2019 Fall

Capstone Design 2

Final report

Team name: Team C

Team professor: 최세범

Teaching assistant: 박기서

Member: 김형준, 부준요, 오창균, HUIZHI, 류상우, 한동훈, 김준형

Submission date: 2019/12/20

Contents

1.Individual expressions

김준형)

현재 인기있는 기술들을 활용하여 로봇을 만드는 수업이었지만 기술의 원리와 이론에 대해 이해하는 것과는 거리가 멀어 아쉬웠습니다.

오창균)

처음 머신러닝을 접하는 계기가 되어 인상적이었습니다. 처음에는 막막하였으나 특강을 통해 자세한 내용을 배울수 있어서 좋았습니다. 한학기 동안 몰두하여 많은 내용을 배울수 있어서 좋았습니다.

HUIZHI)

It was an intense course that I learned a lot from designing a robot. Engineering based designing is to optimize the production and its efficiency. Process is more important than the outcome and I will take what I learned from this course with me to the future. Thankful for being in this team.

류상우)

창시구 1 과 2 크게보면 1 년짜리 수업이고 명성이 자자해 두려움이 컸다. 마지막 과제를 완벽히 수행하지 못해 아쉬움이 있지만 실패를 통해서 많은 것을 배울 수 있었다고 생각한다. 좋은 팀원들을 만나 뜻깊은 한 학기였다고 생각합니다.

부준요)

한 학기 동안 많은 것들을 배울 수 있었습니다. 특히 여태까지 접해보지 못했던 CNN 이나 SLAM, DQN 등을 한 번이라도 사용해 보는 것이 재밌었습니다. 여태까지

접해보지 못했던 내용들을 새롭게 배워야 하고, 이를 실제 코딩으로 구현하는데 많은 시간을 투자해야 하는 터라 다른 과목이 좀 빡빡하게 겹치면 매우 힘들었을 것이란 생각이 들었습니다. 한 학기 동안 같이 해준 팀원들, 도와주신 조교님들, 그리고 저희를 이끌어주신 최세범 교수님까지 정말 감사드립니다.

한동훈)

많은 것을 배울 수 있는 한 학기 였지만 힘든 부분이 많았습니다. 하지만 최근의 트렌드인 인공지능에 대해 알게 되어 좋았습니다. 조원들 모두 열심히 참여해줘서 감사합니다.

김형준)

창시구 1 에 이어 창시구 2 를 듣게 되었는데

SLAM 이라는 기능이 매우 생소했다.

곡선 무빙을 통해 미로를 빠져나가는데 생각보다 제약이 많은 것 같았다.

조금 더 깊게 알아간다면 이 기술의 진가를 알 수 있을 것이라는 생각이 들었다.

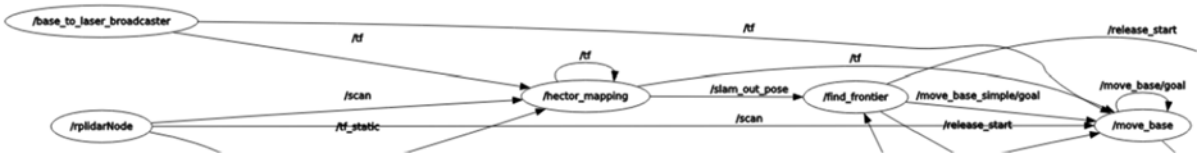
어려운 과제를 함께 해준 팀원들에게 감사한다.

2. ROS

1 SLAM

1.1 Outline

로봇의 미로 통과, 공 수집 후 복귀를 위해 Hector SLAM 을 이용하였다. 아래의 그림은 전체 로봇 시스템 중 SLAM 알고리즘에 관련된 부분을 캡쳐한 것이다.

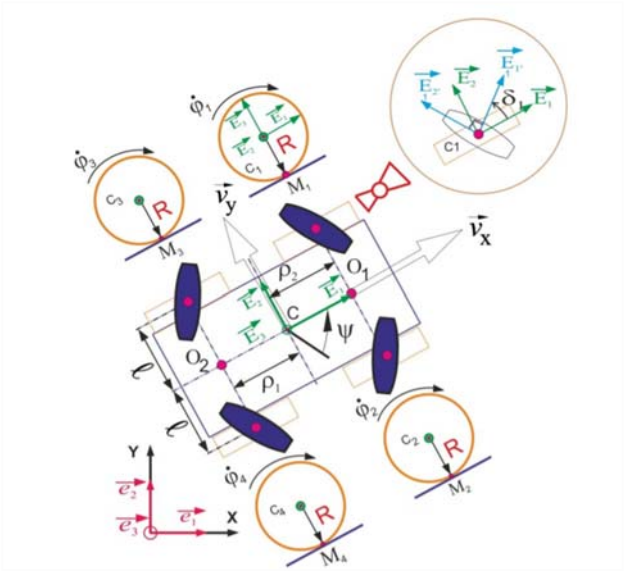


rplidarNode 에서 RPLidar 가 로봇 주위에 존재하는 장애물을 감지한다. 측정 결과를 scan 메시지를 통해 hector_mapping 노드에 전달하면 이를 바탕으로 로봇을 기준으로 미로의 지도(이하 Map)를 생성한 후, Map 내부에서 로봇의 현재 위치를 파악한다. 로봇의 위치 정보와 Map 을 slam_out_pose 로 find_frontier 노드에 전달한다. 이를 토대로 find_frontier 노드는 rpLidar 에서 감지하지 못한 영역(이하 frontier)이 가장 넓은 곳을 목적지로 설정하여 move_base 노드에 전달한다(단, 미로 복귀 시에는 로봇의 시작점으로 전달한다). move_base 노드는 목적지와 로봇의 상대 위치 정보를 기반으로 로봇의 선속도 및 회전 각속도 정보를 cmd_sub 노드에 cmd_vel 메시지로 보낸다. cmd_sub 노드에서는 수신한 cmd_vel 에 따라 모터를 가동한다.

