AUTO Shopping Cart for Senior

A-4

정승환 윤민식
장준하

TABLE OF CONTENTS

01 개요

02 목적 및 필요성

03 구현 시나리오

04 구현 결과

05 시연 영상

06 보완 사항

개요

노약자분들은 쇼핑을 어떻게 할까?

노약자가 원하는 물건을 사기 위해 쇼핑하는 것은 굉장히 어렵다

우선, 거동이 불편하신 분들이 굉장히 많기에 매장을 방문하는 것이 매우 어렵다

방문을 한 뒤에도 여러 물건을 구매하려면 많이 걸어야 할 것이다

인터넷을 사용하여 구매하는 것도 노약자분들 입장에서는 굉장히 어색하고 어렵다

목적 및 필요성

목적

노인들이 마트에서 쇼핑할 때, 이동의 편의성과 안정성을 제공하고,
물건 운반의 부담을 줄여주는 모빌리티 카트를 제작한다.
사용자의 필요에 따라 직접 조작과 자동 경로 이동이 가능하며
이를 통해 노인들의 쇼핑 부담을 줄이고, 만족감을 증폭시킨다.

목적 및 필요성

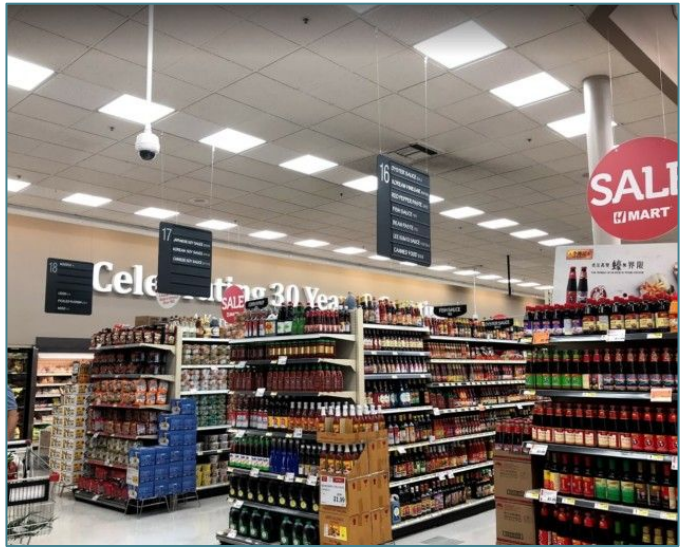
필요성

고령화가 심화되는 사회에서 신체 능력의 저하로 인한 불편함이 증가하고 있다.

특히 인터넷이 익숙하지 않은 노인분들은 생필품 구매조차 부담스러운 일로 여긴다.

따라서 이러한 어려움을 해소시키기 위해 쉽게 물건을 구매할 수 있는 환경을 만들어야한다.

구현 시나리오



카트

-> 매장 내부를 자유롭게 이동할 수 있는 크기

-> 카트를 제어할 수 있는 UI를 구성

-> 사용자가 앉을 공간과 물건을 담을 공간을 확보

-> 안정성을 위한 적절한 속도 제어

구현 시나리오

모바일 매니퓰레이터

-> 목적지에 안전하게 도달할 수 있는 경로 생성

-> 원하는 물건을 잡을 수 있는 매니퓰레이터의 동작

-> 주변과의 충돌을 방지할 수 있는 능력

-> 안정성을 위한 적절한 속도 제어

구현 시나리오

카트 컨트롤러

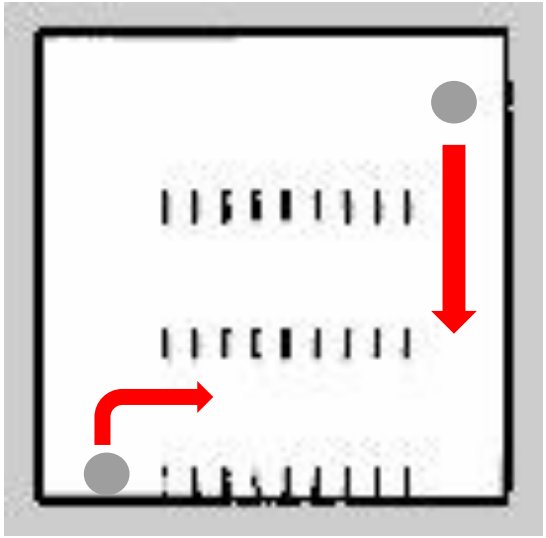
-> 현재 위치를 알 수 있는 매장 지도

-> 모바일 매니퓰레이터를 사용할 수 있는 물건 리스트

-> 카트 전방 사각 지대 방지를 위한 카트 전방 CAM

-> 카트를 수동 조작할 수 있는 버튼

구현 시나리오



시나리오

-> 카트를 수동 조작하며 쇼핑을 즐긴다.

