Capstone Design 1 Final report **잘했조 / 20150314 박성빈**

1. Introduction

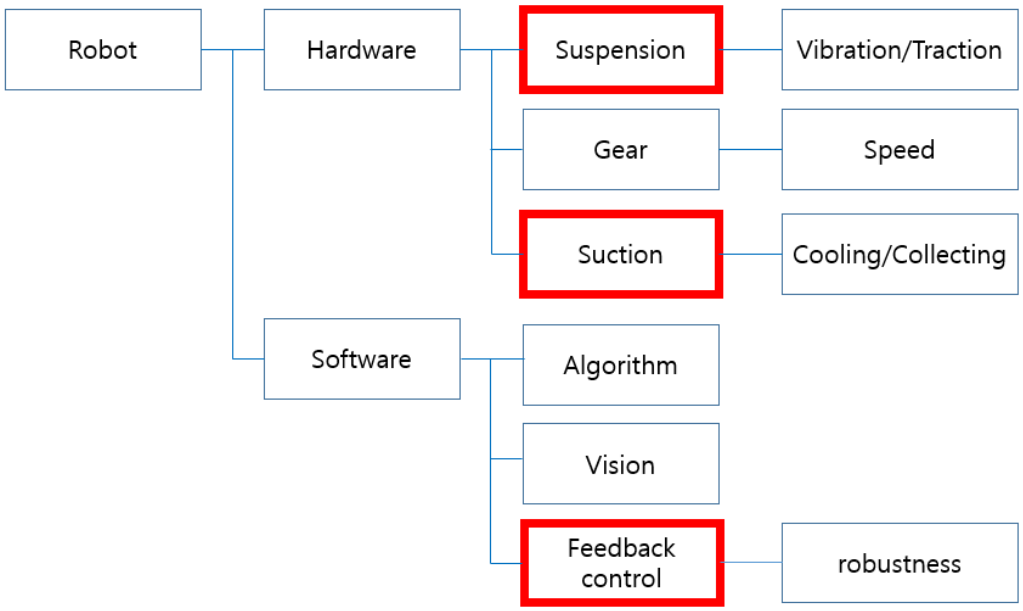
이번 창의적 시스템 구현의 최종 미션은 특정한 Map에서 빨간 공을 피해 파란 공 3개를 먹은 후 도착 지점의 바구니에 파란 공 3개를 두는 것입니다. 미션을 위해 9개의 조가 각기 다른 아이디어를 생각해냈으며, 그에 걸맞는 강점이 존재합니다. 저희 조의 로봇은 Suction motor, Suspension, PD controller라는 3가지 강점을 가지고 있습니다. Suction motor를 이용함으로써 빠르고 정확한 공 흡입 및 컨버터 부분에 열을 식히는 쿨링 작용을, suspension을 설계함으로써 울퉁불퉁한 경로나 로봇의 무게에 변화가 생겨도 문제가 없으며, 마지막으로 ROS에서 PD controller (with low-pass filter)를 구현함으로써 disturbance에 강한 robust한 system을 만들었습니다.

개괄적인 디자인은 아래와 같습니다.

로봇의 정면에는 흡입을 위한 관과 마지막에 공을 뱉어내는 것을 위한 서브 모터가 위의 관에 달려있습니다. 로봇의 아랫면에는 바퀴와 서스펜션, 그리고 그것을 뒷받침하는 프레임이 조화롭게 연결되어 있습니다. 또한, 미션 시간 단축을 위하여 3D-printing으로 직접 기어를 설계하여 기어비를 높였습니다. 로봇의 중간 부분에는 아래에서부터 배터리, NucPC, MyRIO가 3층으로 배열되어 있는데, 이 순서는 가장 무거운 것을 아래에 둠으로써 무게 중심을 가장 낮추기 위함입니다. 무게 중심은 로봇의 안정성과 직결되기 때문에, 급제동 및 급정지 시에도 PD controller를 이용하여 안정적으로 pitching 모션을 제어할 수 있습니다. 그리고 가장 뒷부분의 suction motor에는 여러 개의 관이 달려있는데, 이것은 suction motor에서 발생하는 바람을 이용하여 저희가 3학년 때 배운 열 전달 지식을 통해 열 발생률이 높은 컨버터 및 배터리, 그리고 NucPC로 Forced convection을 일으킴으로써 전반적인 로봇의 온도를 낮추기 위함입니다. "Suction을 할 때만(파란 공을 흡입할