1.2 Node Modification

1.2.1 cmd_sub Code Modification

Mecanum Wheel 의 역학을 기반으로 수신한 cmd_vel 의 속도 정보에 기반하여 로봇이 운동할 수 있도록 cmd_sub 에서 모터 회전속도를 입력하였다. 로봇의 선속도와 로봇 중심 기준의 회전 각속도가 주어졌을 때, 각각의 Mecanum Wheel 의 회전 속도는 다음과 같다.



$$
\begin{aligned}
v_x - v_y - (l+\rho_2)\,\dot{\psi} &= R\dot{\varphi}_1\,,\\
v_x + v_y + \dot{\psi}(l+\rho_2) &= R\dot{\varphi}_2\,,\\
v_x + v_y - \dot{\psi}(l+\rho_1) &= R\dot{\varphi}_3\,,\\
v_x - v_y + (l+\rho_1)\,\dot{\psi} &= R\dot{\varphi}_4\,.
\end{aligned}
$$

v_x, v_y ψ 는 로봇의 x, y 축 방향 선속도, 로봇의 회전 각속도를 의미하며, $\dot{\varphi}_1$~$\dot{\varphi}_4$는 Mecanum Wheel 의 회전 각속도를 의미한다. R 은 Mecanum Wheel 의 반지름이다. 전방 Mecanum Wheel 은 로봇의 중심 C 로부터 x 축 방향만큼 ρ_2만큼, y 축 방향으로 l 만큼 떨어져 있다. 후방 Mecanum Wheel 은 x 축 방향으로 ρ_1만큼, y 축 방향으로 l 만큼 떨어져 있다

(자료 출처: A DESCRIPTION OF THE DYNAMICS OF A FOUR-WHEEL MECANUM MOBILE SYSTEM AS A BASIS FOR A PLATFORM CONCEPT FOR SPECIAL PURPOSE VEHICLES FOR DISABLED PERSONS
(URL: https://www.db-thueringen.de/servlets/MCRFileNodeServlet/dbt_derivate_00030822/ilm1-2014iwk-041.pdf))

1.2.2 Map 관련 yaml 파일 수정

Find_frontier 노드에서는 yaml 형식의 파일을 통해 로봇 경로의 특성을 수정할 수 있다. 로봇의 경로를 결정하는 변수와 역할에 대해 아래의 표에 정리하였다.

변수명	기능
Footprint	로봇의 크기 관련 변수 RPLidar 기준으로 로봇의 x, y 방향 크기 결정
Inflation Radius	장애물의 한 포인트를 중심으로 Inflation Radius 를 반지름으로 하는 가상의 원 영역 회피
Cost Scaling Factor	Costmap 에서 각 장애물 포인트의 Cost Value 조절을 위한 상수 값이 증가하면 Cost Value 가 감소한다

Footprint 와 Inflation Radius 의 경우 크기가 커짐에 따라 벽에 충돌할 위험이 줄어든다는 장점이 있지만 경로 생성 조건이 까다로워져 미로 통과가 어려워진다는 단점이 존재하였다. 따라서 이상적인 값을 실험적으로 결정해야 했다. 초기 실험 당시에는 로봇이 미로 벽에 충돌하는 문제가 빈번하게 발생하였다. 또한 부품의 증가로 인해 로봇 자체의 크기가 커져 시뮬레이션 상 미로를 통과할 수 있음에도 불구하고 충돌이 발생하였다. 따라서 Footprint 와 Inflation Radius 의 값을 기본값에 비해 증가시킴으로써 문제를 해결하였다. 이에 따른 부작용으로 Cost Value 가 지나치게 증가하여 경로 생성에 실패하는 일이 발생하였다. 이를 해결하기 위해 Cost Scaling Factor 의 증가를 통해 전체적인 문제를 해결하였다. 최종적으로 선정된 변수의 값들은 다음과 같다.

footprint: [[-0.36, -0.29], [-0.36, 0.31], [0.2, 0.31], [0.2, -0.29]]
inflation_radius: 0.7
cost_scaling_factor: 1.30

1.2.3 Find_frontier 수정

미로 복귀를 위해 find_frontier 에서 제공하는 목적지의 좌표를 수정하였다. DQN 과정에서 수집된 공의 개수가 6 개이면 DQN 을 중단하고, find_frontier 노드에서 목적지를 미로 시작점 인근으로 지정하여 move_base 노드에 송신하도록 코드를 수정하였다. 아래는 수정된 코드의 일부이다.

```
if(DQN_ON == 2){
    cout<<"go to start point\n";
    goal_x = 0;
    goal_y = -0.6;
    goal_w = 1;
    float distance = sqrt(pow(pose_ptr_->pose.position.x,2)+pow(0.6 + pose_ptr_->pose.position.y,2));
    if(distance< 0.15){
      std_msgs::Int32 release;
      release_start = 1;
      release.data = release_start;
      release_pub.publish(release);
      exit(0);
    }
}
```

2 Readmarker and CNN

우리는 공을 내려놓는 위치와, 내려놓는 공의 색을 결정할 때 두 종류의 코드를 사용하였다. 첫 번째는 ArucoMarker 라는 특정 바코드를 찾아 그 안에 그려진 사진을 정사각형으로 변환하고, 각각의 marker 에 대한 거리와 좌우 픽셀 좌표를 계산해 전달하는 readmarker code 이다. 두 번째는 위 readmarker code 를 통해 얻어진 이미지가 개인지 고양이인지 판단하는 convolutional neural network code 이다.

