Action Designator

Contents

This example will show the different kinds of Action Designators that are available. We will see how to create Action Designators and what they do.

Action Designators are high-level descriptions of actions which the robot should execute.

Action Designators are created from an Action Designator Description, which describes the type of action as well as the parameter for this action. Parameter are given as a list of possible parameters. For example, if you want to describe the robot moving to a table you would need a `NavigateAction()` and a list of poses that are near the table. The Action Designator Description will then pick one of the poses and return a performable Action Designator which contains the picked pose.

Navigate Action

↑ Back to top

We will start with a simple example of the `NavigateAction()`.

First, we need a BulletWorld with a robot.

```
from pycram.worlds.bullet_world import BulletWorld
from pycram.world_concepts.world_object import Object
from pycram.datastructures.enums import ObjectType, WorldMode

world = BulletWorld(WorldMode.GUI)
pr2 = Object("pr2", ObjectType.ROBOT, "pr2.urdf")
```

To move the robot we need to create a description and resolve it to an actual Designator. The description of navigation only needs a list of possible poses.

```
from pycram.designators.action_designator import NavigateAction
from pycram.datastructures.pose import Pose

pose = Pose([1, 0, 0], [0, 0, 0, 1])

# This is the Designator Description
navigate_description = NavigateAction(target_locations=[pose])

# This is the performable Designator
navigate_designator = navigate_description.resolve()
```

What we now did was: create the pose where we want to move the robot, create a description describing a navigation with a list of possible poses (in this case the list contains only one pose) and create an action designator from the description. The action designator contains the pose picked from the list of possible poses and can be performed.

```
from pycram.process_module import simulated_robot

with simulated_robot:
    navigate_designator.perform()
```

Every designator that is performed needs to be in an environment that specifies where to perform the designator either on the real robot or the simulated one. This environment is called `simulated_robot()` similar there is also a `real_robot()` environment.

There are also decorators which do the same thing but for whole methods, they are called `with_real_robot()` and `with_simulated_robot()`.

Move Torso

This action designator moves the torso up or down, specifically it sets the torso joint to a given value.

We start again by creating a description and resolving it to a designator. Afterwards, the designator is performed in a `simulated_robot()` environment.

```
from pycram.designators.action_designator import MoveTorsoAction
from pycram.process_module import simulated_robot

torso_pose = 0.2

torso_desig = MoveTorsoAction([torso_pose]).resolve()

with simulated_robot:
    torso_desig.perform()
```

Set Gripper

As the name implies, this action designator is used to open or close the gripper.

The procedure is similar to the last time, but this time we will shorten it a bit.

```
from pycram.designators.action_designator import SetGripperAction
from pycram.process_module import simulated_robot
from pycram.datastructures.enums import GripperState, Arms

gripper = Arms.RIGHT
motion = GripperState.OPEN

with simulated_robot:
    SetGripperAction(grippers=[gripper], motions=[motion]).resolve().perform()
```

Park Arms

Park arms is used to move one or both arms into the default parking position.

```
from pycram.designators.action_designator import ParkArmsAction
from pycram.process_module import simulated_robot
from pycram.datastructures.enums import Arms

with simulated_robot:
    ParkArmsAction([Arms.BOTH]).resolve().perform()
```

Pick Up and Place

Since these two are dependent on each other, meaning you can only place something when you picked it up beforehand, they will be shown together.

These action designators use object designators, which will not be further explained in this tutorial so please check the example on object designators for more details.

To start we need an environment in which we can pick up and place things as well as an object to pick up.

```
kitchen = Object("kitchen", ObjectType.ENVIRONMENT, "kitchen.urdf")
milk = Object("milk", ObjectType.MILK, "milk.stl", pose=Pose([1.3, 1, 0.9]))

world.reset_world()
```

latest

```
from pycram.designators.action_designator import PickUpAction, PlaceAction, Par

NavigateAction
from pycram.designators.object_designator import BelieveObject
from pycram.process_module import simulated_robot
from pycram.datastructures.enums import Arms, Grasp
from pycram.datastructures.pose import Pose

milk_desig = BelieveObject(names=["milk"])
arm = Arms.RIGHT

with simulated_robot:
    ParkArmsAction([Arms.BOTH]).resolve().perform()

    MoveTorsoAction([0.3]).resolve().perform()

    NavigateAction([Pose([0.78, 1, 0.0],
                         [0.0, 0.0, 0.014701099828940344, 0.9998919329926708])]

    PickUpAction(object_designator_description=milk_desig,
                 arms=[arm],
                 grasps=[Grasp.RIGHT]).resolve().perform()

    NavigateAction([Pose([-1.90, 0.78, 0.0],
                         [0.0, 0.0, 0.16439898301071468, 0.9863939245479175])]]

    PlaceAction(object_designator_description=milk_desig,
                target_locations=[Pose([-1.20, 1.0192, 0.9624],
                                       # [0.0, 0.0, 0.6339889056055381, 0.77334
                                       [0, 0, 0, 1])],
                arms=[arm]).resolve().perform()
```

```
world.reset_world()
```

