SLAM-based navigation of an autonomous driving car (SLAM 기반 자율주행차 내비게이션)

본 논문에서는, LiDAR를 이용한 자율주행 모델 차량을 제시한다.
센서 데이터 및 2D 지도는 지정된 지점에서 목표 지점까지의 최단 경로를 구한다.
우선, 자동차가 주행하는 모든 구역은 LiDAR 센서로 스캔된다. 센서 데이터를 기반으로 모든 장애물이 있는 2D 지도가 생성된다. 이후 이 지도에 자동차의 출발점을 입력하고 주행 장소를 정할 수 있다. 그 차는 어떤 장애물도 부딪치지 않고 출발점에서 목적지까지 가는 가장 짧은 길을 찾을 것이다. 목적지로 가는 길에 LiDAR 센서는 끊임없이 지역 환경을 스캔하고 있기 때문에, 자동차는 미지의 장애물도 탐지할 수 있을 것이고, 필요하다면 경로를 다시 만들 것이다.
지도의 작성과 경로계획은 노트북에서 이루어지며, LiDAR 센서 데이터는 RaspberryPi 보드에 의해 스캔된다.

이 작업의 목적은 LiDAR 센서의 도움을 받아 SLAM 지도를 기반으로 한 자율주행차 개발을 실증하는 것이다. 그러나 이 목표에는 일부 기계 및 소프트웨어 문제가 포함되어 있다. 한편으로 모든 구성품은 차량에 잘 배치되어야 하며 올바르게 연결되어야 한다. 모든 케이블은 바닥에 매달리거나 바퀴를 막지 않고 차를 통과해야 하며 모든 접점은 잘 절연되어야 한다. 반면에 자동차는 전체 움직임을 정확하게 제어하기 위한 소프트웨어를 필요로 한다. 첫 번째 요건은 LiDAR 센서 데이터를 바탕으로 2D SLAM 지도를 만드는 소프트웨어다. 그 후에, 지구 지도에서 가장 짧은 경로를 찾을 수 있는 소프트웨어가 필요하며, 그 중 하나는 지역의 장애물을 자발적으로 피할 수 있어야 한다. 게다가 이 소프트웨어는 모든 중요한 데이터를 교환하기 위해 위피를 통해 자동차의 관리부서와 노트북을 연결해야만 한다. LiDAR 센서는 차 상단에 있어야 하지만, CPU 성능이 요구되기 때문에 경로 찾기 알고리즘과 환경 지도가 노트북에서 실행된다.

About SLAM

SLAM이라는 용어는 동시 localizaton과 mapping의 약자로 명시되어 있다. 그것은 주로 Smith, Self, Cheeseman의 초기 작품을 바탕으로 Hugh Durrant-Whyte와 John J. Leonar[10]에 의해 개발되었다. Durrant-Whyte와 Leonard는 처음에 그것을 SMAL이라고 불렀지만 나중에 더 나은 영향을 주기 위해 바뀌었다. SLAM은 지도를 사용하여 환경을 탐색하는 동시에 이동 로봇에 의해 알려지지 않은 영역의 지도를 작성하는 문제에 대해 관심이 있다. 그래서 로봇은 환경의 어느 곳에서든 시작하며, 현재 인식된 장애물에 상대적인 장소를 언제든지 알 필요가 있다.

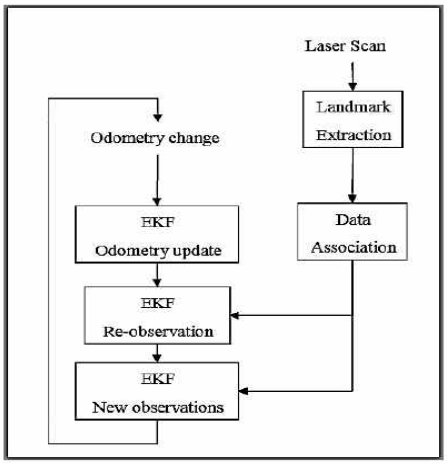
SLAM은 사용할 수 있는 엄청난 양의 하드웨어로 많은 방법으로 구현될 수 있다. SLAM은 하나의 알고리즘이 아니라, 몇 가지 문제를 해결하는 전체 개념에 가깝다. 그것은 "랜드마크 추출, 데이터 연결, 상태 예측, 상태 업데이트, 획기적인 업데이트"와 같은 여러 부분으로 구성되어 있다. 그것은 어떤 로봇을 사용할 것인지와 같이 용도에 따라 다르다. 로봇은 어느 지역에서 운전할 것인가? 정지물체만 있는 홀인가, 지도 제작시 이동이 많은 도심 번화가인가. 로봇에게 2D 또는 3D 비전이 필요한가? SLAM 프로세스를 실행하기 전에 이와 같은 의문점에 대해 답할 필요가 있다. 이들은 모두 잘 작동하는 SLAM 개념을 구현하기 위해 사용한 하드웨어 장치와 알고리즘에 영향을 미친다. 본 논문의 경우 자율주행차는 2D 움직임만 고려한다. 자동차가 운전해야 하는 지역은 오직 정적인 물체가 있는 실내 환경이다.