-> 소비자가 원하는 물건을 선택하여 자동으로 이동한다.

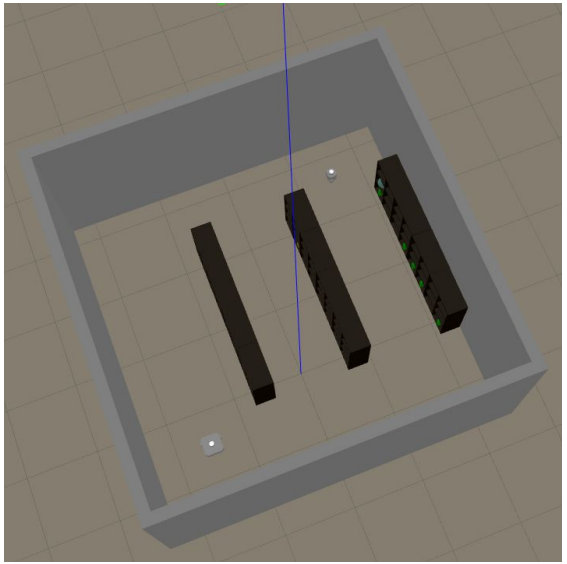
-> 매니퓰레이터도 물건의 위치로 자동으로 이동한다.

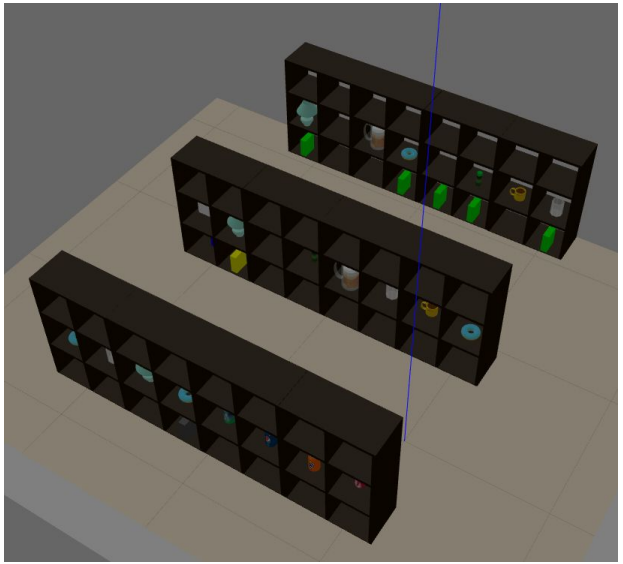
-> 매니퓰레이터가 물건을 잡아 카트에 담는다.

-> 매니퓰레이터가 원위치로 복귀한다.

