2019 Spring ME492 Final Report

Team B
Soonho Yoon, 20110085
HaeKoo Jeon, 20120131
Junyoung Oh, 20140362
Sangbaek Yoo, 20160394

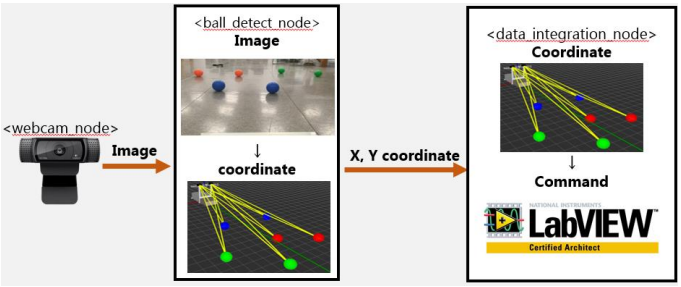
2019.06.16

Contents

1. System Overview

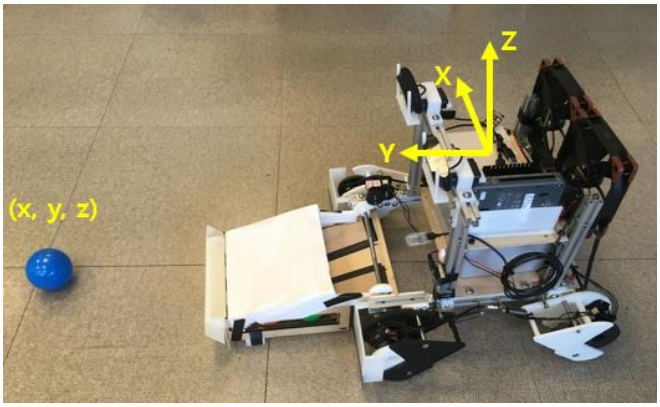
ROS와 통신을 주고받는 하드웨어는 크게 센서와 actuator가 있다. 우리 그룹은 센서로 웹캠 2대를 사용했고, actuator로 주행용 다이나믹셀 4개, 픽업용 다이나믹셀 1개를 사용했다.

웹캠 2대로는 서로 다른 시야를 확보하는데 사용한 것이 아니라, 스테레오 카메라를 구현하여 시스템 정면 시야를 센싱하되, 거리를 정확히 측정할 수 있도록 구성했다. 스테레오 카메라가 시스템으로부터 4m 정도 떨어져 있는 공의 좌표도 오차범위 5cm 이내로 측정하기 때문에 우리 시스템은 공의 좌표 데이터를 실시간 피드백 대신, open-loop 제어에 수차례의 피드백을 주는 형태로 알고리즘을 간소화할 수 있었다. 스테레오 카메라의 이미지 프로세싱은 webcam_node와 ball_detect_node에서 구현했고, 결과적으로 ball_detect_node는 차체에 고정된 좌표축을 기준으로 공의 (x,y)좌표를 출력한다.(Figure 1,2 참조)

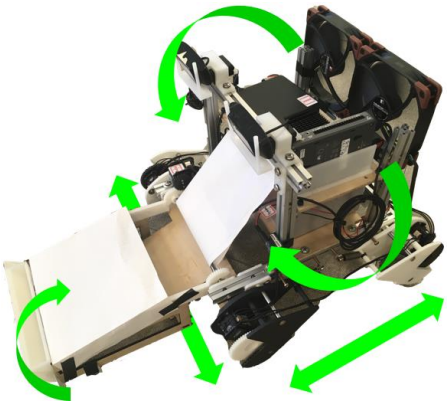


[Figure 1] Image processing steps

하지만 스테레오 카메라의 특성상 차체-카메라 간 진동보다 두 카메라 사이의 뒤틀림 진동에 더욱 민감하다. 시중의 스테레오 카메라는 하나의 body에 두 개의 카메라가 고정되어 있는 형태인 것에 비해 우리가 구현한 스테레오 카메라는 두 카메라를 완전히 고정하기가 어려워 하드웨어로 카메라 간 진동을 극복하기 어려웠다. 위에서 언급했듯 전반적인 알고리즘이 open-loop control에 몇차례의 피드백을 주는 형태이기 때문에, 공의 좌표를 피드백하는 순간에만 진동을 줄이면 문제가 없었다. 그래서 좌표를 측정이 필요한 순간에 잠시 정지하여 정확한 좌표를 측정하고, 그 좌표를 바탕으로 open-loop control하는 형태로 카메라 간 진동 문제를 극복했다.



[Figure 2] Body frame coordinate & coordinate of ball



[Figure 3] System Overview

스테레오 카메라로부터 받은 공의 좌표를 바탕으로 data_integration_node는 시스템을 공까지 정확히 움직이도록 랩뷰를 통해 다이나믹셀에 명령을 내려야 한다. 우리는 알고리즘을 간단하게 만들기 위해 다이나믹셀로 구현 가능한 주행모드 중 전진, 시계방향 회전, 반시계방향 회전만 사용했고, 마지막에 바구니에 정렬할 때만 좌, 우 방향 횡이동을 이용했다.(Figure 3 참조)

전진과 회전 함수 중 open-loop control을 하기 위해서 명령을 내리는 시간 대비 시스템이 실제로 전진하고 회전하는 변위를 측정했다. 이 변위는 바퀴의 속도뿐만 아니라 모터의 급작스러운 가속, 감속에 의한 미끄러짐 등 변수가 많아서 한두 차례의 실험으로 결정하기 어려웠기 때문에 여러 차례의 측정 결과를 이용했다. 측정한 값을 바탕으로 K1, K2에 각각 직진, 회전에 관한 비례상수를 할당하여 목표 거리와 각도에 따른 명령 길이를 조절할 수 있도록 했다.
pickup module을 구동하는 다이나믹셀의 경우 ROS에서 다이나믹셀의 각도를 피드백 받을 수가 없기 때문에 모든 구동이 open-loop control이다. Pickup module의 구동 각도는 파란 공의 pickup과 release에 아주 중요한 요소이기 때문에 여러 차례의 실험을 거쳐 파란 공을 채집하고 내려놓기 가장 적합하고 효율적인 구동 각도를 찾아 그 각도에 도달하는 TCP/IP 메시지 수를 구했고 그 값을 변수 t3와 release 함수에 적용했다.
파란 공을 채집하기 위한 path planning으로 우리 조는 탐욕법을 이용하여, 필드의 왼쪽부터 오른쪽으로 탐색하며 공이 보이는 대로 바로 채집하게 했다. 원거리에서 파악할 수 있는 공의 좌표는 부정확하고, 이동하면서 업데이트하는 최적 경로와의 충돌이 예상되어 미리 경로를 계획하기보다는 탐욕법을 채택했다.
전체적인 stage decision에 대한 알고리즘은 다음과 같다.
파란 공이 보이지 않으면 시계방향 회전, 보이면 파란 공이 시스템 정 가운데 올 때까지 회전 후 직진, 공이 사정거리에 들어오면 픽업 모듈을 들어올려 공을 채집한다. 채집 후엔 마찬가지로 회전 후 공 채집 과정을 반복하여 파란 공 3개를 채집한다. 파란 공 3개 채집을 완료한 후엔 바구니로부터 1m가량 떨어진 지점에 위치하여 바구니를 바라보도록 정확히 정렬한 뒤에 바구니를 향해 전진하여 공을 바구니에 담는다. 이 과정에서 경로 상에 끼어있는 빨간 공은 양면테이프를 이용해 차체 주변에 부착하여 미션 수행에 방해가 되지 않도록 제거했다.
위와 같은 알고리즘으로 미션을 수행하기 위해 회전, 전진, 픽업을 위한 함수들을 만들어 조건에 맞게 이용하도록 코드를 구성했다.

