# 2019 Fall

# Capstone Design 2

## Final report

Team name: Mammonite

Team professor: 박해원 교수님

Teaching assistant: 곽현수 조교님

Member: 정민기, 김민경, 이인표, 김도형, 황석영,

김민석, khalifa, sultan

Submission date: DEC 20 2019

# Contents

# 1 **Individual expressions**

정민기 (software)

이번 창시구 2 를 진행하면서 팀장을 맡았고 박해원 교수님과의 미팅을 주도했습니다. 창시구 1 에서도 팀장을 맡았었기에 팀을 좀더 유연하게 이끌 수 있었다고 생각합니다. 개인 파트로는 프로그래밍을 맡아서 SLAM 부분과 Data Integration 부분의 코딩을 진행했습니다. 창시구 1 때보다 훨씬 커진 스케일의 정보를 다루었기에 코딩 실력이 늘었다고 생각합니다. 창시구는 고난과 역경을 헤쳐나가는 방법을 익히는 좋은 수업입니다. 앞으로 많은 수강생이 있었으면 좋겠습니다.

김민경 (software)

창시구 1 에서는 설계를, 창시구 2 에서는 소프트웨어를 담당하면서 두 분야 모두를 탐구해볼 수 있어 좋은 경험이었습니다. 특히 창시구 2 에서는 머신러닝에 대해 기본적인 내용을 배우고, 실제 로봇에 접목시켜볼 수 있어서 정말 많은 것을 느낄 수 있었습니다. 기계공학과로서 멀게만 느껴졌던 소프트웨어에 한걸음 다가갈 수 있었지만, 다른 한편으로는 머신러닝이 가지는 한계와 단점도 알 수 있었습니다. 어느 것이든 만능은 없었습니다. 엔지니어에게는 상황에 맞춰 적절한 기술을 사용할 수 있는 능력이 중요함을 느꼈습니다.

이인표 (software)

Software 파트를 맡으면서 머신 러닝에 대해 많은 것을 배울 수 있었다. 지난 학기에는 LabVIEW 를 맡아서 코딩을 거의 할 기회가 없었는데 1 학년 이후로 처음 코딩을 해보는 것이라 생소하지만 재미있는 경험이었다. Software 중에서 머신 러닝 파트를 맡아서 머신 러닝에 대해 공부해보고 기계공학과에서 머신 러닝이 어떤 식으로 사용될 수 있는지 알 수 있었습니다. 하지만 머신 러닝이 가진 단점 또한 알 수 있었고 창시구 1 에 비해 더 머신 러닝으로 하는 작업이 더 어렵다고 느꼈습니다.

김도형 (hardware)

solidworks 파트를 맡았다. 창시구 마지막 시현때 성공을 해서 기분은 좋지만, 그래도 조금 더 분석을 정확히 해서 서스펜션을 구현했다면 더 좋았겠다는 후회가 더 많이 든다. 생각했던 것보다 많은 변수가 있어서 예상보다 훨씬 빨리 초안을 마무리해야한다는 교훈을 얻었다. 또, 제작실을 24 시간 쓸 수 있었으면 좋겠다. 부품은 만드는 게 사는것보다 무조건 낫다. 3D 프린팅을 여러번 하는 것보다 설계를 처음에 정확히 하는게 백배 낫다. 그러나 설계를 하다가 막힌 상태에서 하루를 버리는 것보다는 프린팅을 해보는 게 나을 때도 있다. 정확한 근거로 설계를 하되, 예상하지 못한 변수를 발견하기 위해서 실패를 여러번 해봐야 한다. 아이러니하지만 이렇게 말할 수밖에 없다. 정말 많은 시간을 쏟았지만, 많은 시간을 더 효율적으로 쏟지 못해 못내 아쉽다. 그래도 다음에 이런 수업이 있다면 또 듣고 싶다. 실전 경험은 정말 중요하다. 배우는 게 많았다.

김민석 (hardware)

창시구 1 을하고 4 년만에 창시구 2 를 듣는데 처음엔 너무 많이 바뀌어있어서 놀랐습니다. 예전에 비해 과목이 좀 더 체계적으로 바뀌어 좋았습니다. 좋은 교수님, 조교님, 팀원들과 함께해서 더 뜻깊은 경험이었습니다.

황석영 (hardware)

Hardware part 를 진행하며 시간에 쫓기는 일이 많았다. 설계를 바꿀 경우 나사를 포함한 작은 부품까지 모두 바뀌는 경우가 생기는데 이를 주문하는 시간까지 고려하면 1 주일 이상의 시간이 걸린다. 설계를 고칠 때에는 욕심내서 새로운 설계를 하는 것보다 최대한 주어진 부품들을 이용할 수 있는 설계가 필요하다는 것을 깨달았다

Khalifa (hardware)

Comparing to Capstone Design I, I think Capstone Design II was a joyful experience. Of course, I faced difficulties during the semester, but our teamwork managed to cover all these difficulties.

Whether it is a required course or not, Capstone Design II course was the most needed course at that moment to test my abilities in the hardware part and to enhance them as well. At first, I had the feeling of lacking the confidence for the work, but I just did it. If I continued to feel that way, I am sure that at the end, I was going to drop the course at that time, but I didn't give up, I just proceeded to the work, and thanks to my group members, they were encouraging me. I really improved my skills in using Solid-works, and the hand-working as well. Capstone design II is not a course about making a system with required conditions, but a course that teaches you how to manage your time and it teaches you how to make decisions. During the semester, we faced many unsolved problems in the hardware part, so instead of trying to solve them, we just eliminated the root cause of the problem and replaced it with something else. That kind of step made us save time for us, the hardware team, and it made us save time for the software team as well. Sometimes, trying to solve a problem is a problem itself, so we had to make the emergency decisions during that time.

Sultan（OpenCV）

'I am really glad to be able to experience Capstone Design II course during my undergraduate studies in KAIST, even though I am a double major student from Electrical Engineering and Mechanical Engineering. Continuing from what had been left off from Capstone Design I, even though its a tough experience and more challenging than what we' ve done in Capstone Design I, I have learned a lot from this course. For instance, this is my very first chancen in understanding how various Machine Learning techniques that will be able to control the decisions that should be made by our module is made in general. In addition to that, this class provides me a chance in improving my public speaking skills by the two design reviews, which requires myself in going through and explaining all things that had been covered for the entire semester in a certain period of time. Furthermore, I also made some new Korean friends that are nice and helping me strive for improvement during the whole semester taking this course.'

# 2 Software

## 2.1 Introduction

창의적 시스템 구현 2 에서의 미션은 미로를 통과하여 파란 공 3 개, 빨간 공 3 개를 줍고 다시 미로를 통과하여 고양이가 그려져 있는 바구니에는 빨간 공, 강아지가 그려져 있는 바구니에는 파란 공을 내려놓는 것이다. 이 미션을 수행하기 위하여 우리는 SLAM, DQN, CNN 을 이용하였다. 먼저 전체적인 시스템에 관한 설명을 하고 각각의 파트에 대한 코드를 설명할 것이다. 그 후로는 우리 시스템의 장단점 등과 다음에 이 수업을 듣게 될 학생들을 위한 desire schedule 을 제시하겠다.

## 2.2 Overall System

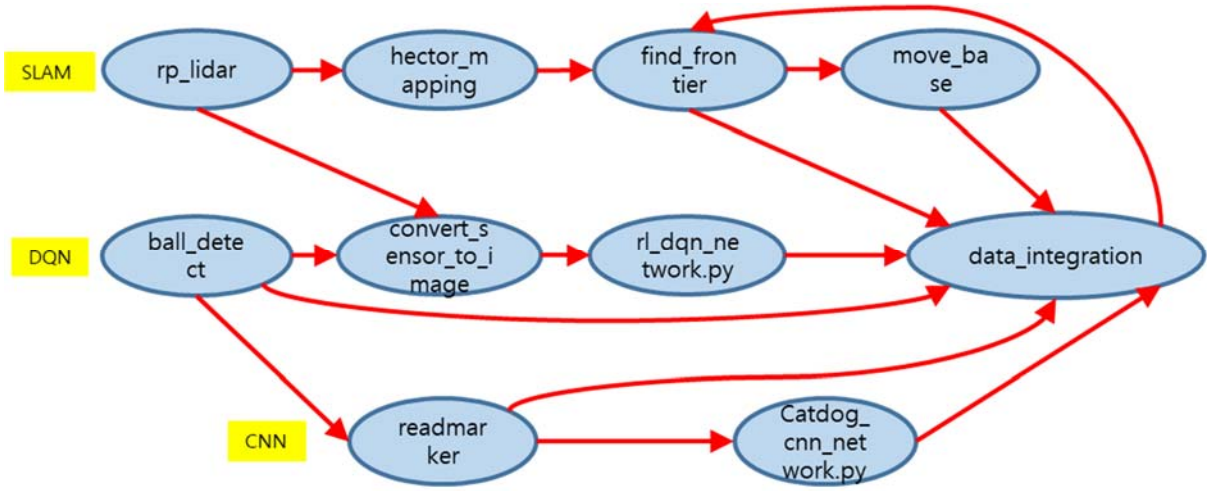



**Figure 1**

위의 Figure 1 은 우리 시스템의 전체 노드와 시스템의 흐름을 나타낸다. 크게 SLAM, DQN, CNN 파트로 나눌 수 있다.

먼저 SLAM 파트를 살펴보면, rp_lidar 에서 LiDAR 를 이용한 scan 데이터를 퍼블리시하고 hetor_mapping 에서는 그 데이터를 받아 맵을 만든다. Find_frontier 에서는 맵 데이터를 받아 goal position 을 잡고 그 값을 move_base 로 퍼블리시한다. move_base 에서는 goal position 까지 가기 위한 path planning 을 하고 그 path 를 따르기 위한 x,y 방향의 linear velocity 와 z 방향의 angular velocity 를 cmd_vel 이라는 메시지에 담아 data_integration 노드로 퍼블리시 한다.

DQN 파트를 살펴보면, ball_detect 에서 웹캠 이미지를 얻고 그 안에서 공의 위치와 색깔 데이터를 convert_sensor_to_image 로 보낸다. convert_sensor_to_image 에서는 scan 데이터도 받아 벽의 위치와 공의 위치를 하나의 DQN input 이미지로 만든다. 특이한 점은 멀리 있는 공과 가까이 있는 공을 보기 위해 카메라 2 대를 앞쪽에 쓰기 때문에 카메라 2 개의 이미지와 LiDAR 데이터를 하나의 이미지로 합쳤다는 것이다. rl_dqn_network.py 에서는 그 이미지를 받아 이미 학습해놓은 network 로 로봇의 action 을 정하고 이를 data_integration 노드로 퍼블리시 한다.

CNN 파트를 살펴보면, readmarker 에서는 ball_detect 에서 웹캠 이미지를 받아 고양이와 강아지 주위에 있는 QR 코드를 인식하고 그것의 좌표를 data_integration 으로 보내고, QR 코드 안에 있는 이미지는 catdog_cnn_network.py 로 보낸다. catdog_cnn_network.py 에서는 들어온 이미지가 고양이 또는 강아지인지 판단하여 그 결과를 data_integration 노드로 보낸다.

data_integration 노드에서는 각 파트에서 보낸 메시지를 받아 실제로 모터 구동을 할 수 있게끔 만들었다.

## 2.3 SLAM

SLAM 은 Simultaneous Localization And Mapping 의 약자로 움직이면서 맵을 만들고 자신의 위치를 파악하는 것을 동시에 진행한다. 우리는 이 알고리즘을 이용해 맵을 만들고 위치를 파악했다. hector_mapping 에서 이 과정을 진행한다. 맵이 나오면 목표지점을 잡고 경로를 찾아 움직이게 하는 것이 우리의 목적이었으며 임의로 미로를 만들고 로봇을 움직이게 하였을 때 만들어진 맵은 다음과 같다.

**Figure 2**

find_frontier 에서는 goal position 을 정하는데, 주어진 코드에서는 맵이 밝혀지지 않은 부분으로 goal position 을 정하게 되어있었다. 하지만 이는 왼쪽으로 미로를 빠져나갈지, 오른쪽으로 나갈지 불확실하기 때문에 우리는 DQN 으로 넘어가기 직전의 좌표를 실험적으로 구하여 goal position 의 좌표를 찍어주었다. 또한, 맵에서 시작위치가 (0,0)이기 때문에 공 줍는 미션을 마치고 돌아올 때도 좌표를 이용하여 goal position 을 정했다.

```
if (ioio == 0){
  goal_position.header.frame_id = "map";
  goal_position.header.stamp = ros::Time::now();
  goal_position.pose.position.x = 2.0;
  goal_position.pose.position.y = -0.95;
  goal_position.pose.orientation.w = 1.0;
  goal_pub.publish(goal_position);



  msg2.data = 0;

  pub1.publish(msg2);
}
```

**Figure 3**

위의 코드는 처음시작해서 미로를 빠져나갈 때 goal position 을 정해주고 이를 퍼블리시하는 코드이다. if 문에서 ioio 는 data_integration 에서 받은 메시지 데이터이고 ioio=0 이면 아직 제자리로 돌아가는 단계가 아니라는 것이다. 즉, 공을 아직 다 줍지 못했을 경우이다. msg2 는 제자리로 돌아 왔는지에 대한 정보를 담은 메시지이다. msg2.data=0 이면 제자리로 돌아오지 않았을 경우이며, msg2.data=1 이면 제자리로 돌아왔을 경우이다. 이는 data_integration 으로 퍼블리시 되어 SLAM 을 끝내고 CNN 파트를 시작하게 만들어준다.

```
else if (ioio == 1 && koko == 0){
  goal_position.header.frame_id = "map";
  goal_position.header.stamp = ros::Time::now();
  goal_position.pose.position.x = 2.7;
  goal_position.pose.position.y = -0.4;
  goal_position.pose.orientation.w = 1.0;
  goal_pub.publish(goal_position);

  if((pose_ptr_->pose.position.x - 2.7)*(pose_ptr_->pose.position.x - 2.7) + (pose_ptr_->pose.position.y + 0.4)*(pose_ptr_->pose.position.y + 0.4) < 0.06){
    koko = 1;

  }

  msg2.data = 0;
  pub1.publish(msg2);

}
```

**Figure 4**

위의 코드는 ioio=1 이고 koko=0 일 때 실행되는 조건문이다. ioio 가 1 이기에 공을 모두 줍고 제자리로 돌아가는 과정이고, koko 는 우리가 만든 변수이다. 원점에서 로봇이 멀리 떨어져 있을 때 원점으로 가는 경로를 못 찾는 문제가 있었다. 우리는 이를 해결하기 위해 공을 다 주운 위치에서 바로 원점으로 가는 경로를 찾는 것이 아니라 중간지점을 만들어 거기로 먼저 간 후 거기에서 원점으로 돌아오는 경로를 찾게 했다. koko 가 0 이면 공을 다 주운 위치에서 중간지점으로 가는 goal position 을 퍼블리시 한다. 우리는 실험적으로 적절한 중간지점을 구했고 이 것의 좌표는 (2.7, -0.4)이다. 안쪽의 if 문은 중간지점에 도착했는지를 확인한다. 로봇의 현재 position 을 확인할 수 있기에 로봇과 중간 지점의 x 와 y 값 차이를 구한 후 제곱의 합을 구하여 0.06 의 범위 이내에 들어오면 중간지점에 도착한 것이라고 인식하게 만들었다. 만약 도착했을 시 koko=1 로 만들어 이제는 원점으로 가는 경로를 찾게 한다. ioio=1 & koko=1 인 조건문도 위의 경우와 같고 다만 goal position 을 원점으로 만들었다. 또한 goal position 에 도착했을 시 koko=2 로 만들어서 else 가 실행되게 한다.

```
else{

  msg2.data = 1;
  pub1.publish(msg2);
}
```

**Figure 5**

else 가 실행된다는 것은 로봇이 원점에 돌아왔다는 것이기에 이제는 msg2.data=1 로 만들어 data_integration 에 퍼블리시 한다. data_integration 에서는 이 데이터를 받아 SLAM 을 끝내고 CNN 파트를 실행하게 된다.

move_base 에서는 find_frontier 에서 퍼블리시하는 goal position 의 데이터를 받아 경로를 만들고 이를 수행하기 위한 x,y 방향의 linear velocity 와 z 방향의 angular velocity 를 cmd_vel 이라는 메시지에 담아 data_integration 노드로 퍼블리시 한다. 여기서 경로를 만들고 cmd_vel 를 정하기 위하여 여러 파라미터를 조정할 수가 있다. find_frontier 패키지의 config 폴더를 들어가면 여러 파일들이 있는데 여기서 우리 로봇의 크기, x,y 방향의 max 또는 min 속도 등을 설정할 수가 있다. 우리는 특이하게 y 방향 속도를 없앴고 로봇이 곡선으로 움직이게끔 만들었다. 이는 실험을 통해 조금씩 값을 변경하였고 결국 우리는 최적의 값들을 구할 수 있었다.

## 2.4 DQN

창시구 1 과 창시구 2 의 차이점 중 하나는 공을 pick-up 하는 방식이다. 창시구 1 에서는 공을 pick-up 하는 모든 상황에 대한 알고리즘을 프로그래밍했다면, 창시구 2 에서는 이를 머신 러닝으로 해결한다. DQN 은 이 pick-up 에 사용되는 머신 러닝 기법 중 하나로 Deep Q-Network 의 약자이다. 강화 학습의 일종으로 창시구 2 에서는 로봇이 스스로 공을 pick-up 하기 위한 행동을 하도록 만드는 것이 DQN 의 목표이다. 이번 프로젝트에서 DQN 학습을 위해

제공되는 파일에 대한 기본적인 설명과 가장 많이 수정할 파일인 simulator.py에 대한 설명과 이를 어떤 식으로 수정했는지에 대해 설명하고자 한다.

