

<미래자동차 로봇 캡스톤 디자인>

쓰레기를 자동으로 받아주는 모바일 로봇

2017103749 이현기

지도교수 : 정지영교수님

경희대학교 소프트웨어융합학과

목차

1. 연구 배경 및 목표	3
2. 기대 효과	4
3. 연구 관련 이론/방법론	5
3.2.1 개발 환경	7
3.3.1 HSV.....	8
3.3.2 노이즈 제거.....	9
3.3.3 Least Square Method(LSM)를 사용한 Curve Fitting.....	9
3.3.4 Wheel Kinematics.....	10
3.3.5 모터 동역학.....	11
3.4.1 ROS 메시지 통신.....	12
3.4.2 회로도	13
4. 연구 과정	14
4.2.1 물체의 선정	14
4.2.2 카메라 값 보정.....	14
4.2.3 이미지 동기화.....	16
4.2.4 물체의 3 차원 정보 획득	16
5. 연구 결과	24
5.2.1 시간 측정 방법.....	24
5.2.2 결과	25
5.3.1 측정 방법	25
5.3.2 결과	25
5.5.1 RGB 이미지와 Depth 이미지의 Exposure Time 에 의한 잔상	27
5.5.2 무선 LAN 연결	29
5.5.3 모터 제어	29
5.5.4 배터리	29
6. 결론	29
7. 참고문헌	30

1. 연구 배경 및 목표

1.1 연구 배경

최근 로봇 및 자율주행 자동차와 같은 산업이 급격하게 발전하면서 외부의 정보를 감지하는 센서 기술의 수요 또한 증가하고 있다. 로봇이나 자율주행 자동차에 중요한 기술 중 하나는 장애물의 3차원 위치 $\{x,y,z\}$ 를 정확하고 빠르게 습득하는 것이다. 물체의 3차원 움직임을 효과적으로 처리 가능한 로봇이 있다면 동적이고 예측 가능한(Predictable) 특성을 가진 장애물 제거, 회피 등에 큰 도움이 될 수 있을 것이다. 따라서 물체의 3차원 경로 예측을 위한 Fig.1 와 같은 시스템을 고안하게 되었다. 3차원 정보를 효과적으로 얻을 수 있는 센서에는 Kinect, Intel Realsense, LiDAR 등이 있는데 우리는 이 중 Kinect 센서를 이용하여 연구를 진행할 것이다.

1.2 연구 목표

Fig.1 와 같이 센서와 로봇을 설치하고, 센서 앞으로 던진 물체를 로봇이 이동하여 자동으로 받아주는 것을 구현하는 것을 목표로 하였으며, 3개의 Omni Directional Wheel을 사용하여 모바일 로봇의 2차원 평면 상 직선 주행을 수행할 것이다. 또한 Kinect 를 이용하여 물체의 3차원 좌표를 실시간으로 획득(Real-Time Communication)하는 것을 목표로 하였다.

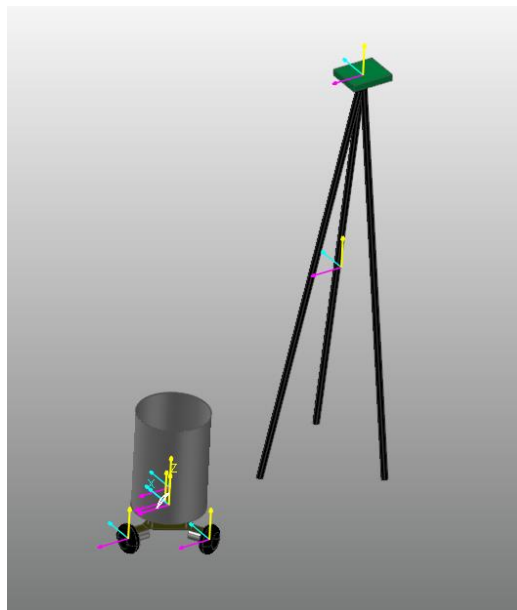


Fig.1 Kinect and Robot

2. 기대 효과

2.1 산업 현장에서의 적용

산업 현장에서 비전 센서를 이용하여 작업을 수행하는 로봇으로 인해 생산 현장에서의 로봇 실용화가 가능하게 되었다. 로봇에는 주로 2 차원 비전 센서가 사용되는데 작업 대상 물체의 영상을 획득하여 대상 물체가 놓인 위치와 회전각을 알아내어 물체를 집어낸다.[1] 또한 3 차원 비전 센서의 패턴화한 빛을 대상 물체에 비추어 획득한 정보를 이용하여 물체의 3 차원 위치와 자세를 추출하여 무작위 상태로 공급되는 부품을 식별하는 작업을 하기도 한다.[2][3] 움직이는 물체의 경로를 실시간으로 처리하는 기술은 산업용 로봇의 활용 가치를 한층 더 확대할 수 있을 것으로 기대된다.

2.2 자율주행 시 움직이는 물체 감지

자율주행 자동차에서의 장애물 인식에 대한 연구는 수없이 이루어지고 있다. 그중 대부분은 LiDAR 와 같은 레이저 스캐너, Monocular Camera, Stereo Camera 와 같은 비전 카메라를 이용하는데, 여기에 깊이 정보가 더해진다면 장애물을 더욱 효과적으로 인식하고 대응할 수 있다.[4][5] 또한 모바일 로봇의 Localization 문제를 해결하는 데에도 Kinect 를 이용하는 연구 또한 활발히 진행되고 있다.[6] 이와 같은 이점을 위해 깊이 카메라를 이용한 물체의 3 차원 좌표 인식 매커니즘은 앞으로 자율주행 자동차에서 큰 역할을 할 수 있을 것으로 기대된다.

2.3 엔터테인먼트 분야

움직이는 물체에 대한 비전 인식은 탁구 로봇이나 애완용 로봇과 같은 실생활 또는 엔터테인먼트 분야에도 적용될 수 있다. “Home Security Pet Robot”에서는 다양한 엔터테인먼트 기능의 구현을 위하여 비평탄면에서의 안정적인 보행 알고리즘, 비전시스템을 이용한 물체추적 알고리즘을 제시하였고[7], “Implementation of a Table Tennis Robot Using Monocular Camera-based 3D Object Recognition Method and Path Planning for 6-Axis Manipulator”에서는 두 대의 Monocular Camera 를 이용해 탁구공의 위치와 속도를 추적하는 기법을 제안하였다.[8] 이처럼 엔터테인먼트 분야에서도 Computer Vision 분야가 점차 널리 이용될 것으로 보인다.

3. 연구 관련 이론/방법론

3.1 사용한 하드웨어

사용한 하드웨어 장치 중 주요 장치들을 Table.1 에 나열하였다.

Azure Kinect	 Fig.2 Microsoft Azure Kinect ¹	Kinect는 깊이 센서(Depth Sensor), 비디오 카메라, 여러 개의 마이크 및 방향 센서(IMU)로 이루어진 소형 일체형 장치로 여러 모드, 옵션 및 소프트웨어 개발 키트 (Software Development Kit) 등을 사용할 수 있다. ² 물체의 3차원 정보 획득을 위해 사용하였다.
Raspberry Pi	 Fig.3 Raspberry Pi 3 B+ ³	1.4GHz 64-bit 쿼드코어 프로세서. Remote PC 및 아두이노와의 시리얼 통신, 아두이노 스케치 작성을 위해 사용하였다.
Arduino Nano	 Fig.4 Arduino Nano ⁴	ATmega328P 칩을 기반으로 한 소형 마이크로프로세서. DC 모터 제어를 위해 사용하였고 3개의 PWM OUT Pin, 6개의 DC OUT Pin 을 사용하였다.
DC Motor RA-20GM	 Fig.5 DC Motor ⁵	직류전동기는 기계적인 동력으로 직류 전력을 전기 기계로 변환하는 장치이다. 가장 일반적인 유형은 자기장이 생성한 힘에 의존한다. ⁶ 사용한 모터의 특성은 Fig.8 와 같다. 감속비는 1/42 를 사용하였다.
L298N	 Fig.6 DC Motor Driver ⁷	ST사의 고전압 고전류 모터 드라이버 칩인 L298N을 탑재한 모터 드라이버이다. 최대 46V까지의 전압에서 동작하며, 연속적으로 2A(피크시 3A)의 출력을 낼수 있는 25W 등급의 칩으로 DC모터와 스텝모터를 동작시킬 수 있다.
SZH-PWSD-016	 Fig.7 DC-DC Converter ⁸	5V 고정 출력 강하형 DC-DC 3A 컨버터. 라즈베리파이의 전원부에 사용하였다.