2.1 Readmarker

이 code 는 ArucoMarker 라는 특정 바코드를 찾아 그 안에 그려진 사진을 정사각형으로 변환하고, 각각의 marker 에 대한 거리와 좌우 픽셀 좌표를 계산해 전달한다. 각각의 정해진 marker 가 있으면, 이 중 네 개의 marker 가 인식되어 사진이 있는 네 점을 확정할 수 있는 경우에만 작동하며, 이외의 경우에는 if 문이 실행되지 않는다. 그렇게 네 점을 확정한 후, 첫 번째로 각 marker 의 depth 와 픽셀 x 좌표를 계산하여, 이를 cmd_sub code 에 publish 하여 차체를 움직이는 근거로 삼는다. 두 번째로는 marker 를 통해 확정지은 네 점의 좌표를 이용하여 polygon 을 잡은 후, 그 polygon 을 정사각형 모양의 이미지로 변환한다. 그 후, 변환된 이미지를 개, 고양이 판단을 위해 neural network code 로 publish 하게 된다.

```
213         for(int i=0;i<4;i++)
214         {
215           if(std::find(markerIds.begin(), markerIds.end(), i+1)!=markerIds.end()){
216             indice[i]= std::distance(markerIds.begin(), std::find(markerIds.begin(), markerIds.end(), i+1));
217             marker_exist_tot = true;
218             marker_exist[i] = true;
219             wall_distance_msg.marker_exist = 1;
220           }
221           else
222             n_image1 = n_image1*0;
223         }
224         for(int i=4;i<8;i++)
225         {
226           if(std::find(markerIds.begin(), markerIds.end(), i+1)!=markerIds.end()){
227             indice[i]= std::distance(markerIds.begin(), std::find(markerIds.begin(), markerIds.end(), i+1));
228             marker_exist[i] = true;
229             wall_distance_msg.marker_exist = 1;
230           }
231           else
232             n_image2 = n_image2*0;
233         }
```

위 코드는 카메라에서 받은 이미지에서 ArucoMarker 가 존재하는지, 또한 네 종류의 marker 가 모두 존재하여 전달해야 하는 꼭지점이 확정되는지 판단하는 코드이다. Marker 가 하나라도 없는 경우 n_image 가 0 으로 바뀌게 되어, 결국 네 종류의 marker 가 다 있는 경우에만 n_image 가 1 로 정해진다.

```
234
235         if(n_image1==1)
236         {
237
238           outImageCorners[0] = cvPoint(0.0, 0.0);
239           outImageCorners[1] = cvPoint(255.0, 0.0);
240           outImageCorners[2] = cvPoint(255.0, 255.0);
241           outImageCorners[3] = cvPoint(0.0, 255.0);
242           innerMarkerCorners_1[0] = markerCorners[indice[0]][2];
243           innerMarkerCorners_1[1] = markerCorners[indice[1]][3];
244           innerMarkerCorners_1[2] = markerCorners[indice[2]][0];
245           innerMarkerCorners_1[3] = markerCorners[indice[3]][1];
246
247           Mat H1 = findHomography(innerMarkerCorners_1,outImageCorners,0);
248           warpPerspective(inputImage, image_1, H1, cv::Size(255, 255));
249
250           distance_mar_1 = find_marker_depth(marker_dis_1);
251           draw_marker_depth(outputImage, innerMarkerCorners_1, distance_mar_1);
252
253           for (int i=0;i<4;i++)
254           {
255             wall_distance_msg.marker_x[i]=innerMarkerCorners_1[i].x;
256             wall_distance_msg.marker_distance[i]=distance_mar_1[i];
257           }
258           wall_distance_msg.marker_exist = 2;
259
260         }
```

위 코드는 marker 를 통해 네 개의 점이 확정된 경우, 그 marker 의 좌표를 이용해 이미지를 256*256 의 정사각형 이미지로 변환하고, 각각의 marker 의 distance 와 픽셀 x 좌표를 계산하여 저장하는 코드이다.

2.2 CNN

CNN 은 readmarker 에서 publish 한 이미지를 subscribe 하여, 이 이미지를 convolutional neural network 를 통과시켜, 이 이미지가 개인지 고양이인지를 판단한 후, 그 결과를 cmd_sub 에 publish 한다. 이 과정을 통해 내가 현재 어떤 색의 공을 release 해야 하는지 판단한다. 이 network training 을 위해 12000 여 개의 개 이미지와 12000 여 개의 고양이 이미지를 사용하였다. Image 를 무선 통신을 통해 subscribe 하기 때문에 전달 속도가 늦어지는 경향을 보였다.

3 DQN

3.1 Map making

실제 환경과 비슷한 환경에서 learning 이 필요 했기 때문에 dqn simulator 에서 map making part 를 수정하게 되었다. 실제 시연에 사용하게 될 map size 의 비율을 그대로 반영하게 되었다. DQN 은 SLAM 을 마치고 작동하게 된다. 이에 따라 SLAM 이 끝난 이후 하드웨어가 존재할 수 있는 위치를 크게 4 가지로 분류하였다. 미로를 오른쪽으로 빠져나올 경우, 하드웨어가 오른쪽을 바라볼 경우와 왼쪽을 바라볼 경우, 미로를 왼쪽으로 빠져나와 오른쪽을 볼 경우와 왼쪽을 볼 경우, 총 4 가지의 상태로 분류하게 되었다. 실제 SLAM 을 진행할 때 미로를 왼쪽으로 나올지 오른쪽으로 나올지 모르기 때문에 두 상황을 분리하였다. 미로를 나온 이후 공이 놓여있는 방향을 바라볼지, 벽 쪽을 바라볼지 모르기 때문에 이 두가지 상황 역시 각각 분리하였다. 공 쪽을 바라 볼 경우, DQN 이 잘 learning 되지만 벽 쪽을 바라볼 경우, 공의 위치를 처음에 확인할 수 없으므로 learning 이 잘 되지않아 다른 상황으로 분류하여 learning 하게 되었다. 4 가지 상황으로 분류하였기에 랜덤 함수를 주어 각각 1/4 확률로 simulator 에 나타나게 하였다. 기존에 존재하였던 벽이 아예 없는 경우는 우리와 무관한 실험이므로 없애게 되었다. 차체가 원점에 존재하도록 map 을 제작해야 했기 때문에 4 가지 상황에 각각 다른 map 을 생성하게 되었지만 기본적인 방법은 모두 동일하다. 우선 기울기에 약간의 랜덤 한 값을 주어 직사각형 꼴의 map 을 생성하였고 필요하지 않는 부분은 부등식의 영역을 이용하여 제거하게 되었다. 하드웨어가 오른쪽을 바라볼 경우와 왼쪽을 바라볼 경우를 분리하기 위해서 중간에 있는 벽의 기울기를 각각 달리하여 제작하게 되었다. 공은 실제 시연과 동일하게 6 개만 생성되도록 하였다. 또한 실제 규칙과 동일하게 벽과 일정 거리 이상 떨어져 생성되게 하였고, 공이 깜빡이거나 움직이는 형상은 그대로 수용하게 되었다. 하드웨어는 항상 미로를 빠져나온 이후 위치에서 시작하도록 설정하게 되었다. 하드웨어의 크기는 좌우가 좁고 앞뒤가 길기 때문에 하드웨어의 사이즈를 인지하지 못하고 벽에 충돌하는 경우를 없애기 위해 4*3 의 큰 하드웨어를 생성하게 되었고, 공은 앞으로만 pick up 할 수 있도록 설정하였다.

