Planning Framework Seminar

2021. 01. 25

Part 2-2. Planning 프레임워크

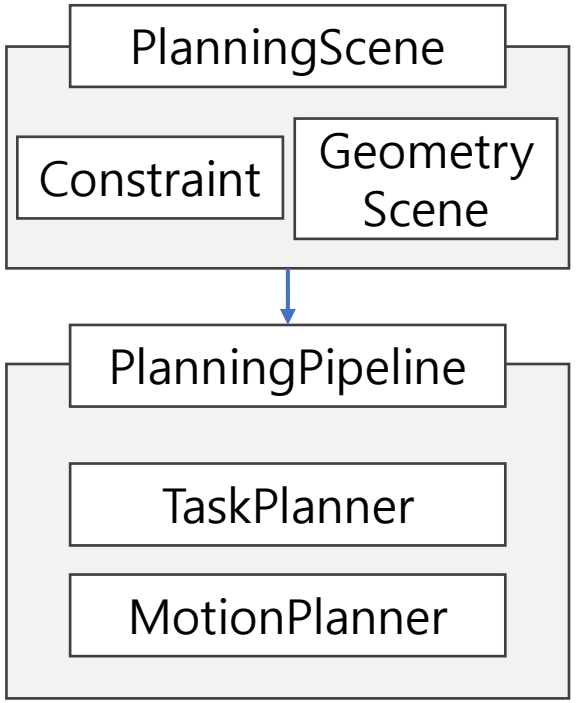
정재봉

Planning 프레임워크 사용 개요

1. 인식을 통한 GeometryScene 구성 (로봇의 초기 상태, 장애물의 위치, target의 위치, sweeping의 way point의 위치 등, planning을 진행하고자 하는 world의 geometry를 입력하는 단계.)

2. PlanningScene 구성 (GeometryScene이 world를 표현한 것일 뿐이라면 표현한 world에 실제 planning에 필요한 정보를 부여하는 단계. Object가 binding 될 수 있는 위치, 수행해야하는 task의 종류 등)

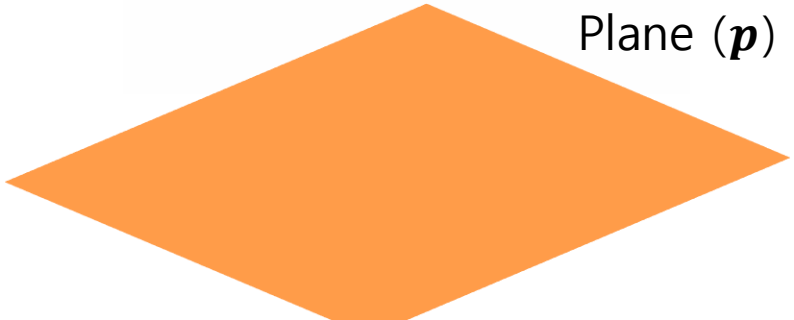
3. Planning (실제 planning 단계)



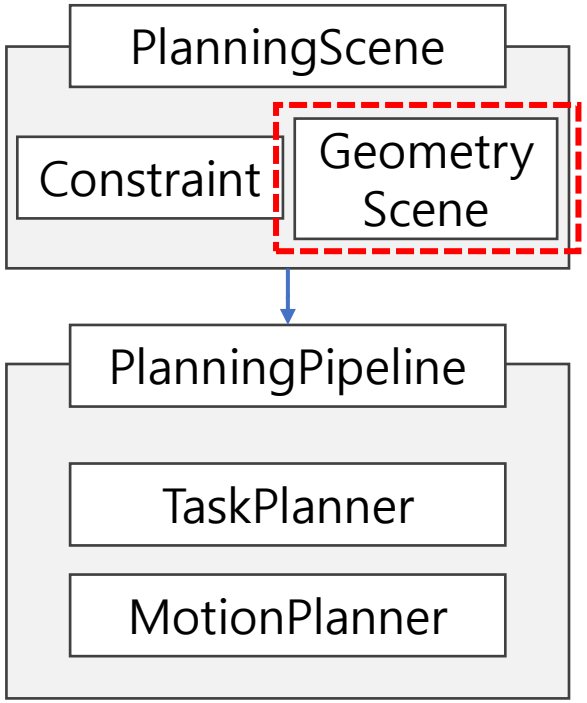
Planning 프레임워크 사용 개요

Gripper ($\boldsymbol{g}$)

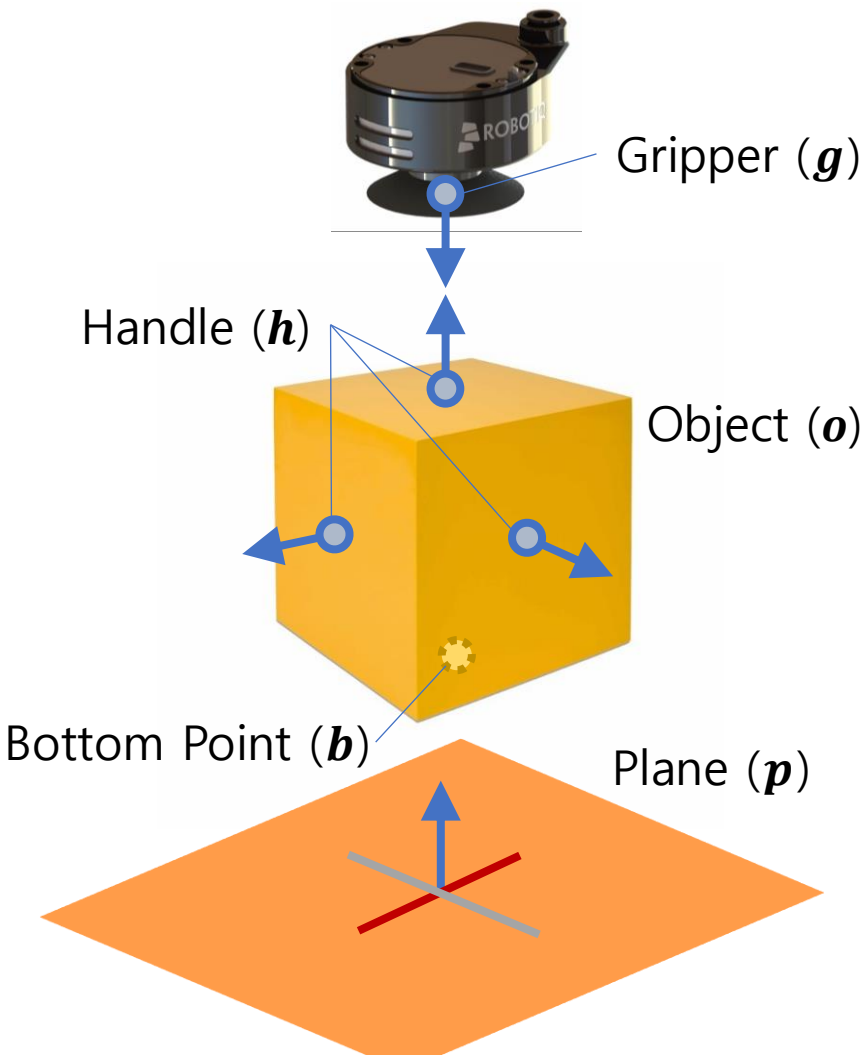
Object ($\boldsymbol{o}$)

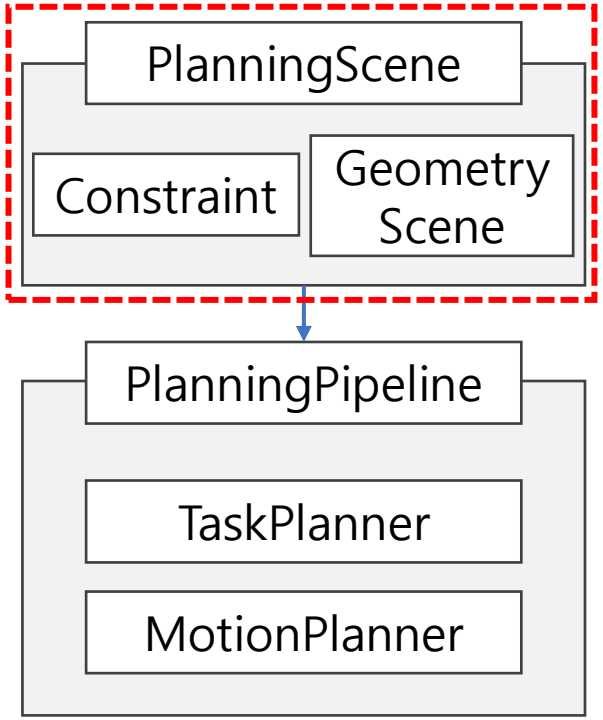


Plane ($\boldsymbol{p}$)



Planning 프레임워크 사용 개요

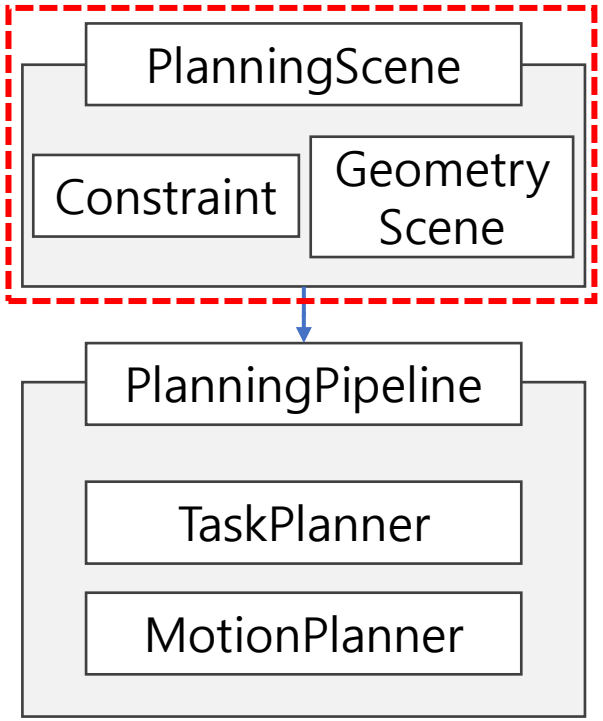




* constraint: 동작을 계획함에 있어서 지켜져야 하는 조건

e.g.) gripper로 object를 잡기 위해서는 gripper의 잡는 면(g)이 object의 잡을 수 있는 면(h)의 좌표축과 일치 혹은 근처에 있어야 한다.

PlanningScene 구성



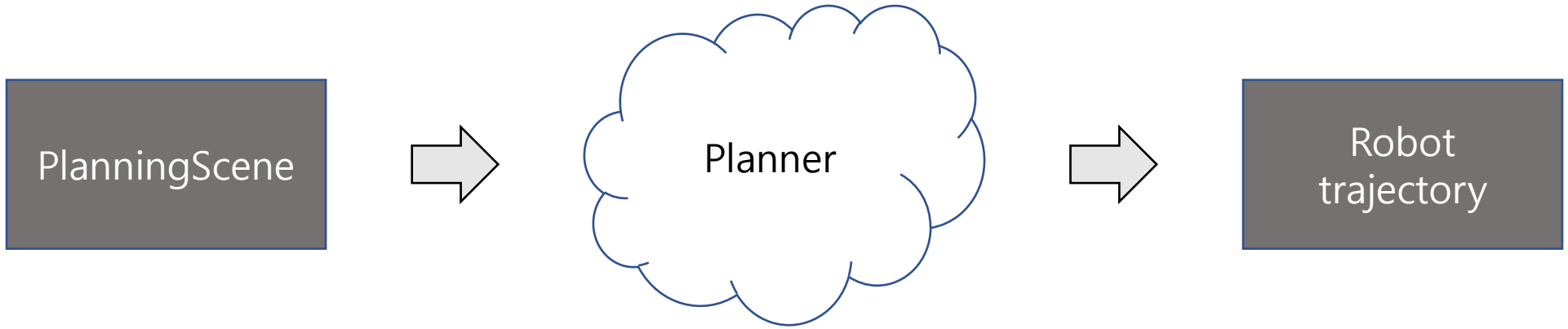
어떻게 계획을 만들 것인가 ?

