영상후처리를 통한 ArucoMarker 위치 및 자세 추정 알고리즘 개선

정대현† · 박유경*
지도교수 이대우**

초 록(Abstract)

본 프로젝트의 설계목표는 OpenCV(Open Source Computer Vision)와 ArucoMarker를 이용한 위치 및 자세 추정과 모델을 이용한 검증이다. 4차 산업 혁명과 함께 컴퓨터 비전의 기술이 빠르게 발전을 하고 있다. 많은 센서 대신 간단한 카메라만으로 자세 추정이라는 목표를 달성하고자 한다. 영상인식에 문제는 카메라의 기구적인 왜곡과 영상의 노이즈다. 문제를 해결하기 위해 카메라 캘리브레이션과 임계 값 처리 등 영상 후처리 기법을 통해 노이즈를 제거하며 자세 추정의 안정성을 높이고자 한다. RGB색상 공간의 원본을 사용했을 때보다 값의 편차가 20% 내외로 감소하였으며 영상의 후처리에 따라 검출의 안정성을 개선할 수 있다.

핵심주제어: OpenCV, ArucoMarker, Pose Estimation, 카메라 캘리브레이션, 영상 후처리

1. 과제 개요

1.1 과제배경

오늘날 4차 산업에서 각광받고 있는 기술 중 컴퓨터 비전이 있다. 사람만의 고유한 기능으로 여겨지던 물체를 구분하거나, 위치와 자세를 인식하는 시각 정보 처리는 컴퓨터 성능의 비약적인 향상과 이미지 처리에 적합한 합성곱 신경망(CNN, Convolutional Neural Network)모델의 발전으로 특정 부분에서는 인간보다 더 뛰어난 기능을 수행하고 있다. 차세대 전기 자동차와 자율주행기능으로 각광을 받고 있는 미국 테슬라의 경우 LiDAR없이 오로지 카메라만을 가지고 자율주행을 구현하겠다는 목표를 세웠으며, 포스코는 OpenCV(Open Source Computer Vision)와 딥러닝 기술을 통해 사물의 표면 결함 검출장치를 개발하였다.[1]

컴퓨터 비전을 이용하여 수행할 수 있는 기능은 굉장히 다양하다. 그 중에서도 위치 및 자세 추정에 대한 것을 다뤄 보기로 했다. 컴퓨터 비전에서 물체의 자세를 추정을 하는 것은 주로 로봇 위치 탐색과 AR 증강현실에서 중요한 역할을 한다. 일반적으로 많이 사용하는 방법은 사용자의 몸에 센서를 부착하거나 스틱 등의 도구를 들고 있어야 한다. 하지만 컴퓨터 비전의 발전으로 웹캠과 같은 간단한 장치를 이용하여 자세 추정을 할 수 있다. 물체의 자세추정은 넓은 산업분야에서 다양하고 유용하게 사용할 수 있다. 예를 들어 무인 비행기나 드론이 자동 착륙을 시도할 때 자이로스코프와 같은 내적인 장치나 또는 외부에 존재하는 식별마커를 통해 현재 상태의 자세 추정이 가능하다면 착륙이 용이 해진다. 그리고 인공위성끼리 도킹을 시도할 때 서로의 자세를 추정할 수 있다면 손쉽게 도킹을 할 수 있다. 또 다른 예시로는 군집 드론에서 앞서간 드론의 자세를 추정할 수 있다면 따라가는 드론 역시 자세 제어가 용이 해진다. 사용자가 드론 조작을 할 때 조이스틱 없이 직관적인 컨트롤러를 개발해볼 가능성도 있다.

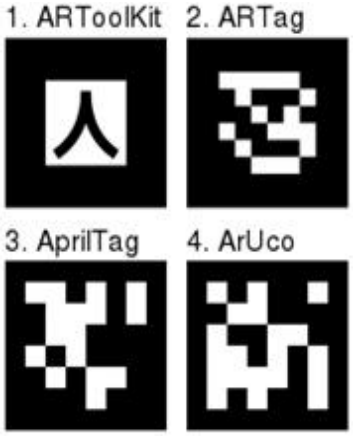
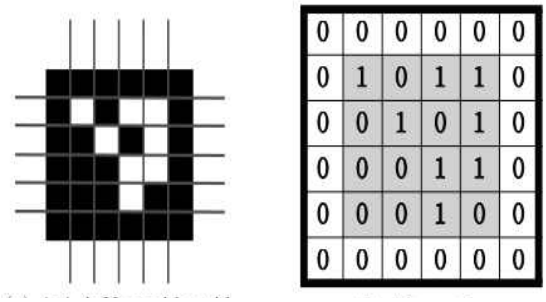


Fig. 1. 다양한 식별마커 종류

본 프로젝트에서는 식별마커(Fiducial marker)를 사용하여 카메라를 기준으로 마커의 상대 자세 추정을 할 것이다. 식별마커는 영상 시스템의 시야에 배치하는 개체로 기준점이 되거나, 자세 추정을 목표하는 대상에 식별마커를 부착하여 자세 추정을 한다.[2] 대표적으로 ARToolKit, ARTag, AprilTag, ArucoMarker등이 있다. 여기서는 컴퓨터 비전 라이브러리인 OpenCV에 내장된 ArucoMarker를[3] 사용하기로 했다.



0	0	0	0	0	0
0	1	0	1	1	0
0	0	1	0	1	0
0	0	0	1	1	0
0	0	0	1	0	0
0	0	0	0	0	0

Fig. 2. ArucoMarker의 구성

Fig. 2는 ArucoMarker의 구성을 표현하고 있다. ArucoMarker는 NxN 크기의 비트와 비트를 둘러싼 테두리로 구성된 정사각형 마커이며 크기에 따라 내부 매트릭스의 크기를 결정한다. 예를 들어 4x4 크기의 마커는 16비트로 구성된다. ArucoMarker의 검은색 테두리는 이미지에서 빠른 감지를 용이하게 하고 이진 코드화를 통해 식별과 오류감지 및 수정이 가능하다.[4] ArucoMarker의 장점은 대표적으로 3가지가 있다. 첫째, ArucoMarker 내부거리와 비트 천이 횟수를 고려한 마커 사전과 간단한 비교만으로 식별자(id)를 구분할 수 있다. 둘째, QR 코드에 있는 일부가 오염이나 손상을 입더라도 이를 복원 할 수 있는 에러 검출 코드가 존재한다. 셋째, 마커 식별에 회전과 방향이 고려되기 때문에 회전되어 있는 마커도 간단히 구분할 수 있다.

1.2 기존 문제점 및 새로운 요구사항

컴퓨터비전에서 가장 큰 문제점은 카메라의 기구적인 왜곡과 외부환경에서 오는 광원 등의 노이즈이다. 카메라 내부의 구조적인 부분에 따라 물체를 받는 과정에서 발생하는 왜곡이 존재하며 외부 광원으로부터 오는 여러 가지 노이즈는 물체의 인식률을 저하시키는 문제가 있다. 어떻게 하면 왜곡과 노이즈를 최소화할 수 있는지가 본 프로젝트의 핵심 사항이다.

단안카메라를 사용해서도 많은 일을 수행할 수 있다. 차선, 보행자, 다양한 신호등, 자동차 경로 내에 있는 다른 차량들을 식별할 수 있다. 단안 카메라 시스템이 신뢰하기 어려워지는 부분은 단안 카메라 센서에서 수신한 평면 2D 프레임에서 현실의 3D 프레임을 계산하는 경우다.

많은 연산이 필요 없는 것은 단안 카메라 시스템의 장점이긴 하지만, 몇 가지 문제가 있다. 단안카메라는 물체가 정확하게 식별될 때 까지는 식별 물체까지의 거리를 가늠할 수 없다. 또한 단순히 2D 프레임에서 물체가 식별 되었다고 하더라도 단순히 그 크기만으로는 거리를 알 수가 없다. 이를 해결하기 위해서 주로 양안카메라를 사용한다. 양안카메라의 원리는 사람의 눈과 같다. 사람은 양쪽 눈을 통해서 어떤 물체를 볼 때 두 눈 사이에 맺히는 상의 위치 차이를 가지고 대상자와의 거리를 가늠한다.

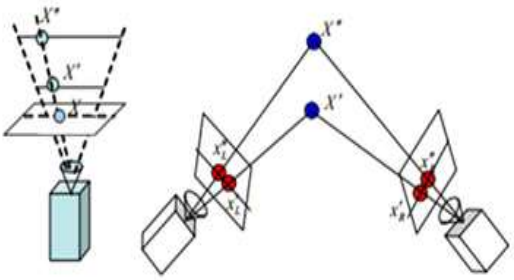


Fig. 3. 단안 카메라와 양안 카메라의 차이

Fig. 3은 단안카메라와 양안카메라가 물체를 받아들이는 차이를 나타낸 그림이다. Fig. 3.의 왼쪽 단안 카메라에서 살펴보듯이 X, X', X''의 거리 정보는 3가지로 다르지만 카메라에서는 한 점에 수렴하여 같은 곳에서 상이 맺히는 것을 알 수 있다. 반면 양안 카메라의 경우 거리 정보가 달라지면 양쪽 카메라와의 거리차이가 생기는 것을 볼 수 있다. 그리고 이 거리차이를 통해서 실제 물체와의 거리를 추정할 수 있다.

컴퓨터 비전에서 양안카메라를 이용하면 렌즈간 거리 차이를 이용해 물체를 3차원으로 인지함으로 형상 정보와 거리정보까지 얻을 수 있다는 장점이 있다. 하지만 양안카메라를 사용할 경우, 두 카메라에 맺힌 상을 비교하는 과정에서 많은 연산 량이 요구되고 카메라 사이에 고정적인 거리를 요구하기 때문에 단안카메라에 비해 많은 부피를 차지한다.

본 프로젝트에서는 최대한 단순하며 이식성이 높은 자세추정 장치를 개발하고자 한다. 따라서 많은 감지기나 고성능 컴퓨터가 아닌 카메라와 싱글 보드 컴퓨터인 라즈베리파이로 구성하였다. 그에 따라 요구되는 연산량은 최대한 적어야 하며, 라즈베리파이의 제한된 성능 안에서 작동함과 동시에 정확도를 확보해야 한다.

1.3 설계목표(개념) 및 해결하고자 하는 문제점

설계목표는 OpenCV와 ArucoMarker를 이용한 위치 및 자세 추정과 모델을 이용한 검증이다. 목표 달성을 위해 컴퓨터 비전에서 보편적으로 사용되는 OpenCV를 이용하여 Fig. 2와 같은 식별마커를 인식하여 해당 마커의 위치 및 자세를 획득한다. 이 과정에서 식별마커인 ArucoMarker에 대한 이해가 필요하다. 본 프로젝트의 설계 프로그램은 임베디드 시스템인 라즈베리파이에서 구동한다.

많은 연산이 요구되고 단안카메라에 비해 비교적 많은 부피를 차지하는 양안 카메라 대신 보다 저렴하고 연산이 적은 단안카메라를 이용하여 물체의 정확한 위치와 자세를 추정하고자 한다. 이로 인해 예상되는 문제점은 3D 프레임에서 2D 프레임으로 전환하면서 발생하는 카메라의 기구적인 왜곡과 거리정보 소실, 외부 환경이나 광원들로 발생하는 노이즈다. 이를 제거하거나 최대한 억제할 때 보다 정확한 자세와 위치 추정이 가능하게 된다. 마지막으로 라즈베리파이의 제한된 성능을 가지고 본 프로젝트의 모델을 구축해야 한다.

1.4 현실적 제한조건

라즈베리파이에 연결되는 장치는 제한적이다. GPIO(General Purpose Input/Output) 확장 보드를 이용하여 센서를 추가할 수 있지만 이 역시 제한적인 범위에서 이루어진다. 라즈베리파이의 전력량이 제한 조건으로 들어가기 때문이다. 부착된 장치들로부터 과다한 전력을 요구받을 경우 보드는 과전압 손상을 입을 수 있다. 이는 시스템 동작 중 불규칙한 무응답, 예상하지 못한 동작에 그치지 않고 치명적인 경우 보드의 영구적인 손상을 입을 수 있다. 따라서 가능한 최소한의 장치를 가지고 시스템을 구축해야 한다.

자세 추정 모델을 만드는데 있어 고가의 스텝모터, 리니어 모터를 이용해서 구현하거나 또는 상용으로 구할 수 있는 반제품, 툴킷(ToolKit)을 이용하면 간단하게 모델을 제작할 수 있다. 하지만 x,y,z의 각 축을 이루기 위해서 각 축당 수십만원을 넘는 비용이 들어가며 개발하고자하는 자세 추정 검증 모델을 제작하는 것에는 적합하지 않다. 최대한 간단하고 합리적인 가격인 모터를 이용하여 전체 모델을 만들어야 한다는 조건이 들어간다. 다양한 재료를 사용하지 못하는 만큼 모터가 받게 되는 하중을 관리하지 못하거나, 물리적인 전선 꼬임문제, 전체 장치를 감당할 토크 힘을 가진 모터를 사용하지 못할 수 있어 모델을 만드는데 여러 가지 조건을 고려해야 한다.

라즈베리파이 처리성능도 제한 조건이 된다. 라즈베리파이는 시간이 지날수록 개선이 된다고 하지만 이는 전 세대 기준이며 대형 프로젝트를 위해 사용하는 워크스테이션이나 일반적인 가정용 PC보다 부족한 성능을 가지고 있다. 20~30개 정도의 라즈베리파이가 동시에 연산을 진행해야만 가정용 데스크탑의 내장 그래픽 카드의 연산능력과 비슷해지는 수준이다. 라즈베리파이에 탑재되는 CPU는 스마트폰에 주로 사용하는 저전력 ARM 기반으로 PC 운영체제 리눅스와 대표적인 스마트폰 운영체제 안드로이드에서 작동이 가능하기에 범용 적으로 사용이 가능하고 이식이 용이하다. 하지만 소형 장치에서 작동해야 하기에 저전력으로 설계되어 성능이 제한된다.