구현 결과

World

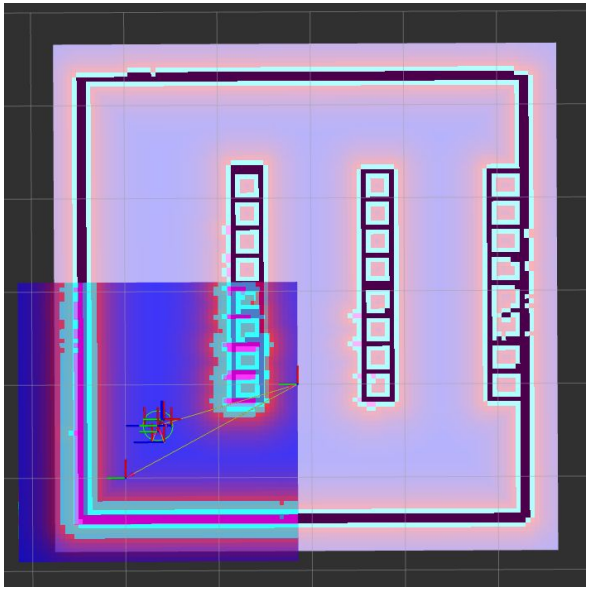




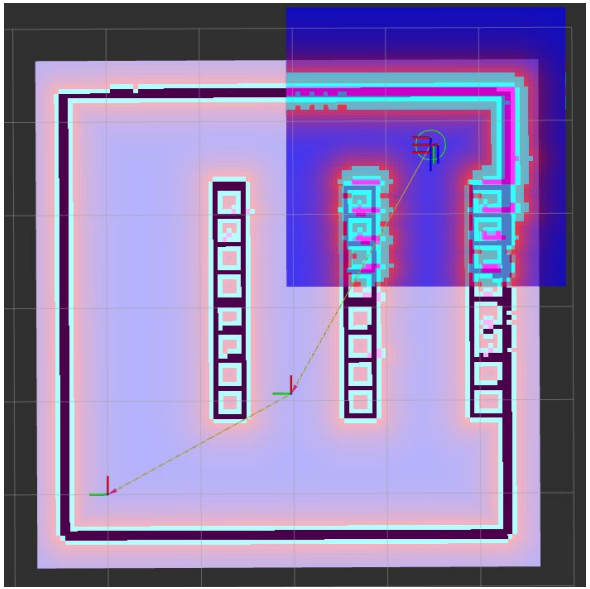
shopping mall

구현 결과

Multi robot navigation



cart nav



moma nav

구현 결과

Code

```
# Names and poses of the robots
robots = [
    {'name': 'cart', 'model': 'waffle', 'x_pose': '-0.44', 'y_pose': '1.5', 'z_pose': 0.01, Y': '0.0'},
    {'name': 'moma', 'model': 'burger', 'x_pose': '2.67', 'y_pose': '-1.5', 'z_pose': 0.01, Y': '1.57'},
    ]
```

```
turtlebot_state_publisher = Node(
    package='robot_state_publisher',
    namespace=namespace,
    executable='robot_state_publisher',
    output='screen',
    parameters=[{'use_sim_time': use_sim_time,
                 'publish_frequency': 10.0}],
    remappings=remappings,
    arguments=[os.path.join(turtlebot3_multi_robot, 'urdf', 'turtlebot3_' + robot['model'] + '.urdf')],
)

# Create spawn call
spawn_turtlebot3_burger = Node(
    package='gazebo_ros',
    executable='spawn_entity.py',
    arguments=[
        '-file', os.path.join(turtlebot3_multi_robot, 'models', 'turtlebot3_' + robot['model'], 'model.sdf'),
        '-entity', robot['name'],
        '-robot_namespace', namespace,
        '-x', robot['x_pose'], '-y', robot['y_pose'],
        '-z', '0.03', '-Y', robot['Y'],
        '-unpause',
    ],
    output='screen',
)
```

구현 결과

Code

```
init_poses = [
    {'x_pose': '0.56', 'y_pose': '-0.35', 'z_org': '0.0', 'w_org': '1.0000000'},
    {'x_pose': '3.74', 'y_pose': '-3.48', 'z_org': '0.7058193356173937', 'w_org': '0.7083918869302647'},
    ]
```

```
for robot, init_pose in zip(robots, init_poses):

    namespace = [ '/' + robot['name'] ]

    # Create a initial pose topic publish call
    message = '{header: {frame_id: map}, pose: {pose: {position: {x: ' + \
        init_pose['x_pose'] + ', y: ' + init_pose['y_pose'] + \
        ', z: 0.1}, orientation: {x: 0.0, y: 0.0, z: ' + init_pose['z_org'] + ', w: ' + init_pose['w_org'] + '}}, }}'
```