Table.1 Hardware

감속모터사양 GEARED MOTOR SPECIFICATION

감속기 길이 (mm) Gear Head L		16,5				20,0				21,8				25,3							
중량(g)		50				59,5				61				64							
감속비 Reduction ratio		1/10	1/13	1/18	1/29	1/42	1/52	1/67	1/80	1/99	1/120	1/150	1/179	1/225	1/275	1/350	1/430	1/549	1/689	1/794	1/866
04 Type 24v	정격토크(gf-cm) Rated torque	160	200	250	410	510	620	810	860	1100	1300	1500	1770	1800	1800	1800	1800	1800	1800	1800	1800
	정격 회전수(RPM) Rated speed	720	580	403	250	176	142	110	93	74	62	49,5	41	34	28,5	23	18,5	15	12	10,5	9,2
	무부하 회전수(RPM) No Load speed	900	692	500	310	214	173	134	112	90	75	60	50	40	32,7	25	20,9	16,3	13	11,3	10,3

장착된 모터사양 INSTALL MOTOR SPECIFICATION

04Type Motor (DC 24v)		
정격 토크 Rated torque	22	(gf-cm)
정격 회전수 Rated speed	7,400	(RPM)
정격 전류 Rated current	110	(mA)
무부하 회전수 No load speed	9,000	(RPM)
무부하 전류 No load current	30	(mA)
정격 출력 Rated output	1.7	(W)

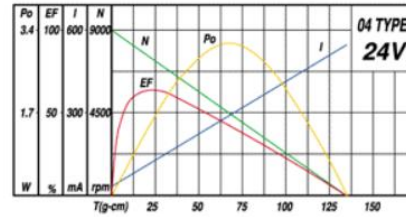


Fig.8 DC Motor Specification⁹

3.2 사용한 소프트웨어

사용한 소프트웨어 및 응용 프로그램들을 Table.2 에 나열하였다.





ROS (Robot Operating System)	 Fig.9 ROS ¹⁰	로봇 소프트웨어 작성을 위한 프레임 워크. Kinect, Remote PC, Raspberry Pi, 아두이노 간의 통신 및 코드 빌드를 담당한다. 개발 언어로는 C++을 사용하였다.
Azure Kinect ROS Driver	https://github.com/microsoft/Azure_Kinect_ROS_Driver	Azure Kinect Developer Kit로부터의 센서 데이터를 ROS 상으로 전송(publish)하는 노드. 오픈소스로 공개되어 있는 코드로 일부 수정해서 사용하였다.
OpenCV	 Fig.10 OpenCV ¹¹	실시간 컴퓨터 비전을 목적으로 한 프로그래밍 라이브러리이다. Kinect로부터 받은 영상 정보를 이용해 물체 인식 작업을 할 때 사용하는 프로그램이다.
RecurDyn	 Fig.11 RecurDyn ¹²	RecurDyn(리커다인)은 다분야 통합 CAE(Computer-Aided Engineering) 동역학 해석 소프트웨어로 그 주요 기능은 다물체 동역학(MBD: Multi-body Dynamics) 해석이다. ¹³ 로봇의 움직임에 대한 간단한 시뮬레이션 작업을 위해 사용하였다.
Arduino IDE	 Fig.12 Arduino IDE ¹⁴	아두이노 통합환경은 편집기, 컴파일러, 업로드 등이 합쳐진 환경이다. 이와 더불어 기타 개발에 필요한 각종 옵션 및 라이브러리 관리를 할 수 있다. 보통 USB를 통해 업로드를 하므로 아두이노 보드는 USB를 UART 통신으로 바꾸는 방법이 제공되고, MCU가 실행할 때는 UART 통신을 이용하여 필요한 통신을 할 수 있다. ¹⁵

Table.2 Software

3.2.1 개발 환경

Table.3 과 같은 개발 환경에서 작업하였다.


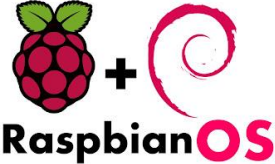

Linux Mint 19.2 Cinnamon	 <p>Fig.13 Linux Mint¹⁶</p>	가장 널리 사용되는 데스크탑 Linux 배포판 중 하나이다. Linux Mint의 목적은 강력하고 사용하기 쉬운 현대적이고 우아하며 편안한 운영 체제를 만드는 것이다. ¹⁷ 개인적인 취향으로 Ubuntu가 아닌 Mint를 사용하였다.
Raspbian Buster	 <p>Fig.14 Raspbian¹⁸</p>	즈베리 파이 재단이 개발한 라즈베리 파이 전용 운영 체제이다. 2015년 이후로 라즈베리 파이 재단에 의해 라즈베리 파이 단일 보드 컴퓨터 계열을 위한 주 운영 체제로서 공식 지원을 받고 있다. ¹⁹ 라즈베리파이에 설치하여 사용하였다.
Visual Studio Code	 <p>Fig.15 Visual Studio Code²⁰</p>	마이크로소프트가 마이크로소프트 윈도우, macOS, 리눅스용으로 개발한 소스 코드 편집기이다. ROS 노드 개발을 위해 사용하였다.

Table.3 Development Environment

3.3 주요 이론 및 적용

3.3.1 HSV

HSV 색 공간 또는 HSV 모델은 색을 표현하는 하나의 방법이자, 그 방법에 따라 색을 배치하는 방식이다. 색상(Hue), 채도(Saturation), 명도(Value)의 좌표를 써서 특정한 색을 지정한다.

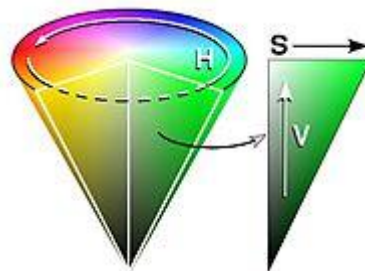


Fig.16 HSV Cone²¹

HSL, HSV, HSI 등의 모델은 기능 감지 또는 이미지 분할을 위해 컴퓨터 비전 및 이미지 분석에서 자주 사용된다. 예를 들어 얼굴, 텍스트, 번호판과 같은 로봇 비전에서의 객체 검출을 위해 사용되거나 이미지 검색, 의료 이미지 분석에서도 사용

되기도 한다.[9] OpenCV에서는 `cvtColor` 함수를 통해 RGB 이미지를 HSV 이미지로 변환하는 것을 지원한다.

3.3.2 노이즈 제거

물체의 정확한 인식을 위해 다음과 같은 노이즈 제거 연산을 사용하였다.

Gaussian Blur : 이미지 처리에서 가우시안 블러는 가우시안 함수로 이미지를 흐리게 한다. 일반적으로 이미지 노이즈를 줄이고 디테일을 줄이기 위해 그래픽 소프트웨어에서 널리 사용되는 효과이다. 이 블러링 기법의 시각적 효과는 반투명 화면을 통해 이미지를 보는 것과 유사하다.

Morphology : 격자 이론에 기초한 이론적 모델로서, 디지털 이미지 처리에 사용된다. 주로 이진영상에서 사용하는데, 두 가지 기본 형태학 연산자는 **Erode**와 **Dilate**이다. 그런 다음 **Opening**, **Closing**, **Gradient** 등과 같은 변형 형태도 사용된다.²²

Erode : 필터 내부의 가장 어두운 값으로 변환(and 연산)

Dilate : 필터 내부의 가장 밝은 값으로 변환(or 연산)

Erode -Dilate의 순서로 연산할 경우 작은 노이즈들을 제거할 수 있으며, **Dilate-Erode**의 순서로 연산할 경우 두개의 인접한 객체를 하나로 합칠 수 있다.



Fig.17 Erode-Dilate Operation Result²³

3.3.3 Least Square Method(LSM)를 사용한 Curve Fitting

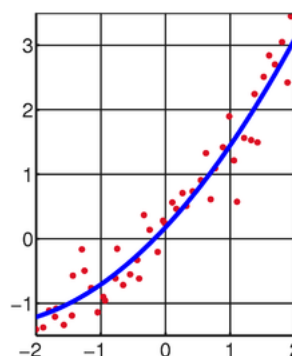


Fig.18 Least Square Method²⁴

최소 제곱의 방법(LSM)은 과도한 결정값을 갖는 (Overdetermined) 시스템의 근사 솔루션, 즉 미지수보다 많은 방정식이 있는 방정식 집합에 대한 표준 접근 방식의 회귀 분석이다. 자료점 (x_i, y_i) 에서 평가된 다항식의 값을 y_{c_i} 라 할 때, 임의의 종속 변수 y_i 와 다항식의 편차를 $y_i - y_{c_i}$ 로 정의한다. 모든 y_i ($i = 1, 2, \dots, N$)의 값에 대한 이 편차의 제곱의 합은 다음과 같다.

$$D = \sum_1^N (y_i - y_{c_i})^2 \quad (3.1)$$

D의 전미분을 0으로 놓으면 얻을 수 있는 $m+1$ 개의 식(m 은 예상되는 모델의 다항식 차수)을 연립하여 풀이하여 회귀 계수 a_1, a_2, \dots, a_m 을 구한다.[10] $m=2$ 일 때의 공식은 아래와 같다.