이러한 라즈베리파이의 제한적인 성능 때문에 본 프로젝트에서는 이미지 정보 처리가 중요하다. 이미지는 각 픽셀당 데이터 값을 저장해야 하기에 많은 양의 연산을 요구한다. SD화질은 한 장당 345,600 화소를 담고 있는 최근 대중적으로 사용하는 FHD(1920x1080)의 경우 2,073,600화소로 SD에 비해 대략 6배의 정보를 담고 있으며 이를 가지고 벡터연산을 수행할 경우 이미지는 보통 3차원의 형태로 이루어지기 때문에 작동시간은 $O(n^3)$이 들어가게 된다. 따라서 라즈베리파이를 통한 이미지 정보처리를 위해서는 라즈베리파이가 처리할 수 있는 정도의 정보량을 고려하며 설계를 수행해야 한다.

2. 설계 과정

설계 과정은 소프트웨어와 하드웨어 파트로 이루어진다. 처음부터 라즈베리파이위에서 모든 코드를 작성하기보다 일반적인 성능의 데스크탑으로 전체 코드를 작성하고 구현 여부를 확인한 다음 라즈베리파이에 이식을 하고 하드웨어를 구축하는 순서로 넘어갈 계획이다.

2.1 OpenCV 소프트웨어 환경 구축

해당 결과물을 도출하기 위해 컴퓨터 프로그래밍 언어인 파이썬과 대표적인 컴퓨터 비전 라이브러리인 OpenCV를 이용하여 작동 가능한 코드를 구성한다. 필요한 준비물은 웹캠을 탑재한 노트북과 사전에 정의된 크기의 ArucoMarker를 이용하여 카메라와 직선거리와 각도를 추정할 수 있는 상대좌표를 구한다. 이를 가지고 카메라 또는 ArucoMarker를 기준으로 하는 자세 추정이 가능하게 한다. 코드 작성을 위해 사용 장비와 라이브러리는 다음과 같다.

- OpenCV 4.5.2 ver.

- python 3.8 ver.
- Surface Laptop4 720pHDf2.0 camera

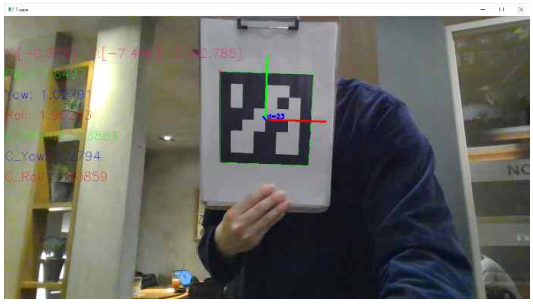


Fig. 4. 식별마커의 위치 및 자세 추정

Fig. 4는 노트북 카메라에서 ArucoMarker를 추적하고 있다. 실시간으로 마커의 평행이동 위치와 각 축의 회전을 구하고 있으며 왼쪽 상단에는 카메라와 ArucoMarker 2가지 기준좌표를 선택하여 서로를 기준으로 두는 상대좌표를 추적하고 있다.

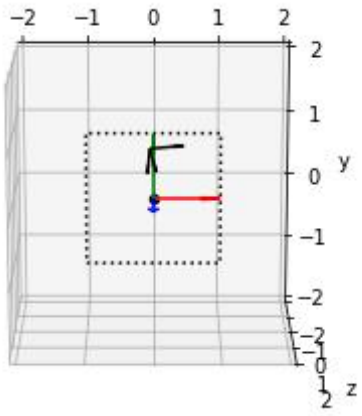
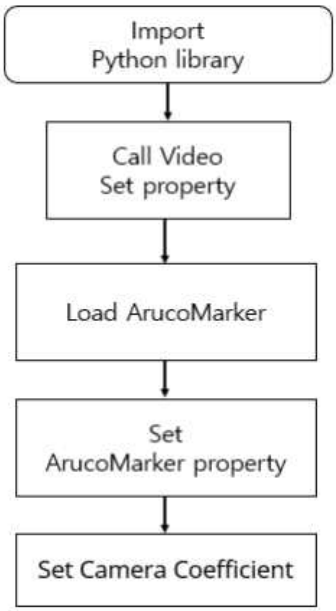


Fig. 5. 카메라와 마커의 좌표

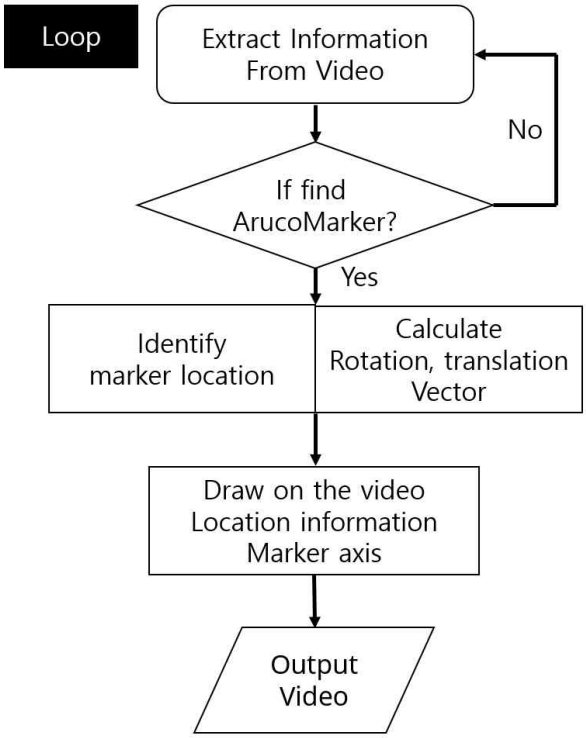
Fig. 5는 카메라를 기준으로 하여 고정한 다음 실제 마커의 동작을 3D좌표로 재구성한 모습이다. 이는 카메라를 기준으로 ArucoMarker는 3개의 회전과 3개의 직선운동 총 6가지 운동을 가상의 공간에서 표현하고 있다. 해당 코드를 설계하는 과정은 4.2.1 회전벡터의 각도 변환 추출에서 자세히 설명한다.

ArucoMarker의 자세와 위치 추정을 위해 Python과 OpenCV로 작성한 기본설정 알고리즘은 다음과 같다.

Code 1은 환경구축에 관한 내용이다. 컴퓨터 비전 라이브러리 OpenCV, 이미지 행렬공간의 선형대수 연산을 위한 Numpy, 시스템의 경로 확인을 위한 os, 시간을 측정하기 위한 Time을 호출한다. 그 다음 사용할 영상의 속성을 결정한다. 그리고 어떤 ArucoMarker를 사용할 것인지 정보를 불러오고 ArucoMarker의 크기를 설정한다. 마지막으로 카메라의 기구적인 왜곡 보정을 위한 카메라 계수를 설정한다.



Code 1. 환경 설정



Code 2. 마커 설정

Code 2는 카메라에서 ArucoMarker를 검출하기 위한 과정을 구현하는 과정이다. 녹화된 영상이라면 영상의 길이만큼 프로그램이 동작 하겠지만 실시간으로 받는 영상은 사용자가 종료명령을 내리기 전까지는 반복해서 작동을 한다. 따라서 영상을 받고 출력하는 과정은 Loop로 반복 진행을 한다. 영상을 받아와서 ArucoMarker를 식별하면 ArucoMarker의 회전벡터와 직선벡터 정보를 획득할 수 있다. 하지만 이렇게 얻은 회전 벡터를 그

대로 사용하기에는 어려운 관계로 로드리게스 좌표계 변환을 이용하여 3x3 회전행렬로 전환한 후 오일러 각을 도입하여 물체의 회전 운동인 Yaw, Pitch, Roll을 구해 화면에 표시한다.

2.2 하드웨어 환경 구축

모델을 구축하기 위해 사용한 라즈베리파이와 모터 부속품, OpenCV 버전과 라이브러리는 다음과 같다.

- OpenCV 4.5.2 ver.
- pigpiod (GPIO제어라이브러리)
- 라즈베리파이4 Model B+ 4GB
- SG-90 180’ Servo Motor
- DC-Motor
- DC-Motor Driver L298n

라즈베리파이에 3개의 서보모터와 3개의 DC모터 총 6개의 모터를 사용하여 ArucoMarker 검증모델을 제어하는 것이 목표이다.

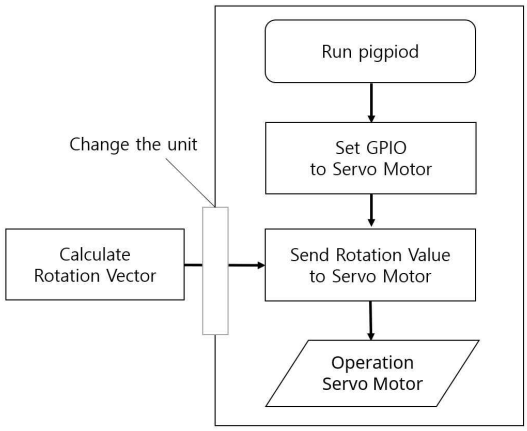
2.2.1 서보모터 회전 모델 제어

Fig. 6. SG-90 180’ ServoMotor

Fig. 6은 자동제어구조 혹은 자동평형 계기에 있어서 입력된 전압을 기계적인 운동을 바꾸어 회전각을 조절하는데 사용하는 전동기인 서보모터이다. [6] 각도의 변위 방향은 펄스 값의 입력에 따라 결정되며, 자체적으로 회전자의 위치를 검출하기 위한 엔코더(Encoder) 또는 레졸버(Resolver)를 탑재하고 있다. 사용자는 펄스 값만 전달해주면 원하는 위치에 따라 서보모터를 간단하게 제어할 수 있다.

Code.3은 서보모터 제어를 위한 순서를 나타낸다. 먼저 서보모터를 제어할 라이브러리를 실행한 다음 조작하고자 하는 서보모터에 각 GPIO Port를 할당한다. 앞서 ArucoMarker의 위치와 자세를 추정했다면 해당 함수 값을 가지고 범위 지정과 단위 변경을 한다.

앞서 ArucoMarker의 자세 정보를 그대로 서보모터에 전해주는 것보다 작동 안정성 보장을 위해 단위 변환을 한 값을 전달한다. 이 프로젝트에서 사용하는 SG-90 180’ 서보모터는 180도의 각도를 움직일 수 있으나 0~180까지 전체 동작을 반복 수행하면 신뢰도가 높은 고가의 서보모터가 아니기에 고장의 우려가 있다. 그래서 끝에서 10도의 여유각을 두었다. 즉 0~10, 170~180의 범위는 사용하지 않으며 서보모터의 가동 범위는 10~170도가 된다.



Code 3. 서보모터 제어

여기서 다시 고려해야할 것은 두 가지로 나누어진다. 첫 번째로 식별마커의 가동범위와 표시되는 값은 -180~180도가 된다. 하지만 서보모터의 작동 범위는 180도이기 때문에 -90~90도로 모델의 가동 범위를 제한한다. 거기에 더해 10도 여유각을 고려하면 총 가동 범위는 -80~80도가 될 것이다. 두 번째는 얼마나 Step 단위를 줘야 하는지 고려해야 한다. 서보모터에 소수점 또는 1~2도에 그치는 적은 차이의 값을 지속적으로 주게 된다면 서보모터에 불필요한 동작과 무리가 갈 수 있다. 그래서 Step는 5도로 설정하였다.

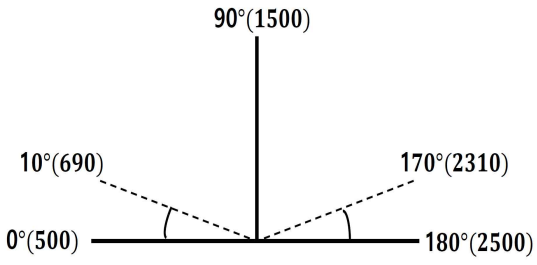
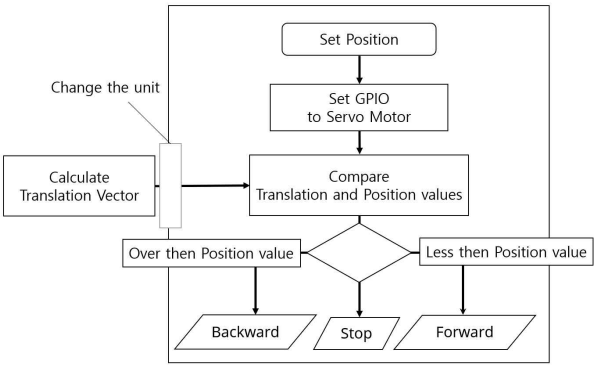


Fig. 7. 서보모터의 제어 범위

2.2.2 DC모터 평행이동 제어

서보모터는 자신의 위치를 기억할 수 있기에 원하는 위치의 펄스 값만 전송하면 끝이지만 DC모터는 현재 자신의 위치를 기억할 수 없으며 시계방향 회전, 반시계방향 회전, 정지 3가지의 명

령만 수행한다. 목표하는 위치와 자신의 위치를 비교할 수 있는 초음파 센서 같은 거리 정보를 획득할 수 있는 장치를 추가로 부착하거나, 아니면 소프트웨어 상에서 자신의 위치를 실시간으로 기록을 하고 추적을 할 수 있어야 한다. 라즈베리파이의 제어 PIN의 제한과 전력공급의 문제로 추가적인 장치 대신 소프트웨어 상에서 이러한 문제를 해결하고자 한다.



Code 4. DC Motor 제어도

Code. 4에서는 DC모터 제어를 위한 순서를 나타낸다. 초기에 프로그램을 시작할 때 수동으로 위치를 0으로 맞춘 다음 들어오는 값과 현재 위치를 비교를 한다. 목표위치가 현재위치보다 클 때 시계방향 회전을 한다. 반대로 목표위치가 현재위치 보다 작을 때는 반대로 반시계방향으로 회전을 한다. 일치 또는 오차범위 내지 허용범위 이내라면 정지 명령을 모터에 전송하는 식이다. 이때 얼마나 이동할 것인가에 대해서는 모터의 종류와 속도에 의해 결정된다. 속도 값을 각각 다르게 주어 같은 시간동안 얼마나 이동하는지 반복적으로 실험해서 적합한 값을 찾는다.

