

2019 Spring ME492 Final Report

Team E

Jeongha Lee, 20160510

Hyunbin Kim, 20150236

Sungmin Park, 20120443

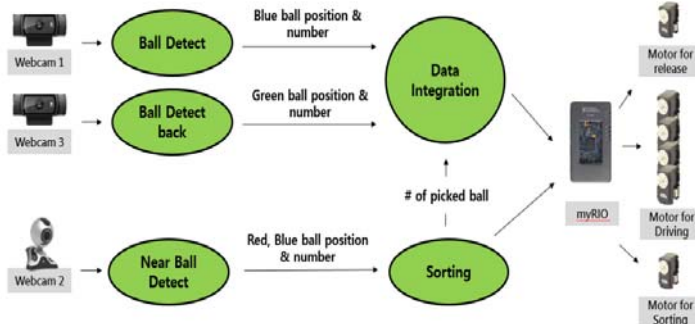
Jekyung Jo, 20160636

2019.06.16

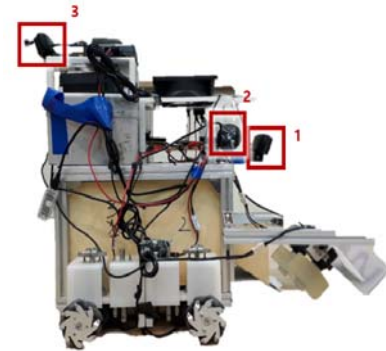
1. Algorithm Philosophy and Ideas

본 팀의 ROS 코드에서 가장 특징 점으로 뽑을 수 있는 것은 회피 기동의 부 존재와 위험상황 대처 능력에 있다. ‘롤러’라는 픽업 모듈의 특성상, 공을 지나가면 픽업이 되는 방식으로 공을 픽업할 때 멈추거나 다른 특정한 모션을 할 필요가 없어 시간이 지연되지 않으며, 파란 공과 함께 경로상에 존재하는 빨간 공까지 모두 픽업하기 때문에 별다른 회피 기동 알고리즘을 고려하지 않았다. 그러나 빨간 공을 수거한다는 특징에 의해서 항상 별개의 형태로 존재하는 ‘sorting’이라는 노드가 있다. 또한 이 노드는 로스 코드의 주요한 부분이 되는 data integration 노드에 종속되지 않고 독립적으로 존재하며, myRIO에게 모터 신호를 보내기도 한다. 두번째 특징인 위험상황 대처 능력은 바구니로의 귀환단계에서 파란 공이 있는지 다시 한번 확인하는 과정이다. 본 팀의 로봇에는 총 3개의 카메라가 부착되어 있는데, 바구니로의 귀환과정에서는 후면 카메라(camera 3)를 이용하기 때문에, 전면 카메라(camera 1)를 통해서, 파란 공이 모두 수거되지 못하였을 때, field안의 남은 파란 공의 개수와 색깔을 파악할 수 있다. 따라서 귀환과정에서 지속적인 피드백을 통해서 픽업하지 못했던 공을 확인, 다시 수거할 수 있는 능력이 있다.

이렇게 두 가지의 큰 특징점과 안정성과 신속성이라는 큰 이상향을 가지고 코드를 작성하였으며, 이 두 가지의 가치에 따라서 로봇의 의사결정에 주요한 부분을 차지하는 ‘data integration 노드’는 여러 버전을 거치며 발전해왔다. 크게는 세 가지의 버전이 있으며, 각각의 버전은 세 개의 독립된 알고리즘이라고 할 수 있을 만큼 판이하게 다르다. 따라서 각각의 세 버전의 설명과 한계점을 통해서 마지막 최종 버전이 나온 배경에 대해서 설명하고, 최종적으로 채택한 알고리즘에 대해 더 자세히 설명하고자 한다. 또한 ROS system은 2개의 노드로 이루어져 있어, sorting 노드에서 빨간 공 혹은 파란 공을 올바른 저장소에 넣기 위해 저장소 앞에 위치한 패널을 움직이는 한편, 현재까지 로봇이 픽업한 파란 공을 개수를 세어 data integration 노드로 보낸다. data integration노드에서는 모든 정보를 통합해, 로봇의 좌우, 전후 병진과 회전운동을 총괄하고 공의 방출을 결정한다.



<그림 1> 전체 ROS 시스템 구성도



<그림 2> 웹캠 부착 위치

2. data integration node

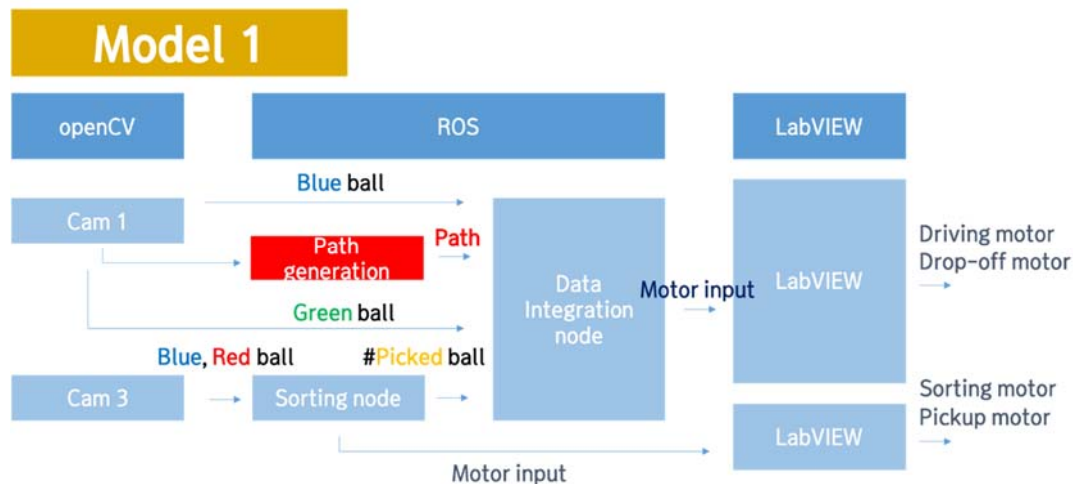
이 노드는 로봇의 의사결정의 주요한 부분을 차지하는 노드로 camera 1, camera 3와 sorting node에서 보내오는 정보를 통합하여, 로봇의 구동을 결정하는 노드이다. 로봇의 병진 운동과 회전 운동을 지시하고, 파란 공이 방출되는 통로에 쓰인 모터에 해당하는 신호를 myRIO에 보냄으로써 파란공을 바구니로 방출하는 역할을 한다.

이렇게 데이터를 통합하고 이를 이용해 로봇의 경로를 결정하는 방법에 있어서 여러 고민을 해왔고, 코드를 작성하고 실험을 반복하면서 생기는 오류들을 해결하기 위해서 코드를 지속적으로 개선하였다. 그 결과 서로 다른 2가지 버전을 거쳐서 최종적인 integration 노드를 결정할 수 있

었다. 최종 버전을 제외한 이전의 버전들을 간략히 설명하고, 각각의 버전으로의 진행을 촉진시킨 한계점에 대해 설명하고자 한다.

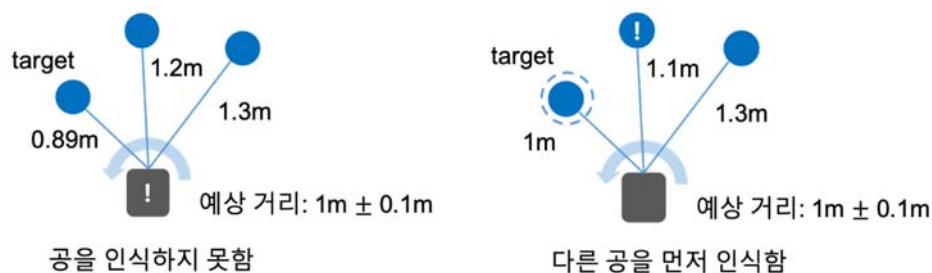
2.1 Version 1

미션 수행을 위한 처음 버전은 그림3 와 같이 3개의 노드로 구성되었다. Path generation node는 처음 카메라로부터 파란 공의 위치 정보를 받아 제자리 회전과 직진으로 구성되는 최단 거리를 계산한 후에 data integration node로 보낸다. Data integration node는 받아온 path 정보대로 로봇이 이동할 수 있도록 motor input 정보를 myRIO로 전달한다. Sorting node는 독자적으로 camera 2로 부터 파란 공과 빨간 공의 위치 정보를 받아 공을 분류하는 motor input 정보를 myRIO로 전달하고 저장한 파란 공의 개수를 유추하여 data integration node에 전달한다.



<그림 3> Version 1 시스템 구성도

Data integration node에서 정해진 path대로 로봇을 이동하기 위해서는 처음에 결정했던 순서대로 파란 공을 인식하고 움직여야 한다. 따라서 본 팀에서는 공과 camera 1 사이의 거리를 이용하고자 하였는데, 처음 path를 계산할 때 파란 공과 로봇, 파란 공과 다음 픽업해야 할 파란 공 사이의 거리를 저장해, 현재 camera 1의 시야에 있는 파란 공의 거리와 비교함으로써 픽업해야 할 공을 결정하는 방법이었다. 그러나 버전 1 구현 결과 측정 거리의 오차로 그림4와 같이 비슷한 거리의 공들이 있을 때는 타겟 공을 아예 인식하지 못하거나 비슷한 거리에 위치한 다른 공을 인식하는 경우가 발생하였다. 이러한 문제들로 인해서, 처음에 계산한 path대로 로봇이 이동하지 않았고, 웹캠에서 공의 반지름을 이용해 거리를 계산하는 과정이 이산적 이기 때문에 거리에서 무시할 수 없는 오차가 계속 발생하여, 로봇이 어떻게 구동될 것인지 예측할 수 없었다. 본 팀은 이러한 문제를 해결하기 위해 새로운 알고리즘을 구상하였다.

