



## 저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

석사학위논문

Autonomous Driving Control of  
Mobile Robots Based on ROS Using  
Deep Learning

동아대학교 대학원

전자공학과

길기종

2017학년도

# Autonomous Driving Control of Mobile Robots Based on ROS Using Deep Learning

by  
GIL KI JONG

Submitted to

The Graduate School of Dong-A University in Partial Fulfillment of the  
Requirements for the Degree of Master of Science  
in  
Electronic Engineering

December 2017

# Autonomous Driving Control of Mobile Robots Based on ROS Using Deep Learning

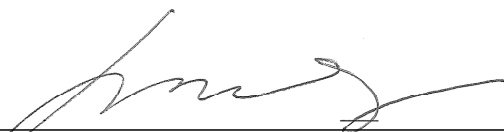
by GIL KI JONG

I have examined the final copy of this dissertation for format and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electronic Engineering.

  
Committee Chair Dr. 강 봉순

We have read this dissertation (thesis) and recommend its acceptance:

  
Committee Vice-chair Dr. 김 종 욱

  
Committee Member Dr. 이 기 동

## 국문초록

# Deep learning을 이용한 ROS기반의 모바일 로봇의 자율 주행 제어

전자공학과 길 기 종

지도교수 김 종 욱

최근 영상처리 기술의 발전과 인공지능 및 Deep learning 기술의 발전으로 말미암아 자율주행 자동차가 앞으로의 시장을 주도할 차세대 기술로 주목되고 있다. 그에 맞추어 본 논문에서는 ROS(Robot Operate System) 기반의 모바일 로봇인 ‘터틀봇3’를 활용하여 라인의 영상처리 이미지 데이터를 추출한 후 라인을 트래킹하는 자율 주행 제어를 구현하였다. 영상처리는 ‘OpenCV’를 기본으로 하여 Canny 알고리즘과 Houghline 알고리즘을 통해 라인을 검출한 후 이미지 데이터를 ROS의 메시지 통신을 통해 전송되고, 전송된 데이터를 바탕으로 모바일 로봇의 주행을 제어한다. Deep Learning을 이용하여 모바일 로봇의 주행 기록을 학습시켜 모바일 로봇의 주행을 보완한다.

주요어: Deep Learning, ROS, Image processing, Autonomous driving, Mobile robot

## Table of Contents

Table of Contents .....	ii
List of Figures .....	iv
List of Tables .....	V
I . Introduction .....	1
II. Outline Mobile robot .....	3
1. Introduction of Mobile robot .....	3
<i>a. Configuration of Turtlebot3 Burger .....</i>	<i>3</i>
<i>b. Motors in Turtlebot3 Burger .....</i>	<i>5</i>
<i>c. Sensors in Turtlebot3 Burger .....</i>	<i>5</i>
2. Modeling of Mobile robot .....	7
III. Image processing .....	9
1. Canny .....	9
2. Hough transform .....	11
3. Line detecting & tracking .....	13
<i>a. Line detecting .....</i>	<i>13</i>
<i>b. Line tracking .....</i>	<i>15</i>

IV. ROS .....	18
1. Introduction of ROS .....	18
2. Message communication .....	19
3. Definition of Messages .....	21
V. Outline Deep Learning .....	23
1. Definition of Deep learning .....	23
2. Introduction of Tensor Flow .....	24
VI. Experiment .....	25
1. Experiment environment .....	25
<i>a. Introduction of track</i> .....	25
<i>b. specification of PC</i> .....	26
2. Experiment flow chart .....	26
3. Packages used in the experiment .....	27
4. Experiment results .....	31
VII. Conclusion .....	34
Reference .....	35
Abstract .....	37

## List of Figures

Fig. 2.1 Turtlebot3 Burger .....	4
Fig. 2.2 Architecture of XM-430 .....	5
Fig. 2.3 Sensors in Turtlebot3 Burger .....	6
Fig. 2.4 Kinematics of Turtlebot3 Burger .....	7
Fig. 3.1 Area by Orientation .....	10
Fig. 3.2 A line represented by $\rho - \theta$ parameters .....	12
Fig. 3.3 Edge detecting using Canny operator .....	13
Fig. 3.4 ROI of image .....	14
Fig. 3.5 Top and bottom of ROI .....	14
Fig. 3.6 Line edge point .....	15
Fig. 3.7 Filter in the image .....	16
Fig. 3.8 Reversal situation .....	17
Fig. 4.1 Message communication structure of ROS .....	18
Fig. 4.2 Moving direction of Mobile robot .....	22
Fig. 5.1 Structure of Tensor flow .....	24
Fig. 6.1 Track .....	25
Fig. 6.2 Flow chart of experiment .....	26
Fig. 6.3 Nodes and topic of keyboard .....	28
Fig. 6.4 Nodes and topic of image processing .....	29
Fig. 6.5 Result of manual driving .....	31
Fig. 6.6 Result of autonomous driving .....	32
Fig. 6.7 Result of autonomous driving applied deep learning .....	32



## List of Tables

Table. 4.1	Built-in field types .....	20
Table. 4.2	std_msgs/Vision.msg .....	21
Table. 4.3	geometry_msgs/Twist.msg .....	21
Table. 4.4	geometry_msgs/Vector3.msg .....	21
Table. 6.1	Specification of PC .....	26

# **I . Introduction**

Autonomous driving vehicles are one of the technologies that have come into the spotlight in the area of the fourth industrial revolution. Autonomous driving vehicles recognize road conditions or changes in its surroundings, and then makes a suitable decision to drive the vehicle to its destination. Currently, global companies including Google are competing in the development of autonomous driving technology, and domestic companies are spurring technological development in large corporations, universities and research institutes. Autonomous driving technology is a cutting-edge technology that includes image processing, LIDAR sensor, ultrasonic sensor, and GPS.

According to a report by the US research firm Navigant Research in the third quarter of 2013, the size of the autonomous vehicle market will grow to 743 trillion won in 2035, and the volume of autonomous vehicle distribution in the world's three major markets (Europe, the Americas and Asia) to 85.4% annually. They predict that the autonomous vehicle market will grow at an astounding pace that far exceeds our predictions [1].

Therefore, this study intends to incorporate deep learning as a key technology of the fourth industrial revolution with autonomous driving technology. Deep learning technology has been studied for a long time, but it is a technology largely imprinted to the public based on the recently developed AlphaGo by Google DeepMind. Currently, a variety of deep learning platforms have been developed, as well as developers, who are interested in deep learning, are applying their own area.

ROS(Robot Operating System) is one of the robot software platforms, and aims to build robotics software development environment that can

collaborate with worldwide [2]. Robotics is also closely related to deep learning technology. Therefore, this study also combines the experiments on the interaction between ROS and deep learning.

In chapter 2, we introduce the ROS-based mobile robot used in the experiment. In chapter 3, we introduce the algorithm and line recognition used in image processing. In chapter 4, we introduce the ROS to understand message communication. In chapter 5, we introduce the tensor flow as deep learning platform. Finally, the experimental results are introduced in Chapter 6 and concluded in Chapter 7.

## II. Outline Mobile robot

### 1. Introduction of Mobile robot

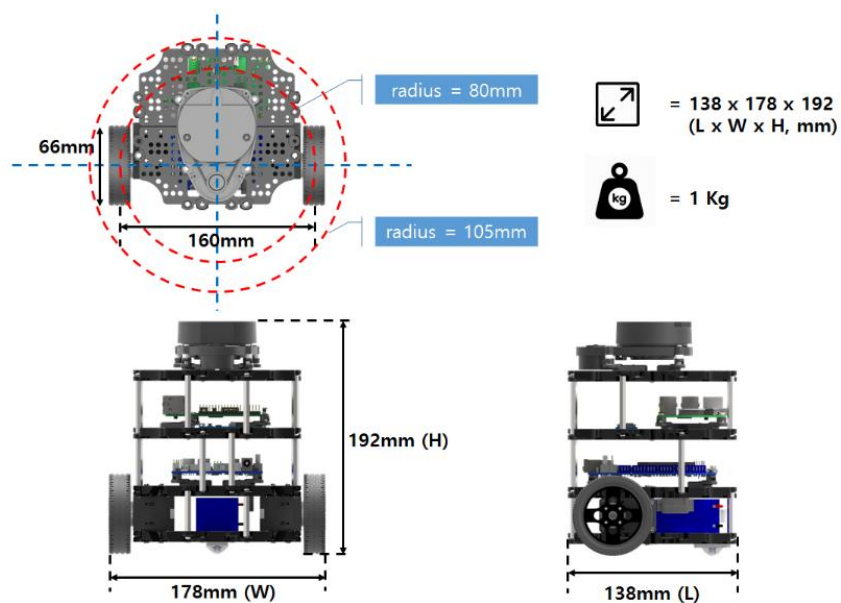
#### *a. Configuration of Turtlebot3 Burger*

In this paper, we used Turtlebot3 Burger based on ROS developed by ROBOTIS as shown in Fig. 1. The mobile robot independently controls and drives two wheels attached parallel to the lower front part. The ball caster attached to the lower rear part of the mobile robot rotates 360 degrees and serves to balance the mobile robot. Following is brief parameters of Turtlebot3 Burger [3].