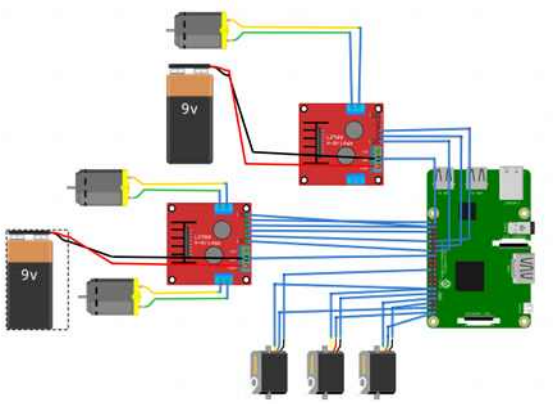


Fig. 8. 라즈베리파이 서보/DC모터 연결도

2.2.3 전체 모델 구축

Fig. 8은 예상되는 모델의 연결도를 표현한 그림이다. 서보모터는 직접적으로 라즈베리파이에서 전력공급을 받을 수 있으나, DC모터의 경우 과전압으로 인한 보드 손상이 있을 수 있기 때문에 9V건전지를 통해 외부전력을 별도 공급한다. 거기에 DC모터는 라즈베리파이 GPIO포트를 통해 바로 제어할 수 없기 때문에 DC모터드라이브를 통해서 DC모터 제어를 편리하게 만든다.

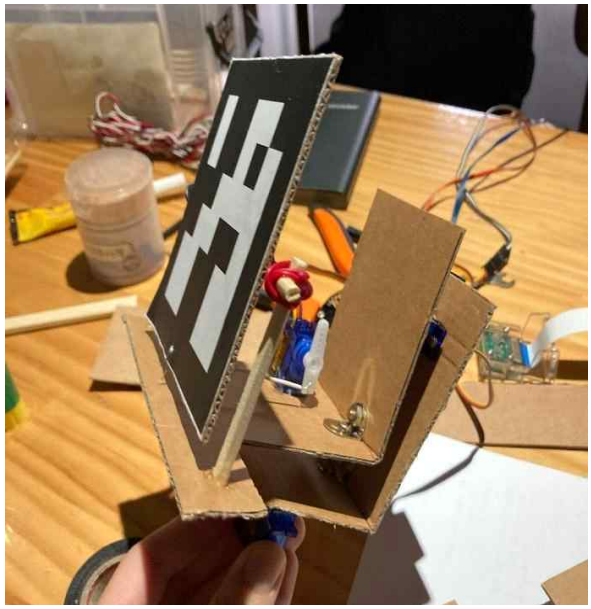


Fig. 9. 프로토타입 제작

실제 설계를 들어가기 전에 종이상자와 플라스틱을 이용하여 프로토타입을 제작한다. 간이 모델로부터 회전 작동 여부를 확인하고 이를 기반으로 치수를 측정하여 최종 설계 모델을 위한 도면을 제작한다.

3D CAD를 이용하여 프로토타입에서 도출한 도면을 만든 다음 항공우주공학과에 비치된 3D Printer MarkBot R+를 이용하여 3D모델을 제작한다. 목표했던 크기가 나오는지, 모터가 들어갈 구멍의 부합 여부를 확인한다.

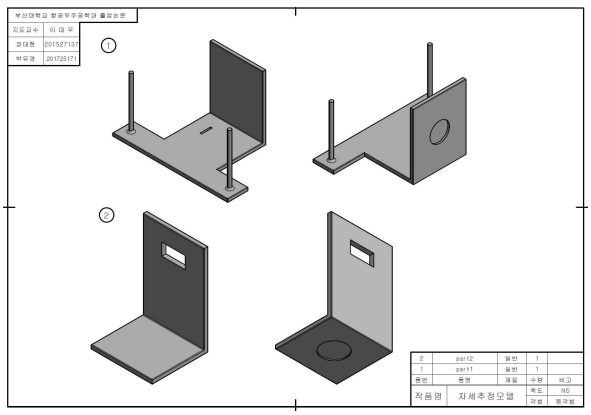


Fig. 10. 회전 장치고정 도면

Fig. 11. 평행이동 간이 모델

평행이동 모델을 구성할 DC모터도 마찬가지로 여러 가지 플라스틱 체인, 고무 체인 등 동력전달 방법을 실험해보며 적합한 모터와 구성을 찾기 위해 반복한다.

3. 과제 해결 방안

3.1 카메라 캘리브레이션을 이용한 왜곡 보정

카메라 캘리브레이션(camera calibration)은 영상처리, 컴퓨터 비전 분야에서 반드시 필요한 과정이다. 사람의 눈을 통해서 보는 세상은 3차원이지만 이를 카메라로 찍게 되면 2차원의 이미지로 변한다. 이 때, 3 차원의 점들이 이미지 상에서 어디에 맺히는지에 대해서 살펴보면 영상 찍을 당시의 카메라의 위치 및 방향에 의해 결정된다. 하지만 실제 이미지는 사용된 렌즈, 렌즈와 이미지 센서와의 거리, 렌즈와 이미지 센서가 이루는 각 그리고 카메라 내부의 기구적인 부분에 의해서 큰 영향을 받는다. 따라서 정확도를 높이기 위해서 카메라의 외부 파라미터와 카메라 자체의 내부적인 파라미터를 변환행렬을 사용한다.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{31} & t_1 \\ r_{21} & r_{22} & r_{32} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = A[R \mid t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

식 1. 카메라 파라미터 변환행렬

카메라의 외부 파라미터는 카메라 좌표계와 월드 좌표계 사이의 변환 관계를 설명하는 파라미터로, 두 좌표계 사이의 회전 및 평행이동 변환으로 표현된다. 따라서 카메라 외부 파라미터는 카메라 고유의 파라미터가 아니기 때문에 카메라를 어떤 위치 어떤 방향으로 설치했는지에 따라 달라지고 또한 월드 좌표계를 어떻게 정했는가에 따라 달라진다. 내부 파라미터는 카메라의 초점거리(focal length: fx, fy), 주점(principal point: cx, cy), 비대칭 계수(skew coefficient: skew_c)가 있다. 이를 포함한 A 행렬을 카메라 내부 파라미터(intrinsic parameter)라고 한다. (X,Y,Z)는 월드 좌표계 상의 3D 점의 좌표, [R|t]는 월드 좌표계를 카메라 좌표계로 변환시키기 위한 회전/이동변환 행렬이며 외부 파라미터(extrinsic parameter)라고 부른다. 내부 파라미터와 외부 파라미터를 합쳐서 camera matrix 또는 projection matrix 라고 한다.

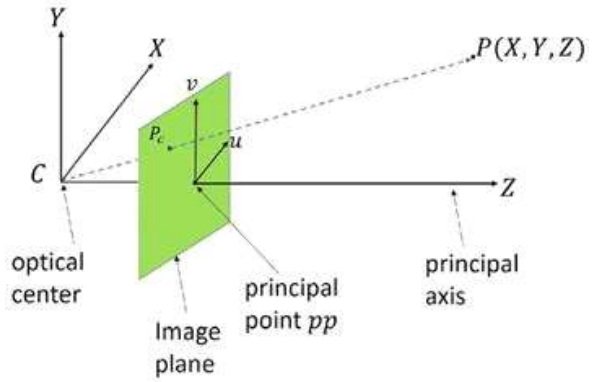


Fig. 12. 주점(principal point) 주축(principal axis)

Fig. 12에서 보는 주점이란 사진의 중심으로, 투영 중심에서 투영면에 내린 수선이 투영면과 만나는 점. 입사동(入射瞳)의 중심을 통하여 사진 면에 직각으로 들어온 광선이 사진 면에 닿은 점을 말한다.

카메라 캘리브레이션을 하기 위해 카메라의 속성부터 검출해야 하며 카메라의 속성은 물리적인 성질에 의해서 결정이 된다. 첫 번째 성질인 카메라의 렌즈 초점 거리를 계산하기 위해 다음과 같은 개념이 필요하다.

초점 거리는 렌즈의 중심점과 초점(피사체)간의 거리를 일컫는 용어로 모든 렌즈는 고정된 초점 거리를 갖는다. 작동 거리가 가변적인 것이 일반적이므로 간단한 계산을 위해 작동 거리 대 초점 길이의 비율에서 시작하는 것이 좋다. 해당 방법을 이용하여 특정 렌즈 초점 길이를 바탕으로 작동 거리를 확인할 수 있다. 작동 거리가 제한적인 경우에는 이 비율을 반전시켜서 초점 길이 대 작동 거리의 비율을 얻는다. 다양한 작동 거리 옵션을 바탕으로 초점 거리 범위를 얻을 수 있다. 렌즈를 선택한 후에는 필요한 작동 거리를 다시 정확하게 계산할 수 있다. 계산은 다음과 같은 방정식을 기반으로 한다.

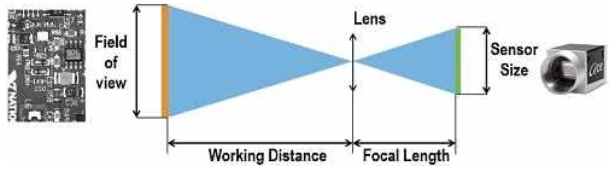


Fig. 13. 카메라와 센서의 변수

$$Focallength \times FOV = Sensor\,Size \times Working\,Distance$$

식 2. 초점거리 공식

* 시야각(FOV): 카메라가 수집해야 하는 검사 대상 영역

* 가장 작은 특징: 이미지에서 검출하려는 가장 작은 특징의 크기

* 작동 거리(WD): 렌즈의 앞면에서 검사 대상 객체까지의 거리

카메라 변환행렬에서 비대칭 계수는 고려하지 않는다. 최근 카메라들은 비대칭 에러가 거의 없기 때문에 카메라 모델에서 보통 비대칭 수까지는 고려하지 않는다. 즉, 비대칭 계수는 0 이다.

내부 요인의 파라미터 값을 구하는 과정은 카메라 자체의 고유 값을 획득하는 방안이다. 카메라 보정 모듈은 입력 영상에 대한 전처리 과정이다. 입력 영상을 흑백영상으로 변환하고 영상 내에 마커 패턴을 검출한다. 그리고 검출된 패턴의 모서리의 3 차원 위치 정보와 영상 내 2 차원 정보를 쌍으로 얻은 후 OpenCV에 내장되어 있는 cal-librateCamera 함수의 입력으로 사용하여 영상에 적용할 카메라 내부 행렬과 왜곡계수를 얻는다. 여기서 왜곡계수는 방사왜곡(radial distortion) 계수(k1, k2) 접선왜곡(tangential distortion) 계수(p1, p2)를 의미한다. 방사왜곡(radial distortion)과 접선왜곡(tangential distortion)의 의미는 다음과 같다.

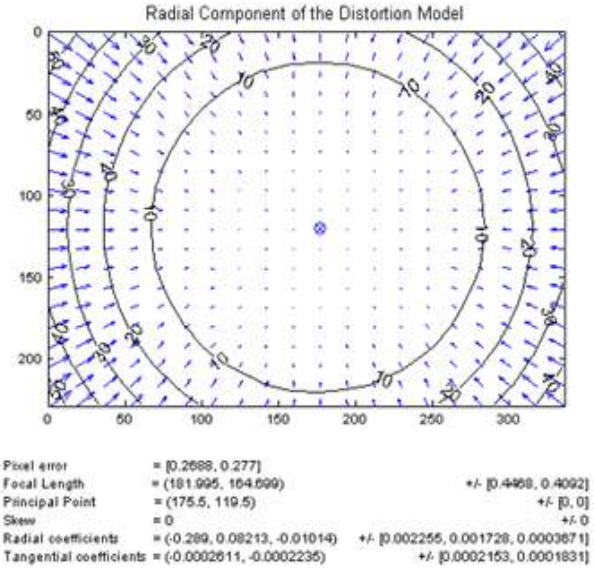


Fig. 14. 방사왜곡(radial distortion)

Fig. 15. 방사왜곡 예시

렌즈는 일반적으로 동그란 형태를 지니게 된다. 하지만 이런 형태의 렌즈는 센서의 가장자리 부근에서 픽셀의 위치가 왜곡되는 것을 볼 수 있다. 이를 방사왜곡이라 하며 렌즈의 중심에서 먼 곳을 통과할수록 빛이 휘어져서 나가기 때문에 발생한다.

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

식 3. 방사왜곡 조정식

식 3에 있는 두개의 항으로 왜곡을 표현하며 테일러 식으로 표현이 가능하다. 왜곡의 정도에 따라서 항의 개수가 결정된다. x,y는 왜곡된 점의 원래 위치를 나타내고 x_corrected와 y_corrected는 보정된 새로운 위치를 표시한다.

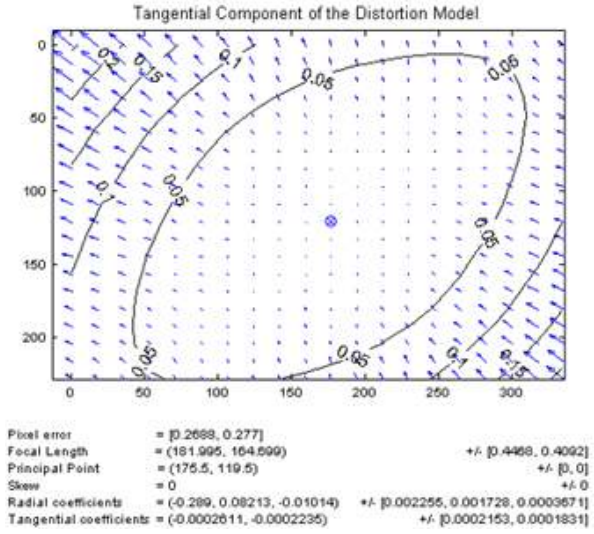


Fig. 16. 접선왜곡(tangential distortion)

접선왜곡은 렌즈와 영상의 평면이 일치하지 않아서 생기는 왜곡이다. 카메라 제조(조립) 과정에서 카메라 렌즈와 이미지센서(CCD, CMOS)의 수평이 맞지 않거나 또는 렌즈 자체의 중심이 맞지 않아서 발생하는 왜곡으로, 아래 Fig. 16과 같

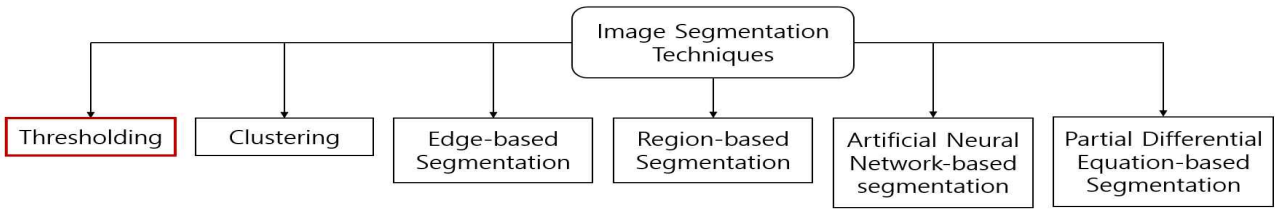


Fig. 17. 이미지 분리 기술 종류

이 타원형 형태로 왜곡 분포가 달라진다.