2. Functions and Variables

(1) 전진 동작 수행을 위한 함수[line 91~145]

linear(x,y) : 파란 공의 x, y 좌표를 받아 공으로부터 0.6m 떨어진 지점까지의 80%를 직진
linear_complete(x,y) : 파란 공의 좌표를 받아 공으로부터 0.6m 떨어진 지점까지 이동
linear_g(x_p, y_p) : 바구니로부터 1.2m 떨어진 지점 (x_p, y_p)의 좌표를 받아 그 지점까지 거리의 70% 전진
linear_g_complete(x_p, y_p) : 점 x_p, y_p)의 좌표를 받아 그 지점까지 이동
moveleft() : 바구니 중앙에 정렬하기 위해 아주 짧게(0.075초) 왼쪽으로 이동
moveright() : 바구니 중앙에 정렬하기 위해 아주 짧게(0.075초) 오른쪽으로 이동

(2) 회전 동작 수행을 위한 함수[line 148~229]

rotation(x,y) : x,y 좌표를 받아 해당 점과 이루는 각도를 계산하여 그만큼 회전
rotation_low(x,y) : 정확한 회전을 위해 점 x,y까지의 각도만큼 상대적으로 느린 속도로 회전
findballcw() : 실시간 피드백 회전을 위하여 단위시간(0.025초)만큼 시계방향으로 회전하는 단위 회전 함수
findballccw() : 단위시간(0.025초)만큼 반시계방향으로 회전하는 단위 회전 함수
findgballcw() : 초록 공을 탐색하기 위해 시계방향으로 25도 회전 후 초록 공의 좌표를 정확하게 측정하기 위해 0.3초 정지

findgballccw() : 반시계방향으로 25도 회전 후 0.3초 정지
CW(theta) : 필요한 회전 각도 theta를 받아 theta만큼 시계방향으로 회전하는 함수
CCW(theta) : theta만큼 반시계방향으로 회전하는 함수

(3) 그 외의 함수[line 234~264]

takerest(d) : time duration d * 단위시간 0.025초 만큼 동작을 정지하는 함수
pick(x,y) : t3*0.025초 동안 pickup frame을 들어 올리며 전진하고, 45cm 만큼 전진한 뒤, 다시 t3*0.025초 동안 전진하면서 pickup frame을 내려 x,y에 위치한 공을 채집하는 함수
release() : 바구니 앞에 주차한 뒤에 pickup frame을 완전히 들어 공을 내려놓는 함수

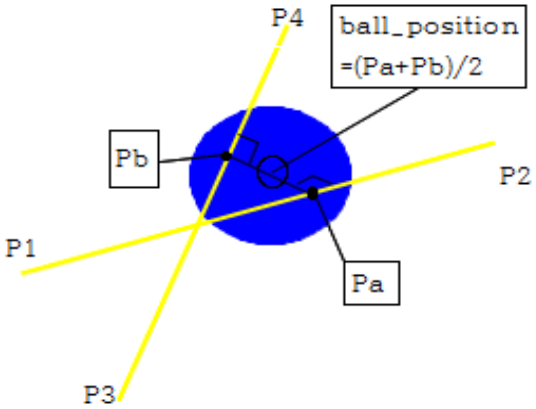
(4) 변수[line 33~88]

ball_X[ ], ball_Y[ ] : stage에 따라 msg에 담긴 파란 공 혹은 초록공의 x,y좌표를 저장하는 변수, 좌표값과 공이 없는 상태를 구분하기 위해 초기값 1000으로 초기화 한다.
data1[ ] ~ data12[ ] : Labview에 특정 동작을 지시하기 위한 TCP/IP 통신 메시지 형태
rest : 바구니를 탐색하기 위해 초록공이 시야에 들어왔을 때, 그 좌표를 정확히 받기 위한 정지 동작을 초록공을 처음 발견했을 때 1회만 수행하도록 기준값을 저장하는 변수
rotatbasket : 바구니에서 1.2m 떨어진 지점에 도달했을 때, 바구니 중심까지 rotation_low를 처음 1회만 실시하도록 하기 위한 기준값을 저장하는 변수
d : 시스템이 파란 공을 채집하기 위해 pick함수를 실행하기 위한 최소 거리.
theta : 회전운동을 위한 함수에서 계산된 각도를 저장하기 위한 변수
t1 : 선형운동을 위한 함수에서 계산한 목표 거리에 따른 직진 명령 길이를 저장하는 변수
t2 : 회전 함수에서 theta와 회전비례상수를 곱하여 구한 회전 명령의 길이를 저장하는 변수
t3 : pick 함수에서 pickup frame을 파란 공을 채집하기에 적합한 각도로 들어 올리기 위한 명령의 길이
t4 : pick함수에서 pickup frame을 들고 파란 공이 frame 바로 아래에 위치하여 frame을 내리는 동작을 수행하기 직전까지 open-loop로 직진하기 위해 계산된 명령 길이
k1 : 1m 직진하는데 필요한 직진 명령 개수. 선형비례상수로 명명하고 사용했다.
k2 : 180deg 회전하는데 필요한 회전운동 명령 개수. 회전비례상수라고 명명하고 사용했다.
cb : stage decision을 위해 채집한 파란 공의 개수를 저장하는 변수
cg : 파란 공을 전부 채집한 뒤, 바구니 앞 1.2m 지점에 정렬하는 동작을 수행했는지 여부를 저장하는 변수. 이 변수가 없으면 1.2m 지점에 정렬한 뒤에도 바구니로 직진하는 동작을 하지 않고 계속 1.2m 지점에만 머무르게 될 것이다.
c80 : 파란 공을 발견하고 채집할 때, 파란 공으로부터 거리 d만큼 떨어진 지점까지 도달하기 위해 피드백을 1번만 수행하기 위해 피드백 여부를 저장하는 변수
c100 : 점(x_p, y_p)까지 도달하는데 피드백을 1번만 수행하기 위해 피드백 여부를 저장하는 변수
count : ball_detect_node에서 받은 메시지에 담긴 공의 개수를 저장하는 변수
theta_g : 차체 고정 좌표계 기준 바구니가 기울어진 각도를 저장하는 변수
xm, ym : 시야에 들어온 파란 공들 중 가장 왼쪽에 있는 공의 좌표를 저장하는 변수. 이 좌표를 기준으로 파란 공 pickup 동작을 수행한다.
x_mid, y_mid : 두 개의 초록 공 좌표를 이용해 바구니의 중심좌표를 계산하여 저장한 변수
x_p, y_p : 바구니로부터 1.2m 떨어진 지점을 계산하여 저장한 변수
gmin_x, gmin_y, gmax_x, gmax_y : 두 개의 초록색 공 중 왼쪽에 있는 공과 오른쪽에 있는 공을

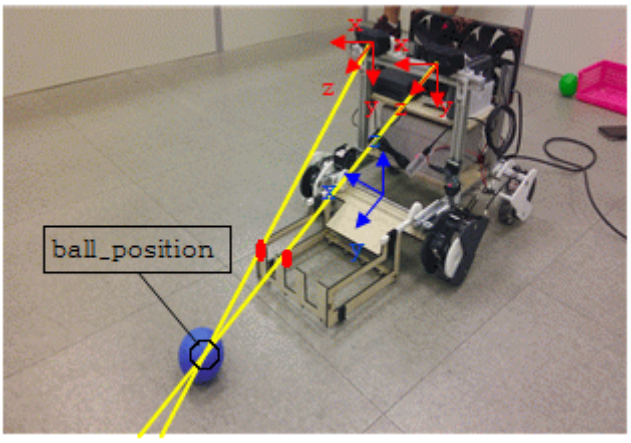
구분하기 위한 변수, min이 왼쪽에 있는 공, max가 오른쪽에 있는 공의 좌표를 저장하는 변수이다.

3. ball_detect node

(1) stereo camera의 개념 설명 및 구현 과정



[Figure 4]



[Figure 5]

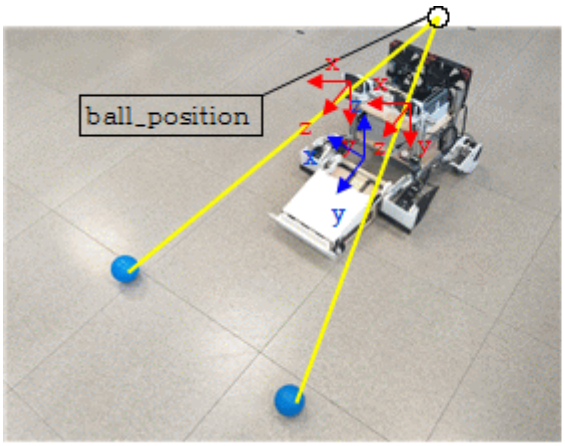
우리 조는 webcam 2대를 이용하여 stereo camera를 구현했다. 기존의 webcam_node와 ball_detect node를 수정하여 stereo camera 코드를 짰다. 기존의 webcam_node는 카메라 1대에서 받은 image를 ball_detect_node에 publish한다. Stereo camera 알고리즘에서는 같은 시점에 2대의 카메라에서 받은 이미지를 동시에 처리해야한다. 먼저 videocapture 함수를 사용하여 2대의 카메라에서 받은 이미지를 각각의 matrix로 바꾼다. 그리고 opencv의 hconcat 함수를 이용하여 2개의 matrix를 나란히 붙여 1개의 matrix로 바꾼다. 두 이미지를 합친 1개의 matrix를 ball_detct에 publish하고 ball_detect node에서는 subscribe한 matrix를 다시 기존의 2개의 matrix로 바꿔 stereo camera 알고리즘에 사용했다.