- Size(L × W × H): 138mm × 178mm × 192mm
- Weight: 1kg
- Controller: OpenCR (32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS))
- SBC(Single-Board Computer): Raspberry pi3
- Actuator: Equipped with a high performance networked actuator module DYNAMIXEL XM-430
- Sensor: a 360° LIDAR, a Gyroscope 3 Axis, a Accelerometer 3 Axis, a Magnetometer 3 Axis
- External Li-polymer Battery (11.1V 1800mAh / 19.98Wh 5C)
- Maximum translational velocity: 0.22 m/s
- Maximum rotational velocity: 2.84 rad/s (162.72 deg/s)



a) Component of Turtlebot3 Burger



b) Size of Turtlebot3 Burger

Fig. 5.1 Turtlebot3 Burger

### *b. Motors in Turtlebot3 Burger*

The Turtlebot3 Burger has two DYNAMIXEL XM-430. DYNAMIXEL XM-430 is a high performance networked actuator module, which has been widely used for building multi-joint robots and mobile robots with reliability and expandability. There are 6 operating modes in XM-430: a current control mode, a velocity control mode, a position control mode, a extended position control mode, a current-based position mode, a PWM control mode. In turtle bot 3, the velocity control mode is set.

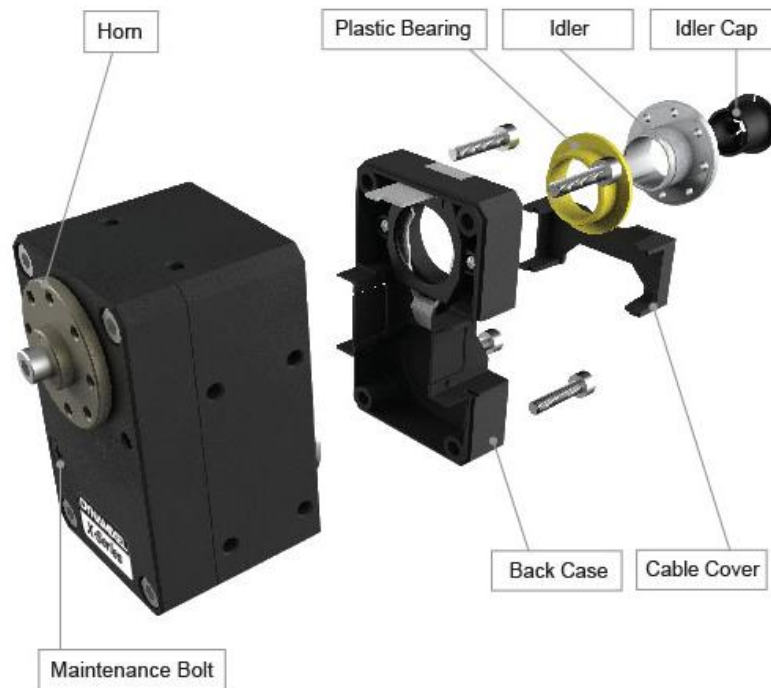
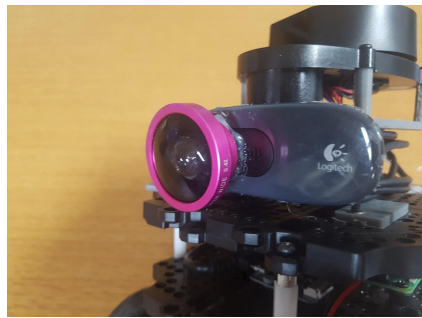


Fig. 2.2 Architecture of XM-430

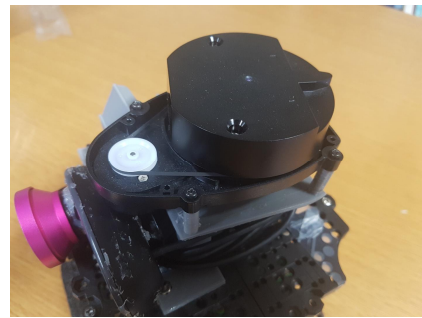
### *c. Sensors in Turtlebot3 Burger*

Figure 5 shows the webcam and the lidar sensor used in this study. The webcam is attached to the upper part of the front part of the mobile

robot and used for inputting a video image for performing autonomous driving. and the LIDAR sensor is attached to the upper end of the mobile robot and used to accurately measure the movement path of the mobile robot.



a) webcam



b) LIDAR

Fig. 2.3 Sensors in Turtlebot3 Burger

## 2. Modeling of Mobile robot

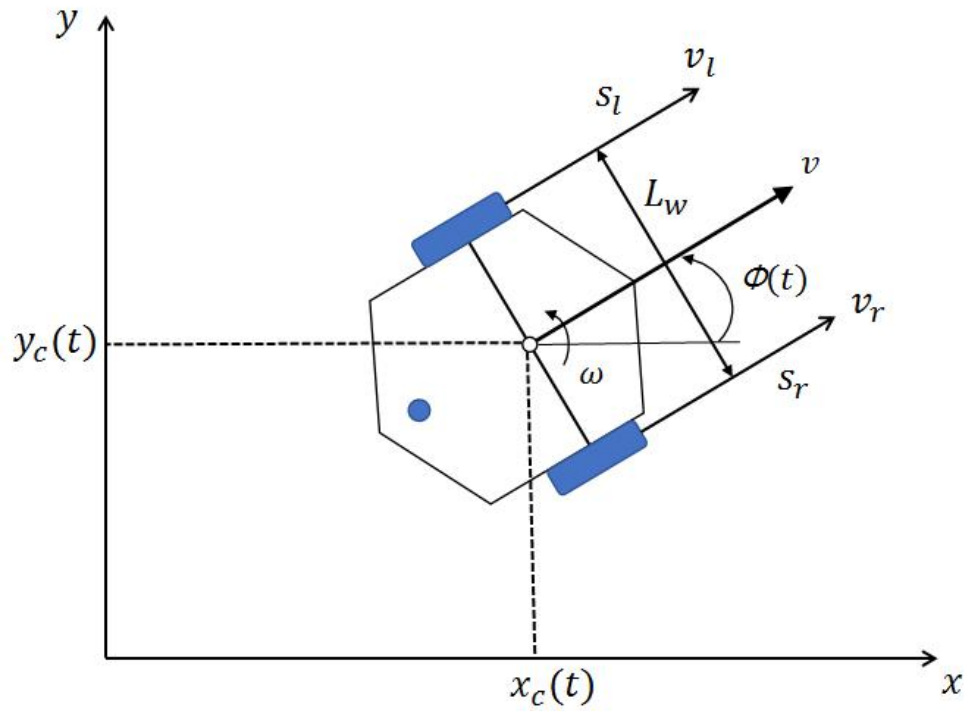


Fig. 2.4 Kinematics of Turtlebot3 Burger

Figure 2.4 shows kinematic modeling of the mobile robot.  $r$  is the radius of the wheel,  $L_w$  is the distance between two wheels.  $s_r$  and  $s_l$  are the moving distance of the right and left wheels,  $v_r(=\dot{s}_r)$  and  $v_l(=\dot{s}_l)$  are linear velocity of the right and left wheels.  $\omega_l(=\dot{\theta}_l)$  and  $\omega_r(=\dot{\theta}_r)$  are angular velocity of the right and left wheels [4].