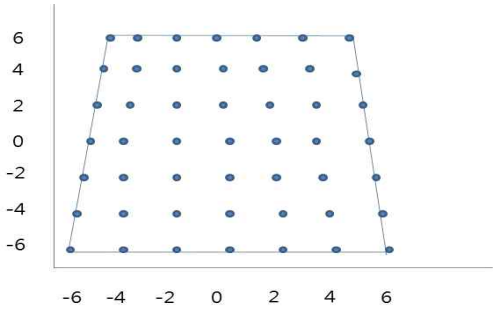


Fig. 18. 접선왜곡 예시

식 4는 접선왜곡 조정식으로 두개의 파라미터 p_1과 p_2로 표현할 수 있다.

$$x_{corrected} = x[2p_1y + p_2(r^2+2x^2)]$$
$$y_{corrected} = y[2p_2x + p_1(r^2+2y^2)]$$

식 4. 접선왜곡 조정식

영상을 획득하는 과정에서 방사왜곡과 접사왜곡 이외에도 여러 가지 왜곡이 존재하나 앞의 두 왜곡에 비하면 영향은 크지 않기에 두 가지 왜곡을 보정하는 것을 목표로 한다.

3.2 이미지 분리 기술

식별마커를 정확하게 검출하기 위해서 영상 후처리와 이미지 분리 기술을 사용하는데 Fig. 17에 보는 여러 가지 기술 중 임계값 처리를 사용한다. 사용한다. ArucoMarker는 흑과 백으로 두 가지 색상으로 이루어진 마커이기에 이미지 이진화(image binarization)를 이용한다.

Fig. 19. Contour 추출 과정 예시

3.2.1 흑백 이미지 사용 및 RGB 영상 후처리

ArucoMarker를 검출하기 위해서 고유의 특징이 무엇인지 정확하게 파악해야 한다. 이를 컴퓨터비전에서는 윤곽(Contour)을 검출한다고 하며 동일한 색 또는 동일한 픽셀값(강도, intensity)을 가지고 있는 영역의 경계선 정보를 뜻한다. 이는 물체의 윤곽선, 외형을 파악하는데 유용하게 사용된다.

먼저 임계값 적용을 위해서 회색조나 흑백이미지를 사용하는데 주로 많이 사용 하는 방법은 GrayScale을 사용해 회색조로 변환을 한다. 변환 공식은 식 5와 같다.

$$Y_{linear} = 0.2126R_{linear} + 0.7152G_{linear} + 0.0722B_{linear}$$

식 5. Luma Coding 변환 공식

GrayScale은 Luma Coding을 이용한 방법[7]으로 R,G,B의 강도에 각 가중치를 곱한 다음 일괄적으로 선형휘도를 구하는 방법이다.

임계값은 다른 말로 문턱값이라고 한다. input으로 들어오는 값이 임계값 이하의 값을 갖는다면 픽셀을 검정색으로 변환하고, 반대로 이상의 값을 갖는 픽셀을 흰색으로 변환한다. 또는 임계값을 기준으로 다른 처리를 한다. 이런 방법으로 흑백으로만 이루어지거나 색상이나 밝기가 조정된 이미지를 얻는다. 보통 사용하는 임계값은 0에서 255의 중간 값인 127으로 설정하고 해당 임계값을 기준으로 픽셀을 전역에 적용한다.

Fig. 19에서 알 수 있듯이 영역을 분할해 임계값을 적용하면 마룻바닥이나 불필요한 영역과 노이즈를 제거하는 효과가 있기 때문에 사물을 탐지할 때 효과적으로 윤곽(Contour)을 추출할 수 있다.

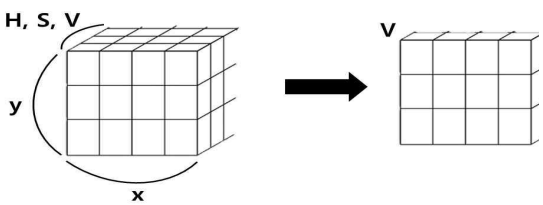
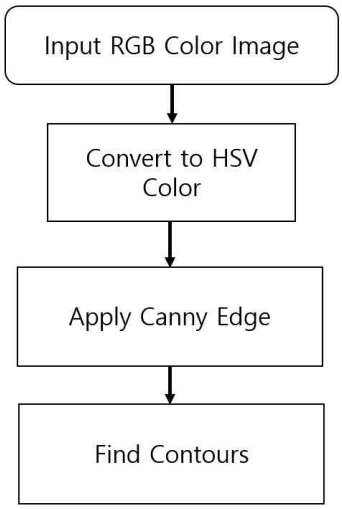


Fig. 20. 이미지 차원의 축소

ArucoMarker은 구성이 흑과 백, 두 가지 색상으로 구성되어 있기 때문에 화면을 흑백으로 바꿔주거나 그에 준하는 영상 처리를 하는 것이 탐지에 유리할 수 있다. Fig. 20처럼 영상을 흑백 이미지로 전환하게 되면 이미지의 차원을 축소할 수 있다. 이로 인해 계산해야 할 연산량이 줄어들게 된다. 그리고 컬러 이미지가 가지고 있던 노이즈도 같이 제거되어 정확도 향상을 얻을 수 있다. 그리고 명암 값만을 가지고 임계점이 뚜렷하게 구분되어 edge detection에 더욱 강력하며 윤곽(Contour) 검출에 유리하다. 영상을 후처리 하거나 특정 윤곽을 검출하기 위한 방법은 탐지하고자 하는 물체의 특성에 따라서 여러 가지 영상 처리 기법이 있다. 마커의 검출률을 올리기 위한 순서도는 다음과 같다.



Code 5. Contour 추출 알고리즘

Code 5는 ArucoMarker에 흑백 처리를 하여 윤곽 검출을 하는 알고리즘을 도식화한 것이다. RGB색상공간으로 들어오는 이미지를 HSV색상공간으로 변환한 다음 Canny Edge기법을 사용하여 영상을 처리한 다음 윤곽을 검출한다. 해당 기법에 대해서는 3.2.3 캐니 에지(Canny Edge)에서 후술한다.

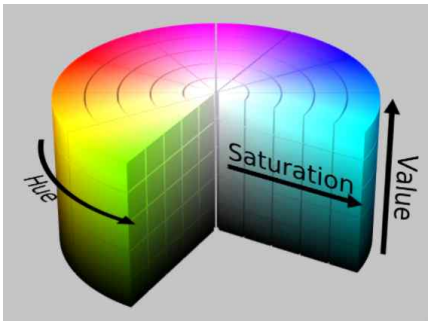


Fig. 21. HSV의 원통형 표현 단면

3.2.2 HSV 색상공간 변환

라즈베리파이 카메라를 통해 촬영된 영상의 기본값은 RGB 색상으로 표현된다. RGB색상 공간은 Red, Green, Blue의 약자로 3원색을 조합해서 표현한다. ArucoMarker의 윤곽을 추출하기 위해서는 RGB색상 보다 회색 음영(Grayscale Color)이나 Fig. 21로 설명되는 HSV 색상이 더 효율적이다.[4] HSV는 색상을 나타내는 H(Hue)와 채도를 나타내는 S(Saturation), 밝기를 나타내는 V(value)로 구성되어 있고 이중 V를 가지고 회색 음영으로 사용할 수 있다.[5] H는 원통형 그림 Fig. 21을 보면 알 수 있듯이 0~360도의 각도로 표현이 가능하나 1바이트에 값 360을 저장할 수 없어 0~180 사이의 정수로 표현하고, S와 V는 0~255 사이의 정수로 표현한다. 색상의 변환은 OpenCV에 내장된 cvtColor() 함수를 사용하여 색상 공간을 변환한다. 그리고 해당 값을 가지고 H, S, V 값을 구분할 수 있는데 이는 3개의 2차원 행렬이 된다. 그런 다음 V값만을 사용한다.

Fig. 22는 Fig 23의 RGB 영상을 받은 다음 이를 HSV각 부분으로 나누었다. 왼쪽부터 색조, 채도, 값에 따른 이미지를 표현하였다. 본 프로젝트에서는 V값 즉 이미지의 밝기 값만을 가지고 영상을 후처리 한다.

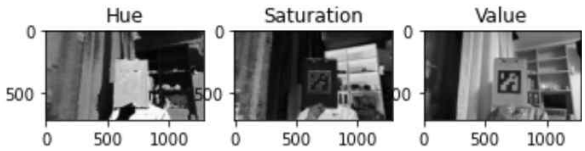


Fig. 22. HSV Color(색조, 채도, 값)

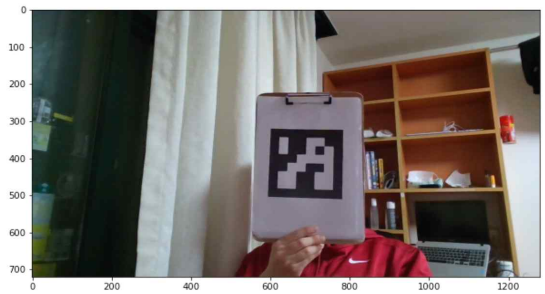


Fig. 23. RGB Color

3.2.3 캐니 에지(Canny Edge)

영상에서 에지(Edge)는 어떤 객체의 가장 바깥 둘레 즉, 객체의 테두리라고 할 수 있다. 해당 테두리는 영상의 객체를 구분하는 지점이며 픽셀의 큰 값에서 작은 값 또는 작은 값에서 큰 값으로 급격하게 변하는 지점이다. 이렇게 픽셀 값이 급격히 변하는 지점, 즉 에지를 찾기 위해서 미분을 통해 기울기 연산을 수행한다. 이는 휴리스틱 탐

색 방식인 Steepest Hill climbing 방법을 이용하여 1차 또는 2차 미분을 통해 찾을 수 있다.[10]

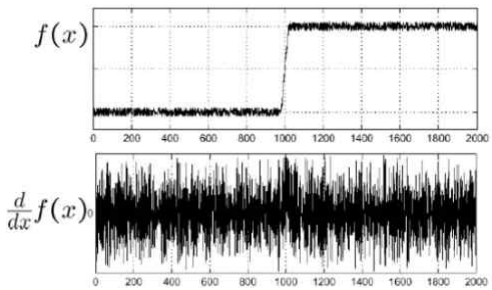


Fig. 24. 미분을 수행한 뒤 발생하는 노이즈

하지만 에지를 찾기 위해선 이미지를 바로 미분부터 수행하지 않는다. Fig. 24에서 보듯이 미분을 수행하고 난 뒤에 잡음(noise)이 급격히 증가하기 때문이다. 이를 해결하기 위해 흐림 처리(blurring) 또는 스무딩(smoothing)을 먼저 사용하여 노이즈를 제거한 다음 미분한다. 블러링과 스무딩을 위한 대표적인 방법으로는 소벨 미분(Sobel derivative), 샤로 필터(Scharr filter), 라플라시안(Laplacian), 캐니 에지(Canny edge) 등이 있으며 이 중에서 캐니 에지 방법을 사용한다.

캐니 에지는 라플라시안 필터 방식을 캐니(J. Canny)가 개선한 방법으로 기존 소벨 에지 검출 방법의 단점으로 지적되는 임계값에 대한 민감도와 휴리스틱한 설정 방식, 그리고 에지 픽셀의 표현 방식에 대한 해결 방식을 제시한다.[11] 에지 검출을 최적화 문제 관점으로 접근한 개념이다. 최적화 문제는 어떤 목적함수(objective function)의 함수 값을 최적화시키는 파라미터 조합을 찾는 문제를 뜻한다. x와 y에 대해 1차 미분을 한 다음 4방향으로 미분을 한다. 그리고 각 결과로 극댓값을 갖는 지점이 에지가 된다. 캐니 에지는 아래의 3가지 부분에서 다른 방법보다 우수하다.[12]

캐니 에지는 다른 에지 검출기보다 성능이 좋으며 잡음에 민감하지 않아 명확한 에지를 검출할 수 있다. 캐니 에지 알고리즘의 동작은 다음 네 가지 순서로 진행된다.

1. 가우시안 필터링: 잡음 제거를 위해 가우스 필터링하여 흐림 효과적용
2. 그래디언트 계산: 기울기 값이 큰 지점을 검출
3. 비최대 억제: 최댓값이 아닌 픽셀을 0으로 변경하여 명백한 엣지가 아닌 값 제거
4. 이중 임계값을 이용한 히스테리시스 에지 트레킹: 히스테리시스 임계값(Hysteresis threshold)을 적용해 윤곽 생성

RGB이미지를 HSV로 변환한 다음 밝기 정보를 담고 있는 Value 값만을 사용하여 가우스 필터링을 거쳐 영상의 노이즈를 줄인다.

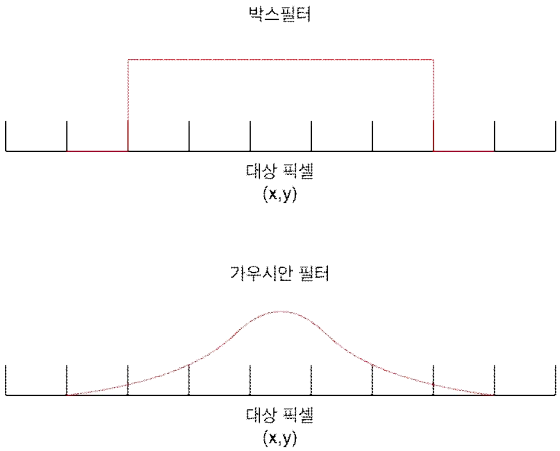


Fig. 25. 박스 필터와 가우시안 필터 차이

Fig. 25의 가우스 분포 그래프는 정규 분포 그래프처럼 가운데가 볼록하게 나온다. 이를 이미지에 적용하면 중앙에 있는 부분을 부각하고 주변을 흐릿하게 하는 마스킹이다. 하지만 가우스 필터 하나만으로는 가장자리가 흐려지고 대비가 줄어든다. 그래서 중앙값 필터를 추가하여 상대적으로 날카로운 가장자리를 유지하면서 노이즈를 제거한다.