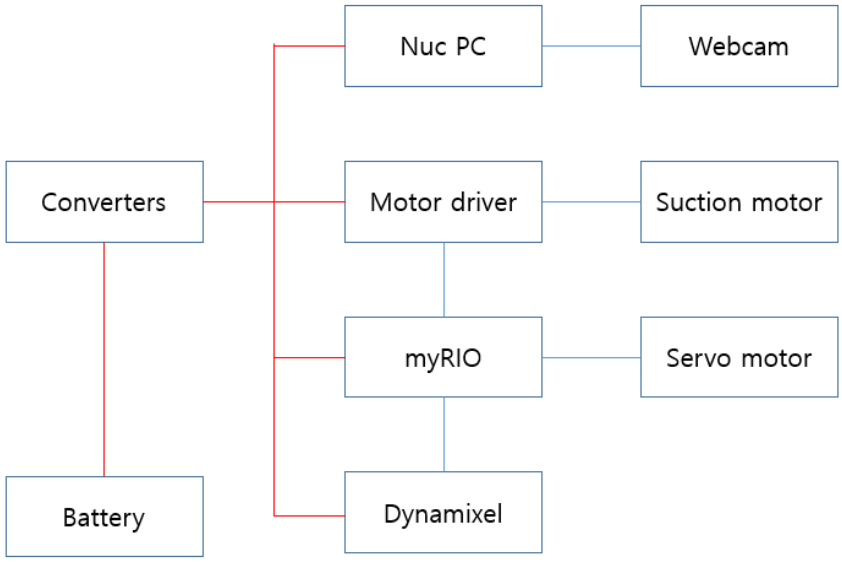
때만) 바람이 나오는데, 너무 비효율적인 것 아닌가?"라고 생각되어, 저희는 suction motor의 rpm으로 크게 1~5단계로 나누어, 파란 공을 흡입할 때에는 최고 단계인 5단계로, 그 외의 단계에서는 forced convection을 위해 3단계로 작동하게 됩니다.

이렇게 저희 로봇을 간략히 설명하였는데, 좀 더 보기 좋게 Diagram으로 표시하면 다음과 같습니다.



저희 로봇은 이렇게 다양한 강점이 있지만, 그 중에 가장 unique하면서 advantageous있다고 생각하는 것이 Suction, Suspension, Feedback control(PD controller)라고 생각합니다.

그 다음은 전체 시스템이 배터리에서부터 전력을 공급받는데, 그 전력이 배터리에서부터 어떻게 전달되는 가를 Diagram 형식으로 그려보면, 다음과 같습니다.



사실 저는 ROS 파트를 맡았기 때문에, 로봇의 전반적인 하드웨어 부분보다는 소프트웨어 부분 중 로봇의 알고리즘을 만드는데 집중하였습니다. 따라서 먼저 제가 이번 학기 동안 성취한 것들에 관하여 자세히 서술한 뒤에 나머지 파트에서 성취한 것들에 관하여 간략히 서술하고자 합니다.

2. Linux Ubuntu 16.04 Download (1st weekend)

먼저, ROS를 이용하기 위해 로봇에 장착되는 NucPC의 OS를 Linux Ubuntu로 설치하였습니다. 알고리즘의 유동성을 위하여 제 개인 노트북에도 Dual-booting 방식으로 Linux Ubuntu를 설치하였습니다. 이렇게 함으로서 제 개인 노트북으로도, NucPC로도 로봇을 움직일 수 있게 됩니다.

3. Fundamentals of ROS (2nd~3rd weekend)

처음 한 달간은 ROS의 기본이 되는 용어 및 통신 방법 등을 직접 구현해보면서 공부하였습니다.

- Node: 최소 단위의 실행 가능한 프로세서입니다.

- Package: 하나 이상의 노드, 노드 실행을 위한 정보 등을 묶어놓은 것으로, master node가 모든 node들의 executive의 역할을 합니다. 최종적으로 구현을 할 때 가장 기본이 되는 단위이며, 저희 조는 test_ws라는 이름의 workspace를 만든 뒤, 그 안에 ros_package를 설치하였습니다. 아래에서 계속 나올 catkin_make라는 것은 workspace를 build 해주는 명령어인데, 알고리즘을 수정할 때마다 이 명령어로 build 해주어야 변경 사항이 로봇의 알고리즘에 반영되게 됩니다.

- Message: 메시지를 통해 노드 간의 데이터를 주고 받을 수 있습니다. 이 메시지의 내용 및 크기는 message 파일을 만들 때 직접 설정할 수 있습니다.

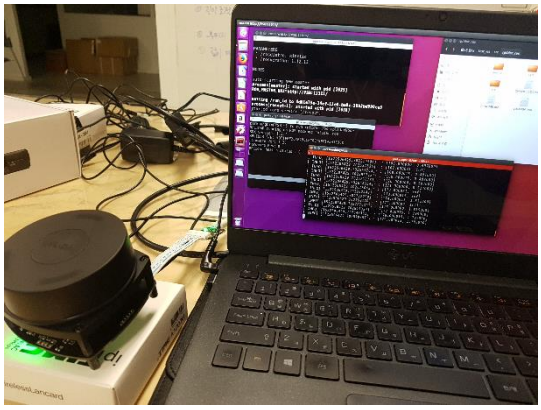
- Topic communication: 일반적인 통신 방법으로 Publisher(정보를 송신)와 Subscriber(정보를 수신)가 존재합니다. 앞으로 사용할 통신 중 대략 80% 이상을 차지할 중요한 부분입니다. Topic communication을 위해 Publisher node와 Subscriber node를 작성하고 catkin_make 후에 roscore를 실행한 뒤 topic communication을 구현하였습니다.

- Service communication: Server와 Client가 존재합니다. Server는 말 그대로 server의 역할을 하며, server를 열기만 해서는 아무 변화가 생기지 않습니다. Client는 server에 정보를 요청하는 역할을 하는데, 그 순간 server는 client가 요청한 정보를 client에 제공하는 역할을 합니다. Topic communication과 달리 1회성 message communication입니다. Topic communication과 마찬가지로 Publisher node와 Subscriber node를 작성하고 catkin_make 후에 roscore를 실행한 뒤 service communication을 구현하였습니다.