Ball_detect node에는 webcam_node에서 받은 이미지를 바탕으로 stereo image processing 코드를 짰다. 스테레오 카메라의 장점은 먼 거리에 있는 공의 좌표도 작은 오차범위 안에서 정확히 측정되는 것이다. 카메라 1대로 공의 좌표를 측정했을 때엔 공의 좌표가 방향성은 맞지만 공의 거리가 실제보다 작게 측정되었다. 하지만 stereo camera를 이용했을 땐 실제 공의 좌표를 3m 거리에서도 오차범위 5cm 내로 구할 수 있었다. Stereo camera의 기본적인 원리는 <Figure 4>과 같다. <Figure 4>의 빨간 점이 실제 카메라에서 보는 공의 좌표이다. 각각의 카메라의 원점과 빨간 점을 연결하여 직선을 만든다. <Figure 5>처럼 직선 2개의 사이의 중점 (Pa+Pb)/2가 ball_position이 된다. ball_detect.cpp의 Line 109~112는 right camera와 left camera의 intrinsic parameter와 distortion_data 값으로 각각의 카메라로 보았을 때 각각의 카메라 좌표계에서 보았을 때의 x, y, z 값을 구할 때 쓰인다. Line 114~115는 두 camera의 좌표계 사이의 rotation matrix와 translation matrix이다. 왼쪽 카메라를 기준으로 하였으므로 왼쪽 카메라 좌표계에 rotation matrix를 곱하고 translation matrix를 더하면 오른쪽 카메라 좌표계가 나온다. Stereo calibration을 통해 Left camera와 Right camera의 intrinsic data,

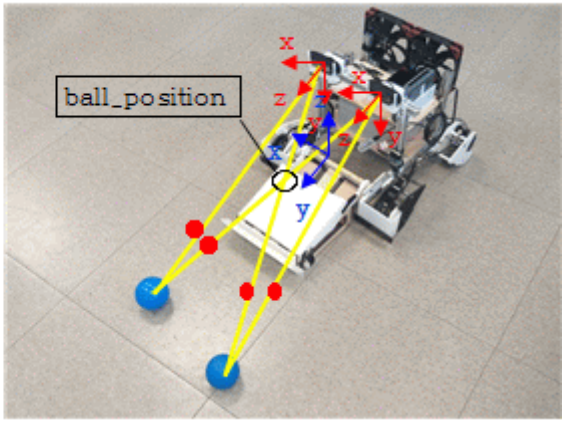
distortion data 그리고 rotation과 translation matrix 값을 알 수 있다. Line 147~178은 pixel2point_1, pixel2point_2라는 함수를 만들었다. 카메라의 이미지상의 pixel 좌표와 반지름값을 넣어주면 intrinsic data를 이용하여 각각의 카메라 좌표계에서 본 x, y, z로 바꾸어주는 함수이다. Line 353~760은 왼쪽 카메라에서 red, blue, green ball을 인식하도록 image를 processing 해서 왼쪽 카메라 좌표계에서 본 좌표를 얻는 과정이다. 이미지를 HSV형태로 바꾼다. 그리고 red, blue, green에 대한 HSV값에서 +-10 정도 범위를 주어서 이미지 상에서 그 pixel 값만 인식하도록 한 뒤 findcontour 함수를 이용하여 contour를 그리고 approxPolyDP 함수를 써서 contour를 다각형으로 바꾼다. 마지막으로 minenclosing함수를 써서 다각형 안에 내접원을 그려서 이미지상에서 공의 중심과 반지름값을 구할 수 있다. pixel2point_1 함수를 이용하여 구한 원의 중심과 반지름을 넣어 왼쪽 카메라 좌표계에서 본 좌표를 구한다. 이 과정 중에 광원을 없애는 code도 짰다. Line 481~507로 이는 red ball에 관한 경우인데 원 2개가 그려졌을 때 2개의 중심점 사이의 거리가 2개의 반지름의 합보다 작다면 원 2개가 겹쳐있는 경우이다. 따라서 앞의 조건을 만족하면 반지름이 더 작은 광원의 원을 지웠다. Line 761~1116은 오른쪽 카메라에 대해서 위와 같은 방식으로 공의 좌표를 구하는 코드이다. Line 1119~1830은 스테레오 카메라 알고리즘을 구현한 코드로, 두 개의 이미지를 이용하여 차체 좌표계에서의 공의 좌표를 구한다. 각각의 카메라 좌표계에서 본 좌표들과 오른쪽 카메라의 원점을 왼쪽 카메라의 좌표계으로 본 좌표로 바꾼다. 이는 rotation matrix와 translation matrix를 이용하여 구할 수 있다. 그리고 그 점들을 카메라의 원점들과 연결해 각각의 카메라마다 직선의 방정식이 나온다. 예를 들어 왼쪽 카메라에 3개 오른쪽 카메라에 3개의 직선이 나왔다고 하면 이는 왼쪽 카메라에서 공이 3개가 보이고 오른쪽 카메라에서 3개가 보이는 것이다. 공의 좌표를 구하려면 < Figure 5>처럼 직선을 2개가 짝을 이루어야 한다. 하지만 어느 직선이 짝이 맞는지 모른다. 따라서 for 문을 이용하여 왼쪽 카메라의 직선 1개를 오른쪽 카메라의 직선 3개와 모두 비교하여 거리가 가장 짧은 것과 짝을 이루도록 코드를 만들었다.

매칭시킨 2개의 직선을 이용하여 왼쪽 카메라 좌표계상에서의 공의 좌표 x,y,z를 구한다. 구한 공의 x,y,z를 차체 기준 좌표계에서 본 좌표로 바꾸어야 한다. 따라서 왼쪽 카메라 좌표계와 차체 기준 좌표계 사이의 rotation matrix와 translation matrix를 알아야한다. Rotation matrix는 왼쪽 카메라 좌표계의 x축을 회전시킨 값이고 translation matrix는 x축으로는 2개의 카메라 사이의 반절거리 만큼 움직이고 z축으로 30cm 만큼 움직인 값이다. 2개의 카메라사이 거리는 stereo calibration을 통해 구하였고 30cm는 자로 재었다. x축을 회전시킨 값은 먼저 임으로 회전 값을 정한 뒤 공2개를 놓고 좌표를 보았을 때 z값이 똑같아질 때가지 회전 값을 바꾸어 주었다. 회전 값이 맞지 않으면 축이 틀어지는 이유는 거리가 멀수록 z축이 비례하게 증가하거나 감소하기 때문이다.

(2) stereo camera를 코드로 구현 시 예외 상황



[Figure 6-1]



[Figure 6-2]

몇 가지 예외 상황이 있다. 예를 들어 <Figure 6-1>과 같이 왼쪽 카메라에 공 1개가 보이고 오른쪽 카메라에 공이 1개가 보이는 상황이다. 직선을 만들어서 짝을 만들었는데 공의 좌표가 차체 뒤에 만들어진다. 이때의 공의 좌표는 왼쪽 카메라 좌표계 기준 z좌표가 음수가 된다. 직선과 직선이 짝을 만들었을 때 z좌표가 음수가 나오게 되면 이는 두 직선 사이의 거리가 다른 직선과 짝을 이루었을 때의 거리보다 짧아도 짝이 아니라고 인식하도록 코드를 구성했다. 또다른 예외로는 <Figure 6-2>와 같은 경우도 있다. 왼쪽 카메라에 공이 2개가 보이고 오른쪽 카메라에도 공이 2개가 보이는 상황이다. 직선 2개가 짝을 이루었는데 x자로 짝이 된 경우이다. 이 말은 즉 공에 맺히게 직선들이 짝을 이루었을 때보다 x자로 하였을 때 두 직선 사이의 거리가 더 짧다는 것이다. 이러한 경우를 방지하기 위해 2개의 직선을 매칭시켰을 때 나온 공의 좌표를 이용하여 ball과 카메라사이의 거리가 한 대의 카메라로 보았을 때의 공의 좌표를 이용해서 구한 ball과 카메라사이의 거리보다 짧으면 두 직선은 매칭이 아니라고 코드를 짰다. <그림 4>에서 보이듯이 카메라 빨간 점은 카메라 한 대로 본 공의 좌표인데 x자로 매칭돼서 나온 공의 좌표보다 거리가 더 긴 것을 볼 수 있다. Line 1831~1989는 공의 좌표를 rviz상에서 볼 수 있도록 msg를 보냈다. 실제로 차체를 구동한 뒤 rosbag record -a 함수를 이용하여 rosbag 파일을 얻어서 rviz상에 볼 수 있었다. 찍힌 좌표들이 지워지지 않고 남아서 좌표값이 튀는지 확인할 수 있었다. Line 1995~2125는 이미지는 subscribe하고 ball 좌표들을 publish 하도록 코드를 짠 부분이다.