로봇 입장에서의 계획이란 결국 궤적(robot configuration의 sequence) 이다.

궤적 생성을 위해서는 end effector의 initial position, orientation과 final position, orientation이 필요

→ rnb-planning 프레임워크는 Planner의 input으로 PlanningScene을 사용한다.
→ 즉, PlanningScene에는 planner가 robot의 initial state와 final state를 결정할 수 있게 해주는 정보가 담겨있어야 함

→ 이러한 정보를 표현하기 위해 **binding**개념을 도입.



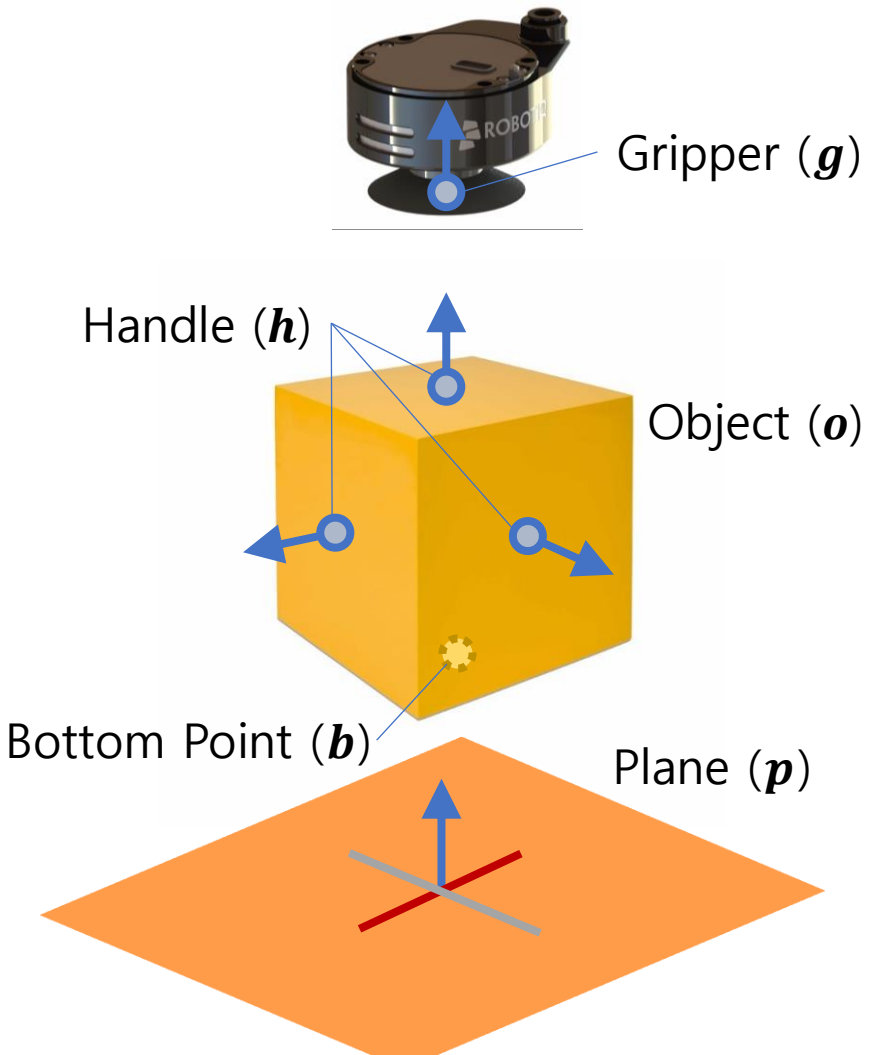
Binding

planning scene상의 어떤 object가 묶여 있는 상태.

e.g.)
Object가 바닥에 놓여있는 상황: p와 b가 binding
Gripper로 object를 집은 상황: g와 h가 binding.

* Binder: object(subject)가 bind될 수 있는 planning scene상의 대상(gripper, floor, ...). 코드 상에서 actor로 명칭 될 때도 있음.
우측 그림에서는 g와 p가 binder에 해당.

* Action point: object의 geometry 상에서 binder와 bind 되는 점. 코드상에서 handle로 명칭 될 때도 있음.
우측 그림에서는 h가 action point에 해당.

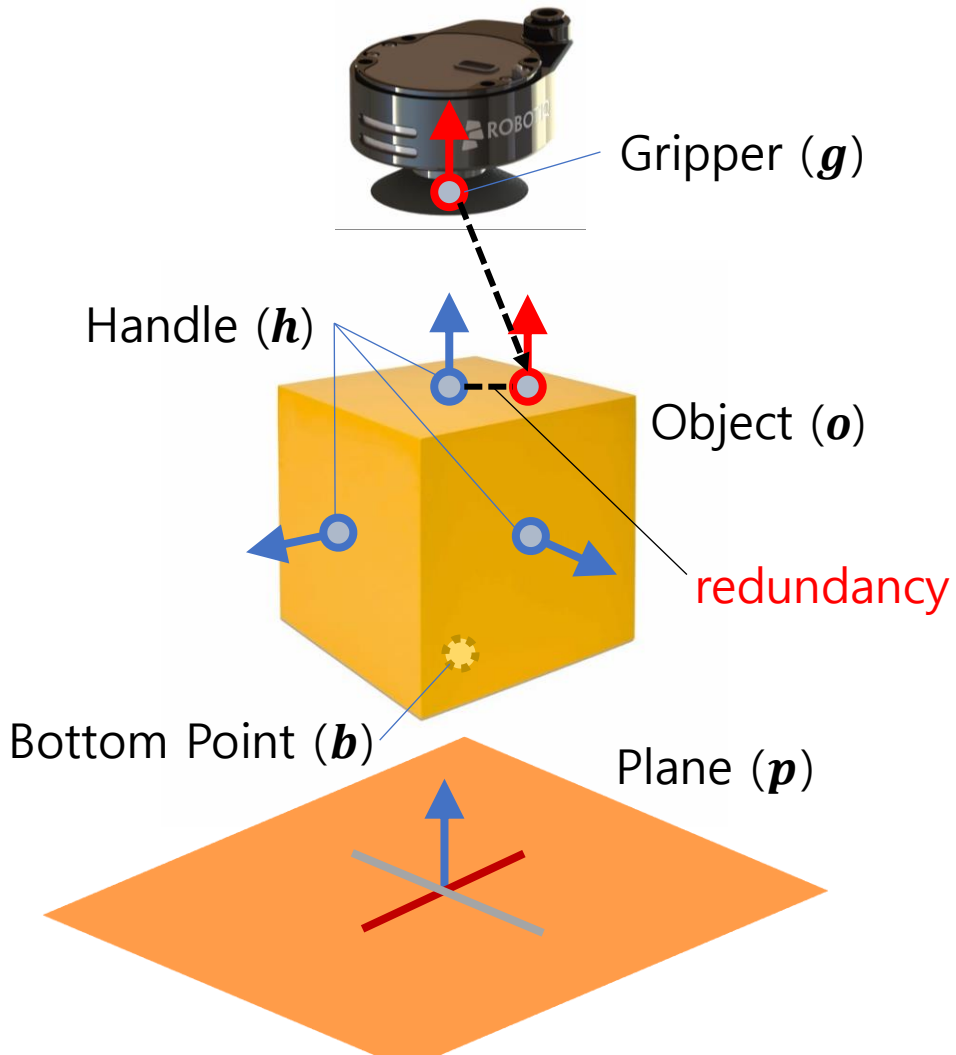


Binding을 이용한 계획 생성

현재 binding 상태와 다음 binding 상태가 주어지면 binder와 action point의 좌표축을 일치 시키는 방법으로 end effector의 initial position, orientation과 final position, orientation을 결정할 수 있다.

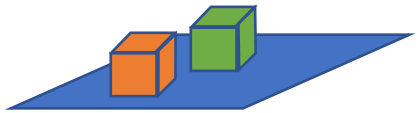
e.g.) gripper로 object를 잡으라는 명령을 내린 경우:

- Gripper의 좌표축을 handle의 좌표축과 일치시켜야 함
- 정확히 면의 가운데를 집을 필요는 없으므로 **handle 주변**의 임의의 position을 end effector의 final position으로 두고 동작을 계획(redundancy)

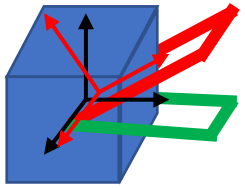


Redundancy

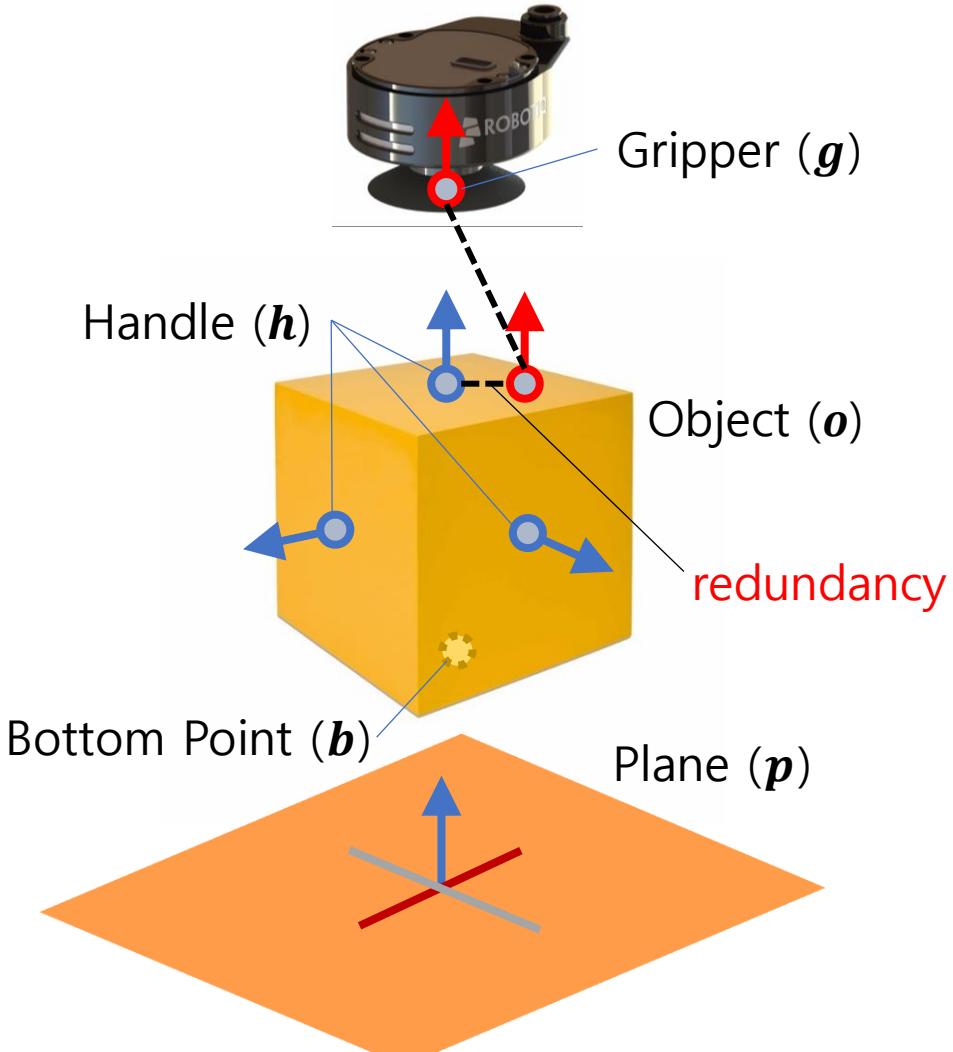
Binding이 이루어질 때 발생하는 여유 자유도.