$$\begin{bmatrix} \sum_{i=1}^N 1 & \sum_{i=1}^N x_i & \sum_{i=1}^N x_i^2 \\ \sum_{i=1}^N x_i & \sum_{i=1}^N x_i^2 & \sum_{i=1}^N x_i^3 \\ \sum_{i=1}^N x_i^2 & \sum_{i=1}^N x_i^3 & \sum_{i=1}^N x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^N z_i \\ \sum_{i=1}^N x_i z_i \\ \sum_{i=1}^N x_i^2 z_i \end{bmatrix} \quad (3.2)$$

3.3.4 Wheel Kinematics

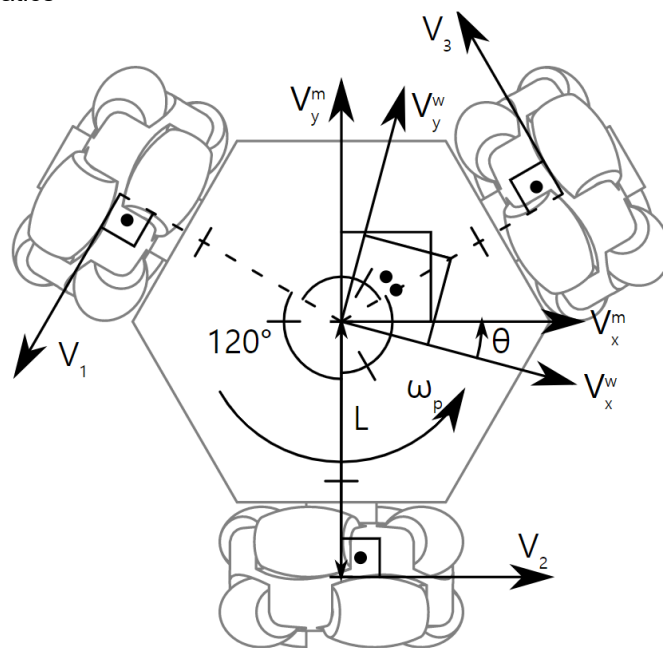


Fig.19 3Wd Omni Directional Wheel Kinematics²⁵

바퀴가 2 개 달린 로봇의 경우 다음과 같은 구속방정식이 존재한다.

$$H \cdot \dot{p} = (\sin \theta \quad -\cos \theta) \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \dot{x} \sin \theta - \dot{y} \cos \theta = 0 \quad (3.3)$$

평면 상에서의 로봇의 자유로운 움직임을 위해 3 개의 자유도(DOF, Degree Of Freedom)를 가진 omni directional wheel 을 사용하였다.

Fig.19 와 같은 omni directional wheel 에서의 기구학(Kinematics)은 다음과 같다.

$$V_x^m = \frac{2V_2 - V_1 - V_3}{3} \quad (3.4)$$

$$V_y^m = \frac{\sqrt{3}V_3 - \sqrt{3}V_1}{3} \quad (3.5)$$

$$\omega_p = \frac{V_1 + V_2 + V_3}{3L} \quad (3.6)$$

위 식을 Inverse Kinematics 로 풀면

$$V_1 = -\frac{V_x^m}{2} - \frac{\sqrt{3}V_y^m}{2} + L\omega_p \quad (3.7)$$

$$V_2 = V_x^m + L\omega_p \quad (3.8)$$

$$V_3 = -\frac{V_x^m}{2} + \frac{\sqrt{3}V_y^m}{2} + L\omega_p \quad (3.9)$$

여기서 우리는 각속도에 대한 작업은 필요 없으므로, $\omega_p = 0$ 으로 가정하고 문제를 해결할 것이다.

3.3.5 모터 동역학

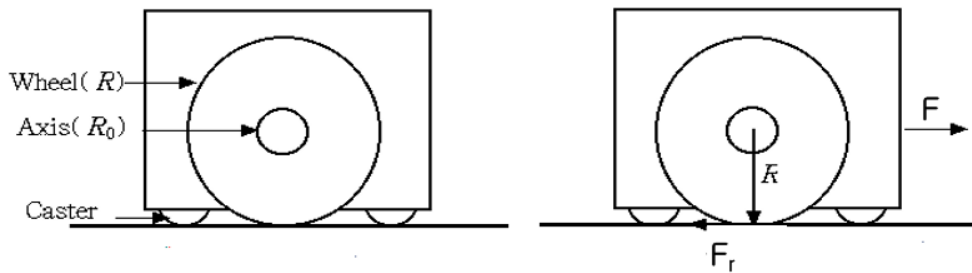


Fig.20 Motor Torque

$$\text{Static Frictional Force} \quad F_r = \mu mg \quad (3.10)$$

$$\text{Rolling Frictional Force} \quad F_G = \frac{F_r R_0}{R} \quad (3.11)$$

$$\text{Torque on one axis (max)} \quad \tau_G = \frac{1}{2} F_G R = \frac{1}{2} \mu mg R_0 \quad (3.12)$$

$$\text{Torque on one motor (max)} \quad \tau_M = \frac{1}{2} \left(\frac{\mu mg R_0}{N \eta_G} \right) \quad (3.13)$$

$$\text{Given} \quad \mu = 0.7, m = 4.2, g = 9.81, R_0 = 0.002, N = 42, \eta_G = 0.5 \quad (3.14)$$

식 3.13 에 주어진 값을 대입하면 $\tau_M = 14(\text{gf-cm})$, 따라서 모터의 정격 출력이 22(gf-cm)이므로 로봇을 구동시키기에 충분하다.

3.4 제안 알고리즘 및 적용

3.4.1 ROS 메시지 통신

노드 간의 메시지 통신 그래프를 Fig.21 에 나타내었다. 그래프의 노드에 대한 설명을 Table.4 에 서술하였다.

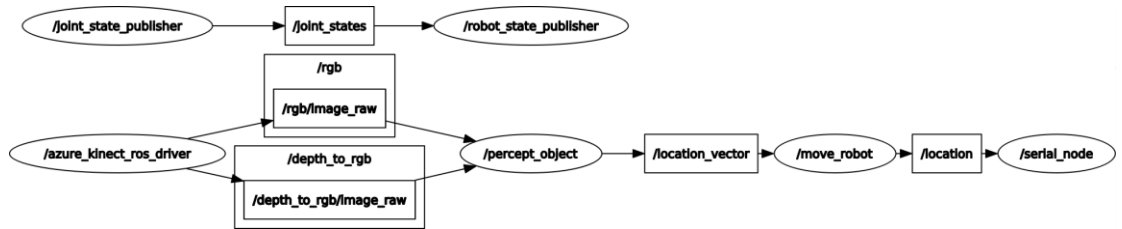


Fig.21 ROS Node Graph

Node	Device	Describe
/azure_kinect_ros_driver	Remote PC	Kinect 와 ROS 간의 인터페이스 노드. RGB Image 데이터와 Depth Image 데이터를 퍼블리시한다.
/percept_object	Remote PC	물체를 인식하는 노드. /azure_kinect_ros_driver로부터 이미지 데이터를 받아 물체를 인식한 후 3 차원 좌표가 담겨진 vector /location_vector를 퍼블리시한다.
/move_robot	Remote PC	/location_vector 메시지를 받아 최종 목표 좌표 /location을 퍼블리시한다.
/serial_node	Raspberry Pi	ROS 와 아두이노와의 연결 인터페이스 노드. 아두이노 스케치로 /location 정보를 전달한다.

Table.4 ROS Node

3.4.2 회로도

DC 모터 구동을 위해 사용한 회로의 회로도를 Fig.22 에 나타내었다.