4. data_integration_node

(1) Prior knowledge

a. data가 의미하는 것(line44~55)

tcp/ip 통신을 통해 myRIO에게 보내주는 데이터 배열에는 3가지의 원소가 존재한다.

<1번째 원소 - 선형 운동>

0 : 정지

1 : 직진 운동 명령

2 : 오른쪽 선형 운동 명령 / -2 : 왼쪽 선형 운동 명령

<2번째 원소 - 회전 운동>

0: 정지

6 : 시계 방향 회전 운동 명령(느린 속력) / -6 : 반시계 방향 회전 운동 명령(느린 속력)

10 : 시계 방향 회전 운동 명령(빠른 속력) / -10 : 반시계 방향 회전 운동 명령(빠른 속력)

회전 운동에서 6과 10은 속력의 크기를 표현한다. 이 값에 비례하게 바퀴가 실제로 회전할 것이다.

<3번째 원소 - 픽업 프레임 움직임>

0: 정지

1 : 픽업프레임을 올리는 명령 / -1: 픽업프레임을 내리는 명령

이 3가지 데이터들을 서로 조합할 수 있고, 선형으로 합해진 운동을 가능하게 할 것이다.

b. 기타 함수 설명

-모든 과정에서 tcp/ip 통신을 통해 myRIO로 데이터를 보낼 때는
write(c_socket, dataN, sizeof(dataN)); (첫번째 인자는 정의했던 소켓, 두번째 인자는 보내줄 데이터, 세번째 인자는 보내줄 데이터에 중에서 보낼 길이, 일반적으로 데이터 전체를 보내줄 때는 그대로 입력하면된다.)
함수를 사용한다.

-좌표들을 받을 때, y좌표값에서 0.02를 빼준 이유는 캠을 기준으로 실제 거리와 측정된 거리와의 오차가 0.02만큼 존재하였기 때문이다.

callback 함수가 돌아가는 동안 새로운 데이터를 받아 callback operation 도중 새로운 msg를 subscribe해서 기존의 callback이 중도 정지되는 상황을 막기 위해 map_mutex.lock()을 걸어준다.(line 269)

callback 함수 안에 매번 해당되는 프로세스가 완료되면 map_mutex.unlock()를 실행시켜 다음 callback함수가 실행되게 한다.(line 488)

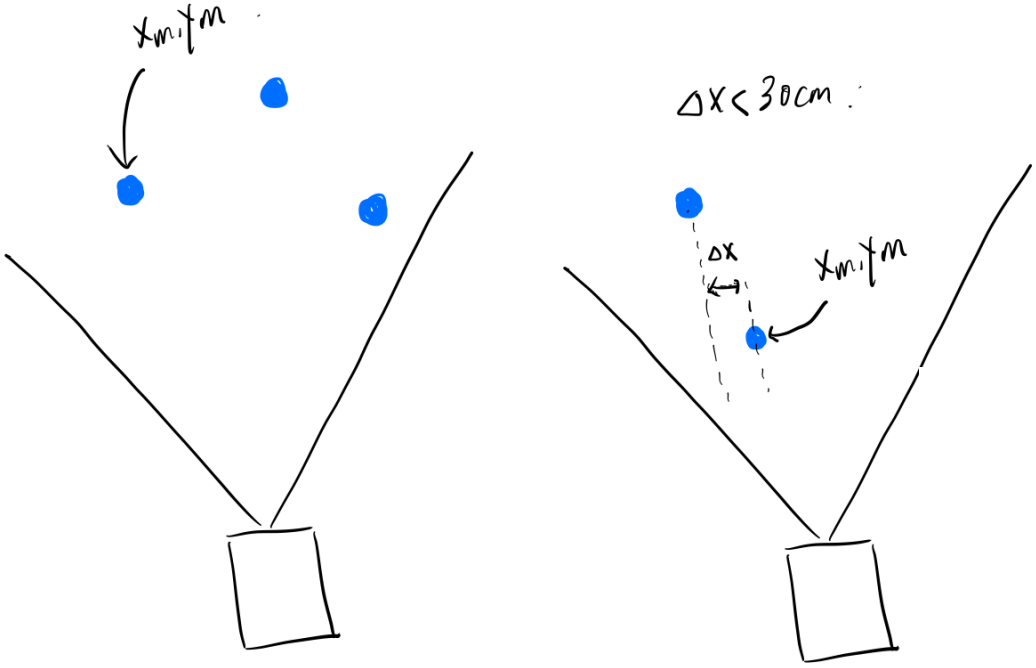
(2) 알고리즘 설명

a. Pick-up Process(line 291~346)

파란공 3개를 줍는 과정이다. 파란 공을 이전까지 주운 횟수를 cb에 저장하고, cb<3일 경우에 파란 공을 줍는 과정에 머무른다. 가장 먼저 카메라에서 받은 파란 공 개수와 좌표들을 정수 count, 배열 blue_x[], blue_y[]에 입력할 것이다. 그 전에 문제가 생기지 않기 위해 blue_x[], blue_y[]에 디폴트 값들을 넣어놓고 항상 시작한다.(line286~289) 공이 하나도 보이지 않을 경우(받은 blue ball count가 0일 경우, line293~296), 우리의 알고리즘에서는 공을

시계방향으로 주워나갈 것이기 때문에 공이 보일 때까지 findballcw() 함수를 통해 시계방향으로 회전하게 한다.
그렇게 해서 공이 보일 경우(count 가 0이 아닐 경우, line297~345), 화면 상에서 파란 공이 여러개 존재할 수 있지만, 이 중에서 우리는 가장 왼쪽에 있는 공을 주울 것이기 때문에 min 을 찾는 알고리즘(line 298~310)을 이용하여 가장 왼쪽에 있는 파란공의 좌표를 xm, ym 에 저장한다. 여기서 다시 두가지로 나뉜다.

i) 가장 왼쪽에 있는 공을 주위로, 30cm 안쪽에 다른 공이 존재할 경우, 특히 이 공의 y 좌표가 가장 왼쪽에 있는 공의 ym 좌표보다 짧다면, 두번째 공을 우선으로 주워야 한다. 만일 그렇지 않다면, 차체가 직진하면서 공을 쳐내서 공의 좌표가 바뀌거나 픽업프레임에 붙어버릴 수 있다. 따라서 문제를 해결하기 위해 이러한 경우가 존재할 때 xm, ym 을 수정하여 두번째 공을 우선순위로 줍게 만든다.(line312~320)



[Figure 7] Exceptional case

ii) 가장 왼쪽에 있는 공을 주위로 공이 존재하지 않는다면 xm, ym 을 그대로 유지한다.

이제부터 공을 줍는 동작을 시킬 것이다. 만일 xm 의 좌표가 카메라의 원점에서 오차 5cm 밖에 존재할 경우, 공의 좌표를 받아와 카메라의 중심으로 정렬될 수 있게 그만큼의 회전 명령 rotation(xm,ym)을 한다.(line 340~344)
이것을 중앙정렬이라 하자. 만일 중앙정렬을 통해 xm 의 좌표가 오차 5cm 안에서 존재할 경우(line323~338), 직진 운동을 시킬 것이다. 우리는 여기서 파란공을 향해 완전히 움직이는 것이 아니라, 공을 주울 최소한의 확보거리 d(=60cm)를 빼고, 거기서 한번의 피드백을 거쳐서 선형운동을 할 것이다. c80(피드백 횟수)가 0일 때, 피드백을 한번도 안한 것으로 판단한다.(line 324~329) 이 안에서 카메라에서 파란공의 거리가 D 로 측정되었을 경우 여기서 d 를 뺀 (D-d)의 거리를 80% 이동하고(linear(xm,ym)) c80(피드백 횟수)의 크기를 1 증가시킨다. 그렇게 c80이 1이 되면 피드백이 이전에 한번 되었다는 것을 판단하고(line 330~337) 다시

