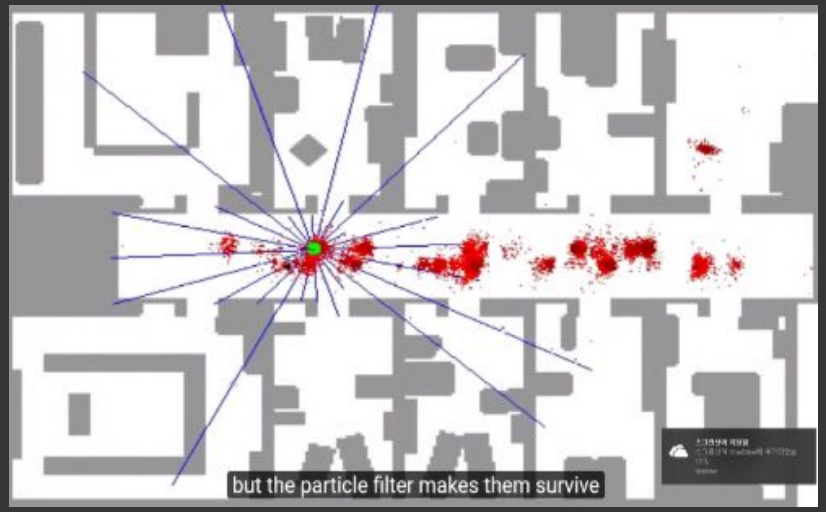
5. **Resampling 과정**: 이렇게 무게가 주어진 Sample들을 이제 다시 무게가 일정한 입자들로 바꾸기 위해서 Resampling을 과정을 거친다. 여기서 Resampling 방법은 여러가지가 있으므로, 이것들에 대해선 따로 다루도록 하겠다. 원리는 간단하다. 원리는 무게가 큰 입자들은 근처에서 잘게 쪼개어 근처에 입자들이 많이 분포되도록 하고, 무게가 작은(확률이 낮은) 입자들은 제거한다. 여기서 명심해야 할 것은 Resampling 과정을 거치기 전과 후의 입자들의 갯수가 일정해야한다는 것이다. X_t (시간 t에서의 입자 set)

6. 이런 방식으로 **반복적으로** 위치를 추적해 나간다.

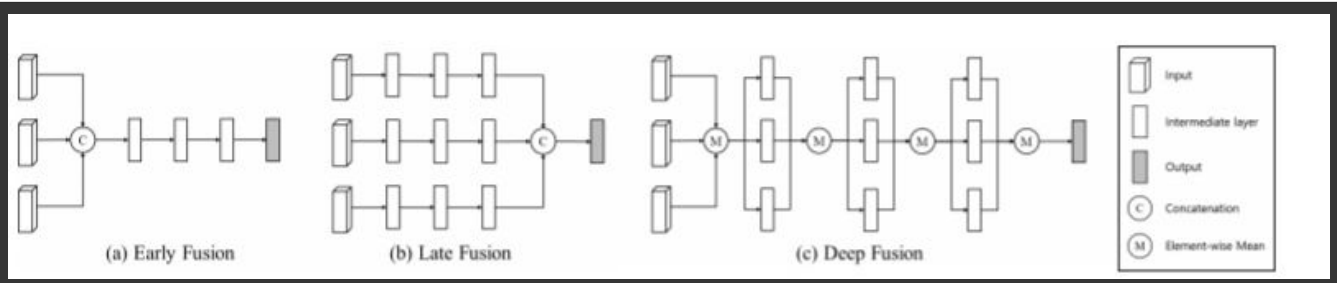

where a robot is located and it has to perform what's called global localization,


In the beginning the particles are uniformly spread,


because of the symmetry of the corridor.


but the particle filter makes them survive

센서 융합 프로세스란 여러 센서에서 검출된 객체를 융합하여 성능을 향상 하는 것을 목표로 하는 프로세스이다.



Early Fusion, Late Fusion

Early Fusion: 서로 다른 센서의 raw data를 사전에 결합하여 알고리즘을 사용하는 것이다. Camera와 LiDAR의 경우에는 RGB 이미지와 point cloud의 거리정보를 이 용하여 feature level에서 결합하여 RGB-D 이미지를 생성하고 객체 검출과 같은 알고리즘에 사용한다.

Late Fusion: 서로 다른 센서를 독립적으로 객체 검출을 한 뒤, 결과를 이용하여 융합을 하는 방법이다.

Deep Fusion

입력된 raw data를 융합을 한 뒤 알고리즘을 이용하는 점은 early fusion과 비슷하다. 서로 다른 센서를 neural network를 이용하여 융합하는 차이가 있다.

