Introduction to Cognitive Robotics

Module 11: Cognition-enabled Robot Manipulation with CRAM
Lecture 1: Fetch-and-place CRAM plan for the PR2 robot with the Bullet real-time physics simulator

www.cognitiverobotics.net

Simple Fetch and Place Plan

There are two ways to follow this lecture

1: To walk through the process, interactively adding and testing functionality

This follows the CRAM zero prerequisites demo tutorial here:

`http://cram-system.org/tutorials/demo/fetch_and_place`

2. To walk through the process with reference to complete implementation

This follows the version of the tutorial here:

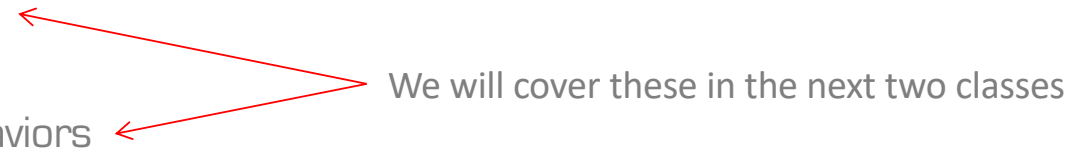
`http://www.vernon.eu/wiki/Zero_Prerequisites_Demo_Tutorial:_Simple_Fetch_and_Place`

Since all of the code is provided, this approach is faster to complete

Simple Fetch and Place Plan

Demonstrate how to write a plan for a simple task

- Pick-and-place
  - Pick an object from one position
  - Place it in another position in the world.
- Error handling
- Recovery behaviors



Environment Setup

Use `roslaunch` to instantiate the required ROS nodes

- PR2 robot
- Kitchen (specified just like a robot)
  - Doors have revolute (rotational) joints
  - Drawers have prismatic (translational) joints
  - Door handles have fixed joints

```
$ roslaunch cram_pick_place_tutorial world.launch
```

Command entered in a terminal

Configuration file

REPL Setup

(Read-Eval-Print Loop)

Use `roslisp_repl` to launch the Lisp compiler's interactive front-end: REPL

```
$ roslisp_repl &
```

Command entered in another terminal

REPL Setup

(Read-Eval-Print Loop)

And then, from REPL, at the Common Lisp User prompt `CL-USER>`

- Load the cram bullet world tutorial
- Make the cram-bullet-world-tutorial package current (this changes the prompt)

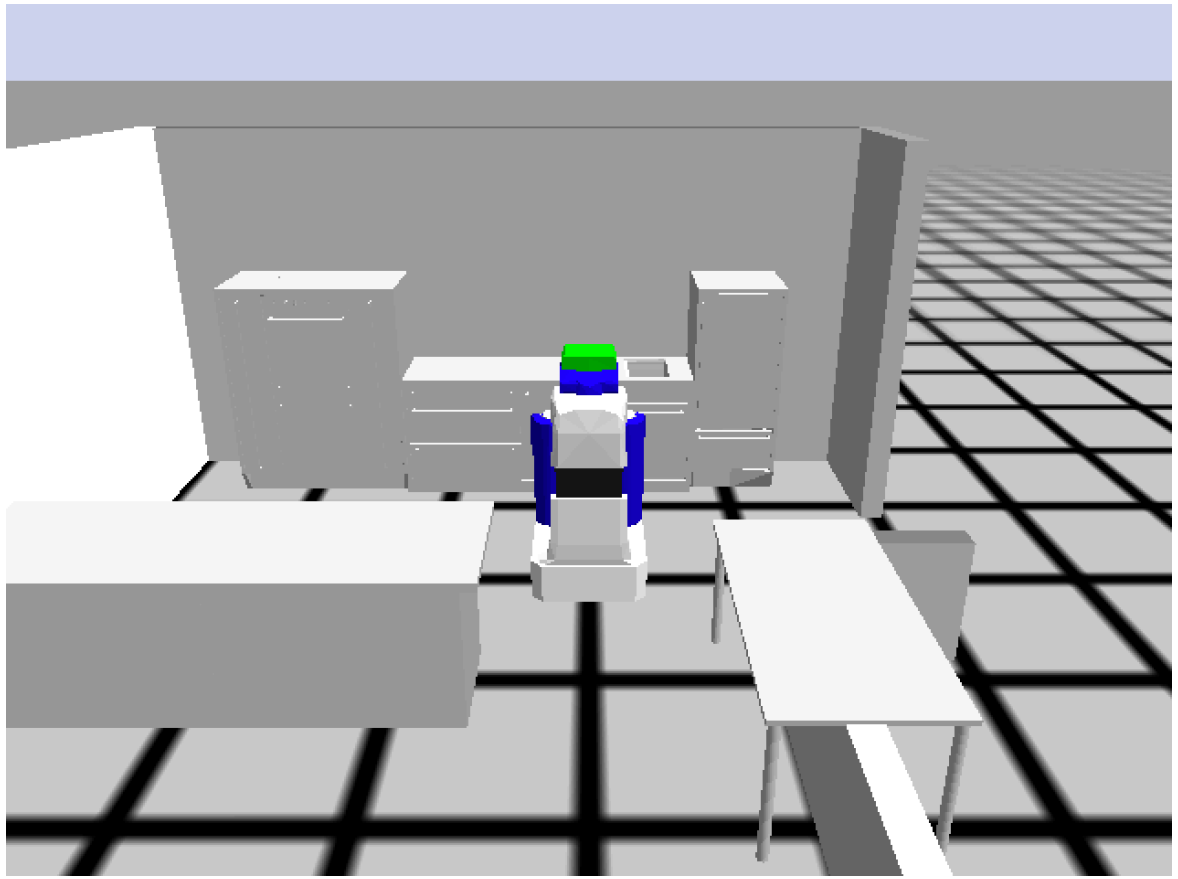
```
CL-USER> (ros-load:load-system "cram_pick_place_tutorial" :cram-pick-place-tutorial)
CL-USER> (in-package :cram-pick-place-tutorial)
```