### 2.4.1 dqn_learning_code

DQN 학습을 위해 제공되는 파일은 dqn_learn.py, dqn_model.py, Network_test.py, simulator.py, 그리고 main.py이다. 우선 dqn_model.py는 DQN에서 neural network를 사용하기 때문에 이 neural network를 정의하는 함수이다. 학습할 때는 거의 수정하지 않는 파일이다. Dqn_learn.py 은 DQN 학습 알고리즘에 대한 파일로 Q function, learning rate 등 학습에 사용되는 변수와 이 변수를 이용한 학습에 대한 함수 등이 정의되어 있다. Dqn_model.py와 비슷하게 거의 수정하지 않지만 만약 DQN의 output을 추가하고 싶다면 추가적인 프로그래밍이 필요하다. Main.py는 dqn_learn.py, dqn_model.py, simulator.py 등 학습에 사용되는 파일들을 이용하여 학습을 진행하는 파일이다. 다른 파일들을 수정한 후 학습을 진행하기 위해서는 이 main.py를 실행해야 한다.

DQN에서 가장 많이 수정할 코드는 앞서 말했듯이 simulator.py 이다. Simulator.py는 말그대로 DQN 학습을 위한 simulator를 만드는 파일이다. 시스템의 시야, 크기, 공을 pick-up하는 region, 맵의 크기, 공의 개수, reward 기준, action 등 다양한 파라미터와 함수를 정의한다. 기본 코드에서 정의된 파라미터 값이 실제 환경과 다르고 각 팀의 로봇의 pick-up 방법이나 크기 등이 모두 다르기 때문에 그 로봇에 맞게 파라미터를 수정해야 하기 때문에 가장 많은 수정이 필요하다.

### 2.4.2 Simulator

#### 2.4.2.1 기존 simulator.py

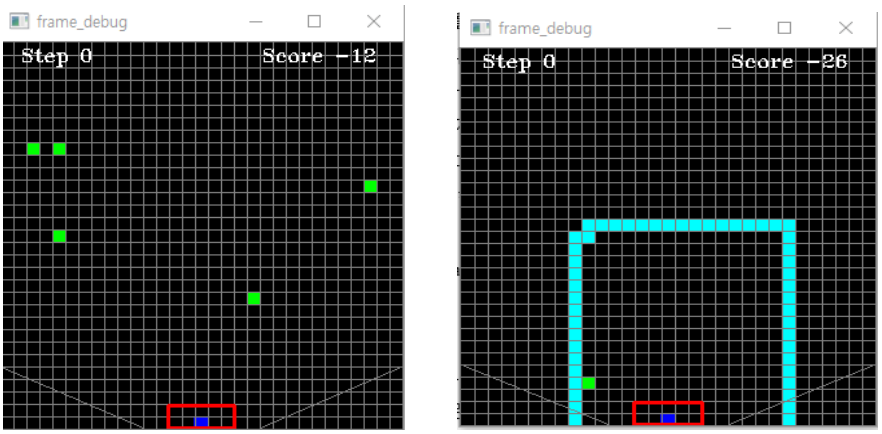



**Figure 6** 기존 simulator.py로 생성한 map

주어진 코드에서 map은 위 Figure 6처럼 벽의 크기, 개수 모두 랜덤하게 생성한다. 공 또한 마찬가지로 최대 20개의 공이 랜덤한 위치에 생성된다. 빨간색 사각형 부분이 차체라고 가정하고 그 위 4칸이 reward region으로 정해져 있다. Action의 수는 상하좌우, 대각선, 양방향 회전까지 총 10개로 이루어져 있다. 이외에도 로봇의 움직임이나 센서, 주변 환경들에 대한 다양한 파라미터들이 정해져 있다. 이 simulator.py로 만든 맵이 실제 환경과 다르다고 생각했기 때문에, 우리 조는 실제 환경과 비슷한 맵을 만들고 로봇의 크기나 pick-up하기 위한 방식 모두 최대한 로봇에 맞춰 프로그래밍 했다.

#### 2.4.2.2 수정한 simulator.py

Simulator.py를 최대한 실제 환경과 비슷하게 구현하면서 여러가지 가정을 했다. 우선 공 하나가 한 칸이므로 한

칸을 7cm로 잡았다. SLAM에서 미로를 왼쪽과 오른쪽으로 탈출할 수 있으므로 두가지 상황을 모두 가정한다. 이러한 가정을 두고 simulator.py를 수정하였다.

벽의 개수나 크기를 임의로 만드는 것이 아닌, 미로를 탈출한 시점으로 고정했다. 기본 코드에서 랜덤하게 벽을 생성하는 것을 수정하여 약 3m x 3m 크기의 벽을 생성했다.

```
while current_ball<6:
    cx = 5 + int(1.0*random.random()*(map_param["height"]-2*trans_scale)+2*trans_scale)
    if mk_prob >= 0.5:
        cy = int(random.randrange(-70, -5))

    else:
        cy = int(random.randrange(0,65))

    if current_ball < 6:
        insert = True
    else:
        insert = False

    if cx > 66:
        insert = False
    if cx < 25:
        insert = False


    for ball in self.balls:
        x_range = cx - ball[0]
        y_range = cy - ball[1]
        x_sq = math.pow(x_range,2)
        y_sq = math.pow(y_range,2)
        distance = math.sqrt(x_sq+y_sq)
        if distance < 16:
            insert = False
```

**Figure 7** 공을 생성하는 code

공의 경우, 수는 빨간 공 3 개와 파란 공 3 개, 총 6 개이다. 공과 공, 공과 벽 사이의 거리는 최소 50cm 이상 떨어져 있으므로 이런 부분까지 모두 고려하였다. 공과 벽의 좌표 cx 와 cy 를 계산할 때, 한 칸을 1 로 계산하면 안되고 trans_scale 같은 변수들을 고려하여 계산하여야 한다. 위 Figure 7 이 공을 생성하는 부분이다.

로봇의 경우에도 실제 크기는 약 40cm x 40cm 이기 때문에 이보다 더 크게 simulator 상에서 5x5 의 크기로 만들었다. Reward region 또한 더 엄격한 기준을 잡기 위해 로봇 위 가운데 한 칸, 오른쪽 한 칸으로 설정했다. 카메라의 시야각도 실제 시야각 56 ° 에서 더 낮춰 50 ° 로 설정하였다.

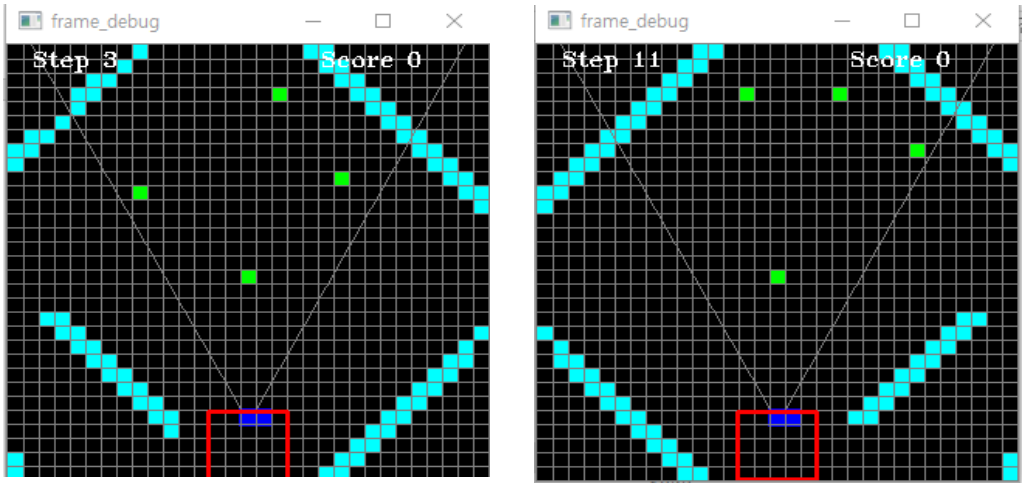



**Figure 8** 수정한 simulator.py 로 생성한 map

위 Figure 8 이 이러한 기준들로 구현한 debug frame 이다. 빨간색 사각형이 로봇, 파란색 칸이 reward region 을 의미한다.

Action 의 경우 실험을 통해 그 개수와 종류를 바꾸어 나갔다. 우선 다른 것은 수정하지 않은 상태에서 기존 10 개의 action 으로 학습한 결과와 y 방향 움직임을 제거하여 직진, 후진 그리고 회전 총 4 개의 action 으로 학습한 결과를 비교한 결과, best reward 가 큰 차이가 없다고 판단되어 y 방향 움직임을 없애는 방향으로 학습을 진행했다. y 방향 움직임을 배제하게 된 이유는 우리 조의 로봇은 서스펜션이 완벽하지 않기 때문에 횡 방향 움직임이 불안정하다고 판단되기 때문이다. 그래서 처음에는 Figure 8 과 같은 map 에 4 개의 action 으로 학습을 진행했으나, 후진이 있는 경우 후진이 필요 없는 경우에도 후진을 하는 경우가 발생하고 실제로 후진이 필요한 상황이 거의 없다고 판단되어 후진까지 제외하여 직진과 회전 총 3 개의 action 만 정의하였다. 3 개의 action 으로 학습한 경우 Network_test.py 로 테스트하면 만족스러운 결과를 보이지만, 이를 실제 로봇으로 옮겨 구동할 경우 문제가 발생했다. 가장 큰 문제는 공과 일정 거리 이상 멀리 있을 경우 직진하지 못하고 회전만 반복하는 것이었다. Reward region 을 2 칸으로 비교적 작게 설정하니 simulator 상에서 공을 최대한 가운데로 align 을 하고, 실제 로봇에서도 정확하게 align 을 맞추려 하는데 카메라로 받은 공의 정보가 부정확하다 보니 이처럼 계속해서 align 을 반복하는 문제가 발생한다. 이 문제를 해결하기 위해 여러가지 방안을 구상한 결과 회전할 때에도 직진하도록 만들었다. 공과 가까이 있을 경우에는 회전만 반복하는 문제가 발생하지 않지만 멀리 있을 경우에만 문제가 발생하기 때문에 회전할 때에도 앞으로 가도록 만들었고 실제 환경에서도 잘 작동하는 것을 확인할 수 있었다.

```
for obstacle in obstacles_temp:
    if (abs((1.0*obstacle[0])/trans_scale-3)) <=1.5  :
        if abs(1.0*(obstacle[1])/trans_scale) <= 2.5 :
            enable_move = False

for ball in balls_temp:
    if (abs((1.0*ball[0])/trans_scale-3)) <=1.5 :
        if abs(1.0*(ball[1])/trans_scale) <= 2.5 :
            enable_move = False
```

**Figure 9** 공, 벽과의 충돌을 구현하는 code

위 Figure 9 은 로봇이 공 또는 벽과 충돌하는 것을 구현했다. 기본 코드에서는 벽과 부딪칠 경우에만 움직이지 못하게 만들었다. 우리 팀은 공과 부딪치는 상황을 만들지 않기 위해 공과 부딪칠 경우에도 enable_move 라는 변수를 False 로 만들어 주었다. 그리고 enable_move 가 False 일 경우 reward 에 -5 를 더하여 최대한 부딪치지 않도록 학습하였다.

```
for i, ball in enumerate(self.balls):
    cx = int(round(1.0*ball[0]/trans_scale))
    cy = int(round(1.0*ball[1]/trans_scale))
    if cx == 5:                                    # reward region에 앞에 공이 들어왔을 때 + reward를 준다
        if cy == 0 or cy == -1:                    # x범위는 실제 시스템보다 약간 작게 만들어 실제 환경에서 정확하도록 만들었다.
            reward = reward +100+50*self.ball_count  # 기본 reward는 500으로 하되, 공을 추가로 먹을 때마다 100점씩을 추가로 reward로
            self.ball_count = self.ball_count + 1    # 넣어 공을 더 많이 주울수록 reward가 크다는 것을 학습시켰다.
        else:
            balls_temp.append(ball)
    else:
        balls_temp.append(ball)

balls_inscreen = []
for ball in balls_temp: # 공이 시야 밖으로 나갔는지 확인하여 시야안에 있다면 balls_inscreen에 이를 추가한다.
    if ball[0] >= ball_blind_ratio * (abs(1.0*ball[1])+1+ball_blind_bias)+5\
        and abs(1.0*ball[1]) <= map_param["center"] and abs(1.0*ball[0]) < map_param["height"]:
        balls_inscreen.append(ball)
```

**Figure 10** DQN 의 reward criteria

Figure 10 은 DQN 에서 reward 를 어떻게 주는지에 대한 부분이다. 기본 코드에서는 reward 가 공을 pick-up 하는 위치에 따라 +7, +3, +1 이었다. 기존에는 map 의 크기는 비교적 작고 공은 많기 때문에 이 reward 로도 충분히 학습이 가능했다. 하지만 바꾼 simulator 에서는 map 은 전보다 훨씬 커졌고 공의 개수는 줄어들었기 때문에 reward 를 작게 할 경우 24 시간 넘게

학습을 진행해도 만족스러운 결과가 나오지 못했다. Map 은 크고 공의 개수는 적기 때문에 pick-up 하기 까지 많은 step 이 필요한데, 그에 비해 reward 가 작기 때문에 학습이 제대로 진행되지 않은 것이다. 그래서 pick-up 할 때의 reward 를 기본 +100, 추가로 공을 pick-up 할 때마다 +(pick-up 한 공)*50 하여 많은 공을 pick-up 할수록 reward 도 커지도록 만들었다.

## 2.5 CNN

창시구 2 에서는 머신러닝을 통해 위에서 언급되었던 navigation 뿐만 아니라, 이미지 분류에도 사용이 된다. 창시구 1 에서는 파란색 공만을 주웠다면, 창시구 2 에서는 빨간색 공과 파란색 공을 모두 주워 서로 다른 바구니에 떨어뜨리는 것이 최종 목적이다. 그리고 이때, 서로 다른 두 바구니를 구분하기 위해 Convolutional neural network (이하 CNN) 을 사용하여 개와 고양이를 분류하는 네트워크를 학습시켜야 했다.

따라서 CNN 파트는 총 두가지의 큰 이슈로 분류가 된다. 먼저, 개와 고양이를 분류하는 네트워크의 정확도를 높이는 것이 첫번째다. 2.4.1 에서는 CNN 네트워크 학습에 사용된 프레임워크에 대해 설명하였다. 다음으로는 정확한 주차다. 개와 고양이를 분류한 뒤에, 원하는 위치에 차체를 정확히 위치시켜야 공을 안정적으로 바구니 안으로 떨어뜨릴 수 있다. 이때는 readmareker.cpp 에서 특정 마커의 픽셀 좌표를 호출하여 정밀한 주차에 사용하였다. 2.4.2 에서는 개와 고양이 경우에 대해 어떻게 마커 좌표를 사용하였는지에 대해 설명해주었다.

### 2.5.1 transfer_learning_tutorial.py

CNN 파트에서 transfer_learning_tutorial.py 와 CNN_test.py 가 제공된다. 각각 CNN 을 학습시키기 위한 네트워크 설정과 학습된 네트워크를 테스트하기 위한 용도로 사용된다. 기본적으로 배포된 CNN_dogcat0810.pt 를 사용하여 CNN_test.py 를 실행시켜보면 빛이 변화할 때나 갑자기 이미지가 detect 될 때, 이미지가 흔들릴 때 꽤 높은 비율로 분류 실패가 발생했다. (Figure 11)

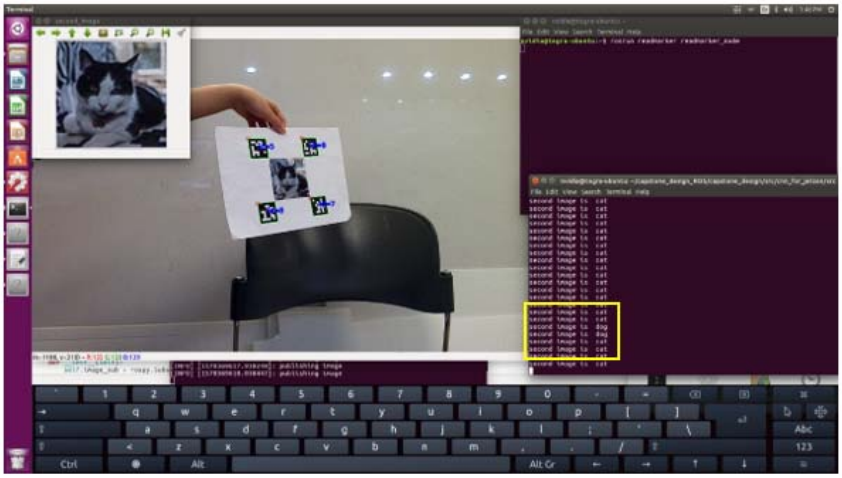


**Figure 11** CNN 분류 오류 예시

Image Classification 의 정확도를 높이기 위해 우리가 transfer_learning_tutorial.py 에서 변화시킬 수 있는 파트들은 크게 Data Augmentation, Number of trained data, Optimizer, Network Model 등이 있다. (이때 Data Augmentation 에서는 창시구 2 에서 제공되는 gpu 인 Jetson 에서 사용가능한 pytorch 0.4.0 버전에서 제공되는 기본 함수들이 최근 버전과는 다르니 확인이 필요하다.) Data Augmentation 에서는 주어진 data 를 더 다양한 가짓수로 늘려 data 양을 확대시키는 파트인데 이때 이미지 크롭이나 반전, 밝기 변화, 기울기 변화, perspective 변화 등을 사용 가능하다. 하지만 이미 우리는 이미지를 readmarker 로 통해 homography 된 이미지를 받아오며, 창시구 데모가 이루어지는 공동강의실 또한 빛의 밝기가 밝아 절대적인 data 양을

늘리는 것 이외의 data augmentation 은 하지 않았다. Data 양은 '자율주행을 위한 프로그래밍' 강의에서 제공되는 25000 여장의 이미지를 사용해 이 중 일부를 train, 나머지를 test 에 사용하였다. Batch size= 16, epoch number=4, learning rate=0.001 로 설정하였으며, optimizer 와 network 모델은 여러 가짓수를 테스트해보며 가장 높은 정확도를 나타내는 조합으로 설정하였다.

```
190  ######################################################################
191  # Finetuning the convnet
192  # ----------------------
193  #
194  # Load a pretrained model and reset final fully connected layer.
195  #
196
197  model_ft = models.resnet50(pretrained=True)
198  num_ftrs = model_ft.fc.in_features
199  model_ft.fc = nn.Linear(num_ftrs, 2)
200
201  model_ft = model_ft.to(device)
202
203  criterion = nn.CrossEntropyLoss()
204
205  # Observe that all parameters are being optimized
206  optimizer_ft = optim.Adagrad(model_ft.parameters(), lr=0.001)
207
208  # Decay LR by a factor of 0.1 every 7 epochs
209  exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
210
```

**Figure 12** transfer_learning_tutorial.py 에서 network 모델과 optimizer 설정 부분

**Figure 12** 와 같이 그때 사용된 최종 optimizer 은 Adagrad, Network model 은 Resnet 50 이었다. 이 조합으로 사용할 경우 학습 시간은 Jetson 에서 약 4 시간, 창시구실의 데스크탑에서는 30 분 정도의 시간이 소요되었다. 최종 사용된 네트워크는 학습 결과 약 0.9896 의 validation accuracy 를 보였다.