Early fusion과 Deep fusion 같은 경우에는 검출 알고리즘 전 미리 센서 데이터를 융합하여 사용하는 방법이지만, 하나의 센서라도 동작을 하지 않는다면 전체 시스템에 영향을 주게 된다.

하지만 Late fusion의 경우에는 각 센서에 서 객체 검출을 한 뒤에 융합하므로 하나의 센서가 동작하 지 않아도 다른 하나의 센서를 이용하여 지속해서 객체를 검출해 나갈 수 있지만 카메라와 라이다 센서 데이터에 대 하여 개별적으로 객체 검출을 해야 하므로 computing power가 많이 소모된다.

개체 인지 및 추적

라이다는 깊이 정보에 대한 정확도가 높아서 개체 감지 및 추적 작업에 주로 활용한다.

최근에는 딥 러닝 기반의 개체 감지 및 추적의 정확도도 높아졌다.

개체 인지 작업은 딥 뉴럴 네트워크 중에서도 “CNN(합성 신경망)”을 많이 사용한다.

파이프라인은 다음과 같은 계층으로 구성된다.

(1) 컨볼루션 계층은 입력 이미지로부터 다양한 특징을 추출하기 위한 여러 가지 필터를 갖고 있다. 각 필터마다 ‘학습 가능한’ 매개변수 집합을 갖고 있는데, 학습 과정에 이 값을 채운다.

(2) 활성 계층은 타깃 뉴런의 활성 여부를 결정한다.

(3) 풀링 계층은 매개변수의 개수를 줄여서 신경망의 연산을 줄이는 방식으로 표현 공간의 크기를 줄인다.

(4) 완전 연결 계층은 이전 계층에서 활성화된 뉴런이 모두 연결된 상태다.

개체 추적이란 원하는 개체를 선택해서 그 개체의 이동 궤적을 자동으로 추정하는 기법이다.

이 기법을 적용해 현재 차량과의 충돌을 방지하는 데 활용한다.

딥 러닝을 통해 보조 이미지를 활용해 여러 층으로 쌓은 오토 인코더에 오프라인에서 범용 이미지 특징을 학습시킨다.

이렇게 오프라인에서 학습한 모델을 실제 주행에 적용한다.

파악한 주변 환경으로 의사결정 단계에서 안전하고 효율적인 동작 계획을 실시간으로 생성한다.

Decision

동작 예측

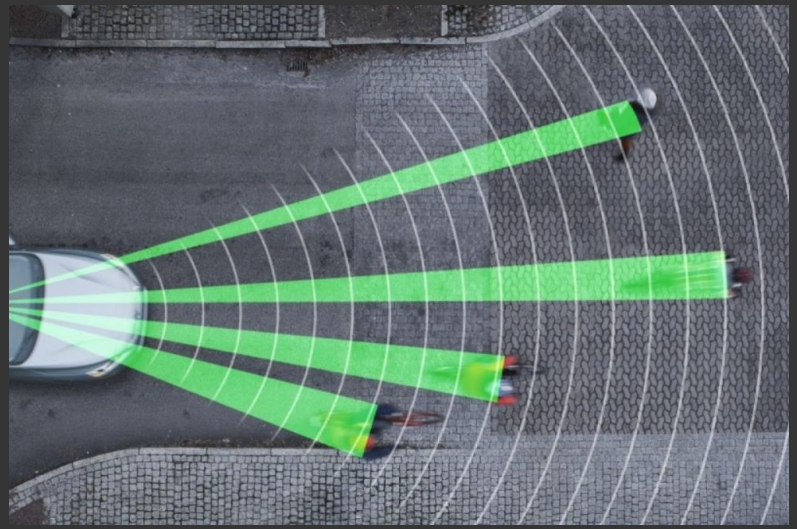
안전한 주행을 위해서는 의사결정 단계에서 주변에 있는 차량의 동작을 예측한 결과를 토대로 동작 계획을 수립해야 한다.

즉, 다른 차량의 도달 가능한 지점에 대한 확률 모델을 만들고 도달 가능한 지점에 대한 확률 분포를 구하는 기법을 사용해야 한다.

경로 계획

최적의 경로는 단순하게 생각하면 모든 경로를 탐색한 뒤 비용 함수로 찾아내면 된다. 하지만 그에 따른 연산 시간 때문에 실시간으로 사용하기 힘들다. 따라서 계산 복잡도를 줄이기 위해 확률 기반 기법을 주로 사용한다.