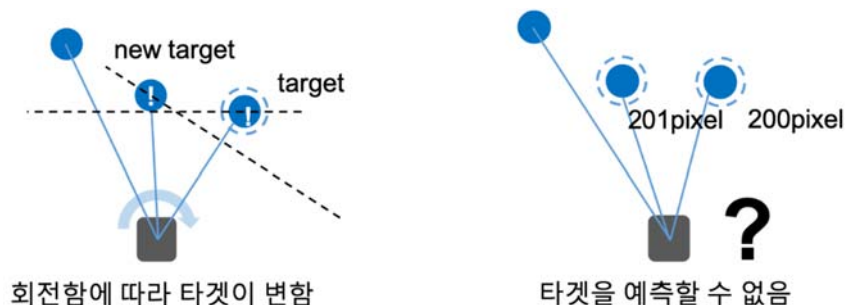


<그림 4> Version 1의 문제

2.2 Version 2

버전 2에서는 path를 계산하더라도, 로봇이 그대로 이동하지 못하기 때문에 최단거리를 계산하는 path generation node를 파기하고 현재 공들 중에서 가장 가까운 공을 타겟으로 설정하는 새로운 알고리즘을 설정하였다. 또한 구동 과정에서 사용하는 데이터를 거리 값이 아닌 상대적으로 robust한 공의 픽셀 y좌표 값을 사용하게 되었다. 따라서 버전 2에서는 매 주기마다 가장 가까운 공(픽셀 y좌표가 가장 작은)을 타겟으로 설정하여 이동한다.

그러나 버전 2에서는 공의 픽셀 y좌표를 사용한다는 특징 때문에 그림5와 같이 가장 가까운 공이 회전에 따라 변하거나, y좌표가 비슷한 공이 있을 때, 어떤 공을 향해서 이동할지 예측할 수 없었다. 심지어 가장 가까운 공이 계속 바뀌어서 두 공 중, 어느 공으로도 향하지 못하고 중심으로 직진하는 경우도 관찰되었다. 결과적으로 로봇이 반대 방향으로 회전하며 공을 찾거나 전체 이동 거리가 길어지는 등의 비효율적으로 움직이는 현상이 발생했던 것이다. 로봇의 이동이 예측 가능하지 않았기 때문에 불안정성이 커지고 이에 따라 최적화 과정이 어려워졌다. ‘y좌표가 가장 작은 공으로 향한다’는 알고리즘의 문제였기 때문에 다른 방식으로 경로를 결정하는 알고리즘을 구상하게 되었다.



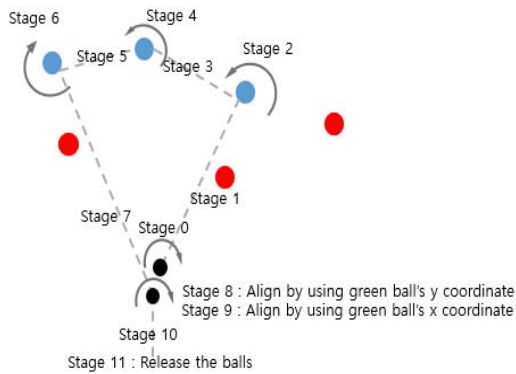
<그림5> Version 2의 문제

2.3 Version 3

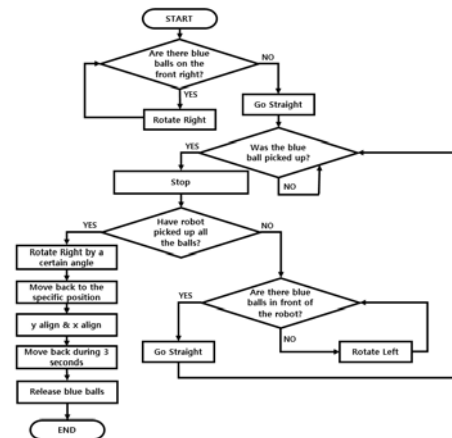
2.3.1 Philosophy and Ideas

마지막 최종 버전은 아주 간단한 명제를 가지고 있다. ‘가장 오른쪽 공을 먼저 먹는다’이다. 따라서 출발점에서는 무조건 오른쪽으로 회전하고, 파란 공을 수거하는 동안에는 왼쪽으로 회전한다.

시간을 절약하기 위한 2가지의 장치를 마련하였는데, 하나는 바구니로 돌아올 때 후진 이동을 한다는 점과, 마지막 파란 공을 수거한 후에는 이전 과정에서 계산한 로봇의 총 회전각을 이용해서 시간을 절약할 수 있는 회전 방향을 결정한다는 점이다. 또한 로봇의 행동을 명령하기 위해서 각각의 지점을 stage로 나누어서 stage number에 따라서 로봇이 특정한 기능을 수행할 수 있게 하였다. 이렇게 stage로 나누어서 좋은 점은 두가지가 있다. 로봇이 구동되는 과정에서 어떤 stage에 머무르는지를 확인함으로써 각 과정에서 생긴 오류를 손쉽게 파악할 수 있었다. 다시 말해, 각각의 stage로 로봇의 구동이 나누어져 있고, 각 과정에서는 특정한 조건을 만족하도록 되어 있었기 때문에 로봇이 예측하지 않은 행동을 할 때 디버깅이 쉬웠다. 또 다른 장점으로 로봇의 구동을 손쉽게 조정할 수 있다는 것이다. 이 장점이 가장 잘 쓰인 곳은 위기사항 대처 프로세스였는데, 로봇은 웹캠으로부터 데이터를 받을 때 마다 전역 변수인 stage number의 값을 통해서 어떤 스테이지를 실행해야 하는지 결정한다. 따라서 전역변수인 stage number만 변경하면, 스테이지 간의 이동이 굉장히 자유롭다. 예를 들어 마지막 단계인 stage 7(후면 카메라를 이용해 바구니 앞의 특정한 점으로 귀환하는 stage)에서 전면카메라를 통해서 픽업되지 못한 파란공을 발견했을 때 이 웹캠 정보를 바탕으로 파란공의 개수가 1개이면 stage 4로, 2개이면 stage2로 이동하여 다시 공을 줍는 과정을 수행할 수 있게 된다.



<그림 6> stage 소개



<그림 7> 알고리즘 순서도

2.3.2 Explanation of ‘Version 3’ code

2.3.2.1 Functions

함수 이름	함수 기능	코드 설명
ballInit()	데이터 초기화	<pre>for(int i = 0; i < 20; i++){ ball_X[i] = 0; ball_Y[i] = 0;</pre> <p>전면 카메라로부터 받은 파란 공의 픽셀 좌표는 ball_X, ball_Y array 에 저장되는데, 각 루프가 돌 때마다 웹캠으로부터 새로운 값을 받아야 하기 때문에 for문을 이용해 각 array의 데이터를 0으로 초기화 시킨다.</p>
ball_G_Init()	데이터 초기화	<pre>for(int i = 0; i < 20; i++){ ball_X_G[i] = 0; ball_Y_G[i] = 0;</pre> <p>후면 카메라로부터 받은 녹색 공의 픽셀 좌표는 ball_X_G, ball_Y_G array 에 저장되는데, 각 루프가 돌 때마다 웹캠으로부터 새로운 값을 받아야 하기 때문에 for문을 이용해 각 array의 데이터를 0으로 초기화 시킨다</p>
camera_Callback	Camera 1에서 받은 데이터를 전역 변수들에 저장	<pre>int count_b = position->size; /*position 메시지 형태에서 size에 해당하는 count_b에 전면 웹캠에 보이는 파란공의 ball_number = count_b; /*ball_number는 전역변수로서 전면 카메라 웹캠의 데이터가 업데이트 될 때마다 본 ballInit(); // 새로운 값들을 넣기 위 for(int i = 0; i < count_b; i++){ ball_X[i] = position->img_x[i]; ball_Y[i] = position->img_y[i];</pre> <p>본 함수는 camera 1 데이터를 subscribe 할 때마다 콜</p>

백되는 함수로써, 기능은 camera 1으로부터 받은 데이터의 값들을 전역 변수에 저장 하는데에 있다.

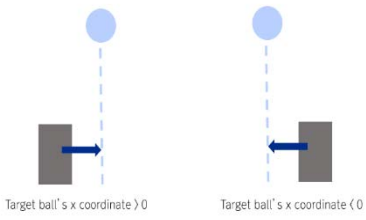
core_msgs파일에 있는 position메세지 파일 형태를 취하는 메세지를 받게 된다. position 메세지 형태에서 size에 해당하는 값(size에는 camera 1에 보이는 파란 공의 개수 값이 들어가 있다.)을 count_b 변수에 저장하였는데, count_b에 camera 1에 보이는 파란 공의 갯수를 저장한 것이다. ball_number는 전역변수로써 전면 카메라에 보이는 파란 공의 개수를 의미한다, 이 변수는 노드 전체에서 쓰이기 때문에 웹캠의 데이터가 업데이트 될 때마다 본 함수가 작동되면서 같이 업데이트 된다. 새로운 값들을 넣기 위하여 ball_X.ball_Y array의 이전의 값들을 모두 0으로 초기화 하고, for문을 전면카메라에 보이는 파란 공의 개수 만큼만 돌림으로써 쓰레기 값이 들어가지 않도록 하였다.for 문을 통해서 position 메세지 형태의 img_x, img_y에 해당하는 값(파란 공의 픽셀 좌표 값)을 ball_X, ball_Y array에 저장한다. 이 과정을 통해서 ball_X, ball_Y array에는 전면 카메라에 보이는 파란 공의 픽셀 좌표 정보가 입력되게 된다.

back_camera_Callback Camera 3에서 받은 데이터를 전역변수들에 저장

```
int count_g = position_back->size3;
/*position_back 메세지 형태에서 size3에 해당
count_g에 후면 웹캠에 보이는 녹색공의 갯수를
ball_number_G = count_g;
/*ball_number_G는 전역변수로써 후면 카메라에
웹캠의 데이터가 업데이트 될 때마다 본 함수가
ball_G_Init(); // 새로운 값들을 넣기 위하여
for(int j = 0; j < count_g; j++){//for문
    ball_X_G[j] = position_back->img_x3[j];
    ball_Y_G[j] = position_back->img_y3[j];
```

본 함수는 camera 3의 데이터를 subscribe 할 때마다 콜백 되는 함수로써, 기능은 camera 3로부터 받은 데이터의 값들을 전역 변수에 저장하는 데에 있다.