구현 결과

Topic list

```
/cart/amcl/transition_event
/cart/amcl_pose
/cart/behavior_server/transition_event
/cart/behavior_tree_log
/cart/bond
/cart/bt_navigator/transition_event
/cart/camera/camera_info
/cart/camera/depth/camera_info
/cart/camera/depth/image_raw
/cart/camera/image_raw
/cart/camera/points
/cart/clicked_point
/cart/cmd_vel
/cart/cmd_vel_nav
/cart/controller_server/transition_event
/cart/cost_cloud
/cart/downsampled_costmap
/cart/downsampled_costmap_updates
/cart/evaluation
/cart/global_costmap/costmap
/cart/global_costmap/costmap_raw
/cart/global_costmap/costmap_updates
/cart/global_costmap/footprint
/cart/global_costmap/global_costmap/transition_event
/cart/global_costmap/published_footprint
/cart/global_costmap/voxel_marked_cloud
/cart/goal_pose
/cart/imu
```

```
/cart/initialpose
/cart/joint_states
/cart/local_costmap/clearing_endpoints
/cart/local_costmap/costmap
/cart/local_costmap/costmap_raw
/cart/local_costmap/costmap_updates
/cart/local_costmap/footprint
/cart/local_costmap/local_costmap/transition_event
/cart/local_costmap/published_footprint
/cart/local_costmap/voxel_grid
/cart/local_costmap/voxel_marked_cloud
/cart/local_plan
/cart/marker
/cart/odom
/cart/particle_cloud
/cart/particlecloud
/cart/plan
/cart/plan_smoothed
/cart/planner_server/transition_event
/cart/received_global_plan
/cart/robot_description
/cart/scan
/cart/smoother_server/transition_event
/cart/speed_limit
/cart/tf
/cart/tf_static
/cart/transformed_global_plan
/cart/velocity_smoother/transition_event
/cart/waypoint_follower/transition_event
/cart/waypoints
```

```
/moma/amcl/transition_event
/moma/amcl_pose
/moma/behavior_server/transition_event
/moma/behavior_tree_log
/moma/bond
/moma/bt_navigator/transition_event
/moma/clicked_point
/moma/cmd_vel
/moma/cmd_vel_nav
/moma/controller_server/transition_event
/moma/cost_cloud
/moma/downsampled_costmap
/moma/downsampled_costmap_updates
/moma/evaluation
/moma/global_costmap/costmap
/moma/global_costmap/costmap_raw
/moma/global_costmap/costmap_updates
/moma/global_costmap/footprint
/moma/global_costmap/global_costmap/transition_event
/moma/global_costmap/published_footprint
/moma/global_costmap/voxel_marked_cloud
/moma/goal_pose
/moma/imu
```

```
/moma/local_costmap/costmap
/moma/local_costmap/costmap_raw
/moma/local_costmap/costmap_updates
/moma/local_costmap/footprint
/moma/local_costmap/local_costmap/transition_event
/moma/local_costmap/published_footprint
/moma/local_costmap/voxel_grid
/moma/local_costmap/voxel_marked_cloud
/moma/local_plan
/moma/marker
/moma/odom
/moma/particle_cloud
/moma/particlecloud
/moma/plan
/moma/plan_smoothed
/moma/planner_server/transition_event
/moma/received_global_plan
/moma/robot_description
/moma/scan
/moma/smoother_server/transition_event
/moma/speed_limit
/moma/tf
/moma/tf_static
/moma/transformed_global_plan
/moma/velocity_smoother/transition_event
/moma/waypoint_follower/transition_event
/moma/waypoints
```

구현 결과

UI

Flask

마트의 MAP에서 본인의 카트와
매니퓰레이터의 실시간 위치 표시

상품으로 이동할 수 있게
각각의 좌표를 저장한
리스트

전방 사각지대를 확인할
수
있는 카트 CAM

카트를 수동 조작할 수
있는
방향 버튼

구현 결과

UI

```
# 카트의 포지션
self.sub_pose_cart = self.create_subscription(PoseWithCovarianceStamped,'/cart/amcl_pose',self.cart_pose_callback,10)
# 모바일 매니퓰레이터 포지션
self.sub_pose_moma = self.create_subscription(PoseWithCovarianceStamped,'/moma/amcl_pose',self.moma_pose_callback,10)
# 캠에서 받아오는 이미지
self.sub_cam_img = self.create_subscription(Image,'/cart/camera/image_raw',self.cam_callback,10)

# 수동 조작을 위한 publisher
self.pub_vel = self.create_publisher(Twist,'/cart/cmd_vel',10)
# 자동 조작을 위한 publisher
self.pub_pose = self.create_publisher(PoseStamped,'/cart/goal_pose',10)
# 자동 조작을 위한 publisher
self.pub_moma_pose = self.create_publisher(PoseStamped,'/moma/goal_pose',10)
```