가우스 필터링을 통과한 영상에 대해 기울기가 큰 값을 검출한다. Fig. 25는 원본에서 추출한 value 영상에 표준편차를 달리한 가우스 필터링을 적용하였다. 표준편차에 따라서 흐림 정도를 조절할 수 있다.

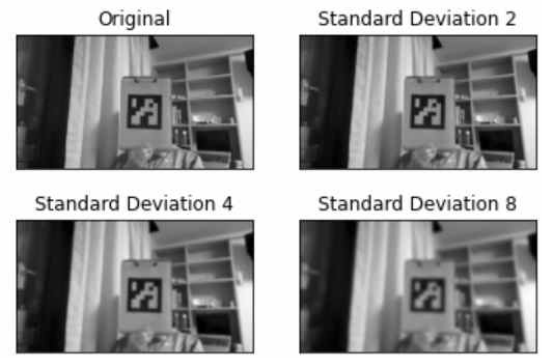


Fig. 26. 가우시안 필터 표준편차 차이비교

3.2.4 임계값(threshold) 설정

일반적으로 이진화는 하나의 임계값(threshold)을 영상 전체에 적용하여 임계값보다 크면 흰색, 작으면 검은색으로 반환한다. 이는 영상의 밝고 어두움을 임계값을 기준으로 나누게 되는 결과가 된다. 캐니 에지 알고리즘은 영상에서 에지로 판

단되는 부분을 흰색으로 그렇지 않은 부분은 검은색으로 이진화 한다.

cv2.threshold(img, threshold_value, value, flag)

Threshold 임계값을 설정하는 매개변수는 4가지로 왼쪽부터 순서대로 다음과 같이 설명한다.

1. img : 적용할 GrayScale 이미지
2. Threshold_value : 픽셀 임계값
3. value : 픽셀 문턱값 보다 클 때 적용되는 최댓값(적용되는 플래그에 따라 픽셀 문턱값 보다 작을 때 적용되는 최댓값)
4. flag : 문턱값 적용 방법 또는 스타일 설정

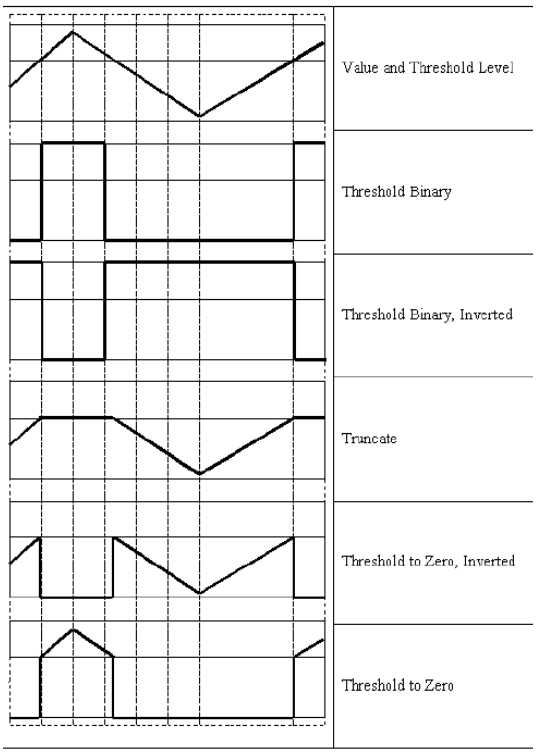


Fig. 27. OpenCV에서 제공하는 Threshold Flag Types

Fig. 27은 OpenCV에서 제공하는 Threshold flag type 이다. 이들 모두 적용해보는 것이 아닌 식별 마커를 잘 구분할 것으로 예상되는 Threshold Binary, Truncate, Threshold to Zero 3가지 방법을 우선 적용한다. 나머지 방법은 inverse 값을 반환하는데 흑과 백이 아닌 반대로 백과 흑을 반환하는 차이기 때문이다. 기본적으로 마커는 검은색 안에 흰색이 들어가는 구조이기 때문에 inverse는 사용하지 않는다.

$$dst(x,y) = \begin{cases} \max value & (\text{if } src(x,y) > threshold) \\ 0 & otherwise \end{cases}$$

식 6. binary 임계값 처리 방법

cv2.THRESH_BINARY: 픽셀 값이 threshold_value 보다 크면 value, 작으면 0으로 할당한다.

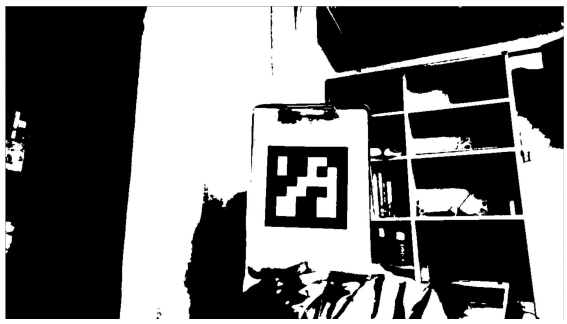


Fig. 28. THRESH_BINARY 적용

cv2.THRESH_TRUNC: 픽셀 값이 threshold_value 보다 크면 threshold_value, 작으면 픽셀 값 그대로 할당한다.

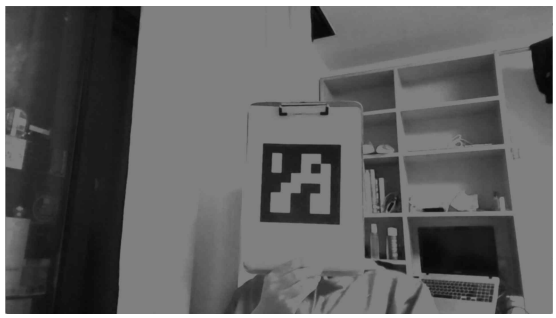


Fig. 29. THRESH_TRUNC 적용

$$dst(x,y) = \begin{cases} src(x,y) & (\text{if } src(x,y) > threshold) \\ 0 & otherwise \end{cases}$$

식 7. Trunc 임계값 처리 방법

cv2.THRESH_TOZERO: 픽셀 값이 threshold_value 보다 크면 픽셀 값 그대로, 작으면 0으로 할당한다.

Fig. 30 THRESH_TOZERO 적용

$$dst(x,y) = \begin{cases} threshold & (\text{if } src(x,y) > threshold) \\ src(x,y) & otherwise \end{cases}$$

식 8. ToZero 임계값 처리 방법

cv2.THRESH_TOZERO: 픽셀 값이 threshold_value 보다 크면 픽셀 값 그대로, 작으면

0으로 할당한다.

임계값을 단순하게 적용했을 때 보기에는 마커의 식별이 분명하지만 치명적인 문제가 있다. 임계값을 이미지 전체에 적용하여 처리하기 때문에 하나의 이미지에 음영이 다르면 일부 영역이 모두 흰색 또는 검정색으로 보이게 된다. 이는 오히려 검출률을 떨어뜨리는 문제가 발생한다. 해당 문제를 해결하기 위해서 이미지의 작은 영역별로 thresholding을 조정하는 방법이 있다. 이때 사용하는 함수는 adaptiveThreshold() 이다.

adaptiveThreshold 임계값을 설정하는 매개변수는 6가지로 왼쪽부터 순서대로 다음과 같이 입력한다.

1. img : 적용할 GrayScale 이미지
2. maxValue : 픽셀 임계값 설정
3. adaptiveMethod : thresholding value를 결정하는 계산 방법
4. thresholdType : 문턱값 적용 방법 또는 스타일
5. blockSize : thresholding 을 적용할 영역 사이즈, 영역의 사이즈는 3, 5, 7, 9 같은 홀수를 사용해야 한다. 짝수개의 사이즈는 중앙점이 없기 때문이다.
6. C :평균이나 가중평균에서 차감할 값

Adaptive Method는 여기서 두 가지로 더 나누어지는데 평균값처리와 가우시안 처리 방법이 있다.

1. cv2.ADAPTIVE_THRESH_MEAN_C

픽셀 (x, y)를 중심으로 blockSize * blockSize 안에 있는 픽셀값의 평균에서 C를 뺀 값을 임계값으로 설정한다. 즉 주변영역의 평균값으로 결정한다.

2. cv2.ADAPTIVE_THRESH_GAUSSIAN_C

픽셀 (x, y)를 중심으로 blockSize * blockSize 안에서 가우시안 윈도우와의 교차상관을 통해 만들어진 가중치 합에서 C를 뺀 값을 임계값으로 설정한다. 즉 가우시안 값으로 결정한다.

결과적으로 adaptiveThreshold를 적용했을 때 경계점을 분명하게 한다는 점에서 에지 검출에 탁월하지만 검출 대상인 식별마커를 기준으로 놓는다면 마커를 오히려 흐리게 만들어 버린다. blockSize의 크기를 좀 더 크게 조정하면 마커도 선명하게 만드는 만족할 만한 결과를 얻을 수 있을지 모르나 카메라에 맺히는 마커의 크기, 거리에 따라 만족하는 blockSize가 달라진다. 마커의 크기뿐 아니라 거리가 고정된다는 조건이 추가될 때 사용할 수 있기 때문에 이번 목표에는 부합하지 않아 실제 환경에서 사용하지 않았다. adaptiveThreshold 대신 임계값을 조절하는 방법을 적용해 본다.

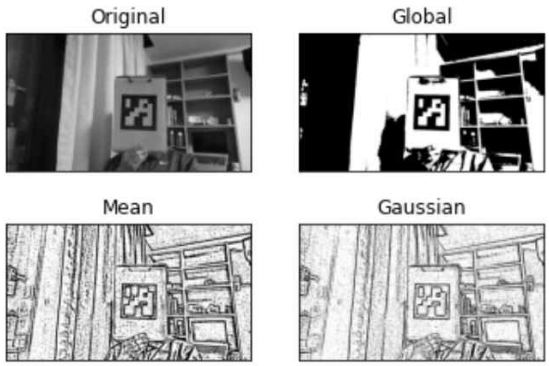


Fig. 31. adaptiveThreshold를 적용한 사진

지금까지 thresholding 처리를 하면서 임계값을 사용자가 결정하여 parameter로 전달하였다. 일반적으로 이진화 하기 좋은 적당한 임계값을 주었으며 앞서 지금까지 0에서 255의 중간 값인 127을 선택했다. 또는 사용자가 직접 trial and error 방식으로 여러 번 반복하여 가장 적절하다고 생각하는 임계값을 선택하였다. 이런 수작업이 아닌 이미지 히스토그램이 두 개의 봉우리를 가지는 bimodal image의 경우는 임계값을 어느 정도 정확하게 계산할 수 있다.

Fig. 31.의 이미지 히스토그램은 이미지의 콘트라스트(Contrast), 밝기(Brightness), 색감 분포와 같은 이미지 특성을 시각화한 모형이다. 여기서 두 개의 봉우리를 가진 특성을 가진 이미지를 bimodal image라고 하며 이를 이용하여 임계값을 계산할 수 있다.

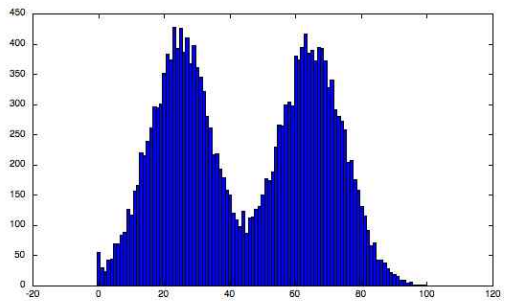


Fig. 32. bimodal image

이미지가 이런 종류의 bimodal image일 때 다른 영상보다도 bimodal image인 경우 잘 작동하는 Otsu의 이진화(Otsu's Binarization)를 사용을 시도해볼 수 있다. Otsu 알고리즘은 임계값을 임의로 정해 픽셀을 두 부류로 나누고 두 부류의 명암 분포를 구하는 작업을 반복한다. 모든 경우의 수 중에서 두 부류의 명암 분포가 가장 균일할 때의 임계값을 선택한다.[13] 적용 방법은

cv2.threshold() 함수의 flag parameter에 추가로 cv2.THRESH_STSU를 적용하면 된다. 이때 임계값 parameter은 0으로 전달하면 된다.

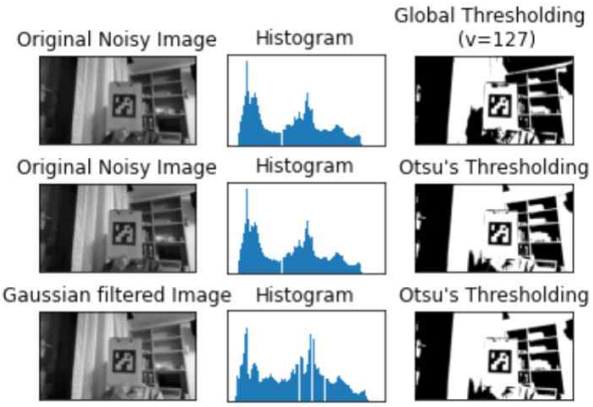


Fig. 33. Otsu's thresholding을 사용한 이미지

Fig.33는 Global threshold값, Otsu thresholding적용, Gaussian blur를 통해 noise를 제거한 후 Otsu thresholding적용 결과이다. 앞에서 임계값 flag에 따라 다른 결과를 얻었던 것처럼 각 flag에 Otus를 모두 각각 적용해 볼 수 있다. 앞서 선택한 3가지 임계점 방법인 Threshold Binary, Truncate, Threshold to Zero에 대해서 적용해 본 다음 결과를 비교를 통해 식별마커 검출에 가장 적합한 Threshold Flag Types을 탐색한다.

본 프로젝트에선 ArucoMarker의 외곽선이 아닌 ArucoMarker 자체를 검출하는 것이 중요하다. 따라서 캐니 에지 마지막 부분인 외곽선 검출 과정은 생략하고 지금까지 수행한 여러 가지 임계값 후처리 영상을 가지고 식별마커 탐지를 수행한다. 먼저 3가지 방식의 임계값 처리, 그리고 Otsu를 추가한 영상 또는 단순히 내장된 GrayScale 변환을 수행할 때 처리를 측정하여 검출 안정성이 높은 것을 선택한다.