3.2 Reward and Action

로봇의 가운데 배치된 파이프를 통해 공을 pick up 하기 때문에 정면으로 공으로 다가가는 것이 중요하다. 이에 따라 가운데 직진으로 공을 pick up 할 경우 reward 를 150 주었다. 중앙과 그 양 옆을 차등을 주어 reward 를 주었는데 camera 2 를 이용하여 중앙으로 정확히 들어오지 않아도 align 이 잘 됨을 확인하여 공이 직진으로 들어오기만 하면 동일하게 150 점을 주게 되었다. 양 옆에 가이드가 부실하였기 때문에 양 옆이나 뒤로 공이 들어온 것은 pick up 을 하지 않은 것으로 판별하게 되었다. 매 step 당 reward 는 -1 점을 주어 최대한 적은 step 으로 미션을 수행 할 수 있도록 하였고 벽에 충돌할 경우에는 -2 점을 주어 충돌을 하지 않도록 하였다. Learning 이후 하드웨어가 공을 모두 pick up 하지 않았음에도 제자리에서 회전 만 하여 미션을 종료하는 경우가 발생함을 확인 할 수 있었다. 이와 같은 상황을 방지하기 위해 remain ball 을 계산하는 함수를 만들게 되었다. 공이 무조건 6 개 존재하므로 먹은 개수를 빼서 remain ball 의 개수를 계산하게 되었고, 스텝이 종료된 이후 remain ball 의 개수와 150 을 곱해 reward 에서 빼게 되었다. 공을 6 개 모두 pick up 해야만 DQN 을 종료하고 다시 SLAM 에 진입하게 되므로 모든 공을 pick up 하는 것은 매우 중요한 사안이었기 때문에 큰 감점을 주게 되었다. Step 의 경우 100 step 이 너무 적다고 판단하여 200 까지 늘리게 되었다.

Action 의 경우에는 처음에 모든 action 을 다 주게 되었다. (상하좌우, 양쪽 대각선, 양쪽 회전) 하지만 하드웨어의 특성상 대각선으로 이동하는 경우 모터에 큰 압력이 가해져 이 action 을 최소화 하는 것이 좋다고 판단하여 빼게 되었다. 또한 너무 많은 action 이 있을 경우 learning 에 많은 시간이 필요했으며 잘 수렴하지 않는 결과를 볼 수 있었다. 이에 따라, 최종적으로는 앞으로 가는 action 과 양쪽 회전 총 3 가지 action 만을 이용하여 learning 하게 되었다.

3.3 convert_sensor_to_image

이 코드는 ball detect node 에서 공의 개수와 위치, LIDAR 를 통해 얻어진 obstacle 좌표, SLAM 의 진행 여부인 release_start 를 subscribe 받는다. 이 후, 공의 개수가 3 개 이상이거나, SLAM 에서 목표한 지점까지 진행했다는 정보를 전달받으면, tf_pub 가 true 로 변하고, 이 후 subscribe 한 공의 좌표와 obstacle 의 좌표를 이용하여 93*93 흑백 map 을 만들어낸다. 그 후 만들어진 map 을 rl_dqn_network 에 publish 하게 된다. 또한, 이 코드에서 현재 프로그램이 어느 단계에 있는지 알려 주는 action_tf/Int8 msg 를 cmd_sub 로 publish 하게 된다. Action_tf msg 는 0 으로 초기화되며, dqn 단계에

들어서는 조건인 공 3 개 이상이나 SLAM 에서 목표한 지점까지 이동했다고 판단했을 때 1 로 바뀌어 dqn 이 시작했다는 것을 알려 주고, 이 후 다시 공을 6 개 먹었다고 판단되면 2 로 바뀌어 dqn 의 종료를 알린다.

3.4 rl_dqn_network

이 코드는 conver_sensor_to_image code 에서 publish 한 map 을 미리 learning 시킨 network 를 통과시켜, 현재 상태에서 차체가 가야 하는 action 을 얻어낸 후, cmd_sub 에 publish 한다. 이 때 action/Int8 msg 를 통해 publish 하는데, 우리 팀의 경우 action 이 0 인 경우 직진, 1 인 경우 turn counterclockwise, 2 인 경우 turn clockwise 를 의미한다.

4. Subsystems and Data Integration

4.1 Data Integrate Package

4.1.1 Ball Collect Node

공의 suction 을 위해서 작동하는 node 이다. 이 노드는 webcam 2 에 파란 공이나 빨간 공이 감지 되었을 때 로봇이 취해야 할 행동을 정해준다. 이 노드에는 총 두개의 callback 함수가 있고, 9 개의 보조 함수들이 있다. 이 함수들을 이용하여 이 노드는 cmd_sub 노드에게 myRio 통신으로 전해줄 data 값들을 전달한다.

먼저 두 개의 callback 함수들에 대해 서술하겠다. Camera2_callback 함수는 webcam2 에서 detect 한 공의 개수와 위치를 받는다. 만약 webcam2 내에 두 개 이상의 공이 감지되었을 경우와 노이즈로 인한 오류를 없애기 위해 가장 가까이 위치한 공을 기준으로 판단을 내리도록 시스템을 설계했다. 따라서 이 노드는 지정된 전역 변수에 가장 가까운 위치의 빨간 공, 혹은 파란 공의 x 좌표와 z 좌표를 assign 한다.

exitCallback 함수는 이 노드의 프로세스를 종료시키는 함수이다. 필드 위의 모든 공을 집으면 이 노드의 역할은 전부 끝나므로 다른 노드들의 연산의 속도만 늦추게 된다. 이 함수는 ball_count 노드에서 보내주는 여태 집은 공의 개수 데이터를 지속적으로 받는다. 집은 공의 개수가 6 개 이상이 되면은 exit(0)함수를 통해 프로세스를 종료시킨다.