초반에는 Nuc 에서 받아온 카메라 데이터로 Jetson 에서 classification 하고 이를 다시 Nuc 으로 보내는 것까지 구축하였다. 그러나, 최종 데모에서는 회로상의 문제로 인해 Jetson 을 달 수가 없었고, 이로 인해 nuc 에서 모든 연산을 해결해야 했다. 이때 Nuc 에서 Image Classification 의 딜레이가 매우 심했다. 우리 로봇의 경우 공을 버리는 순서가 정해져 있어 고양이를 먼저 찾아야 하는데, 이때 발생하는 연산의 딜레이가 나중에 개를 detect 해야 하는 순간에까지 계속 이어져 문제가 발생했다. 때문에 의도적으로 코드에서 publish 를 막는 조건문을 만들었고, 자세한 내용은 2.5 의 marker_Callback 에 설명해두었다.

## 2.5.2 readmarker

Readmarker 의 기본적인 역할은 2.2 의 overall system 에서 얘기한 바와 같이 마커 내의 이미지를 크롭하여 네트워크를 전달하는 것이다. 하지만 우리는 readmarker 픽셀 좌표를 사용해 자동차의 주차에 사용하였다. 주차에 사용되는 코드는 2.6 의 marker_Callback 에 설명해두었으며, 이 장에서는 readmarker 에서 어떤 좌표값을 사용하였는지에 대해 서술하였다.

앞서 언급한바와 같이 차체의 특성으로 인해 공을 분류하는 순서가 정해져있다. 이때 전면 카메라로는 강아지를, 후면 카메라로는 고양이를 detect 해야 한다. 먼저 차체가 회전하다가 후면 카메라를 detect 했을 경우, 처음 접근할 때에 수평을 맞추기 위해서 크롭된 이미지 좌우 길이인 length3 과 length4 의 픽셀상의 y 값을 출력하였으며, 이때 이미지 네 꼭지점의 중점을 사용해 차체의 위치를 맞추었다. 그리고 가까이 가면서 마커 이미지가 일부 안보이기 때문에 이때에는 indice 가 5 인 마커 하나만 사용하여 차체를 얼라인 하였다. 고양이의 경우 마커 4 개가 보일 때는 marker_num=2, 아닐 경우 marker_num=3 을 출력하여 두 경우를 구분했다.

**Figure 13** 벽과 가까워질 때 주로 사용하는 좌표와 가까이 갔을 때 사용하는 마커

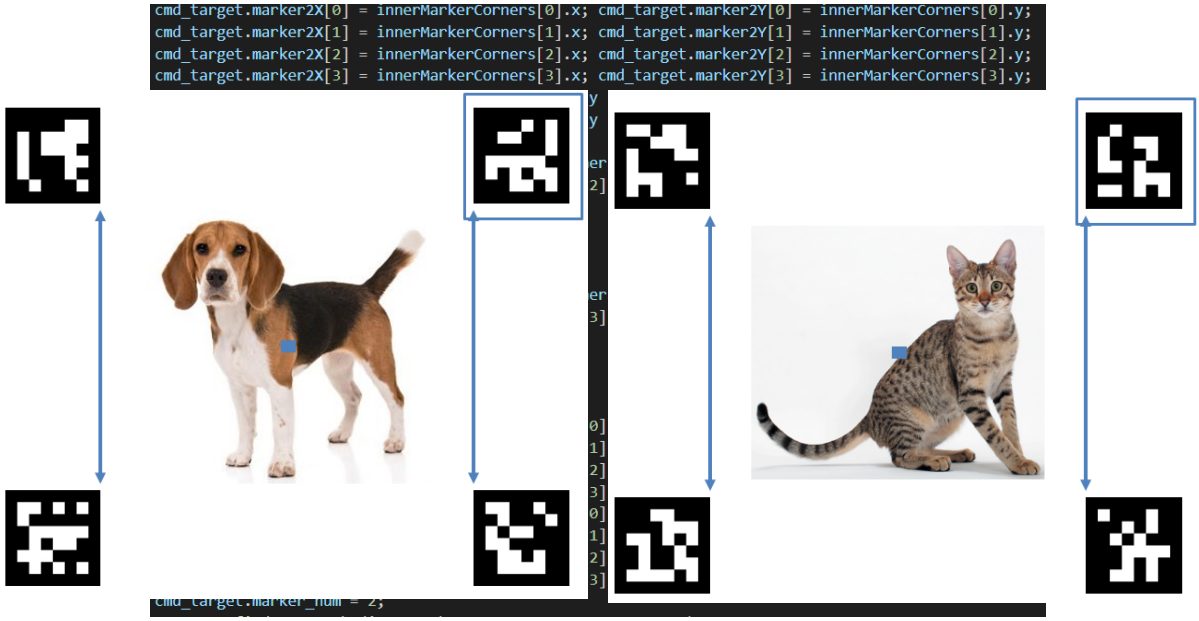



**Figure 14** readmarker 에서 추가되는 관련 코드-마커 4 개가 모두 보일 때 (n_image2)

```
else if(n_image2==0){
  if(markerIds[0]==5 || markerIds[0]==6 || markerIds[0]==7 || markerIds[0]==8){
  innerMarkerCorners[0] = markerCorners[indice[4]][2];
  innerMarkerCorners[1] = markerCorners[indice[5]][3];
  // innerMarkerCorners[2] = markerCorners[indice[6]][0];
  // innerMarkerCorners[3] = markerCorners[indice[7]][1];
  cmd_target.marker_num = 3;
  cmd_target.marker2X[0] = innerMarkerCorners[0].x; cmd_target.marker2Y[0] = innerMarkerCorners[0].y;
  cmd_target.marker2X[1] = innerMarkerCorners[1].x; cmd_target.marker2Y[1] = innerMarkerCorners[1].y;
  cmd_target.length3 = 0;
  cmd_target.length4 = 0;
  cmd_target.markerX[0] = markerCorners[indice[5]][0].x;
  cmd_target.markerX[1] = markerCorners[indice[5]][1].x;
  cmd_target.markerX[2] = markerCorners[indice[5]][2].x;
  cmd_target.markerX[3] = markerCorners[indice[5]][3].x;
  cmd_target.markerY[0] = markerCorners[indice[5]][0].y;
  cmd_target.markerY[1] = markerCorners[indice[5]][1].y;
  cmd_target.markerY[2] = markerCorners[indice[5]][2].y;
  cmd_target.markerY[3] = markerCorners[indice[5]][3].y;
  // cmd_target.marker1X[2] = innerMarkerCorners[2].x; cmd_target.marker1Y[2] = innerMarkerCorners[2].y;
  // cmd_target.marker1X[3] = innerMarkerCorners[3].x; cmd_target.marker1Y[3] = innerMarkerCorners[3].y;
  // cmd_target.length1 = markerCorners[indice[0]][2].y - markerCorners[indice[3]][1].y;
  // cmd_target.length2 = markerCorners[indice[1]][3].y - markerCorners[indice[2]][0].y;
```

**Figure 15** readmarker 에서 추가되는 관련 코드-마커 4 개가 다 보이지는 않을 때 (n_image2)

개를 detect 하는 전면 카메라의 경우도 후면 카메라보다 시야각이 좁기 때문에 가까이 가면 아래의 마커가 보이지 않게 된다. 따라서 개를 detect 하는 경우도 위와 비슷하게 마커 4 개가 모두 detect 되는 경우와 아닌 경우로 나누었다. 개와는 달리 마커 4 개가 보일 때는 marker_num=0, 아닐 경우 marker_num=1 을 출력하여 두 경우를 구분했다. 마커 4 개가 모두

detect 되는 경우는 이미지 4 개의 픽셀상 좌표값을 변수로 저장하였고, 아닌 경우에는 marker indice[1]의 픽셀상 네 좌표값을 모두 저장하였다. 전체적인 코드의 흐름은 figure 15 와 동일하다.

## 2.6 data_integration

data_integration 에서는 SLAM, DQN, CNN 파트를 통합하여 모든 미션을 문제없이 수행하기 위한 작업을 한다. 먼저, 전체적인 코드 구조와 함수들을 설명하고 각 파트 별로 나누어 설명하겠다. 우리는 data_integrate 패키지에서 SLAMmotor 라는 파일 안에 이 코드를 만들었다.

### 2.6.1 Overall Code Structure

```
void msgCallback{
    if(SLAM1_active == 0 && SLAM2_active == 0){}
    else{
      ~~~~~~~~~
      if(condition == satisfied){
        SLAM1_active = 0; SLAM2_active = 0; DQN_active = 1;
      }
    }
void dqnCallback
void markerCallback
void catdogCallback


int main(){
    SLAM1_active = 1; SLAM2_active = 0; DQN_active = 0; CNN_active = 0;

    while(ros::ok){
      ros::spinOnce();
    }
}
```

**Figure 16**

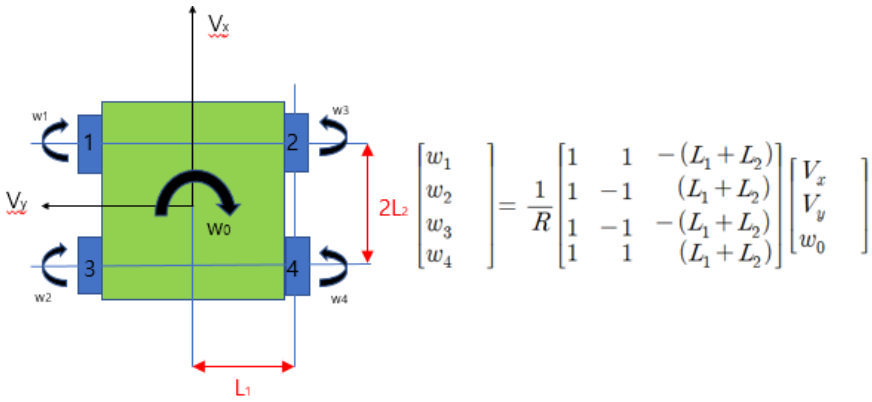
전체적인 구조를 간단하게 설명하면 위의 그림과 같다. msgCallback 은 SLAM 의 output 인 cmd_vel 를 받았을 때, dqnCallback 은 DQN 의 output 인 action 을 받았을 때, markerCallback 은 CNN 에서 QR 코드의 좌표값을 받았을 때, catdogCallback 에서는 CNN 에서 고양이인지 강아지인지의 결과를 받았을 때 실행이 된다.

우리는 모든 노드를 실행하고 있기에 항상 이 Callback 함수들은 동시에 실행되고 있다. 하지만 모터에 가해지는 명령은 하나여야 하기에 우리는 이를 수행하기 위해 SLAM1_active, SLAM2_active, DQN_active, CNN_active 라는 변수를 만들었다. 이 변수가 1 일 경우 하나의 Callback 함수만 실행되는 효과가 있도록 코드를 만들었다. 그 방법은 Callback 함수 들어가자 마자 조건문을 하나 넣어서 그 active 변수가 0 이면 {}로 만들어 아무것도 실행하지 않고 Callback 함수를 빠져나오게 했다. SLAM1 은 미로를 빠져나갈 때, SLAM2 는 다시 원점으로 돌아올 때를 의미한다. 만약 어느 한 단계가 마무리 되어 다른 단계로 넘어가고 싶을 때에는 Callback 함수 내의 else 부분에서 active 변수들을 바꿔준다. 그러면 다른 Callback 함수가 실행될 수 있게 할 수 있다. 이런 방법으로 수많은 메시지를 받는 data_integration 에서 순차적으로 관리를 할 수 있었다.

```
void motor(float linear_x, float linear_y, float angular_z, float picks){
  data[2] = -(1/(WHEEL_RADIUS*gear_ratio)) * (((linear_x) - (linear_y) - (WHEEL_SEPARATION_WIDTH + WHEEL_SEPARATION_LENGTH)*(angular_z))) *30/M_PI; //3
  data[1] = (1/(WHEEL_RADIUS*gear_ratio)) * ((linear_x + linear_y + (WHEEL_SEPARATION_WIDTH + WHEEL_SEPARATION_LENGTH)*angular_z)) *30/M_PI; //2
  data[0] = -(1/(WHEEL_RADIUS*gear_ratio)) * ((linear_x + linear_y - (WHEEL_SEPARATION_WIDTH + WHEEL_SEPARATION_LENGTH)*angular_z)) *30/M_PI; //0
  data[3] = (1/(WHEEL_RADIUS*gear_ratio)) * ((linear_x - linear_y + (WHEEL_SEPARATION_WIDTH + WHEEL_SEPARATION_LENGTH)*angular_z)) *30/M_PI; //1
  data[4] = picks;
}
```

**Figure 17**

위의 함수는 모터 구동을 위한 함수이다. data[]는 myRio 와의 통신에서 바퀴하나의 rpm 을 의미하고 여기서 우리가 rpm 을 정해주면 모터는 이대로 구동된다.




$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \frac{1}{R}\begin{bmatrix} 1 & 1 & -(L_1+L_2) \\ 1 & -1 & (L_1+L_2) \\ 1 & -1 & -(L_1+L_2) \\ 1 & 1 & (L_1+L_2) \end{bmatrix}\begin{bmatrix} V_x \\ V_y \\ w_0 \end{bmatrix}$$

**Figure 18**

위의 식을 통해 바퀴 하나 당 rpm 을 구했다. 즉 x,y 방향의 linear velocity 와 z 방향의 angular velocity 를 입력하면 모터의 rpm 을 정해서 원하는 대로 로봇을 움직이게 할 수 있다. data[4]는 공을 줍기 위한 모터를 의미하고 1 일 때 공을 줍게 된다.

Change Conditions

SLAM1 -> DQN : arrival at goal position area or the nearest ball distance < 0.8m

DQN -> SLAM2 : pick ball number >= 6

SLAM2 -> CNN : arrival at goal position area

**Figure 19**

각 단계로 넘어가는 조건은 다음과 같다. SLAM1 에서 DQN 으로 넘어가는 조건은 지정해준 위치에 로봇이 도착했거나 가장 가까운 공과의 거리가 0.8m 이내일 경우이다. DQN 에서 SLAM2 로 넘어가는 조건은 pickup 한 공의 개수가 6 개 이상일 경우이다. SLAM2 에서 CNN 으로 넘어가는 조건은 지정해준 위치에 로봇이 도착했을 경우이다.