3.3 pigpiod 라이브러리 사용

본 프로젝트에서 사용하는 서보모터에서는 지터링(jittering)이라 불리는 떨림 현상이 발생한다. 멀티태스킹을 지원하는 운영체제에서 동작할 때 서보모터의 떨림 현상이 발생한다. 이를 해결하기 위해 pigpiod 라이브러리를 사용한다. 해당 라이브러리는 C언어 기반으로 이루어져 있다. 같은 고급 언어로 분류되는 C언어와 파이썬이지만 둘은 메모리 접근 방식에서 차이를 보이는데 파이썬은 가비지컬렉션(GC, Garbage Collection)을 사용한다. 메모리 관리 기법 중의 하나로 프로그램이 동적으로 할당했던 메모리 영역 중에서 필요 없는 영역을 자동으로 해제하는 기능이다.[14] 이를 사용하면 유효하지 않은 포인터 접근, 이중 해제, 메모리 누수 등을 방지한다는 장점이 있다.

하지만 할당했던 메모리를 언제 해제할지 결정하는데 컴퓨터 자원을 계속해서 점유함과 가비지컬렉션이 발생하는 시점을 예측하기 어려워진다. 할당된 메모리가 해제되는 정확한 시점을 알 수 없어 프로그램이 예측 불가능하게 동작 하거나 불규칙적으로 시스템이 정지할 가능성이 있다. 이런 특성은 실시간 시스템에 적합하지 않다. 반면 C언어는 가비지컬렉션 처럼 자동으로 메모리를 관리하지 못하지만 동적 메모리 관리를 지원하며 이는 명시적인 지점에서 메모리 할당과 해제를 수행한다. 따라서 직접적인 메모리 접근과 정확한 시점의 제어 명령을 통해 이루어진다. 이는 빠른 동작에 유용하며 실시간 시스템에 적합한 특성을 지니고 있다. pigpiod는 C언어의 메모리 접근 특성을 이용하여 하드웨어 제어와 연산처리 부분을 나누어 서로 간섭을 최소화하였다. 프로그램 동작 전 사용자가 직접 제어하지 않고 백그라운드에서 돌면서 여러 작업을 하는 pigpiod 데몬을 먼저 실행시켜 시스템 자원을 일정 부분 점유하여 독점적으로 서보모터를 제어할 자원을 확보한다. 이와 같은 이유로 라즈베리파이에 내장된 GPIO 라이브러리가 아닌 pigpiod를 이용하여 서보모터를 제어하기로 결정하였다.

4. 개념설계 및 상세설계(계산)

4.1 개념설계

개념설계를 통해 우리가 어떤 것을 만들 계획인지 정의한다. 설계를 하드웨어와 소프트웨어 부분으로 구분하였으며 해당 목표를 수행하기 위한 설계 과정은 다음과 같다.

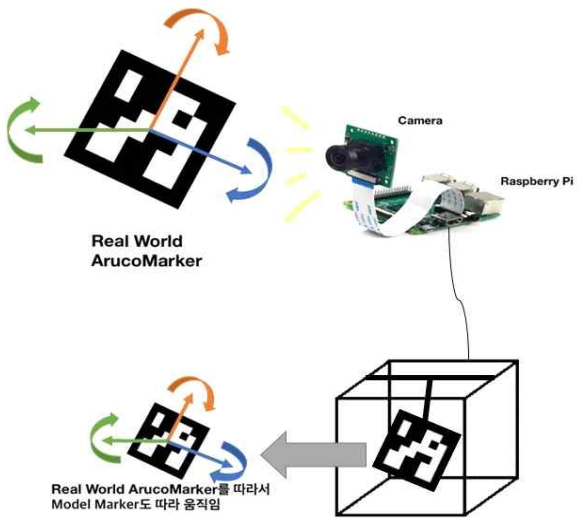
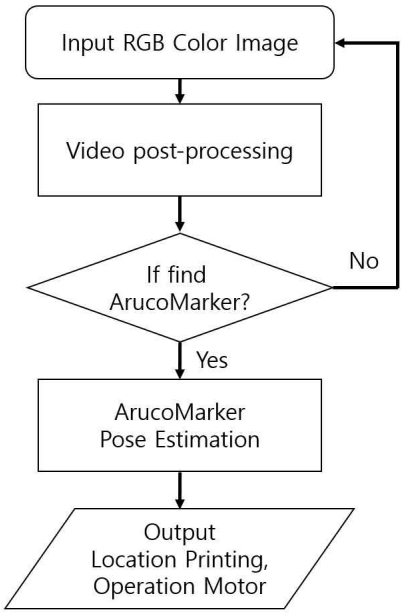


Fig. 34. 하드웨어의 설계 예상도



Code 8. 소프트웨어 설계 전체 알고리즘

이전 설계상에서의 완성은 Fig. 34 같이 나온다. 실제 마커를 라즈베리파이에 연결된 카메라에 보여주면 식별마커 또는 카메라를 기준좌표로 위치와 각도를 추출하고 각 해당하는 값을 3개의 서보모터와 3개의 DC모터에 전달해 준다. 이어 해당 값을 전달받은 모터는 실시간으로 모형을 따라서 동작하게 하며 이를 가지고 실제로 마커를 정확하게 인식하는지 한 번 더 검증을 거치게 된다.

후처리 방법이 결정된 다음 알고리즘은 Code 8.의 순서도를 따라 카메라로부터 이미지를 입력받고 앞서 선택한 후처리 방법으로 영상을 처리한다. 후처리 된 이미지에 마커가 있는지 판단을 하며 마커가 있다고 판단을 하면 해당 마커의 자세를 추정을 하고 추정한 값을 화면에 출력과 동시에 모터에 전달을 하게 된다. 여러 가지 후처리 방법 중 외부 광원에 지나치게 반응하지 않으며, 안정적으로 식별마커를 검출할 수 있는지가 선정 기준이 된다.

4.2 상세설계

개념설계를 바탕으로 실제 설계(상세설계)를 시작할 때 직관이 아닌 데이터를 기반으로 이루어지며 왜 이런 결정을 하였는지 뒷받침할 수 있는 명확한 근거를 제시한다.

4.2.1 회전벡터의 각도 변환 추출

OpenCV의 estimatePoseSingleMarkers() 함수를 사용하면 ArucoMarker의 자세 추정을 할 수 있다. 하지만 함수의 반환값으로 출력하는 인자는 모두 벡터 값으로 카메라를 기준으로 하는 회전 벡터와 평행이동 벡터를 반환한다. 이는 회전 행렬에 대한 가장 간결하고 압축적인 표현 방식이기 때문이다.[15]

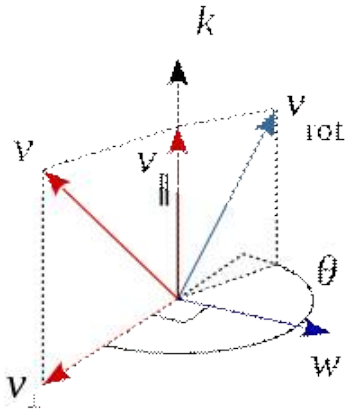


Fig. 35. Rodrigues 회전 표현

3차원에서 회전 변환은 3x3 행렬로 표현하지만 Rodrigues 회전 공식을 사용하면 3차원 회전 변환을 4개의 값(회전축 벡터 + 회전각) 만으로 표현할 수 있다. Fig. 35에서 벡터 k를 k에 평행하고 수직인 성분으로 분해한 다음 수직 성분만 회전시켜 v를 Θ각도로 회전한다. Θ 값은 다음 식1에 의해 계산될 수 있다. 이 식을 Rodrigues' rotation formula라고 부른다.

$$P_{rot} = pcos\theta + (v \times p)\sin\theta + v(v \bullet p)(1-\cos\theta)$$

식 9. Rodrigues' rotation formula

이때, v는 단위벡터(unit vector)이며 회전 방향은 오른손 법칙을 따른다.

$$P_{rot} = \left(\cos\theta I + \sin\theta \begin{bmatrix} 0 & -v_z & v_z \\ -v_z & 0 & -v_x \\ -v_y & -v_x & 0 \end{bmatrix} + (1-\cos\theta)vv^T \right) p$$

식 10. 행렬 형태의 표현

식 9 로부터 Rodrigues v = (v_x,v_y,v_z), Θ에 대응하는 회전변환 행렬 R이 식 10와 같음을 알 수 있다.

$$R = \cos\theta I + \sin\theta \begin{bmatrix} 0 & -v_z & v_z \\ -v_z & 0 & -v_x \\ -v_y & -v_x & 0 \end{bmatrix} + (1-\cos\theta)vv^T$$

식 11. 회전 변환 행렬 R

식 11 은 임의의 회전축에 대한 회전을 회전변

환 행렬로 표현할 수 있다.

OpenCV에서는 회전변환행렬과 Rodrigues 표현 사이의 상호 변환을 위해 Rodrigues()란 함수를 제공한다. 하지만 OpenCV에서의 Rodrigues 표현은 보다 간결한 형태로 단 3개의 값만으로 회전변환을 표현한다. 이로부터 회전각(Θ) 및 회전축 벡터(v)는 다음과 같이 추출한다.

$$\theta = \sqrt{a^2 + b^2 + c^2}$$

식 12. 회전각 벡터

$$v = \left[\frac{a}{\theta}, \frac{b}{\theta}, \frac{c}{\theta}\right]$$

식 13. 회전축 벡터

식 12과 식 13를 통해 벡터에서 회전각을 추출할 수 있다. 이어서 회전행렬을 오일러 각도를 이용하여 재구성한다. 오일러 각(Euler angle)은 강체가 놓인 방향을 3차원 공간에 표시하기 위해 레온하르트 오일러가 도입한 세 개의 각도이다.[16] 3차원 회전 군의 한 좌표계로 3차원 공간에 놓인 강체의 방향은 오일러 각도를 사용하여 세 번의 회전을 통해 얻을 수 있다.

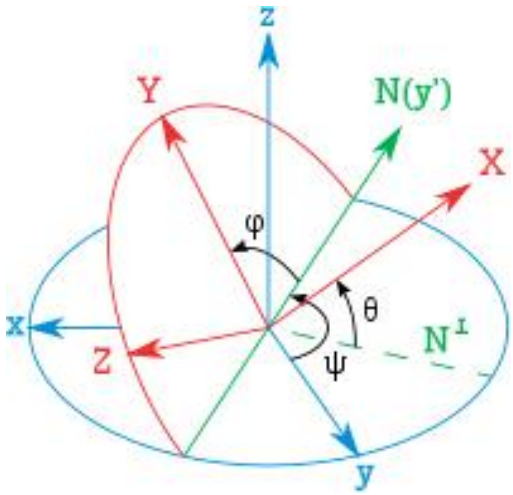


Fig. 36. Tait-Bryan 각도. z - y '- x ″ 시퀀스

오일러 각을 사용하기 위해서는 회전축 시퀀스를 결정해야 한다. 항공우주분야에서 주로 사용하는 Tait-Bryan 각도를 이용하여 회전을 설명하며 회전을 설명하는 6가지 Tait-Bryan 시퀀스 중 ZYX를 사용한다.

ZYX 시퀀스를 사용하는 이유는 Fig. 36와 같이 항공 우주에서 주로 사용하는 강체의 회전 설명에 적합한 요(yaw), 피치(pitch), 롤(roll)에 대응되기 때문이다. Tait-Bryan 각도의 명명법을 따라 다음 행렬을 사용한다.

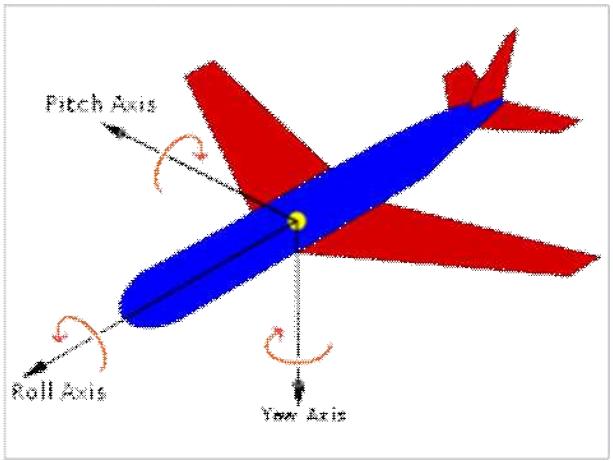


Fig. 37. 주축을 따르는 항공기 운동 표현

$$Z_1 Y_2 X_3 = \begin{bmatrix} c_1c_2 & c_1s_2s_3 - c_2s_1 & s_1s_2 + c_1c_3c_2 \\ c_2c_1 & c_1c_2 + s_1c_2c_3 & c_3c_1c_2 - c_1s_2 \\ -s_1 & c_2c_3 & c_2c_3 \end{bmatrix}$$

식 14. Tait-Bryan 각도의 명명법에 따른 행렬

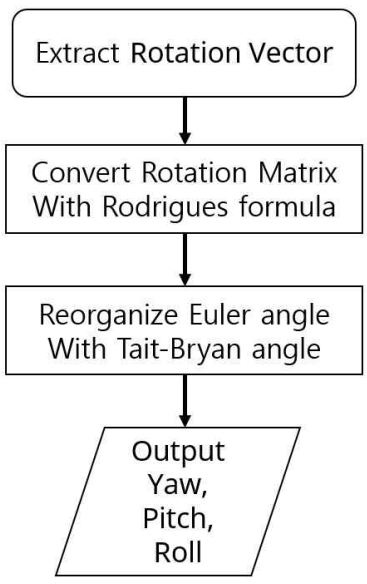
앞에서 구한 회전 행렬을 가지고 식 14.를 이용하면 ZYX 시퀀스의 각도를 구할 수 있다.

$$\alpha = \arctan\left(\frac{R_{21}}{R_{11}}\right)$$

$$\beta = \arctan\left(\frac{-R_{31}}{\sqrt{1 - R_{31}^2}}\right)$$

$$\gamma = \arctan\left(\frac{R_{32}}{R_{33}}\right)$$

식 15. 회전행렬에서의 ZYX 시퀀스의 각도 표현



Code 9. Yaw, Pitch, Roll 추출 알고리즘

수식에 따라 행렬 계산을 사용하여 값을 구할 수 있도록 프로그래밍 하였다. Code 9.는 순서도를 이용하여 정리한 것이다. OpenCV 내장함수를 이용하여 식별마커의 회전벡터를 추출하고 이를 로드리게스 공식을 이용하여 회전행렬로 변환한다. 오일러 각도를 이용하여 강체의 회전운동 설명에 적합한 Yaw, Pitch, Roll을 출력한다.