Callback 함수들의 제외한 9 개의 보조 함수들에 대해 간략히 서술하겠다. dataInit 함수는 global data 로 지정된 data 변수들을 초기화 하는 함수이다. 단, data20 과 data21 변수는 초기화하지 않는다. data20 과 data21 은 로봇의 빨간 공과 파란 공이 어디로 저장될지 정해주는 sorting stick 을 움직이는 data 이다. dataInit 함수는 매우 자주 사용되고, 그때마다 이 stick 의 위치가 초기화 되면은 공의 분류가 제대로 되지 않는 경우가 발생하기 때문에 초기화를 매번 시켜주지 않는다.

choose_pick 함수는 일단 카메라에 잡힌 가장 가까운 공의 색이 빨간 공일 경우와 파란 공의 경우로 if 문을 나눈다. 그 이후, 만약 공이 카메라와 0.25cm 보다 가까우면 sorting stick 을 움직이는 data20 과 data21 에 값을 부여한다. 공이 카메라에 잡히는 즉시 sorting stick 을 움직이지 않고 공과 충분히 가까운 경우에만 움직이는 이유는 다음과 같다. 만약 카메라에 빨간 공과 파란 공이 동시에 보이고 빨간 공이 더 가까운 경우를 상정하자. 설계한 시스템 대로 로봇이 움직인다면 로봇은 먼저 빨간 공을 집을 것이다. 이때, 빨간 공이 파이프 안으로 진입 한 후에 sorting stick 까지 도달하는 시간은

약 1 초이다. 문제는 이미 빨간 공은 카메라에 없고 파란 공만이 존재하므로 만약 sorting stick 을 바로 움직인다면 빨간 공이 지정된 storage 로 들어가기도 전에 sorting stick 이 움직이고 말 것이다. 이러한 일을 미연에 방지하기 위해 공이 충분히 가까워진 경우만 sorting stick 을 움직이다.

그 후 choose pick 함수는 공의 색에 따라서 pick_up_blue 혹은 pick_up_red 함수를 call 한다. 이 두 함수의 역할은 오직 집는 공의 색만 다르고 완전히 일치한다. 굳이 함수를 분리해 놓은 이유는 공의 색이 다르면 공의 위치 데이터를 저장해 놓은 변수의 이름이 다르기 때문에 함수 작성의 편의를 위해서이다. 두 함수는 공이 카메라 기준으로 오른쪽에 있으면 오른쪽으로, 왼쪽에 있으면 왼쪽을 한다. 그리고 공이 카메라 가운데에 있으면 직진한다.

turn_CCW, turn_CW, move_forward 함수는 그 이름에서 유추 할 수 있듯 로봇이 반시계, 시계 회전 운동, 직진 운동을 하게 만드는 노드이다. 각자의 로봇의 바퀴를 움직이는 data 번호는 00, 01, 04, 05 이다. 그리고 suction_check 함수를 통해 suction 을 작동시키는 data16 값을 지정한다. 이 함수는 카메라에서 공이 없어져도 suction 이 일정 시간 동안 켜지도록 만든 함수이다. 이 적절히 변환된 변수들 값을 publish 한다. 이 함수들은 전부 초반에 first_2 라는 글로벌 변수를 10 으로 초기화 하는데, 이 상세한 이유는 후술하겠다.

cmd_sub 는 여러 시스템의 명령을 동시 받는다. SLAM, DQN, ball collect, ball release 총 네 가지의 명령이 동시에 들어오고, 상황에 따라 어느 명령을 따를 것인지 골라야 한다. 따라서 각자의 노드는 cmd_sub 의 선택을 돕기 위해 상황에 따라서 어느 명령을 선택할 것이지 각자의 변수를 publish 한다. ball collect 노드는 message 에 로봇을 움직이기 위한 data 명령어 이외에 pick_start 변수를 보내준다. 이 변수는 평소에 0 이지만, webcam2 에서 공을 감지할 경우 1 로 지정되며, cmd_sub 는 우선적으로 ball_collect 노드에서 보내주는 명령을 따르도록 프로그래밍 되어 있다.

이런 식으로만 시스템을 작동하면 심각한 문제가 생긴다. Suction 으로 인해 공이 파이프에 들어간 순간부터 공은 카메라의 시야에서 사라진다. 그렇다면 로봇은 아직 공을 온전히 다 빨아들이기 전에 cmd_sub 는 pick_start 변수를 0 으로 받을 것이고, 아직 공이 파이프 위로 다 올라가기도 전에 명령 권한이 DQN 으로 넘어가게 되면서 석션이 꺼져버린다.

따라서 suction 을 시작한 이후에 공이 camera 시야에서 벗어나도 로봇 명령 주도권이 아직 ball collect 에게 몇 초간 있어야 한다. 이 문제를 해결하기 위해 pick_start 변수가 바로 0 으로 돌아가지 않고 몇 초간 2 가 되도록 설정했다. suction 을 제외한

로봇의 모든 움직임을 멈추고 몇 초간 pick start 변수로 2 를 publish 하는 함수가 바로 stop 함수이다. 이 함수는 turn_CCW, turn_CW, move_forward 함수에서 10 으로 초기화된 first_2 변수를 0.1 씩 감소시키고, first_2 변수가 양수일 때 pick_start 변수를 2 로 publish 한다. first_2 변수가 계속 0.1 씩 감소하여 0 이 되면 마지막으로 이 함수는 pick_start 로 0 을 publish 한다. 이로써 이 노드는 cmd_sub 에 대한 명령권을 잃는다.