$v$  and  $\omega$  are the moving velocity and angular velocity of the center point of the mobile robot and can be calculated from the following formula.



$$v(k) = \frac{v_r(k) + v_l(k)}{2} = r \frac{\omega_r(k) + \omega_l(k)}{2} \quad (2.1)$$

$$\omega(k) = \frac{v_r(k) - v_l(k)}{L} = r \frac{\omega_r(k) - \omega_l(k)}{L} \quad (2.2)$$

The two-dimensional coordinates  $(x_c(k), y_c(k))$  and angles  $(\phi(k))$  of the center of the wheel axis of the robot can be expressed by the following discrete-time state equation.

$$\mathbf{p}(k) = \mathbf{p}(k-1) + \begin{bmatrix} vT \cos\left(\phi(k-1) + \frac{\omega T}{2}\right) \\ vT \sin\left(\phi(k-1) + \frac{\omega T}{2}\right) \\ \omega T \end{bmatrix}, \mathbf{p}(k) = \begin{bmatrix} x_c(k) \\ y_c(k) \\ \phi(k) \end{bmatrix} \quad (2.3)$$

where,  $T$  is the sampling time and  $\mathbf{p}(k)$  is the state vector of the mobile robot.

### III. Image processing

#### 1. Canny

In this study, edge recognition was used to recognize lines. An edge is a feature that represents the boundary of an object in an image, a point where the pixel value discontinuity. Edges are generally represented in the outline of an object and have important information such as the size, shape, and location of the object [5].

There are various operators for edge detection. Among them, Canny operator is known as the most optimized edge detection method compared to other operators and widely used. Therefore, in this study, the boundary of line was detected using Canny edge operator. The Canny edge operator classifies a pixel as an edge if the gradient value of one pixel in the direction in which the change in brightness value is largest is larger than the gradient value of neighboring pixels. The Canny edge operator detects an edge from an image by the following process:

1) Smoothing of image through application of Gaussian smoothing mask or Gaussian function.

$$C(x,y) = G(x,y;\sigma) \cdot f(x,y) \quad (3.1)$$

2) Vertical and horizontal edges are detected through first order differential operator

$$P(x,y) = \frac{C(x,y+1) - C(x,y) + C(x+1,y+1) - C(x+1,y)}{2} \quad (3.2)$$

$$Q(x,y) = \frac{C(x,y) - C(x+1,y) + C(x,y+1) - C(x+1,y+1)}{2} \quad (3.3)$$

3) Determine the size and orientation of the variation for each pixel

$$M(x, y) = \sqrt{P^2(x, y) + Q^2(x, y)} \quad (3.4)$$

$$\theta(x, y) = \tan^{-1} \frac{Q(x, y)}{P(x, y)} \quad (3.5)$$

Figure 3.1 shows each area along the orientation of the pixel. It is divided into four sections.

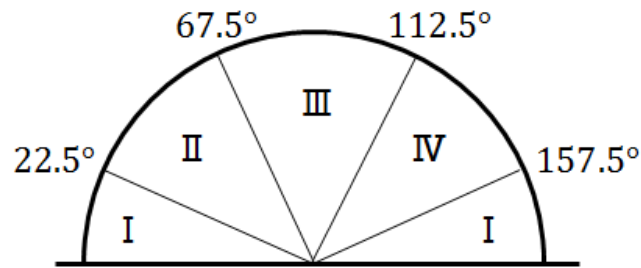


Fig. 3.1 Area by Orientation

where:

I : 0 ° ~ 22.5 ° & 157.5 ° ~ 180 °

II : 22.5 ° ~ 67.5 °

III : 67.5 ° ~ 112.5 °

IV : 112.5 ° ~ 157.5 °

4) The size of the transition of any pixel is larger than the two neighbor values in the transition direction, the pixel is marked with an edge, otherwise it is marked with a background.

5) Remove unnecessary edges with threshold

## 2. Hough transform

The Hough transform is a method of detecting the straight line according to each point when several points are distributed in the image space [6].

A pixel in an image can be represented by an infinite number of straight lines passing through that point, and each straight line can be expressed as an equation (3.6) in image space.

$$y = ax + b \quad (3.6)$$

Also, many points in the image space can be transformed from the parameter space into equation (3.7).

$$b = -ax + y \quad (3.7)$$

All points on the same straight line in the image space are represented by a large number of straight lines passing through the common point in the variable space, and the desired straight line can be detected by examining their accumulator.

However, the above method has a disadvantage in that the slope and the slice value become infinite. To remedy the disadvantages, distant-angular parameters space transformation is mainly used [7]. As shown in Figure 3.2, there is a straight line A and a straight line B perpendicular to the straight line and passing through the origin. And the angle between the straight line B and the x axis is  $\theta$ , and the distance between the intersection of the two straight lines and the origin is  $\rho$ . The transformation equation can be expressed as:

$$\rho = x \cos \theta + y \sin \theta \quad (3.8)$$

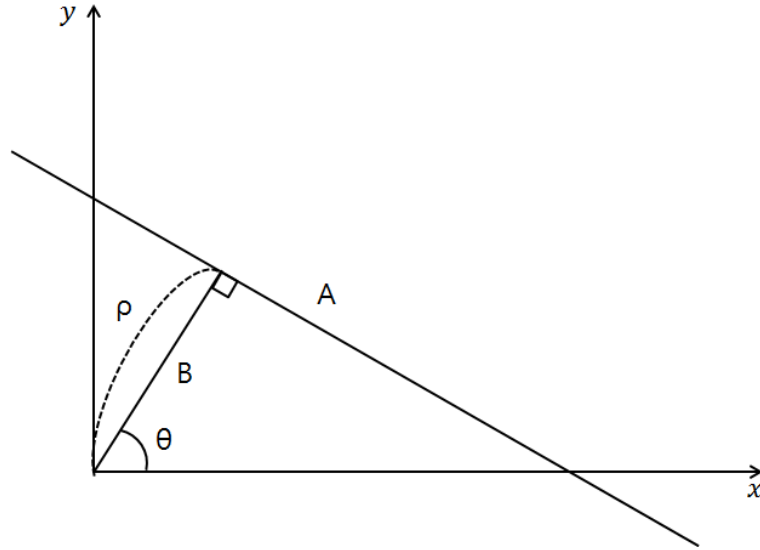


Fig. 3.2 A line represented by  $\rho - \theta$  parameters

where:

$$\rho : \rho \geq 0$$

$$\theta : 0^\circ \leq \theta \leq 180^\circ$$

Given a point in the image space, all straight lines through it are plotted as a sinusoidal curve by equation (3.9) in the parameters space.

$$\rho = x_0 \cos \theta + y_0 \sin \theta \quad (3.9)$$

Since  $\rho$  and  $\theta$  are parameters, all straight line group passing through  $(x_0, y_0)$  will draw sine curves as one trajectory in the parameter space. Therefore, all points on the straight line are drawn by several sinusoids according to equation (3.9) [8]. If one finds the point  $(p_0, \theta_0)$  at which these curves intersect the most, one straight line can be obtained by equation (3.10).

$$\rho_0 = x_0 \cos \theta_0 + y_0 \sin \theta_0 \quad (3.10)$$

### 3. Line detecting & tracking

#### *a. Line detecting*

The pixel value of the line is detected in the image ( $240 \times 320$ ) through the method described above. First, an edge is detected in an image using a canny operator.



a) Original image



b) Edge image

Fig. 3.3 Edge detecting using Canny operator

Second, It removes unnecessary part( $0 \leq y \leq 150$ ) of image processing and designates the part( $150 < y \leq 240$ ) which has only line information by using ROI.(Region Of Interest) as shown in Figure 3.4.