## 2.6.2 msgCallback（SLAM）

```
void msgCallback(const geometry_msgs::Twist::ConstPtr& cmd_vel)
  if(SLAM1_active == 0 && SLAM2_active == 0){
    }

  else{

    data[2] = -1.15*(1/(WHEEL_RADIUS*gear_ratio)) * (((cmd_vel->linear.x) - (cmd_vel->linear.y) - (WHEEL_SEPARATION_WIDTH + WHEEL_SEPARATION_LENGTH)*(cmd_vel->angular.z))) *30/M_PI; //3
    data[1] = 1.15*(1/(WHEEL_RADIUS*gear_ratio)) * ((cmd_vel->linear.x + cmd_vel->linear.y + (WHEEL_SEPARATION_WIDTH + WHEEL_SEPARATION_LENGTH)*cmd_vel->angular.z)) *30/M_PI; //2
    data[0] = -1.15*(1/(WHEEL_RADIUS*gear_ratio)) * ((cmd_vel->linear.x + cmd_vel->linear.y - (WHEEL_SEPARATION_WIDTH + WHEEL_SEPARATION_LENGTH)*cmd_vel->angular.z)) *30/M_PI; //0
    data[3] = 1.15*(1/(WHEEL_RADIUS*gear_ratio)) * ((cmd_vel->linear.x - cmd_vel->linear.y + (WHEEL_SEPARATION_WIDTH + WHEEL_SEPARATION_LENGTH)*cmd_vel->angular.z)) *30/M_PI; //1
    data[4] = 0;
    ROS_INFO("linear_vel.x: %.2f  augular_vel.z: %.2f", cmd_vel->linear.x, cmd_vel->angular.z);
```

**Figure 20**

위의 코드는 SLAM 단계일 때 실행되어야 하는 msgCallback 함수의 코드이다. SLAM1_active, SLAM2_active 가 모두 0 일 때는 {}로 Callback 함수를 빠져나가게 만들었으며 둘 중의 하나가 1 일 때는 cmd_vel 로 들어오는 데이터로 모터를 구동하게끔 만들었다.

```
if((cmd_vel->linear.x)*(cmd_vel->linear.x) + (cmd_vel->linear.y)*(cmd_vel->linear.y) + (cmd_vel->angular.z)*(cmd_vel->angular.z) < 0.000001){
  SLAM1_active = 0;
  DQN_active = 1;
  SLAM2_active = 0;
}


if(r_ball_number1 > 0){
  if(!nearest_r_posi.empty()){
    if(nearest_r_posi[4]<0.8){

      SLAM1_active = 0;
      DQN_active = 1;
      SLAM2_active = 0;
    }
  }
}
```

**Figure 21**

위의 코드는 Callback 함수 내의 else 에서 다음단계로 넘어가는 조건문이다. 첫 번째 if 문에서는 로봇의 속력이 매우 작아지면 지정한 위치에 도달한 것이기에 이 경우 DQN 으로 넘어가게끔 만들었다. 두 번째 if 문은 가장 가까운 공의 거리가 로봇으로부터 0.8m 이내 일 경우 다음 단계로 넘어가게 만든 것이다. r_ball_number1 은 위의 카메라에서 본 빨간 공의 개수를 의미하며, r_ball_number2 는 픽업하기 위한 밑의 카메라에서 본 빨간 공의 개수를 의미한다. 위와 같은 if 문을 r_ball_number1, r_ball_number2, b_ball_number1, b_ball_number2 4 가지 경우로 모두 적어서 어떤 공이든 어떤 범위 이내에 들어오면 DQN 으로 넘어가게끔 만들었다.

DQN 이 끝났을 시 다시 SLAM2_active 가 1 이 되어 이 Callback 함수가 다시 실행되고, 원점에 도착했을 때에는 find_frontier 에서 도착했다는 데이터를 data_integration 에 보내고 SLAM2_active=0, CNN_active=1 로 만든다. 이 부분은 메인함수 내부의 while 문에 구현되어 있다.

### 2.6.3 dqnCallback (DQN)

이 Callback 함수는 크게 3 부분으로 나눌 수 있다. 먼저 DQN_active=0 일 때 빠져나가게 하는 경우, 공이 카메라에 한 개도 없을 경우, 그 외의 경우로 나눌 수 있다. 이는 if, else if, else 로 구성하였고 공이 하나도 없을 경우에는 시계반대방향으로 회전하게끔 만들었다. else 에서는 DQN 에서 보내온 action 대로 행동하되 픽업하는 위치 기준으로 공이 10cm 이내로 들어오면 픽업 알고리즘으로 로봇이 행동하게끔 만들었다.

```
if(!nearest_r_posi2.empty() && ball_check == 0){
  if(nearest_r_posi2[4] < 0.6){
     data[17] = 1;
    if(abs(nearest_r_posi2[3]) > theta_c_i1 && nearest_r_posi2[0] > 0){ //두번째 웹캠으로 얼라인. 2
      motor(0, 0, M_PI/60, 0);
    }
    else if(abs(nearest_r_posi2[3]) > theta_c_i1i && nearest_r_posi2[0] < 0){ //두번째 웹캠으로 얼
      motor(0, 0, -M_PI/60, 0);
    }
    else if(nearest_r_posi2[4]>p_c){ //2webcam, 얼라인 되었을 때 앞으로 천천히 전_
      motor(0.1, 0, 0, 0);
      data[17] = 0;
    }
    else if(nearest_r_posi2[4]<0.35){
      motor(0.1, 0, 0, 0);
    }
    else{
      motor(0.01, 0, 0, 1);
      data[17] = 1;
      int i =1;
      while(i<100){
        motor(0.01, 0, 0, 1);
        write(c_socket, data, sizeof(data));
        i++;
        ros::Duration(0.02).sleep();
      }

      nearest_r_posi2.clear();
      ball_check = 1;
      red_pickball_num++;
      cout<<"redpickball = "<<red_pickball_num<<", "<<"bluepickball = "<<blue_pickball_num<<endl;
      if(red_pickball_num + blue_pickball_num > 5){
        DQN_active = 0;
        SLAM2_active = 1;
        nearest_b_posi.clear();
        nearest_r_posi.clear();
        nearest_b_posi2.clear();
        nearest_r_posi2.clear();
        //dataInit();
        int yo =1;
        while(yo<100){
          yo++;
          motor(0, 0, 0, 0);
          data[17]=1;
          write(c_socket, data, sizeof(data));
          ros::Duration(0.02).sleep();
        }
      }
```

**Figure 22**

위의 코드는 픽업 알고리즘 코드이다. 가장 가까운 공이 범위 이내로 들어오면 if, else if, else if 등을 통해 align 을 하고 공과의 거리가 p_c 가 될 때까지 앞으로 전진한다. 0.35 보다 작을 때 앞으로 가는 명령은 노이즈 때문에 생기는 문제를 막기 위함이다. 만약 공과의 거리가 p_c 보다 작게 되면 else 로 넘어가서 while 문이 돌아가는 동안 픽업 blade 를 돌리게 된다. 또한 이 때 픽업한 공의 개수를 하나 추가하고 만약 6 개 이상일 시 DQN_active 를 0 으로 바꾸고 SLAM2_active 를 1 로 만들어준다. 밑에 아무런 모터 구동이 없는 while 문을 추가한 이유는 시간 간격을 두지 않으면 공을 두 개 먹었다고 인식을 하는 문제가 있었기 때문에 이를 막기 위함이다. 이와 같은 방식으로 빨간 공이나 파란 공이 범위 이내에 들어왔을 때 공을 픽업하는 알고리즘을 코드로 구현했다.

```
switch (msg->data){

  case 0: // Forward

      motor(0.13, 0, 0, 0);
      break;

  case 1: // ccw

      motor(0.13, 0, M_PI/40, 0);
      break;
  case 2: // cw

      motor(0.13, 0, -M_PI/40, 0);
      break;



    default:
        ROS_INFO("NOT VALID action/int8, out of range (0~7)");
    break;
```

**Figure 23**

위의 코드는 공이 범위 이내에 들어오지 않았을 경우 action 대로 행동하는 부분의 코드이다. DQN 에서 보내준 대로 행동하며 전진, 전진 & 좌회전, 전진 & 우회전 세가지의 경우가 있다.

## 2.6.4 markerCallback (CNN)

markerCallback 의 경우 readmarker 노드로부터 marker 의 좌표 데이터를 받는 부분과 이 데이터로 로봇을 제어하는 부분으로 나눠져 있다.

```
if(cmd_target->marker_num == 0){
    length = cmd_target->marker1X[1]- cmd_target->marker1X[0];
    length1 = cmd_target->markerX[1]-cmd_target->markerX[0];
    center = (cmd_target->marker1X[0]+cmd_target->marker1X[1])/2;
    height1 = cmd_target->length1;
    height2 = cmd_target->length2;
    height11 = -cmd_target->markerY[0]+cmd_target->markerY[3];
    height22 = -cmd_target->markerY[1]+cmd_target->markerY[2];
    center1 = (cmd_target->markerX[0]+cmd_target->markerX[1]+cmd_target->markerX[2]+cmd_target->markerX[3])/4;
}
else if(cmd_target->marker_num == 1){
    center = (cmd_target->marker2X[0]+cmd_target->marker2X[1])/2;
    length = cmd_target->marker2X[1]- cmd_target->marker2X[0];
    length1 = cmd_target->markerX[1]-cmd_target->markerX[0];
    height1 = cmd_target->length3;
    height2 = cmd_target->length4;
    height11 = -cmd_target->markerY[0]+cmd_target->markerY[3];
    height22 = -cmd_target->markerY[1]+cmd_target->markerY[2];
    center1 = (cmd_target->markerX[0]+cmd_target->markerX[1]+cmd_target->markerX[2]+cmd_target->markerX[3])/4;
}
```

**Figure 24** readmarker 로부터 좌표 데이터를 받는 code

우선 위의 코드가 좌표 데이터를 받는 부분이다. 우선 readmarker 노드에서 몇 번째 이미지를 보는지에 따라 marker_num 이라는 변수에 0 또는 1 을 저장하여 publishing 하고 이에 따라 값을 받도록 만들었다. 좌표 데이터는 이미지가 개일 경우 오른쪽 위의 marker 만을 사용하여 움직이고 고양이일 경우 위 두개의 marker 를 이용하여 움직인다. 이러한 이유는 앞의 카메라로 개를 찾아야 하는데, 이 카메라의 위치 때문에 두개의 marker 를 볼 수 있는 범위가 매우 한정적이기 때문이다. 'length' 변수는 두 marker 사이의 거리, 'length1' 은 오른쪽 위 marker 의 길이로 로봇과 이미지 사이의 거리를 비교할 때 사용한다. 'center' 와 'center1' 은 각각 두 marker 의 중심 좌표, 오른쪽 위 marker 의 중심 좌표로 로봇과 이미지의 중심을 맞추기 위해 사용한다. Height 의 경우 로봇과 이미지의 수평을 맞추기 위해 사용하는

데이터로 ‘height1’, ‘height2’는 위 marker와 아래 marker사이의 거리이고 ‘height11’과 ‘height22’는 오른쪽 위 marker의 세로 길이이다.

```
switch(phase){
    case 0:
        if(catdog != "cat"){
        motor(0, 0, M_PI/35, 220);
        std::cout<<"turn_left_0"<<std::endl;
        }
        else if(catdog == "cat"){
            phase = 1;
        }
        break;
    case 1:
    read_stop = 1;
        if(length<70){
            if((height1-height2)>1){
                // turn_left()
                motor(-0.1, 0, M_PI/50, 220);
                std::cout<<"turn_left"<<std::endl;
            }
            else if((height1-height2)<-1){
                // turn_right()
                motor(-0.1, 0, -M_PI/50, 220);
                std::cout<<"turn_right"<<std::endl;
            }
            else if(center>330){
                motor(-0.1, 0.05, 0, 220);
                std::cout<<"move_left"<<std::endl;
            }
            else if(center<310){
                motor(-0.1, -0.05, 0, 220);
                std::cout<<"move_right"<<std::endl;
            }
            else{
                motor(-0.1, 0, 0, 220);
                std::cout<<"go_straight"<<std::endl;
            }
            }
```

```
else if(length>70 && length<210){
    if((height1-height2)>1){
        // turn_left()
        motor(-0.05, 0, M_PI/50, 220);
        std::cout<<"turn_left"<<std::endl;
    }
    else if((height1-height2)<-1){
        // turn_right()
        motor(-0.05, 0, -M_PI/50, 220);
        std::cout<<"turn_right"<<std::endl;
    }
    else if(center>330){
        motor(-0.05, 0.05, 0, 220);
        std::cout<<"slow_move_left"<<std::endl;
    }
    else if(center<310){
        motor(-0.05, -0.05, 0, 220);
        std::cout<<"slow_move_right"<<std::endl;
    }
    else{
        motor(-0.1, 0, 0, 220);
        std::cout<<"go_straight"<<std::endl;
    }
}
else if(length>210){
    phase = 2;
    motor(0,0,0,220);
    std::cout<<"stop"<<std::endl;
}

break;
```

**Figure 25** 고양이 이미지 일 때 작동하는 code

Figure 24은 이미지가 고양이일 때 로봇을 구동하는 코드이다. 우선 CNN이 ‘cat’을 보낼 때까지 반시계 방향으로 회전하고 ‘cat’을 받았을 경우 멈추고 case1으로 넘어간다. Case1에서는 height1과 height2의 차이를 비교하여 수평을 맞추고, center의 값을 이용하여 중심을 맞추었다. 중심과 수평이 모두 맞을 경우 일정 거리까지 앞으로 간다. Length의 비교를 70과 210으로 나눈 이유는 length가 70보다 큰 경우 일정 거리 이상 가까운 상황이기 때문에 속도를 낮춰 align을 더 정확하게 하기 위해서다. Length가 210보다 큰 경우 바구니에 도착한 상황이기 때문에 공을 release하고 case2로 넘어간다. Case1의 시작에 read_stop이라는 변수가 지정되어 있는데, 이 변수는 readmarker가 CNN으로 이미지를 publish하는 것을 막기 위해 만들었다. 우리 조는 jetson을 사용하지 않기 때문에 CNN의 연산 속도가 매우 느리다. 만약 publish를 막지 않을 경우 고양이 이미지가 있는 바구니에 공을 release한 뒤에도 계속해서 고양이 이미지를 연산하고 있기 때문에 이를 방지하기 위해 case1이 실행되는 동안에는 readmarker에서 publish하는 것을 막았다.

```
case 2:
read_stop=0;
    webcam_change = 1;
    if(catdog != "dog"){
        std::cout<<"go_straight"<<std::endl;
        motor(0.1, 0, 0, 220);
    }
    else if(catdog == "dog"){
        phase = 3;
    }
    break;
```

**Figure 26** 개 이미지일 때 작동하는 code

Case2와 case3는 이미지가 개일 때의 상황으로 위 case0, 1과 사용하는 변수와 값을 제외하고는 거의 동일하다. 추가되는 부분은 case2에서 webcam_change라는 변수를 1로 만든 것이다. 우리 조에서는 CNN에서 뒤의 카메라와 앞의 카메라를 모두 사용하기 때문에 ball_detect 노드에서 readmarker 노드로 이미지를 publish할 때 어떤 이미지를 보낼지 선택해야 한다. 이를 위해 case2가 실행될 경우 webcam_change를 1로 바꾸어 ball_detect 노드에서 보내는 이미지를 뒤의 webcam에서 앞 webcam으로 변화시켜준다.

2.7 Philosophy and Ideas

우리의 철학은 simple 이다. 필요 없는 부분은 과감히 빼서 코드를 간단하게 만들려고 노력했고 그 결과 data_integration 에서 모든 데이터를 처리하게끔 만들었다. 다른 노드들은 필요 없다고 생각했기 때문이다. 데이터 흐름을 복잡하지 않게 하기 위해 노력했고, 위의 Overall system 에서 확인할 수 있듯이 복잡하지 않은 rqt_graph 를 얻을 수 있었다. 또한 실제로 미션을 수행하는 것은 로봇이기에 실험을 중요하게 생각했다. 수많은 실험으로 SLAM 파트에서 적절한 파라미터를 찾았고 미션을 성공하는 확률이 높아지도록 계속해서 튜닝을 하였다.

SLAM 에서는 미로를 빠져나가기 위한 목표지점을 우리는 직접 실험을 통한 좌표로 설정해서, 로봇이 오른쪽 또는 왼쪽으로 빠져나갈 수 있는 불확실성을 없애고 항상 오른쪽으로 빠져나가게 만들었다. 이는 미로 통과 확률을 확연히 높여주었다. 돌아오는 것 또한 좌표를 설정해서 돌아오게끔 만들었다. SLAM 에서의 특징을 뽑자면 y 방향 linear velocity 를 0 으로 주어서 횡 방향 움직임을 없애고 로봇이 자동차처럼 곡선으로 움직이게끔 한 것이다. 이는 바닥이 불안정할 경우 횡 방향 움직임에서 오류가 많았기에 취한 조치이고 이로 인해 더욱 안정적으로 미션을 수행할 수 있었다.

DQN 에서 큰 특징은 map 을 가능한 실제 환경과 비슷하게 만든 것과 action 의 수를 최대한 줄인 것이다. Map 을 더 랜덤하게 만들 경우 예상치 못한 상황에 대한 대처는 잘되겠지만, 실제 데모에서는 벽의 위치나 크기, 공의 개수 등이 고정되어 있기 때문에 이러한 map 에서 학습하는 것이 더 효율이 좋을 것이라 판단하여 map 을 만들었다. Action 의 경우 여러가지 조합을 만들어 학습해 보았지만, action 의 수가 많을 경우 이상한 행동을 반복하는 경우가 발생하기 때문에 수를 최대한 줄였다. 또한 SLAM 과 마찬가지로 횡 방향 움직임이 불안정하기 때문에 아예 배제하고 action 을 구성했다. Pick-up 하는 것 또한 하나의 action 으로 만들어 학습해 보았으나, 이 경우 학습이 거의 진행되지 않는 문제가 발생했다. 만약 우리 조처럼 pick-up 하기 위한 추가적인 행동이 필요할 경우, simulator.py 의 action 을 추가하기 보다는 새로운 변수를 만들고 simulator.py 가 아닌 dqn_learn.py 를 수정해야 한다.

CNN 파트는 이미지 분류와 마커를 이용한 차체의 주체 두가지 task 를 가지고 있었다. 이미지 분류의 경우 학기 시작과 함께 가장 먼저 시작한 부분이기도 하고, 기본적으로 주어진 코드 내에서 크게 변화 시킬 부분이 없기 때문에 가장 짧은 시간내에 끝낼 수 있는 파트였다. 기존에 주어진 모델의 성능을 크게 향상시켰던 부분은 data 양이라고 생각된다. 최종적으로는 22500 장의 이미지를 사용하여 train 시켰으며, optimizer 와 network 모델 또한 여러 테스트를 거쳐 가장 적합한 조합을 사용하였다. 그러나 특정 종이나 동물의 옆모습과 같은 극단적인 상황의 이미지의 경우 아직 정확도가 떨어지는 부분이 있었기 때문에, 최종 데모에서 실수를 줄이고 싶다면 더 다양한 이미지 데이터 셋을 찾는 것이 중요할 것으로 생각된다. 차체의 주차는 하드웨어의 카메라의 시야각도 문제를 극복하기 위해 마커 하나만을 이용하여 차체를 정밀하게 주차한 것이 안정적으로 미션을 수행할 수 있었던 이유였다. 다만, CNN 과 별개로 카메라 성능 또한 고려해야 함을 강조하고 싶다. 우리 조의 경우 항상 공동강의실의 모든 불을 다 켜야 카메라에서 marker 를 움직이는 상황에서 detect 할 수 있었다. 어두운 공간에서 카메라가 더 많은 빛을 확보해야 하는데 이 과정에서 카메라의 blur 가 심해져 차체가 움직이면서 마커를 detect 하기 어려워지기 때문이다. 따라서 CNN 을 하기 전 카메라의 성능과 마커 detect 에 적합한 빛 조합을 을 미리 테스트 해보는 것을 추천한다.