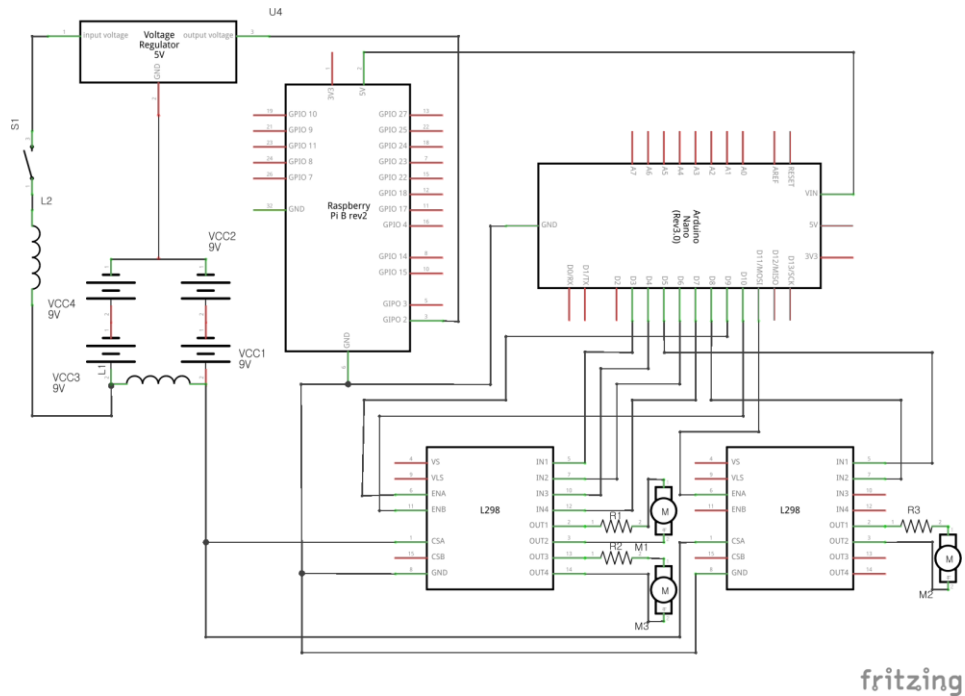


Fig.22 Circuit

4. 연구 과정

4.1 초기 설정 및 좌표계

Fig.23 와 같이 Kinect 는 삼각대를 이용하여 지면으로부터 1300(mm) 높이에 고정하였고 로봇의 초기 위치는 Kinect 전방으로 500(mm)만큼 떨어진 곳에 위치하였다. 좌표계의 경우 로봇의 중심축과 지면이 만나는 점을 원점으로 하였고(로봇의 입구 부근에서는 물체가 직선 운동에 가깝게 운동하므로 로봇의 높이는 무시하였다.) 축의 방향은 Fig.23 에 표기된 것과 같다. 로봇의 작업공간(Workspace)는 원점을 중심으로 $1(m^2)$ 의 정사각형 평면으로 정의하였다.

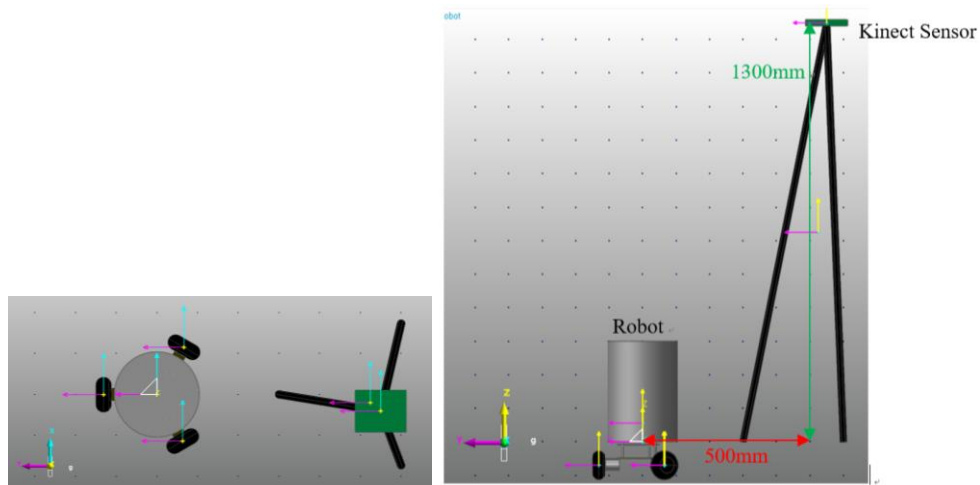


Fig.23 초기 위치 및 좌표계 설정

4.2 물체의 3 차원 좌표 인식

4.2.1 물체의 선정

물체는 다음과 같은 조건을 거쳐 선정하였다. 크기가 충분히 크고, 주변 환경의 색에 비해 채도가 높아야 하며 공기의 저항에 대한 영향은 작아야 한다. 주변에서 흔히 볼 수 있는 테니스공을 우선적으로 사용하여 실험을 진행하였으며, 이외의 물체들에 대한 결과는 5.4 절에 서술하였다. 또한 물체는 포물선(2 차함수) 운동을 한다고 가정한다.

4.2.2 카메라 값 보정

Kinect 의 Depth Sensor 의 설정값은 Table.5 과 같이 하였다.

Type	Value	Unit
Mode	WFOV 2x2 binned	
Resolution	512x512	pixel
FOI	120x120	degrees
FPS	30	
Operating range*	0.25 - 2.21	m
Exposure time	12.8	ms

Table.5 Depth Sensor Value

*15% to 95% reflectivity at 850nm, 2.2 $\mu\text{W}/\text{cm}^2/\text{nm}$, random error std. dev. ≤ 17 mm, typical systematic error < 11 mm + 0.1% of distance without multi-path interference.

Kinect 의 Color camera 의 설정값은 Table.6 와 같다.

Type	Value	Unit
Resolution	1280x720	pixel
Aspect Ratio	116:9	
FPS	30	
Nominal FOV(HxV)	90x59	degree
Format	BGRA	
Exposure time	8330	us
Brightness	128	0~255
Contrast	5	1~10
Saturation	32	0~63
Sharpness	2	0~4
GAIN	255	0~255
Backlight Compensation	ON	

Table.6 RGB Camera Value

4.2.3 이미지 동기화

정확한 좌표 데이터의 획득을 위해서는 수신하는 RGB 이미지와 Depth 이미지의 동기화(Synchronize)가 필요하다. ROS 에서 지원하는 `message_filters::Synchronizer` 함수에서 근사 시간 동기화 알고리즘을 사용할 수 있었다. 이에 대한 결과를 5.4.1 절에 서술하였다.

4.2.4 물체의 3 차원 정보 획득

Azure Kinect ROS Driver 패키지에서는 RGB 영상과 RGB 영상의 좌표계로 변환된 Depth 카메라의 영상을 30 프레임의 속도로 받아올 수 있다. 이를 이용하여 물체의 3 차원 좌표를 얻는 방법을 아래와 같이 구현하였다.

물체 인식은 다음의 단계를 통해 진행되었다.

1. Fig.24 와 같은 HSV 범위 추출 프로그램을 이용하여 물체의 HSV 색상 정보에 대한 범위를 얻는다.

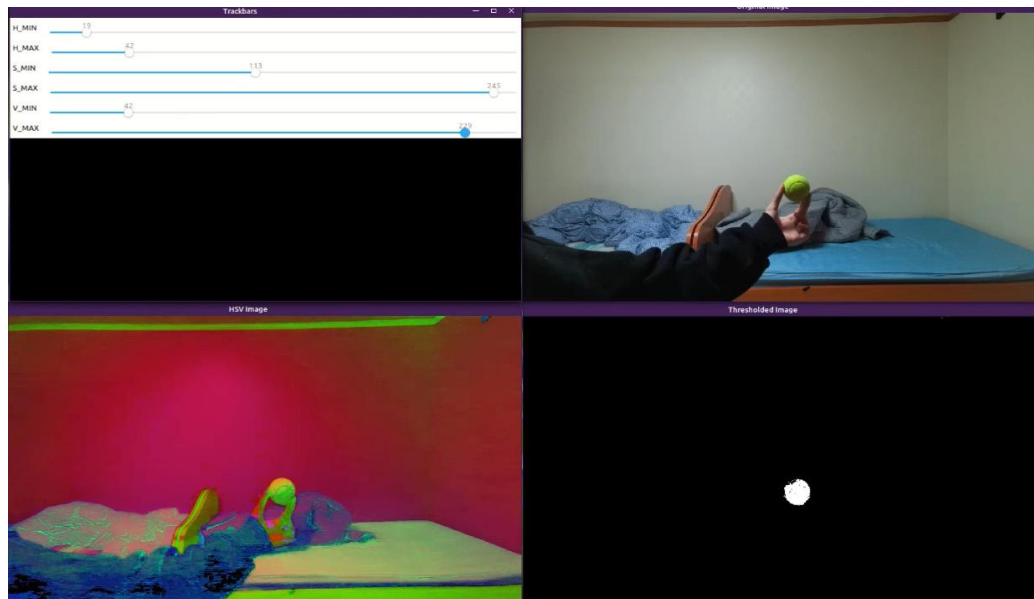


Fig.24 HSV 범위 추출 프로그램의 UI 및 이미지 처리 결과²⁶

2. 영상의 프레임 한 개를 RGB 이미지로 받아온다.

3. `cvtColor` 함수를 통해 RGB 이미지를 HSV 이미지로 변환한다.

4. 1.에서 알아낸 HSV 색상 범위를 이용하여, OpenCV 의 `inRange` 함수를 통해 HSV 이미지를 Fig.25 과 같은 흑백 이미지로 변환한다. 이때 물체가 있는 영역은 흰색,

나머지 영역은 검은색의 비트 이미지로 변환된다.