Position redundancy



Orientation redundancy



Binding(BindingTransform)

Binding 상에서의 transformation 정보를 담고 있는 information holder 객체.

Action Point, Action Point가 붙어 있는 geometry, Binder, Binder가 붙어있는 geometry, geometry가 연결되어 있는 robot link등의 정보를 numpy array로 만든 SE(3)형태로 저장하고 있음.

State 클래스의 속성으로 BindingTransform이 들어가 있고, 이를 from_state에서 to_state로 가는 동작을 계획 할 때 활용하게 됨.

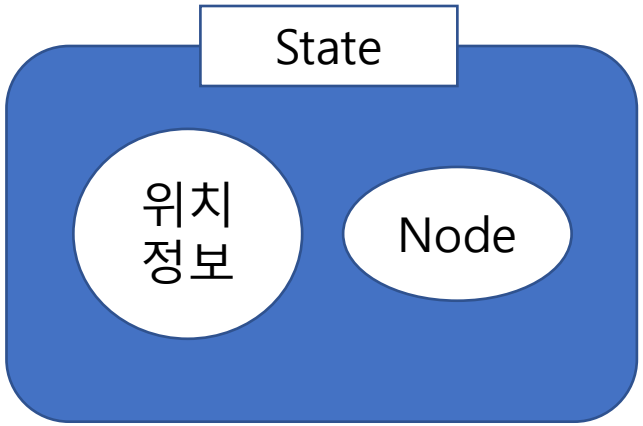
→ MoveitPlanner 클래스의 plan_algorithm 메소드를 보면 BindingTranform 객체에 있는 정보와 configuration Q 정보를 이용하여 motion plan(from_state부터 to_state까지의 로봇 configuration의 sequence)을 찾는 것을 확인할 수 있음.

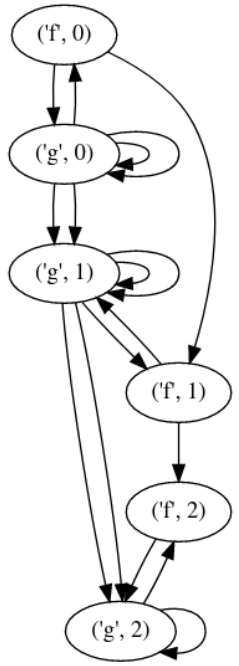
Node

PlanningScene의 상태를 tuple로 나타낸 것.

Motion plan에 관여하는 함수들의 입력인자로 사용된다.

각 원소는 PlanningScene 상에 있는 object가 binding되어 있는 binder에 해당.
→ Node의 원소의 개수는 PlanningScene 상의 object의 개수와 같다.





Planner가 찾아낸 (floor, 0) 상태 부터 가능한 node의 연결들.
* Node의 두 번째 원소인 숫자는 SweepTask의 상태를 표한 것으로, 추후 자료에서 설명 예정.

Code

1. Create binder

```
pscene = PlanningScene(gscene, combined_robot=crob)
```

```
# create PlacePlane on geometry "floor" and "goal"
# when point is not set, the entire upper surface of the geometry becomes valid binding area.
# when point is set, the specific point becomes the only valid binding point.
pscene.create_binder(bname="floor", gname="floor", _type=PlacePlane, point=None)
pscene.create_binder(bname="goal", gname="goal", _type=PlacePlane, point=(0,0,0.005))
```

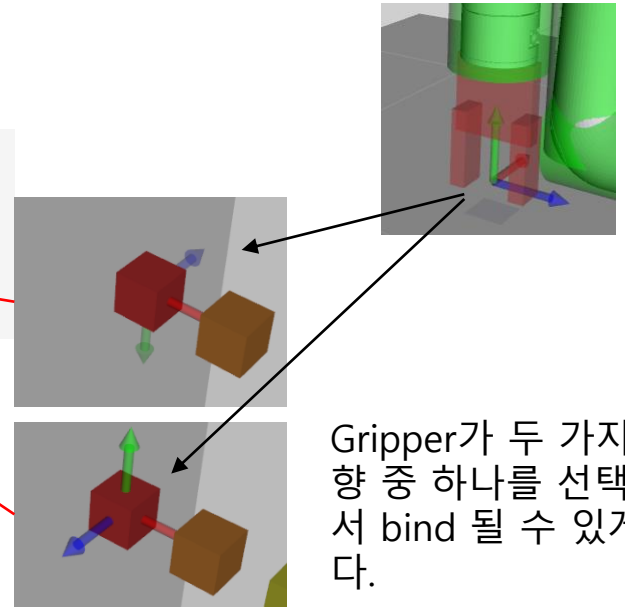
floor geometry의 모든 부분에 binding가능

goal geometry의 (0,0,0.005) 위치에만 binding 가능

2. Create subject

```
## create box object with grasping points along positive & negative y-direction and placement point in the bottom face
box_obj = pscene.create_subject(oname="box1", gname="box1", _type=CustomObject,
                                action_points_dict = {
                                    "handle1": Grasp2Point("handle1", box1, [0,0,0], [-np.pi/2,0,0]),
                                    "handle2": Grasp2Point("handle2", box1, [0,0,0], [np.pi/2,0,0]),
                                    "bottom": PlacePoint("bottom", box1, [0,0,-0.026], [0,0,0])})
```

Action point의 클래스 별로 redundancy 등 성질이 각각 정의 되어 있다.



Gripper가 두 가지 방향 중 하나를 선택해서 bind 될 수 있게 한다.

Code

3. Initialize state

```
initial_state = pscene.initialize_state(crob.home_pose)
print(initial_state.node)
```

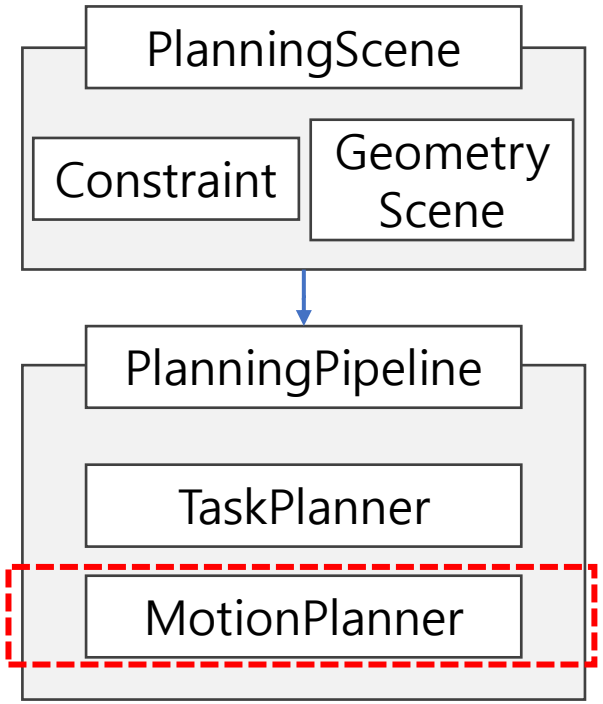
```
('floor', 0)
```

← Binder와 action point들 사이의 transformation 정보를 통해 best binding을 찾아서 initial state로 설정한다.

State: Planning scene의 상태를 나타내는 클래스

- binding_state: planning scene상의 object들의 binding 상태를 나타내는 dictionary ({'subject name' : BindingTransform})
- state_param: binding 외 정보 (e.g. sweep task의 각 way point가 경유 되었는지에 대한 Boolean)

Motion Planning



Motion Planning

1. Motion planning을 위한 플래너 객체 생성
   - 상용 motion planner인 Moveit planner를 사용.
   - Moveit planner를 사용하기 위한 interface 클래스인 MoveitPlanner(pscene)를 생성한다.

2. 현재 상태 및 목표 상태 정의
   - 현재 PlanningScene의 상태(object의 현재 binding 정보 등)를 정의한다.
   - 현재 상태에 대한 정보는 State 클래스를 활용한다
   - 목표하는 상태를 node 형식으로 표현한다.

3. 현재 가용한 목표 binding 정의
   - 현재 binding 상태와 목표 binding 상태에 대한 정보를 바탕으로 다음 State의 binding이 될 수 있는 후보들을 수집
   - get_available_binding_dict(from_**state**, to_**node**)를 활용한다.
   - 시작 상태는 scene의 object들의 binding 및 정확한 위치까지 모두 알고 있으므로 State 클래스인 from_state로 주어진다.
   - 목표 상태의 경우는 모든 정보를 정해주는 것이 아니고 object가 bind 되어야 하는 geometry만 정해주는 것이므로 node 형식으로 정해준다.
   - 목표 상태인 **to_node를 달성할 수 있는 binding은 여러 개가 존재할 수 있으므로 이를 수집하는 것**.

Motion Planning

4. Binding의 파라미터 정의
   - 다음 상태를 정확히 정의하는 작업.
   - 3.에서 수집한 후보 binding 중 하나를 임의로 선택하고 해당 binding안에서 물체의 위치까지 임의로 sample 한다 (Binding이 정해 지면 해당 binding에 소속된 action point가 허용하는 redundancy 내에서 임의의 위치를 가져도 되기 때문에 임의로 한점을 sample해서 정해 주어야 한다).
   - sample_leaf_state(from_state, available_binding_dict, to_node) 함수를 이용한다.
   - available_binding_dict는 get_available_binding_dict 함수의 반환 결과이다.

5. 플래닝 수행
   - 시작 상태와 목표 상태가 모두 정확하게 정해졌으므로 1. 에서 생성한 motion planner를 이용하여 motion plan을 진행한다.
   - 4.에서 to_state를 임의로 sample 했기 때문에 planning이 실패할 수 있다.
   - 실패할 경우 3.~5. 과정을 motion planning이 가능한 to_state가 sample 될 때까지 반복한다.
   - Motion planner 객체의 plan_transition(from_**state,** to_**state**) 함수가 motion planning을 수행한다.