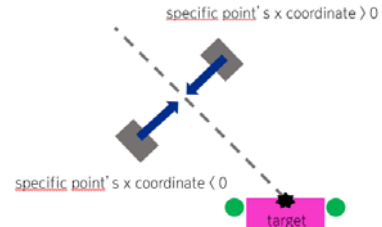
core_msgs파일에 있는 position_back메세지 파일을 취하는 메세지를 받게 된다 position_back 메세지 형태에서 size3에 해당하는 값(size3에는 녹색 공의 개수 값이 들어가 있다.)을 count_g 변수에 저장하였는데, count_g에 후면 웹캠에 보이는 녹색 공의 갯 수를 저장한 것이다. ball_number_G는 전역변수로써 후면 카메라에 보이는 녹색 공의 개수를 의미한다, 이 변수는 노드 전체에서 쓰이기 때문에, 웹캠의 데이터가 업데이트 될 때마다 본 함수가 작동되면서 같이 업데이트 된다. 새로운 값들을 넣기 위하여 ball_X_G.ball_Y_G array의 이전의 값들을 모두 0으로 초기화 하고, for문을 camera 3에 보이는 녹색 공의 개수만큼만 돌림으로써 쓰레기 값이 들어가지 않도록 하였다. for문을 통해서 position_back 메세지 형태의 img_x3, img_y3에 해당하

		<p>는 값(녹색 공의 픽셀좌표값)을 ball_X_G, ball_Y_G array 에 저장한다.이 과정을 통해서 ball_X_G, ball_Y_G array에는 camera 3에 보이는 녹색 공의 픽셀 좌표 정보가 입력되게 된다.</p>
sorting_Callback	<p>sorting_node로 부터 받는 데이터를 전역변수에 저장</p>	<pre>picked_ball = ball->data;</pre> <p>본 함수는 메시지가 subscribe 될 때마다 콜백 되는 함수로써, 기능은 sorting_node로 부터 받은 메시지를 전역변수인 picked_ball에 저장하는 데에 있다.</p> <p>sorting_node에서 메시지를 publish할때 std_msgs의 Int32 형태의 메시지를 사용하였는데, 보낼 데이터가 int 형태의 정수형 변수 하나였기 때문에, 새로운 메시지 형태를 만들지 않고, std_msgs library를 이용하는 것이 편리하다고 판단해서 였다. data 에는 픽업된 파란 공의 개수 데이터가 들어가 있으며, 이 값을 전역 변수인 picked_ball에 저장하였다. 즉, picked_ball 전역 변수가 의미하는 것은 지금까지 픽업한 파란 공의 개수이다</p>
go_straight	<p>공과 로봇을 일직선상에 맞추면서 직진</p>	<pre>dataInit(); // 새로운 if(ball_x > 0){ /* 입력받은 ball_x의 data[5] = 0.8; //앞 data[4] = 0.2; //오른쪽 } else{ //ball_x값이 음수이면 data[5] = 0.8; data[4] = -0.2; //왼쪽 } }</pre>  <p><그림 8> 좌우 정렬</p> <p>본 함수의 기능은 공의 x좌표값을 입력 받아 이 좌표를 이용해 공과의 수평을 맞추면서 직진을 하는 데에 있다. myRIO에 보내는 속도 신호는 합쳐서 1로 제한하였으며, 코드에서 쓰인 전진 속도인 0.8과 좌우 병진 이동 속도인 0.2는 여러 번의 실험을 통해서 최적의 속도라 판단, 결정하였다. 이때 공의 x좌표 값으로는 로봇이 향해서 직진하게 될 공의 x좌표를 입력 받는다. 새로운 명령을 저장하기 위해서 이전에 data array(myRIO에 보내는 명령을 저장한 array)에 있었던 값을 초기화하고,입력 받은 공의 x좌표 값이 양수이면 로봇이 공보다 왼쪽에 있다는 것을 의미하므로, 직진 신호를 주는 동시에 우측으로 이동하는 신호를 준다. data[5]에 해당하는 값은 앞뒤 이동에 관한 명령으로 양수면 앞으로 직진을 의미한다. data[4]에 해당하는 값은 좌우 이동에 관한 명령으로 양수면 오른쪽 병진 이동을 의미한다 입력 받은 공의 x좌표 값이 음수이면 로봇이 공보다 오</p>

른쪽에 있는 것으로, 직진과 동시에 좌측 이동 신호를 준다.data[4]는 좌우 이동에 관한 명령으로 음수면 왼쪽 병진 이동을 의미한다.

go_back 두 녹색 공의 중간 지점을 로봇과 일직선상에 놓으면서 후진

```
dataInit(); // 새로  
  
if(target > 0){  
    /*후면 카메라 상  
    data[5] = -0.8;  
    data[4] = -0.2;  
}  
else{  
    data[5] = -0.8;  
    data[4] = 0.2;  
}
```

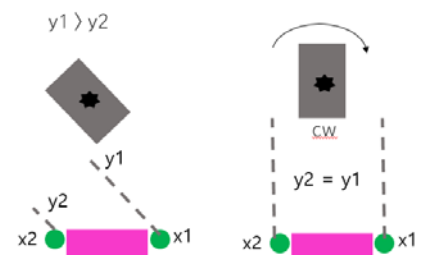


<그림 9> 좌우 정렬 후진

본 함수의 기능은 target 값을 입력 받아 target이 되는 지점과의 수평을 맞추면서 후진하는 데에 있다. 이 함수는 바구니로 귀환할 때 쓰이게 된다 이때 target값으로는 두 녹색공의 중심의 x좌표값을 입력 받게 된다. 새로운 명령을 저장하기 위해서 이전에 data array(myRIO에 보내는 명령을 저장한 array)에 있었던 값을 초기화 하고, go_straight함수에서 쓰였던 방법과 동일한 방법으로 로봇을 target 지점과 일직선상에 놓으며 후진이동 시킨다. 이때, 후면 카메라 상에서 보이는 값을 비교하는 것이기 때문에 명령이 go_straight과는 반전된다

y_align 회전을 통해 바구니와 평행하게 정렬

```
dataInit(); // 새로  
  
if(y1 > y2){  
    /*y1, y2값을 입력  
    y1값은 상대적으로  
    따라서 y1>y2인 상태  
    반대의 경우에는 왼쪽  
    이때 y1=y2인 경우  
    (픽셀값을 이용하기  
    data[9] = 0.2; //  
}  
else{  
    data[9] = -0.2; //  
}
```



<그림 10> y align

본 함수의 기능은 로봇의 후면이 바구니를 평행하게 바라보게 하는 것이다. 일단 새로운 명령을 저장하기 위해서 이전에 data array(myRIO에 보내는 명령을 저장한 array)에 있었던 값을 초기화 한다. y1, y2값을 입력 받는데, 이때 y2값은 후면 카메라에서 x좌표가 가장 큰 값을 가지는 녹색 공의 y좌표이고,y1값은 상대적으로 작은 x좌표를 가지는 녹색 공의 y좌표값이다. 따라서 y1>y2인 상태에서는 오른쪽 회전을 통해서 바구니와 평행하게 위치하도록 보정한다.반대의 경우에는 왼쪽 회전을 통해서 바구니와 평행하게 위치하도록 한다. 이때 y1=y2인 경우에는 바구니와 평행하게 있다고 판단하게