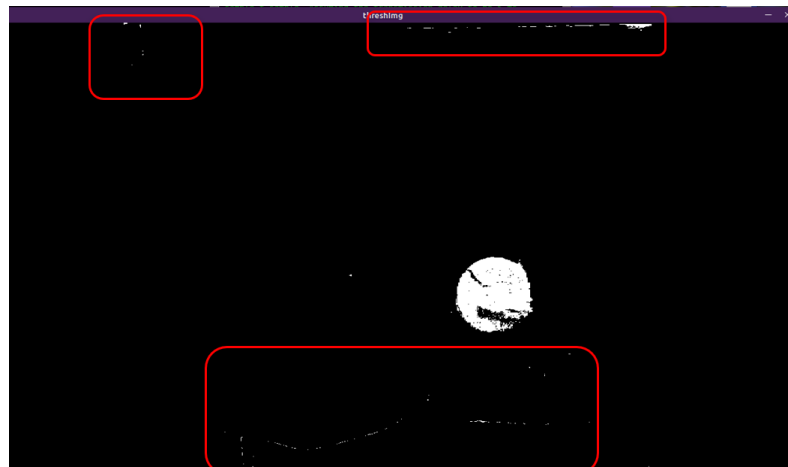


Fig.25 Binary Image

5. GaussianBlur 함수를 통해 물체 주변의 노이즈를 제거한다.

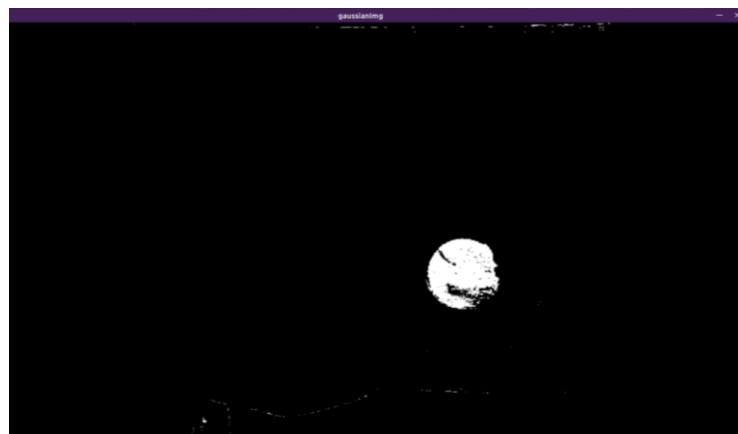


Fig.26 Gaussian Blur

6. erode, dilate 함수를 각각 5 번씩 적용시켜 모든 노이즈를 제거한다. Fig.25 에 적색으로 표시한 노이즈가 모두 제거되어있는 것을 볼 수 있다.

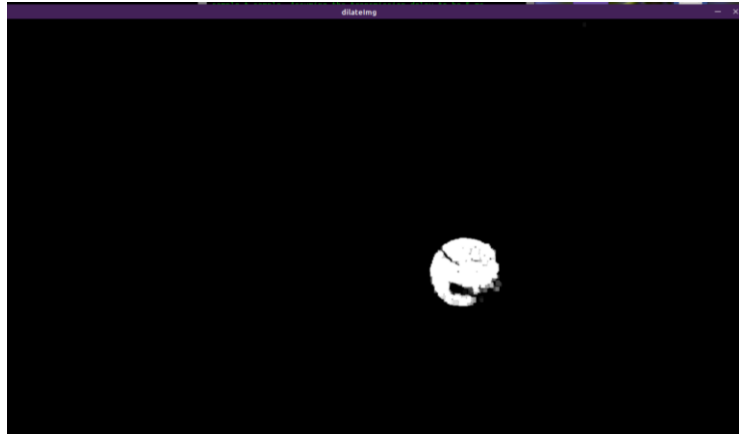


Fig.27 Opening Operation

7. HoughCircles 함수를 통해 물체를 인식한다. 또는 White Pixel 들의 무게중심을 구하는 함수 cvFindContours 를 이용해 물체를 인식한다.

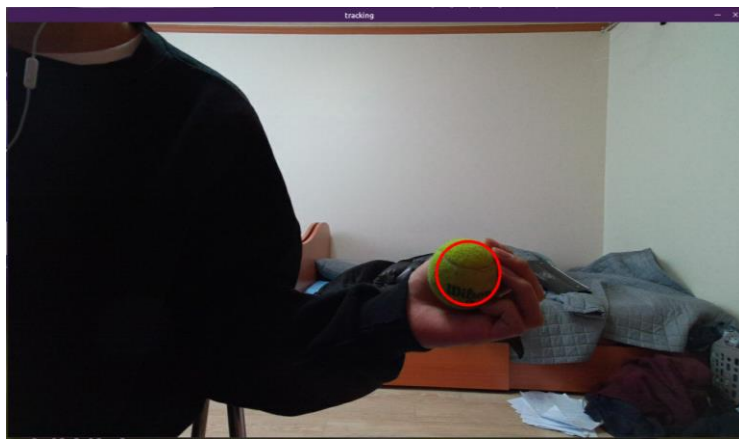


Fig.28 Detection Result

위와 같은 과정으로 물체를 인식하고 나면, 물체의 위치에 해당하는 픽셀 값을 반환받는다. Depth 카메라의 이미지가 RGB 이미지의 좌표에 맞추어져 있기 때문에 픽셀의 값을 그대로 이용하여 해당하는 픽셀의 깊이(Depth) 정보를 획득하였다. Kinect 는 Azure Kinect ROS Driver 패키지에서 작동을 선언하기 때문에 percept_object 노드에서는 Kinect 의 device handle 에 접근할 권한이 없다. 따라서 k4a_calibration_2d_to_3d 함수를 바로 사용할 수 없었고 Azure Kinect ROS Driver 패키지의 코드를 수정하여 percept_object 노드와 서비스 통신을 통해 변환된 3 차원 직교(Cartesian) 좌표 데이터를 얻을 수 있었다.

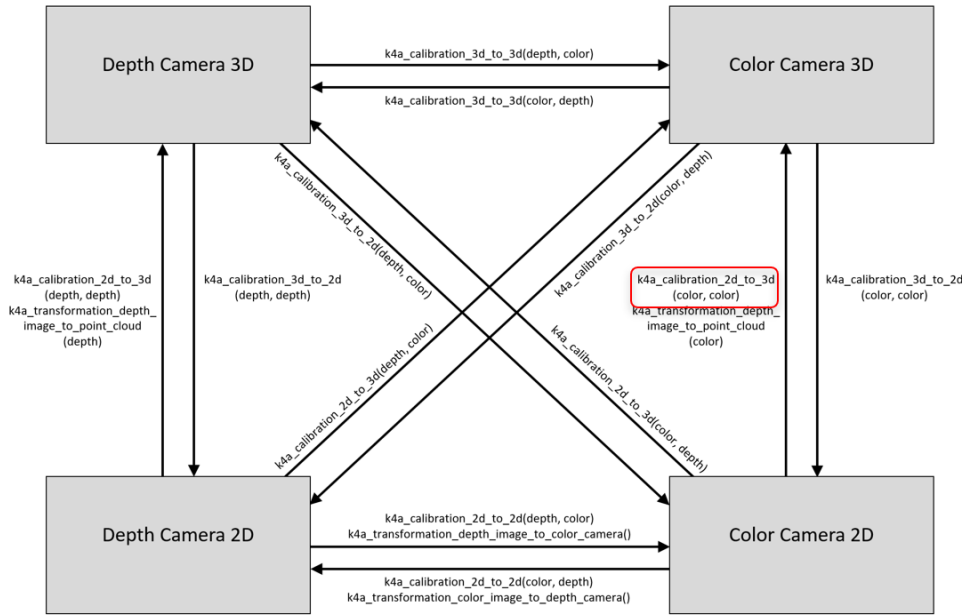


Fig.29 Transform Functions²⁷

Fig.29 에 적색으로 표시된 $k4a_calibration_2d_to_3d(color, color)$ 는 픽셀의 x,y 위치와 depth 정보, Reference Frame 을 매개변수로 하는 함수이다. 위의 함수를 이용하여 물체의 3 차원 직교 좌표 데이터를 받아올 수 있었다.

4.3 데이터 수집

N 개의 3 차원 데이터가 쌓일 때 까지 4.2.4 절의 작업을 반복하였다. 여기서 시간 복잡도에 $O(N)$ 이 곱해진다.(5.2.2 절 참조)

4.4 Curve Fitting 을 통한 목표지점 계산

N 개의 데이터가 모이면, 3.3.3 에서 설명한 LSM 을 통해 2 차함수의 계수를 얻는다. 간단한 역행렬 계산을 통해 2 차함수의 계수를 구할 수 있는데, 이 작업을 x-z 평면과 y-z 평면에 대해 2 번 진행한다. 각각의 평면에서 2 차함수 = 0 이라는 방정식의 해를 구하면, 물체가 떨어질 {x,y} 좌표를 구할 수 있다. Fitting 의 결과를 Fig.30 에 나타내었다. 적색 점들로 이루어진 곡선은 x-z 평면 Fitting 결과이고, 녹색 곡선은 y-z 평면, 분홍색 점들은 물체의 3 차원 데이터 샘플들이다.