카트와 매니퓰레이터 제어를 위한 각종 **topic** 처리

구현 결과

UI

```
def update_img(self):
    per_pixel = 0.05
    global base_image
    img = base_image.copy()

    cart_x = -(self.cart_position['y'] - 0.39)
    cart_y = -(self.cart_position['x'] - 4.24)
    cart_map_x = round(cart_x/per_pixel) + 8
    cart_map_y = round(cart_y/per_pixel) + 4

    moma_x = -(self.mo_ma_position['y'] - 0.39)
    moma_y = -(self.mo_ma_position['x'] - 4.24)
    moma_map_x = round(moma_x/per_pixel) + 8
    moma_map_y = round(moma_y/per_pixel) + 4

    cv2.rectangle(img, (cart_map_x,cart_map_y),(cart_map_x+1,cart_map_y+1),(0,0,255,255),-1)
    cv2.rectangle(img, (moma_map_x,moma_map_y),(moma_map_x+1,moma_map_y+1),(255,0,0,255),-1)
```

카트, 모바일 매니퓰레이터의 좌표 변화가 있을 때마다 호출되는 함수
-> 실시간 정보를 **MAP**에 업데이트하는 기능을 구현

구현 결과

UI

```
def send_cmd_vel(self, msg):
    if msg == 'up':
        self.linear_x += 0.01
    elif msg == 'down':
        self.linear_x -= 0.01
    elif msg == 'left':
        self.angular_z += 0.01
    elif msg == 'right':
        self.angular_z -= 0.01
    elif msg == 'stop':
        self.linear_x = 0.0
        self.angular_z = 0.0

    cmd_vel = Twist()
    cmd_vel.linear.x = self.linear_x
    cmd_vel.angular.z = self.angular_z
    self.pub_vel.publish(cmd_vel)
```

카트의 수동 조작을 위한 함수
-> **teleop**와 유사한 기능을 합니다

```
def send_goal(self,x,y):
    msg = PoseStamped()
    msg.pose.position.x = float(x)
    msg.pose.position.y = float(y)
    msg.header.frame_id='map'
    self.pub_pose.publish(msg)

    msg_moma = PoseStamped()
    msg_moma.pose.position.x = float(x + 0.5)
    msg_moma.pose.position.y = float(y)
    msg_moma.header.frame_id = 'map'
    self.pub_moma_pose.publish(msg_moma)
```

자동 동작을 위한 함수
-> 카트를 특정 상품의 위치로 보내고, 그 근처로
모.마를 함께 보낸다.

구현 결과

UI

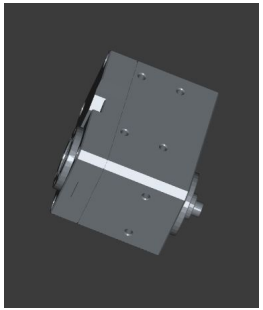
```python
def cam_callback(self,msg):
    cv_image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
    _, jpeg = cv2.imencode('.jpg', cv_image)
    self.latest_frame = jpeg.tobytes()
```

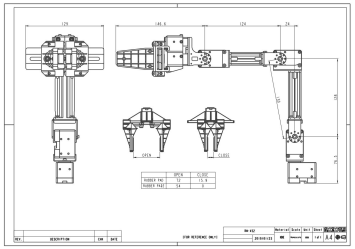
```python
@app.route('/cam_feed')
def cam_feed():
    def generate_frames():
        while True:
            if ros_node.latest_frame:
                yield (b'--frame\r\n'
                       b'Content-Type: image/jpeg\r\n\r\n' + ros_node.latest_frame + b'\r\n')
    return Response(generate_frames(),mimetype='multipart/x-mixed-replace; boundary=frame')
```

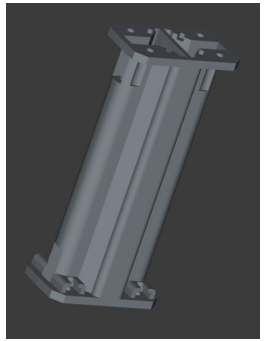
캠에서 받아오는 실시간 이미지를 처리하는 함수
-> http 통신을 사용해 웹 페이지로 영상 정보를 전송