List of Abbreviations

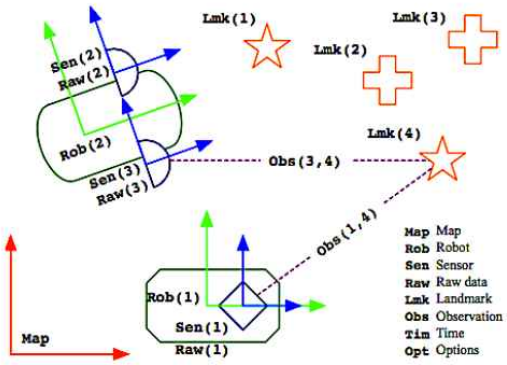
AMCL	Adaptive Monte Carlo Localization
CPU	Computer Processing Unit
D	Dimensional
EKF	Extended Kalman Filter
GUI	Graphical User Interface
LiDAR	Light Detection and Ranging
ROS	Robot Operating System
SCM	Software Configuration Management
SLAM	Simultaneous Localization and Mapping
TUM	Technical University Munich
XML	Extensible Markup Language

The SLAM process

SLAM의 목적은 로봇이 움직이는 지역의 지도를 만드는 것이다. 기본 SLAM 과정은 여러 단계로 이루어진다. 이 모든 단계들은 다른 방법으로 행해질 수 있다. 아래 그래픽은 연장된 칼만 필터를 사용한 SLAM 프로세스의 예를 보여준다. EKF는 최초의 확률론적 SLAM 알고리즘 중 하나이다. SLAM 프로세스가 어떻게 작동하는지 설명하는 것이 종종 선택되지만, 실제로는 다른 알고리즘도 사용할 수 있다.



첫 번째 단계는 측정 장치로 초기 포즈 환경을 스캔하는 것이며, 이 경우 LiDAR 센서 데이터를 수집한다. 이 데이터에는 로봇의 초기 위치에 기초해 이른바 랜드마크라고 불리는 거리가 얼마나 멀리 떨어져 있는지, 또한 그 위치가 어디인지에 대한 각도를 포함하고 있다. 이러한 랜드마크들은 지역 환경의 장애물들과 같은 다양한 지점들이며, 로봇까지의 거리와 EKF의 가장 많은 입력이 그들의 각도와 결합된다.



EKF는 로봇의 위치가 현재 지도상에 있다고 추정하고 필요한 경우 로봇의 타도법을 업데이트하기 위한 알고리즘이다. 로봇의 오도메트리 데이터는 지도에 있는 로봇의 위치를 포함하고 있다. 이 자료는 지도를 정확하게 작성하려면 매우 정확해야 한다. 지도는 대형 상태 벡터 적층 로봇과 랜드마크 상태로, 여기서 R은 로봇이고 M은 랜드마크 상태의 집합이다. EKF 알고리즘은 상태 벡터의 평균 및 공변량 매트릭스를 사용하여 가우스 변수에 의한 지도를 모델링하며, x와 P로 표시한다. SLAM과 EKF의 목적은 이 지도를 항상 최신 상태로 유지하는 것이다.

$$x = \begin{bmatrix} R \\ M \end{bmatrix} = \begin{bmatrix} R \\ L_1 \\ \vdots \\ L_n \end{bmatrix} \qquad P = \begin{bmatrix} P_{RR} & P_{RM} \\ P_{MR} & P_{MM} \end{bmatrix} = \begin{bmatrix} P_{RR} & P_{RL_1} & \dots & P_{RL_n} \\ P_{L_1R} & P_{L_1L_1} & \dots & P_{L_1L_n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{L_nR} & P_{L_nL_1} & \dots & P_{L_nL_n} \end{bmatrix}$$

맨 처음에 지도는 아무런 랜드마크 없이 시작하고 초기 로봇 포즈는 지도의 원점으로 설정되므로 n = 0, x = R이다.