Code

from_state와 to_state가 정해졌을 때의 동작 계획(Task planning X)

```
initial_state = pscene.initialize_state(crob.home_pose)
print(initial_state.node)
```

```
('floor', 0)
```

```
mplan = MoveitPlanner(pscene, motion_filters=[GraspChecker(pscene)])
```

← 상용 motion planner 사용(Moveit Planner)

```
from_state = initial_state.copy(pscene)
to_node = ("gripper", 0)
for _ in range(10):
    available_binding_dict = pscene.get_available_binding_dict(from_state, to_node)
    to_state = pscene.sample_leaf_state(from_state, available_binding_dict, to_node)
    with gtimer.block("plan"):
        Traj, LastQ, error, success, binding_list = mplan.plan_transition(from_state, to_state)
    if success:
        break
if success:
    pscene.set_object_state(from_state)
    gscene.show_motion(Traj, period=0.01)
    pick_state = pscene.rebind_all(binding_list, LastQ)
else:
    print("Solution not found. Please try again, try to find error if it keeps failing.")
    raise(RuntimeError("Motion plan failure"))
```

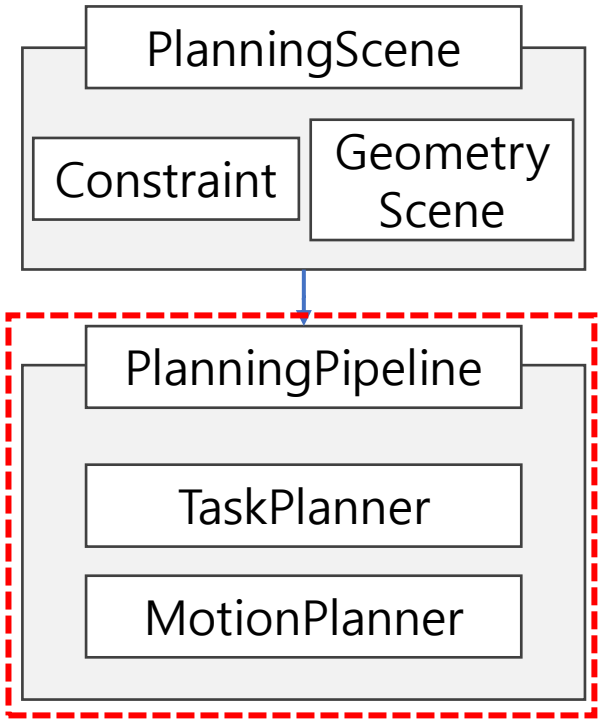
to_state와 from_state가 차이가 있는지 확인→차이가 있는 경우 다음 binding으로 가능한 조합을 모두 찾아서 dictionary 형태로 반환 {'box1' : [('handle1', 'grip0'), ('handle2', 'grip0')] }

← avaiable_binding_dict에서 random하게 to_state를 선정

← Sample한 to_state로의 motion을 plan

← 성공시 motion 실행.

Task Planning



Task Planning

Pipeline 객체 생성.

```
ppline = PlanningPipeline(pscene)
ppline.set_motion_planner(MoveitPlanner(pscene))
```

구성했던 planning scene 입력, motion planner로 Moveit Planner 설정.

```
from pkg.planning.task.rrt import TaskRRT
ppline.set_task_planner(TaskRRT(pscene))
```

Task planner로 RRT 방식 탐색 알고리즘 설정(Random-MMP)

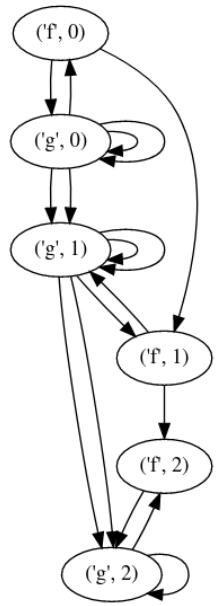
```
ppline.search(initial_state, goal_nodes=[("goal", 2)], verbose=True,
              display=False, dt_vis=0.01, timeout_loop=10, max_solution_count=1,
              timeout=0.5, timeout_constrained=2, multiprocess=False)
schedules = ppline.tplan.find_schedules()
schedules_sorted = ppline.tplan.sort_schedule(schedules)
ppline.play_schedule(ppline.tplan.idxSchedule2SnodeScedule(schedules_sorted[0]), period=0.01)
```

도달한 state가 goal이 될 때까지 schedule 탐색

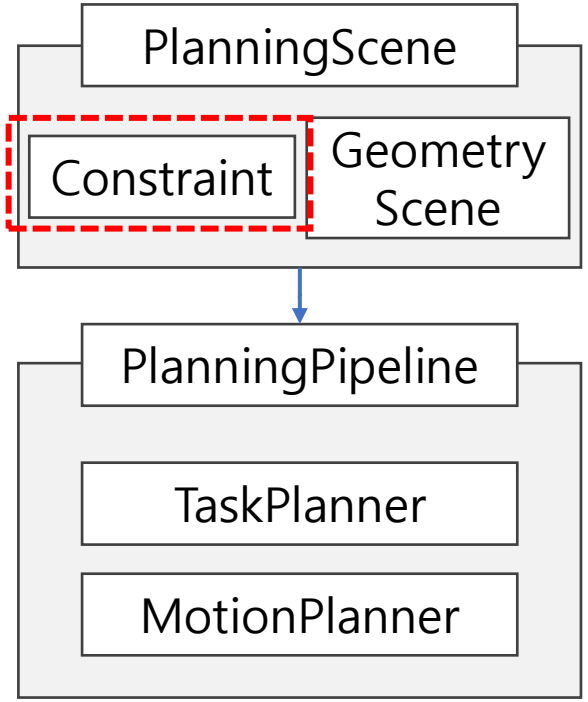
Goal state까지 가는 schedule 탐색

Schedule: 한 state에서 다른 state로 옮겨가는 동작
e.g.) ('floor', 0) -> ('gripper', 0)

Goal state에 도달하기 전까지 생성한 tree →



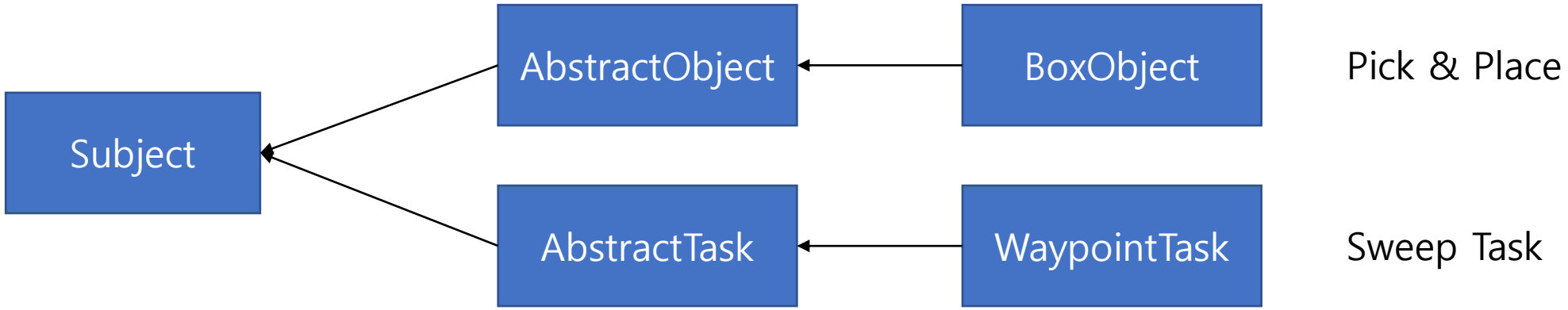
Expanding Planner



WaypointTask

두개의 way point가 주어졌을 때, 첫 번째 way point에서 두 번째 way point로 end effector와 평면의 접촉을 유지하며 이동하는 task.

Sweep task의 경우 pick & place task와는 다르게 task의 처음과 끝 상태 외에도 중간 과정도 고려 되어야 하기 때문에 task 자체가 하나의 클래스로 구현 된다.



PlanningScene상에서 해결 되어야 할 task는 Subject 단위로 관리된다(node의 원소 개수 == Subject의 수)

```
initial_state = pscene.initialize_state(crob.home_pose)
print(initial_state.node)
```

```
('floor', 0)
```

Pick & Place task의 대상이 되는 BoxObject 한 개와
WaypointTask 한 개가 PlanningScene 상에 존재

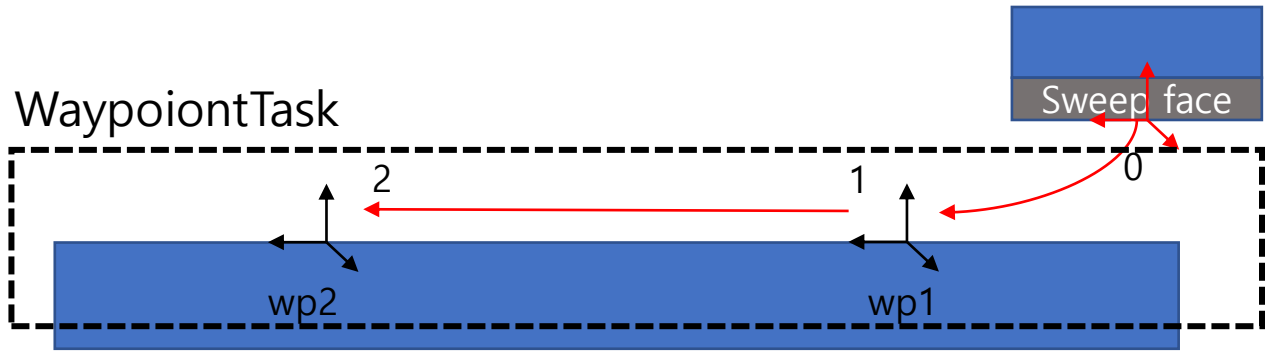
WaypointTask

Sweep하는 면을 binder로 정의하고 sweep 되는 면의 way point를 action point로 정의하여 동작을 계획한다.

WaypointTask의 경우 현재 상태까지 경유를 마친 way point의 수를 state_param 변수에 저장한다.

WaypointTask의 경우 way point가 정해지기 때문에 initial binding 상태와 final binding 상태가 이미 정해져 있기 때문에 현재 상태까지 경우된 way point의 수만 알고 있으면 목표 binding 상태를 알 수 있다.

→ Node에 WaypointTask의 상태를 표기할 때 현재까지 경유 된 way point로 표기 한다.



Code

release/4.1.ConstrainedMotionTasks.ipynb 참조

1. SweepTask

- Plane constraint만을 가지는 WayPointTask
- Actor를 plane에 접촉한다는 조건만을 만족 시키면서 두 way point를 지나는 동작을 계획

```
sweep = pscene.create_subject(oname="sweep", gname="floor", _type=SweepTask,
                              action_points_dict = {"wp1": SweepPoint("wp1", wp1, [0,0,0.005], [0,0,0]),
                                                    "wp2": SweepPoint("wp2", wp2, [0,0,0.005], [0,0,0])})
pscene.create_binder(bname="brush_face", gname="brush_face", _type=SweepTool, point=(0,0,-0.015), rpy=(0,0,0))
```

Orientation redundancy 있음

Plane Constraint

- Plane constraint는 SweepTask 클래스의 make_constraints 메소드에서 필요한 MotionConstraint 객체를 반환하도록 해서 설정 할 수 있다.(self.geometry = floor, fix_surface = True, fix_normal = True)

```
def make_constraints(self, binding_from, binding_to, tol=None):
    if binding_from is not None and binding_from.actor_name == binding_to.actor_name:
        return [MotionConstraint([self.geometry], True, True, tol=tol if tol is not None else self.tol)]
    else:
        return []
```

Code

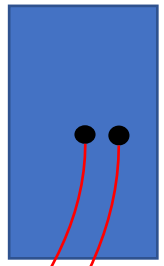
release/4.1.ConstrainedMotionTasks.ipynb 참조

2. SweepLineTask

- Plane constraint와 line constraint를 가지는 way point task
- Actor를 plane에 접촉한다는 조건만을 만족 시키면서 두 way point를 지나는 동작