	<p>된다.(픽셀 값을 이용하기 때문에 로봇의 좌우 위치와 상관없이 바구니와 평행하게 align되면 $y1=y2$가 된다) data[9]에 해당하는 값은 회전에 관한 명령으로 양수면, 오른쪽 회전을 의미한다.</p>		
<p>x_align 병진이동을 통해 바구니를 정면으로 바라보도록 정렬</p>	<div data-bbox="695 392 922 739" data-label="Text"> <pre>dataInit(); // 새로운 명령을 저장하기 위 if(x_center > 40){ /* 이때 두 녹색공의 후면웹캠의 오른쪽에 */ data[4] = -0.3; // } else{ data[4] = 0.3; // }</pre> </div> <div data-bbox="1002 427 1342 678" data-label="Image"> </div> <div data-bbox="965 696 1197 730" data-label="Caption"> <p><그림 11> x align</p> </div> <div data-bbox="689 748 1388 1261" data-label="Text"> <p>본 함수의 기능은 로봇이 바구니와 평행하게 align된 후에 좌우 병진 이동을 통해서 로봇의 후면이 바구니를 정면으로 바라보게 하는 것에 있다. 이때 입력 받는 값은 두 녹색 공의 중심의 x좌표이다 새로운 명령을 저장하기 위해서 이전에 data array(myRIO에 보내는 명령을 저장한 array)에 있었던 값을 초기화 하고, 두 녹색 공의 중심좌표에서 40만큼 더한 지점을 기준으로 이 값이 양수이면 왼쪽, 음수이면 오른쪽으로 이동한다. 이때 두 녹색공의 중심좌표에서 40만큼 더한 지점을 기준으로 잡는 이유는 로봇이 파란 공을 저장하고 방출하는 통로가 후면 웹캠의 오른쪽에 치우쳐 있기 때문에, 파란 공이 방출되는 통로가 바구니의 중심으로 맞춰지도록 하기 위해서이다.</p> </div> <div data-bbox="831 1265 1244 1574" data-label="Image"> </div> <div data-bbox="746 1579 1326 1612" data-label="Caption"> <p><그림 12> 로봇에서 파란 공 방출 통로의 위치</p> </div> <tr> <td data-bbox="193 1621 683 1655"> <p>point_turn 회전을 명령</p> </td><td data-bbox="689 1621 1388 2027"> <div data-bbox="695 1621 1136 1827" data-label="Text"> <pre>dataInit(); // 새로운 명령을 저장하기 위 if(direction == CCW){ // turn ccw data[9] = -0.5; } else if(direction == CW){ // turn cw data[9] = 0.5; }</pre> </div> <div data-bbox="689 1836 1388 2027" data-label="Text"> <p>기능은 0또는 1로 direction 값을 입력 받아 왼쪽 혹은 오른쪽으로 회전하도록 명령한다. 입력 받은 값이 0이면 왼쪽 회전을, 1이면 오른쪽 회전을 명령한다. 명령을 주기 전에, 새로운 명령을 저장하기 위해서 이전에 data array(myRIO에 보내는 명령을 저장한 array)에 있</p> </div> </td></tr>	<p>point_turn 회전을 명령</p>	<div data-bbox="695 1621 1136 1827" data-label="Text"> <pre>dataInit(); // 새로운 명령을 저장하기 위 if(direction == CCW){ // turn ccw data[9] = -0.5; } else if(direction == CW){ // turn cw data[9] = 0.5; }</pre> </div> <div data-bbox="689 1836 1388 2027" data-label="Text"> <p>기능은 0또는 1로 direction 값을 입력 받아 왼쪽 혹은 오른쪽으로 회전하도록 명령한다. 입력 받은 값이 0이면 왼쪽 회전을, 1이면 오른쪽 회전을 명령한다. 명령을 주기 전에, 새로운 명령을 저장하기 위해서 이전에 data array(myRIO에 보내는 명령을 저장한 array)에 있</p> </div>
<p>point_turn 회전을 명령</p>	<div data-bbox="695 1621 1136 1827" data-label="Text"> <pre>dataInit(); // 새로운 명령을 저장하기 위 if(direction == CCW){ // turn ccw data[9] = -0.5; } else if(direction == CW){ // turn cw data[9] = 0.5; }</pre> </div> <div data-bbox="689 1836 1388 2027" data-label="Text"> <p>기능은 0또는 1로 direction 값을 입력 받아 왼쪽 혹은 오른쪽으로 회전하도록 명령한다. 입력 받은 값이 0이면 왼쪽 회전을, 1이면 오른쪽 회전을 명령한다. 명령을 주기 전에, 새로운 명령을 저장하기 위해서 이전에 data array(myRIO에 보내는 명령을 저장한 array)에 있</p> </div>		

		<p>있던 값을 초기화 하고 시작한다. 이 함수는 픽업할 파란 공을 searching 할 때 쓰이게 된다.</p>
point_turn_final		<p>기능과 코드는 point_turn과 동일하나, 명령하는 회전 속도만 다르다. 이 함수는 두 녹색 공의 중심을 searching 할 때 쓰이게 된다.</p>
point_turn_slow		<p>기능과 코드는 point_turn과 동일하나, 명령하는 회전 속도만 다르다. 이 함수는 두 녹색 공의 중심의 좌표를 이용해 로봇을 바꾸니 앞으로 align 할 때 쓰이게 된다</p>
release_ball	공 방출	<pre>dataInit(); data[14]=1;</pre> <p>본 함수의 기능은 로봇이 바꾸니 앞으로 align되었을때 공을 방출하기 위해 저장소 문에 쓰인 모터를 움직여 파란 공 저장고 문을 여는 것이다 새로운 명령을 저장하기 위해서 이전에 data array(myRIO에 보내는 명령을 저장한 array)에 있었던 값을 초기화하는 과정이 선행된다. data[14]에 해당하는 값은 파란 공을 방출하는 통로에 달린 문에 쓰인 모터를 통제하는 것으로 이 값이 1이 되면 문이 바깥쪽으로 열리면서 파란 공이 방출된다.</p>
find_final_position	두 녹색 공 사이에 중간지점 계산	<pre>green_ball_center_x = (x1 + x2)/2; green_ball_center_y = (y1 + y2)/2;</pre> <p>본 함수의 기능은 두 녹색 공의 좌표를 입력 받아 두 녹색 공의 중심 좌표를 찾는 데에 있다. 입력 받은 값을 이용해 중심의 좌표를 계산해서 각각 전역 변수인 green_ball_center_x, green_ball_center_y에 저장한다. 이때 계산된 녹색 공의 중심 좌표를 이용해서 로봇을 바꾸니 앞으로 위치시키게 된다.</p>
green_ball_sorting	녹색공 좌표를 왼쪽 부터 재배열	<pre>if(ball_X_G[0] > ball_X_G[1]){ // ball_X_G[0]값이 더 크다면 bc float tempGX = ball_X_G[0]; float tempGY = ball_Y_G[0]; ball_X_G[0] = ball_X_G[1]; // ball_Y_G[0] = ball_Y_G[1]; ball_X_G[1] = tempGX; // 원래 b ball_Y_G[1] = tempGY; }</pre> <p>본 함수의 기능은 웹캠에서 무작위로 받은 녹색 공에 대한 좌표를 재배열하는 데에 있다. 두 녹색 공 중, 후면 카메라 상에서 가장 오른쪽에 위치한 녹색 공의 좌표를 ball_X_G[1], ball_Y_G[1]에 저장한다. 즉 왼쪽에 위치한 공의 좌표를 array 상의 0 자리에, 오른쪽에 위치한 공의 좌표를 array 상의 1자리에 넣어 재배열하는 것이다 ball_X_G[0]값이 더 크다면 ball_X_G[0]에 해당하는 공이 더 오른쪽에 위치한다는 것으로 array 상의 0과1자리에 위치한 두 데이터 값을 바꾼다. (ball_X_G[1]의 값을 ball_X_G[0]로 대입한다 원래 ball_X_G[0]이었던 값을 ball_X_G[1]으로 대입한다.)</p>

find_target

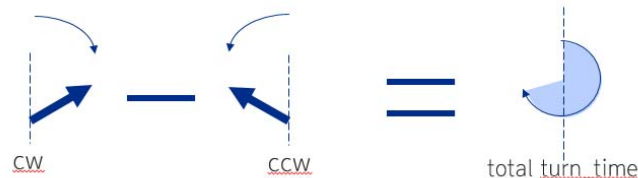
가장 오른쪽에 위치한 파란 공을 찾음

```
for(int i = 0; i < ball_number; i++){  
    if(ball_X[i]>target_distance){  
        /*저장된 target_distance 값과 ball_X값을  
        이 값을 target_distance에 새롭게 저장해  
        target_distance = ball_X[i];  
        target_x = ball_X[i];  
        target_y = ball_Y[i];
```

본 함수의 기능은 camera 1에서 보이는 파란 공의 x좌표를 비교해, 가장 오른쪽에 위치한 파란 공의 좌표를 전역 변수인 target_x, target_y에 저장하는 데에 있다. target_distance값을 -1000으로 초기화하는데, 이때 -1000으로 초기화 하는 이유는 for문을 돌려서 처음 ball_X[0]값과 비교할 때 무조건 ball_X[0]값을 target_distance 변수에 저장하기 위해서이다. 저장된 target_distance 값과 ball_X값을 비교해서 ball_X값이 더 크면 ball_X[i]가 더 오른쪽에 있다는 의미가 됨으로 이 값을 target_distance에 새롭게 저장해 다음 ball_X[i+1]값과 비교한다 for문을 다 돌게 되면, 가장 큰 x좌표를 가지는 ball_X, ball_Y값이 target_x, target_y에 저장되게 된다.

2.3.2.2 Main function

가장 먼저 subscriber를 선언하는데, ball_detect_node(camera 1), sorting_node, ball_detect_back_node로 부터 integration에 쓰일 데이터를 subscribe한다. 각각이 subscribe될 때마다 그에 해당하는 콜백 함수들이 실행되도록 하였다. 이때 코드를 시작하기 전에 while문을 돌려 각 노드로부터 받은 메시지를 받은 후에 1초동안 대기하게 하였는데, 이는 카메라에서 처음 데이터를 받을 때 처음 몇개는 쓰레기 값이 subscribe되는 것을 확인하였고, 이에 쓰레기 값을 사용하지 않기 위해 1초간 정지하여, reasonable한 데이터가 들어올 때까지 기다리게 한 것이다. while문이 돌고 나면 로봇의 구동이 시작되는데, 로봇의 구동을 결정하는 주요한 변수는 stage_number로, 구동과정에서 로봇이 위치한 stage를 규정하고, 수행해야할 활동을 결정한다. 이 변수는 구동이 시작되기에 앞서, 출발선에 해당하는 stage_number = 0 로 초기화 시킨다. turn_time변수 역시 0으로 초기화 시키는데, 이 변수는 stage 1~4에서 로봇의 회전 방향에 따라 더하거나 빼면서 turn_time을 계속해서 갱신한다. turn_time 변수는 반시계 방향으로 돌 때 루프주기만큼 더해지고, 시계방향으로 돌 때 루프주기만큼 감하여 진다.