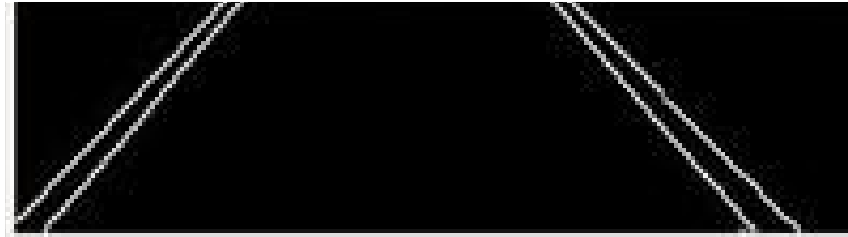


Fig. 3.4 ROI of image

Third, to obtain the angle of the line, divide the previously obtained ROI image vertically as shown in Figure 3.5.

where, a range of top image is  $150 < y \leq 170$ , and a range of bottom image is  $190 < y \leq 210$ .



Fig. 3.5 Top and bottom of ROI

Finally, we use the hough line function(libraries in OpenCV) to get the edge point of the line in the image [9]. as shown in Figure 3.6. Using the hough line function, we can get the coordinates of both ends of a straight line. The starting and ending points are divided according to the value of  $x$ . The point at which  $x$  is low is the starting point, and the high point is the end point.

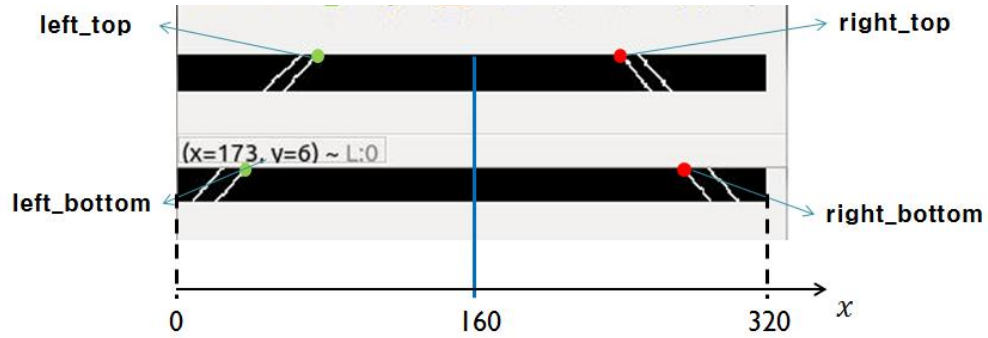


Fig. 3.6 Line edge point

Therefore, in the case of the left line, the coordinates of the end point should be obtained, and in the case of the right line, the coordinates of the starting point should be obtained by equation (3.11) and (3.12).

$$left\_top, bottom = min(x_1) \quad \text{if } x_1 > 160 \quad (3.11)$$

$$right\_top, bottom = max(x_2) \quad \text{if } x_2 \leq 160 \quad (3.12)$$

where:

$x_1$ : end point of line

$x_2$ : start point of line

#### b. Line tracking

To do line tracking, first set the initial  $x$  values of the right and left lines. In this experiment, the initial  $x$  value of the left line is 80, and the initial  $x$  value of the right line is 240. Then, the difference between the  $x$  value in the previous frame and the current frame is  $D$ .  $D$  is sensitive to FPS(Frames Per Second) and image processing time( $s$ ). They have the following relationship.



$$D \propto \frac{1}{FPS} \quad , \quad D \propto s \quad , \quad s \propto \frac{1}{FPS} \quad (3.13)$$

As FPS increases, D decreases. However, as FPS increases, s decreases and consequently D also increases. So we have to set the appropriate FPS for the image processing speed.

The figure 3.7 shows the filter on the image. In this experiment, the filter size is 30. If the size of D is smaller than F, the coordinate value of the current frame is updated.

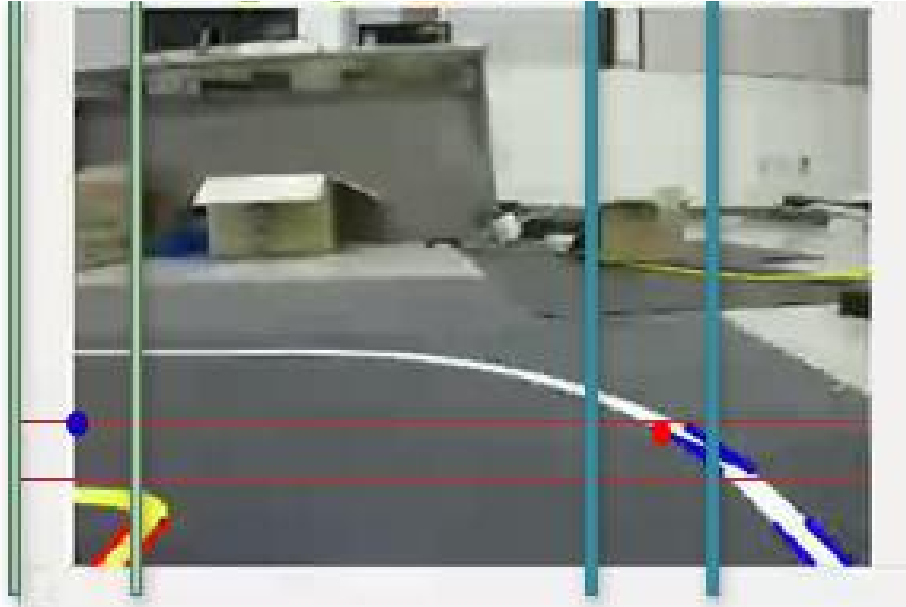


Fig. 3.7 Filter in the image

As shown in figure. 3.8, in the case of the rapid curve section, there occurs a section where the x values of the two lines become similar. A reversal situation occurs in which the x value of both lines is changed.

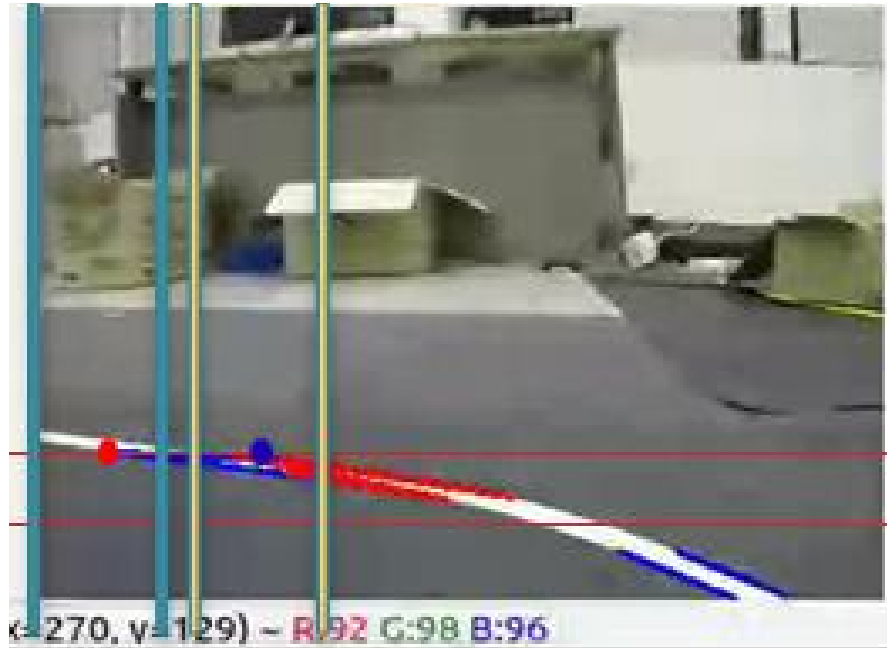


Fig. 3.8 Reversal situation

To prevent this situation, if the x value of right is less than 50, the left line detection is stopped. Similarly, if the x value of left is larger than 270, the right line detection is stopped. Track the x value of the line every frame using the method described above.

## IV. ROS (Robot Operating System)

### 1. Introduction of ROS

ROS is an open-source platform that is sponsored by the Open Source Robotics fund. ROS provides hardware abstraction, sub-device control, implementation of commonly used functions, inter-process message passing, package management, libraries required for development environment, and various development and debugging tools needed for robot application development [10]. ROS uses a traditional operating system and is a hardware abstraction concept that controls robots and sensors essential for robot application development and is a support system for user's robot application program development. ROS also supports messages from one operating system, but it can also handle messages from different operating systems and hardware programs.

