

ROS-sim Homework

Seolyoung Jeong, Ph.D.

경북대학교 IT 대학

HW02)

Motion Control and Mapping

Motion Control and Mapping

◆ Gazebo 기반 Turtlebot 경로 이동

- Gazebo에서 Turtlebot을 이동할 경로를 정하고 .txt 파일로 저장
- .txt 파일을 읽어서 로봇을 이동하고 그 결과를 Gazebo에서 시뮬레이션
- OpenCV를 통하여 시각화 (visualization)

Motion Control and Mapping

◆ 로봇 이동 경로 저장 (.txt)

- Gazebo 실행 (turtlebot3 패키지)
- turtlebot world model 추가
- keyboard teleoperation으로 로봇을 이동
- /odom 정보를 확인하면서 이동할 경로를 임의로 정한다.
- 출발점 위치를 (x_0, y_0) , 도착점을 (x_{n-1}, y_{n-1}) ,
중간 경유 점은 (x_1, y_1) , (x_2, y_2) , (x_3, y_3) , 등으로
.txt 파일을 생성하고 저장
- 출발, 도착점을 포함하여 최소 10개의 점의 좌표 저장
- 하나의 경로는 하나의 파일에 저장한다.
- .txt 파일의 첫줄에 경로점의 개수를 저장해도 됨
- 로봇이 이동할 경로는 장애물을 피해서 설정하여야 함

Motion Control and Mapping

◆ Turtlebot world model 추가 방법

- Launch file 사용

```
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

- Gazebo → insert 탭에서 추가

Motion Control and Mapping

자율주행 node 프로그램 구현

◆ 경로파일을 이용하여 자율주행

- .txt 파일을 열어서 경로 정보를 읽는다.
- 로봇의 초기위치는 (turtlebot3의 gazebo 시작 위치) (x_0, y_0)
- 이동할 위치정보 $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_{n-1}, y_{n-1})$ 은 .txt file에서 읽어서 사용

◆ 로봇의 이동

- 시작점에서 순서대로 이동 정보를 (ω, δ) 로 변환하고 수업에서 실습한 turtle_position_move.cpp를 참고하여 로봇이 스스로 이동
- 회전 및 이동 속도는 적당히 천천히 설정 (너무 빠르지 않도록)
- 경유지점에서 1초 쉬었다가 다시 회전 및 이동 반복함
- Gazebo에서 이동하는 로봇을 시뮬레이션
- 노드 프로그램의 콘솔에서 로봇의 현재 위치도 출력해야함 (/odom topic 에서 읽어서)

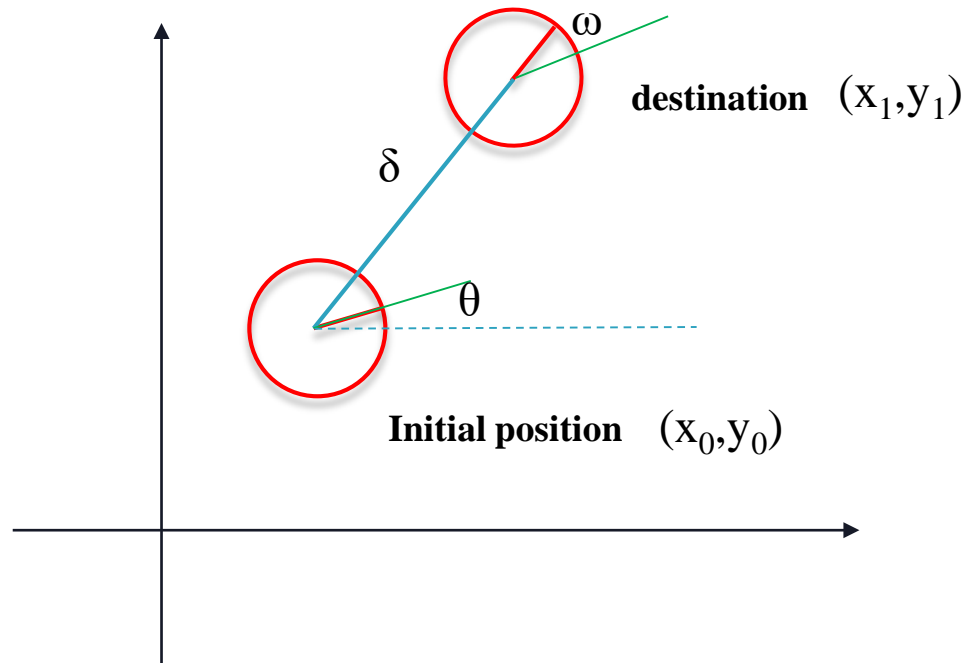
Motion Control and Mapping

- ◆ **OpenCV로 윈도우창을 만들고 로봇의 위치와 이동 경로 표시**
 - Gazebo로 로봇의 이동을 확인한다.
 - OpenCV로 윈도우를 생성하고, 로봇의 위치와 이동 경로를 실시간으로 시각화 한다.
 - 로봇의 시작점, 목표점, 경유점 등은 circle로, 이동경로는 line으로 표시 (경로를 미리 표시하지 말고 로봇이 $(x1, y1)$ 에서 $(x2, y2)$ 로 이동하면 $(x1, y1)$ 과 $(x2, y2)$ 사이의 경로는 직선으로, 그리고 $(x2, y2)$ 에 circle로 표시)
 - gazebo와 opencv window의 로봇의 위치는 서로 일치해야 함

Motion Control and Mapping

◆ 초기위치 (x_0, y_0) 이고 목적지가 (x_1, y_1) 일때 로봇의 회전 및 이동 (ω, δ)

- $\delta = \text{sqrt}((x_1 - x_0)^2 + (y_1 - y_0)^2)$
- $\omega = \tan^{-1}((y_1 - y_0)/(x_1 - x_0))$



Motion Control and Mapping

◆ OpenCV window 화면에 lidar 스캔 정보의 표시

- OpenCV 화면에는 경로의 정보 표시 및 lidar 스캔 정보도 실시간으로 표시
- Lidar 정보는 로봇의 현재위치에서 획득한 scan 정보만 표시함
- 실습자료 turtlebot3_lidar_viewer.cpp 참조

◆ 최종 화면 출력

- .txt 경로 정보
- Gazebo (turtlebot3 시뮬레이션)
- OpenCV window (경로 및 scan 정보 시각화)
- ROS node 화면 (로봇의 위치 (x,y) 출력)

Robot Motion and OpenCV

◆ ROS 및 OpenCV 출력 화면의 예

The image shows a Gazebo simulation environment with a robot (a small black and red vehicle) moving through a field of white cylinders. The environment is bounded by green walls. A terminal window in the bottom left shows ROS commands and output. A blue box in the top left contains a file named 'Robotpath.txt' with a list of coordinates. A blue box in the top right shows a graph of the robot's path and scan data. A blue box in the bottom left shows the robot's current position (x,y).

Robotpath.txt