구현 결과

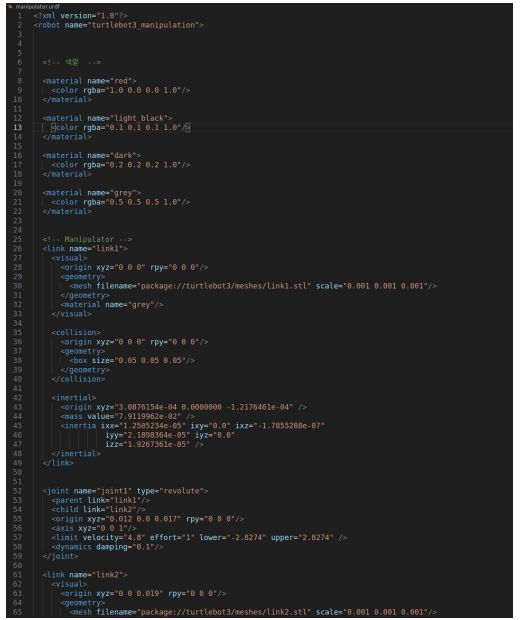
manipulator







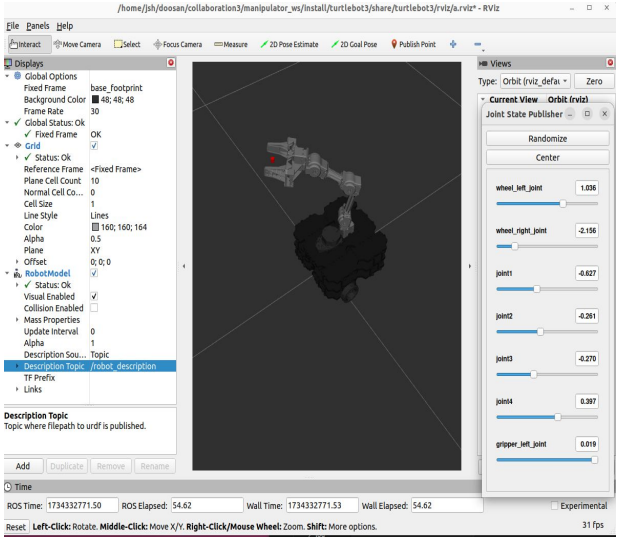
open source -> CAD



URDF

구현 결과

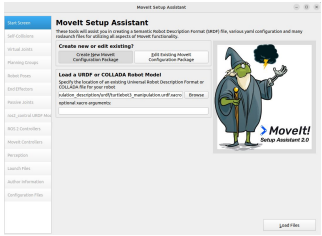
manipulator

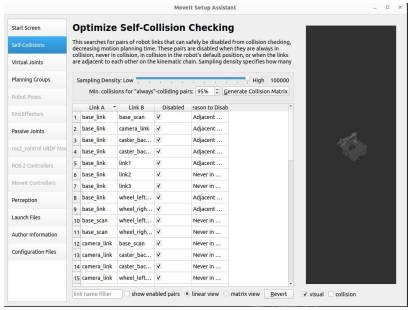


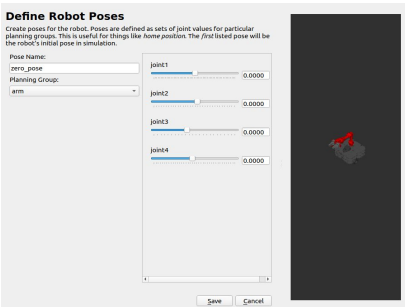
URDF 결과

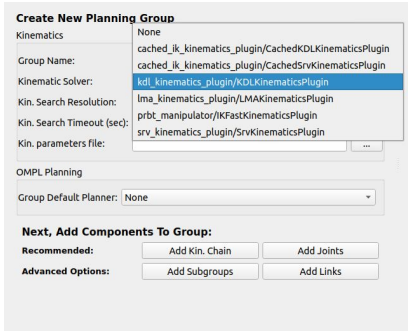
구현 결과

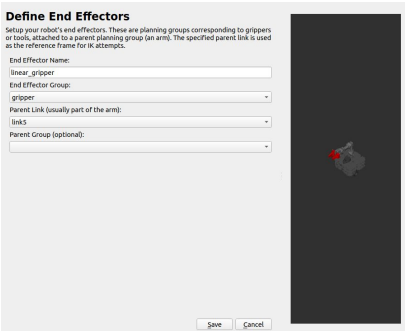
manipulator









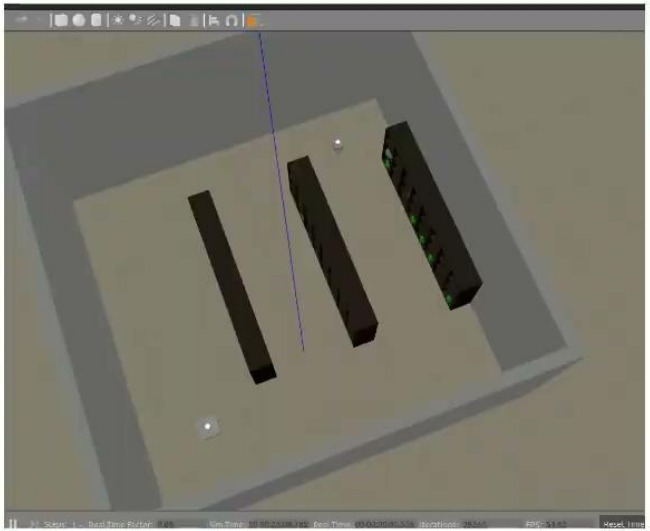


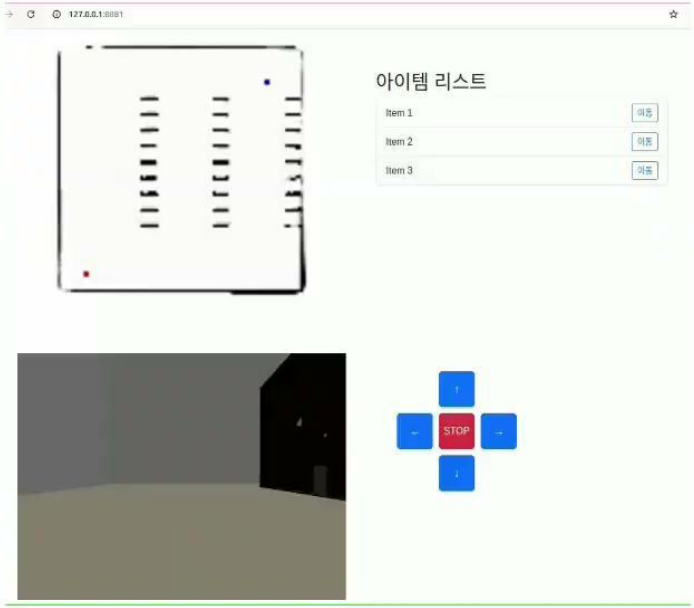
MoveIt

- 로봇 manipulator의 경로 계획, 조작
- 로봇의 조인트 각도를 위한 역기구학이 존재함