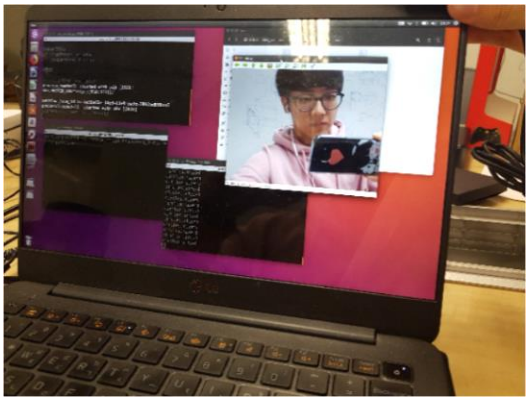
- Advanced topic communication: 여기서는 좀 더 나아가 기존의 코드를 수정하여 Publisher와 Subscriber의 역할을 동시에 하는 node를 만들어 Publisher -> Publisher + Subscriber ->

Subscriber 형태의 좀 더 복잡한 통신을 구현하였습니다.

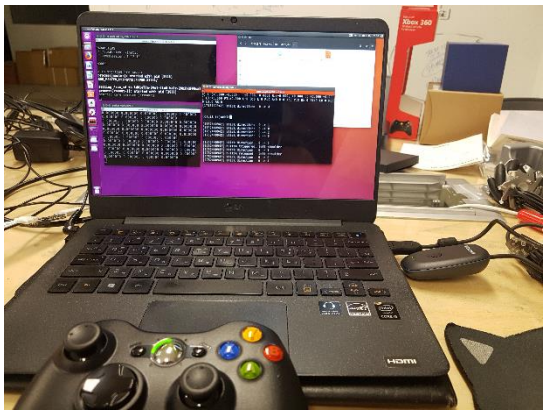
4. Application of the rplidar, webcam, and xbox-controller (4th weekend)



Rplidar 실행을 위해 먼저 컴퓨터와 rplidar를 cable로 연결한 뒤, 노트북의 주인인 나 자신이 이 cable의 데이터를 허용하겠다고 2가지 명령어를 입력하니 rplidar가 연결되었음을 확인할 수 있었습니다. Roscore를 실행한 뒤, Rplidar package에 있는 rplidar_ros 및 rplidarNode, 그리고 rplidar_ros에 있는 rplidarNodeClient를 실행하여 lidar의 데이터를 받을 수 있었습니다.



Webcam 실행을 위해 먼저 컴퓨터와 webcam을 cable로 연결한 뒤, Image_transport_tutorial package에 있는 image_transport_tutorial my_publish와 image_transport_tutorial my_subscriber를 실행하여 webcam의 데이터를 받을 수 있었습니다.



MyRIO part에서 TCP/IP 통신 작업을 마친다면 MyRIO와 NucPC가 통신하겠지만, 이번에는 두 파트 모두 준비가 되어있지 않으므로 연습 삼아 제 노트북에서 서버를 열어서 Xbox-controller를 작동시켜보았습니다. 우선, comm_tcp server_node 4000으로 서버를 연 뒤, xbox_ctrl xbox_ctrl_client를 실행하면 xbox-controller를 사용할 수 있습니다.

5. Communication between NucPC and MyRIO (5th weekend)

MyRIO의 IP주소로 설정한 뒤, MyRIO에서 서버를 열면, NucPC의 ROS가 클라이언트로 통신을 하

는 것을 구현하였습니다. 통신 상태는 Comm_tcp 노드를 통해 확인할 수 있습니다. 하지만 Xbox-controller로 바퀴를 제어하지는 못하였다. 아마도 LabVIEW 프로그램에서 data를 받는데, 약간의 오류가 있는 듯 하였다. 이것만 해결되면 우선 MyRIO와 NucPC 사이의 통신은 문제 없을 것 같다.

6. Integration of ROS and OpenCV (6th weekend)

Ball detect node를 (아마도?) 완벽히 이해하는 경지에 이르렀다. 시험 공부를 하느라 ROS를 오래 간 놓고 있었기에 통합하는데 꽤 오랜 시간이 걸렸다. 자그마치 C++ 공부를 포함하여 40시간 정도 소모한 것 같다. Ball detect node가 어떤 정보를 받아서 어떤 정보를 ROS에 제공하는지, 통신 관계 및 여러가지 함수들에 대하여 깊게 알게 되었다. 또한, 최종 골인 지점에 초록색 공을 둔다는 소식을 듣고, OpenCV 친구들이 초록색을 마킹하는 것도 만들어주었고, 빨간색 공만을 마킹하던 볼디텍 코드를 다시 전면 수정하여 세 가지 색(빨간색, 초록색, 파란색)의 공 전부를 한 번에 마킹할 수 있는 코드를 만들었다.

7. Main Algorithm (7th~8th weekend)

저희는 처음부터 크게 2가지 과정(blue ball을 먹는 과정과 최종 지점인 green ball까지 가는 과정)으로 나누어서 생각했다. 먼저, blue ball을 먹는 과정에 대해 4가지 step으로 나누었다.