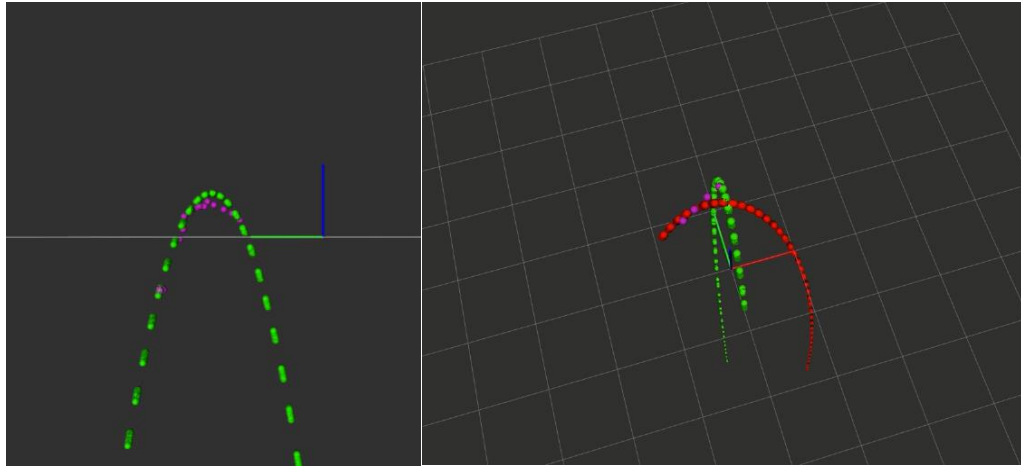


Fig.30 2 차원 및 3 차원에서의 경로 추정 결과

4.5 라즈베리파이와 통신

4.4 에서 구한 $\{x,y\}$ 좌표를 TCPROS 통신을 통해 라즈베리파이로 전송한다. 이에 대한 시간 지연이 발생하였고 5.5.2 절에 서술하였다.

4.6 아두이노와의 통신

아두이노 IDE 는 라즈베리파이에 설치되어있는 ROS 와의 연동을위해 `rosserial` 패키지를 사용한다. 시리얼 통신을 통해 아두이노 IDE 에 메시지를 전송한다. 아두이노 IDE 에서는 메시지가 들어오면, `callback` 함수를 통해 해당하는 $\{x,y\}$ 좌표로 이동하는 함수를 실행한다.

4.7 모터 제어 알고리즘

3.3.3 절에서 설명한 공식을 이용해 PWM 신호를 이용해 모터 드라이버에 신호를 주어 모터의 속도를 조절할 수 있었다. 다음과 같은 알고리즘을 이용해 로봇이 정확한 좌표에 도달할 수 있도록 구현하였다.

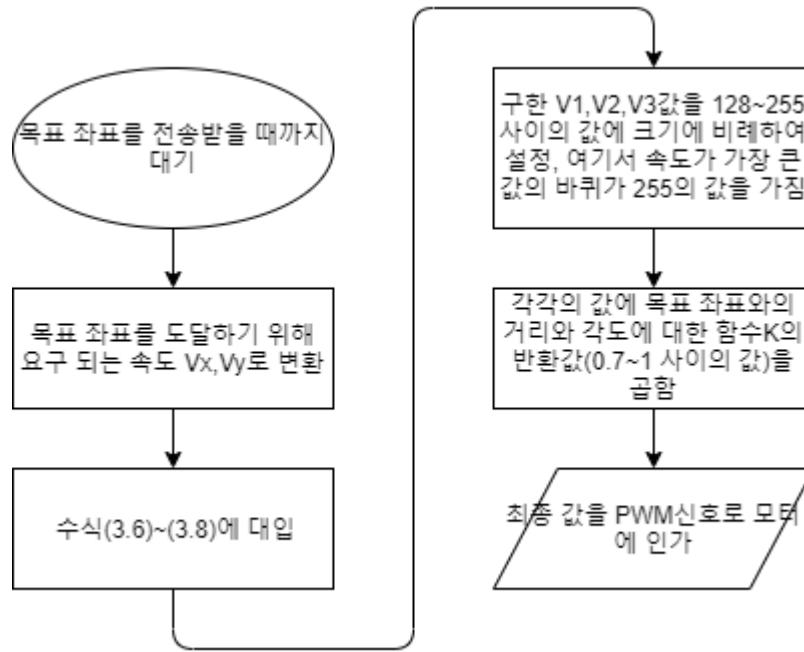


Fig.31 모터 제어 알고리즘

여기서 구한 V_1, V_2, V_3 값의 크기에 비례하여 128~255 사이의 PWM 값을 준 이유는 외부 토크에 의한 작용을 고려하였기 때문이다. 모터에 가하는 PWM 값이 모터의 속도가 아닌, (모터가 내는 토크 – 외부에서 가해지는 토크)에 근사적으로 비례하기 때문이다.

함수 $K(\theta)$ 는 다음과 같이 설계하였다.

$$p = \text{속력이 가장 빠른 바퀴의 번호 (1,2,3)} \quad (4.1)$$

$$\phi_p = p \text{ 번 바퀴의 이동 방향 벡터가 } x \text{ 축과 이루는 각도}$$

$$\phi = \text{Atan2}(V_y^m, V_x^m) \quad (4.2)$$

$$K(\theta) = \frac{\sqrt{3}}{2\cos(\phi - \phi_p)} \quad (4.3)$$

모터의 작동 시간 t 는 다음과 같다.

$$t = T_0 \sqrt{x^2 + y^2} \quad (T_0 = \text{const}) \quad (4.4)$$

DC 모터의 특성 상 t 초 후 전압 공급이 끊겨도 바로 정지하지 않기 때문에 50ms 동안 회전 방향의 반대 방향으로 전압을 인가하였다. EMF(Electromotive Force)의 발생은 무시하였다.

4.8 하드웨어 설계 및 가상 모델 구현

RecurDyn 프로그램을 이용해 시뮬레이션을 위한 가상 모델을 Fig.32 와 Fig.33 과 같이 설계하였다. 치수는 실제 로봇의 치수와 같으며 단위는 mm 이다.

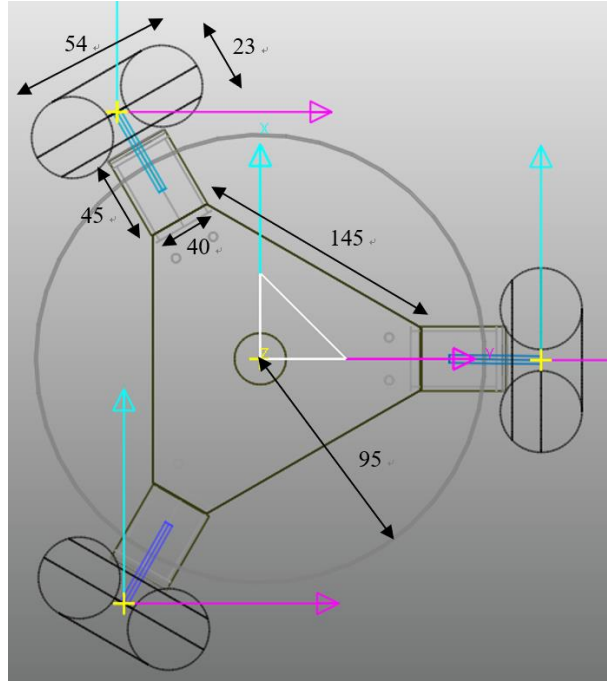


Fig.32 XY Plane

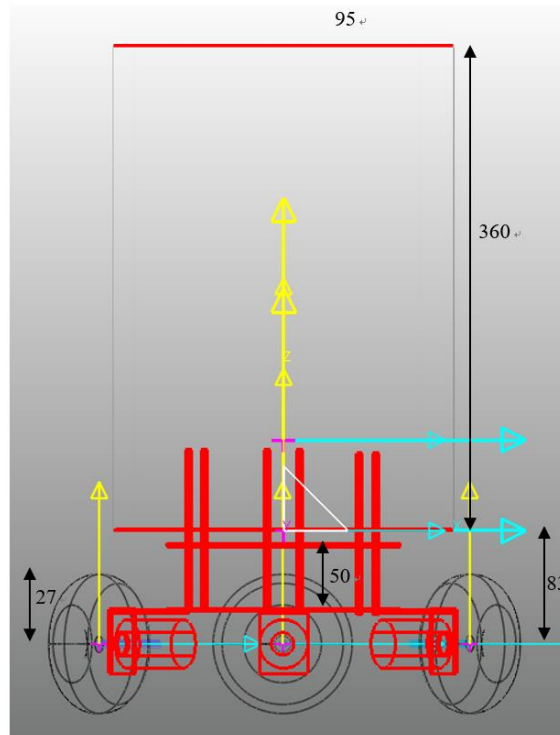


Fig.33 XZ Plane

4.9 시뮬레이션

동영상 참조

4.10 예외처리

Expected Error 의 종류는 다음과 같다.