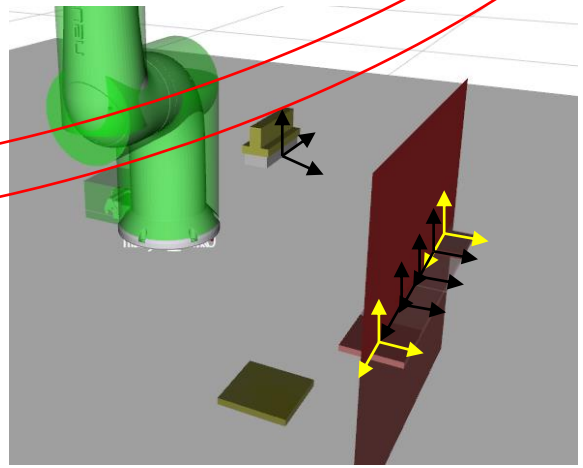
```
sweep = pscene.create_subject(oname="sweep", gname="floor", _type=SweepLineTask,
                              action_points_dict = {"wp1": SweepFrame("wp1", wp1, [0,0,0.005], [0,0,0]),
                                                    "wp2": SweepFrame("wp2", wp2, [0,0,0.005], [0,0,0])})
pscene.create_binder(bname="brush_face", gname="brush_face", _type=SweepFramer, point=(0,0,-0.015), rpy=(0,0,0))
```

Orientation redundancy 없음



- Line constraint 역시 make_constraint 메소드를 통해 설정할 수 있다.

```
def make_constraints(self, binding_from, binding_to, tol=None):
    if binding_from is not None and binding_from.actor_name == binding_to.actor_name:
        tol = tol if tol is not None else self.tol
        if self.fix_direction:
            return [MotionConstraint([self.geometry], True, True, tol=tol),
                    MotionConstraint([self.geometry_vertical], True, False, tol=tol),
                    MotionConstraint([self.geometry_vertical], True, False,
                                     T_tool_offset=SE3(np.identity(3), self.center_dir*tol*10), tol=tol)]
        else:
            return [MotionConstraint([self.geometry], True, True, tol=tol),
                    MotionConstraint([self.geometry_vertical], True, False, tol=tol)]
    else:
        return []
```



Geometry vertical 시각화

Code

release/4.1.ConstrainedMotionTasks.ipynb 참조

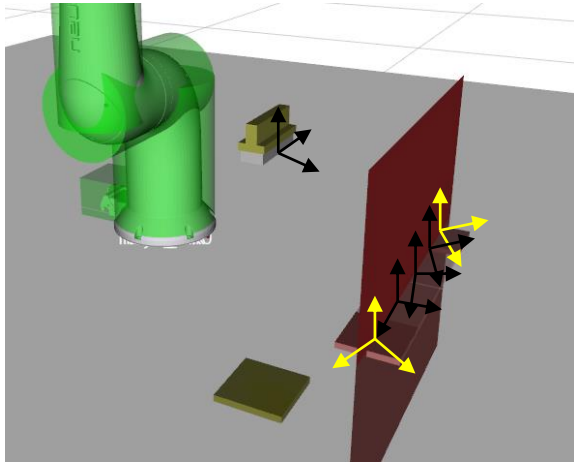
2. SweepLineTask without orientation constraint

- SweepLineTask는 두 way point를 지나는 동작을 계획할 때 orientation도 동일하게 유지.
- SweepLineTask의 fix_direction을 False로 설정하여 orientation constraint를 해제할 수 있다.

```
sweep = pscene.create_subject(oname="sweep", gname="floor", _type=SweepLineTask,
                              action_points_dict = {"wp1": SweepPoint("wp1", wp1, [0,0,0.005], [0,0,0]),
                                                    "wp2": SweepPoint("wp2", wp2, [0,0,0.005], [0,0,0])})
sweep.fix_direction = False # disable direction constraint
```

- fix_direction이 False인 경우, line_constraint가 하나뿐이기 때문에 orientation의 제약이 없어지게 된다.

```
def make_constraints(self, binding_from, binding_to, tol=None):
    if binding_from is not None and binding_from.actor_name == binding_to.actor_name:
        tol = tol if tol is not None else self.tol
        if self.fix_direction:
            return [MotionConstraint([self.geometry], True, True, tol=tol),
                    MotionConstraint([self.geometry_vertical], True, False, tol=tol),
                    MotionConstraint([self.geometry_vertical], True, False,
                                     T_tool_offset=SE3(np.identity(3), self.center_dir*tol*10), tol=tol)]
        else:
            return [MotionConstraint([self.geometry], True, True, tol=tol),
                    MotionConstraint([self.geometry_vertical], True, False, tol=tol)]
    else:
        return []
```



Geometry vertical 시각화

SweepTask(WayPointTask)

계획 목표: scene에서 보이는 brush를 집어서 way point 사이를 sweep하고 brush를 goal에 둔다.

```
initial_state = pscene.initialize_state(crob.home_pose)
print(initial_state.node)
```

```
('floor', 0)
```

Node에 binding된 geometry이름 대신 도달했던 way point의 수가 저장됨.

get_available_bindings 함수에서 가능한 다음 action_point를 찾을 때도 way point의 순서대로 다음 action_point 선정.

<Plan 결과>

```
('floor', 0)->('grip0', 0)
('grip0', 0)->('grip0', 1)
('grip0', 1)->('grip0', 2)
('grip0', 2)->('goal', 2)
('goal', 2)->('goal', 2)
```

Brush가 floor에서 gripper에 binding (brush는 gripper의 좌표축과 같이 움직이게 된다)

Brush face(아랫면)가 way point를 지나는 동작 계획(이때 brush가 gripper와 binding된 상태 이므로 gripper와 brush가 같이 동작을 수행한다.)

Brush를 goal과 binding.(brush가 더 이상 gripper와 binding 되어있지 않으므로 brush는 gripper와 함께 움직이지 않는다)

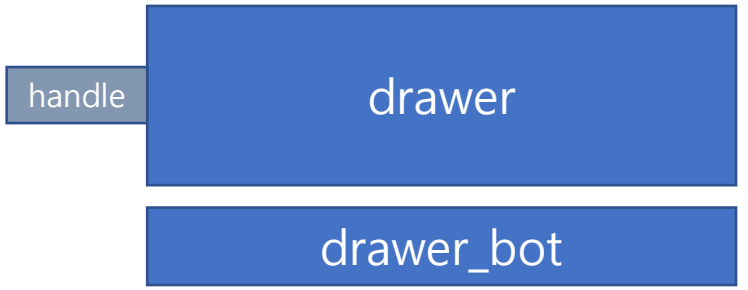
Drawer Task

앞서 설명한 내용을 바탕으로 서랍을 당겨 열었다가 밀어서 다시 닫는 task를 구현.

```
drawer_bot = gscene.create_safe(GEOTYPE.BOX, botname, link_name=link_name,
                                center=center, rpy=rpy,
                                dims=(dims[0] + wall_thickness, dims[1] + wall_thickness, wall_thickness),
                                fixed=True, collision=True, color=color)
```

```
handle = gscene.create_safe(GEOTYPE.BOX, "{}_handle".format(dname), link_name=link_name,
                            center=(-dims[0] / 2 + wall_thickness / 2 - handle_size[0] / 2, 0, 0), dims=handle_size,
                            fixed=False, collision=True, color=color, parent=drawer.name)
```

```
drawer = gscene.create_safe(GEOTYPE.BOX, dname, link_name=link_name,
                            center=center, dims=np.subtract(dims, wall_thickness + clearance),
                            fixed=False, collision=True, color=color)
```



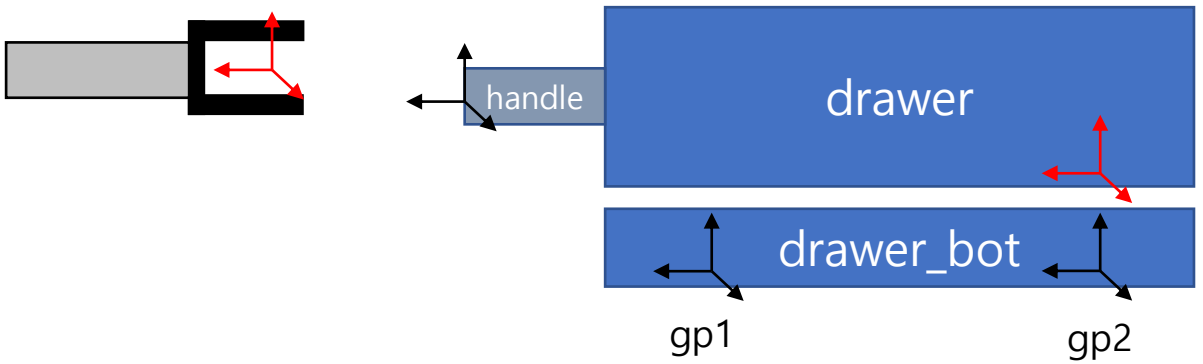
Drawer Task

Planning scene에 object들 추가.

```
handle_s = pscene.create_subject(oname="handle", gname=handle.name, _type=CustomObject,
                                 action_points_dict={gpoint.name: gpoint, ppoint.name: ppoint})
drawer = pscene.create_binder(bname=handle.name, gname=handle.name, _type=SweepFramer, point=(0, 0, 0),
                              rpy=(0, 0, 0))
```

```
sweep = pscene.create_subject(oname=dname, gname=drawer_bot.name, _type=SweepLineTask,
                              action_points_dict={gp1.name: gp1,
                                                  gp2.name: gp2}, one_direction=False)
```

Initial state: (handle, 0)



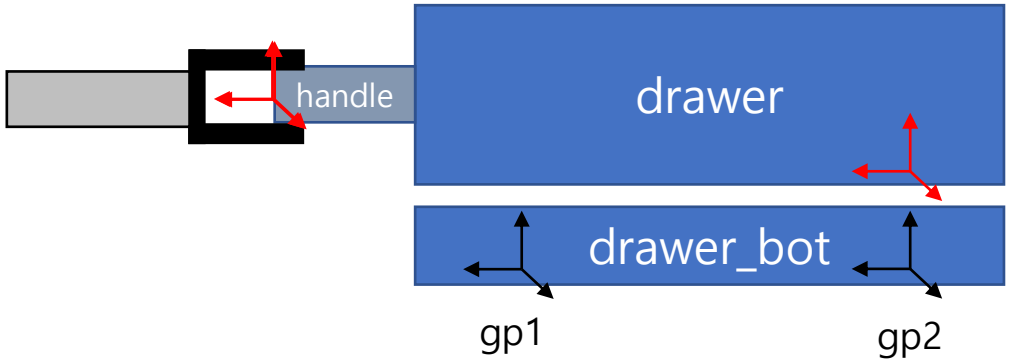
Drawer Task

Planning scene에 object들 추가.

```
handle_s = pscene.create_subject(oname="handle", gname=handle.name, _type=CustomObject,
                                 action_points_dict={gpoint.name: gpoint, ppoint.name: ppoint})
drawer = pscene.create_binder(bname=handle.name, gname=handle.name, _type=SweepFramer, point=(0, 0, 0),
                              rpy=(0, 0, 0))
```

```
sweep = pscene.create_subject(oname=dname, gname=drawer_bot.name, _type=SweepLineTask,
                              action_points_dict={gp1.name: gp1,
                                                  gp2.name: gp2}, one_direction=False)
```

Initial state: (handle, 0)
→ (grip0, 0)
→ (grip0, 1)