<그림 13> turn_time 계산 과정

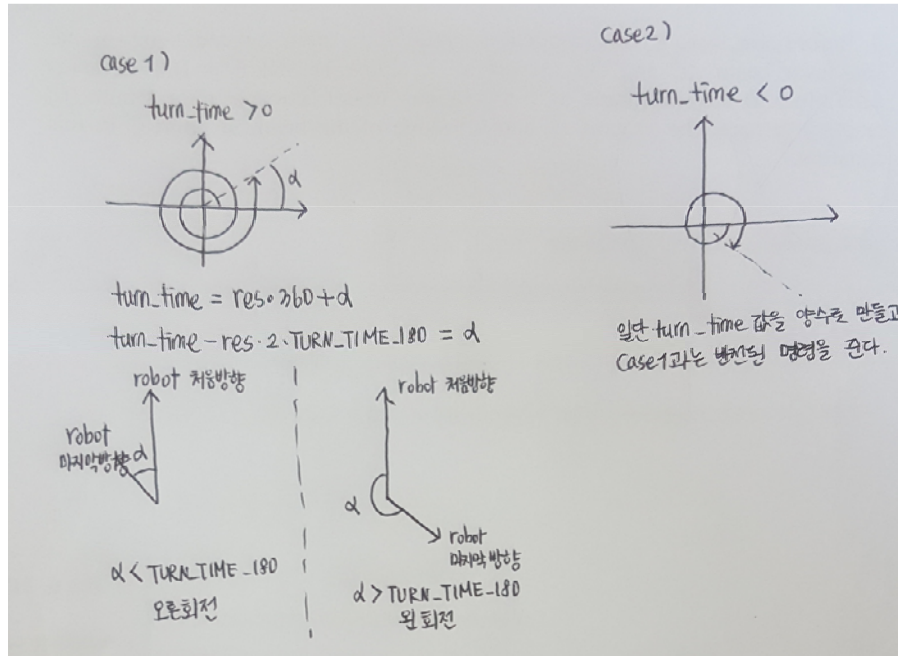
Main 함수 안에서는 크게 두 개의 while 문 안에서 구동 명령을 내리게 되는데, 첫번째 while 문에서는 stage 0~5까지 진행되고, 두번째 while문에서는 일반상황에서는 stage 6~11이, 비상상황에서는 stage 2~11이 진행될 수 있다. 또한 각 while문 안의 가장 마지막 줄은

`write(c_socket, data, sizeof(data));` 로써, myRIO에게 96byte짜리 data array 신호를 보낸다. 첫번째 while 문 안에서, ‘`ros::spinOnce()`’ 를 이용해 메시지를 한번 subscribe해 콜백 함수들을 실행시킨다. 가장 먼저 ‘`find_target`’ 함수를 이용해 camera 1시야에서 가장 오른쪽에 위치한 파란 공의 좌표를 받아온다. 이제 전역 변수인 `stage_number`의 값에 따라 해당하는 if문 안으로 들어가 코드가 진행되게 되는데, 각각의 stage의 기능과 코드의 설명은 아래의 표에 요약하였다.

Stage number	기능 및 코드 설명
0	목표는 가장 오른쪽에 있는 파란 공을 향하도록 로봇을 회전시키는 것이다. 따라서 로봇이 회전을 통해서 가장 오른쪽에 있는 파란 공을 바라보게 되었을 때(로봇이 회전을 통해 가장 오른쪽에 위치한 파란 공의 좌표를 오차범위내에 위치시키면) <code>stage_number</code> 를 증가시켜 다음 stage로 이동시킨다. 오차범위 안에 없다면 왼쪽 혹은 오른쪽 회전을 통해서 파란 공의 좌표를 오차범위 안으로 위치시킨다. 이때 회전 방향에 따라 <code>turn_time</code> 변수에 루프 주기를 더하거나 뺀다.
1	목표는 현재 로봇이 바라보고 있는 파란 공을 향해 직진하는 것으로, 파란 공을 픽업할 때 까지 직진하며, 픽업하면 stage를 이동시킨다. 이때 픽업된 파란 공의 개수를 뜻하는 전역 변수인 <code>picked_ball</code> 이 0이라면 아직 파란 공이 픽업되지 않았다는 뜻으로, 직진을 명령한다. 이 변수가 0이 아니게 되었을 때 파란 공이 픽업되었다고 판단, <code>stage_number</code> 를 증가시켜 다음 stage로 넘어가며, 직진 시에는 ‘ <code>go_straight</code> ’ 함수를 이용해 파란공의 중심을 향하도록 좌우로 이동하며 직진한다.
2	목표는 다음에 픽업할 파란 공의 찾는 한편, 찾은 후에는 파란 공을 정면으로 바라보도록 맞추는 것(가장 오른쪽에 위치한 파란 공의 x좌표가 오차범위내에 있다면 로봇이 픽업할 파란 공을 바라보고 있다고 판단)이다. 무조건 가장 오른쪽에 있는 파란 공을 픽업한다는 알고리즘에 의해서 첫번째 파란 공을 픽업한 후에는 왼쪽으로 회전하면서 다음 공을 찾는 것이 가장 회전 시간을 줄일 수 있는 방법이기, camera 1에서 파란 공이 보이지 않을 때는 무조건 왼쪽으로 회전하도록 하였다. 이때 회전 방향에 따라 <code>turn_time</code> 변수에 루프주기를 더하거나 뺀다.
3	Stage 1과 기능과 코드가 동일하며, stage 3에서는 전역 변수인 <code>picked_ball</code> 이 2가 되면 파란 공을 픽업했다고 판단한다.
4	Stage 2와 기능과 코드가 동일하다.
5	목표는 마지막 남은 파란 공을 픽업할 때까지 직진하고, while문을 나가는 것이다. <code>picked_ball == 3</code> 이면 마지막 파란 공을 픽업했다고 판단하고 다음 stage로 이동시키는 한편, 다음 구동 명령이 있을 때까지 연산 시간이 걸리기 때문에 data array(myRIO에게 보내는 데이터)를 초기화 시켜, 로봇을 정지시킨다. 이후 break문을 통해 while 문을 빠져나간다.

첫번째 while문을 빠져나와 로봇이 마지막 공을 픽업한 자리에서 바구니로 귀환하기위해 어떤 방향으로 회전하는 것이 경제적인지 계산한다. 앞선 과정에서 기록되고 있었던 `turn_time`을 이용하는데, `turn_time`이 양수일 때는 이 시간을 실험을 통해 측정된 시간(`TURN_TIME_180` : 180도 돌 때 걸리는 시간)의 두배로 나누어서 로봇이 회전한 시간을 ‘`n번 회전한 시간 + 360도 이하로 회전한 시간`’ 으로 나눌 수 있다. 이때 `res`값이 `n번` 회전하는데 걸렸던 시간으로 (`n`은 양수), 따라서 `turn_time - res*2*TURN_TIME_180`은 로봇이 360이하로 회전한 시간이 된다. 따라서 이 값이 180도를 회전하는데 필요한 시간보다 작다면 오른쪽으로 회전하는 것이 시간을 줄일 수 있고 크

다면 왼쪽으로 회전하는 것이 시간을 줄일 수 있다. turn_time이 음수일 때는 이 값을 양수로 만들어 같은 과정을 진행한 후에 방향을 정할 때 반대로 정한다. 이와 같이 최소로 회전할 수 있는 방향이 정해지게 되며, 이 코드가 while 문 밖에서 이뤄지는 이유는 연산과정에서 지속적인 피드백이 필요한 것이 아닌 오직 한번만 실행되면 되기 때문이다



<그림 14> total angle 을 이용한 turn direction 결정

앞선 과정을 통해서 정해진 회전 방향을 'turn_direction' 이라는 변수에 저장하고 두번째 while 문이 시작되기 전에 curr_time 변수를 0으로 초기화 시키는데, 이 변수는 후에 stage 10 과 11에서 특정시간동안 후진, 공의 방출을 하기 위해 경과된 시간을 측정하기 위해서 사용된다. 두번째 while에서는 일반 상황에서 stage 6~11이 진행되고, 파란 공 세 개를 모두 픽업하지 못한 비상상황(파란 공을 한 개 혹은 두개만 픽업하였을 때)에서는 stage 2 혹은 stage 4로 돌아가 stage 2~11가 다시 진행된다.

```
if (stage_number > 5){
    if (ball_number == 2){ //
        /*stage_number가 5초과
        실패했다는 뜻으로 첫번째
        프로세스를 반복한다.*/
        stage_number = 2; //전역
        picked_ball = 1; //전면
    }
    if (ball_number == 1){
        /*stage_number가 5초과
        실패했다는 뜻으로 두번째
        프로세스를 반복한다.*/
        stage_number = 4; //전역
        picked_ball = 2; //전면
    }
}
```

비상 상황은 바구니로 귀환하는 모든 과정(stage 6~11)에서 camera 1에 파란 공이 보이는 경우로, 그 개수에 따라서 이전 stage로 돌아가 다시 파란 공을 픽업한다. stage_number가 5초과 일 때 camera 1에 2개의 파란 공이 보인다는 것은 이전 stage에서 파란 공을 픽업하는데 실패했다는 뜻으로 첫번째 파란 공을 픽업한 후에 다음 픽업할 파란 공을 searching 하는 과정을 수행하는 stage 2로 돌아가 프로세스를 반복한다. 전역 변수인 stage_number에 2를 대입함으로써, 손쉽게 이전 stage로 돌아갈 수 있다. 또한 전면 카메라에 파란 공 2개가 보인다는 것은 픽업한 파란 공이 1개라는 의미이므로 전역 변수인 picked_ball에 1을 대입한다. 또 다른 경우로, stage_number가 5초과 일 때 camera 1에 1개의 파란 공이 보인다는 것은 역시 이전 stage에서 파란 공을 픽업하는데 실패했다는 뜻으로 두번째 파란 공을 픽업한 후에 다음 픽업할 파란 공을 searching 하는 과정을 수행하는

stage 4로 돌아가 프로세스를 반복한다. 이 과정 역시 전역 변수인 stage_number에 4를 대입함으로써, 손쉽게 이전 stage로 돌아갈 수 있다. 또한 파란 공 1개가 보인다는 것은 픽업한 파란 공

이 2개라는 의미이므로 전역 변수인 picked_ball에 2를 대입한다.