파란공의 좌표를 받아 중앙정렬을 시켜 (D-d)의 거리를 완전히
이동한다(linear_complete(xm,ym)).
이후 파란공과의 차체 사이의 남은 d의 거리에서 픽업을 하는 과정을 실행(pick(xm,ym))한다.
픽업 과정은 다음과 같다.
pick(xm,ym)에 cb를 1 증가시키고 새로운 공을 탐색하기 때문에 이전에 변경되었던 c80을 0으로 초기화 해준다.(line336) 이 모든 과정이 파란공 하나를 집는 과정이다. 이렇게 파란공3개를 집을 때까지 같은 프로세스를 반복한다.

b. Release Process(line 347-482)

파란 공 3개를 모두 다 수집한 후 바구니에 배출까지 우리가 사용할 알고리즘은 다음과 같다.

i) 초록 공 두 개 보일 때까지 회전

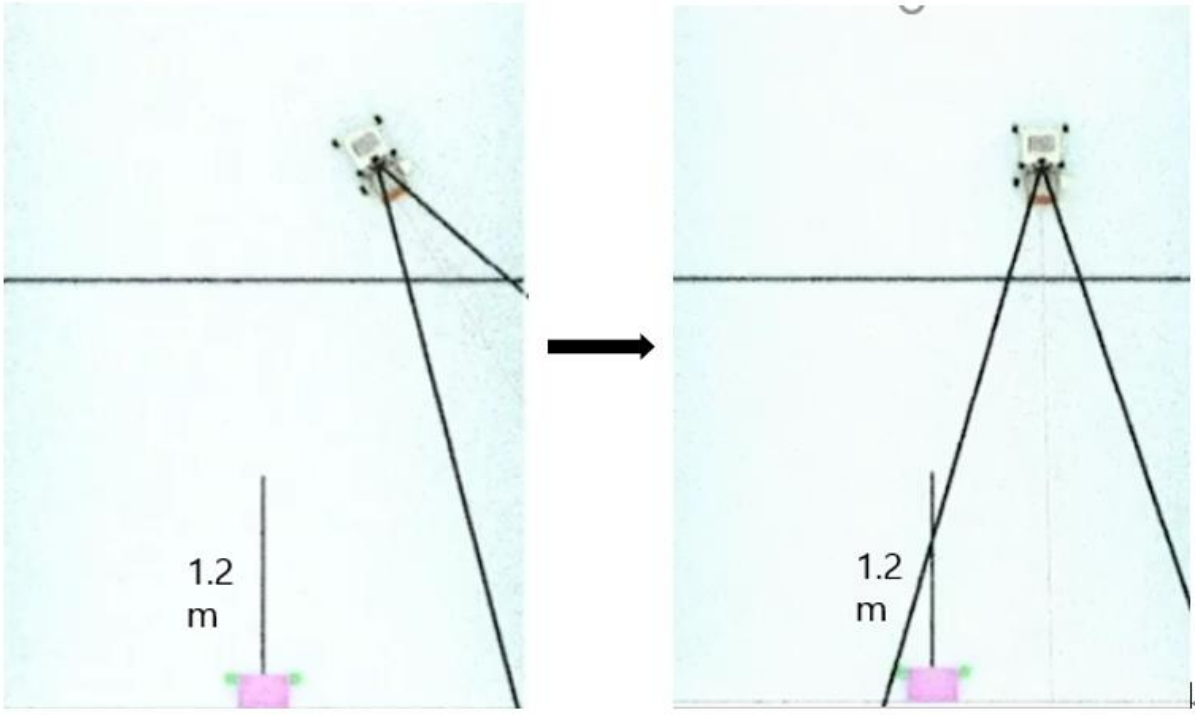
ii) 바구니 중심으로부터 1.2m 지점까지 70% 이동+중심을 보게 회전

iii) 바구니 중심으로부터 1.2m 지점까지 이동+중심을 보게 회전

iv) 공 배출을 위한 차체 정렬

v) 직진 후 배출

i) 초록 공 두 개 보일 때까지 회전(line 347~367)



[Figure 8] Rotating until two green balls are detected

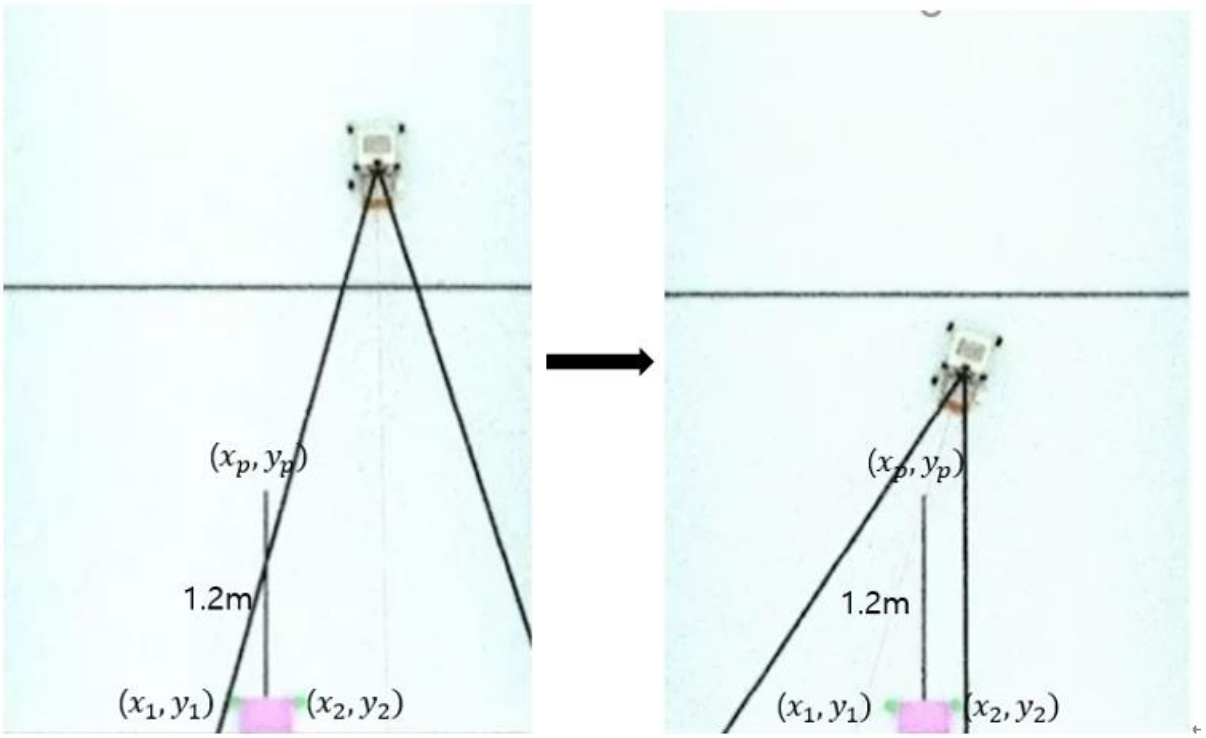
파란 공 3개를 다 수집 후 배출을 위한 바구니 정렬은 양 끝에 있는 초록 공 두 개의 좌표를 이용하여서 하게 된다. 이 초록 공들의 좌표를 array g_img_x와 g_img_y에 저장하고, 보이는 초록 공의 개수 (g_size) 값을 count에 저장해준다.

cb=3, 즉 파란 공 3개를 모두 수집 완료한 후 바구니 위치를 파악하기 위해서는 초록 공 두 개가 카메라에 잡혀야 한다. 이를 위해 count 값이 2가 아닐 경우 findgballcw()를 이용해 차체를 회전시킨다. findgballcw()함수의 내용을 보면 (line 188) 시계방향으로 회전시키는 data5와 아무 행동을 하지 않는 data8이 같이 존재하는 것을 확인할 수 있다. 즉, 일정 각도만큼 회전하고 잠시 멈추는 것을 하나의 function으로 같이 정의하였는데 이는 stereo 카메라의 특성상 차체의 미세한 진동에도 큰 영향을 받아 회전 중 초록 공이 시야에 잡히더라도 detect를 하지 못하는 경우가 많았기 때문이다. 한 번에 너무 큰 각도로 회전하게 되면 초록 공 2개가 모두 잡히는 경우를 그냥 넘겨버릴 수 있고, 너무 작은 각도로 회전하면 멈추는 시간이 길어지기 때문에 최적의 값을 찾아야 하는데 여러 번의 실험 결과 회전의 TCP/IP message의 길이를 20으로 결정하게 되었다.

초록 공이 두 개 다 보이는 순간 if(count!=2) 조건문에서 빠져나오게 되는데(line 358) 이 경우 정확한 초록 공의 좌표를 받기 위해서 takerest(12)를 추가로 주었다. Count 값이 2가 된 직후 딱 한 번만 하기 위하여 rest라는 int를 정의하였고 초기값을 0으로 설정하였다. rest==0일 때만 takerest(12)가 추가로 실행되게 하였고 이후 rest=1로 바꾸어 줌으로써 이 loop이 한 번만 진행되도록 하였다.