이 노드의 전체적인 시스템을 요약은 다음과 같다. 평소에는 pick_start 변수를 0 으로 지정하여 cmd_sub 에 대한 명령권이 없다. webcam_2 가 공을 감지하면 pick_start 변수를 1 로 지정하여 cmd_sub 에서 명령권을 가져오는 동시 로봇에게 적당한 명령을 위한 data 를 publish 하다. 로봇이 공을 suction 한 이후, 시스템은 stop 함수를 통해 webcam2 에 공이 없음에도 불구하고 몇 초간 명령권을 유지한다. 그 이후 다시 pick_start 변수를 0 으로 설정하여 처음부터 다시 반복한다. 집은 공의 개수가 총 6 개가 되면 시스템 프로세스는 종료된다.

4.1.2 Ball Release Node

ball release 노드는 로봇이 처음부터 끝까지 항시 작동하는 노드이다. 이 노드의 명령권은 find_frontier 노드에서 보내주는 release_start 변수에 의해서 정해진다. 이 변수는 로봇이 모든 공을 집은 이후, SLAM 으로 다시 로봇이 시작한 위치로 돌아가면 1 로 설정된다. cmd_sub 는 release_start 명령이 1 인 경우 ball release node 의 명령에 따라 로봇을 움직인다.

ball release 노드는 두개의 callback 함수를 가지고 있다. catdogCallback 함수는 Jetson 에서 CNN 이 판별한 개, 고양이 정보를 수신한다. markerCallback 함수는 rgdb camera 로 통해 알아낸 image marker 의 위치와 거리 정보를 수신하다. 그리고 로봇이 취할 행동을 정해주는 step 함수 중 하나를 switch 문과 call 할 함수를 정하는 step 변수를 통해 call 한다. 만약 step 변수가 정수 N 이라면, markerCallback 함수는 stepN 함수를 call 한다.

release node 에는 step 함수가 step0 부터 step9 까지, 총 10 개가 존재한다. 이 함수들은 로봇이 각 단계에 따라 취해야 할 행동을 지정하며, 각 step 함수들은 지정된 조건이 만족 되었을 때 로봇이 다음 step 의 움직임을 따르도록 전역 변수 step 값을 조정해준다. 각 step 함수의 역할은 다음과 같다.

step0 함수는 로봇이 가장 먼저 취하는 행동을 명령을 통한다. 이 함수는 로봇이 readmarker 를 발견할 때까지 천천히 로봇을 회전하는 함수이다. 네 개의 마커가 동시에 카메라의 시야에 들어오면 step0 함수는 step 변수에 1 을 더한다.

step1 함수는 로봇과 마커의 거리가 0.5 미터 이하가 될 때 까지 천천히 이동하는 함수이다. step1 에서는 마커의 위치가 카메라의 중앙에 오도록 이동하며, 만약 마커가 카메라 중앙에서 벗어나면 회전을 통해 로봇의 각도를 조절한다. 마커의 거리가 지정된 거리보다 작아지면 step 변수에 1 을 더한다.

step2 함수는 마커와 로봇의 거리가 0.5 미터 보다 가까워졌을 때, 로봇의 위치를 최종적으로 정렬하는 노드이다. 로봇은 최대한 벽과 수직으로 바라도록 정렬한다. 좌측 마커와 우측 마커의 거리 차이가 5cm 이상이 될 때 로봇은 두 마커의 거리가 비슷해지도록 회전한다. 마커의 중심 위치가 중앙에서 크게 벗어나면 로봇은 좌우 움직임을 통해 마커가 로봇 중앙에 오도록 한다. 로봇이 마커를 수직으로 바라보고, 마커가 로봇의 중앙에 위치하면 로봇은 마커의 거리가 0.37 미터가 될 때까지 직진한다. 모든 조건이 만족되면 step 변수에 1 이 더해진다.

step3 함수는 CNN 에서 보내준 데이터를 이용하여 빨간 공, 혹은 파란 공을 방출한다.

step4 함수는 로봇을 open loop 으로 10cm 정도 후진하게 만든다. 이 step 이 존재하는 이유는 robot 이 너무 마커와 가깝게 있으면 storage pipe 가 마커를 가리는 경우가 있기 때문이다. 따라서 로봇을 약간 후진 시켜 줘서 마커의 시야를 다시 확보해준다.

step5 함수는 마커가 카메라의 중앙에 벗어나지 않도록 하면서 천천히 후진한다. 이 함수는 마커와 1.6 미터 이상 멀어질 때까지 후진하거나, 혹은 카메라 시야에서 마커가 일정 시간 동안 보이지 않을 때까지 후진한다. 위 두 조건 중 하나라도 만족 되면 step 을 increment 한다.

step6 부터 9 는 step0 부터 step3 까지 역할이 동일하다. 다만, 이미 공을 방출한 마커로 다시 접근하지 않기 위해서 이미 접근한 마커의 정보와 일치하지 않은 마커를 향해 다가간다.

이 노드에도 ball collect 노드와 마찬가지로 로봇의 움직임을 결정하는 turn_CW, turn_CCW, move_forward, stop 함수가 정의되어 있다. 추가적으로 정의된 move_left 와 move_right 함수는 로봇이 좌우 수평으로 움직일 수 있게 정의된 함수이다. 이 노드의 stop 함수는 로봇을 멈추는 것 외에는 다른 기능을 가지고 있지 않다.

4.2. Cmd_sub Package

4.2.1 Ball Count Node

ball count 노드는 로봇이 여태 집은 공의 개수를 세는 노드이다. 광센서와 LED 가 연결된 아두이노는 평소에는 '0'을 20hz 로 송출한다. 만약 파이프에 공이 지나가면서 LED 를 가리면 아두이노는 '1'을 송출한다. ball count 노드는 그 신호를 수신하여 여태 몇 개의 공을 집었는지 공의 개수를 센다.

문제는 공이 센서를 얼마만큼의 시간 동안 가릴 것인지는 알 수 없다. 따라서 1 신호가 한번만 들어올 수도 있고, 공이 천천히 내려가서 1 신호가 여러 번 들어올 수도 있다. 따라서 1 신호가 들어올 때마다 공의 개수를 늘이면 한 개의 공을 여러 개로 인식할 수 있다.