(1) Step 1 - Pick blue balls

camera에 blue ball이 detect되고 있는 상황에서 blue ball까지 어떻게 접근할 것인가'에 관한 알고리즘을 짰다. 간략한 알고리즘은 다음과 같고, 실제로 구현에 성공하였다.

여러 개의 blue balls이 발견되었을 때, camera로부터 모든 blue ball의 거리를 각각 계산하여, 가장 가까운 거리에 있는 blue ball을 target으로 정한다. Target인 blue ball의 (x, y, z) 좌표를 받아온다. 먼저, z가 충분히 가까워질 때까지 직진한다. 그 다음, x로 각도를 계산하여, camera의 중심과 blue ball의 중심을 일치시키도록 시계 방향이나 반 시계 방향으로 회전한다. 이 모든 것이 만족하면, 그대로 직진하게 됩니다.

(2) Step 2 - Avoid red balls

Red ball을 피하는 Algorithm을 짜기 위해 먼저, red ball과 blue ball이 각각 있거나 없는 경우, 즉, 총 4가지의 경우를 나누어서 알고리즘을 짰다.

Red ball이 없는 경우에는 지난 주에 구현했던 알고리즘을 조금 변형하여 사용하였고, red ball이 있는 경우에는 red ball과 webcam 사이의 거리가 어느 정도인지에 따라 red ball을 피해서 blue ball을 찾을지 혹은 그냥 나두고 blue ball을 찾을지 적절한 조건을 부여하여 알고리즘을 구현했다.

(3) Step 3 - Counting

Blue ball을 먹을 때 마다 개수를 count하는 알고리즘을 구현했다. 이는 나중에 LabVIEW에서 suction motor 제어에 성공하면, suction motor에 어떤 data를 줄지 결정하는데 사용될 것이다.

(4) Step 4 - Measure present angle

Turn_CW와 Turn_CCW라는 void 함수를 제작한 뒤, 그것에 따라 angle을 계산하는 알고리즘을 구현했다. 이로써 로봇은 처음 시작한 방향으로부터 얼만큼의 각도로 회전하였는지 알 수 있고, 언제든지 그 정보를 이용할 수 있기에 유용하게 사용될 것 같다.

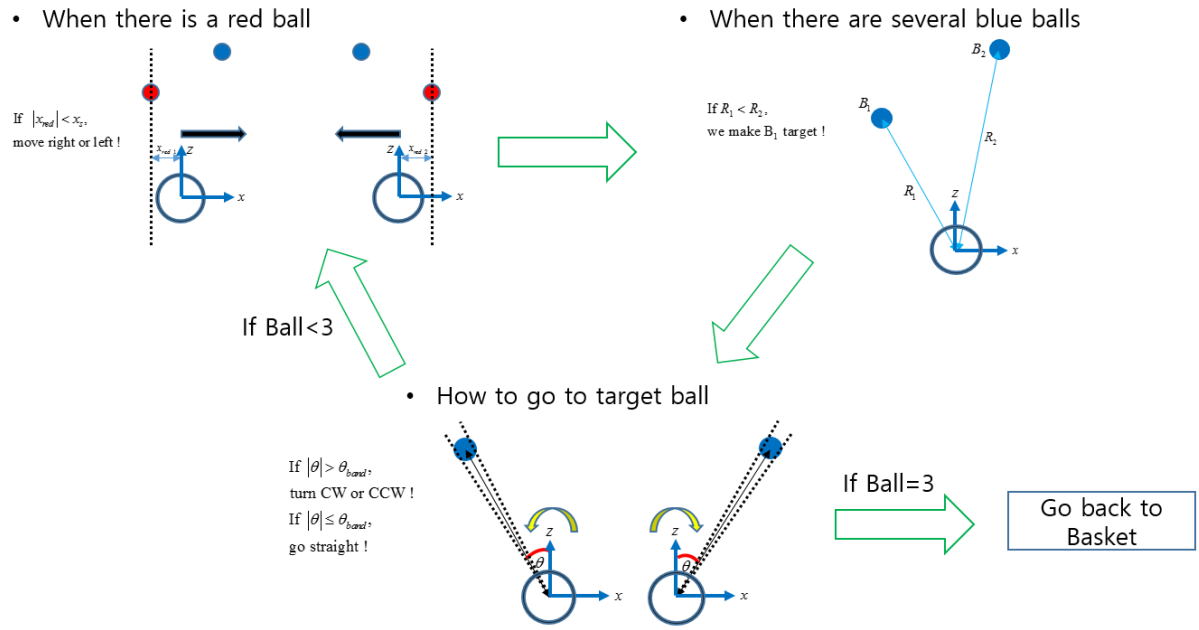
그 다음은 blue balls을 다 먹었으니, 최종 지점은 green balls로 어떻게 돌아갈 것인가에 관한 알고리즘이다. 여러 개의 시도를 해보았는데, 그 중 가장 의미 있었던 2가지만을 서술하였다.