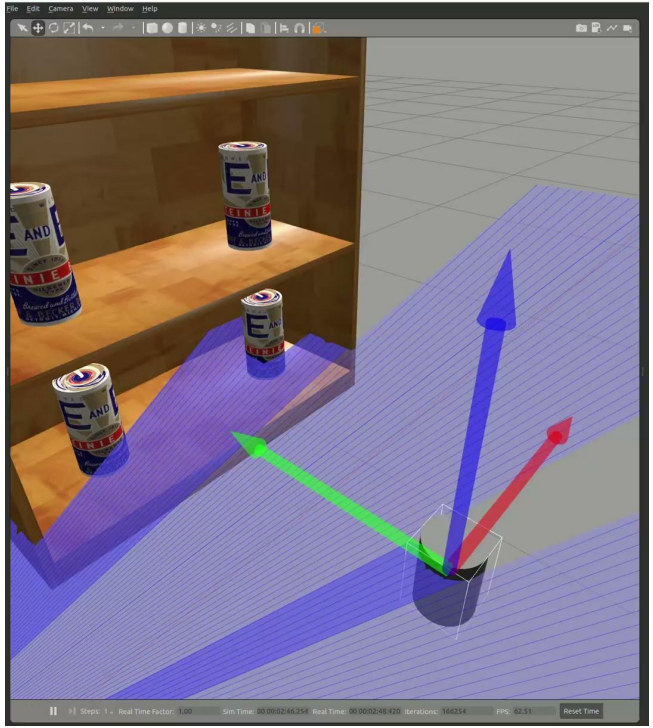
⇒ Path planning

시연 영상





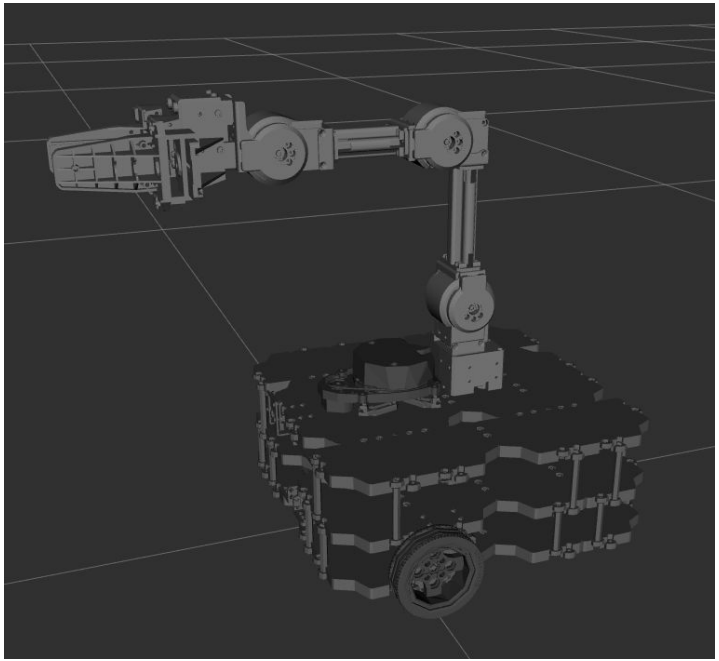
시연 영상

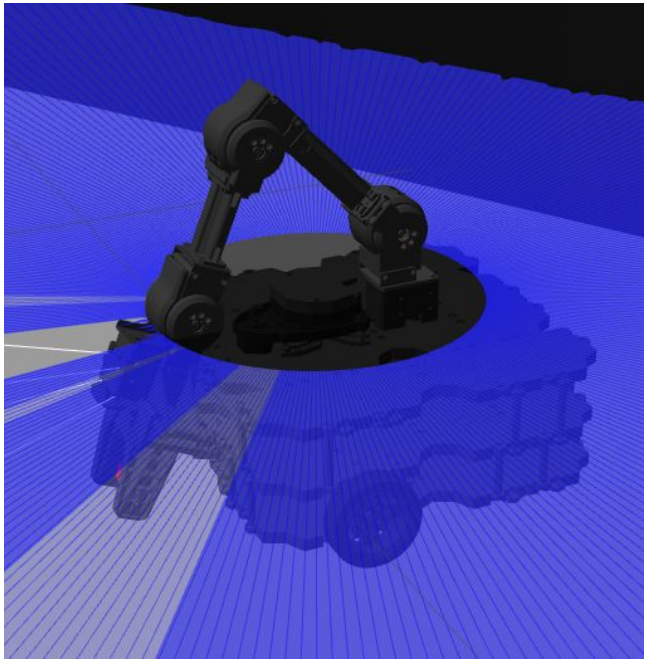

File Edit Camera View Window Help
Steps: 1
Real Time Factor: 1.00
Reset Time

시행착오

- Multi robot nav
  - 하나의 rviz창에 2개의 로봇을 띄우려다 실패
  - Multi robot nav를 구현하는데 namespace구분에 어려움을 겪음
- UI
  - Flask가 단일 스레드로 실행되기 때문에 영상 정보를 받아올 때,
    다른 영상과 정보와 충돌이 일어나는 현상이 있었다.
- Manipulator
  - Gazebo상에 moma를 올리는데에 시간을 너무 낭비하였음
  - MoveIt을 가지고 제어를 시도함
    1. 버전 전환으로 인한 데이터 부족(구버전 -> 신버전)
    2. 오류 발생(joint 파라미터 오류)
  - 가상 시뮬레이션 문제
    시뮬레이션에 사용된 물체의 질량 정보와 매니퓰레이터의 충돌 범위가 충분히 정의 x
    관성(inertial) 값이 부정확하여 정상적으로 작동하지 않았음

시행착오





보완 사항

- Multi robot nav
  - Navigation params tuning
  - Trajectory상에 다른로봇 등장 시 로봇이 어쩔줄 몰라하는 현상
  - waffle모델이 navigation중 경로를 이탈하는 현상 발생
  - turtlebot -> cart & moma
- UI
  - 미적 감각의 부족으로 UI 디자인을 개선하지 못해 소비자의 시각을 만족시키지 못했다.
  - '로봇 -> 서버 -> 웹 페이지' 로 이어지는 통신 경로에 전송되는 영상 정보가 매끄럽지 못하였다. 영상이 매우 버벅거리는 현상이 발생(통신 과정 최적화가 필요)
- Manipulator
  - MoveIt을 이용해서 매니퓰레이터를 능숙하게 통제할 수 있어야 한다.
  - Pick and place 적용

QnA