4.2.2 카메라 캘리브레이션 결과

Fig. 38. 카메라 캘리브레이션 과정

3.1에서 설명한 카메라 캘리브레이션을 이용하여 입력으로 들어오는 영상의 왜곡을 보정하여 카메라의 기구적 왜곡을 개선할 수 있다. Fig. 38는 체스판 보드를 이용하여 카메라 캘리브레이션을 하는 것이다. 동일한 크기의 체스판을 다양한 거리와 각도에서 촬영한 다음 이를 가지고 체스판의 특징점의 차이를 비교하여 카메라 계수를 찾아낸다. 본 프로젝트에서 수행한 라즈베리파이의 카메라 캘리브레이션 결과와 파라미터의 값은 다음과 같다.

$$s\begin{bmatrix}x\\y\\1\end{bmatrix}=\begin{bmatrix}914.426678 & 0 & 640\\ 0 & 914.426678 & 360\\ 0 & 1 & 914.426678\end{bmatrix}\circ \begin{bmatrix}r_{11}\,r_{12}\,r_{31}\,t_1\\ r_{21}\,r_{22}\,r_{32}\,t_2\\ r_{31}\,r_{32}\,r_{33}\,t_3\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}=A[R\mid t]\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}$$

식 16. 카메라 캘리브레이션 결과

- Focal length

f_x = 914.426678, f_y= 914.426678

- Principal point

c_x = 640.000000, c_y = 360.000000

- radial distortion 계수

k_1 = -0.006462, k_2 = 0.852628

- tangential distortion 계수

p_1 = -0.003371, p_2= -0.003302

4.2.3 비디오 후처리 결과 비교

기본적으로 일반적인 RGB 사진을 그대로 사용하는 것보다 최소한 GrayScale 등의 흑백 처리를 하거나 HSV에서 V 값만 사용하는 것이 더 뛰어나다고 알려져 있다. 이를 가우시안 필터로 노이즈를 제거하고 임계점 기법을 통해 차이를 극대화하는 것이 식별마커를 검출하는데 더 도움이 되는지를 비디오 후처리 결과 데이터 분석 통해서 검증한다. 동일한 환경에서 라즈베리파이에 연결한 카메라와 식별마커를 고정시킨 다음 같은 거리에서 반복적으로 측정을 실행한다. 100cm 거리에서 식별마커의 pitch를 35도로 두고 나머지를 원점에 가깝게 두었다. 반복적으로 측정을 했을 때 불안정하거나 오차가 심하다면 측정값의 분포는 커지기 때문에 표준 편차는 커지게 되며 안정적으로 측정이 이루어진다면 오차는 줄어들 것이고 통계적인 표준편차는 더 작게 나타날 것이다. 여러 가지 조건에서 얼마나 안정적으로 측정이 가능한지 확인해 본다.

Table 1. Origin의 표준편차

	Pitch	yaw	roll
표준편차	0.455846	0.307932	0.161899
	x	y	z
표준편차	0.013079	0.050849	0.077839

Table 2. GrayScale의 표준편차

	Pitch	yaw	roll
표준편차	0.710796	0.384159	0. 18413
	x	y	z
표준편차	0.018197	0.055181	0.141256

Table 3. HSV 분해 (Value 값만 사용) 표준편차

	Pitch	yaw	roll
표준편차	0.373595	0.356534	0.081717
	x	y	z
표준편차	0.037300	0.028597	0.128527

맨 처음으로는 기본값으로 들어오는 RGB 사진과 단순히 GrayScale을 적용한 방법과 HSV로 색상 공간을 변경한 다음 Value 값만 추출한 것을 비교해 보았다. 통상적으로 외곽선 검출 등에 있어서는 컬러 이미지 보다 흑백 이미지를 사용하는 게 효과가 더 좋을 것이라고 생각했으나, 위의 3개 중 검출 값이 가장 많이 튀는 것은 GrayScale로 변환했을 때이다.

Table 4. Binary의 표준편차

	Pitch	yaw	roll
표준편차	0.583243	0.341963	0.223821
	x	y	z
표준편차	0.015866	0.044379	0.194858

Table 5. Trunc의 표준편차

	Pitch	yaw	roll
표준편차	0.454588	0.305962	0.152649
	x	y	z
표준편차	0.010644	0.037798	0.092501

Table 6. ToZero의 표준편차

	Pitch	yaw	roll
표준편차	0.706449	0.337682	0.149674
	x	y	z
표준편차	0.011289	0.05022	0.125794

HSV에서 V를 추출한 다음 Binary, Trunc, ToZero를 이용하여 임계값 처리를 한 비교한 것이다. 하지만 의외로 3결과 모두 Origin을 기준으로 비교를 해보았을 때 객관적으로 더 나은 성능을 보여주지 못했으며 이미지 상으로는 가장 선명하게 보였던 Binary의 경우 Origin에 비하면 y값의 표준편차를 제외하고 모두 증가하였다. 더해서 Binary는 광원과 음영에 취약하다 Fig. 39.에 사용한 ArucoMarker은 흑백으로 프린트된 그림이지만 빛에 따라서 전체를 흰색으로 표시하거나 조금이라도 어두워지면 흑으로 덮어버린다. 지나치게 외부환경에 민감한 특성은 사용할 수 없다. 이어서 각 임계값 기술에 Otsu를 적용한 방법을 차례대로 비교해 본다.

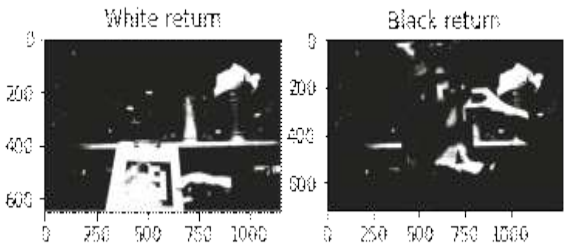


Fig. 39. 광원에 취약한 Binary 임계값 적용

Table 7. Otsu + Binary의 표준편차

	Pitch	yaw	roll
표준편차	0.336801	0.238597	0.092925
	x	y	z
표준편차	0.008774	0.030061	0.056198

Table 8. Otus + ToZero의 표준편차

	Pitch	yaw	roll
표준편차	0.428568	0.326283	0.151563
	x	y	z
표준편차	0.013863	0.045095	0.09374

Otsu 기술을 앞서 3가지 임계값 처리에 적용했다. Trunc의 경우 Otus를 적용하자 오히려 마커 식별 범위가 현저하게 떨어졌다. Binary에서 문제없이 검출하는 장면에서도 Trunc에 Otsu를 적용하는 순간 검출을 하지 못했기에 측정에서 제외하였다. Binary에 Otsu를 적용한 기술은 이전과 마찬가지로 광원에 지나치게 민감하다는 문제점을 그대로 지니고 있었으며 마지막으로 사용 가능한 것은 ToZero에 Otsu를 적용한 것이다. 광원에 민감하지 않고 검출률 역시 저하되지 않았으며 맨 Table 1과 Table 8을 비교해 보았을 때 기준으로 삼은 Origin보다도 나은 성능을 보여주었다. 표준편차는 최소 22.5% 감소하였으며 최대 42.6%까지 감소하였다.

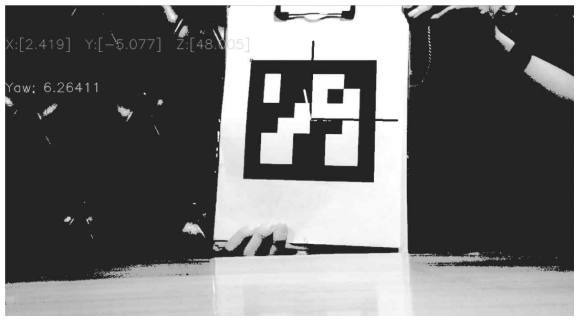


Fig. 40. Otus+ToZero를 이용한 측정

Table 9. 표준편차 결과 정리

표준편차	Origin	ToZero	Otsu+ToZero
pitch	0.455846	0.706449	0.336801
yaw	0.307932	0.337682	0.238597
roll	0.161899	0.149674	0.092925

Table 9. 에서 정리한 결과를 보면 단순히 ToZero만 사용했을 경우 Origin보다도 못한 결과를 보여주었으나 Otus를 적용하자 전체적으로 검출의 표준편차가 현저하게 떨어진 것을 확인할 수 있다. Origin과 Otus+ToZero를 비교하였을 때 pitch에서는 26.1% 표준편차가 감소하였으며, yaw는 22.5% 감소, roll은 42.6% 감소하였다. 최종적으로 이번 실험에서 마커를 추정하기 위해서 사용할 비디오 후처리 기술은 ToZero + Otus를 이용하여 임계값을 자동으로 설정했으며 가우시안 필터의 표준편차를 5로 사용했다.

Fig. 41은 원본 RGB 색상 공간을 Origin으로 두고 Otus+ToZero의 식별마커 측정값을 그래프화 했다. 초록색 선은 기준점으로 pitch 각도를 35도로 두고 식별마커를 측정한 값에서 Origin의 범위는 (34.6~36.0) 1.4으로 값이 분포하는 형태를 보이나 Otus+ToZero의 경우 범위는 (34.4~35.3) 0.9로 범위가 0.5 감소했으며 Origin에 비해서 기준으로 삼은 35도에 가까운 값이 분포되는 것을 볼 수 있다.

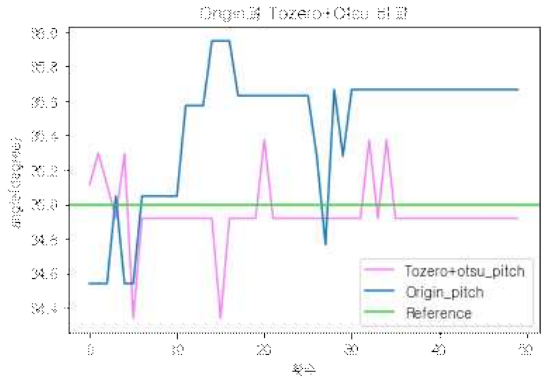


Fig. 41. Otus+ToZero를 이용한 측정값 그래프

Table 10. 평균오차와 범위 변화량 비교

	Origin	ToZero+Otsu
최대	35.94743	35.37376
최소	34.53911	34.33658
범위	1.408316	1.037175
평균오차	0.449721	0.055261

Table 11.은 최대 최소와 범위 평균오차를 비교했다 먼저 측정 범위는 1.408에서 1.037로 0.371 감소하였으며, 평균오차의 경우 35도를 기준으로 0.450에서 0.055로 대폭 감소하는 결과를 얻을 수 있었다.

4.2.4 마커의 거리와 크기에 따른 표준편차 변화

Table 11. 마커 크기에 따른 표준편차 변화
(거리 150cm)

Size	20cm	15cm	10cm
표준편차	0.479562	0.662548	0.842138

Table 12. 거리에 따른 표준편차 변화
(마커 크기 15x15cm)

Distance	150cm	100cm	50cm
표준편차	0.662548	0.342701	0.241673

Table 11과 12는 후처리를 하지 않은 원본 RGB 색상에서 마커와 카메라의 거리를 150cm으로 고정을 하고 마커의 크기만 변화시키며 측정한 결과와 마커의 크기를 15cm으로 고정을 하고 거리를 조절해 측정한 결과 값이다. 거리는 가까울수록 마커의 크기는 클수록 마커의 식별력은 더 강해지는 결과가 있다. 하지만 실제로 사용한다고 가정했을 때 마커의 크기를 조절하는 것은 특정한 상황이 아니라면 의미가 없으며 마커의 크기를 무작정 키우는 것 역시 현실적인 해결책이 되지 못한다. 다만 A4의 크기를 고려하였을 때 최대 20cmx20cm 크기의 마커를 사용할 수 있기에 A4 기준 가장 큰 크기의 마커를 담아서 진행을 하였다.

5. 최종설계 결과물

5.1 설계 과제물

설계 과제물은 크게 두 가지로 회전모델과 평행이동 모델로 구분된다. 회전모델은 서보모터 3개로 3가지 회전축을 구성하며, 평행이동 모델은 3개의 DC모터로 3축의 평행이동 하는 것이 목표이다. 결과물은 다음과 같다.

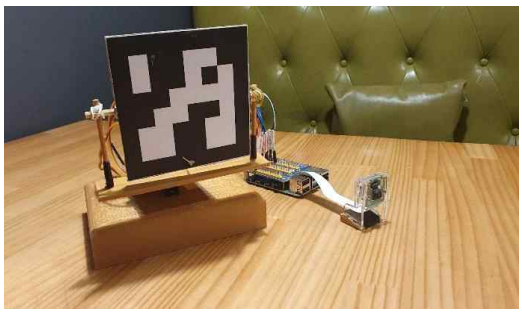


Fig. 42. 회전 모델의 조립 사진

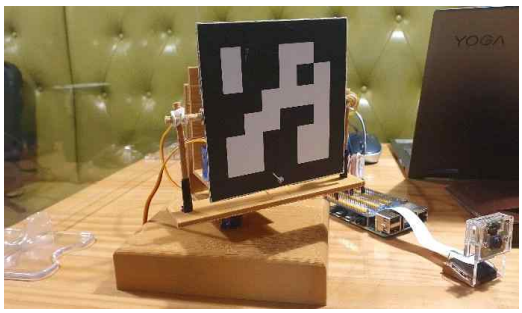


Fig. 43. 마커와 모델의 Yaw 45도 회전

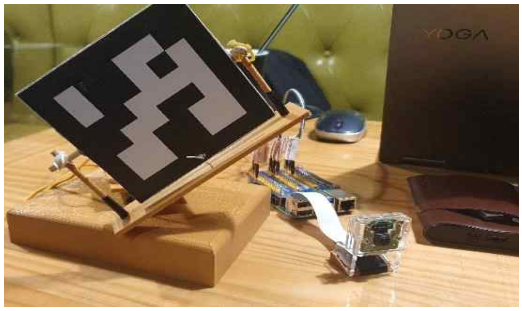


Fig. 44. 마커와 모델의 Roll 45도 회전

Fig. 43와 Fig. 44는 실제 세계에서 식별마커의 움직임을 따라 모델이 동작하는 모습이다. 하지만 프로토타입은 종이상자로 만들었지만 실제 모델은 3D프린터에서 사용하는 재료인 필라멘트 PLA로 제작하여 자체의 하중이 증가하였고 이 증가된 하중을 서보모터가 감당하지 못하여 프로토타입에선 발생하지 않았던 떨림이 발생하였다.