(5) Step 5 - Go to the goal point

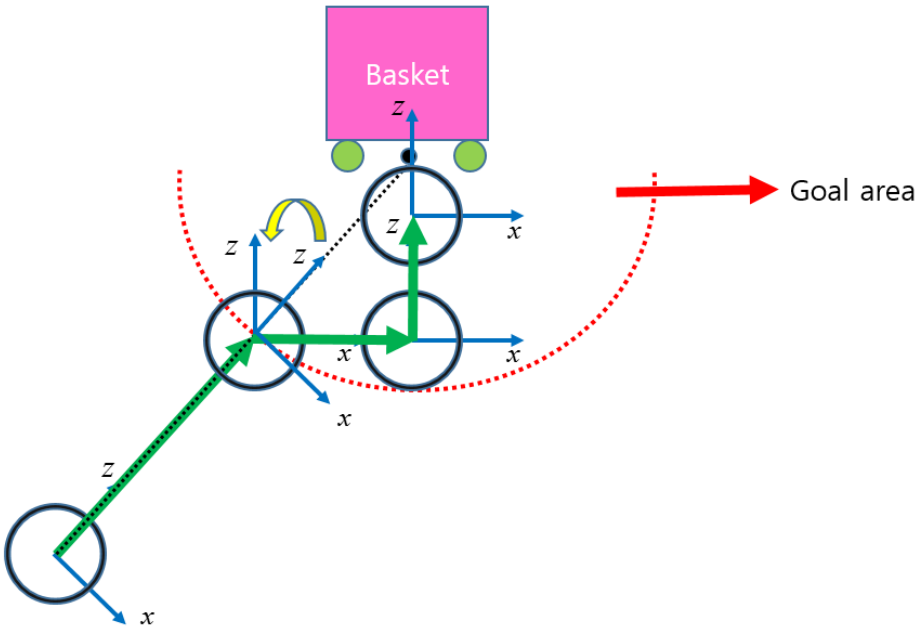
(1차 시도) Blue ball 3개를 suction 한 뒤에 어떻게 goal point로 나아갈 지에 관한 알고리즘을 만들었다. 우선 green ball이 detect 되는 경우와 되지 않는 경우로 나누어서 각각 알고리즘을 구현했다. 또한, green ball에 가까워지면서 detect되다가 안되는 경우에 어떻게 해야 바구니 중심에 정확히 도달할 수 있는지는 현재 알고리즘을 계속 생각해보고 있는 실정이다

(2차 시도) 기존의 green ball 코드는 멀리서 green ball 2개의 중점에 맞춰서 가는 것이었지만, 이것은 마치 피타고라스의 직각삼각형에서 빗변을 제외한 나머지 두 변을 통해서 골인 지점까지 가는 것으로 상당히 경로가 비효율적이라 생각했다. 그리하여 빗변으로 바로 가다가 어느 정도 가까워지면 align을 시켜서 골인 지점으로 들어가는 코드를 새로 만들었고, 여러 번의 실험을 통해 튜닝에 성공하였다.

** 글로만 설명하기엔 어려운 부분이 있기에, 이해를 돕기 위해 그림을 첨부하였습니다.



- How to go back to the basket

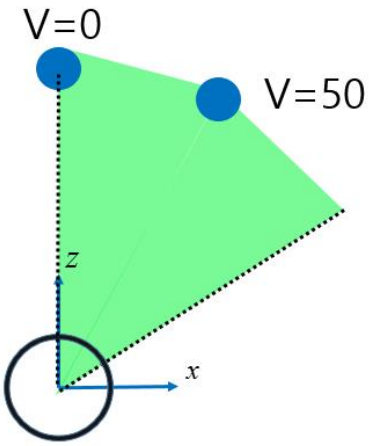


이렇게 크게 2가지 과정으로 Algorithm 부분은 어느 정도 일단락 되었습니다.

8. PD controller & Low-pass filter (9th~10th weekend)

Algorithm 부분을 마친 뒤, 어떻게 로봇의 속도를 제어할 것인가를 LabVIEW와 ROS에서 고민하다가 Time-control도 사실상 ROS에서 하고 있기 때문에, 바퀴 속도 제어도 ROS에서 하는 것이 편할 것 같아 ROS 파트가 맡았습니다. 저희 조는 크게 3가지 방법을 시도하였습니다. (1. Bang-bang control/on-off control, 2. Scheduling/open-loop control, 3. PD controller/closed-loop control)

먼저, 처음 시도했던 bang-bang control/on-off control에 관해서 입니다.



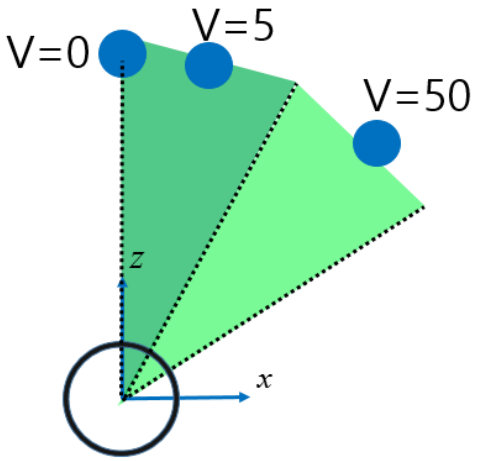
Blue ball의 각도가 특정 범위에 들어오기 전까지는 최고 속도인 50으로 회전하고, 범위 안에 들어오면 속도를 0으로 바꾸는 제어 방식입니다. 이것은 최고 속도인 50에서 0으로 감속하는데 충분

한 시간이 필요하기 때문에(Time delay), 정확한 제어가 불가능하여 문제 점이 많았습니다.