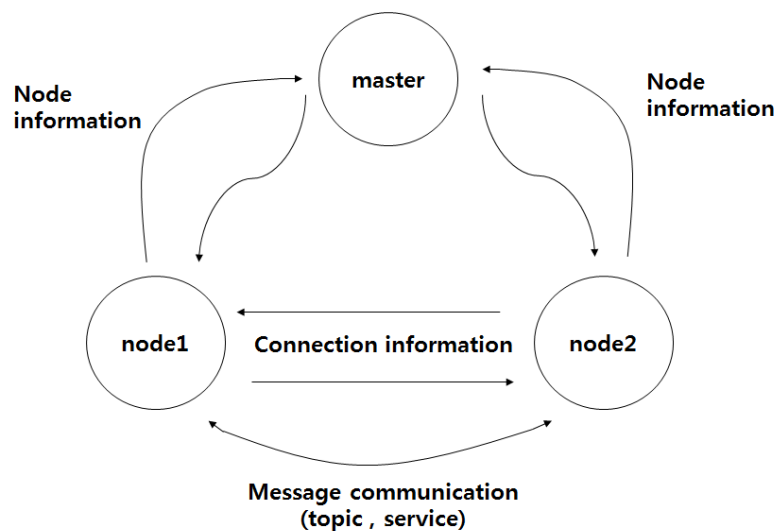


Fig. 4.1 Message communication structure of ROS

Nodes and master are the core components of ROS. A node is the smallest execution processor that runs in the ROS, and is one executable program. The master acts as a name server for node-to-node connections and message communication. When the master is run, the master registers the names of each running node and can receive information as needed. Executing nodes can send and receive data between nodes via a master in the form of a message. Figure 4.1 shows the ROS message communication structure at two nodes. Communication methods using messages include TCP / IP ROS and UDP ROS, and there are topics of unidirectional message sending and receiving method and services of bidirectional message request and response method.

## 2. Message communication

A message communication is a type of data used to transfer data between nodes. There are three types of messages communicate between nodes:

- Topic: Send and receive unidirectional messages
- Service: Request and response bidirectional message
- Action: Goal and feedback bidirectional message

Messages can be structured from simple data types such as integer, floating point, and boolean to data structures containing messages in messages such as 'geometry\_msgs', as well as arrays of messages. These messages consist of two types: field type and field name [11]. The field type is the data type and the field name can be used to access the value from the message.

table 4.1 shows that some of the built-in field types that we can use in our message:

**Table 4.1** Built-in field types

Primitive type	Serialization	C++	Python
bool	unsigned 8bit int	uint8_t	bool
int8	signed 8bit int	int8_t	int
uint8	unsigned 8bit int	uint8_t	int
int16	signed 16bit int	int16_t	int
uint16	unsigned 16bit int	uint16_t	int
int32	signed 32bit int	int32_t	int
uint32	unsigned 32bit int	uint32_t	int
int64	signed 64bit int	int64_t	long
uint64	unsigned 64bit int	uint_t	long
float32	32bit IEEE float	float	float
float64	64bit IEEE float	double	float
string	ascii string(4)	std::string	str
time	secs/nsecs unsigned 32bit ints	ros::Time	rospy.Time
duration	secs/nsecs signed 32bit ints	ros::Duration	rospy.Duration

A special type of ROS message is called message headers. Headers can carry information such as time, frame of reference or frame\_id, sequence number. Using headers, we will get numbered messages and more clarity in who is sending the current message. The header information is mainly used to send data such as robot joint transforms.

### 3. Definition of Messages

In this study, the x value of the left and right of the top image and the x value of left and right of the bottom image are transmitted through the topic. The topic is named Vision.msg and the message type is std\_msgs/float32 as shown in table 4.2. This msg file is for the purpose of this study.

**Table 4.2** std\_msgs/Vision.msg

float32	left_top
float32	left_bottom
float32	right_top
float32	right_bottom

And The topic controlling the motor of the mobile robot is named cmd\_vel, and the message type is geometry\_msgs/Twist as shown in table 4.3.

**Table 4.3** geometry\_msgs/Twist.msg

Vector3	linear
Vector3	angular

This msg file is a typical topic used to transmit translational (=linear) and rotational (=angular) velocity values. where, vector3 has the form shown in Table 4.4 below.

**Table 4.4** geometry\_msgs/Vector3.msg

float64	x
float64	y
float64	z

For translational velocity, the unit of  $x$  is m/s, and For rotational velocity, the unit of  $z$  is rad/s. For example, the  $x$  value is given as 0.05, the mobile robot advances at 0.05m/s in the  $x$ -axis direction. The  $z$  value is given as 1.0, the mobile robot rotates at a rate of 1.0 rad/s as a counterclockwise rotation about the  $z$  axis.

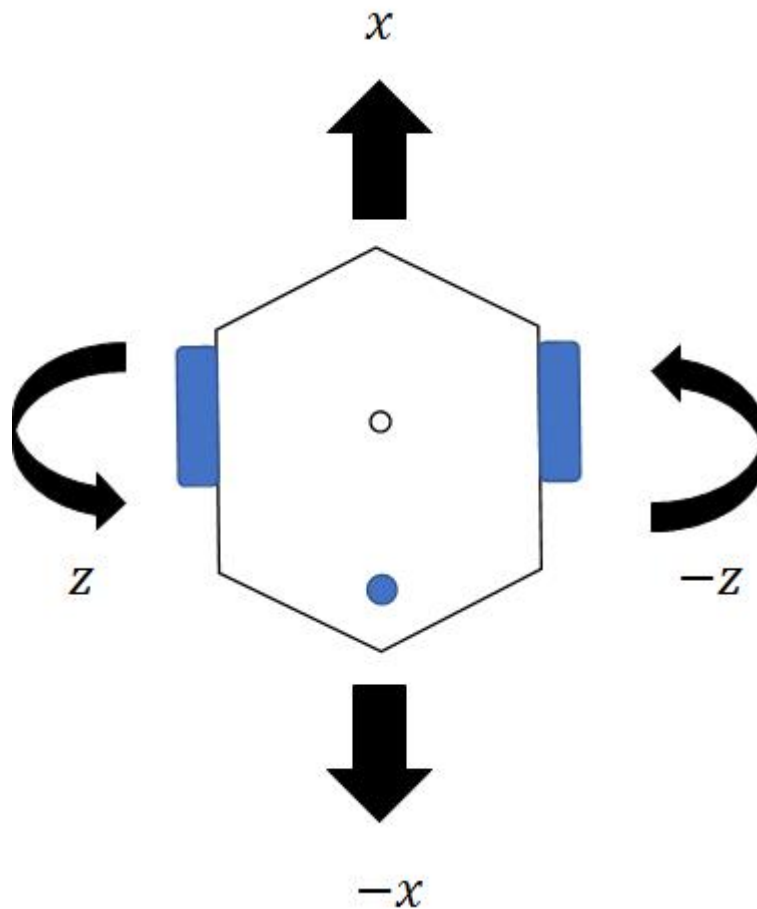


Fig. 4.2 Moving direction of Mobile robot

## V. Outline Deep Learning

### 1. Definition of Deep learning

Deep learning is defined as a set of machine learning algorithms that try to achieve a high level of simulation through a combination of several nonlinear transformation techniques. In a large framework, it can be said that machine learning is a way of teaching human thinking to computers [12].

Machine learning is a way to improve the performance of specific task through experience. There are various methods such as learning process through various experiences and data, decision trees, clustering, Bayesian networks, association rules, inductive logic programming, and genetic algorithms. Artificial neural networks are also one of the methods of deep learning.

Deep learning is an area of machine learning that has recently become a hot issue due to the development of big data, algorithms, and hardware. It refers to the problem of solving complex nonlinear problems based on artificial neural network theory using unsupervised learning. In other words, it is called pattern recognition in that it learns patterns from experiences of many events and makes judgments based on them. It is similar to the human brain mimicking the way information is handled to find patterns in a lot of data and then to identify objects [13]. Deep learning allows a computer to perceive, reason, and make decisions on its own, even if a person does not set all criteria.