$$x = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad P = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

로봇이 움직이기 시작하자마자, 로봇의 타원법이 바뀔 것이다. 로봇 모션은 일반적인 시간 업데이트 함수에 기초한다. 상태 벡터 x에 기초해 제어 벡터 u와 섭동 벡터 n을 사용한다.

$$x \leftarrow f(x, u, n)$$

이동 중에 그리고 모든 새로운 위치에서, LiDAR는 환경의 새로운 데이터를 수집한다. 중요한 랜드마크는 다시 추출되어 로봇이 이전에 본 랜드마크와 연결된다. 재관찰된 점에 기초하여 로봇은 EKF에서 새로운 위치를 갱신할 수 있다. 이것은 로봇이 항상 현재 위치에서 상대적인 거리와 각도를 가지고 있기 때문에 가능하다. 관찰은 일반적인 관측 기능에 기초하는데, 여기서 y는 잡음 측정, x는 풀상태, h()는 관측기능, v는 측정소음이다.

$$y = h(x) + v$$

Implementation

1. Used Hardware

1.1. The car

전체 프로젝트의 기본은 당연히 차(car)이다. 그것은 금속으로 만들어진 전기 구동 자동차로, 12볼트의 입력 전압이 필요하다. 차축은 55cm, 전체 길이는 70cm, 폭은 55cm. 그것의 최대 속도는 80 km/이지만, 이 프로젝트의 최대 속도는 0.55 m/s이다. 시험 실패 시 장애물이나 사람을 손상시키지 않고 실내에서 안전하게 운전할 수 있는 좋은 속도다.

(a) Front of the car (b) Back of the car

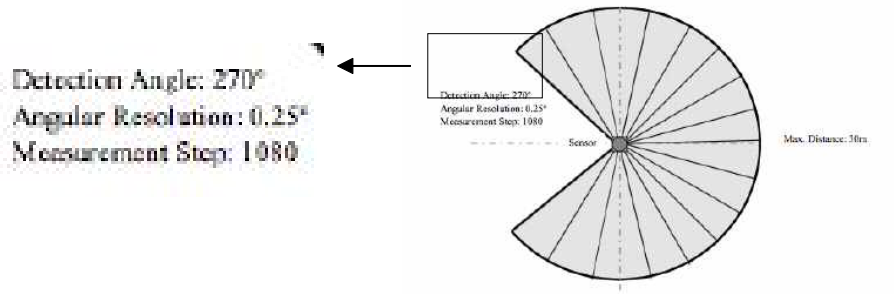
사진에서 볼 수 있는 것처럼 자동차는 많은 센서와 다른 전자 장치들을 포함하고 있다. 모터 관리로서 자동차는 필요한 값을 엔진에 공급하기 위해 작은 아르뒤노 보드를 가지고 있다. 속도값과 도 값으로 PWM 신호를 엔진에 백분율로 전송해 자동차의 조향 각도를 제어할 수 있다. 전진·후진주행도 가능하다. 입력 값은 RaspberryPi 보드에 의해 직렬 USB를 통해 전송된다.
차에는 라즈베리파이 보드도 심어져 있다. 운영체제로서 우분투 메이트 16.04를 실행하고 있으며, ROS 배분이 설치되어 있다. 노트북에서 실행되는 ROS 마스터와 와이파이를 통해 통신하는 것이 임무다. RaspberryPi 보드는 조향 각 및 속도 값과 같이 마스터로부터 필요한 데이터를 수신한다. 이 데이터는 Laptop에 의해 계산되고 RaspberryPi 보드는 그들을 Arduino 보드로 전달한다.

1.2. The laptop

전체 프로젝트의 통제 기반은 일반적인 소니 바이오 노트북이다. 운영체제로 리눅스 14.04 인디고를 운영 중이며 ROS 배포도 설치됐다. Intel Pentium 듀얼 코어 프로세서는 경로 계획 알고리즘을 실행하고, 지도를 제시하고, 자율 주행 자동차에 필요한 현지화를 계산하는 데 사용된다. 노트북에는 프로젝트에 필요한 모든 도구가 설치되어 있고 ROS 마스터는 이 기계에서 실행 중이다. LiDAR의 스캔 데이터 및 속도 명령과 같이 소위 말하는 모든 주제의 데이터는 ROS 마스터에게 공개된다. 전체 시각화도 그것에 의해 이루어진다. 자동차의 시발점을 설정할 수 있고, 만들어진 SLAM 지도에서 목표를 설정할 수 있다.