Decision



장애물 회피

안전성을 위한 장애물 회피 메커니즘을 최소한 두 단계 이상으로 구성한다.

1) 능동형 메커니즘: 교통 예측을 기반으로 처리한다. 차량이 주행하는 동안 교통 예측 메커니즘을 통해 충돌까지 남은 시간이나 최소 거리에 대한 추정치 등을 계산하며, 이렇게 구한 정보를 토대로 필요한 시점에 장애물 회피 메커니즘이 작동해 주변 경로를 다시 계획한다.
2) 반응형 메커니즘: 위 메커니즘으로 처리가 불가할 경우 레이더 데이터 기반의 수동형 메커니즘이 작동한다. 이동 경로상의 물체가 레이더에 감지되면, 주행 제어 시스템에 개입해 장애물을 피하도록 조작한다.

자율 주행 클라이언트 시스템

클라이언트 시스템은 실시간 및 신뢰성 요구사항을 만족하도록 앞에서 소개한 여러 알고리즘을 통합한다.

이 과정에서 다음과 같은 몇 가지 어려움이 발생한다.

1) 센서에서 생성된 방대한 양의 데이터를 충분히 처리할 수 있을 정도로 프로세싱 파이프라인이 빨라야 한다.
2) 어느 한 부분에서 오류가 발생하더라도 복구할 수 있을 정도로 충분히 견고해야 한다.
3) 또한 이 모든 연산을 처리하는 데 있어 엄격한 에너지 및 리소스 제약사항도 만족해야 한다.

ROS

ROS는 로보틱스 애플리케이션에 특화된 강력한 분산 컴퓨팅 프레임워크이다.

로컬라이제이션같은구체적인 작업은 ROS 노드에서 구동하는 데 “토픽”과 “서비스”를 통해 서로 통신한다.

자율 주행 시스템 용도로 굉장히 뛰어난 프레임워크이지만, 다음과 같은 문제점이 있다.

1) 신뢰성: ROS에서 마스터는 하나만 존재하고 장애가 발생한 노드를 복구하기 위한 모니터가 없다.
2) 성능: 메시지를 브로드캐스트 방식으로 보내는 과정에서 같은 메시지가 여러 번 중복될 수 있는데, 이로 인해 성능 저하가 발생한다.
   - 브로드캐스트 방식: 단일 노드로부터 해당 네트워크의 모든 노드에게 소식(패킷/프레임 등)을 전하는 것
3) 보안: 인증과 암호화 메커니즘이 없다.

ROS의 구조

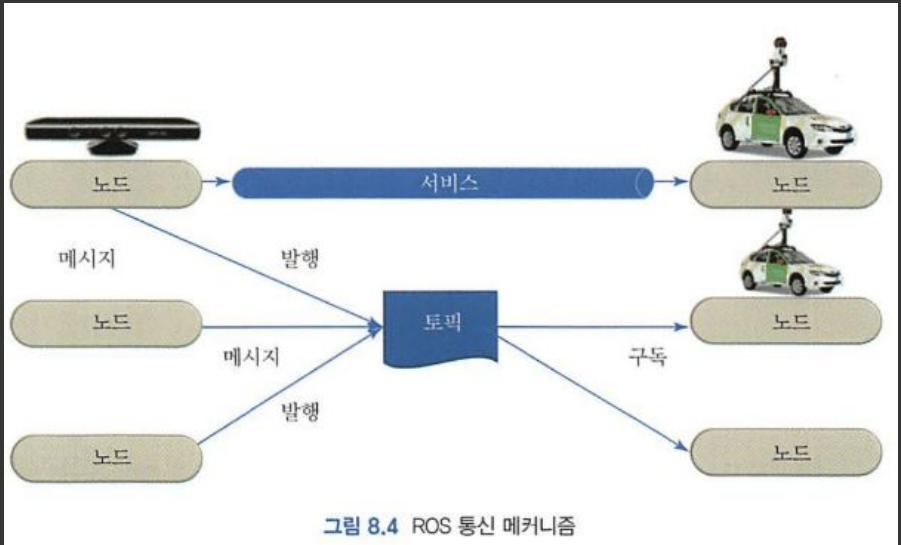


그림 8.4 ROS 통신 메커니즘

노드: ROS에서 최소 단위의 실행 프로세스를 가리키는 용어이다. (하나의 프로그램이라고 생각) 하나의 목적에 하나의 노드를 개발하는 것이 좋음

마스터: 각 노드간 발견을 할 수 있도록 도와주는 네임 서버와 같음.