비상상황 여부를 확인하고 해당이 없다면 코드는 정상적으로 stage 6부터 시작되게 되는데, 이전 while 문을 이미 나왔기에 stage 0 에서 5까지의 똑같은 코드를 다시 한번 넣어주었다.

Stage number	기능 및 코드 설명
6	<p>목표는 로봇의 camera 3가 두 녹색 공의 중심을 바라보게 하는 데에 있다.</p> <p>후면 카메라에 녹색 공 2개가 모두 보일 때, 두 녹색 공의 중심의 좌표를 찾고('find_final_position' 함수 이용), 중심의 좌표가 오차범위내에 있다면 로봇의 후면이 두 녹색 공의 중심을 바라보고 있다고 판단, 다음 stage로 넘어가는 한편, 오차범위 밖에 있다면, 양쪽 회전을 통해 오차범위 안으로 위치시킨다. 동시에 화면에 보이는 녹색 공에 대한 무작위적 좌표를 왼쪽에 있는 공부터 ball_X_G, ball_Y_G array 에서 0자리에 오도록 재배열하고 ('green_ball_sorting' 함수 이용) prev_ball_X_G 라는 전역 변수에 후면 카메라 기준으로 왼쪽에 보이는 녹색 공의 좌표를(ball_X_G[0]) 저장한다.(이때 저장한 값을 stage 7에서 사용한다.)</p> <p>반면에 camera 3에 녹색 공이 1개 이하로 보일 때, 녹색 공 2개가 모두 보일 때까지 정해진 방향으로 회전한다(이때 정해진 방향은 while에 들어오기 전 계산하였던 turn_direction 값이다.)</p>
7	<p>목표는 바구니에서 150 픽셀만큼 떨어진 지점까지 후진으로 로봇을 이동시키는 데에 있다.</p> <p>후면 카메라에 녹색 공 두개가 모두 보일 때, 무작위적으로 subscribe된 녹색 공의 좌표를 가장 왼쪽에 있는 공부터 ball_X_G, ball_Y_G array에 0자리에 넣고 나머지를 1자리에 넣어 재배열하고('green_ball_sorting' 함수), prev_ball_X_G 라는 전역 변수에 후면 카메라 기준으로 왼쪽에 보이는 녹색 공의 좌표를(ball_X_G[0]) 저장한 뒤 두 녹색 공의 중심의 좌표를 찾는다.('find_final_position' 함수) 바구니 앞으로 이동하는 동안에는 go_back 함수를 이용해 두 녹색 공의 중심을 로봇의 중심으로 맞추면서 후진하는 한편, 바구니 앞에서 150픽셀만큼 떨어진 지점에 도달했다고 판단되면 다음 stage로 이동한다. (이때 green_ball_center_y를 거리가 아닌 픽셀 값으로 비교하기 때문에 로봇의 x축 방향 위치에는 상관없이 바구니 앞에서 150픽셀만큼 떨어진 지점에 도달할 수 있다.)</p> <p>후면카메라에 공이 한 개 이하로 보일 때는 이전에 저장한 가장 왼쪽에 위치한 녹색공의 좌표(prev_ball_X_G)와 현재의 값을(ball_X_G[0]) 비교해 큰 차이가 나지 않는다면 camera 3에 보이는 한개의 녹색 공이 가장 왼쪽에 위치한 녹색 공이라고 판단 오른쪽으로 회전해 두 녹색 공이 모두 보이도록 한다. 반대로 큰 차이가 있을 경우에는 왼쪽으로 회전해 두 녹색 공이 모두 보이도록 한다. 이 과정의 마지막에도 prev_ball_X_G 에 ball_X_G[0]을 저장함으로써 다음 루프에서 비교할 때 사용할 수 있게 한다.</p>
8	<p>목표는 로봇이 향하는 방향을 바구니와 평행하게 align하는데에 있다.</p> <p>후면 카메라에 두개의 녹색 공이 모두 보일 때는 일단 green_ball_sorting 함수를 실행하고, prev_ball_X_G 에 ball_X_G[0]를 대입한 뒤 두 녹색 공의 y 픽셀 좌표 ball_Y_G[0], ball_Y_G[1]의 차이가 오차범위보다 큰지 확인한다. 이 차이가 오차범위보다 크다는 것은 로봇이 바구니와 평행하지 않게 있다는 뜻이므로 y align함수를 이용해 로봇을 바구니와 수평 하게 맞춘다.</p> <p>만약 후면 카메라에 보이는 녹색 공이 하나일 때는 이전 루프에서 저장하였</p>

	던 prev_ball_X_G값을 이용해 stage 7에서 소개된 방법과 동일하게 로봇을 정렬시킨다.
9	<p>목표는 바구니와 평행하게 바라보고 있는 로봇을 좌우로 이동시켜, 바구니 앞으로 이동시키는 데에 있다.</p> <p>두 녹색 공이 모두 보일 경우, 'green_ball_sorting()' 'prev_ball_X_G = ball_X_G[0]' 'find_final_position' 코드를 차례대로 실행하고(stage 7,8과 반복되는 내용이기 때문에 따로 언급하지 않았다), abs(green_ball_center_x - 40) 값이 오차범위 안에 위치할 때까지 xalign함수를 이용해 로봇을 좌우로 이동시킨다. 이때 두 녹색 공의 중심의 x좌표(green_ball_center_x)보다 더 40픽셀만큼 더 이동하도록 하였는데, 이는 파란 공 저장고의 방출 문이 후면 웹캠을 기준으로 오른쪽으로 치우쳐 있기 때문이다.(후면 웹캠과 바구니의 위치를 맞추는 것이 아닌 공이 방출되는 통로와 바구니를 동일선상에 놓기 위함이다.) 이후 전역 변수인 curr_time값을 flag_time에 저장하는데 이는 stage 10에서 특정 시간 동안 후진 명령을 줘야하기 때문에 경과된 시간을 확인하기 위해서 이전 stage에서의 curr_time 값을 기억해 놓는 것이다.</p> <p>녹색 공이 한 개만 보일 경우 stage 7,8에서 사용한 방법과 동일하게 회전하면서 두 녹색 공이 모두 보이도록 한다.</p>
10	<p>목표는 3.05초동안 후진해서 바구니 앞에 도착하는 데에 있다. 3.05초는 로봇이 바구니로부터 150픽셀만큼 떨어져 있을 때 바구니 바로 앞으로 오기 위해 후진해야만 하는 시간으로 실험을 통해서 결정하였다. 루프가 반복될 때마다 루프주기만큼 더해지는 curr_time과 이전 stage에서 저장한 flag_time의 값을 비교해 시간이 경과되었는지 판단한다. curr_time - flag_time은 stage 10 시작 후 경과된 시간을 의미하며, 이 값이 3.05초 이상이면 다음 stage로 넘어가는 한편 stage 11에서 쓰이기 위해서 flag_time에 현재의 curr_time값을 다시 저장한다.</p>
11	<p>목표는 파란 공 저장고의 문을 열어 공을 방출시키는 데에 있다. Stage 10과 동일한 방법으로 경과 시간을 측정해서 3초동안만 저장고 문을 여는 신호를 보낸 후 while문을 빠져나가 모든 구동을 끝낸다.</p>

마찬가지로 while 문의 마지막 부분에는 'write(c_socket, data, sizeof(data))' 가 있어 myRIO로 data[24] array를 보내고 루프주기만큼 변수 curr_time을 증가시킨다. while문을 나와서는 소켓을 닫고 노드가 끝난다.

3. Sorting node

이 노드는 camera 2로부터 들어오는 공의 좌표 값을 이용해 로봇과 가장 가까이에 있는 공의 색깔을 판단하여 각각의 공을 각각의 저장소에 올바르게 넣는 것과, 수거한 파란 공의 개수를 세는 데에 있다. 이에 따라 두 가지의 버전을 거치며 발전하였으며, 두 버전은 수거한 파란 공의 개수를 세는 알고리즘에만 차이점이 있기에 이전 버전에 대해서는 다른 점만 언급하고, 최종 버전을 상세하게 설명하고자 한다.

3.1 Version 1

가장 처음 버전에서는 파란 공의 픽업 여부를 판단하기 위해서 camera 2의 시야에서 파란 공이 있다가 사라지면 공을 픽업했다고 판단, 픽업한 공의 개수를 증가시켰다. 그러나 크게 두가지의 문제점이 있었는데, OpenCV에서 파란 공이 아닌 다른 물체를 파란 공으로 한 번이라도 인식한다면, 공을 픽업했다고 판단하고 다음 stage로 넘어가는 치명적인 문제점이 있었으며, 파란 공을

픽업 하지 않고, 그냥 camera 2의 시야에서 스쳐 지나가도 공을 픽업했다고 인식해 다음 stage로 넘어가는 또 다른 문제점이 있었다. 이러한 앞선 이유들로 인해서 수거한 파란 공의 개수를 세는 알고리즘의 개선 필요성을 느끼게 되었고, 더욱 안정성을 높이기 위해서 새로운 버전을 고안하게 되었다.