## 2. Introduction of Tensor Flow

TensorFlow is an open source software library for machine learning used in Google products. It was created by the Google Brain team for research and product development within Google and was released as an Apache 2.0 open source license on November 9, 2015[1]. TensorFlow runs on mobile platforms like Android and iOS, as well as on several CPUs and GPUs on desktop or server systems on 64-bit Linux and MacOS(using the CUDA extension to perform common operations on the GPU). The tensor flow operation is represented by a stateful dataflow oriented graph.

TensorFlow provides the Python API and also provides a little C / C++ API. In this study, the tensor flow was written in Python, The input of the tensor flow is the x value of the line according to the image and cmd\_vel, the output is the angle\_gain value of the line for motor control as shown in figure. 5.1. It learns the angle\_gain value according to the image data to increase the safety of autonomous driving.

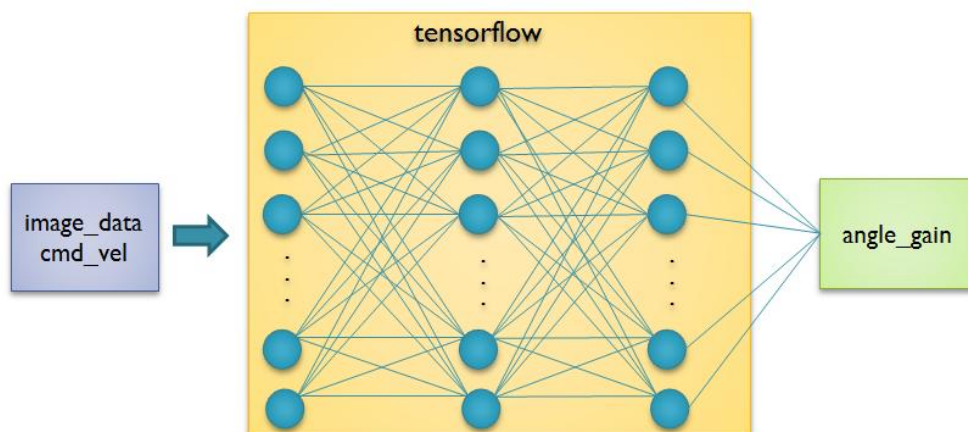


Fig. 5.1 Structure of Tensor flow

## VI. Experiment

### 1. Experiment environment

#### *a. Introduction of track*

Figure 6.1 shows the track on which the mobile robot performs autonomous driving. The track size is  $2\text{m} \times 2\text{m}$ . There are straight and curved sections, the left line is yellow and the right line is white.

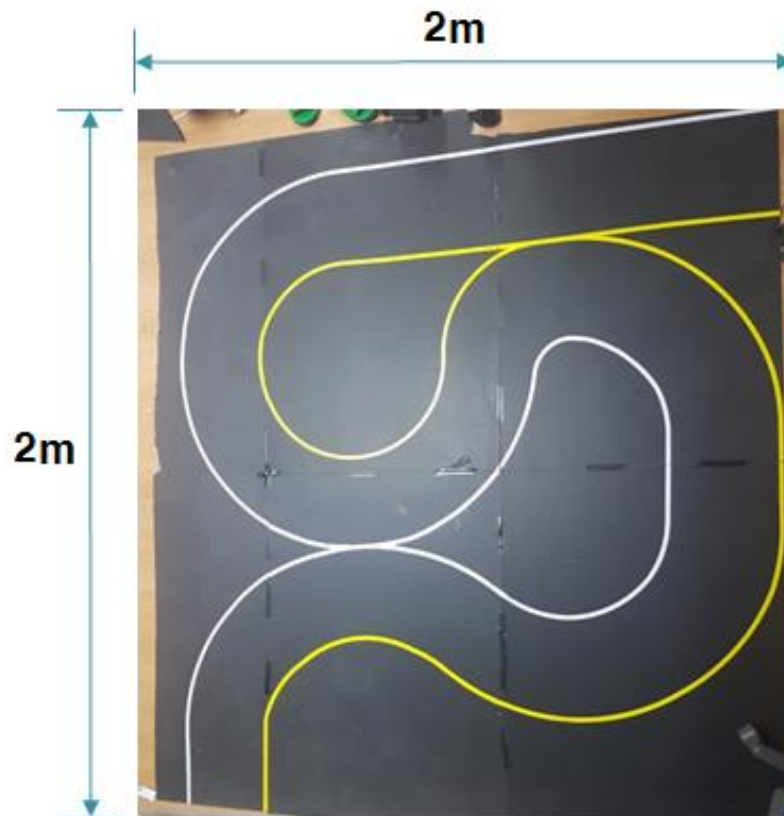


Fig. 6.1 Track

### b. Specification of PC

The specifications of PC used in this experiment are shown in the following table 6.1. This pc runs the master server of the ROS. The ROS version of the PC is the same kinetic as turtlebot3 burger.

**Table 6.1** Specifications of PC

CPU	Intel i5-825U @1.6GHz quad core
RAM	4GB DDR4
SSD	128GB
O/S	Linux ubuntu 16.04 64-bit
Wifi	802.11ac
GPU	Geforce MX150 2GB

## 2. Experiment flow chart

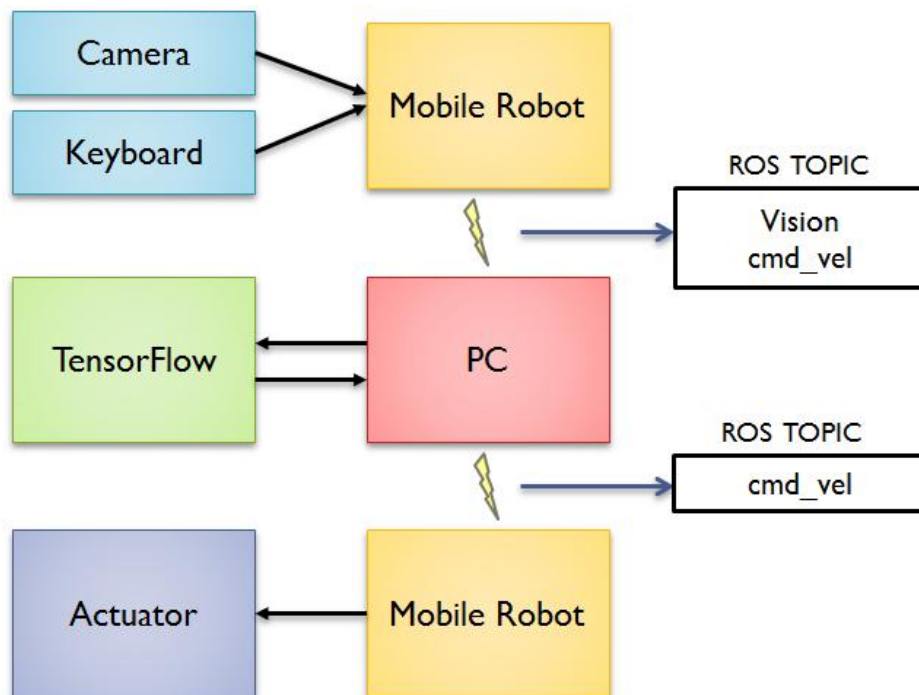


Fig. 6.2 Flow chart of experiment

The flow chart of experiment is shown in the figure 6.2. The image data is always transmitted to the PC when the mobile robot is running. And then, The `cmd_vel` obtained from autonomous driving using image processing or manual driving using the keyboard is input to the tensor flow. By outputting appropriate `angle_gain` through machine learning using tensor flow, autonomous driving of mobile robot is complemented.

### 3. Packages used in the experiment

The mobile robot used in this experiment was a robot developed by ROBOTIS, and it was used based on the basic package provided by ROBOTIS.

- ***turtlebot3***: `bringup`, `description`, `slam`, `navigation`, `teleop` etc. Basic packages that can be implemented with turtleBot3 are included.
- ***turtlebot3\_msg***: `MotorPower.msg`, `PanoramaImg.msg`, `SensorState.msg`, `SetFollowState.srv`, `TakePanorama.srv` etc. Messages or service files used in turtlebot3 are included.
- ***hls\_lfcd\_lds\_driver***: Package for using LIDAR sensor in turtlebot3

The following image processing package for autonomous driving was developed for this experiment.