*XMLRPC통신을 이용해 각 노드들과 통신함

*노드들을 관리함. 각 노드들은 생성되면서 마스터에 발행자, 구독자, 토픽, 서비스의 각 이름, 메시지 형태, URI 주소와 포트 등을 등록함. 이 정보들을 기반으로 각 노드들끼리 서로 통신함

XMLRPC 프로토콜: 노드들이 마스터와 통신할 때, 노드간 접속 요청 및 응답할 때 이용함

*각 노드간 통신에는 TCP/IP 계열의 TCPROS을 사용하기도 함

*메시지 통신은 마스터를 거치지 않고 바로 통신하기 때문에 TCPROS를 사용

패키지: ROS를 구성하는 기본 단위이다. 패키지에는 하나 이상의 노드가 포함 됨

메시지: 각 노드간 주고받는 데이터를 말하며, 그 안에 다양한 데이터 타입들을 정의해서 사용할 수 있음

*메시지를 사용하는 통신 방법으로 TCPROSS, UDPROS 방식 등이 있음

토픽: 어떠한 메시지를 전달할 '주제'

*구독자가 마스터에게 특정 토픽에 대한 구독을 요청하면, 마스터는 등록되어 있는 발행자 정보를 구독자에게 전달하고, 구독자는 발행자 노드에 직접 연결해서 메시지를 전달받을 수 있게 됨

발행자: 특정 토픽에 대해 메시지를 전송하는 노드

구독자: 특정 토픽에 대한 메시지를 전달받는 노드

*발행자/구독자의 통신은 비동기 방식으로 이루어짐

서비스: 요청과 응답이 동시에 이루어져야 하는 동기 방식이 필요할 때 사용

*요청이 있을 때 응답하는 서비스 서버와 요청하고 응답 받는 서비스 클라이언트로 나뉨

*토픽과 달리 일회성 메시지 통신이며, 요청과 응답이 완료되면 두 노드간의 연결은 끊어짐

문제점/해결방안

신뢰성

ROS 구현은 마스터 노드가 유일하다. 따라서 마스터 노드가 죽으면 전체 시스템이 죽는다. 이는 안전에 치명적이고 이를 해결하기 위해 '주키퍼'와 유사한 메커니즘을 구현한다.

ZooKeeper 기반의 **ROS:** 메인 노드가 죽으면 백업 노드가 이어받아 정상구동하게 한다. 또한 주키퍼 메커니즘을 통해 모니터링 하다가 장애가 발생한 노드를 감지하면 이를 재구동한다.

성능

1) 대부분 로컬 노드끼리는 루프백 메커니즘으로 통신한다. 이러한 루프백 파이프라인을 통과할 때마다 **20ms**가량의 오버헤드가 발생한다.

-> 이러한 문제를 해결하기 위해 공유 메모리 메커니즘을 활용해 메시지가 **TCP/IP** 스택을 거치지 않고 목적지 노드로 곧바로 가도록 처리한다.

2) ROS 노드가 메시지를 브로드캐스팅하는 과정에서 여러 개의 복사본을 만들기 때문에 시스템의 대역폭을 상당히 잡아먹는다.

-> 브로드캐스팅 대신 멀티캐스트 메커니즘으로 대체하여 시스템 처리량 향상

*멀티캐스팅: 전송할 데이터의 복사가 서버가 아닌 라우터에서 발생되어 서버의 부하를 없애고 (모든 노드에게 데이터를 보내는 브로드캐스트와 달리) 멀티캐스트 주소를 공유한 노드에게만 복사한 데이터를 주는 방식

보안

1) 해커가 **ROS** 노드에 침투해서 메모리를 끊임없이 할당하는 작업을 수행해 시스템 메모리 전체를 고갈시키는 방식으로 노드를 다운 시킬 수 있다.

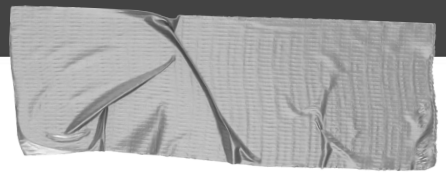
-> **LXC**(리눅스 컨테이너)를 사용해 각 노드마다 허용된 리소스 양을 제한하는 방식으로 해결

추가로 **LXC**에서 제공하는 "샌드박스 메커니즘"을 통해 노드를 외부로부터 보호할 수 있어서 리소스 누수 현상도 방지할 수 있다.

2) ROS 노드끼리 주고받는 메시지를 중간에 가로채는 중간자 공격을 시도할 수 있다.