두번째로는, 속도를 50에서 0으로 바로 낮추는 것이 아닌 50에서 5로 낮추고, 그 다음 정지하는 scheduling 방식을 이용하였습니다.

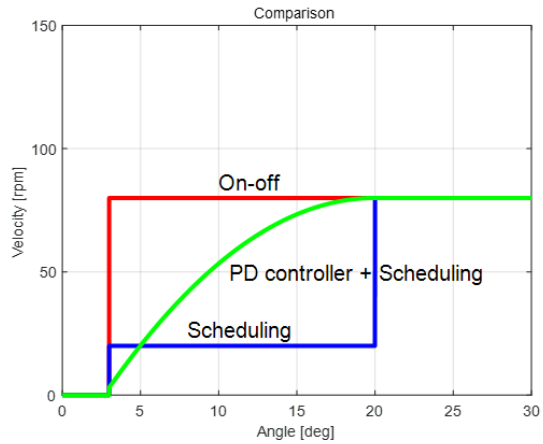
On-off control
- Fast speed -> more error
- Slow speed -> more time

Scheduling
- Slow speed when angle is small
  ->reduce error
- Fast speed when angle is big
  ->reduce time



이것은 문제점은 거의 없었지만, 좀 더 정확한 제어를 하여 정밀도를 높이고 싶었기에 아예 속도 자체를 closed-loop control로 제어하는 것이 좋다고 판단하였습니다.

그리하여 마지막은 PD controller를 직접 설계하여 closed-loop control을 구현하였습니다.



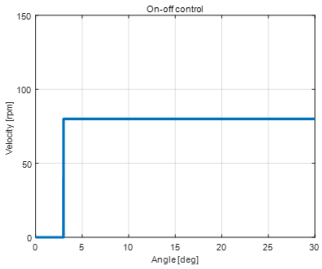
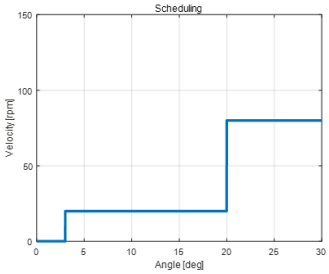
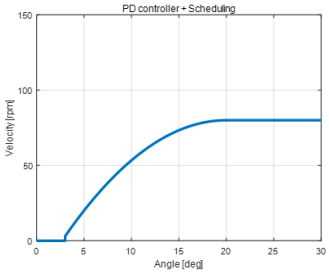
Advantage of Feedback control : Robustness

- Uneven floor
- Deformation of Hardware
- External disturbances

PD controller는 closed-loop control의 error에 비례하는 항(P gain)과 error의 미분 값에 비례하는 항(D gain)을 튜닝하여 속도를 제어하는 제어기입니다. 뿐만 아니라, error의 미분 값을 이용하면 noise가 증폭될 수 있기 때문에 이것을 방지하기 위하여 error의 미분 값에 low-pass filter 처리를 하였습니다. 결과적으로 속도를 직접 제어하니 확실히 정밀도도 높아졌을뿐더러 미션 성공률도 높아졌습니다.

이 3가지 방식을 한번에 비교하면 다음과 같습니다. (1. Bang-bang control/on-off control, 2. Scheduling/open-loop control, 3. PD controller/closed-loop control)

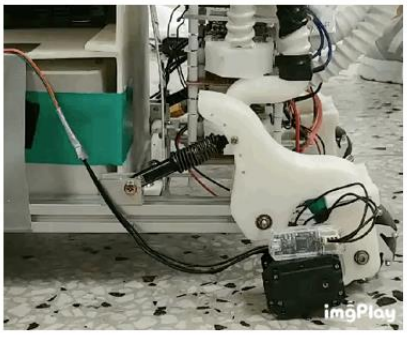
- Robot Direction control using feedback control

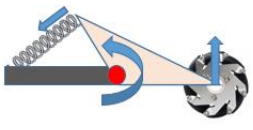
Bang-bang Control (On-off Control)	Scheduling (Open Loop)	PD Control (Closed Loop)
		
• Error due to deceleration time • Jerk due to discontinuous velocity profile	• Reduced error and Jerk • Open loop control	• Even less error and Jerk • Robust due to closed loop control

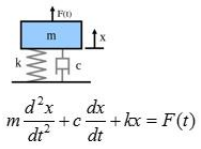
지금부터는 제 파트는 아니지만 저희 조에서 굉장히 강점이라고 생각되는 중요한 부분들을 몇 가지 소개하도록 하겠습니다.

9. Suspension

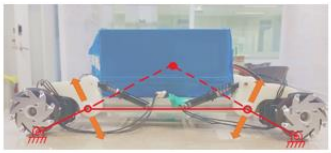
저희 조는 로봇의 구조와 무게 중심을 고려하여 서스펜션의 각도를 선정하여 설계하였습니다.