이후 두 개의 초록 공의 좌표를 통해 그 중점을 계산하고 x_mid와 y_mid로 정의해준다. (line 363)

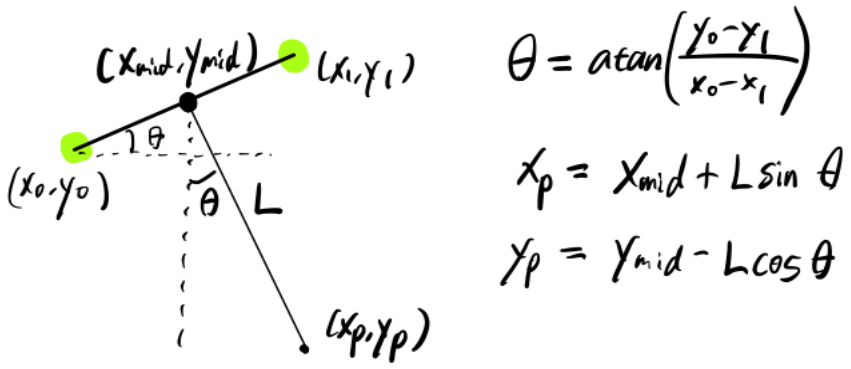
ii) 바구니 중심으로부터 1.2m 지점까지 70% 이동+중심을 보게 회전(line 368~399)



[Figure 9] Moving to (xp,yp)+rotation, first feedback process

바구니 중심으로부터 1.2m 떨어진 지점의 좌표는 아래 그림과 같이 두 초록 공의 좌표값들로부터 구할 수 있다.

이 좌표의 x 값과 y 값을 각각 x_p 및 y_p로 정의해준다. (line 370~372)

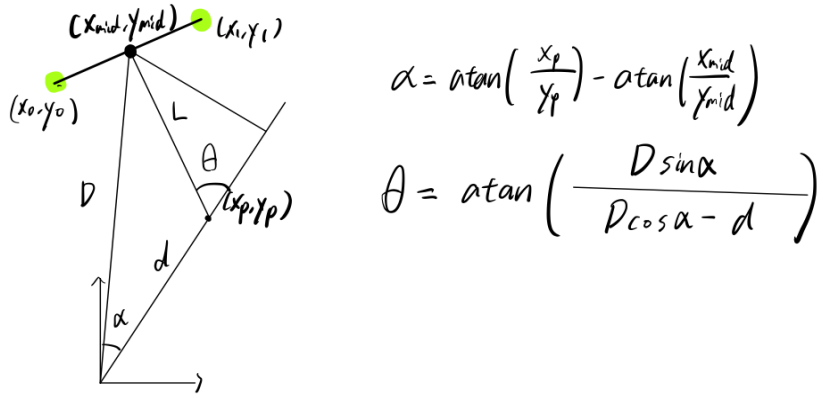


[Figure 10] Calculation for (xp,yp)

여기서 L=1.2m이다.

직진하기 전, (x_p,y_p)로 이동하기 위하여 이 점을 바라보게 회전하는 함수인 rotation(x_p,y_p)를 실행시킨다. 그 후, 지정된 좌표의 70%를 이동하는 함수 linear_g를 이용하여 전진하게 한다.(line 377~380)

전진 후 차체가 다시 초록 공 2개를 바라보게 해야 하는데 이 각도 값은 다음과 같이 구하였다.

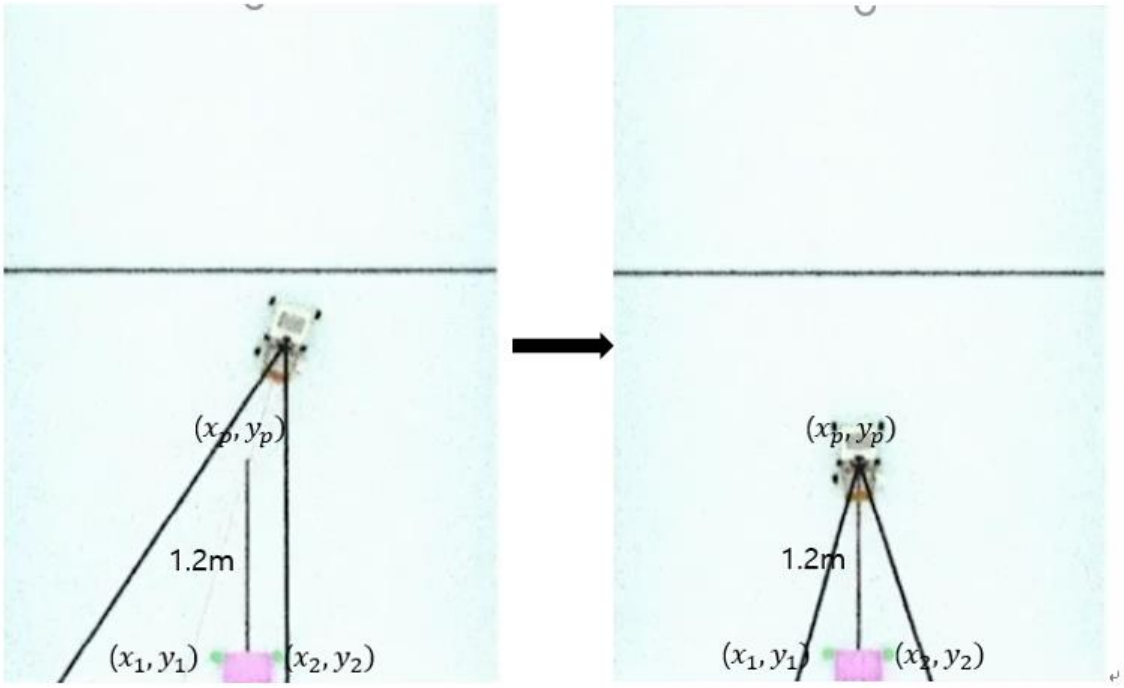


[Figure 11] Calculation for theta

여기서 $D = \sqrt{x_{mid}^2 + y_{mid}^2}$ 이고 θ 값을 theta_c로 정의하였다.

theta_c가 음수인 경우 마이너스를 취해줘서 크기 값으로 변환하였다. (line 385) x_mid와 x_p 값을 비교하여 x_mid가 작은 경우 지정된 각도만큼 반시계방향으로 회전하는 CCW 함수를 통해 theta_c만큼 회전하게 하고, x_mid가 더 큰 경우 시계방향으로 회전하는 CW 함수를 이용하였다. (line 390~395) 70% 이동 및 회전의 feedback을 한 번만 실행시키기 위해 c100이라는 int를 정의하였고, 회전 후 c100++을 실행시킴으로써 위 feedback이 실행되어야 할 조건 (c100==0)에 위반되도록 하였다.

iii) 바구니 중심으로부터 1.2m 지점까지 이동+중심을 보게 회전 (line 400~420)



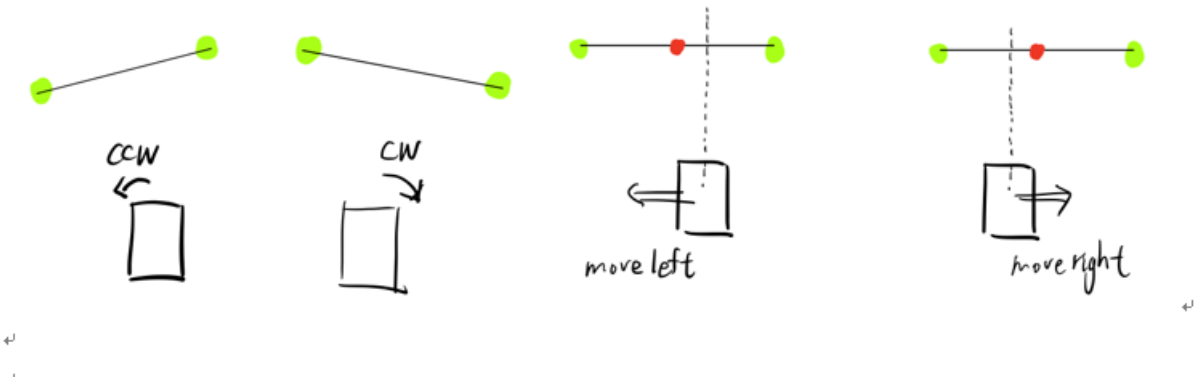
[Figure 12] Moving to (xp,yp) & rotation

