Location Designator

Contents

This example will show you what location designators are, how to use them and what they are capable of.

Location Designators are used to semantically describe locations in the world. You could, for example, create a location designator that describes every position where a robot can be placed without colliding with the environment. Location designator can describe locations for:

- Visibility
- Reachability
- Occupancy
- URDF Links (for example a table)

To find locations that fit the given constrains, location designator create Costmaps. Costmaps are a 2D distribution that have a value greater than 0 for every position that fits the costmap criteria.

Location designators work similar to other designators, meaning you have to create a location designator description which describes the location. This description can then be resolved to the actual 6D pose on runtime.

Occupancy

We will start with a simple location designator that describes a location where the robot can be placed without colliding with the environment. To do this we need a BulletWorld since the costmaps are mostly created from the current state of the BulletWorld.

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

world = BulletWorld()
kitchen = Object("kitchen", ObjectType.ENVIRONMENT, "kitchen.urdf")
```

Next up we will create the location designator description, the `CostmapLocation()` that we will be using needs a target as a parameter. This target describes what the location designator is for, this could either be a pose or object that the robot should be able to see or reach.

In this case we only want poses where the robot can be placed, this is the default behaviour of the location designator which we will be extending later.

```
from pycram.designators.location_designator import CostmapLocation

target = kitchen.get_pose()

location_description = CostmapLocation(target)

pose = location_description.resolve()

print(pose)
```

Reachable

Next we want to have locations from where the robot can reach a specific point, like an object the robot should pick up. This can also be done with the `CostmapLocation()` description, but this time we need to provide an additional argument. The additional argument is the robot which should be able to reach the pose.

Since a robot is needed we will use the PR2 and use a milk as a target point fo reach. The torso of the PR2 will be set to 0.2 since otherwise the arms of the robot will be too

low to reach on the countertop.

```
pr2 = Object("pr2", ObjectType.ROBOT, "pr2.urdf")
pr2.set_joint_state("torso_lift_joint", 0.2)
milk = Object("milk", ObjectType.MILK, "milk.stl", pose=Pose([1.3, 1, 0.9]))
```

```
from pycram.designators.location_designator import CostmapLocation
from pycram.designators.object_designator import BelieveObject

target = BelieveObject(names=["milk"]).resolve()
robot_desig = BelieveObject(names=["pr2"]).resolve()

location_description = CostmapLocation(target=target, reachable_for=robot_desig

print(location_description.resolve())
```

As you can see we get a pose near the countertop where the robot can be placed without colliding with it. Furthermore, we get a list of arms with which the robot can reach the given object.

Visible

The `CostmapLocation()` can also find position from which the robot can see a given object or location. This is very similar to how reachable locations are described, meaning we provide a object designator or a pose and a robot designator but this time we use the `visible_for` parameter.

For this example we need the milk as well as the PR2, so if you did not spawn them during the previous location designator you can spawn them with the following cell.

```
pr2 = Object("pr2", ObjectType.ROBOT, "pr2.urdf")
milk = Object("milk", ObjectType.MILK, "milk.stl", pose=Pose([1.3, 1, 0.9]))
```

```
from pycram.designators.location_designator import CostmapLocation
from pycram.designators.object_designator import BelieveObject

target = BelieveObject(names=["milk"]).resolve()
robot_desig = BelieveObject(names=["pr2"]).resolve()

location_description = CostmapLocation(target=target, visible_for=robot_desig)

print(location_description.resolve())
```

Semantic

Semantic location designator are used to create location descriptions for semantic entities, like a table. An example of this is: You have a robot that picked up an object and should place it on a table. Semantic location designator then allows to find poses that are on this table.

Semantic location designator need an object from which the target entity is a part and the URDF link representing the entity. In this case we want a position on the kitchen island, so we have to provide the kitchen object designator since the island is a part of the kitchen and the link name of the island surface.

For this example we need the kitchen as well as the milk. If you spawned them in one of the previous examples you don't need to execute the following cell.

```
kitchen = Object("kitchen", ObjectType.ENVIRONMENT, "kitchen.urdf")
milk = Object("milk", ObjectType.MILK, "milk.stl")
```

```
from pycram.designators.location_designator import SemanticCostmapLocation
from pycram.designators.object_designator import BelieveObject

kitchen_desig = BelieveObject(names=["kitchen"]).resolve()
milk_desig = BelieveObject(names=["milk"]).resolve()

location_description = SemanticCostmapLocation(urdf_link_name="kitchen_island_s
                                               part_of=kitchen_desig,
                                               for_object=milk_desig)

print(location_description.resolve())
```

latest

Location Designator as Generator

Location designator descriptions implement an iter method, so they can be used as generators which generate valid poses for the location described in the description. This can be useful if the first pose does not work for some reason.

We will see this at the example of a location designator for visibility. For this example we need the milk, if you already have a milk spawned in you world you can ignore the following cell.

```
milk = Object("milk", ObjectType.MILK, "milk.stl")
```

```
from pycram.designators.location_designator import CostmapLocation
from pycram.designators.object_designator import BelieveObject

target = BelieveObject(names=["milk"]).resolve()
robot_desig = BelieveObject(names=["pr2"]).resolve()

location_description = CostmapLocation(target=target, visible_for=robot_desig)

for pose in location_description:
    print(pose.pose)
```

Accessing Locations

Accessing describes a location from which the robot can open a drawer. The drawer is specified by a ObjetcPart designator which describes the handle of the drawer.

At the moment this location designator only works in the apartment environment, so please remove the kitchen if you spawned it in a previous example. Furthermore, we need a robot, so we also spawn the PR2 if it isn't spawned already.

```
kitchen.remove()
```

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

```
pr2 = Object("pr2", ObjectType.ROBOT, "pr2.urdf")
pr2.set_joint_state("torso_lift_joint", 0.25)
```

```
from pycram.designators.object_designator import *
from pycram.designators.location_designator import *

apartment_desig = BelieveObject(names=["apartment"])
handle_desig = ObjectPart(names=["handle_cab10_t"], part_of=apartment_desig.res
robot_desig = BelieveObject(names=["pr2"])

access_location = AccessingLocation(handle_desig.resolve(), robot_desig.resolve
print(access_location.pose)
```

Giskard Location

Some robots like the HSR or the Stretch2 need a full-body ik solver to utilize the whole body. For this case the `GiskardLocation()` can be used. This location designator uses giskard as an ik solver to find a pose for the robot to reach a target pose.

Note: The GiskardLocation relies on Giskard, therefore Giskard needs to run in order for this Location Designator to work.

```
from pycram.designators.specialized_designators.location.giskard_location impor

robot_desig = BelieveObject(names=["pr2"]).resolve()

loc = GiskardLocation(target=Pose([1, 1, 1]), reachable_for=robot_desig).resolv
print(loc.pose)
```

If you are finished with this example you can close the world with the following cell:

```
world.exit()
```

latest