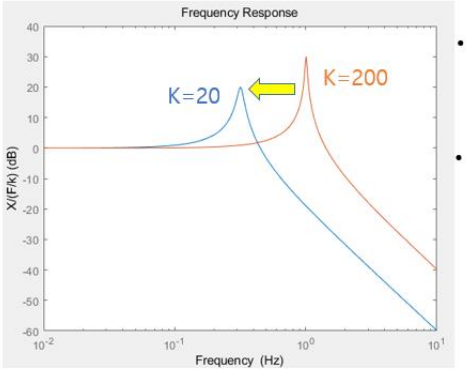




$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = F(t)$$



1. Reduce Vibration



- High frequency vibration ->vision problem
- With smaller K, less vibration in high frequency region

2. Increase Traction

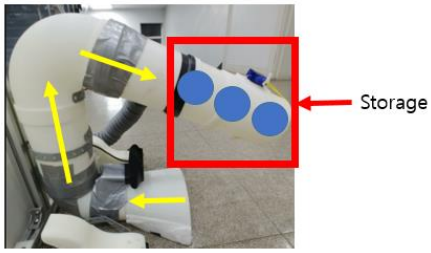
Without Suspension

With Suspension

10. Suction Process

Suction motor를 이용하여 공을 흡입하고 뱉어내는 일련의 과정들은 다음과 같습니다.

- Easy to collect balls



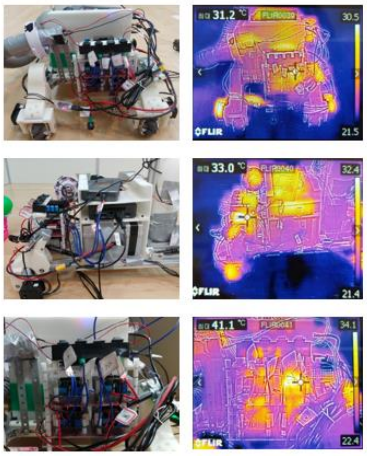
- Storage faces the front of the robot

- Accurate because it uses camera vision
- Fast because it doesn't turn around

Suction pipe 입구 부분을 넓게 설계하여 precision이 높지 않아도 blue ball을 쉽게 흡입할 수 있도록 하였습니다. 이렇게 하게 되면, red ball도 쉽게 흡입할 수 있다는 단점이 생기지만, 그것을 방지하기 위하여 camera에서 red ball을 피해야 하는 범위를 넓힘과 동시에 suction pipe 입구 부분의 옆쪽에 테이프를 붙여 red ball이 옆에 있으면 입구로 들어오지 않고 테이프에 붙음으로써 red ball을 절대 먹지 않도록 만들었습니다.

11. Heat analysis by Cooling from Suction motor

열 전달에서 배운 공식을 이용하여 배터리, 컨버터 및 NucPC 부분의 열을 계산하고, 그것에 맞게 suction motor를 가동시켰습니다. (흡입 시 suction motor 5단계, 평상 시 suction motor 3단계)



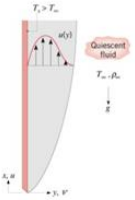
1. Free convection

$T_\infty = 20^\circ\mathrm{C},\ T_s = 40^\circ C,\ L = 0.05\mathrm{m}$

$$Ra_L = \frac{g\beta(T_s - T_\infty)L^3}{\nu\alpha} = 2.1796\times10^5$$

$$\overline{Nu_L} = 0.59 Ra_L^{\frac{1}{4}} = 12.7481$$

$$\therefore \bar{h} = \frac{\overline{Nu_D}k}{L} = \boxed{6.7565\ \mathrm{W/m^2K}}$$



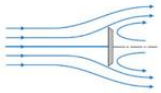
2. Forced convection

$T_\infty = 20^\circ\mathrm{C},\ T_s = 40^\circ C,\ D = 0.05\mathrm{m}$

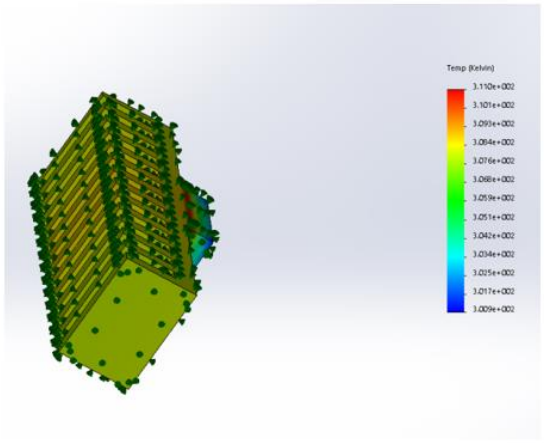
$$\mathrm{Re}_D = \frac{VD}{\nu} = 9259$$

$$\overline{Nu_D} = 0.191\mathrm{Re}_D^{\frac{2}{3}}\mathrm{Pr}^{\frac{1}{3}} = 75.1$$

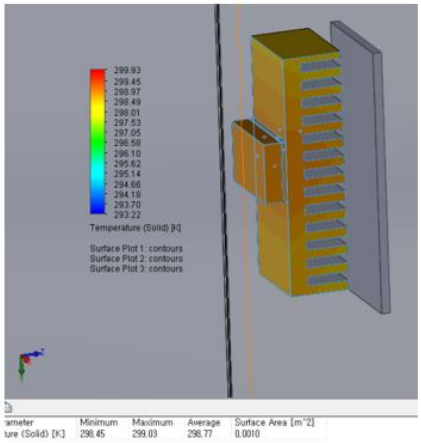
$$\therefore \bar{h} = \frac{\overline{Nu_D}k}{D} = \boxed{39.8\ \mathrm{W/m^2K}}$$



- Without Cooling ($T_{max} = 38℃$)



- With Cooling ($T_{max} = 26℃$)



흡입 시 suction motor 5단계, 평상 시 suction motor 3단계의 cooling system을 구현하였을 때, 온도가 섭씨 12도 가량 줄어드는 것을 확인할 수 있었습니다.