이 문제를 간단하게 해결하기 위해 이 노드에서 길이가 10 인 1 차원 array 를 두개 정의했다. 이 array 의 이름은 각각 ball_pass 와 prev_ball_pass 이다. 이 노드의 msgCallback 함수는 아두이노가 보내주는 신호를 이 array 에 순서대로 저장한다. 만약 array 가 꽉 차면 다시 처음으로 돌아가서 정보를 저장한다. 따라서 이 array 는 총 0.5 초 길이의 아두이노 신호를 저장하게 된다.

ball_pass 는 현재 들어온 데이터를 포함한 array 이고, prev_ball_pass 는 바로 전 loop 의 ball_pass 값을 저장하고 있다. 아두이노에서 신호 1 이 들어왔을 때 prev_ball_pass 에 신호 1 이 단 한 개라도 저장되어 있는지 확인한다. 만약 신호 1 이 저장되어 있으면 이 신호는 이미 count 가 된 공의 것이다. 만약 prev_ball_pass 에 신호가 0 밖에 없다면 이 신호는 새로 들어온 공의 신호이므로 공의 개수에 1 을 더해준다.

4.2.2 Cmd Sub Node

cmd sub 노드는 개의 노드가 보내주는 명령 신호를 선별하여 어느 신호를 따를 것인지 복합적으로 판단한다. 이 노드는 총 9 개의 메시지를 subscribe 하고, 총 9 개의 callback 함수가 존재한다. 각 함수의 역할을 간단히 서술하겠다.

msgCallback 함수는 SLAM 에서 최종 로봇의 이동 방향을 결정하는 move base 노드에서 송신하는 cmd_vel 을 처리한다. cmd_vel 을 data 로 변환 하는 과정에 대한 자세한 설명은 SLAM 파트에 서술되어 있다.

dqn_intCallback 함수는 DQN 에서 보내주는 조건 신호를 받아 전역 변수 DQN_ON 에 저장한다. 이 신호가 정확히 어떠한 의미를 가지는지는 DQN 파트에 서술되어 있다.

ball_numCallback 함수는 현재 RGDB 카메라에 보이는 공의 개수를 전역 변수 seen_ball 에 저장한다.

dqnCallback 함수는 dqn 이 보내는 명령을 해독한다. dqn 은 직진, 반시계 회전, 시계 회전, 총 세가지의 신호를 cmd_sub 에게 전송한다. 직진 신호는 0, 그리고 반시계와 시계 방향 회전 신호는 각각 1 과 2 이다. 이 함수는 이 신호를 dqn_switch 변수에 지정한 다음에 switch 문을 통해 로봇을 조정하는 data 값을 바꾼다. 단, seen_ball 변수가 0 일 경우(즉, 카메라에 보이는 공이 없을 경우), DQN 이 보내주는 신호는 무시하고 dqn_switch 를 1 로 바꾼다. 이 이유는 카메라에 공이 없을 경우 DQN 의 판단력이 매우 저하되기 때문에 이 경우에 로봇이 무조건 반시계 방향으로 회전하게 값을 변질시켰다.

countCallback 함수는 ball count 노드가 보내주는 공의 개수를 수신하여 변수 picked_ball 에 저장한다.

collectCallback 함수는 ball collect 노드가 보내는 명령을 수신한다. 이 노드가 작동하는 조건은 start_pickup 이 1 일때와 2 일때이다. 이에 대한 자세한 설명은 ball collect 노드에 서술되어 있다.

relstartCallback 함수는 find frontier 노드에서 release_start 변수를 수신한다. 이 변수에 대한 자세한 설명은 ball release 노드에 서술되어 있다.

releaseCallback 함수는 release_start 변수가 1 일대 release node 에서 송출한 값을 data 에 입력한다.

ballpicked 함수는 아두이노 신호를 수신한다. 아두이노는 공이 광센서를 가리고 있으면 1, 가리고 있지 않으면 0 을 송출한다. 이 신호가 0 이면 이 함수는 suc_shutdown 변수를 0, 그렇지 않을때는 1 로 설정한다. 이 함수와 변수가 존재 이유는 다음과 같다. 로봇의 석션 파이프가 연결된 위치는 공이 지나가는 파이프의 측면이다. 석션에 의해 빨아올려진 공은 중력에 의해 자연스럽게 storage 로 내려가야 한다. 하지만 석션의 힘에 의해 공이 파이프를 따라 내려가지 않고 석션 파이프에 연결된 곳에 붙어버리는 경우가 발생한다. 광센서는 석션 파이프 위치와 동일하게 배치되어 있었으므로 만약 공이 석션 파이프와 연결된 구멍에 붙어버리면 아두이노는 계속 신호 1 을 송출할 것이다. collectCallback 에서 석션을 키고 끄는 data[16]에 ball collect 가 보내는 신호와 suc_shutdown 변수를 곱해줘서 만약 공이 석션 파이프에 막힌 경우에는 석션 명령이 1 이 들어와도 0 이 되도록 하여 suction 을 껐다.

4.Hardware

1. 기본구조

기본적으로 전체적인 모양은 1 학기 때의 모델을 많이 참고하였습니다. 이번학기에는 Lidar 를 사용해야 해서 높이 제한이 약 50cm 가 있었습니다. 이에 맞추어 디자인을 수정하다 보니 지난 학기보다 더 밀집되고 무게중심도 많이 낮추었습니다. 부피를 많이 차지하는 배터리를 더 낮은 곳에 배치하여 성공적으로 높이를 낮추었고 최종 모델의 높이는 40cm 였습니다.

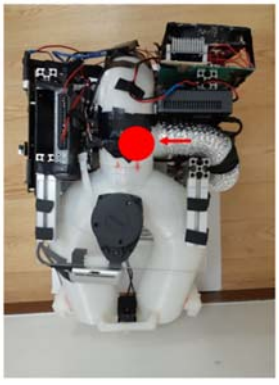
또한 지난 학기 기어가 꽉 맞물리지 않아 헛도는 현상을 해결하기 위해서 기어박스도 수정을 하였습니다.

전체적인 로봇의 무게가 10.8kg 으로 꽤나 무거웠고, 이에 따라 바퀴 모터가 받는 하중이 늘어나 과부하가 생기는 문제가 발생하였습니다. 바퀴의 과부하를 막고 하중을 분산하기 위해서 파이프 밑에 베어링을 부착하였습니다.