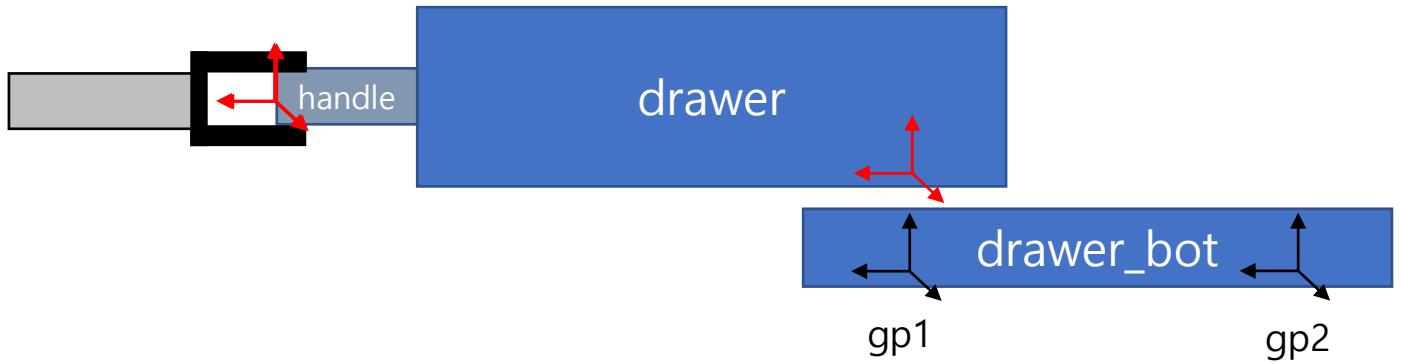
Drawer Task

Planning scene에 object들 추가.

```
handle_s = pscene.create_subject(oname="handle", gname=handle.name, _type=CustomObject,
                                 action_points_dict={gpoint.name: gpoint, ppoint.name: ppoint})
drawer = pscene.create_binder(bname=handle.name, gname=handle.name, _type=SweepFramer, point=(0, 0, 0),
                              rpy=(0, 0, 0))
```

```
sweep = pscene.create_subject(oname=dname, gname=drawer_bot.name, _type=SweepLineTask,
                              action_points_dict={gp1.name: gp1,
                                                  gp2.name: gp2}, one_direction=False)
```

Initial state: (drawer_bot, 0)
→ (grip0, 0)
→ (grip0, 1)
→ (grip0, 2)



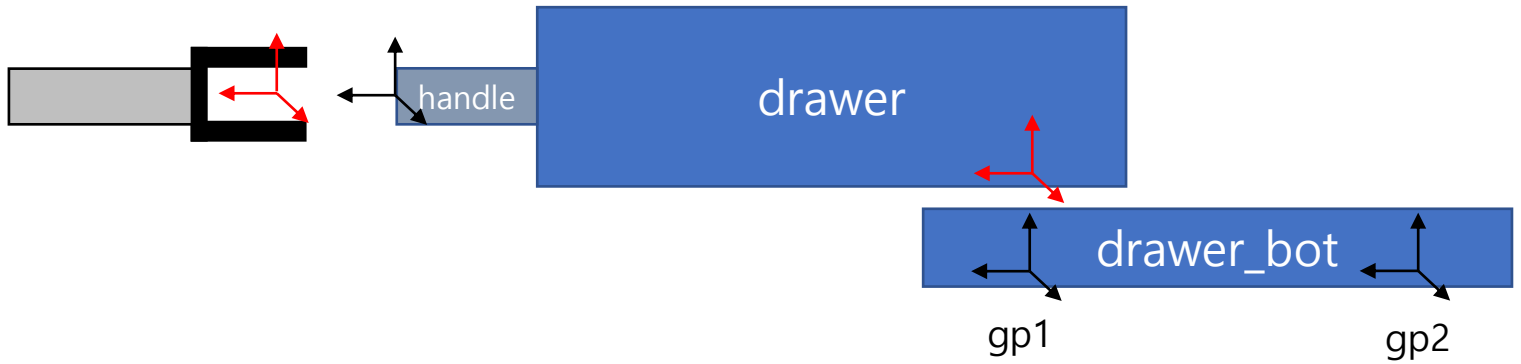
Drawer Task

Planning scene에 object들 추가.

```
handle_s = pscene.create_subject(oname="handle", gname=handle.name, _type=CustomObject,
                                 action_points_dict={gpoint.name: gpoint, ppoint.name: ppoint})
drawer = pscene.create_binder(bname=handle.name, gname=handle.name, _type=SweepFramer, point=(0, 0, 0),
                              rpy=(0, 0, 0))
```

```
sweep = pscene.create_subject(oname=dname, gname=drawer_bot.name, _type=SweepLineTask,
                              action_points_dict={gp1.name: gp1,
                                                  gp2.name: gp2}, one_direction=False)
```

Initial state: (drawer_bot, 0)
→ (grip0, 0)
→ (grip0, 1)
→ (grip0, 2)
→ (drawer_bot, 1)



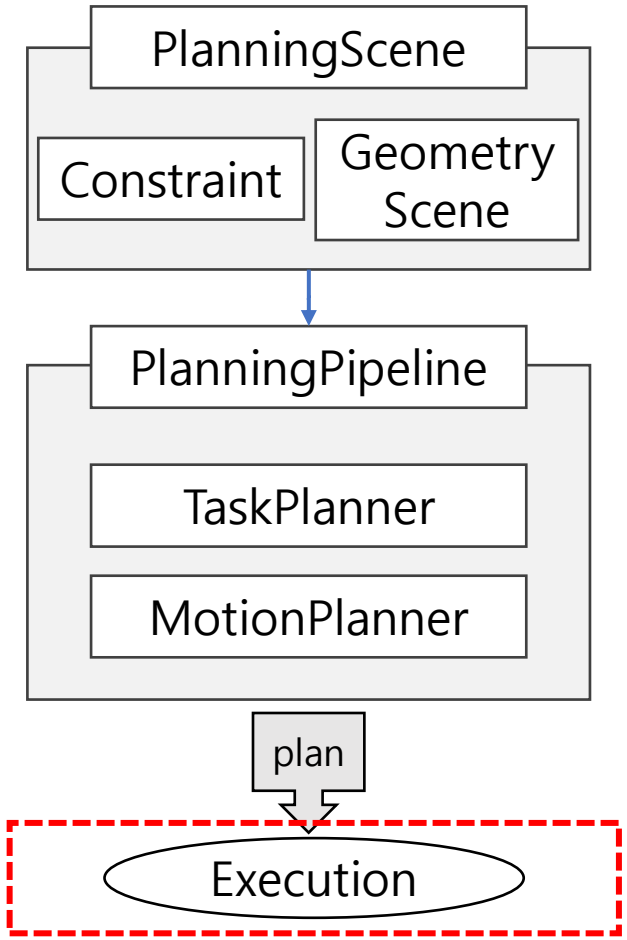
Drawer Task

DrawerTask를 추가하는 예제는 src/scripts/developing의 AddingTasks-Drawer.ipynb 파일에 있음.

* 이 script를 실행시킬 때, motion planning이 안될 수 있음.
해결 방법: motion planner의 incremental_constraint_motion을 True로 세팅한다.

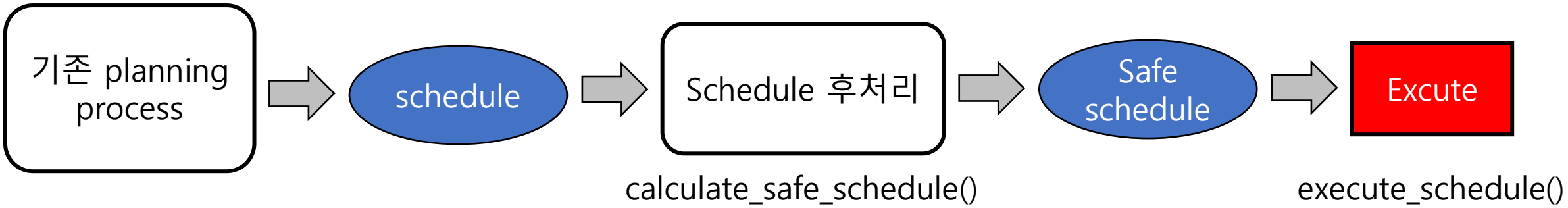
→ Motion planner는 motion constraint가 주어진 경우 point sampling과 해당 motion constraint를 만족 시키는 manifold에 대한 projection을 반복하여 constraint를 만족 시키는 motion을 찾아낸다.
→ 이는 load가 큰 작업이기 때문에 drawer task와 같이 sweeping하는 지역이 서랍 틀로 둘러 쌓여 있는 경우 motion plan이 실패할 가능성이 높아진다.
→ incremental_constraint_motion 옵션을 True로 세팅한 다는 것은 위의 과정을 생략하고 단순히 initial_state와 final_state를 interpolation하는 motion을 만들어 내게 한다는 것을 의미
→ Drawer Task의 경우 직선으로 서랍을 당기는 motion이기 때문에 True로 하여도 큰 문제는 없으나, constraint가 곡선으로 주어지는 Task의 경우는 문제가 발생한다.

Execution in Real Robot



Execution in real robot

release/5.PlanningAndExecution.ipynb 참조



Schedule 후처리 및 execution

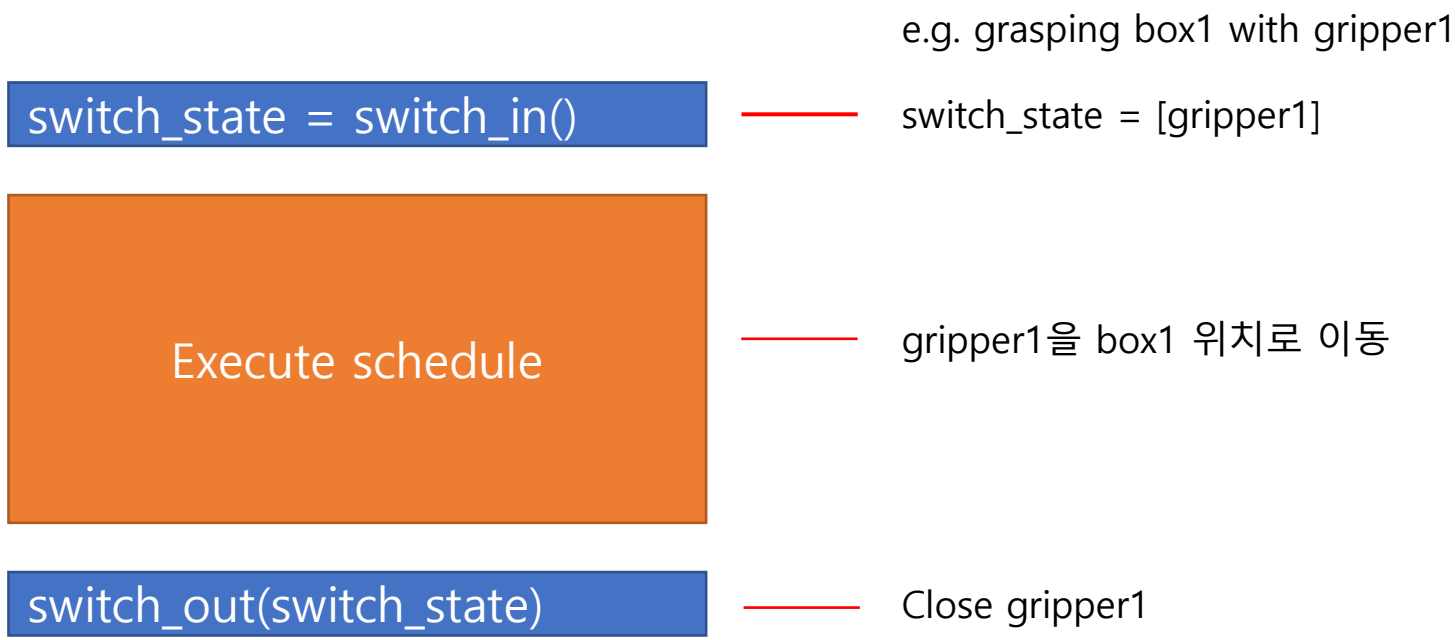
- calculate_safe_schedule(pscene, schedule, vel_lims=0.2, acc_lims=0.5)
  - 기존 planning 과정을 통해 계획한 motion trajectory가 제한 속도와 제한 가속도를 가지도록 schedule의 trajectory를 수정한다.