1.3. The measurement device

전체 측정은 단일 LiDAR 센서인 호쿠요 UTM-30LX-EW에 의해 이루어진다. 차량 앞에 놓여져 반경 270도로 환경을 스캔한다. 스캔 속도가 25ms/스캔인 0.1~30m의 정확한 검출 범위를 가지고 있다. 각도는 0.25도.

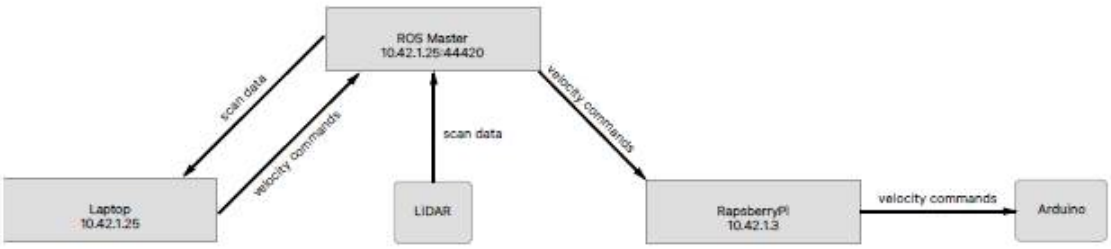


전체 데이터는 이더넷을 통해 목적지 장치로 전송된다. 출력 데이터 유형은 LiDAR 주위의 0도에서 270도까지의 거리 값을 포함하는 배열이다. 그러므로 LiDAR는 매 25ms마다 270 * 4 = 1080 이중 값의 배열을 발표한다. 그것은 노트북에 직접 연결되어 있다.

2. Software

2.1. Operating system

ROS는 자율주행차의 운영체제로 전체 프로젝트의 기본이다. 그것은 노트북과 라즈베리피에 설치된다. ROS 시스템에는 모든 기기가 등록되고 데이터를 게시하고 다른 노드에서 새 기기를 수신할 수 있는 마스터가 필요하다.
이 경우 이 마스터는 노트북에서 실행 중이며 두 장치는 모두 임시 네트워크에서 와이파이(Wifi)를 통해 마스터에 연결된다. 마스터는 모든 자유 포트에서 시작할 수 있으며, 이 경우 포트 44420이 사용된다. Laptop은 계산된 속도 명령을 발표해야 하므로 RaspberryPi가 그것들을 Arduino 보드로 전달할 수 있다. LiDAR 센서는 스캔한 데이터를 공개한다. 이 프로젝트에서 사용되는 모든 알고리즘과 툴은 ROS 시스템을 기반으로 한다.



2.2. Mapping

SLAM 기반 내비게이션은 SLAM 지도가 필요하다. 항법(navigation) 부분의 기본이다. 상세한 지도여야 하므로 현지화 알고리즘이 제대로 작동할 것이다. 이 논문에 사용된 지도는 모두 '히터_슬람도서관'이 만든 것이다.
이 알고리즘은 LiDAR 센서 데이터를 사용하여 전체 영역의 지도를 만들고, 그 후에 자동차가 주행해야 한다. 이 알고리즘의 접근법은 이른바 FastSLAM을 하는 것이다. 전체 영역은 점유 2D 그리드 맵으로 표시된다. LiDAR 센서의 업데이트 속도가 높기 때문에 근사 데이터만 사용할 수 있다. 센서에서 스캔한 엔드포인트 데이터는 추정된 플랫폼 방향 및 조인트 값을 사용하여 점 구름으로 변환된다. 스캔 매칭 알고리즘으로서, endpoint z 좌표에 기초한 필터링만으로 충분하므로, "의도된 스캔 평면의 임계값 내에 있는 엔드포인트만 스캔 매칭 프로세스에 사용된다."

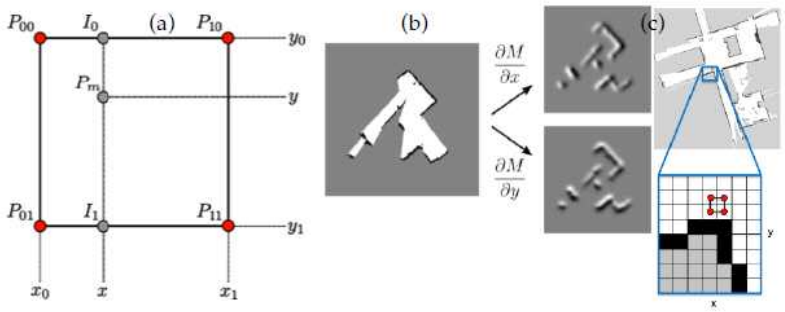


Figure 3.4: *Hector_SLAM filter method.* „(a) Bilinear filtering of the occupancy grid map. Point P_m is the point whose value shall be interpolated. (b) Occupancy grid map and spatial derivatives."[18] (c) A small zoom of the whole grid map.