3.2 Version 2

3.2.1 Philosophy and Ideas

Sorting node는 크게 두가지의 주목할 점이 있다. 첫째는 camera 2의 시야에 있는 공들 중 가장 가까이에 있는 공의 색깔에 따라서 공 저장소 앞의 패널을 회전시키기 위한 myRIO신호를 보내는 것이고(이 과정을 sorting 과정이라고 자체적으로 명명하였다.) 둘째는 파란 공의 좌표가 지속적으로 감소하고, 일정한 범위 안으로 들어오면 파란 공을 픽업했다고 판단, 픽업한 공의 개수를 세고 이를 토폭 메시지의 형태로 메인 노드인 data integration으로 보내는 것이다(이 과정을 counting 과정이라고 자체적으로 명명하였다.).

3.2.2 Explanation of Sorting ‘Version 2’ code

3.3.2.1 Functions

3.3.2.1.1 dataInit()

data array의 데이터를 모두 0으로 초기화시킨다. 이때 data array는 myRIO에게 보내는 명령을 저장하는 array로써 myRIO에게는 8 byte 의 data[2] array의 형태로 신호를 보내지만 실제로 제어에 사용되는 부분은 0자리에 위치한 데이터 만이다. data[0] = 0이면 저장고 앞쪽에 위치한 패널을 공이 파란 공 저장소쪽으로 가도록 움직이며, data[0] = 1이면 저장고 앞쪽에 위치한 패널을 공이 빨간 공 저장소쪽으로 가도록 회전한다. 이때 카메라에 빨간 혹은 파란 공이 보이지 않는다면 패널은 항상 파란 공 공이 저장소쪽으로 가도록 회전된 상태로 유지된다. (data[0]=0일때의 상태와 동일하게 유지된다.)

3.3.2.1.2 camera_Callback

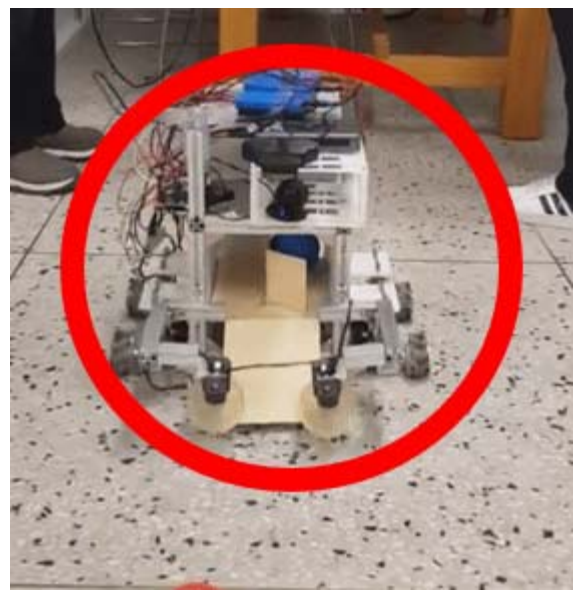
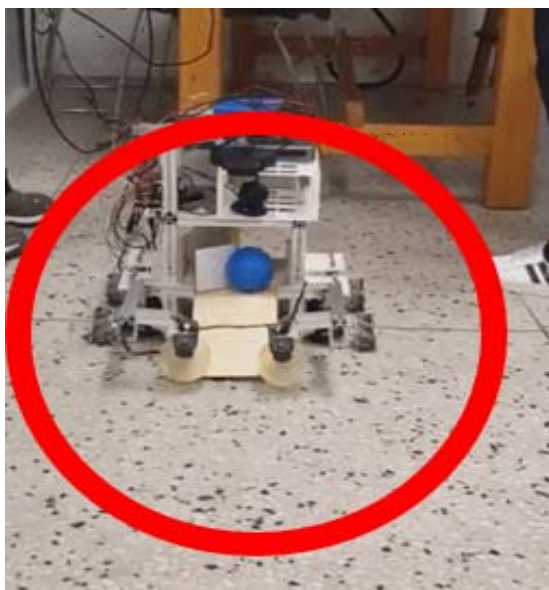
본 함수의 기능은 camera 2의 데이터를 처리하는 ball_detect_near_node의 메시지를 subscribe하면, sorting_node 내의 전역 변수에 그 데이터를 저장하는 것이다. Core_msgs 폴더의

‘position’ 메시지 형식의 size 자리의 값(camera 2에 보이는 파란 공의 개수), size2 자리의 값(camera 2에 보이는 빨간 공의 개수)를 각각 ball_number(camera 2 시야에 보이는 파란 공의 개수를 의미하는 전역 변수), ball_number_R(camera 2 시야에 보이는 빨간 공의 개수를 의미하는 전역 변수)에 저장한다. 또한 for문을 카메라에 보이는 파란 공의 개수만큼 반복하면서 전역 변수인 ball_X, ball_Y, ball_distance에 메시지에서부터 받은 파란 공의 좌표 값들을 저장한다. 이때 ball_X는 카메라에 보이는 파란 공의 픽셀 x좌표, ball_Y와 ball_distance는 카메라에 보이는 파란 공의 픽셀 y좌표이다. 역시 또 다른 for문을 카메라에 보이는 빨간 공의 개수만큼 반복하면서 전역변수인 ball_X_R, ball_Y_R, ball_distance_R에 메시지에서 부터 받은 빨간 공의 좌표 값들을 저장한다. 이때 ball_X_R는 카메라에 보이는 빨간 공의 픽셀 x좌표, ball_Y_R와 ball_distance_R은 카메라에 보이는 빨간 공의 픽셀 y좌표이다.

3.3.2.2 Main function

메인 함수 안의 코드는 크게 sorting 과정과 counting 과정으로 나뉘게 되는데, sorting 과정의 경우에는 camera 2에서 보이는 공의 좌표들을 비교해서 가장 가까이에 위치한 파란 공의 픽셀y좌표와 빨간 공의 픽셀 y좌표를 각각 저장한 후, 두 값을 비교해서 로봇과 가장 가까이에 위치한 공의

색깔이 무슨 색인지 확인하는 것으로 시작한다. for문을 파란 공의 개수만큼 돌려서 웹캠에 보이는 모든 파란 공의 y좌표를 비교하는데, for문의 i 번째 루프에서 ball_Y array에 i번째 자리에 있는 좌표가 이전에 가장 가까운 파란 공의 좌표인 near_ball보다 작다면 그 공의 y좌표값을 전역 변수인 near_ball에 저장해서 i+1번째 파란 공과 다시 비교하는 방식이다. 이 같은 과정을 빨간 공의 경우에도 반복해서, for문을 빨간 공의 개수만큼 돌려서 웹캠에 보이는 모든 빨간 공의 y좌표를 비교한다. 만약 이와는 별개로 웹캠에 빨간 공, 파란 공 중 그 어떤 공도 보이지 않는다면 저장소 앞에 위치한 패널을 파란 공 저장소 쪽으로 열어 놓는다. 앞선 과정에서 구한 near_ball(가장 가까이 위치한 파란 공의 y좌표)과 near_ball_R(가장 가까이 위치한 빨간공의 y좌표)를 비교하고, 가장 가까이에 위치한 공의 색깔이 빨간색일때는 공 저장고 앞에 위치한 패널을 빨간 공 저장소쪽으로 열고(data[0]=0입력), 가장 가까이에 위치한 공의 색깔이 파란색일때는 공 저장고 앞에 위치한 패널을 파란 공 저장소쪽으로 연다.(data[0]=1입력)



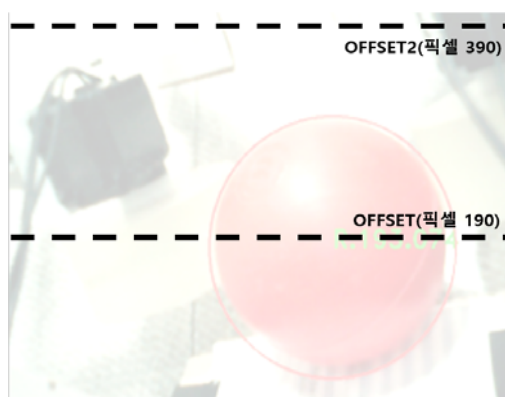
<그림 15>가장 가까운 공이 파란 공(data[0]=0) <그림 16>가장 가까운 공이 빨간 공(data[0]=1)

Sorting 과정 이후에는 counting 과정 코드로 넘어가게 되는데, 이 과정은 로봇이 현재까지 픽업한 파란 공의 개수를 count하는 것이 목적으로, 가장 가까운 파란 공의 픽셀 y좌표가 일정시간 동안 감소하고, 특정한 y좌표이하의 값을 가질 때 파란 공이 픽업되었다고 판단한다.