## 2.8 Desired Code Schedule

| | 9월 | | | | 10월 | | | | 11월 | | | | 12월 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| week | 1st | 2nd | 3rd | 4th | 1st | 2nd | 3rd | 4th | 1st | 2nd | 3rd | 4th | 1st | 2nd | 3rd | 4th |
| 일정 | | | | | | 중간 발표 | 시험기간 | | | | | | 최종 발표 | | | |
| CNN | | | | | | | | | | | | | | | | |
| SLAM | | | | | | | | | | | | | | | | |
| DQN | | | | | | | | | | | | | | | | |
| Integration | | | | | | | | | | | | | | | | |

**Figure 27** Desired Code Schedule

창시구 2 를 수강하며 느낀 점을 바탕으로 구상한 스케줄이다. 스케줄을 보면 알다시피, 개강 초 기본적인 내용을 배우는 시간과 시험기간을 빼면 프로그래밍에 쏟을 수 있는 시간이 크게 많지 않다는 것을 알 수 있다. 때문에 학기 초에 미리 코드 관련된 스케줄을 타이트하게 짜둘 것을 추천한다. 그리고 코드를 담당한 팀원들간의 역할 배분 또한 미리 고려되어야 효율적으로 코드 작성과 테스트를 진행할 수 있을 것이다.

CNN 의 경우 미리 제공되는 코드에 크게 수정할 것이 없으며 관련 노드들 또한 제공이 되기 때문에 가장 먼저 끝낼 수 있는 파트라 생각된다. 이번 창시구 2 에서는 중간발표 이틀 전에 CNN demo 가 있었지만, 그 이전에 끝낼 수 있는 task 였다. CNN 은 트레이닝에 dqn 보다 비교적 적은 시간이 걸리고, 데이터 양이나 data augmentation 부분을 좀 더 신경쓴다면 충분히 높은 정확도의 모델을 뽑을 수 있을 것이다. Jetson 과 Nuc 부분의 통신 문제도 크게 어렵지 않다. 9 월 초~중순까지 배운 머신러닝 내용을 바탕으로 약 1 주정도면 CNN 파트의 image classification 부분이 완료가 될 것이다. 다만 이 시기에 CNN 파트 담당자들은 CNN 이 끝난 뒤, 차체의 align 방식을 미리 구상하는 것을 추천한다. 후반부 DQN 의 시간이 생각보다 오래걸릴뿐더러, data integration 에도 꽤 많은 시간이 소요된다.

수업에서는 CNN 내용이 끝나는 대로 바로 SLAM 에 관련된 강의가 시작된다. SLAM 파트 또한 중간 발표때까지 최대한 많이 끝내 두는 것을 추천한다. 창시구 2 에서도 하드웨어의 완성은 11 월 초~중반에 끝날 것으로 예상된다. 따라서 우선 창시구 1 에서 만들어 두었던 하드웨어를 가지고 빠르게 SLAM 을 구현해보고, 나머지는 하드웨어가 완성된 뒤에 약간씩 튜닝을 통해 코드를 수정해나가는 것이 효율적이라 생각된다.

DQN 은 중간고사가 끝난 뒤 바로 코드를 공부하고 학습을 시작하는 것을 추천한다. 네트워크 하나의 트레이닝에 보통 10 시간 이상이 소요된다. 그러나 조마다 배분된 데스크탑은 1 개뿐이기 때문에 reward 조합을 여러가지 테스트해볼 수 있는 시간이 크게 없다. 제공되는 dqn 코드는 dqn 배경 지식이 크게 필요치 않기 때문에 바로 코드를 공부할 수 가 있을 것이다. 시뮬레이터를 구성, 차체의 크기나 공을 먹는 부분들은 주어진 map 이나 완성된 하드웨어를 갖고 바로 변경해둘 수 있다. 그리고 dqn 을 공부하며 바로바로 reward 조합이나 네트워크 조건들을 변경해가며 러닝을 하루라도 빨리 시작하는 것이 좋다.

그리고 dqn 모델이 어느정도 나올 시점부터는 전체 코드를 통합하는데 주력해야 한다. 생각지도 못한 설계 문제가 발견될 수도 있고, 카메라나 컴퓨터에 문제가 발생할 수도 있기 때문이다. 그리고 최종 데모에서 팀에게 필요한 빛의 조건, 이미지 높이나 위치 등을 찾아 두어야하는 시기기도 하다. 따라서 최대한 빨리 통합을 하고 남은 기간동안 문제점을 찾고 해결하는 과정을 반복하며 전체적인 완성도를 높여나가는 것을 추천한다.

# 3. Hardware

[week 1]

창의적 시스템 구현의 첫 시간에, 저희는 전 팀이 만든 차체를 분석했다.
1 학기의 차량은 공을 발견하면 앞쪽에 위치한 빗자루로 공을 쓸어담아서 옆의 경사면에 있는 공 보관소로 넘긴 후 바구니에 릴리즈하는 구조였다. 전반적으로 굉장히 잘 만들어져 있고 변형이 쉬운 구조물이었기 때문에, 저희는 이 구조를 그대로 가져가면서 추가된 미션을 수행하기 위해 약간의 변형만 해서 차체를 만들기로 했다.

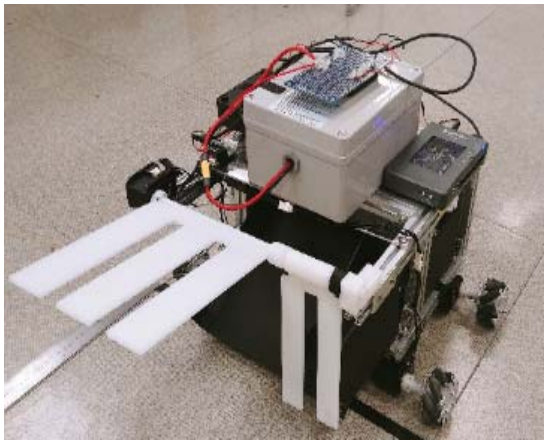


Fig 28. 기존의 차체

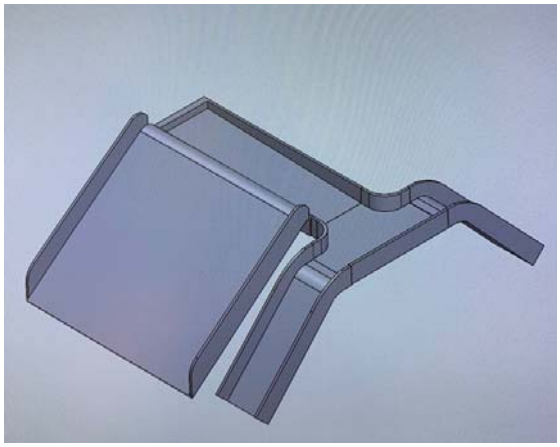


Fig 28-2. 새로운 미션에 맞는 새로운 모듈

기존의 차체는 픽업 모듈에서 공을 쓸어담기만 하면, 나머지는 경사면에 의해서 떨어지기만 하는 구조였기 때문에, 경사면을 하나 더 추가해서 놓는 구조를 생각했다.
또한, 저희는 차체를 만들면서 공을 분류하는 구조 이외에 새로운 모터를 설치하지 않기로 했다. 모터를 새로 설치하면 이에 맞는 회로 설계와 코드 수정이 필요하고, 새로운 모터를 설치하면 열 발생이 늘어나므로 가능하다면 새 모터를 설치하지 않기로 했다.
그리고 차체에 새로운 구조를 추가하면서 최대한 길이와 폭을 작게 만들기로 했다. 미로를 탈출하기 위해서는 차량이 좁은 틈 사이로 빠져나가야 하는데, 차량의 폭이 너무 넓거나 길이가 너무 길면 차량이 빠져나가지 못하거나 빠져나가는 데 걸리는 시간이 길어지기 때문에 차량을 작게 만들면 미션이 쉬워질 것이기 때문이다.

[week 2]

저희는 앞의 팀에서 만들어 놓은 구조를 최대한 깨지 않고 뒤쪽에 새로운 경사면을 만들어 파란 공 또한 분류가 가능하게 발전시키고자 하였다. 그리고 첫 주에 정한 가정을 구체화했다.
이에 따라서 뒤쪽 경사면을 제작할 때 앞쪽 경사면과 똑같이 세 개의 공을 담을 수 있는 공간이 있어야 하며, 공을 1 초 이내로 빠르게 내보낼 수 있고, 또한 차량의 너비와 길이를 최소화하자는 가정을 바탕으로 제작했다.
이에 따라서 Fig 2 와 같이 새로운 구조를 추가하였다. 1 주차에 생각했던 방식은 차량의 길이를 너무 많이 증가시켰기 때문에, 과감히 대칭성을 버리고 뒤쪽 경사면을 90 도 돌려서 뒤쪽 공 보관소를 만들었다. 하지만 공을 릴리즈하는 부분이 옆쪽으로 나 있으면 공을 릴리즈하기 위해서 추가적으로 차체를 90 도 돌리고 차량을 옆으로 이동시켜서 공을 릴리즈해야 했다. 이 방법은 너무 위험할뿐더러, 미션 시간을 너무 많이 잡아먹는다고 판단했기 때문에, 공을 릴리즈하는 부분은 여전히 뒤쪽 방향으로 하는 방법을 찾았다.
따라서 저희는 뒤쪽 공을 담는 경사면의 한 벽을 테이프로만 부착해서 Fig 2 와 같이 바깥쪽으로 돌아갈 수 있게 설계하였다.
또한 저희는 앞에서 모터 없이 공을 릴리즈하기로 했기 때문에, 동력 없이 공을 릴리즈하기 위해서 회전하는 문을 오뚜기 형식으로 제작했다. 다시 말해 회전하는 문의 무게중심을 매우 아래쪽에 두어서 공 세 개와 차체의 관성력을 버틸 수 있도록 설계하였다.


Fig 29. 뒤쪽 경사면의 설계

[week 3~4]

뒤쪽 경사면은 앞쪽 경사면과 다르게 너비 방향으로 길게 되어있다. 그러나 차체가 바구니에 닿으면 공은 길이 방향으로도 떨어져야 했으므로, 뒤쪽 경사면을 앞쪽 경사면과 다르게 높이 방향을 제외한 두 축 방향으로 모두 내었다.

또 바구니에 도달할 시 공을 막고 있던 문이 돌아가서 공을 릴리즈하도록 설계하였다. 그러나 오뚜기 방식의 회전문은 관성력에 매우 취약하고, 살짝 열린 틈으로 공이 끼어 버리면 공이 속수무책으로 빠져나와 버린다는 사실을 알았다. 따라서 저희는 오뚜기 방식의 회전문을 포기하고 트리거를 활용한 새로운 무동력 매커니즘을 만들었다.

평소에는 뒤쪽 공 보관소에 있다가 바구니가 회전문을 밀어낼 때 공이 쏟아져 나와야 했으므로.회전문을 밀어내는 힘을 바구니의 무게와 마찰계수를 고려하여 회전문을 자석으로 잡아두었다. 자석이 너무 약하면 진행 도중에 공이 쏟아져나오고, 자석이 너무 강하면 바구니가 회전문을 밀어내지 못했기 때문에, 네오디뮴 자석이 아닌 고무 자석을 활용하여 트리거를 설계하였다.

이제 차량이 고양이 사진을 센싱하면, 차량이 고양이 방향으로 계속 이동해 바구니 방향으로 곧장 이동하게 됩니다. 어떤 모터가 회전문을 열라는 신호를 보내는 대신에, 회전문이 바구니에 밀리면서 문이 회전해 공을 릴리즈하게 된다.

Fig 30. 자석을 활용한 뒤쪽 공 릴리즈

Fig 31. 추가한 뒤쪽 공 보관소

Fig 32. 내부 모듈의 구조

또한 공을 색깔에 따라서 분류하기 위해서 앞 뒤 경사면 사이에 새로운 경사면을 하나 추가하고 이에 매우 가벼운 서보 모터를 분기점에 두어 공의 색깔에 따라 경사면의 방향을 바꾸는 구조를 설계하였다.

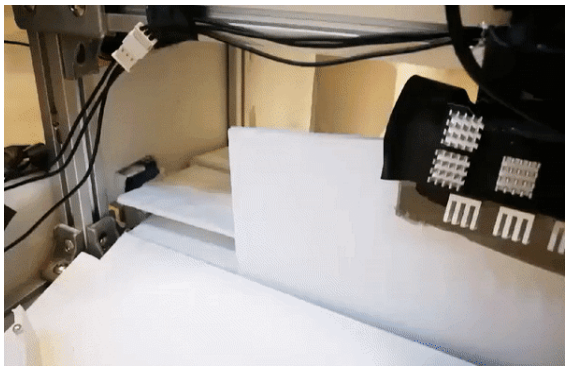


Fig 33. 서보 모터를 장착한 경사면

[week 5~6]

저희 조의 차량은 기존의 차량에 서스펜션이 없어 차체의 안정성이 떨어지므로, 새로운 서스펜션을 디자인해 차체의 안정성을 높이고자 하였다. 저희 차량의 무게는 8kg 정도였고, 새로 추가될 모듈의 무게를 계산하면 9kg 까지 늘어날 것이므로, 이 무게에 맞는 서스펜션의 구조를 찾아보았다.

저희는 RC 카의 서스펜션 구조를 많이 참고했다. 그러나 대부분은 왼쪽 바퀴와 오른쪽 바퀴를 하나의 링크로 연결해놓은 듯한 구조를 띠고 있고, 차량의 중간부에 쇽 업소버를 지지할 지지대가 있었다. 그러나 저희 모듈은 그 부분에 공을 픽업하고 릴리즈하는 모듈이 들어가 있어 그 구조를 채택하기가 힘들었다.

그래서 저희는 바퀴 하나마다 독립적으로 돌아가는 서스펜션 구조를 찾아보았다. 대표적으로 사용되는 서스펜션 구조인 double A 구조와 Mcpherson single strut 구조 사이에서, 저희는 디자인 변경이 쉬운 Mcpherson single sturt 구조를 사용했다. 용수철은 9kg 을 4 개의 쇽업쇼바가 견딜 수 있게끔 선정했다. 여기서 저희는 9kg 의 차체를 견딜 수 있는 스프링을 구할 수 없다는 판단을 하고, 용수철 상수가 높은 금형 스프링을 구매하고 이를 지지할 수 있는 피스톤을 3D 프린팅하서 쇽 업소버를 제작하였다.

이 과정에서 많은 시행착오가 있었다.

우선 3D 프린팅된 피스톤은 어쩔 수 없이 마찰이 발생했다. 마찰을 줄이기 위해서 피스톤 사이의 공차를 약간 늘리고 사포로 표면을 매끈하게 만들어서 문제를 해결해야 했다.

또한 금형 스프링의 길이가 맞지 않아서 몇 번 새로 금형 스프링을 사야 했다. 이 과정에서 많은 시간을 소모했다.

또한 기어박스에 쇽 업소버를 장착하기 위해서 기어박스의 외부에 힌지를 추가하였다. 또한, 기어비를 기존의 2.7:1 에서 4:1 까지 올리기 위하여 기어를 새로 제작하였으며, 여기에 맞추어 기어박스의 크기를 늘렸다.. 이에 맞추어 20mm, 80mm 지름의 기어를 주문하였다.

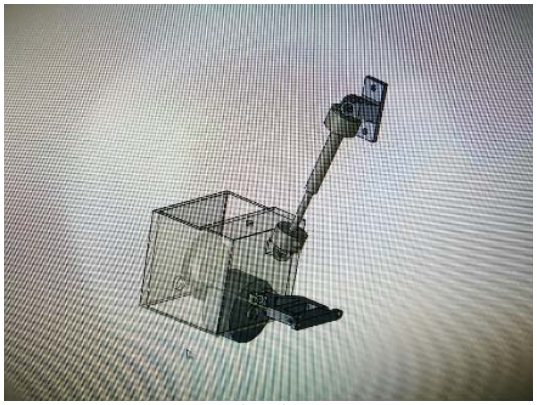


Fig 34. 초반 서스펜션 구조

또한 차량을 조립하면서 차체에는 작은 문제가 조금씩 발생했다.
먼저, 차체가 알 수 없는 이유로 조금씩 내려앉기 시작했다. 저희는 이것이 차체 피로도 때문이라고 예상했고, 폼보드로 같은 매커니즘을 보강 구조를 추가하여 새로 제작하여 해결했다.
또한 공을 분류하는 경사면이 흔들리는 문제가 발견되었다.
저희는 벽면에 설치된 서보 모터에 폼보드 판을 수직으로 끼워서 분류 경사도를 제작했는데(Fig 7). 이 과정에서 서보 모터가 폼보드 판을 제대로 지지하지 못해서 경사면이 축 방향 뿐만이 아니라 다른 방향으로도 회전하는 문제가 있었다.
저희는 이를 서보 모터가 벽면에 정확히 고정되어있지 않았다고 판단해서 벽면을 알루미늄 프로파일에 더 단단하게 지지했다. 하지만 실험을 계속하면서 서보 모터가 폼보드를 정확히 고정해주고 있지 않다는 결론을 내렸고, cantilever beam 의 형태로 고정되어 있던 폼보드 판의 반대쪽또한 철 핀으로 고정해서 양단 고정형 beam 의 형태로 변형시켰다.