로봇의 자세 추정은 간단한 반복으로 이루어진다. 먼저 현재의 자세 추정에 기초해 엔드포인트를 지도에 투영한다. 다음으로, 스캔 끝점의 지도 점유 확률 구배를 추정한다. 마지막으로 포즈 평가를 구체화하기 위해 가우스-뉴턴 반복을 수행한다.

$$H = \left[\nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}\right]^T = \left[\nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}\right]^T$$

로봇과 센서 데이터의 포즈 추정에 기초해 해당 지역에 포함된 모든 장애물을 가지고 잘 정의된 맵을 구축할 수 있다.

2.3. Navigation

첫 번째 단계는 이 논문의 탐색 부분에 기초해 SLAM 지도를 만드는 것이었다. 다음 단계는 네비게이션 자체다. 한편으로 자동차는 지도상의 예상 출발점에서 목적지까지 유효한 글로벌 계획을 찾아야 하기 때문에 내비게이션은 쉽지 않다. 반면에 자동차는 지역 환경에 반응해야 한다. 예를 들어 미지의 장애물이 나타나거나 차가 작은 간격을 통과해야 하는 경우. 그래서 네비게이션 부분에는 두 개의 다른 플래너가 포함되어 있다. 하나는 소위 지역 계획표고 하나는 글로벌 계획표다.
글로벌 루트는 ROS에 있는 navigation_stack 라이브러리의 global_planner에 의해 계산된다. 그것은 Dijkstra 알고리즘에 의해 이루어진다. 그래프에서 노드 A에서 B까지의 최단 경로를 찾는 알고리즘이다. 그 생각은 기본적으로 매우 직관적이다. 처음에는 전체 그래프를 취하여 하나의 노드를 시작점으로 설정하고 다른 모든 노드와의 거리를 무한대로 설정한다. 알고리즘에는 두 개의 대기열이 포함되어 있는데, 하나는 방문 노드가 모두 비어 있고, 하나는 초기화할 때 빈 노드와 다른 모든 노드가 시작 노드를 예상한다. 방문하지 않은 노드가 있는 대기열이 비어 있지 않은 경우 최소 거리를 가진 대기열을 선택하고 방문한 대로 표시한 후 새로운 최단 경로를 찾는지 확인한다. 사용할 수 있는 값이 하나 있으면 최단 경로의 새 값으로 설정한다.

```
dist[s] ←0                                (distance to source vertex is zero)
for all v ∈ V-{s}
    do dist[v] ←∞                         (set all other distances to infinity)
S←∅                                       (S, the set of visited vertices is initially empty)
Q←V                                       (Q, the queue initially contains all vertices)
while Q ≠∅                                (while the queue is not empty)
do  u ← mindistance(Q,dist)               (select the element of Q with the min. distance
    S←S∪{u}                               (add u to list of visited vertices)
    for all v ∈ neighbors[u]
        do if  dist[v] > dist[u] + w(u, v)        (if new shortest path found)
           then    d[v] ←d[u] + w(u, v)           (set new value of shortest path)
                                                  (if desired, add traceback code)
return dist
```

Pseudocode Dijkstra algorithm.