- execute_schedule(safe_schedule)
  - 계획된 schedule을 실제 로봇이 수행하도록 한다.
  - Planner 상에서의 grasp: gripper를 action point로 가져간 뒤 binding state 변경.
  - 실제 로봇에서의 grasp: gripper를 action point로 가져간 뒤 gripper를 열고 닫아야 한다.

  → GraspModeSwitcher 도입.

GraspModeSwitcher(ModeSwitcherTemplate)

- switch_in(self, snode_pre, snode_new)
  - grasp_dict에서 grasp 상태가 바뀌는 gripper의 리스트(switch_state)를 반환
- switch_out(self, switch_state, snode_new)
  - switch_state에 들어가 있는 gripper를 여닫음.

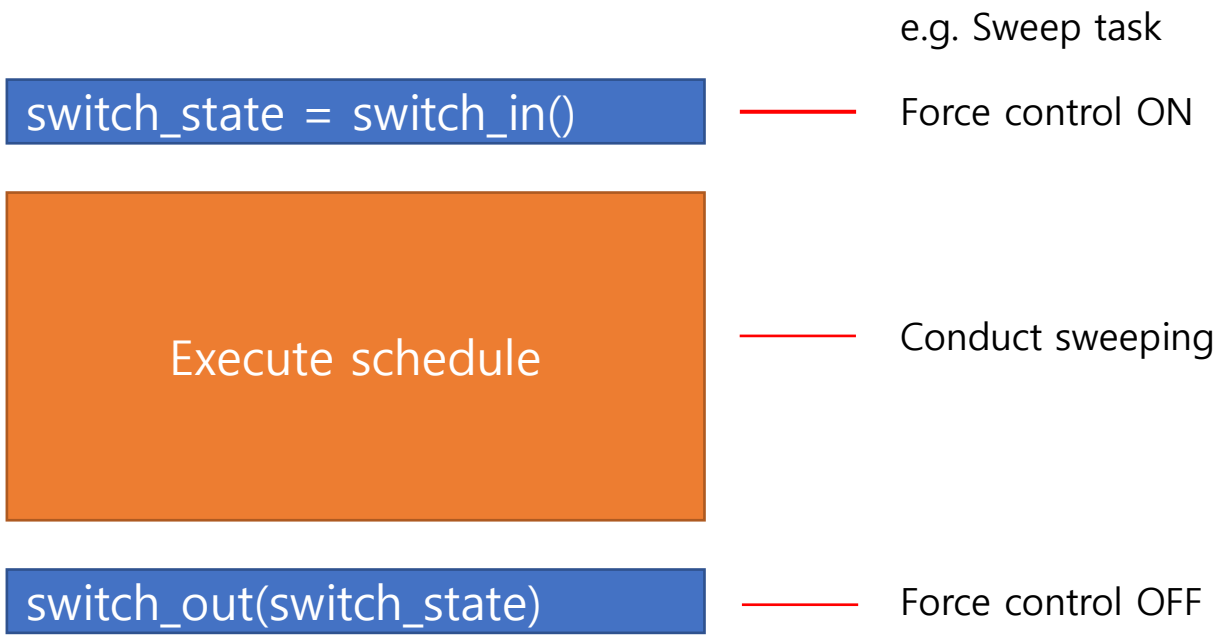


ForceOnlyModeSwitcher(ModeSwitcherTemplate)

For sweep task

- switch_in(self, snode_pre, snode_new)
  - Sweep task 수행을 위한 force control ON.

- switch_out(self, switch_state, snode_new)
  - Sweep task가 끝났으므로 force control OFF.



CombinedSwitcher(ModeSwitcherTemplate)

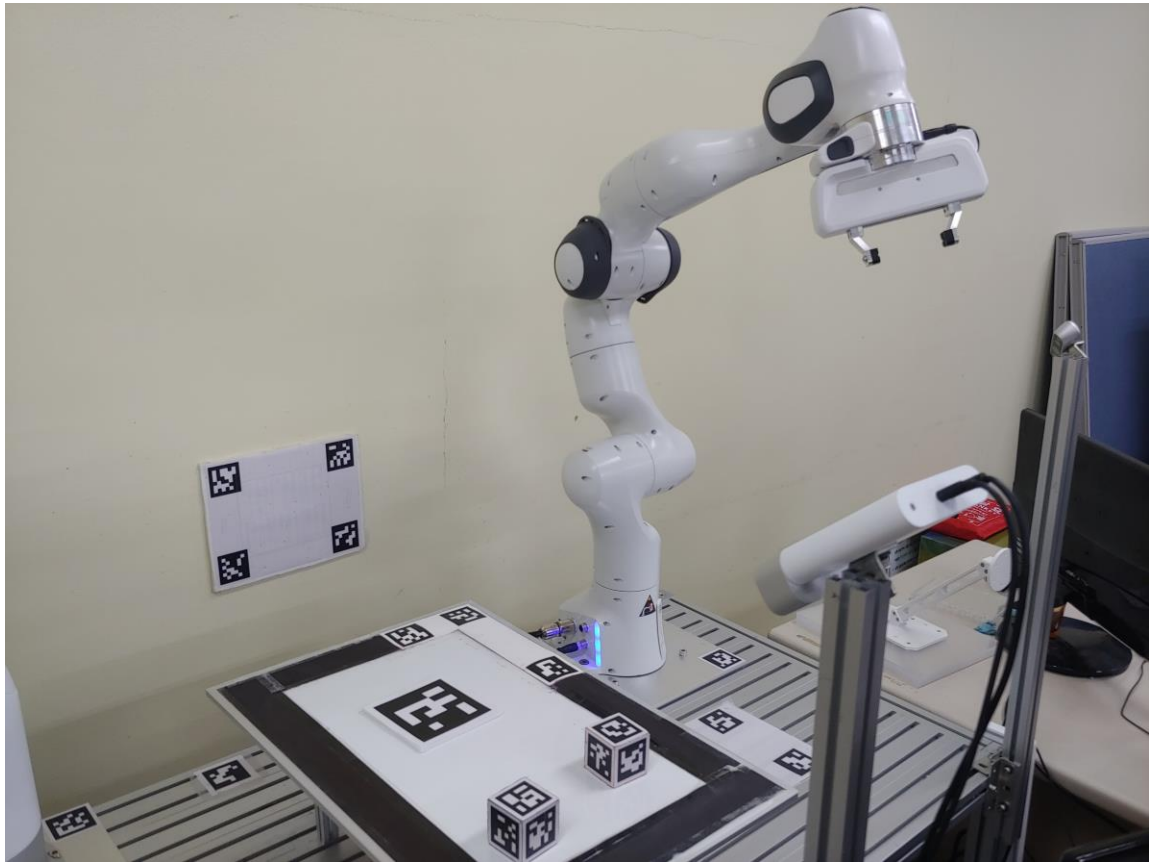
Plan에 한가지 task만 있는게 아닐 경우를 위한 switcher

- __init__(self, switch_list)
  - 각 task를 위한 switcher를 리스트 형식으로 저장

- switch_in(self, snode_pre, snode_new)
  - 반복문을 통해 switch_list에 있는 switcher들의 switch_in 메소드를 각각 실행
  - 실행 결과인 switch_state도 리스트 형태로 저장하여 반환한다.

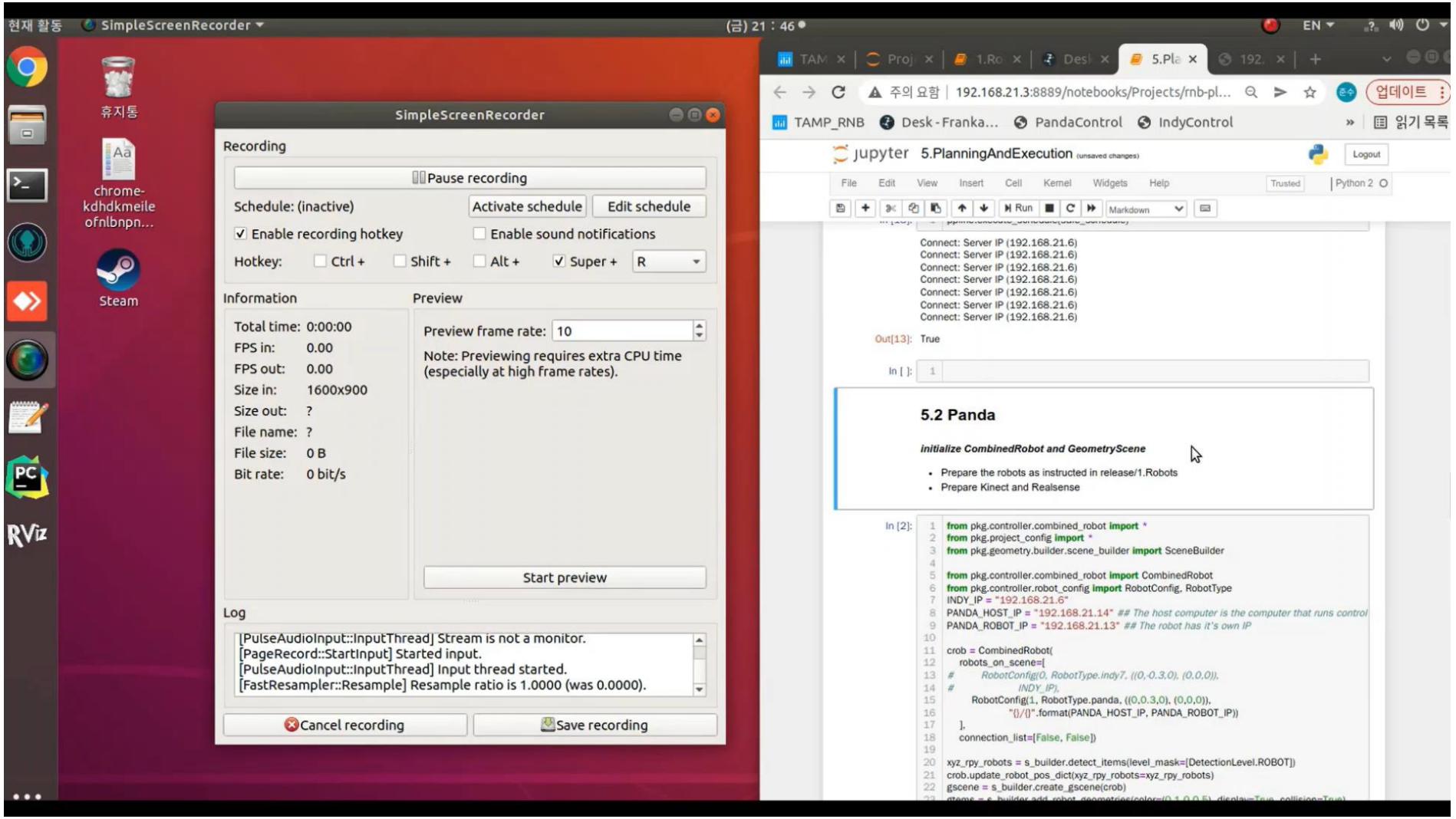
- switch_out(self, switch_state, snode_new)
  - 반목문을 통해 switch_list에 있는 switcher들의 switcher_out 메소드를 각각 실행

Demonstration



사전 준비 상태

Demonstration



Demonstration

Appendix

RRT Algorithm

Motion planner의 planning 알고리즘은 필요에 따라 변경 가능하다.