앞에서 theta_c만큼 회전함으로써 차체는 초록 공 2개를 detect하고 있는 상황이다. 마찬가지로 x_p 및 y_p 값을 계산하여 rotation 함수를 통하여 (x_p,y_p)를 바라보게 하고, 이번에는 지정된 좌표의 70%가 아닌 100% 전진하는 linear_g_complete 함수를 통하여 (x_p,y_p)에 도달하도록 한다. 여기서도 앞선 과정과 같이 theta_c를 계산하고 회전시킴으로써 초록 공 2개를 다시 detect 할 수 있도록 만들어준다.

iv) 공 배출을 위한 차체 정렬(line423~467)

cg=1인 상황이다. 이때부터는 바구니로 정확히 도달할 수 있도록 차체를 정렬하는 과정이다. 그 전에 두 초록공을 향해 회전하는 과정(cg=0)에서 오차가 존재할 수 있으므로 rotatbasket 이라는 변수를 이용하여 정렬하기 전(rotatbasket=0)에 두 초록공의 중점 좌표를 향해 평소보다 느린 속력으로 중앙정렬을한다.(이때부터는 정확한 측정이 요구되기 때문에 느린 속력으로 회전하는 함수 rotation_low(x_mid, y_mid)를 사용한다.) 그리고 rotatbasket 을 1로 바꾸어준다. (line423~426)
이후(rotatbasket=1) 두 초록공의 좌표를 이용하여 정렬을 시작한다. 우선, 두 공의 x 좌표를 이용하여 왼쪽에 있는 공과 오른쪽에 있는 공을 gmin, gmax 로 정의해서 각각의 x ,y 좌표를 받은 데이터를 통해 저장한다.(line 428~439) 정렬에서는 수평정렬(차체가 바구니에 대해 기울어져 있는 상태를 정렬), 중앙정렬(두 초록공의 중점 좌표를 향해 중앙정렬)이 필요한데, 수평정렬을 먼저 시작한다.(line443~456) 어긋나 있는 상태의 유무를 두 초록공의 y 좌표를 이용하여 알 수 있는데, y 좌표의 차이가 1.5cm 이상일 경우, 어긋나있는 것으로 간주하여 회전운동을 하여 수평을 맞춘다.

이를 완료하면 이제 중앙정렬을 한다.(line457~467) 중앙 정렬은 두 공의 중점좌표에서 x_mid 좌표를 이용하여 할 수 있다. 원점에서 5cm 밖에 존재할 경우, 방향에 따라 수평운동을 통해 정렬한다.(x_mid>5cm=>moveright()*5, x_mid<-5cm=>moveleft()*7, 이렇게 두 함수의 길이에 차이를 둠으로써 무한 루프에 빠지지 않게 할 수 있다.) 이렇게 중앙정렬까지 모두 맞추면 바구니를 향해 직진 운동할 준비가 완료되었다.



[Figure 13] Perpendicular and lateral alignment

v) 직진 후 배출(line468~477)

바구니를 향해 직진 운동하기 전에, 바구니에 있는 턱에 걸리지 않기 위해 픽업 프레임을 공이 나가지 않을 정도로 data[1]*12을 보내주어 약간 올린다.(line 469~472) 이후 픽업 프레임의 길이를 뺀 길이만큼 linear_g_complete(x_mid,y_mid-0.13)을 통해 직진 운동을 한다. 그리고 release()를 통해 최종적으로 픽업프레임을 완전히 올려 공이 바구니로 release 되게 한다.
이렇게 모든 과정이 완료되면, release()에 cb 를 1 증가시키는 과정이 담겨 있기 때문에 cb=4일 때로 넘어간다.(line 484~487) 이 때가 되면 모든 과정이 완료되었다는 것으로 판단하여 차체의 모든 모터를 정지시킨다.

c. Main function(line 492-512)

메인 함수(line492~512)에서는 노드, subscriber, tcp/ip 를 위한 요소들을 정의하고 콜백함수를 실행시키기 위한 while 문을 담는다.
subscriber 에서는 ball_detection 노드에서 실시간 최신 데이터만 받아올 것이기 때문에 queue=1로 설정한다.(line 496)
Tcp/ip 통신을 위한 소켓을 정의한다.(line497)
socket 안에 첫번째 인자에 PF_INET 를 넣어주면 IPV4 타입을 사용한다는 것이다.
두번째 SOCK_STREAM 을 넣어주면, 연결지향형 소켓을 만들겠다는 것이다. 연결지향형 소켓이란, 소켓끼리 서로 연결된 상태에서 통신을 하는 것으로, TCP/IP 통신이 이에 해당한다.
소켓의 구성요소를 담을 구조체를 생성한 것으로, IP 주소와 PORT 번호를 통신에 맞게 입력해준다.(line498~500)
통신이 성공했는지의 유무를 판단하기 위해 통신에 실패하였을 경우 fail이 뜨게 if문을 정의했고,(line502~506) 정상적으로 연결되었을 경우는 로스가 돌아가는 동안 spinOnce()를 통해 콜백함수를 실행시키게 한다.(line508~510)

5. Problems & Solutions

(1) Feedback의 도입 및 횟수 optimization

두 대의 카메라를 통해 stereo 기법을 사용함으로써 기존의 한 대의 카메라에서 나타난 오차와 비교하여 현저히 줄어든 값의 오차를 얻을 수 있었다. 그러나 우리가 처음에 3m를 전진하라고 명령을 주어도 정확하게 그 지점에 도착할 수는 없기 때문에 중간에 feedback이 존재하여야 한다고 생각하였다.
우리가 처음 사용한 방법은 다음과 같다.

a. 공이 2m 이상 떨어져 있으면 1m 전진
b. (공과의 거리-d)의 70%를 이동 X2
c. 공으로부터 d(0.6m) 떨어진 지점까지 실시간 feedback을 통해 직진+회전
d. d만큼 떨어진 지점 도착 시 pickup function 실행

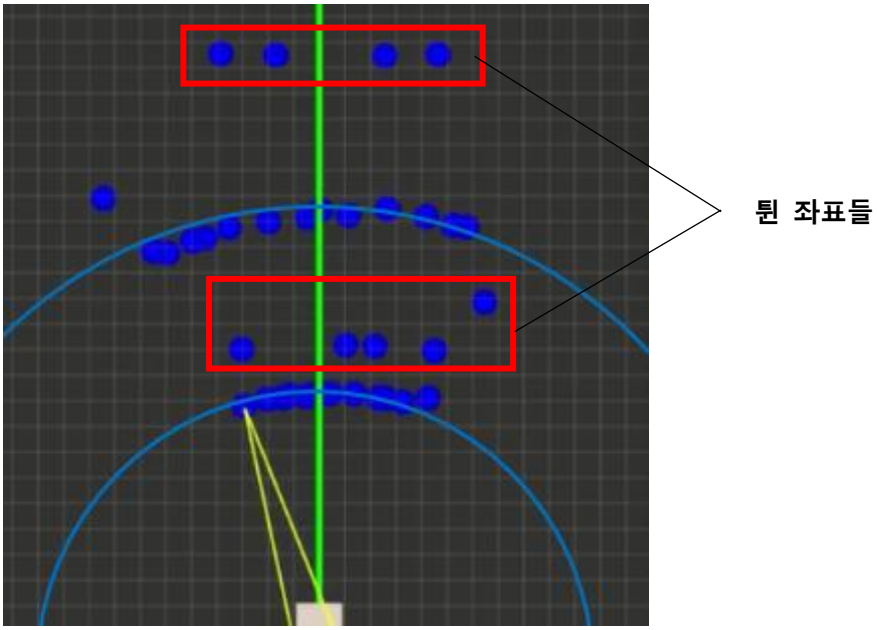
1m를 전진하고 70%의 feedback을 두 번 실행함으로써 공으로부터 1m 이내에 위치하게 할 수 있었고, 거기에 실시간으로 feedback을 통해 직진하면서도 xm을 0에 근사하게 유지(차체 중심에 위치)할 수 있었고, 안정적으로 pickup function을 실행시킬 수 있었다.
그러나 1m 전진 및 70% 이동을 두 번 실행하면서 이미 3번의 feedback을 사용하였고, 거기에 추가로 실시간 feedback까지 이용하면서, feedback의 횟수가 너무 늘어나 버렸다. 이로 인해 feedback 사이의 takerest로 인한 멈추는 시간이 많이 증가하였고, a~d의 과정에 걸리는 시간이 많이 늘어나게 되었다.
우리의 차체 특성상 pickup 범위가 넓기 때문에 매우 높은 정확도가 필요 없었기 때문에 시간을 줄이기 위해 feedback 횟수를 줄이는 방법을 고려해보았다. 우리가 최종적으로 선택한 방법은 다음과 같다.

a. (공과의 거리 - d)의 80%를 이동
b. pickup function 실행

위처럼 한 번의 feedback만을 거쳐 공을 pickup 했을 때 공을 정확히 잘 수집했고 이에 드는 시간 역시 현저히 줄일 수 있었다.