Look At

Look at lets the robot look at a specific point, for example if it should look at an object for detecting.

latest

```
from pycram.designators.action_designator import LookAtAction
from pycram.process_module import simulated_robot
from pycram.datastructures.pose import Pose

target_location = Pose([1, 0, 0.5], [0, 0, 0, 1])
with simulated_robot:
    LookAtAction(targets=[target_location]).resolve().perform()
```

Detect

Detect is used to detect objects in the field of vision (FOV) of the robot. We will use the milk used in the pick up/place example, if you didn't execute that example you can spawn the milk with the following cell. The detect designator will return a resolved instance of an ObjectDesignatorDescription.

```
milk = Object("milk", ObjectType.MILK, "milk.stl", pose=Pose([1.3, 1, 0.9]))
```

```
from pycram.designators.action_designator import DetectAction, LookAtAction, Pa
from pycram.designators.object_designator import BelieveObject
from pycram.datastructures.enums import Arms
from pycram.process_module import simulated_robot
from pycram.datastructures.pose import Pose

milk_desig = BelieveObject(names=["milk"])

with simulated_robot:
    ParkArmsAction([Arms.BOTH]).resolve().perform()

    NavigateAction([Pose([0, 1, 0], [0, 0, 0, 1])]).resolve().perform()

    LookAtAction(targets=[milk_desig.resolve().pose]).resolve().perform()

    obj_desig = DetectAction(milk_desig).resolve().perform()

    print(obj_desig)
```

Transporting

latest

Transporting can transport an object from its current position to another target p
similar to the Pick and Place plan used in the Pick-up and Place example. Since we need an

Object which we can transport we spawn a milk, you don't need to do this if you already have spawned it in a previous example.

```
kitchen = Object("kitchen", ObjectType.ENVIRONMENT, "kitchen.urdf")
milk = Object("milk", ObjectType.MILK, "milk.stl", pose=Pose([1.3, 1, 0.9]))
```

```
from pycram.designators.action_designator import *
from pycram.designators.object_designator import *
from pycram.process_module import simulated_robot
from pycram.datastructures.pose import Pose
from pycram.datastructures.enums import Arms

milk_desig = BelieveObject(names=["milk"])

description = TransportAction(milk_desig,
                              [Arms.LEFT],
                              [Pose([-1.35, 0.78, 0.95],
                                    [0.0, 0.0, 0.16439898301071468, 0.98639392
with simulated_robot:
    MoveTorsoAction([0.2]).resolve().perform()
    description.resolve().perform()
```

Opening

Opening allows the robot to open a drawer, the drawer is identified by an ObjectPart designator which describes the handle of the drawer that should be grasped.

For the moment this designator works only in the apartment environment, therefore we remove the kitchen and spawn the apartment.

```
kitchen.remove()
```

```
apartment = Object("apartment", ObjectType.ENVIRONMENT, "apartment.urdf")
```

```
from pycram.designators.action_designator import *
from pycram.designators.object_designator import *
from pycram.datastructures.enums import Arms
from pycram.process_module import simulated_robot
from pycram.datastructures.pose import Pose

apartment_desig = BelieveObject(names=["apartment"]).resolve()
handle_deisg = ObjectPart(names=["handle_cab10_t"], part_of=apartment_desig)

with simulated_robot:
    MoveTorsoAction([0.25]).resolve().perform()
    ParkArmsAction([Arms.BOTH]).resolve().perform()
    NavigateAction([Pose([1.7474915981292725, 2.6873629093170166, 0.0],
                         [-0.0, 0.0, 0.5253598267689507, -0.850880163370435])])
    OpenAction(handle_deisg, [Arms.RIGHT]).resolve().perform()
```

Closing

Closing lets the robot close an open drawer, like opening the drawer is identified by an ObjectPart designator describing the handle to be grasped.

This action designator only works in the apartment environment for the moment, therefore we remove the kitchen and spawn the apartment. Additionally, we open the drawer such that we can close it with the action designator.

```
kitchen.remove()
```

```
apartment = Object("apartment", ObjectType.ENVIRONMENT, "apartment.urdf")
apartment.set_joint_state("cabinet10_drawer_top_joint", 0.4)
```

```
from pycram.designators.action_designator import *
from pycram.designators.object_designator import *
from pycram.datastructures.enums import Arms
from pycram.process_module import simulated_robot
from pycram.datastructures.pose import Pose

apartment_desig = BelieveObject(names=["apartment"]).resolve()
handle_deisg = ObjectPart(names=["handle_cab10_t"], part_of=apartment_desig)

with simulated_robot:
    MoveTorsoAction([0.25]).resolve().perform()
    ParkArmsAction([Arms.BOTH]).resolve().perform()
    NavigateAction([Pose([1.7474915981292725, 2.8073629093170166, 0.0],
                         [-0.0, 0.0, 0.5253598267689507, -0.850880163370435])])
    CloseAction(handle_deisg, [Arms.RIGHT]).resolve().perform()
```

```
world.exit()
```