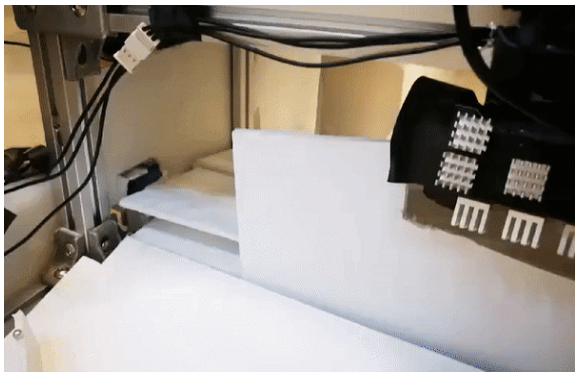


Fig 35. 서보 모터를 활용한 공 분류 경사면

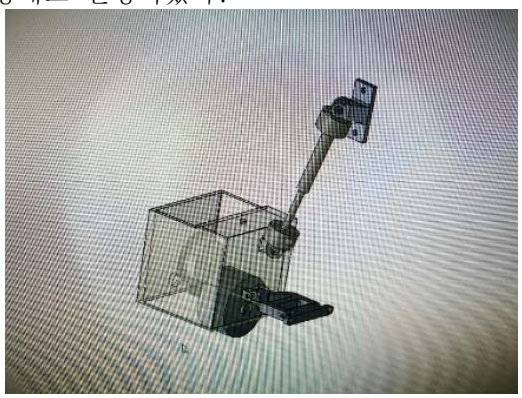


Fig 36. 초반 서스펜션 시스템

또한 뒤쪽 공 보관소의 트리거 진행 도중 공의 무게를 견디지 못하고 무너지는 문제가 발생함에 따라서 트리거의 자석의 길이를 늘려서 해결하였다.

[week 7~8]
기어를 장착하는 과정에서 문제가 생겼다. 도착한 기어는 너무 무거웠고, 예상 외로 기어의 보스가 너무 커서 기어박스에 모두 들어가지 않았다.
기어박스를 새로 설계해 3D 프린팅을 하고, 쇽 업소버를 알루미늄 프로파일에 고정시켜줄 힌지 구조 또한 설계해서 만들었다. 하지만 결정적으로 저희가 마주한 가장 큰 문제점은 서스펜션이었다. 서스펜션을 전부 설치한 후에야 저희의 서스펜션 시스템이 기어박스에 걸리는 모멘트를 잡아주지 못했다는 사실을 알아챘고, 이를 잡으려고 짧은 시간 안에 해결할 수 있는 해결책을 내 보았지만 해결하지 못하고, 이 때문에 저희는 서스펜션 구조를 처음으로 롤백해야 했다.

[week 9~10]
중간평가 이후, 성공하지 못했던 서스펜션 구조를 다시 설계하고 기어박스의 크기를 줄이는 작업을 시작하였다.
기어박스의 길이를 최대한으로 줄이고, 기어비를 올리는 방향으로 설계의 틀을 잡았다. 그래서 과감히 기어의 모듈을 줄이고 기어의 개수를 늘렸다. 새로운 기어박스는 차량의 너비를 40mm 까지 줄여 주었다. 또한 기어박스의 조립을 용이하게 하기 위해서 기어박스를 뚜껑과 몸체 부분으로 분리하였다.

Fig 37. 새로운 기어박스 구조(축은 drill pin 으로 교체함)

또한, 저희는 새로운 서스펜션 구조를 생각했다. 기어박스에 걸리는 모멘트를 잡아주기 위해서, 4-bar linkage 를 알루미늄 프로파일과 기어박스 사이에 다는 구조를 구상했다. 기존에 잘못 참고하였던 Mcpherson single strut 구조 또한 그 구조를 가지고 있었지만, 그냥 4 bar linkage 에 쇽 업소버를 달기로 합의하였다.

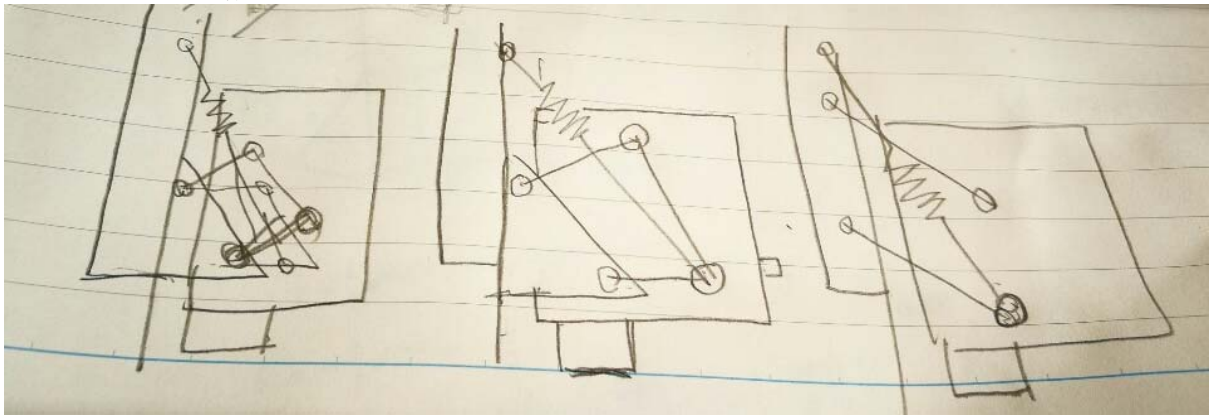


Fig 38. 바뀐 서스펜션 시스템

또, 4 bar linkage 를 설치할 때, 기어박스에 모멘트가 걸리는 것을 최대한도로 막기 위해서 기어박스와 차체를 이어주는 두 링크의 크기를 같게 만들어 평행사변형으로 제작하였다, 또한 링크를 고정해 줄 수 있는 골무를 flexible TPU 필라멘트로 출력했다.
또한 프린팅된 쇽 업소버의 낮은 피로도 때문에 어쩔 수 없이 시판하는 쇽 업소버를 구매했다. 대신 스트로크가 긴 쇽 업소버를 선택해서 평형점을 조절해 차량 하중을 지지할 수 있도록 구매하였다.

## [week 11]

4 bar linkage 를 구성하는 링크가 도착했고, 조립을 시작하였지만 또다시 크고 작은 문제가 발생하였다. 기어박스에 또다시 모멘트가 발생해서 덜그덕거리는 소리가 났고, 저희는 이 문제가 링크 사이에 있는 기하공차 때문이라고 생각해서 나사를 더 조이고 헐렁거리는 부품을 다시 제작했다.

또한, 기어박스에 들어간 나사들이 돌아가면서 다른 나사를 건드리는 문제가 발생했다. 설계할 때 여기까지 고려해야 했었지만 이를 간과해서 큰 문제가 발생해 버렸다. 머리가 없는 나사를 구해서 이를 해결했다. 그러나 결국 모멘트 문제를 해결하지 못해서 기어박스 부분을 1 학기 때로 롤백했다.

[week 12~13]

Fig39. 뒷부분 볼 컨테이너

모든 부품을 롤백한 뒤로, 더 이상 하드웨어가 완성될 때까지 기다릴 수 없었기 때문에 기존의 방식을 유지하면서 진동을 잡는 방법을 고려하였다. 저희가 서스펜션을 만들려는 가장 큰 이유는 미션을 진행하는 동안 네 바퀴가 모두 땅에 닿아 있게 하기 위함이었다.
하지만 기존 방식으로는 미션을 진행하는 동안 간혹 4 점 지지가 되지 않는 부분이 있었기 때문에, 해결 방법으로 기어박스를 잡고 있는 알루미늄 프로파일 전체를 눌러 주어서 무게의 균형을 맞춰주었다. 그래서 스프링을 다음과 같이 프로파일에 연결하여 무게의 균형을 맞춰 주었다.

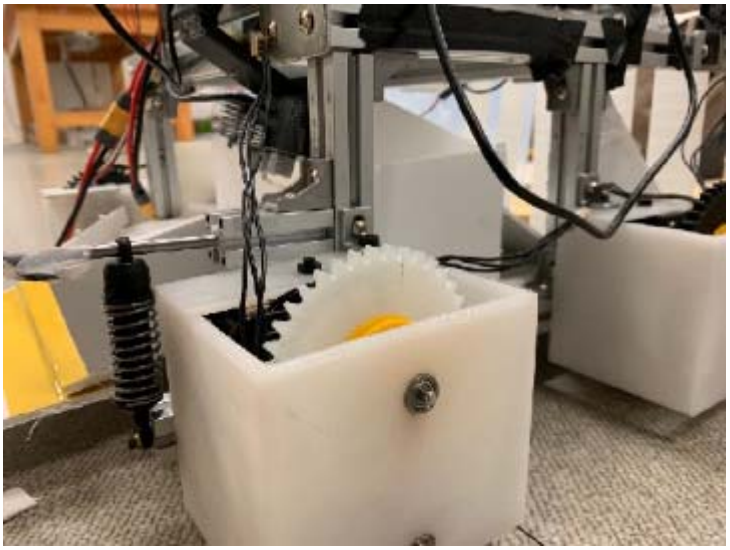


Fig40. 최종 기어 사진

여전히 빠른 속도로 이동하거나 회전을 할 때는 문제가 발생했지만, 용수철의 길이를 조정하여 문제를 그때그때 해결하였다.
또한, 뒷부분 공 컨테이너 유닛을 수정하였다. 실제로 차량을 작동시켜 보니, 차량이 바구니에 닿을 때 회전하는 문 부분의 길이가 너무 길어서 SLAM 이 살짝만 틀어져도 바스켓에 공을 넣지 못하는 문제가 발생했다. 그래서 뒷부분 회전하는 문의 길이를 최대한 줄이기 위한 계산을 시행하였다.

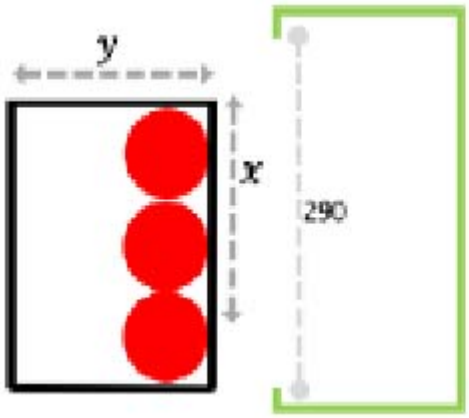



Fig41.

우선 공 세개가 들어갈 공간만 남겨두고 나머지 부분은 모두 잘라내었다.

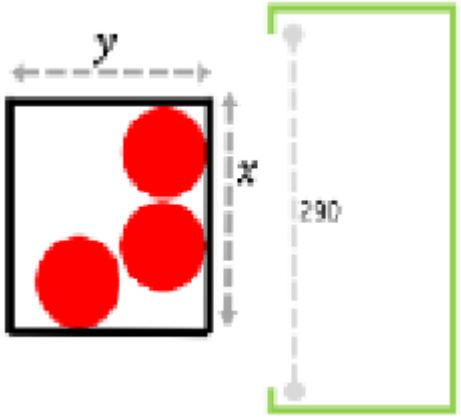



Fig42.

그래도 공간이 부족했기 때문에, 모듈의 길이를 약간 늘린 후 공 하나를 위와 같이 배치하여 공간을 더 줄였다.

그러나 이 시기에 개/고양이의 사진을 센싱하고 다가갈 때 오차가 많이 발생했기 때문에, 저희는 아예 회전문의 길이를 최소화할 수 있는 방법을 생각했다. 그래서 다음과 같이 두 개의 공에 접하는 선으로 지지대를 만들어 그 위치까지에 해당하는 회전문을 모두 잘라버렸다.

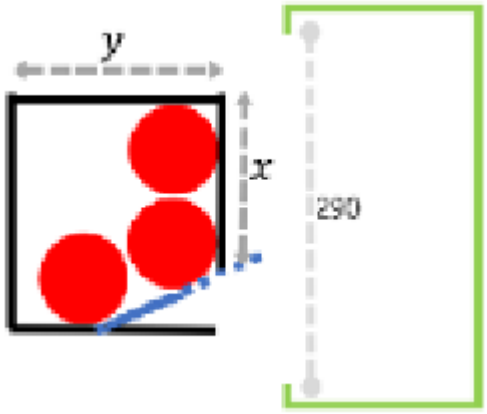



Fig43.

이렇게 뒤쪽 모듈의 길이를 줄이자, 바구니의 길이인 290mm 의 절반도 안되는 크기로 회전문의 길이를 줄일 수 있었다. 이 이후에는 바구니에 회전문이 걸려서 릴리징이 되지 않는 문제는 없어졌다.

또, 균열이 나 있던 있던 알루미늄 프로파일을 새 것으로 교체했다.

[week 14]

.서보 모터를 고정하는 폼 보드를 PLA 판으로 교체하였다. PLA 판을 볼트로 알루미늄 프로파일에 고정해서 서보 모터가 더 안정적으로 돌아갈 수 있게 했다. 무게가 늘어났지만, 서보 모터가 문제를 일으키는 경우는 더 줄어들었다. 차체가 거의 완성되었기 때문에 heat solution 을 생각해보았다.

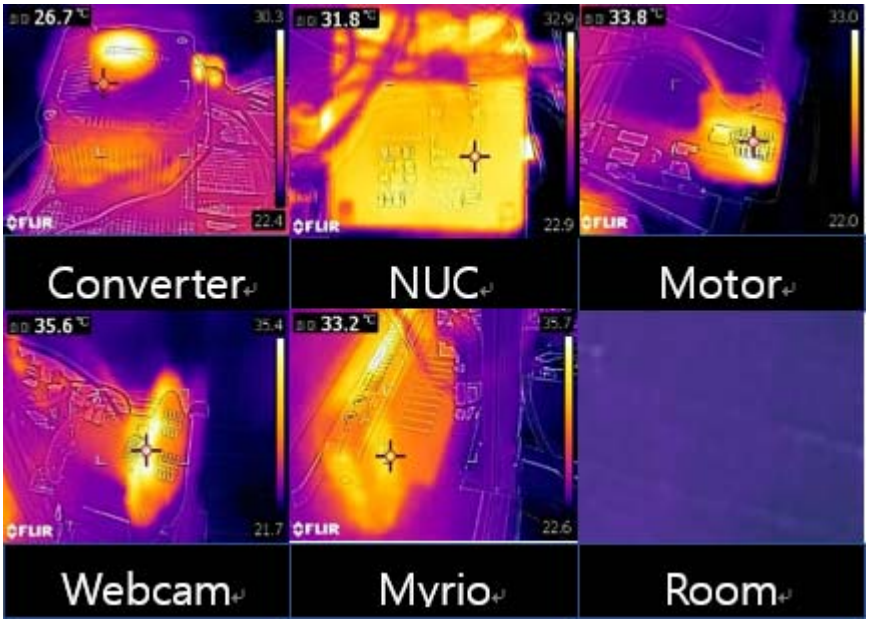



Fig 44. 1 학기 열 사진

Figure 은 봄학기 창의적 시스템 구현 1 에서의 heat solution 이후 주요 부품들의 열 사진이다. 당시에는 기준 온도가 35 도 였지만 창의적 시스템 구현 2 가 되면서 기준 온도가 32 도로 바뀌었다. 이번 학기 온도 기준으로 Motor, Webcam, Myrio 가 32 도가 넘고 NUC 도 온도가 거의 32 도에 근접한다. 또한 창의적 시스템 구현 2 가 되면서 Webcam 을 1 개 더 추가하고 Servo Motor 가 1 개 더 추가되어 추가적으로 heat solution 을 진행했다.

Fig 45. Motor, Fin

Heat solution 은 전 학기와 동일하게 Motor Fan, Fin 을 사용했다. 가장 온도가 높은 Converter 에 Motor Fan 을 사용하고 나머지 Webcam 3 개, Motor, Servo Motor, NUC, Myrio 등에는 Fin 을 사용해 heat solution 을 진행했다. 모든 과정이 끝난 후 사진은 아래 Figure 와 같다.