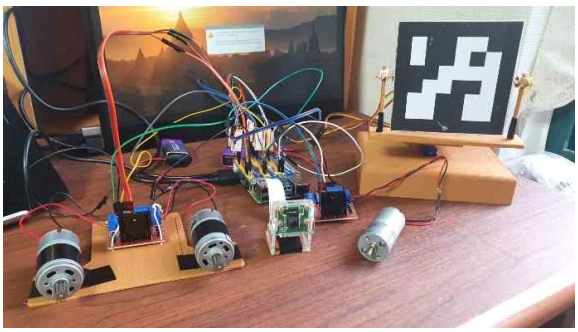


Fig. 45. 전체 모델 조립 사진

처음 예상했던 DC모터와 서보모터 전체를 조립한 사진이다. 하지만 처음에 목표했던 평행이동 모델 구축에는 도달하지 못하였다. 이유는 6. 결론에서 후술한다.

5.2 소요비용

Table 13. 설계에 들어간 소요비용

품명	수량	단가
스텐와이어 2.0mm 100M	1	28,000원
2중 구멍 커넥팅 로드 64502B 10개입 [SZH-GNP382]	1	1,500원
와이어로프 슬리브 M2.5 - 땅콩형 [SZH-WSD005]	10	80원
라즈베리파이용 GPIO E3P 확장보드 [SZH-AT037]	1	6,400원
라즈베리파이 카메라모듈 V2, 8MP (RPI 8MP CAMERA BOARD)	1	33,000원
[정품] SG90 360도 디지털 서보모터	4	3,600원
5V 학생실습용 DC모터 [SZH-MT001]	3	600원
2A L298 모터드라이버 모듈 (아두이노 호환) [SZH-EK001]	2	2,000원
메탈 컴파운드기어 3.85ø 15/55T [SZH-GNP179]	2	1,500원
RS390 피니언기어 DC모터 [SZH-GNP208-1]	2	12,800원
플라스틱 기어 78개입 세트 [SZH-GNP231]	1	6,900원
메탈 샤프트 3ø 150mm 10개입 [SZH-GNP339]	1	2,400원
소형 DC기어드모터 [SZH-GNP093]	1	14,000원
2GT6mm 타이밍 벨트 [SZH-BPM005]	1	4,500원
[리퍼제품]체인&스프라켓 키트 (Chain and sprocket kits)	1	1,000원
[리퍼제품] 아타치먼트 체인 (40-1 K1/2-35LINK)	1	1,000원
7인치 와이어 스트립퍼 (AWG20~30) [K11710]	1	3,700원
플라스틱 래크기어 125mm [SZH-GNP291]	6	700원
[리퍼제품]MSGB048E01	1	3,000원
비흐름성 순간접착제 AXIA 659 GEL, 20g	1	3,500원
6F22(9V,FC-1)Be	3	1,200원
Snap.I-3(for 9V)	3	230원
라즈베리파이4 (4GB) 스타터 키트 MicroSD 용량 선택:16GB	1	103,000원
합계 (VAT 포함)	49	299,489원

6. 결론

6.1 최종결과와 설계목표의 부합여부 분석

일반적으로 물체 검출에 있어서 수학적인 객체 검출 알고리즘 자체에 집중하거나 추가적인 장치 또는 식별마커 이외에 대상에 다른 표식 등을 추가하는 방법이 대부분이었다. 본 프로젝트에서는 식별마커 검출에 있어 다른 외적인 요소 대신 오로지 영상 후처리하는 방법으로 개선하는 쪽으로 접근하여 목표에 부합하는 결과를 얻을 수 있었다.

일반적으로 RGB 색상 공간의 영상보다는 GrayScale 등의 회색 조나 흑백 사진을 사용하는 것이 좋은 것으로 알려졌으나 실제 식별마커를

대상으로 실험한 결과 오히려 표준편차가 증가하는 결과를 얻었으며 단순히 HSV 색상 공간으로 나눈 V(Value)만 사용했을 때도 마찬가지로 큰 효과를 얻을 수 없었다. 하지만 임계값 적용 기법과 클래스 내 분산 값을 최소화하는 임계값을 결정해 주는 Otsu 알고리즘을 사용했을 때 유효한 결과를 얻을 수 있었다. RGB 색상공간 원본에 비해서 ToZero+Otsu를 적용했을 때 표준편차가 적게는 22.5% 감소하였으며 최대 42.6%까지 감소하기도 했다. 고성능의 카메라를 사용하거나 추가적인 장치 없이 영상의 후처리를 통해서 더 나은 결과를 얻을 수 있었다. 영상의 왜곡을 최소화하고 어떤 이미지 색상 공간을 선택하며 여기에 어떤 임계값 적용 해서 노이즈를 제거할 것인가는 더 나은 식별마커 검출에 있어 개선 방안이 될 수 있다.

모델 제작에 있어서는 목표에 달성하지 못하였으며 여러 복합적인 원인이 겹쳤다. 회전 모델에서는 Yaw, Pitch, Roll에 따라 서보모터에 회전값을 전해주는 것이 가능했지만 모델 제작에서 불안정하였다. 프로토타입 모델은 종이 상자와 플라스틱 더미를 이용하여 간략하게 만들었을때는 그 무게가 가벼워서 문제없이 작동하였으나, 프로토타입모델을 기반으로 도면 제작을 한 다음 3D 프린터를 이용해서 출력했을 때는 똑같은 크기와 형태를 지녔지만 다른 결과가 나왔다. 모델의 무게가 처음 보다 증가하면서 전체 무게중심과 서보모터에 걸리는 토크 힘이 달라지게 되는데 이는 의도하지 않는 방향에 힘이 실리게 되어 서보모터는 제자리로 돌아가려는 반대 힘을 반복적으로 주게 되었다. 이렇게 되자 전체 모델은 심한 진동 현상이 반복되어 결국 분리해서 사용할 수 밖에 없었다.

평행이동 모델 역시 마찬가지다. 처음에 계획했던 구상도와 달리 회전 모델의 크기는 커졌으며 이를 움직이기 위한 프레임 크기 역시 비례해서 커지게 되었는데 처음에 설계한 라즈베리파이의 DC모터 공급전력은 9V였지만 이를 만족할 만한 모터와 모터드 라이버를 찾지 못하였다. 더해서 모델의 크기에 따라 전선의 길이가 급격하게 증가하게 되었는데 제작에 사용한 전선은 반복적인 움직임에 취약하여 단선으로 인해 전체 동작의 안정성을 보장할 수 없었다. 결국 고정된 상태에서 간략한 동작만을 확인하였다.

6.2 향후 개선되어야 할 점

라즈베리파이는 주기적으로 제품을 출시하고 있으나 이번 실험에 사용한 라즈베리파이 모델4는 2019년도 생산품으로 빠르게 성장하는 반도체 성능과 비교하면 느린 축에 속하며, 실시간, 부드러운 이미지 처리를 하기에는 부족한 성능을 보였다.

Table 14. 라즈베리파이의 프레임 처리 성능

	HD	FHD
총프레임	619프레임	621프레임
처리시간	107.8초	219.1초
초당 프레임 처리	5.7 프레임	2.8 프레임

처음 계획에서는 영상의 input과 우리가 보는 output을 일치시켜 실시간성을 보장하려고 했으나, HD 기준 30프레임 유지가 힘들어 계획을 수정하였다. 설계한 시스템에서는 HD 기준 초당 5.7프레임을 처리할 수 있었으며 FHD에서는 1초당 2.8프레임을 처리할 수 있었다. 초당 프레임 처리를 높이기 위해서 SD로 화질을 더 떨어뜨리게 되면 영상 전체 시야각이 좁아지는 문제가 있기에 입력에 대한 출력의 실시간성 보장을 제외하였다. 최근 반도체 메모리와 소형컴퓨터 기술 발달이 빠르게 이루어지고 있는 가운데 차기 라즈베리파이나 소형 컴퓨터에서는 영상처리에 만족할 만한 성능을 기대한다. 아니면 가격대를 달리해서 고가의 엔비디아 Jetson을 사용한다면 고해상도 고 주사율의 영상 사용이 가능할 것으로 기대되며, 이번 실험에 사용한 단안카메라 대신 양안카메라를 사용한 이미지 처리가 가능하다면 사전에 크기를 정하지 않은 마커를 자유롭게 사용이 가능할 것이다.

이미지 후처리에 사용하지 못한 여러 기술들이 있다. 앞서 임계값 처리에서 ArucoMarker를 탐지할 때 광원 또는 음영에 취약한 부분을 보였는데 이 부분에 대해서 영상의 밝기 값을 조절하는 방법이 있다. 색온도 조절, 화이트 밸런스 조절, 히스토그램 평활화 등이 대표적인데 이를 이용하여 극단적으로 밝은 곳은 조금 어둡게 반대로 지나치게 어두운 곳은 조금 밝게 처리를 할 수 있다면 외부 광원에 대한 저항을 키워볼 수도 있을 것이다. 또는 밝기 값을 처리하기 보다 마커 자체에 추가적인 잉크를 칠하는 것도 고려해 볼 수 있다. 예를 들어 상업적으로 구매가 가능한 세상에서 가장 어두운 색상 도료 중 하나인 BLACK3.0의 경우 가시광선 흡수율이 99%에 달하는데 150ml 기준 원화 3만 원 정도면 구매가 가능하다. 이를 식별마커에 칠한다면 외부 광원에 대해서 더 강한 저항성을 가질 수 있을 것이라 본다.

이번 모델 제작에 있어서 여러 가지 이슈가 동시에 발생했다. 계획 변경으로 인한 일정 관리의 어려움이 있었다. 모델 제작으로 잡은 일정계획에 뒤늦게 주제 설정과 재검토에 들어가게 되

면서 전체적으로 일정이 밀리게 되었다. 그 다음으로는 프로토타입 모델과 실제 모델의 차이를 고려하지 못하였다. 단순히 프로토타입에서 작동되는 것을 기반으로 도면을 만들고 3D프린터로 제작을 하였으나 무게가 달라지면서 다른 동작 결과를 얻게 되었다. 다양한 형태의 도면을 제작하면서 만족할 만한 무게와 견고함을 확보하는 모델을 제작을 시도해본다.

7. 팀원 역할 및 기타

Table 15. 팀원 역할 및 기타

담당자	담당역할	보고서의 해당 부분
정대현	ArucoMarker의 특성 / 단안 카메라의 특징과, 카메라 캘리브레이션	1.2 기존 문제점 및 새로운 요구사항
정대현	라즈베리파이의 성능과, 효과적인 마커 검출 방법 / 마커 검출을 위한 엣지 탐색 방법	3.2.1 흑백이미지 사용 및 RGB 영상 후처리
박유경	물체 운동을 표현을 위한 6가지 표현 / 물체의 운동 이해	4.2.1 회전벡터의 각도 변환 추출
정대현	3차원의 물체를 2차원으로 이동하는 과정에서 왜곡 현상 / 카메라 캘리브레이션을 위한 보정 과정 학습	3.1 카메라 캘리브레이션을 이용한 왜곡 보정
박유경	SG-90 서보모터는 각도가 아니라 지정된 위치로 이동 / pigpio 라이브러리의 작동 원리	2.2.a 라즈베리파이 서보모터 구축
박유경	DC모터를 연결하기 위해서는 모터드라이브가 필요 / DC모터 드라이버 회로도 이해	2.2.b 라즈베리파이 DC모터 구축
정대현	영상 후처리에 따른 임계값 영상처리 기법	3.2.4 임곗값(threshold) 설정

참고문헌

[1] KR101863196B1 - 딥러닝 기반 표면 결함 검출장치 및 방법

[2] Minhua Ma; Lakhmi C. Jain; Paul Anderson (25 April 2014). Virtual, Augmented Reality and Serious Games for Healthcare 1. Springer Science & Business. ISBN 978-3-642-54816-1.

[2] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez. 2014. "Automatic generation and detection of highly reliable fiducial markers under occlusion". Pattern Recogn. 47, 6 (June 2014), 2280-2292. DOI=10.1016/j.patcog.2014.01.005

[3] Optical Flow-Based Marker Tracking Algorithm for Collaboration Between Drone and Ground Vehicle KIPS Tr. Software and Data Eng. Vol.7, No.3 pp.107~112 pISSN: 2287-5905

[4] Hanbury, Allan; Serra, Jean (December 2002). A 3D-polar Coordinate Colour Representation Suitable for Image Analysis. Pattern Recognition and Image Processing Group Technical Report 77. Vienna, Austria: Vienna University of Technology.

[5] J. W. Song, Content-based Image Retrieval Using HSV Color and Edge Orientation, Journal of Korean Institute of Information Technology, Vol. 16, No. 5, pp. 113-118, Nov. 2018.

[6] Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, pp. 111-114, ISBN 0-13-790395-2

[7] Canny, J., A Computational Approach To Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6):679-698, 1986

[8] Nalini Rizkyta Nusantika; Xiaoguang Hu; Jin Xiao (2021), Improvement Canny Edge Detection for the UAV Icing Monitoring of Transmission Line Icing IEEE ISBN 978-1-6654-2248-2

[9] Generalization of Otsu's binarization into recursive colour image segmentation by Acuna, R.G.G.1; Tao, J.2; Klette, R.2 In: International Conference Image and Vision Computing New Zealand, 2015 International Conference on Image and Vision Computing New Zealand, IVCNZ 2015.

(International Conference Image and Vision Computing New Zealand, 28 November 2016, 2016-November)

[10] RECURSIVE FUNCTIONS OF SYMBOLIC EXPRESSIONS AND THEIR COMPUTATION BY MACHINE (Part I) in Communications of the ACM in April 1960

[11] Guillermo Gallego and Anthony J. Yezzi. A compact formula for the derivative of a 3-d rotation in exponential coordinates. Journal of Mathematical Imaging and Vision, 51:378‑384, 2014.

[12] Novi Commentarii academiae scientiarum Petropolitanae 20, 1776, pp. 189‑207 (E478)

[13] Henderson, D. M. (1977-06-09). "Euler angles, quaternions, and transformation matrices for space shuttle analysis": 12‑24.

[14] Poynton, Charles A. "Rehabilitation of gamma." Photonics West'98 Electronic Imaging. International