(2) 카메라 좌표 튐 현상

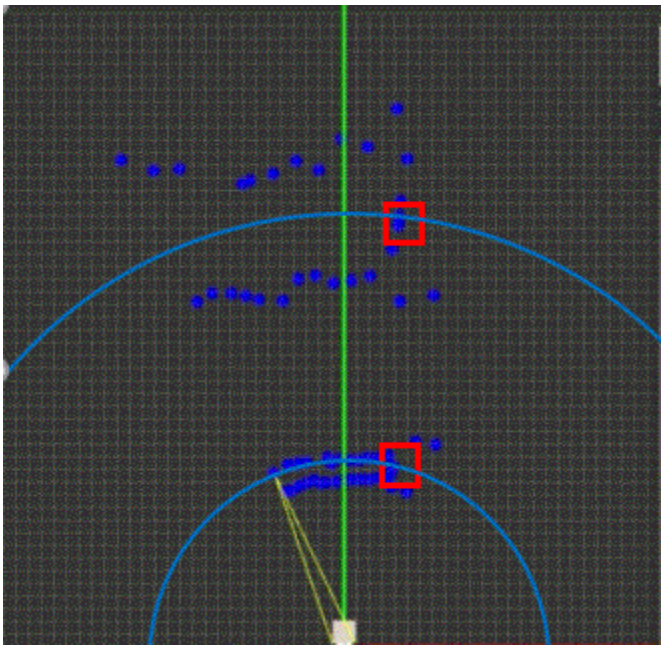
두 대의 카메라를 사용해 stereo 기법으로 이용한 결과 공의 좌표 정확도는 매우 높아졌다. 그러나 stereo 기법의 특성상 작은 진동에도 이는 엄청난 영향을 주었는데, 실제로 차체가 움직일 때, 특히 회전 중에 아래 그림과 같이 공의 좌표 값이 불규칙적으로 튀는 현상을 확인할 수 있었다.



[Figure 14] Ball coordinates read while rotation

우리의 알고리즘에서 공의 좌표는 다음 행동을 결정하기 때문에 정밀한 값을 받아내는 것이 매우 중요했다. 이 좌표 튐 현상을 방지하기 위해 우리는 다음 행동을 하기 위한 좌표 값을 받기 전에 잠시 멈추는 행동을 추가하기로 했다. 가령 이전에 회전 -> 직선 운동이었더라면 그 사이에 멈춤을 추가하여 회전 -> 멈춤 -> 직선 운동으로 바꾸는 것이다. 우리는 이 멈추는 함수를 takerest로 정의하였다.

takerest함수를 도입함으로써 기존의 좌표 튐 현상을 대처할 수 있었다. 차체의 회전 운동 중 이 함수를 도입한 결과는 아래 사진과 같다.



[Figure 15] Ball coordinates read while rotation (+takerest)

빨간색의 사각형 안에 있는 파란 점이 회전운동 중 takerest(12) 때 읽히는 좌표의 값이다. 회전운동 중 인식되는 좌표의 값, 즉 다른 파란 점들의 위치와 비교해보면 잠시 멈추는 함수를 도입함으로써 훨씬 높은 정확도의 좌표 값을 받을 수 있다는 것을 알 수 있다.

(3) 초록 공을 이용한 바구니 정렬

초기에는 파란 공 3개를 다 수집 후 회전하다가 초록 공 2개가 모두 detect 되면 그 좌표를 읽어 직진한 후 release를 하려고 하였다. 그러나 우리의 앞부분이 바구니의 입구 크기와 거의 일치하였고, 바구니의 앞부분과 차체가 평행하게 정렬되지 않은 상태였기 때문에 공을 성공적으로 배출하는 데 한계가 있었다. 그래서 우리는 바구니로 바로 가지 않고, 바구니의 중점에서 어느 정도 떨어진 지점으로 먼저 간 후, 중앙 및 수평 정렬을 거친 뒤 직진하여 정확도를 높이기로 하였다.
처음에는 이 거리를 0.8m로 설정하였는데 이 결과 카메라와 초록 공의 사이 거리가 짧아 초록 공 두 개가 같이 인식되지 않는 경우가 많았다. 반면 거리를 늘려 1.5m로 설정한 결과, 약간의 각도 틀어짐에도 1.5m를 전진하면서 오차가 커져 공 배출에 실패하는 경우가 발생하였다. 여러 번의 실험 결과 1.2m로 설정 시 두 문제를 모두 해결할 수 있었다.
문제는 바구니의 중점에서 1.2m 떨어진 지점 (x_p,y_p)로 움직이면서다. 이 지점으로 직진 운동 후 다음 운동((x_p,y_p)의 갱신 및 초록 공의 좌표를 이용한 중앙 및 수평 정렬)을 위해서 두 초록 공의 좌표가 필요한데 많은 경우 카메라의 시야 내에 모두 존재하지 않았다. 이 경우 차체를 시계방향이나 반시계방향으로 회전시키면서 두 초록색 공이 모두 카메라의 시야 내에 들어오도록 해야 하는데, 방향을 잘못 설정 시 쓸모 없는 회전을 한 바퀴 하게 되는 것이므로 낭비되는 시간이 매우 늘어나게 된다. 이를 해결하기 위해 rotat이라는 bool을 선언하며 모든 경우의 수를 나누어 시계방향으로 회전해야 할 경우에는 rotat에 true를 선언해줬고, 반시계방향으로 회전해야 하는 경우에는 rotat에 false를 선언해주는 알고리즘을 고안했었다. 그러나 예상보다 예외의 경우의 수가 매우 많았고 원하는 움직임과는 반대의 행동을 하는 경우도 많이 발생하였다.
이를 해결하기 위해 강제로 초록 공 두 개를 바라보도록 하기로 하였다. 삼각함수를 이용하여 몇 도를 돌아야 다시 바구니의 중점을 바라보게 될지 계산하였고, 직선 운동 후 이 각도만큼 회전을 시킴으로써 항상 카메라에 두 개의 초록 공이 인식되도록 할 수 있었다.

6. Desired coding schedules

주차	내용
Week1,2	조 편성 및 역할 분담
Week3	ROS 공부 - 창시구 교재 ROS part 기반 - 창시구 교재에 유용한 knowhow가 많이 포함되어있음. 필독강추!!!
Week4	ROS 공부 - ROS 교재 기반 - Node, topic 등 ROS 프로그램 구조 이해 - 기본적인 ROS 구동 방법 공부 - Publish & subscribe 사용법
Week5	1차 발표 / keep studying c++ & ROS - 당장 할 일이 없어 공부에 게을러질 수 있지만 나중에 밤을 덜 새고 싶으면 열공해야하는 시기!!
Week6	정해진 하드웨어 기반 알고리즘 구상 및 코드화 - 알고리즘 도식화(ex. 파란공을 어떻게 탐색할지, path planning, 바구니 정렬 방법 등) - 각 알고리즘을 코드로 변환
Week7	구현한 알고리즘 테스트 및 xbox controller 기반 시스템 구동 - xbox_ctr node를 사용하면서 write function(Labview에 TCP/IP 통신을 하는 기능) 사용법 숙지
Week8	[중간고사 기간]
Week9	발표 준비 및 코드 debugging - Debugging에 대한 고찰 필요 - 문제점을 찾기 위해 debugging index(cf. ROS_INFO())를 어디에 어떻게 넣을지?
Week10	2차 발표
Week11	파란공 줍기 function & 바구니 앞 정렬 function 구동 성공 - data_integration_node를 통째로 테스트, 수정, debugging하기보다는 각각의 function, operation을 분리하여 테스트, 디버깅 후 합치는 것이 보다 효율적이다.
Week12	개별적으로 완성한 함수를 모두 합쳐 실험하고 debugging - 몇 번 성공했다고 자만하면 안된다. 수십 번 실험해보고, 여러가지 예외 상황을 가정하여 실험하고 개선해야한다.
Week13	Try & Error, Debugging - 계속 실험 & debugging - 시연 장소에 맞게 최적화
Week14	최종 발표, 시연 / 시연 환경 꼼꼼히 체크 - 조명 체크, 바구니 옆 초록색 공의 위치 정밀하게 체크 등