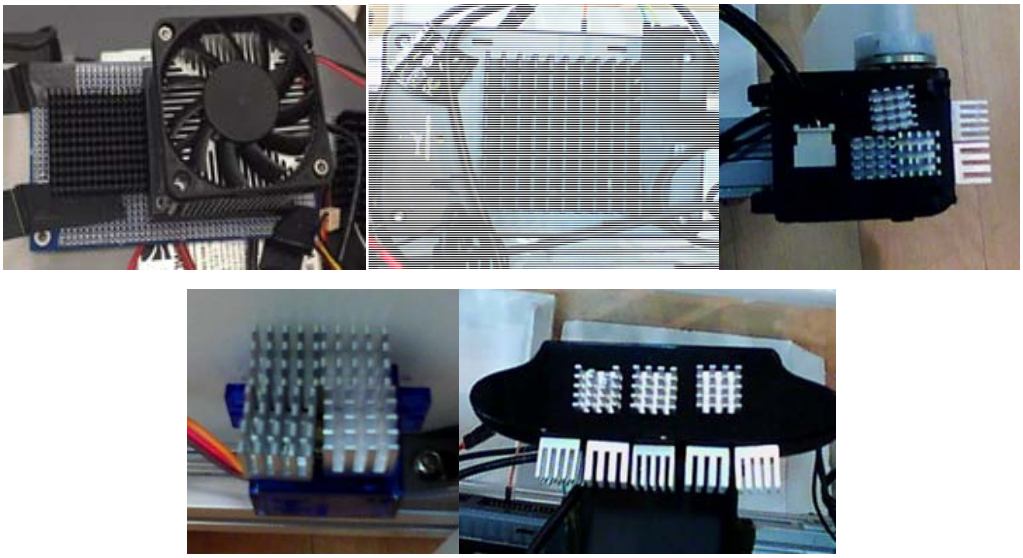


Fig 46. Fin 을 붙인 부품

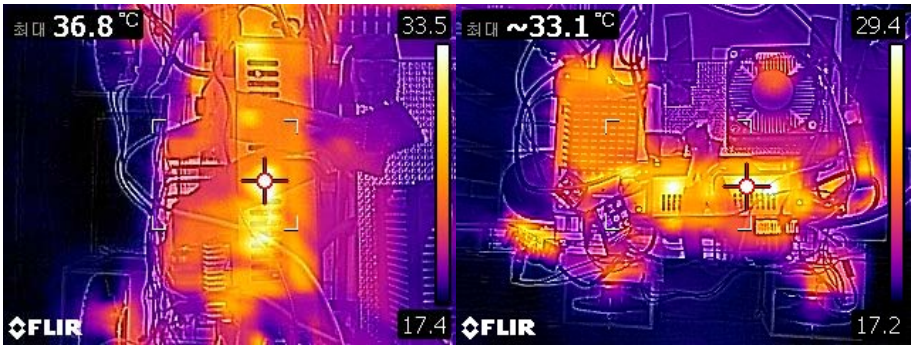



Fig 47. NUC 의 열 분포

NUC 에서는 자체적으로 온도를 낮추기 위해 뜨거운 바람이 나오는데 이로 인해 바람이 나오는 부분의 온도가 높게 측정되고 주변 부품들의 온도가 올라가는 문제가 있었다. 위의 Figure 에서 보이듯이 뜨거운 바람이 나오는 중에 온도를 측정하면 36.8 도 까지 올라가고 오랜 시간 NUC 을 사용하면 뜨거운 바람이 나오는 중이 아니어도 해당 부분의 온도가 33.1 도 까지도 올라간다. 이를 해결하기 위해 Motor Fan 과 NUC 의 위치를 조정하여 Converter 의 Motor Fan 이 NUC 의 heat solution 에도 도움이 되도록 하였다. 아래 Figure 은 Motor Fan 을 on, off 했을 때의 온도를 측정한 결과이다. Figure 에서 보이듯이 Motor Fan 을 on 했을 때는 24.8 도로 측정되고 off 했을 때는 30.7 도로 측정된다. 즉 Converter 의 Motor Fan 을 통해 NUC 의 온도에도 영향을 주도록 하였다.

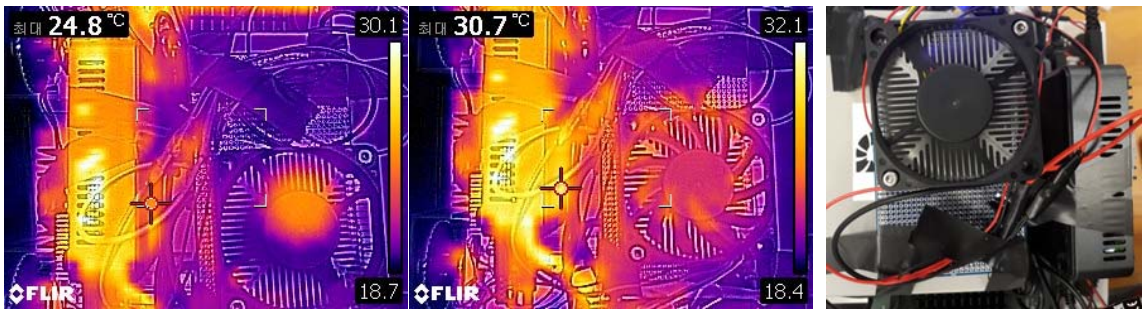



Fig 48. Fan 사용후 NUC 의 열 분포

[week 15]
서보 모터가 안에서 망가져 버렸다. PLA 판에 서보 모터를 무리하게 끼워서 발생한 문제라 생각되어 서보 모터를 새로운 것으로 교체하고, PLA 판도 공차를 늘려서 새로 만들었다.
픽업 모듈이 망가져서 새로운 픽업 모듈을 프린팅했다.
또, SLAM의 위치가 너무 높아 벽 너머를 보는 상황이 발생했다. SLAM을 고정하는 알루미늄 프로파일을 잘라내서 길이를 줄이는 데 성공했다.
조정이 끝난 앞, 뒤쪽 카메라를 고정시켰다.

[week 16]
잘 돌아가던 서보 모터가 첫 신호를 잡지 못하고 무시해버리는 경우가 발생했다. 이런 경우가 처음이어서 당황하다, 박해원 교수님께서 이런 경우는 서보 모터의 static friction 때문이라고 조언을 해 주셨다. 그래서 static friction을 줄이기 위해서 서보 모터를 계속 가동시켰고, 그제야 문제가 해결되었다.

# 4. Presentation

Throughout this Capstone Design II course, we have succeeded in making some design advancement in both software level and hardware level throughout our whole system for performing more complex tasks that needs to be performed by our module. These advancements, which has been explained in detail during both the first design review at the first half of the semester and the final presentation which was held at the end of the semester, will be explored in detail below.

During the first half of this Capstone Design II course, our group performed some changes in it's system design using the module that were specifically made for accomplishing goals that were fixed in Capstone Design I. As we might have remembered from Capstone Design I, our goal in system design was to avoid red balls, pick up three blue balls, and release them into the basket, In order to do so, we designed a pickup and releasing blade that is connected by a slope made of lightweight cardboard at each end. We did the following so that when a ball is gathered by the blade, it will move towards the end of the releasing blade. However, since we need to perform three additional missions; gather additional red balls, distribute balls according to its color, and release these balls into two different baskets, the following hardware design improvements were made.

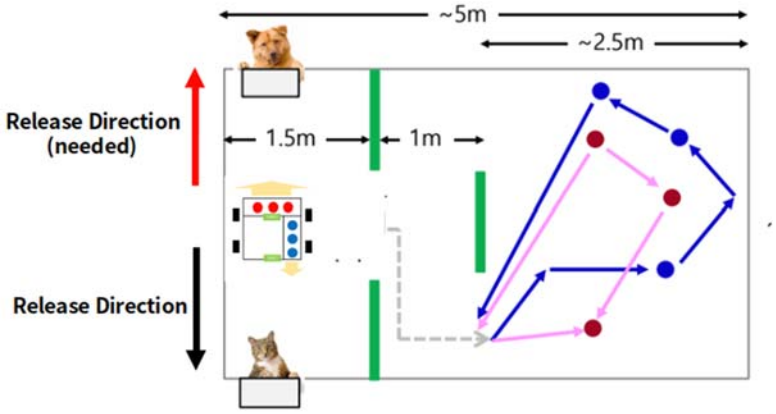



Figure 49

During the first half of Capstone Design II, we made an additional red ball container that will be useful for storing three red balls at the rear part of our platform. Using the TRIZ method, we finalize our design idea by placing the red ball container in asymmetrically perpendicular orientation with respect to the original blue ball container. It was made in that way so that the length of our module is minimized; making our module more compact and reliable. In addition to that, we managed to handle the ball distributions by placing an adjustable slope that is controlled by a servo motor inside our chassis. This will be useful in segregating balls so that a blue ball goes to its container whereas a red ball goes into its corresponding container. Finally, since we have discussed in detail that we tried to make the rear container to be perpendicular to the original blue ball container, it is now necessary to put a design consideration for releasing these red balls. Using the universality principle in TRIZ method, we made the wall that holds the red balls at the rear container to act as a slope that is useful for releasing these balls. In order to do so, we first connect the wall and bottom slope with a pin joint attached to a weak magnet, so that when the container wall collides with the bottom of the basket, it will trigger the slope and eventually release the balls inside the rear container.

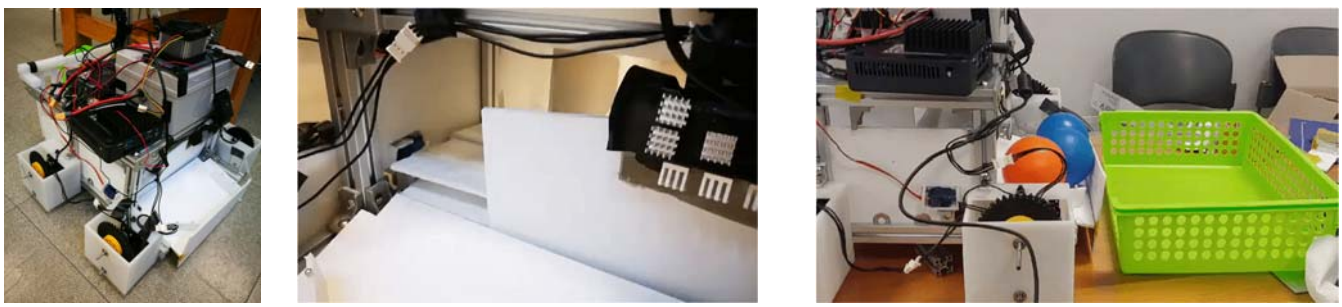


Figure 50

Beside performing some modifications on the hardware design, our group also utilized additional modules that are useful for accomplishing the set of tasks that will be done by the module in the software level. In Capstone Design II, now we are using Convolutional Neural Network (CNN), which is known as a class of deep networks most commonly applied to analyzing and performing image recognition. In Capstone Design II, This network is utilized for classifying whether an image that is observed by our webcam is a cat image or a dog image, which is crucial for making sure that our module releases red balls when a cat image is detected by the module, and when a dog image is detected by the module itself, it will begin releasing the blue balls within it. To execute these tasks correctly, therefore our group should make an attempt in improving the performance of CNN that will be utilized so that it will become a highly robust image recognition system. This can be done easily by doing trial-and-error method through changes that are made in the network training parameters while we are training this image-classifying network. After successfully tuning the network parameters that will result in the best network performance, we'll obtain a network that is able to perform the image recognition properly under any extreme conditions that our platform may encounter in.

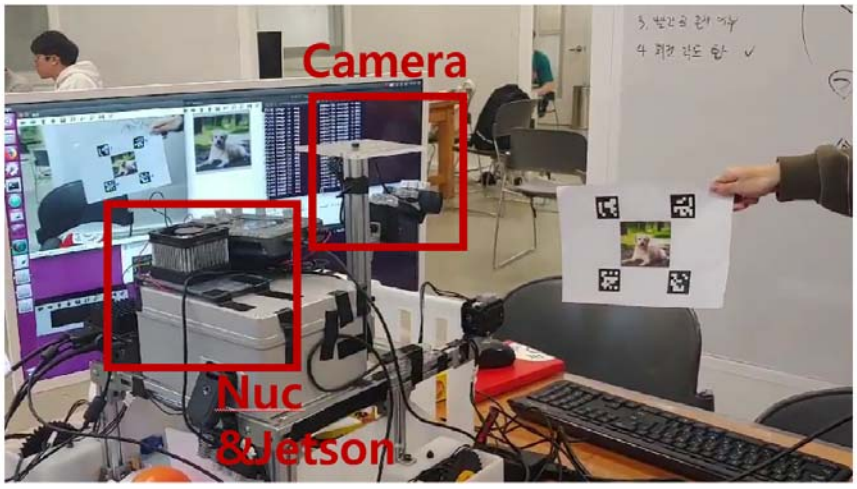
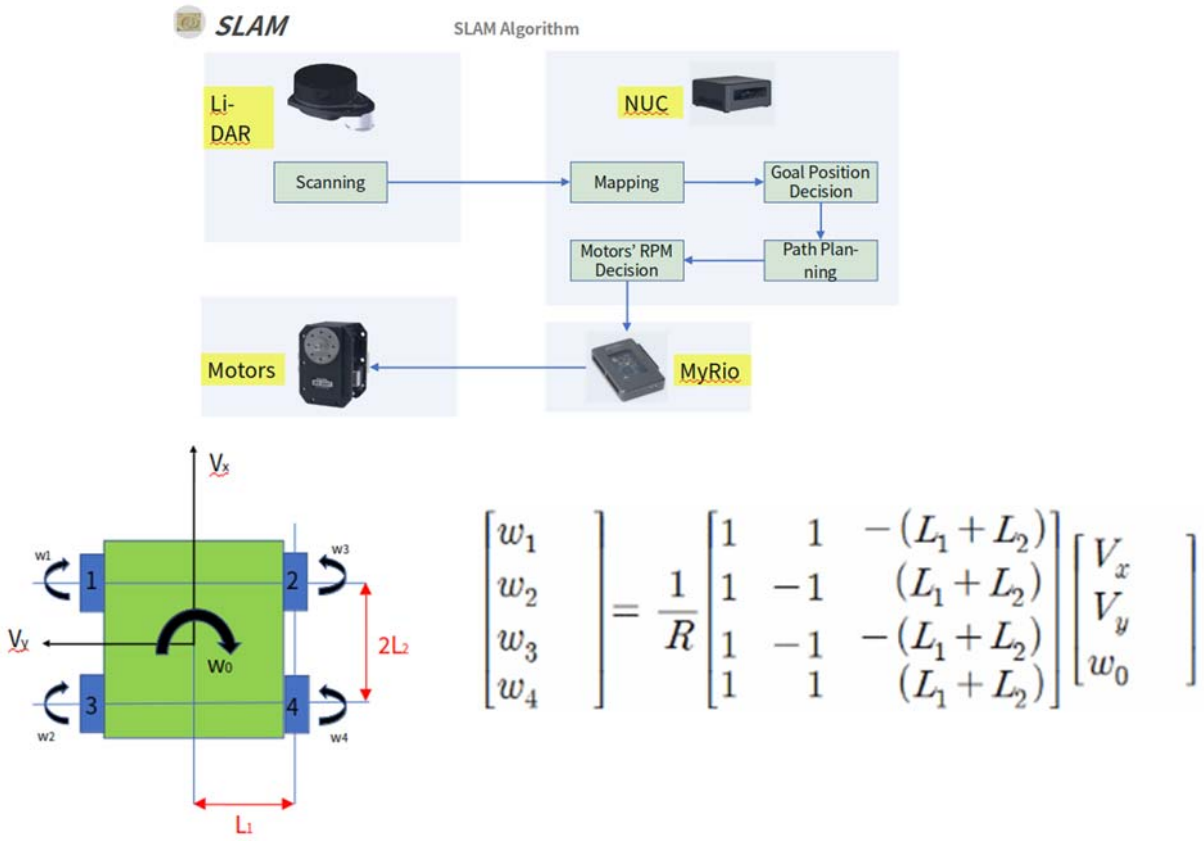


Figure 51

Another module that we utilize in Capstone Design II is SLAM, also known as Simultaneous Localization And Mapping. SLAM is mainly useful for performing maze escape that it may encounter while performing the tasks that need to be done. To utilize SLAM in our module, we firstly need to place the LiDAR at a distance of 45 cm so that it can properly detect walls and its surrounding environment. In order to understand how SLAM algorithm works, first of all, LiDAR needs to scan the wall and transmits the data to NUC, which will use the scanned data for mapping. Mapping data is then utilized for determining module's end position. Then, using our customized module data like size, the optimum path, velocities and orientation are decided, we perform the calculation of motor's RPM by using the equation denoted below by the previously explained parameters, which will be transmitted to myRIO. myRIO makes wheels rotate, that the module will be able to move. Once it moves, LiDAR scans new data and the similar procedure goes on until the module reaches its final destination. The SLAM algorithm, motor's RPM calculation, as well as picture of how it practically works is shown as below.




$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \frac{1}{R} \begin{bmatrix} 1 & 1 & -(L_1+L_2) \\ 1 & -1 & (L_1+L_2) \\ 1 & -1 & -(L_1+L_2) \\ 1 & 1 & (L_1+L_2) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ w_0 \end{bmatrix}$$

Figure 52

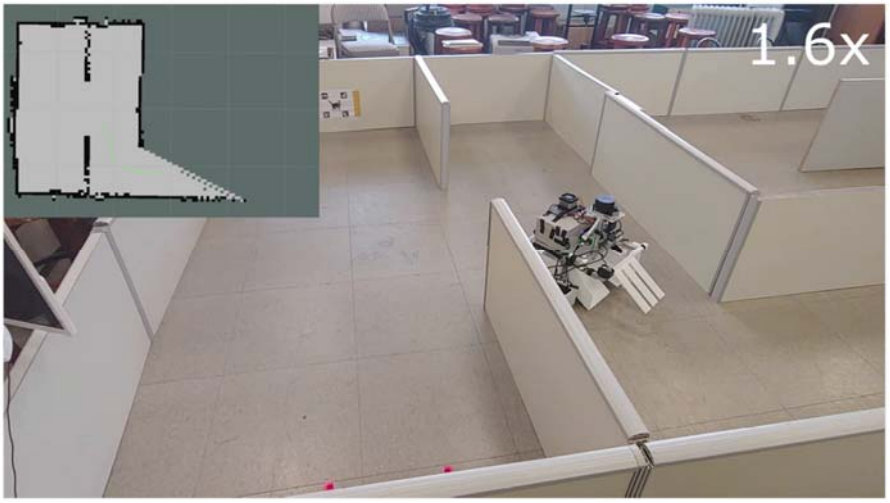



Figure 53

Picking up where we have left off from the first half of Capstone Design II, we also perform several tweaks and modifications in both hardware and software parts within our module and finalize the design of our module that will be able to perform tasks successfully. To briefly understand how our module works, we are going to the system level tour within our module that can be shown in the following figure. The module is arranged in the following manner in order to achieve the lowest center of gravity..