Type	Contents
OUT_OF_DETECTION_AREA_ERROR	물체가 작업 공간으로부터 지나치게 먼 위치에서 감지되었을 때 발생한다. 좌표를 3 차원 데이터 벡터에 삽입하지 않는다.
SERVICE_FAILURE_ERROR	percept_object 노드와 Azure Kinect ROS Driver 와의 서비스 통신이 원활하지 않을 경우 발생한다.
OUT_OF_WORKSPACE_ERROR	구한 목표좌표가 작업 공간 밖에 있어서 로봇이 이동할 수 없는 경우 발생한다.
FITTING_ERROR	물체 좌표 인식에서의 Random Error 에 의해 Fitting 과정에서 이차항의 계수가 양의 값으로 계산되었을 때 발생한다.

Table.7 Error Types

4.11 전체 알고리즘 흐름도

전체 프로세스의 흐름을 플로우 차트로 Fig.34 에 나타내었다.

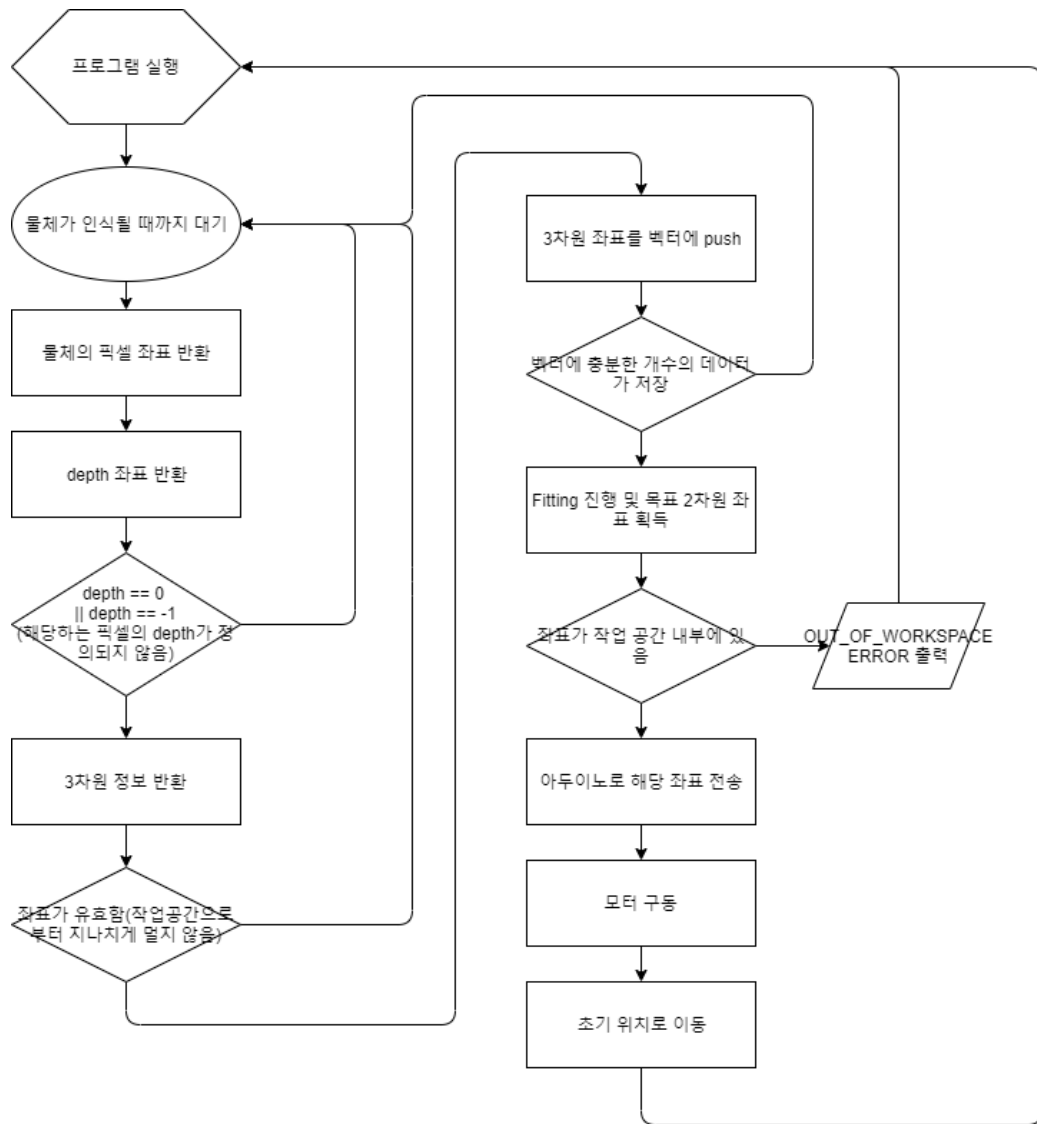


Fig.34 전체 알고리즘 흐름도

5. 연구 결과

5.1 하드웨어

5.2 3 차원 데이터의 개수에 대한 시간 소모

5.2.1 시간 측정 방법

물체의 3 차원 좌표를 저장하는 벡터에 두 번째 원소(첫 번째 원소는 Service Queue(Size = 1, 4.2.4 절 참조)에 있던 무의미한 원소로 버린다.)가 삽입되는 시각을 시간 변수 t_1 에 저장하였고, 목표 좌표가 출력되는 시각을 시간 변수 t_2 에 저장하였다. $t_2 - t_1$ 의 값을 데이터의 개수 $N(N = 5, 6, 7, 8, 9, 10)$ 에 대해 각각 20 번 씩 측정함으로써 목표 좌표를 계산하는데 걸리는 시간을 측정하였다.

OUT_OF_DETECTION_AREA_ERROR 가 발생한다면 시간 소모가 늘어나는 것은 당연하므로 Error 발생 빈도를 같이 표기하였다.

5.2.2 결과

Data Size (N)	OUT_OF_DETECTION_AREA_ERROR Count (20 tries)	Error Occur Rate(%) (E/20xN)	Time Consumption(ns)
5	4	4	171622968
6	3	2.5	202742010
7	6	4.28	235083220
8	6	3.75	241951152
9	9	5	304586728
10	7	3.5	320387409

Table.8 Time Consumption

5.3 3 차원 데이터의 개수에 대한 성공 확률

5.3.1 측정 방법

로봇을 작동시키고 행한 실험이다. 로봇이 물체를 받는 경우(물체가 다시 튕겨나온 경우 포함)를 성공이라고 정의하였다. 데이터의 개수 N 이 충분하지 않아 목표 좌표가 출력되지 않는 경우와 OUT_OF_WORKSPACE_ERROR 가 발생한 경우에는, 해당 시도는 표본에서 제외하였다.

5.3.2 결과

Data Size (N)	Success Count	Success Rate(%)
5	2/20	10
6	1/20	5
7	3/20	15
8	3/20	15
9	1/20	5
10	2/20	10

Table.9 Success Rate

5.4 테니스공 이외의 물체

탁구공을 사용하였을 때는 탁구공의 크기가 작아 5.6.1 절에 서술할 잔상에 의한 오차가 심하게 발생하였다. Table.10 에 성공 확률을 나타내었고 테니스공에 비해 성공 확률이 현저히 떨어지는 것을 확인할 수 있다. 흰색 아기 북극곰 인형을 사용하였을 때는 배경의 색과 물체의 색이 비슷하여 뒤쪽 조명이 어두운 특정한 상황에서만 성공 데이터가 나올 수 있었다.

Data Size (N)	Success Count	Success Rate(%)
5	0/20	0
6	2/20	10
7	3/20	15
8	1/20	5
9	1/20	5
10	1/20	5

Table.10 탁구공

Data Size (N)	Success Count	Success Rate(%)
5	0/20	0
6	2/20	10
7	2/20	10
8	3/20	15
9	1/20	5
10	3/20	15

Table.11 흰색 복극곰 인형

5.5 오차 및 실패의 원인 분석

5.5.1 RGB 이미지와 Depth 이미지의 Exposure Time 에 의한 잔상

4.2.2 절의 Table.5 과 Table.6 에서 볼 수 있듯이 RGB 카메라와 Depth 센서의 Exposure Time 은 각각 8.33,12.8(ms)이다. 물체가 빠르게 움직이는 구간 (물체를 던지기 시작한 구간) 에서는 위의 시간동안 물체가 움직이므로 RGB 이미지에 잔상이 발생한다. 또한 Depth 센서에서는 물체의 움직임에 의해 물체가 움직인 영역에 대해서 Depth 가 부정확하게 측정된다. 이는 물체의 Depth 와 배경의 Depth 를 모두 측정하면서 무효화 픽셀(Invalidated Pixel, <https://docs.microsoft.com/ko-kr/azure/Kinect-dk/depth-camera> 참조)이 발생하기 때문이다. Fig.36 에서 적색으로 표시한 부분이 무효화된 픽셀이고, 녹색으로 표시한 부분이 유효한 픽셀, 청색으로 표시한 거리는 RGB 카메라와 Depth 센서의 촬영 시각 및 Exposure Time 의 차이 때문에 나타나는 위치 에러이다. Fig.35 의 녹색 점에 위치한 픽셀의 Depth 값을 사용하기 때문에, Fitting 도중 에러가 나타날 확률이 높다. 또한 한 장의 Frame 을 건너뛰으로써 더 충분한 3차원 데이터를 모으는 데 시간이 더 오래걸리게 한다. Exposure Time 을 줄이면 이와 같은 에러가 줄어들지만 화면이 너무 어두워져 물체를 잘 인식하지 못하는 문제가 발생하였다.