Dijkstra 알고리즘은 전체 그래프에 음의 전환이 포함되지 않은 경우에만 사용할 수 있다. SLAM 알고리즘에 의해 생성된 2D 그리드 맵의 경우가 이에 해당한다. 모든 픽셀을 하나의 노드로 하여 전체 맵을 그래프로 볼 수 있다. 이 픽셀들 사이의 전환은 항상 같은 값을 가지며, 장애물에 대한 원성을 예상한다. 전이가 전혀 없다. 설정된 시작점은 시작 노드로, 목적지는 엔드 노드로 보인다. 이런 방법으로 전체 지도에서 자동차의 최단 경로를 얻을 수 있다.
그 다음으로, 자율주행차를 올바르게 항해하기 위해 필요한 것은, 지역항로를 특정하는 현지 플래너다. 글로벌 플래너는 기록된 지도를 바탕으로 A에서 B까지의 최단거리 노선만 계획하지만, 이 노선에 미지의 장애물이 있다면 차가 충돌할 것이다. 한 가지 더 신경 써야 할 것은 자동차의 애커맨 조향장치여서 최소한의 회전반경을 가지고 있다. 이 논문에 사용된 차는 2미터 정도 된다. 이 글로벌 기획자는 로봇이 즉석에서 켤 수 있는 것과 자동차의 차원에 대해서도 신경쓰지 않고 계산한다. 이 문제를 해결하기 위해 특히 Ackermann 조향장치가 있는 로봇과 같은 자동차용 내비게이션 스택용 ROS 라이브러리인 teb_local_planner를 사용한다. teb_local_planner는 LiDAR 센서 데이터를 알고리즘에 통합하여 주행 중 미지의 장애물을 탐지하므로, 필요할 경우 자발적으로 로컬 경로를 재계산한다. 지역 계획자는 또한 그 경로를 기반으로 속도 명령을 계산한다. 이 계획자는 많은 구성이 필요하다. 왜냐하면 그것은 입력으로서 많은 다른 정보를 필요로 하기 때문이다. 그들은 모두 계산 결과와 성능에 영향을 미친다. 예를 들어 최소 조향각, 차량의 속도 및 노선이 장애물로부터 얼마나 멀리 떨어져 있어야 하는지를 아는 치수를 예로 들 수 있다. 성능향상에 대해서는 임시지도의 크기와 그 해상도도도 정의할 수 있지만, 해상도가 낮을수록 장애물을 검출할 수 있다. teb_local_planner 매개변수의 전체 문서는 ROS wiki의 해당 절에서 사용할 수 있다. 이 논문에 사용된 구성 값은 부록 B 장에서 찾을 수 있다.

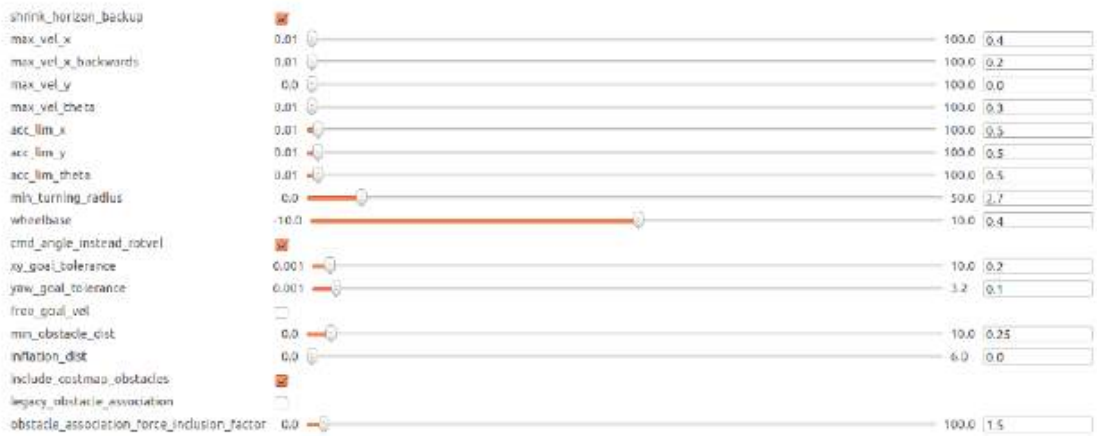


Figure 3.5 *Examples of some teb_local_planner arguments*. This picture shows a few of the required parameters for the *teb_local_planner*

2.4. Localization

잘 수행된 네비게이션에 성공하기 위해서는 자율주행차가 지도 내에서 정확히 국산화되어야 한다. 주행의 시작과 전체 경로에서 차량의 오데오미터는 속도와 조향 각도와 같은 속도 명령에 기초하여 계산된다. 하지만 차는 추정 지점에서 시작되기 때문에 타원법이 전혀 정확하지 않다. 지도 내에서 자동차의 보다 정확한 위치를 파악하기 위해서는 현지화 알고리즘이 필요한 이유다. 이것은 지역 계획자에 의한 정확한 경로의 완벽한 건축을 극대화한다.
이를 위해 적응형 몬테카를로 현지화(amcl)를 접근법으로 삼는다. ROS용으로 구현된 입자 필터를 이용해 지도에서 로봇을 현지화하는 알고리즘이다. 알고리즘은 마치 hector_slam 라이브러리에서 한 것처럼 레이저 센서에 의해 만들어진 알려진 지도를 필요로 하며, 그 동작과 감지를 바탕으로 지도 내에서 자동차의 포즈를 추정하는 것이 과제다. 알고리즘은 로봇의 자세의 초기 신뢰와 함께 시작하는데, 이 경우 차량의 예상 출발점은 무엇인가? 로봇의 상태는 현재의 모든 시간 단계 k에서 추정될 필요가 있다. 이 문제는 x를 상태 벡터로 하여 "모든 측정에 대해 조건화된 현재 상태의 후위 밀도 p(xk|Zk)를 구성하는 데 관심이 있는 베이시안 필터링 문제의 한 예다.