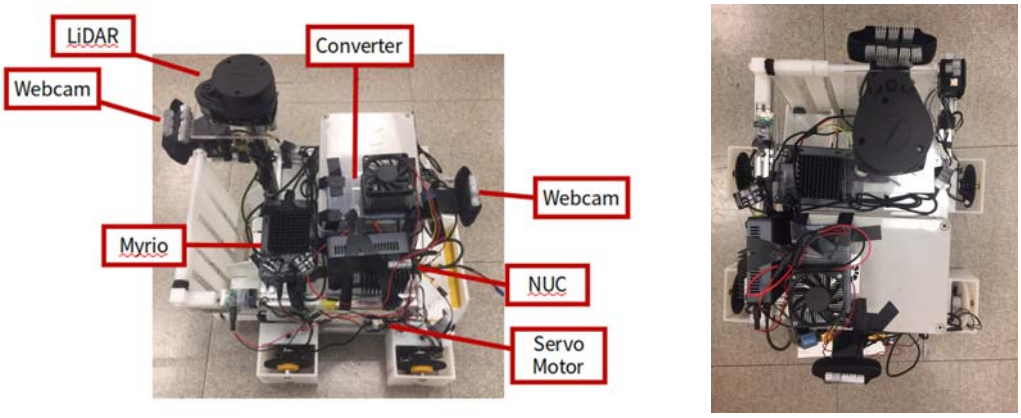



Figure 54

In describing how our module works, first of all, our module consists of actuators that covers both motor and servo motor for pickup and distributing balls according to its color. In addition to that, we also have a releasing module in both front and rear end that has been covered thoroughly in the previous paragraphs. It is also important to note that our module is now equipped with three webcams instead of the original two we had. The first two webcams are installed for checking the ball color and ball positions at the front part of our module, whereas the last webcam is definitely utilized for finding the image of cat so that it will be able to release the red ball at the rear container. Also, we installed a fan that will be useful for controlling the temperature of our system, as we may observe in the figure below. Finally, an additional spring was attached at the front half of our module that will balance the tilting that occurs at the front half of our module, which can also be observed below.

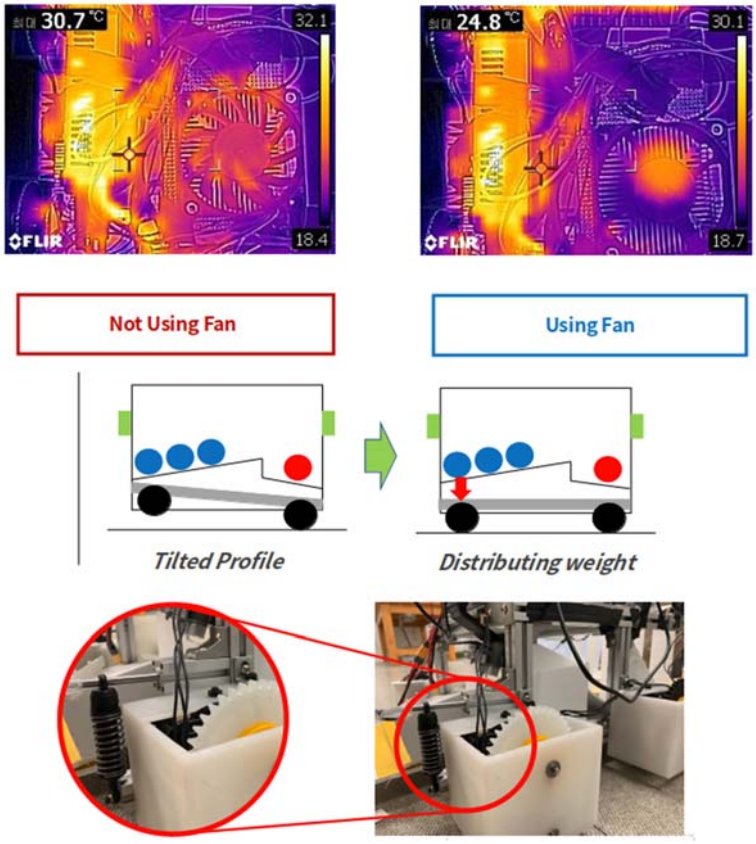



Figure 55

In addition to these module descriptions we've made, we put an emphasis in solving the contradiction problem that we encountered in designing our module. The first contradiction we encounter is when we are figuring out where to place the red ball container. As we may have noticed before, we would like to move our module as short as it can be while having two separate ball containers for releasing our balls. Therefore, instead of rotating the module to release the balls from the front part of our module, we instead release balls at the rear part from the rear container within the module itself. After coming up with possible placements of the red ball container within the system, we came up with the asymmetric design that has been chosen before. Similarly, we

would like to release balls at the rear container without using any power asserted from the motor, but in order to do so, an adequate amount of force is required to release them. In solving this problem, we decided to release these balls when the rear container wall hits the basket; acting as a bridge for these balls to enter the basket.

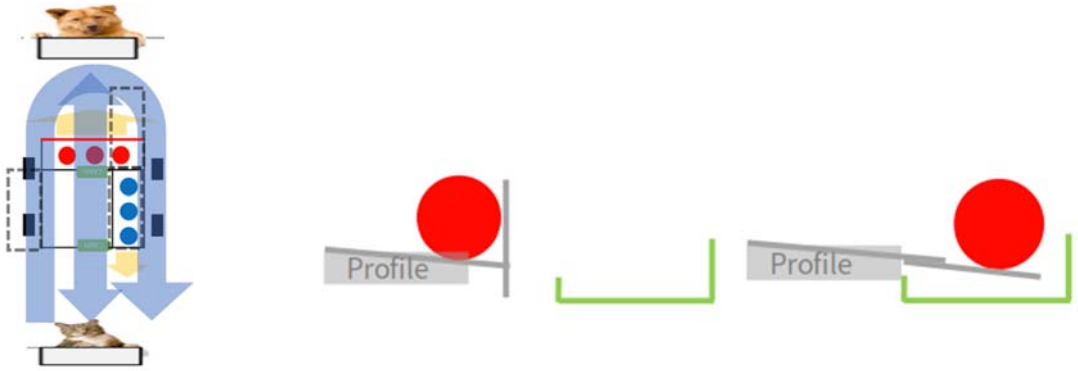



Figure 56

Another contradiction problem that we need to tackle is when we are about to design the red ball container. In designing our rear slope, there's a chance that the rear part of our module hits the end of the basket. We also figured out that additional space of 30mm is needed for releasing red balls properly, and releasing these balls quickly is a top priority. Therefore, we made specific goals that our module needs to do. First, we need to fix the width of the rotating wall to be shorter than half of the basket's width. Next, we need to make sure that the upper bound of the container's length is 110mm, and fix the release time of the ball to be shorter than a second. In order to do so, we begin shortening the width of the rotating wall. However, when we shrink the width of the rotating wall, we noticed that length required to maintain the balls inside the container will increase. Therefore, to solve this problem, we decided to create a support tangent to two balls that were tilted before, resulting in the decrease of width of the rotating wall. When we do so, the width of the rotating wall will be proportional to the container's length. Now, for the timing constraint, from the x-t and y-t plot of our releasing container, we consider only the white-colored area that satisfies previous goals we've set. By this, we found out valid theta values, which is the angle between support and rotating wall, and we are utilizing the lower bound of these valid theta values for obtaining the smallest possible dimension that satisfies the design constraints that were fixed before.

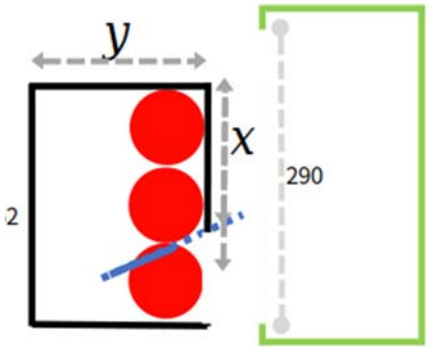



Figure 57

After going through the hardware descriptions that our module has, we now proceed in analyzing our module in software level. In order to understand how our module works properly, the following flowchart is presented to understand how our module decides to perform its tasks in Capstone Design I. Basically, we fix our module to perform three main tasks as shown in the previously noted diagram. First of all, it needs to detect ball within its surroundings. When a blue ball is detected, our module will start approaching that ball and pick it up when it's possible to do so. Once there's no blue ball in the field, our module will start approaching the basket to release these balls.

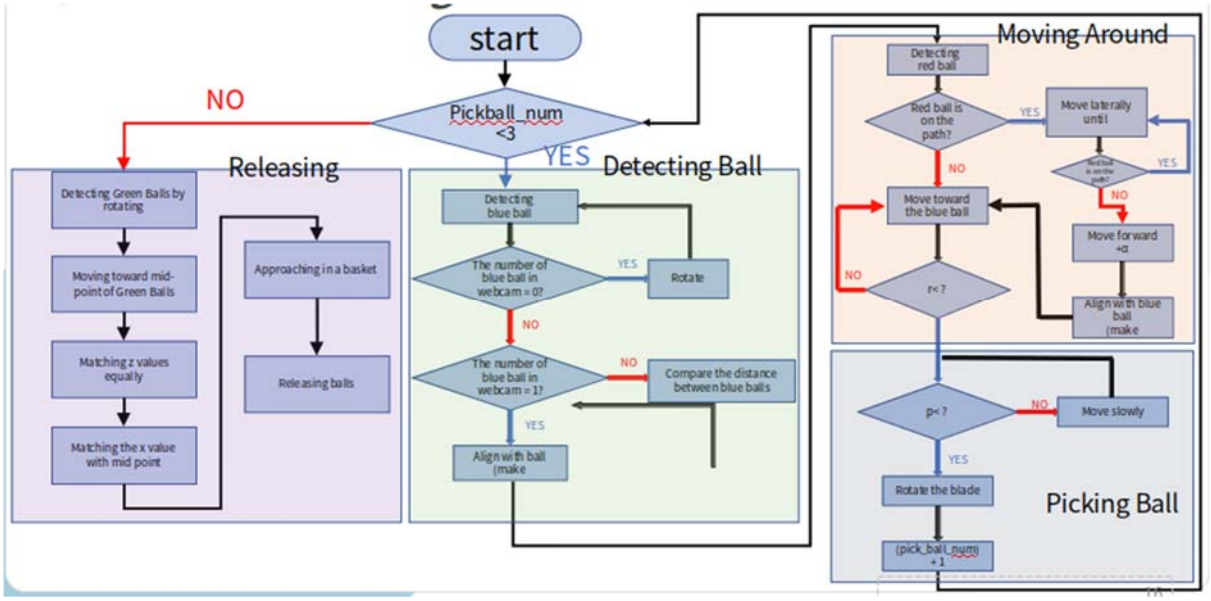



Figure 58

However, for this Capstone Design II project, instead of controlling what the module needs to do specifically, we are utilizing both DQN and CNN that mainly decides what our module needs to do, as shown on the following diagram. Since we have covered CNN in the previous paragraphs, now it's time to describe how DQN tries to accomplish the set of tasks that needs to be done by our module.

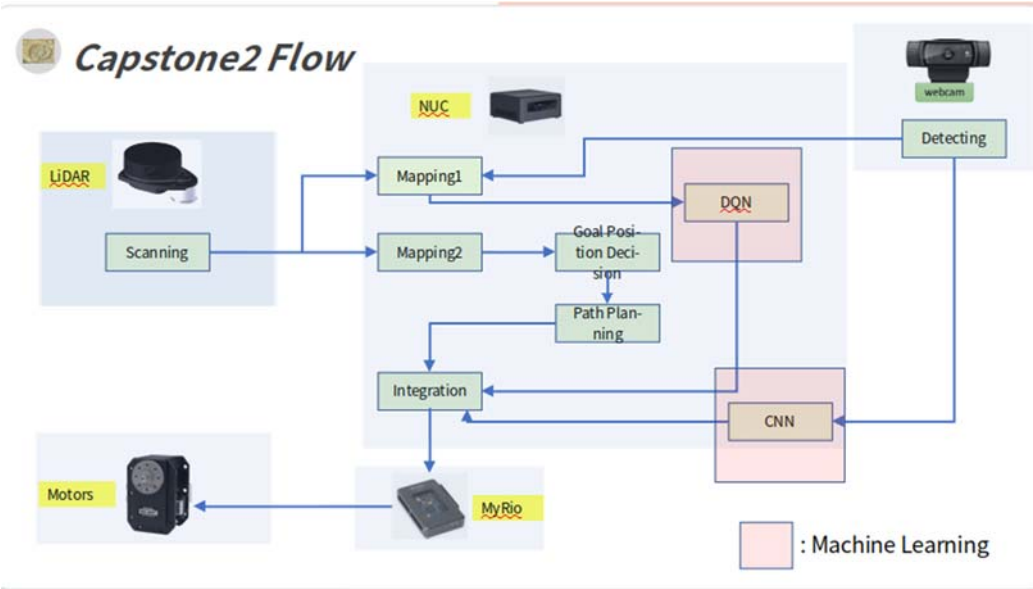



Figure 59

DQN,

which stands for Deep Q-Network, is an algorithm that aims to update Q functions after each iteration to return sequence of actions that maximizes the expected scores. It is utilized for finding the optimum path that will be taken by our module that utilizes it to accomplish the missions that our module needs to do in Capstone Design II. To simulate how the trained DQN works within the environment employed for accomplishing tasks that we need to perform, we first tried to simulate its performance in the virtual machine first before proceeding to try it out in the real environment.


Figure 60

To obtain the best possible model of DQN that will be utilized for accomplishing the tasks that we need to perform, we fix several positive and negative reward policies that our module receives throughout each iterations where it is performing tasks needed. First of all, the module will receive +200 points when there exists a ball right in front of the reward region, and we put an additional 50 points for each ball being successfully picked up to the whole system. When the module collides with either obstacle or ball, -10 points will be given, and -0.5 points will be given when zero balls are detected by our module. These set of policies is hugely beneficial for our module in approaching a ball and picking it up when it's possible in an effective manner.

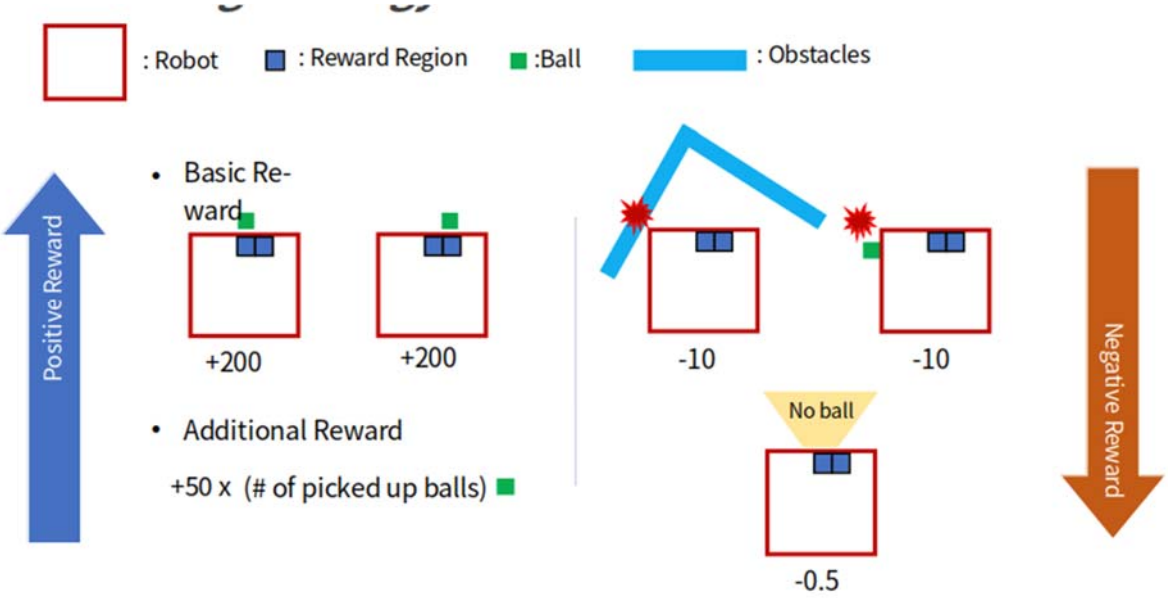



Figure 61

From these, we might notice some differences between the hierarchical software implementation used in Capstone Design I and Capstone Design II. In Capstone Design I, as we might have learned in the previous slides, we are mainly in charge of determining policies that our module needs to perform, so bugs within our policies can be spotted and fixed easily. However, in Capstone Design II, we need to deal with more constraints that need to be tuned to obtain the optimized network through training that takes approximately 16 hours. In addition to that, these constraints that have been explained before in optimizing our network have no clear objectives in what it tries to accomplish, so that changing a constraint within the network itself will always make our module perform in a way that we won't be able to expect. This makes it harder for us in determining the best performing network

hard to be analyzed, and it takes a lot of training and error experiment in finally figuring out which module performs the best as we would expect.

In preparing both the first and final design review, our group decided to start creating the presentation files one week before the presentation day. Once we are finished with the presentation files, the presenter skim through all the slides that were made and start making a presentation script to explain what is going on inside the presentation file. But before finalizing the presentation, we decided to put out a rehearsal with our advising Professor in order to gain feedback with respect to our presentation contents and what the presenter needs to emphasize more in the slides that has been made before. After going through these revisions, we finally decide to put out one more rehearsal to check whether the presentation covers what needs to be explained in the grading criteria as well as checking the time that has been passed on the presentation itself. From these processes, we are able to perform the presentation in a clear and concise manner as we have expected.