- ***autonomous\_driving***: Image processing, control of mobile robot, messages files used in turtlebot3 are included.

*a. Manual driving using keyboard*

The following figure 6.3 shows the topic and nodes for manual driving using the keyboard.

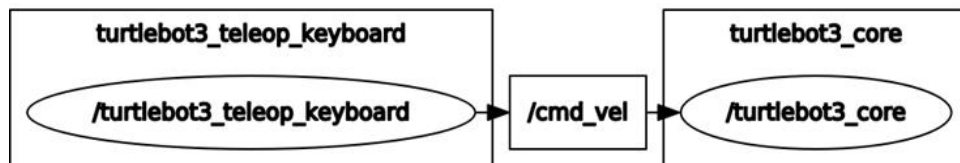


Fig. 6.3 Nodes and topic of keyboard

The command is as follows:

1) ***roscore***

Run the ROS master node.

2) ***roslaunch turtlebot3\_bringup turtlebot3\_robot.launch***

Run the OpenCR node as controller of turtlebot3, turtlebot3\_core node for communication, and LIDAR node.

3) ***roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch***

Run the turtlebot3\_teleop\_keyboard node. This node controls the translational and rotation velocity with the w, x, a, d and s key on the keyboard.

where:

- w: Increase the  $x$  value by 0.01.
- x: Decrease the  $x$  value by 0.01.
- a: Increase the  $z$  value by 0.1.
- d: Decrease the  $z$  value by 0.1.
- s: Set the  $x$  and  $z$  value by 0.

*b. autonomous driving using image processing*

The following figure 6.4 shows the topic and nodes for autonomous driving using the image processing.

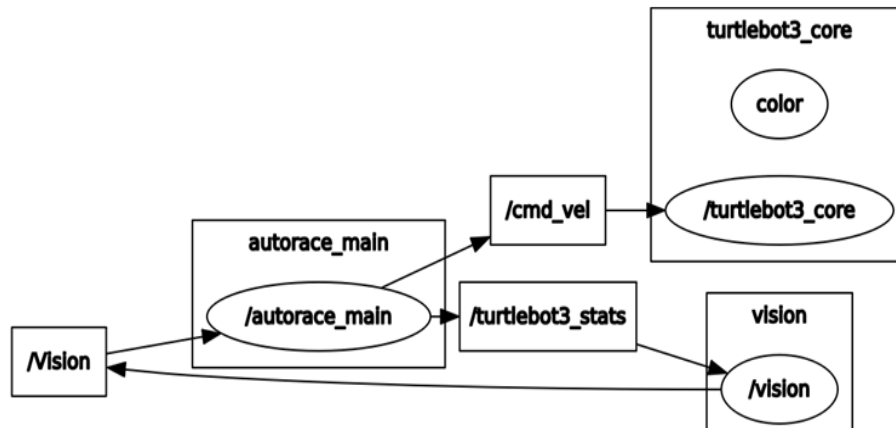


Fig. 6.4 Nodes and topic of image processing

The command is as follows:

1) ***roscore***

Run the ROS master node.

2) ***roslaunch turtlebot3\_bringup turtlebot3\_robot.launch***

Run the OpenCR node as controller of turtlebot3, turtlebot3\_core node for communication, and LIDAR node.

3) ***roslaunch autonomous\_driving cam.py***

Run the image processing node in autonomous\_driving package.

4) ***roslaunch autonomous\_driving autonomous\_main***

Run the main processing node in autonomous\_driving package. This

node calculates the cmd\_vel using the  $x$  value of the each line received from the vision topic by the following equation 6.1 and 6.2.

$$z = \frac{x_{r_1} - x_{rc}}{x_{rc}} \cdot (x_{r_1} - x_{r_2}) \cdot \alpha \quad (6.1)$$

$$z = \frac{x_{l_1} - x_{lc}}{x_{lc}} \cdot (x_{l_1} - x_{l_2}) \cdot \alpha \quad (6.2)$$

where:

$x_{r_1}$ : The  $x$  value of right\_top

$x_{r_2}$ : The  $x$  value of right\_bottom

$x_{rc}$ : The  $x$  initial value of right\_top

$x_{l_1}$ : The  $x$  value of left\_top

$x_{l_2}$ : The  $x$  value of left\_bottom

$x_{lc}$ : The  $x$  initial value of left\_top

$\alpha$ : The angle\_gain

## 4. Experiment results

Experiment results were obtained by measuring the path moved by mobile robot using LIDAR sensor.

The following figure 6.5 shows the path that mobile robot is controlled by manual driving with keyboard. Because the person controlled directly, there is a characteristic that the path has a lot of straightness. And, the path is not clear.

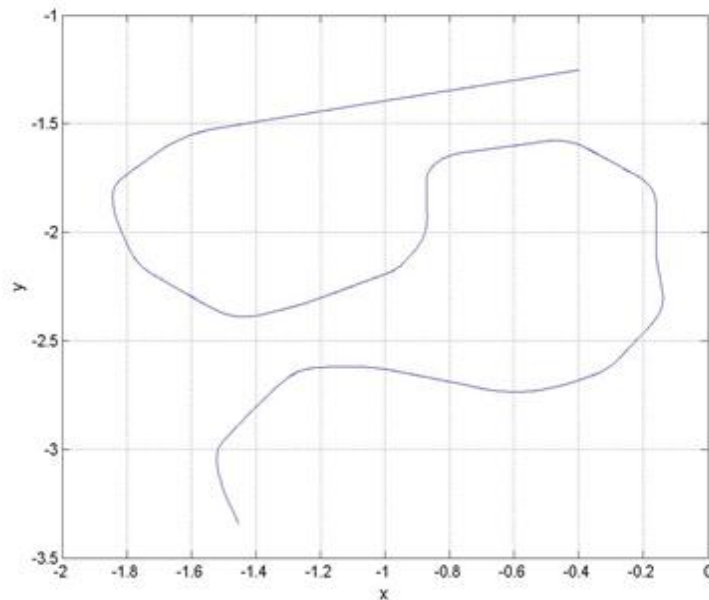


Fig. 6.5 Result of manual driving

The following figure 6.6 shows the path that mobile robot is controlled by autonomous driving using image processing. There is a characteristic that the track drives smoothly but the path deviates from a certain area. where, the value of  $\alpha$  is 2.0. This value is obtained by tuning data.