$$p(x_k|Z^k) \qquad x = [x, y, \theta]^T \tag{6}$$

몬테카를로 필터를 사용하는 특별한 경우 밀도는 N개의 랜덤 입자로 표현된다.

$$S_k = \{s_k^i; i = 1..N\}$$

적절한 현지화를 위해서는 각 단계마다 밀도를 재귀적으로 계산해야 한다. 이것은 포식 단계와 업데이트 단계의 두 단계로 이루어진다.
예측 단계에서 모션 모델은 로봇의 현재 위치를 예측하는 데 사용된다. time-step k의 state x는 control input Uk-1이 알려진 이전 time-step k-1에서만 정상이다. 모션 모델은 조건 밀도로 표시된다.

$$p(x_k|x_{k-1}, u_{k-1})$$

Bayesian 필터링에서 상태 벡터에 대한 예측 밀도는 통합에 의해 계산된다.

$$p(x_k|Z^{k-1}) = \int p(x_k|x_{k-1}, u_{k-1}) p(x_{k-1}|Z^{k-1}) dx_{k-1}$$

몬테카를로 국산화 작업은 이전의 시간 단계에서 계산한 Sk-1의 입자 집합으로 시작하고 밀도 p(xk|si k-1 , uk-1)로부터 샘플링하여 각 입자 S^i k-1에 모션 모델을 적용한다. 그래서 각 입자 S^i(k-1)에 대해 샘플 S^i(k)가 그려진다.
업데이트 단계에서 측정 모델을 사용하여 센서로부터의 정보를 통합하여 (6)에 기술된 밀도 함수를 구한다. 각 측정 Zk는 이전의 측정과는 조건적으로 독립적이며 측정 모델은 가능성 측면에서 제공된다. 로봇이 주어진 위치 Xk에서 Zk를 관찰한다는 뜻이다.

$$p = (z_k|x_k)$$

Xk에 대한 후위 밀도는 현재 Bases 정리를 사용하여 구한다.

$$p = (x_k|Z^k) = \frac{p(z_k|x_k) p(x_k|Z^{k-1})}{p(z_k|Z^{k-1})}$$

몬테 카를로스 국산화 알고리즘은 첫 번째 단계에서 생성된 샘플 Sk의 중량 M^i(k)에 의해 측정 Zk와 가중치를 고려한다.

$$m_k^i = p(z_k|s_k^{'i})$$

그런 다음 각 j = 1에 대해 {S^i(k) , M^i(k)}의 샘플 sj k가 그려진다.N. 전체 알고리즘은 재귀적으로 계산된다. 이전 p(X)의 임의의 샘플 S = {Si}로 필터 k = 0을 초기화한다.

Results

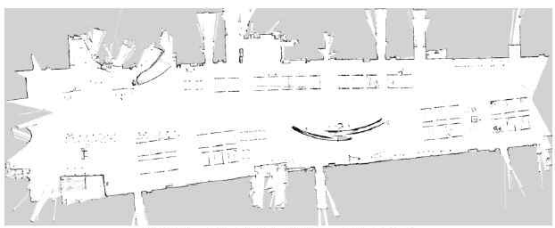
1. The map

나는 LiDAR 센서를 제어하는 방법이 다른 실내 환경에서 지도 알고리즘을 시험했다. 가장 신경 써야 할 것은 레이저 센서가 항상 위치가 있거나 포즈 추정과 지도 제작이 실패한다는 점이다. 센서를 손에 쥐고 주변을 뛰어다니며 지도를 만들려 했지만 너무 많이 흔들려 정확한 지도를 만들었다. 문제는 센서가 한 층만 스캔하기 때문에 위치가 곧지 않으면 일치하는 부분이 작동하지 않는다는 것이다. LiDAR 센서의 위치가 잘 고정되어 있어 전체 실내의 정확한 SLAM 지도를 만드는 데 완벽하게 효과적이다. 또 다른 문제는 유리문과 같은 장애물이 하나로 감지되지 않는다는 것이다. 알고리즘을 시작하기 전에 픽셀 단위의 맵 크기를 정의할 필요가 있다.