- RRT
  - 초기 configuration에서 시작하여 목표 configuration에 도달할 때까지 tree를 생성해 나가고, 목표 상태가 도달 되면 tree search를 통해 동작 계획을 생성
  - 기본 옵션으로 선택 되어 있는 plan 알고리즘

- RRTBi
  - Tree를 생성할 때, 초기 configuration에서뿐 아니라 목표 configuration에서도 역방향으로 tree를 생성해 나간다.

- RRTStar
  - Tree를 생성하는 과정에서 하나의 leaf가 생성될 때마다 최적의 경로를 계산하면서 tree를 생성해 나간다.
  - Optimal한 동작을 찾을 수 있다
  - RRT에 비해 계획 시간이 현저히 길어진다.

GeometryScene 구성

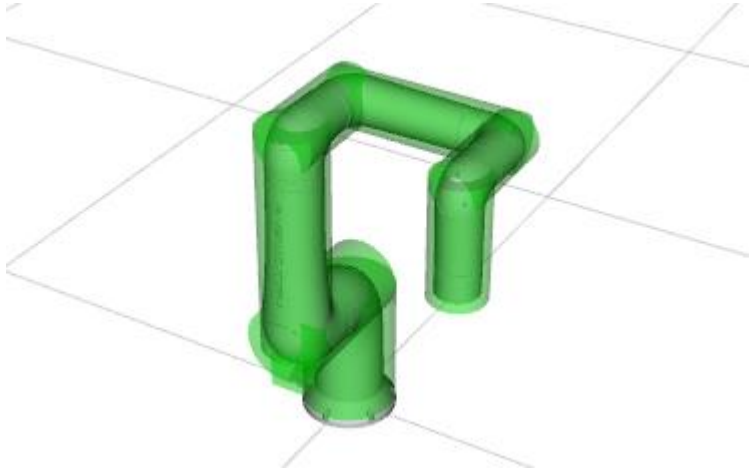
release/4.PlanningPipeline.ipynb 파일 참조

1. Initialize CombinedRobot and GeometryScene

```
crob = CombinedRobot(robots_on_scene=[
    RobotConfig(0, RobotType.indy7, ((0,0,0), (0,0,0)), None)]
                    , connection_list=[False])
gscene = s_builder.create_gscene(crob)
gtems = s_builder.add_robot_geometries(color=(0,1,0,0.5), display=True, collision=True)
gscene.show_pose(crob.home_pose)
```

Plan에 포함되는 로봇 리스트

로봇을 포함하는 geometry 구성(xacro파일 읽는 과정 포함)

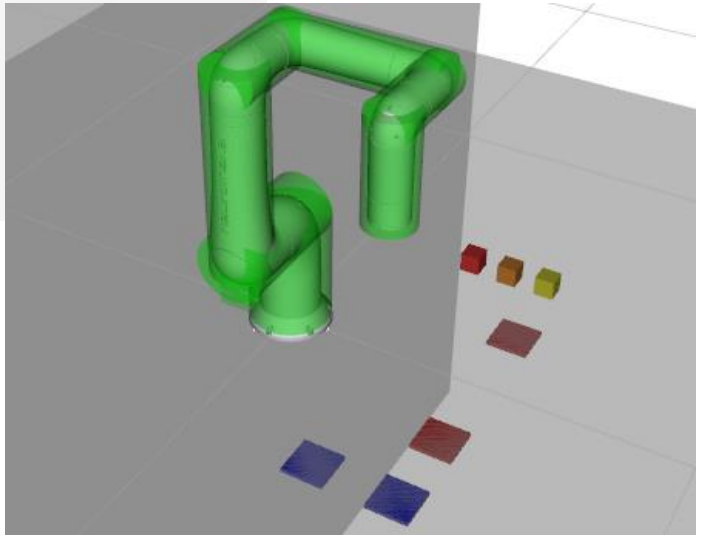


GeometryScene 구성

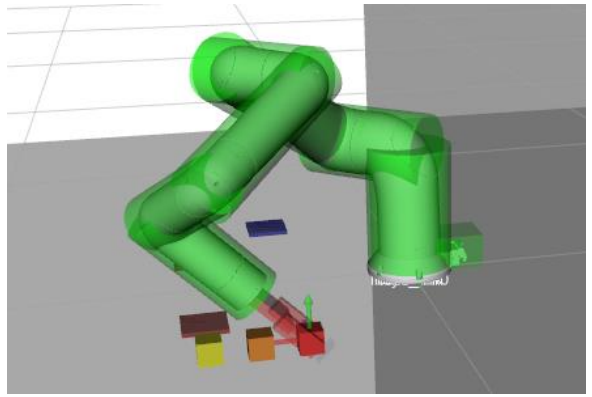
2. Add geometries

```
# add environments (fixed=True for non-movable geometries)
wall = gscene.create_safe(GEOTYPE.BOX, "wall", "base_link", (3,3,0.01), (-0.2,0,0),
                          rpy=(0,np.pi/2,0), color=(0.8,0.8,0.8,0.5), display=True, fixed=True, collision=True)
floor = gscene.create_safe(GEOTYPE.BOX, "floor", "base_link", (3,3,0.01), (0,0,0),
                          rpy=np.random.rand(3)*0, color=(0.8,0.8,0.8,0.5), display=True, fixed=True, collision=True)
#                           rpy=np.random.rand(3)*np.pi/18, color=(0.8,0.8,0.8,0.5), display=True, fixed=True, collision=True)
wp1 = gscene.create_safe(GEOTYPE.BOX, "wp1", "base_link", (0.1,0.1,0.01), (0.5,-0.2,0),rpy=(0,0,0),
                         color=(0.8,0.2,0.2,1), display=True, fixed=True, collision=False, parent="floor")
wp2 = gscene.create_safe(GEOTYPE.BOX, "wp2", "base_link", (0.1,0.1,0.01), (0.5,0.2,0), rpy=(0,0,0),
                         color=(0.8,0.2,0.2,1), display=True, fixed=True, collision=False, parent="floor")
goal = gscene.create_safe(GEOTYPE.BOX, "goal", "base_link", (0.1,0.1,0.01), (0.3,-0.4,0),
                          rpy=(0,0,0), color=(0.2,0.2,0.8,1), display=True, fixed=True, collision=True)

# add movable (fixed=False for movable geometries)
box1 = gscene.create_safe(GEOTYPE.BOX, "box1", "base_link", (0.05,0.05,0.05), (0.3,0.4,0.031),
                          rpy=(0,0,0), color=(0.8,0.2,0.2,1), display=True, fixed=False, collision=True)

obstacle = gscene.create_safe(GEOTYPE.BOX, "obstacle", "base_link", (0.05,0.05,0.05), (0.5,0.4,0.031),
                          rpy=(0,0,0), color=(0.8,0.8,0.2,1), display=True, fixed=False, collision=True)
```

Parent를 지정해 주면 parent에 해당하는 object의 기준의 좌표에 geometry 배치 가능.



Binding(ActionPoint, Binder)



```
{'grip0': {'w': 0.7710485482796943}, 'handle2': {'w': 0.0}}
('gripper', 'floor', 0)
```

Action Point: Object상에서 binding이 되는 점.

Binder: Action point를 붙일 수 있는 planning scene 상의 물체(점).

→ Binder와 Action Point를 설정하고 이 두 점의 좌표축을 맞추는 동작을 계획.

Redundancy: Binder와 Action Point를 일치 시키려고 할 때, 허용되는 오차의 범위(Action Point마다, Binder마다 각기 다른 redundancy를 동작의 종류를 고려하여 정해준다).

→ 본 프레임워크에서는 목표로 하는 Action Point와 Binder가 정해지면 각각의 total redundancy의 범위를 계산하고 범위 내의 한 값을 random하게 정한 뒤 motion planning을 진행.
→ Random하게 정한 redundancy로 motion plan이 불가능할 경우 반복문을 통해 새로운 redundancy 값을 sample해서 다시 motion plan.

sample_leaf_state() 함수를 참고하면 위에서 기술한 과정과 같이 from_state와 to_state (목표로 하는 binding)를 정하고 sample_redundancy() 함수를 호출하여 임의의 redundancy를 정하는 것을 확인 할 수 있음.

SweepTask(WayPointTask)

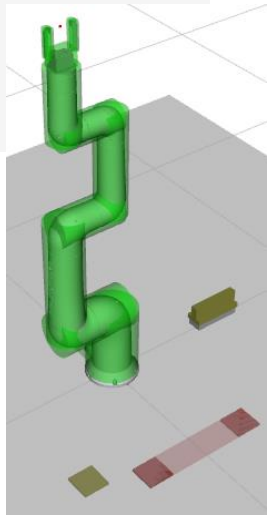
SweepTask(WaypointTask)

Object 각각을 정의하는 대신 sweep 하는 작업 자체를 정의하는 방식.

release/4.1.ConstrainedMotionTasks.ipynb 참조

<Planning Scene 구성>

```
brush_body = gscene.create_safe(gtype=GEOTYPE.BOX, name="brush_body", link_name="base_link", dims=(0.2,0.07,0.02),
                  center=(0,0.5,0.045), rpy=(0,0,np.pi/2), color=(0.7,0.7,0.3,1), display=True, collision=True, fixed=False)
brush_handle = gscene.create_safe(gtype=GEOTYPE.BOX, name="brush_handle", link_name="base_link", dims=(0.2,0.03,0.05), center=(0,0,0.035), rpy=(0,0,0),
                  color=(0.7,0.7,0.3,1), display=True, collision=True, fixed=False, parent="brush_body")
brush_face = gscene.create_safe(gtype=GEOTYPE.BOX, name="brush_face", link_name="base_link", dims=(0.19,0.06,0.03), center=(0,0,-0.025), rpy=(0,0,0),
                  color=(0.8,0.8,0.8,1), display=True, collision=False, fixed=False, parent="brush_body")

brush = pscene.create_subject(oname="brush", gname="brush_body", _type=CustomObject,
                              action_points_dict={"handle": Grasp2Point("handle", brush_handle, [0,0,0], [np.pi/2, 0, 0]),
                                                  "face": PlacePoint("face", brush_face, [0,0,-0.015], [0,0,0])})
```



```
pscene.create_binder(bname="grip0", gname="grip0", _type=Gripper2Tool, point=(0,0,0), rpy=(0,0,0))
pscene.create_binder(bname="floor", gname="floor", _type=PlacePlane, point=None)
pscene.create_binder(bname="goal_bd", gname="goal", _type=PlacePlane, point=(0,0,0.005), rpy=(0,0,0))
```

```
sweep = pscene.create_subject(oname="sweep", gname="floor", _type=SweepTask,
                              action_points_dict = {"wp1": SweepPoint("wp1", wp1, [0,0,0.005], [0,0,0]),
                                                    "wp2": SweepPoint("wp2", wp2, [0,0,0.005], [0,0,0])})
```

Pick & place를 위한 brush(object) sweep task가 scene 상에 존재.