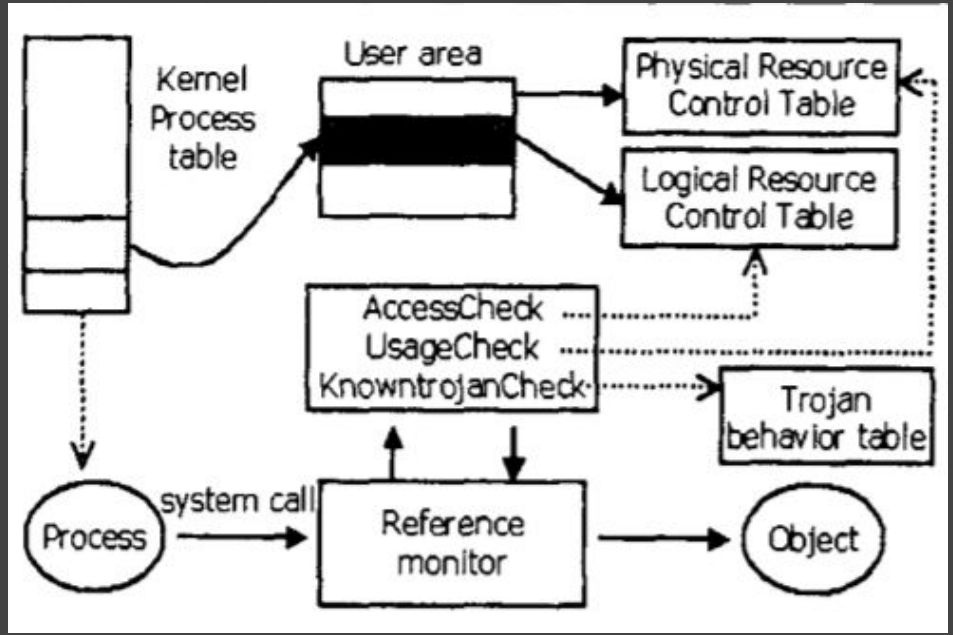
-> 메시지를 암호화해야 함

*샌드박스 메커니즘: 외부 프로그램을 별도로 격리된 가상공간에 가두어 작동하게 만드는 방법



샌드박스 메커니즘

프로그램이 시작되어 메모리에 로드 될 때, 프로그램마다 각각의 설정된 보안 정책을 동시에 불러들여 테이블로 만들어 놓는다. 이 보안정책은 각각의 프로세스에서 참조가 가능하도록 커널내의 프로세스 영역의 사용자 영역에 보안정책을 참조할 수 있도록 연결시키고, 프로세스가 참조모니터를 통하여 자원을 요청할 때 발생하는 시스템 콜을 변경하여 접근권한검사, 자원사용량검사, 알려진 해킹프로그램의 일반적인 특징검사를 통하여 보안정책에 설정된 권한허용과 자원사용량을 위반하는지 검사하고 감시하게 된다. 접근권한과 자원사용이 정당하면 참조모니터를 통하여 시스템 자원의 요청을 사용할 수 있도록 허가하고, 이에 위반되면 자원할당에 대한 요청은 금지시켜서 악성코드의 악성행위를 방지할 수 있도록 구성됨



하드웨어 플랫폼

이 플랫폼은 여러가지 문제점을 가지고 있다. 자율 주행 선도업체에서는 신회성과 성능을 위해 하드웨어 플랫폼이 2개의 컴퓨트 박스로 구성되어 있으며, 첫 번째 박스에 장애가 발생하자마자 두 번째 박스가 그 역할을 넘겨받고, 두 박스 모두 한계 상태에 도달할 경우 5000W의 전력소모가 발생하면서 엄청난 열을 발산한다. 게다가 2000~3000만원에 육박하는 고성능의 장비들을 사용하여 일반인이 감당하기 힘들다.

이러한 문제를 해결하기 위해 모바일용 ARM SoC를 사용하여 실전 요구사항 수준에 가까운 간단한 비전 기반 자율 주행 시스템을 구현한다. (ARM의 전력 소비량: 15W)

로컬라이제이션 파이프라인은 초당 25개의 이미지를 처리할 수 있다. (요구사항 초당 30개)
딥 러닝 파이프라인은 초당 2~3개의 물체를 인지할 수 있었고 계획 및 제어 파이프라인은 6ms 이내에 경로를 계획할 수 있다.
이렇게 구축한 시스템을 차량에 장착해 로컬라이제이션 과정에 아무런 손실 없이 시속 5마일 정도의 속도로 달릴 수 있다.

*SoC: 한 개의 칩에 완전 구동이 가능한 제품과 시스템이 들어 있는 것