Fig.35 RGB Image



Fig.36 Depth Image at Same Time

```

hyeongilee@hyeongilee-900X5N: ~/catkin_ws 66x19
5-90.9755 / -134
6-86.6356 / -127
7-83.2879 / -120
send data
i : 463 / j : 657 / d : 606
rece data
x : -171.563 / y : 106 / z : 414.253
Csize : 0
send data
i : 415 / j : 395 / d : 1869
rece data
x : -681.987 / y : 1369 / z : 611.289
[ERROR] [1575785698.571885819]: OUT_OF_DETECTION_AREA_ERROR
rgb / sec : / nsec : 350289340
dph / sec : / nsec : 350323340
  
```

Fig.37 RGB 카메라와 Depth 센서의 프레임 차이(약 33us)

5.5.2 무선 LAN 연결

만약 Remote PC 와 Raspberry Pi 가 연결되어 있는 무선 LAN(Local Area Network)에 많은 사람들이 접속해 있는 경우라면 통신 속도가 느려질 수 있다. 이로 인해 개인용 무선 네트워크를 이용했을 때는 나타나지 않던 로봇의 움직임에서 나타나는 지연 현상이 공용 무선 네트워크를 사용했을 때 나타났었고, 실시간성을 보장해야 하는 시스템에서 이와 같은 지연은 치명적일 수 있다.

5.5.3 모터 제어

엔코더가 달려있지 않은 DC 모터를 사용했을 때의 문제점은 단순히 모터의 특성(Fig.8)에 의존해 움직이는 속도를 예상해야 한다는 것이고 이는 DC 모터 각각의 특성이 다르기 때문에 시스템 오차가 발생할 수 있다. 만약 엔코더가 달려있는 모터를 사용했었다면 오차가 줄어들었을 것이다.

5.5.4 배터리

본 실험에서는 DC 모터 구동을 위해 9V 알카라인 배터리 6 개를 직, 병렬로 연결하여 27V 전압을 주었는데, 알카라인 배터리의 특성 상 순간전류에 약해 수명이 빨리 줄어들어 모터에 충분한 토크를 전달하지 못해 모터가 회전하기 시작한 시간이 늦어질 수 있다. LiPo 배터리와 같은 고용량 배터리를 이용하는 것이 좋은 해결 방안이 될 수 있다.

6. 결론

이번 캡스톤디자인 수업에서는 Azure Kinect 센서를 이용하여 3 차원상의 테니스공의 위치를 검출하여 이를 받는 쓰레기통 로봇을 구현하였다. Kinect 에서 검출된 물체의 3 차원 좌표를 수집하여 회귀 분석을 통해 물체가 떨어질 위치를 예측할 수 있었다. 하지만 5.5.1 절에서 서술하였듯이 카메라의 잔상으로 인한 오차 때문에 깊이 측정이 제대로 이루어지지 않을 때가 많았고, 고속으로 움직이는 물체 대한 잔상이 발생하였을 때 이를 보정할 수 있는 기법에 대한 연구가 필요해 보인다.

실험 결과 3 차원 데이터를 7~8 개 수집하여 사용할 때의 성공 확률이 15 % 로 가장 높았고, 시간 소모는 데이터의 개수에 비례하여 선형적으로 증가하는 것을 볼 수 있었다. 따라서 데이터의 개수가 9 개 이상일 때는 로봇이 이동하기 전에 이미 물체가 바닥에 떨어지는 현상이 발생하였다. 테니스공을 사용했을 때의 성공 비율이 다른 물체에 비해 대체로 높았던 것 또한 확인할 수 있었다.

7. 참고문헌

- [1] Kim Kye Kyung, Kang Sang Seung, Kim Joong Bae, Lee Jae Yeon, Do Hyun Min, Choi Taeyoung, Kyung Jin Ho, 2013, “Object Recognition Method for Industrial Intelligent Robot”, Journal of the Korean Society for Precision Engineering, Volume 30, Issue 9, pp.901~908
- [2] Jong-Kyu Oh, Sukhan Lee, 2012, “Stereo Vision Based Automation for a Bin-Picking Solution”, International Journal of Control, Automation and Systems, Volume 10, Issue 2, pp 362~373
- [3] K.Rahardja, A.Kosaka, 1996, “Vision-based binpicking: Recognition and localization of multiple complex objects using simple visual cues”, Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96
- [4] Jeff Michels, Ashutosh Saxena, Andrew Y. Ng, 2005, “High speed obstacle avoidance using monocular vision and reinforcement learning” ICML '05 Proceedings of the 22nd international conference on Machine learning’, pp.593~600
- [5] Andreas Wedel, Uwe Franke, Jens Klappstein, Thomas Bros, 2006, “Realtime Depth Estimation and Obstacle Detection from Monocular Video”, Pattern Recognition, 28th DAGM Symposium, Berlin, Germany, September 12-14, 2006, Proceedings, pp 475~484
- [6] N.Ganganath, H.Leung, 2012, “Mobile robot localization using odometry and kinect sensor”, 2012 IEEE International Conference on Emerging Signal Processing Applications
- [7] Heejin Lee, Changyong Yoon, Youngwan Cho, 2009, “A Home Security Pet Robot”, THE JOURNAL OF KOREAN INSTITUTE OF NEXT GENERATION COMPUTING, 5(3), pp.4~12.
- [8] Jae-Seong Yoon, Chang-Hee Kwak, Young-Hyeon Kim, Won-Ho Kim, Seung Ho Ok, 2018, “Implementation of a Table Tennis Robot Using Monocular Camera-based 3D Object Recognition Method and Path Planning for 6-Axis Manipulator”, Journal ofVolume 16, No 8, pp 35~44
- [9] Performance Comparison between RGB and HSV Color Segmentations for Road Signs Detection
- [10] Richard S. Figliola, Donald E. Beasley, 2014, “Theory and Design for Mechanical Measurements”, 6th Edition, WILEY, Hoboken, New Jersey, pp.148~149

이미지 및 소스코드 출처

-
- ¹ <https://azure.microsoft.com/ko-kr/services/kinect-dk/>
 - ² <https://azure.microsoft.com/ko-kr/services/kinect-dk/>
 - ³ <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>
 - ⁴ <https://store.arduino.cc/usa/arduino-nano>
 - ⁵ http://www.dnjmall.com/shop/goods/goods_view.php?&goodsno=122&category=009
 - ⁶ <https://ko.wikipedia.org/wiki/%EC%A7%81%EB%A5%98%EC%A0%84%EB%8F%99%EA%B8%B0>
 - ⁷ <http://www.devicemart.co.kr/goods/view?no=1327613>
 - ⁸ <http://www.devicemart.co.kr/goods/view?no=1321161>
 - ⁹ http://www.dnjmall.com/shop/goods/goods_view.php?&goodsno=122&category=009
 - ¹⁰ <https://www.ros.org/>
 - ¹¹ <https://opencv.org/>
 - ¹² https://twitter.com/RecurDyn_Tech
 - ¹³ <https://support.functionbay.com/ko/page/single/1/recurdyn-%EC%A0%9C%ED%92%88-%EA%B0%9C%EC%9A%94>
 - ¹⁴ <https://www.arduino.cc/>
 - ¹⁵ https://ko.wikipedia.org/wiki/%EC%95%84%EB%91%90%EC%9D%B4%EB%85%B8_IDE
 - ¹⁶ <https://linuxmint.com/>

-
- ¹⁷ <https://linuxmint.com/>
¹⁸ <https://www.raspberrypi.org/>
¹⁹ <https://www.raspberrypi.org/>
²⁰ <https://code.visualstudio.com/>
²¹ https://ko.wikipedia.org/wiki/HSV_%EC%83%89_%EA%B3%B5%EA%B0%84
²² <https://dyndy.tistory.com/252>
²³ https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html
²⁴ <https://ko.wikipedia.org/wiki/%EC%B5%9C%EC%86%8C%EC%A0%9C%EA%B3%B1%EB%B2%95>
²⁵ <https://github.com/GuiRitter/OpenBase>
²⁶ 소스코드 출처 : <https://gist.github.com/pknowledge/aa1469b7ba8cd652adb652d4359ef4f0>
²⁷ <https://docs.microsoft.com/ko-kr/azure/Kinect-dk/use-calibration-functions>