작년 석션조와 달리 파이프 배출구를 일자가 아닌 한 곳으로 모아는 형태로 디자인하여 공을 배출할 때 로봇제어에 여유를 둘 수 있도록 하였습니다.

2. 석션 & 파이프

우리가 사용한 석션 모터의 Air flow rate 가 28cfm 이었고 이는 공을 줍기에 충분한 값이었습니다. 파이프의 지름은 지난 학기와 동일 하게 continuity principle 과 Bernoulli equation 을 통해 구하였고 그 값은 40mm 이다. 지난학기와 달리 이번에는 공을 2 종류로 구분하여 주워야 했고, 우리는 2 종류의 공을 각각 보관 및 배출 할 수 있도록 파이프를 디자인 했습니다.



파이프 위에 Lidar 를 설치하기 위해서 석션 모터와 연결되는 부분은 측면에 배치하였습니다. 또한 석션이 되는 부분에 광센서를 부착하여 공이 석션이 끝났을 때, 공을 카운트하도록 디자인했습니다. 석션 중 공이 석션 입구를 막아 문제가 되기도 했습니다. 이를 해결하고자 석션 입구에 굴곡을 주기도 하였고, block 을 만들어 공이 파이프에 끼지 않도록 노력했습니다. 하지만 이와 같은 노력에도 개선되지 않았습니다. 다시

살펴보니 근본적인 원인은 파이프 내에 접합부에 남아있던 글루건이 공의 흐름을 방해하고 있었습니다. 그래서 파이프 내부의 글루건을 제거를 통해 석션의 문제는 해결하였습니다.

석션 이후의 파이프가 충분한 경사를 지녀야 공이 아래로 원활하게 굴러 저장 될 텐데, 이번의 경우 배터리와의 접촉 및 로봇 사이즈를 축소하다 보니 경사각이 크게 나오지 못했습니다. 이 때문에 석션 이후 석션 모터를 꺼야 공이 아래로 내려와 솔팅되는 문제가 있었습니다.

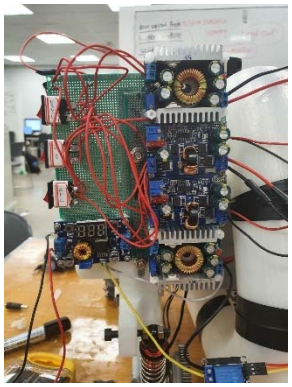
3. 솔팅시스템(sorting system)

우선 파이프 내부에서 공을 솔팅을 하기 위해 모터를 이용하기로 하였습니다. 모터에 막대를 달아 신호에 따라 한쪽으로 공이 저장되게끔 유도하기로 하였습니다. 솔팅 막대와 파이프 사이의 공간이 충분하지 못해 공이 지나가지 못하는 문제가 발생하였고 또한 막대의 내구성문제로 막대의 파손이 발생하기도 하였습니다. 이를 해결하고자 막대의 각도와 두께를 여러 번 수정하면서 진행했습니다. 그러나 결국 막대만으로는 100%문제를 해결하지 못하였고, 솔팅하는 파트의 파이프 직경을 늘려 공간을 충분히 확보함으로써 문제를 해결하였습니다.

4. 배출시스템(releasing system)

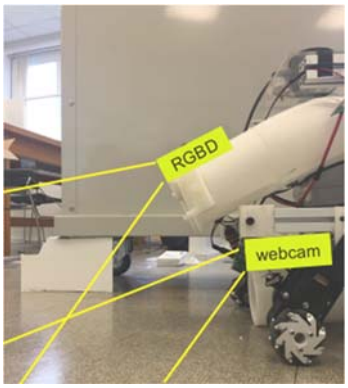
우리는 최대한 적은 수의 모터를 사용하기를 원했습니다. 그래서 두 개의 출구를 하나의 모터로 제어하기를 희망하였고, 그에 맞는 디자인을 생각하였습니다. 그래서 반원 모양의 뚜껑 덮개를 통해서 양쪽 파이프의 뚜껑을 동시에 막으면서, 각각 열 수 있도록 디자인하였습니다. 모터의 고정을 위해서 파이프 사이의 추가적인 지지대를 추가하였습니다.

5. 컨버터



처음에는 지난 학기의 스위치를 받아서 컨버터만 연결하였습니다. 이 후에 스위치를 새로 만들자는 의견이 있어 스위치를 달면서 스위치 하나로 컨버터 전부 ON/OFF 하도록 하였습니다. 이 후 테스트하면서 불필요한 전력소비도 많고, 누전의 우려가 있어 소프트웨어 팀의 요청에 따라 NUC, Jetson, 그리고 나머지 3 개의 파트로 나누어 스위치를 설치하였습니다. 나중에 열을 낮추고자 fan 을 설치하였는데, fan 의 전력 또한 컨버터를 통해 전달하였습니다.

6. 기타



카메라는 총 두 개로 RGBD 하나, webcam 하나를 설치하였습니다. RGBD 를 높은 곳에 설치하여 넓은 영역을 담당하고, webcam 은 파이프입구 부근에 달아 공이 석션 할 때 모션을 관찰하도록 하였습니다.

서스펜션은 지난 학기 때부터 잘 되지 않는다는 의견이 있었습니다. 이것을 해결하고자 서스펜션의 스프링 K 값을 낮추기로 하였고, 스프링의 일부분을 제거하였습니다. 이후 서스펜션이 잘 작동하지 않는 다는 느낌이 강하였습니다. 윤활유도 바르고 댐핑 오일도 추가하는 등의 노력을 하기도 하였습니다. 하지만 원하는 성능이 나오지 않아 구매처에 문의한 결과, 조립상에 문제가 발생했을 가능성을 제기해주셨고, 서스펜션을 재분해 및 조립을 하여 서스펜션을 고치는데 성공하였습니다. 샤프트와 축이 조립과정에서 확실히 고정되지 않았던 것이 문제가 되어 계속해서 마찰이 발생했던 것으로 생각됩니다.

창시구 C 조

감사합니다.