12. OpenCV - Multi-detection problem

OpenCV에서 다음과 같이 코드를 수정하여 Multi-detection 문제를 해결하였습니다.

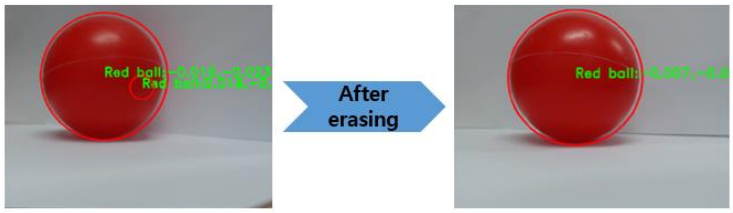
```
Point2f one, two;
float x1, x2, y1, y2;
float r1, r2;
float l;

size_t contour_b = contours_b.size();
for( size_t i = 0; i< contour_b; i++ ){
        if(radius_b[i] > iMin_tracking_ball_size){
            for(size_t j=0; j<contour_b; j++){
                for(size_t k=0; k<contour_b; k++){
                    one = center_b[j]; two = center_b[k];
                    r1 = radius_b[j];  r2 = radius_b[k];

                    x1 = one.x; y1 = one.y;
                    x2 = two.x; y2 = two.y;
                     l = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));

                    if(r1+r2>l){
                        if(r1>r2){ //!!!
                            radius_b.erase(radius_b.begin()+k);
                            center_b.erase(center_b.begin()+k);
                            contours_b.erase(contours_b.begin()+k);
                            contour_b--;
                            j--;
```

- Remove extra contour circle data with vector.erase function



Contour_size=3 Contour_size=1

13. Final Result

Demo 전에 20~30번 가량 실험해보았습니다. 그 결과로서 평균적으로 50초 정도의 시간 안에 미션을 완료하였습니다. (가장 빨랐던 Trial은 40초 만에 미션 완료하였습니다) 하지만, 정작 Demo때에는 갑자기 원인 불명의 delay가 생겨 공 하나가 suction되지 않는 상황이 발생하였습니다. 아마도 MyRIO Wi-Fi connection error로 추정됩니다. 그렇게 미션은 파란 공 2개를 주워서 가져오는데 40초가 걸림으로써 끝이 났습니다.

14. Review & Difficulties

창의적 시스템 구현을 통하여 정말 많은 것을 배웠습니다. 저의 주관적인 의견으로는 이 과목은 기계공학부에 들어와서 2년간 배웠던 고체역학, 열역학, 동역학, 유체역학, 응용전자공학, 진동공학, 재료의 가공과 이해, 시스템 모델링 및 제어, 공학설계, 자동제어 등의 지식을 전부 총 집합시켜 활용하는 아주 중요한 과목이라고 생각합니다. ROS part의 일원으로서 C++ programming을 배우지 않은 상태에서 이 미션을 성공시키는 것이 조금 많이 힘들었습니다. 한번도 C++ programming을 배워본 적이 없었기에 무작정 조교님께 질문하거나, 직접 도서관에서 책을 빌려보거나 혹은 인터넷 검색을 통해 맨 땅에 헤딩하는 느낌으로 공부해야 했기 때문입니다. 하지만 이것을 제외하고는 학문적으로 딱히 큰 어려움은 없었던 것 같습니다. 그저 제가 배웠던 지식들을 하나씩 활용하면서 "이론은 쉬운데, 적용은 어렵구나."라는 것을 절실히 느꼈고, 이러한 과정을 통해서 제 자신이 많이 성장할 수 있었다고 생각합니다.

사실 저는 학문적인 어려움보다 조모임이 가장 힘들었다고 생각합니다. 저희 조는 시작부터 1명이 드랍하는 바람에 다른 조보다 적으면 1명 많으면 2명 적은 7명의 인원으로 시작하였습니다. 각기 자라온 배경이 다르고 알고 있는 지식이 다른 7명의 사람들의 의견 차이를 어떻게 하나의 의견으로 모아갈 것인가를 비롯하여 시키지 않으면 아무것도 하지 않는 조원에 대하여 어떻게 잘 구슬려서 함께 같이 나아가야 하는지 등의 힘든 점이 많았습니다. 하지만 이런 경험을 통해 사회로 나가기 앞서 사회가 어떤 곳인지를 배우는 아주 좋은 과목이라고 생각합니다. 이것 이외에도 다양한 에피소드가 있었지만 아무쪼록 저희 조는 모두들 열심히 참여하여 성공적으로 끝낼 수 있었고, 덕분에 행복하게 이번 학기를 마칠 수 있을 것 같습니다. 감사합니다.