Figure 4.1: *SLAM map of a main floor*. The map shows the main floor of my home. The stripes on the right side of the picture shows perfectly the problem of a glass door as obstacle.

모든 장애물은 검은색 픽셀로 표시되며, 모든 여유공간은 자동차가 주행할 수 있는 넓은 옅은 회색 영역이다. 나의 집은 SLAM 지도의 작은 예에 불과하다. 의자 위에 리다 센서와 노트북을 고정시켜 녹화한 뒤 이리저리 옮겼다. 사진 오른쪽의 줄무늬는 센서가 유리문을 장애물로 감지하면 어떤 일이 일어났는지 완벽하게 보여준다. 레이저 빔이 이런 종류의 표면에 반사되지 않기 때문에, 그것은 감지되지 않는다.
다음 시험은 자동차에 리다르를 고정시킨 상태로 지도를 기록하는 것이었다. 왜냐하면 그 후의 내비게이션 부분의 경우 지도는 정확한 높이에 기록되어야 하기 때문에 모든 장애물을 amcl 알고리즘에서 다시 맞출 수 있다. 지도를 만들기 위해 나는 손으로 조종하여 차를 움직였다.

(a) Robotics lab of the TUM chair VI



(b) Hall of the FMI building of the TUM

Figure 4.2: *SLAM map of TUM rooms*. These tow pictures present the slam map of the (a) robotic lab of the chair VI and of the (b) hall of the FMI building of the Technical University Munich.

실험실의 지도는 단지 하나의 방에 불과하며, 단지 LiDAR의 부착과 위치를 시험하기 위한 것이었다. 그 홀의 큰 지도는 알고리즘이 거대한 실내에서도 잘 작동한다는 증거다. 그것은 학생들을 위한 두 개의 유명한 슬라이드와 모든 식사 벤치와 같은 모든 장애물을 매우 정확하게 보여준다. 이 지도는 내비게이션과 현지화 부분의 모든 진행 중인 테스트의 기본이다. 그것은 많은 장애물을 가지고 있는 거대한 크기 때문에 시험 영역으로서 완벽하지만, 몇몇 시험 실패 시 부서질 수 있는 비싼 것 같은 어떤 중요한 것들도 없다.

Conclusion

본 논문에서는 ROS를 운영체제로 하여 완전 자율주행모델을 실시한다. 자동차의 전체 내비게이션은 기록된 SLAM 지도를 기반으로 한다. 그 차는 LiDAR 센서와 지도를 이용하여 지역을 항해할 수 있다. 그것은 지도와 모든 차트가 있는 장애물을 통해 유효한 경로를 성공적으로 찾을 수 있다. 또한 주행 중 미지의 장애물에 자발적으로 반응하여 필요할 경우 경로를 재생할 수 있다.
이 지도는 ROS용 hector_slam 도서관이 작성한 2D 그리드 기반 SLAM 지도. LiDAR 센서 데이터와 로봇의 근사 위치를 바탕으로 작성됐다.
자동차의 전체 내비게이션 부분은 두 개의 다른 플래너로 나뉜다. 하나는 추정 출발점에서 정해진 목표까지 최단 경로를 계산하는 글로벌 플래너다. 이 계산은 기록된 지도와 Dijkstra 알고리즘 내의 모든 차트에 기초한 것이다. 두 번째 계획자는 현재 사건들에서 가장 좋은 경로를 찾기 위한 지역 계획자다. 그것은 만약 LiDAR 센서가 미지의 장애물을 탐지한다면 국지적인 경로를 재확보할 것이다. 순위 없이 그 길을 따라갈 수 없다면, 그것은 또한 그 길을 다시 만들 것이다. 지역 계획자는 속도 명령도 계산한다. 지도 내에서 그리고 전체 드라이브 중에 로봇을 현지화하기 위해 몬테카를로 현지화 알고리즘을 사용한다. 환경을 스캐닝하고 측정값이 지도 내에서 도표로 표시된 장애물과 일치하는지 여부를 점검한다. 이 매칭은 로봇의 모션 모델에 기초한다. 이러한 데이터를 바탕으로 알고리즘이 추정할 수 있는 것은 지도 내의 현재 위치와 방향이다.

휴.....저장을 잘하자^^ -번역 문지양-