near_distance 값은 가장 가까운 파란 공의 픽셀 y좌표값을 저장하는 변수로 1000으로 초기화하는 이유는 for문에서 p가 0일때 무조건 ball_X[0]값을 near_distance에 저장하기 위해서 이다. 웹캠에 보이는 파란 공의 개수만큼 for문을 반복하는데, near_distance는 p-1까지 비교한 ball_Y 값중 가장 작은 값을 대입한 것으로, 이 값보다 ball_Y[p]값이 더작다면 ball_Y[p]값을 near_distance에 새롭게 저장한다. near_ball_x, near_ball_y는 가장 가까이 위치한 파란 공의 x,y좌표를 저장하는 변수이다.또한 웹캠에서 가끔 0값을 잘못 보낼 때를 대비해서 파란 공의 픽셀y좌표가 0인 값은 의사결정 과정에서 사용하지 않도록 하였다. 웹캠에 보이는 파란 공의 개수가 0이 아닐 때 이전 루프에서 저장된 가장 가까운 파란 공의 y좌표 값인 past_near_ball과 현재의 가장 가까운 파란

공의 y좌표 값인 near_ball_y를 비교함으로써 파란 공의 y좌표가 이전 루프에 비해서 줄어들었는지 확인한다. 이전의 좌표와 비교해 2픽셀이상 줄어들었을 때는 judge_constant라는 변수의 값을 증가시키는데, 이때 차이가 2픽셀 이상일 때 만 이 변수의 값을 증가시키는 이유는 웹캠에서 보내오는 정보가 공이 같은 위치에 있더라도 1픽셀이하의 오차로 계속해서 다르게 측정될 수 있

기 때문에 실제로 공이 가까워지는 것과, 정지해 있으나, 웹캠의 부정확성으로 인해서 그 좌표 값에 차이가 나는 것을 구분하기 위함이다. judge_constant는 가장 가까운 파란 공의 좌표가 얼마 동안 줄어 들었는지 척도가 되는 변수로, 이 값이 크면 클수록 오랜 시간 동안 파란 공이 가까워지고 있음을 나타내며, 이 전의 루프와 비교하였을 때 파란 공의 y좌표가 감소하였다면, 이를 증가시킨다. 이 변수를 사용하는 이유는 파란 공의 좌표가 지속적으로 감소하고 있는 경우와, 한 두 번 감소하고 증가하는 경우를 구분하기 위해서다. 또한 마지막에는 현재 가장 가까운 파란 공의 y좌표를 past_near_ball에 저장해 다음 루프 때 비교과정에서 사용할 수 있게 한다.



<그림 17> Camera 2 시야에서 OFFSET의 위치



<그림 18> Camera 2의 시야

가장 가까운 파란 공의 y좌표가 OFFSET이하 이고 영이 아닌 것이 공이 픽업되었는지 판단하는 첫번째 기준인데, OFFSET 값은 실험을 통해 결정하였다. 여러 번의 실험을 통해서 파란 공이 픽업되었을 때 가장 마지막으로 저장되는 near_distance값이 최대 190이었고 따라서, OFFSET 값은 190으로 저장하였다. 그림을 통해 알 수 있듯이 camera 2의 시야에서 픽셀 190은 롤러 바로 위에 공이 위치해 있을 때의 값이다. 또한 웹캠이 가끔씩 0.0000과 같은 신뢰할 수 없는 값을 보내는 것이 확인되어 이를 의사결정 과정에서 배제하기 위해 'near_ball_y != 0'의 조건을 추가하였다. 두번째 조건은 judge_constant 변수가 FREQUENT초과 인지 확인하는 것인데, 이는 파란 공의 y좌표가 지속적으로 감소해왔다는 것을 의미한다. 이때 FREQUENT값도 실험을 통해서 결정했는데, 파란 공이 픽업 되었을 때 가질 수 있는 최소의 judge_constant 값을 관찰하였고, 이를 바탕으로 FREQUENT값을 5로 결정하였다. 만약에 두 조건을 만족한다면 pick_w 변수에 1을 저장한다. 이와는 반대로, 가장 가까운 파란 공의 y좌표가 OFFSET값보다 크거나 웹캠 상에서 파란 공이 보이지 않을 때 pick_w 변수에 0을 저장하는데, 이때 pick_w 변수는 픽업된 파란공의 갯수를 계산하는데 사용하는 변수로서 평소에는 0값을 가지다가 파란 공이 픽업되는 순간에 1이 되었다 다시 0으로 돌아온다. 따라서 pick_w 변수 값은 루프가 돌 때마다 비교해 0에서 1이 되는 순간에 파란 공이 픽업되었다고 판단하였다. 예를 들어

'pick_w : 00000001111000000000011100000' 이면 2개의 공이 픽업되었다고 판단하는 것이다.

웹캠에서 보이는 가장 가까운 파란 공의 좌표가 OFFSET2보다 크면 past_near_ball 값을 1000으로 초기화 한다. 이는 past_near_ball에는 전 루프에서 가장 가까운 파란 공의 y좌표가 저장되기 때문에 공이 픽업되고 난 후에는 가장 가까운 공의 좌표가 OFFSET2보다 클 때 이를 새롭게 접근하는 공이라고 판단하고, 이 새로운 공의 좌표 변화를 확인하기 위해서, 이전의 값을 초기화를 해줘야만 한다. 앞서서 언급하였듯이 'past_pick_w-pick_w<0' 조건을 만족하는 경우는 pick_w가 0에서 1로 변하는 경우 밖에 없으므로 그 순간 공이 픽업 되었다고 판단, 픽업된 파란 공의 개수에 해당하는 변수(picked_ball_num)를 증가시킨다. 메시지의 data 자리에 picked_ball_num 변수 값을 저장하고 공이 픽업 되었으므로, 픽업 여부에 사용되었던 변수인 judge_constant를 초기화하

는 동시에 메시지를 publish한다. (이 메시지는 data integration node가 받는다.) 마지막으로 현재 루프에서는 pick_w값을 past_pick_w에 저장한다. 파란 공 뿐만이 아니라 빨간 공도 픽업되는 개수를 count하는데, 이 과정은 파란 공의 과정과 변수만 달라질 뿐 동일하기 때문에 따로 언급하지 않았다. 이때, 빨간 공의 픽업 여부를 판단하는 이유는 빨간 공이 픽업된 후에 잠깐의 지연 시간을 주어서, 패널이 바로 움직이지 않도록 하기 위함이다. 지연시간은 while문을 코드에 넣음으로써 줄 수 있었는데, 만약에 지연시간을 주지 않는다면 빨간 공이 저장소로 들어가는 통로로 다 들어가기 전에 파란 공 저장소쪽으로 문이 열리게 되어, 빨간 공이 끼는 현상이 발생했었다. 이를 해결하기 위하여, 잠깐의 지연시간을 통해서 빨간 공이 저장소 안으로 들어갈 수 있는 시간을 마련하였다. 즉, 전역 변수 picked_ball_num_R가 증가되는 순간에 while 문을 이용해서 0.5초 동안 지연시간을 주어 저장소 앞에 위치한 패널이 빨간 공 쪽으로 열려있는 상태를 0.5초 동안 유지해 빨간공이 저장되는 시간을 마련하는 것이다.

4. Conclusion & Future Work

Data integration node와 sorting node의 주요 특징들을 통해 결과적으로 다음 세 가지 목표를 이룰 수 있었다.

- 예측 가능한 로봇의 움직임을 통해 미션 수행 안정성을 보장할 수 있었다.
- 효율적인 움직임으로 미션을 수행하는 시간을 단축시킬 수 있었다.
- 비상 상황 대처 알고리즘으로 미션을 더 안정적으로 수행할 수 있었다.

미션 수행 안정성을 위해서 매 순간 가장 오른쪽에 있는 공을 타겟으로 선택하는 예측 가능한 로봇의 주행 방법을 사용하였다. 또한 정확성을 위해 공을 향해 직진할 때, 매 순간 공을 로봇의 중심에 위치하도록 좌우로 이동하는 직진을 사용하였다. 추가로 알고리즘을 스테이지로 나누어 비상 상황 시에 원하는 스테이지로 돌아가 다시 미션을 수행할 수 있다.

시간 단축을 위해서는 예측 가능한 경로로 반대 방향 회전을 제거하였고, 공을 분리하는 기능을 통해 빨간 공이 있어도 회피하지 않고 이동할 수 있게 하였다. 또한 후면 카메라를 이용한 후진으로 바구니에 도착하기 때문에 추가 회전을 피할 수 있었다. 마지막으로 바구니로 후진 시에 전면 카메라에 미처 잡지 못한 파란 공이 보이면 다시 pick up 스테이지로 돌아가 공을 잡는 비상 대처 알고리즘을 통해 안정성을 확보하였다.

Sorting node를 통해 저장하고 있는 파란 공의 개수를 예측함에도 불구하고 파란 공이 동시에 두 개가 들어오거나 빨간 공과 함께 갑작스럽게 들어오는 경우 예측한 파란 공의 개수가 실제와 다른 경우가 존재하였다. 이를 위해 추후에는 pick up module 입구를 바라보는 카메라를 공을 저장하는 storing module로 이동하고 카메라로 인식하는 공을 분리하는 작업을 통해 더욱 안정성을 확보할 수 있을 것이다. 또한 공을 끌고 가다가 벽에 부딪히는 상황에도 대처하기 위해 RPLidar를 이용하여 벽을 회피하는 알고리즘을 창의적 시스템 구현 2에서 추가할 예정이다.

5. Desired coding schedules

후년의 창의적 시스템 구현에서 ROS 파트를 맡는 친구들이 있다면 본 팀에서는 아래의 스케줄을 참고하기를 추천한다. 일단 3월 2,3주는 ROS 프로그램을 전반적으로 이해하고, 앞으로 진행할 코딩의 기반이 될 C++ 언어에 대해 학습하는 것을 목표로 하였다. 사실 코딩을 위한 학습량이 많은 편은 아니지만, 기계과에 3년 동안 재학하면서 코딩이라는 생소한 부분에 대해 처음 접하는