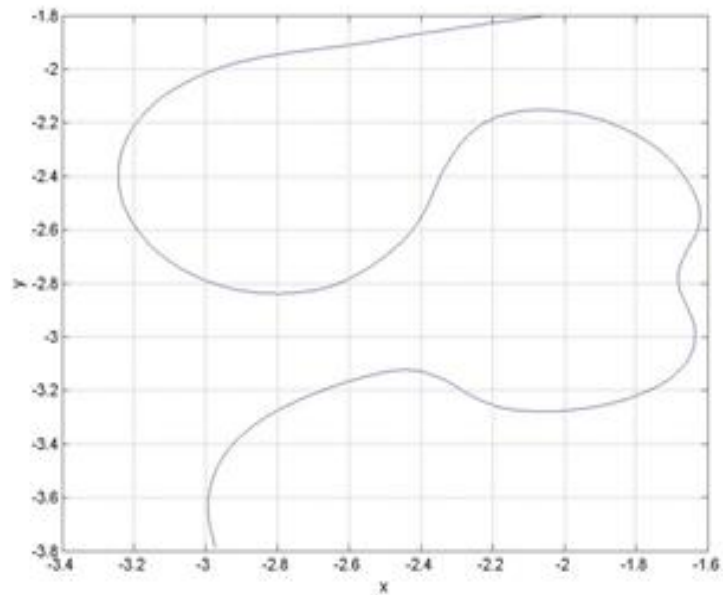
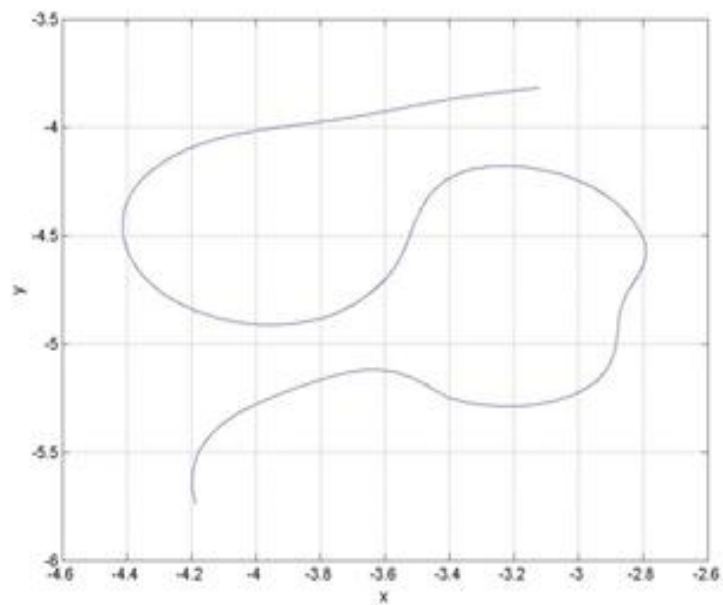
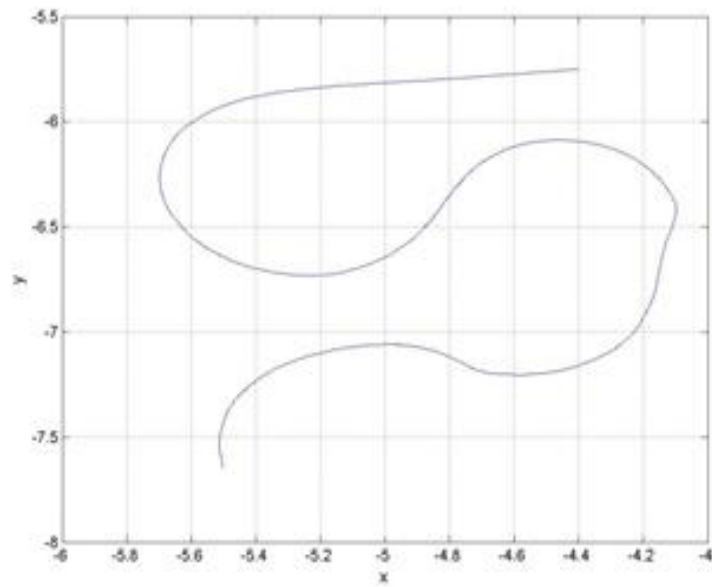


Fig. 6.6 Result of autonomous driving

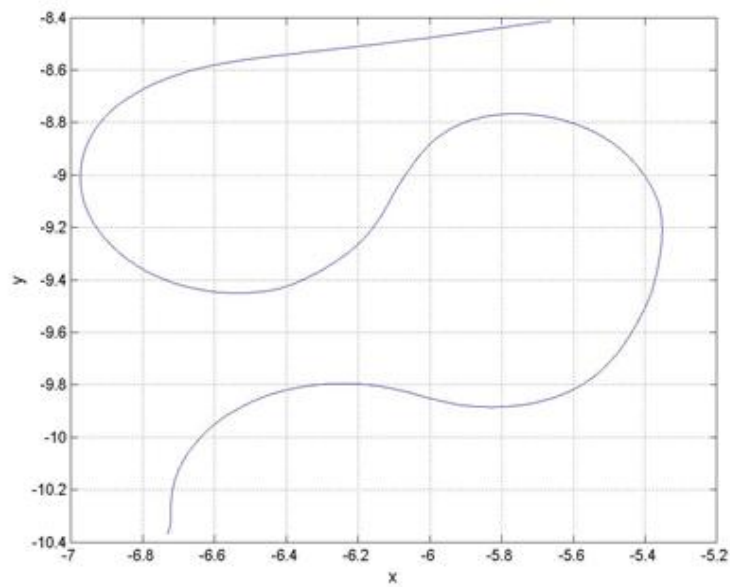
The following figure 6.7 shows the path according to the number of driving(=  $N$ ) by applying deep learning.



a)  $N = 50$



b)  $N = 100$



c)  $N = 200$

Fig. 6.7 Result of autonomous driving applied deep learning

## VII. Conclusion

In this paper, we have experimented on autonomous driving control of ROS-based mobile robot using deep learning. When autonomous driving is controlled using only image processing, the autonomy of driving is insufficient because the parameters for controlling the motors are fixed. To compensate for a disadvantage, we applied the deep learning and applied the appropriate parameters momentarily. As a result, it was confirmed that autonomous driving of the mobile robot was supplemented,

As image processing technology develops, autonomy of autonomous driving will increase, but image processing technology is limited in case of unexpected accident or situation. Deep learning enhances the ability of mobile robot to judge themselves through machine learning, so convergence with two technology has a great synergy about autonomous driving technology.

ROS is a key platform for robot control. In the future, deep learning technology will be fused with robot in various ways. One of them, ROS, was successfully used for deep learning experiments.

In the future, it is expected that we will need to define a message file for deep learning. And then, deep learning and ROS will be applied to many kind of fields as well as autonomous driving.

## Reference

- [1] Lee B-Y, “Trends and prospects of autonomous driving technology development at home and abroad”, Journal of The Korean Institute of Communication Sciences, Vol. 33, no. 4, pp. 12-15, 2016.
- [2] Cousins. S, “Is ROS Good for Robotics?”, IEEE robotics & automation magazine, Vol. 19, no. 2, pp. 13-14, 2014.
- [3] <http://turtlebot3.robotis.com>
- [4] Lee S-M, “Speed Control of Mobile Robot and the study on Auto-Parking”, Master dissertation, Dept of Electronics Eng., Dong-A Univ., Pusan, Korea, 2003.
- [5] Park J-H, “Object Tracking and Elimination using Level-of-Detail Canny Edge Maps”, Hongik Journal of Science and Technology, Vol. 9, 2005.
- [6] Zengqiang Ma, Xiaoyun Liu, Zheng Liu, Sha Zhong and Yusi Zhang, “An Improved Hough Transform Algorithm Based on Reduced Particle Swarm Optimization and its Applications in the Train Wheel Image Detection”, International Journal of Multimedia and Ubiquitous Engineering, Vol. 11, no. 11, 2016.
- [7] Wi S-M, “A study on Object recognition and Track Initialization using Hough Transform“ Master dissertation, Dept of Electronics Eng., Kwang-woon Univ., Seoul, Korea, 1993
- [8] Choi Y-H, “A study on a face recognition and a performance comparison by Threshold change of Canny edge operator“ Master dissertation, Dept of EIC , Kwang-woon Univ., Seoul, Korea, 2017
- [9] Joe Minichino and Joseph Howse, “Learning OpenCV 3 Computer Vision with Python - Second Edition”, Packt Publishing, 2015
- [10] Enrique Fernández, Luis Sánchez Crespo, Anil Mahtani and Aaron Martinez, “Learning ROS for Robotics Programming - second edition”,

Packt Publishing, 2015

- [11] Lentin Joseph, “Mastering ROS for Robotics Programming”, Packt Publishing, 2016
- [12] Kim Y-J, “Analysis of Image Big Data Using Deep Learning”, Ph.D. dissertation, Dept of Statistics ,Chung-Ang Univ., Seoul, Korea, 2017
- [13] Cheon K-B, “DC Motor Control with Deep Learning”, Master dissertation, Dept of Electronics Eng., Kyungpook Univ., Daegu, Korea, 2014

## **Abstract**

# Autonomous Driving Control of Mobile Robots Based on ROS Using Deep Learning

*by*

*GIL KI JONG*

*Dept. of Electronic Engineering*

*Graduate School, Dong-A University*

*Busan, Korea*

Recently, development of image processing technology and development of artificial intelligence and deep learning technology have made autonomous vehicles appear as next generation technologies that will lead the market in the future. In this paper, we implemented autonomous driving control which extracts image processing image data of line and tracks the line by using 'Turtle Bot 3' which is a mobile robot based on ROS. Image processing is based on 'OpenCV', detects lines through Canny algorithm and Houghline algorithm, then transmits image data through ROS message communication and controls mobile robot's driving based on transmitted data. Deep Learning is used to supplement the running of the mobile robot by learning the driving record of the mobile robot.