```
10
0 0.0 0.0
1 1.5 1.2
2 2.4 1.6
3 2.8 3.4
4 .....
```

로봇 이동 시뮬레이션

로봇의 경로 및 scan 출력

로봇의 위치 (x,y) 출력

Terminal Output:

```
ros@ubuntu:~$ roslaunch image_view image_view.launch camera:=/camera/rgb/image_raw
[ INFO ] [1469753964.484174747]: Using transport "raw"
^Cros@ubuntu:~$ roslaunch image_view image_view.launch camera:=/camera/rgb/image_raw
[ INFO ] [1469753964.484174747]: Using transport "raw"
terminate called after throwing an instance of 'boost::exception_detail::clone_base'
  what():  boost::mutex lock failed in pthread_mutex_lock: Invalid argument
Aborted (core dumped)
ros@ubuntu:~$ roslaunch image_view image_view.launch camera:=/camera/rgb/image_raw
[ INFO ] [1469754025.303750076]: Using transport "raw"
^Cros@ubuntu:~$ roslaunch image_view image_view.launch camera:=/camera/depth/image_rect
[ INFO ] [1469754093.285496492]: Using transport "raw"
```

Path Graph Data:

Point	X	Y
0	0.0	0.0
1	1.5	1.2
2	2.4	1.6
3	2.8	3.4
4	2.8	3.4
5	2.4	1.6
6	1.5	1.2

Status Bar: Steps: 1 Real Time Factor: 0.75 Sim Time: 00:00:01:46.814 Real Time: 00:00:02:14.818 Iterations: 106814 FPS: 44.2178 Reset Time

HW03)

Collision Avoidance using Lidar Scan

Collision Avoidance using Lidar Scan

◆ Gazebo 기반 Turtlebot 장애물 감지 및 이동

- turtlebot world 모델에서 로봇을 이동
- 전방에 장애물이 발견되면 로봇을 회전하여 다시 이동 반복
- 로봇이 장애물에 부딪히지 않도록 함

Collision Avoidance using Lidar Scan

◆ 로봇의 이동 및 장애물 탐지

- 로봇은 기본적으로 직진한다고 가정 (linear velocity)
- lidar 스캔 정보를 subscribe하고 로봇의 좌표에서 x축을 기준으로 $\pm \Phi$ 각도 이내의 3차원 점들의 거리값의 평균이 δ meter 이내 이면 로봇을 정지하고 $\pm \theta$ 로 회전함
- Φ, δ, θ 는 임의로 정함
- 로봇의 회전방향은 + 또는 - 를 랜덤으로 정함
- 로봇을 회전 후 다시 직선 이동함
- 로봇이 직선이동 및 회전은 적절히 천천히 이동해도 됨

Collision Avoidance using Lidar Scan

◆ 로봇 이동 및 장애물 탐지 node 구현

- 로봇의 위치 정보(x,y)는 /odom에서 가져와서 출력
- 전면 장애물까지의 평균거리 δ 를 계속 출력
- 장애물이라고 판단되면 terminal 화면에 "장애물발견"으로 출력

Homework 제출

◆ HW02 결과물

- 구현한 node program 및 txt 파일
- node 실행 Ubuntu 화면 동영상 캡처 (동영상 파일 또는 URL)
 - 화면내용: 실행 terminal + Gazebo + OpenCV + txt파일

◆ HW03 결과물

- 구현한 node program
- node 실행 Ubuntu 화면 동영상 캡처 (동영상 파일 또는 URL)
 - 화면내용: 실행 terminal + Gazebo + OpenCV

◆ LMS 제출

- 제출기한: ~5.22(수)
- HW02, HW03에 각각 업로드

Any Questions...
Just Ask!