- Start everything up

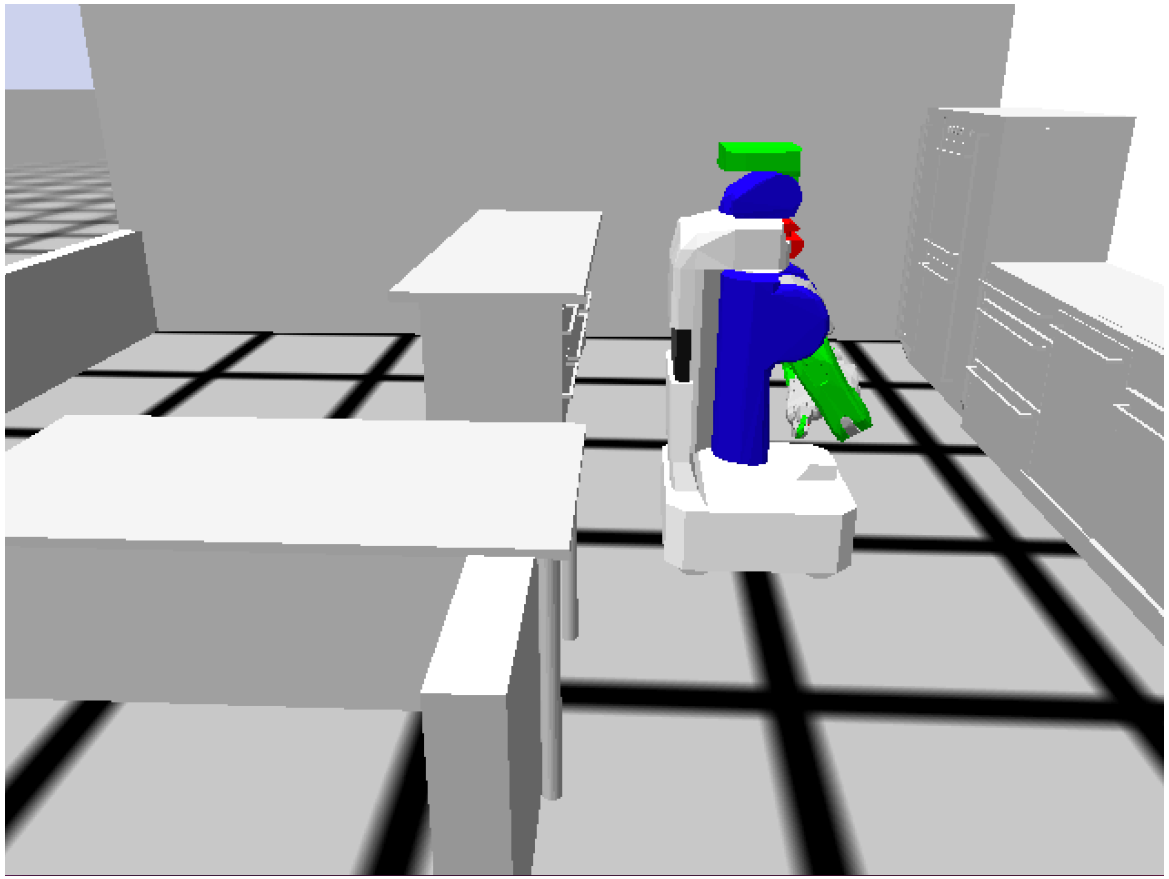
```
PP-TUT> (roslisp-utilities:startup-ros)
```

  - This will take some time (up to a minute)
  - It will launch a Bullet Real-Time Physics Simulation visualization window

Bullet Real-Time Physics Simulation



Bullet Real-Time Physics Simulation



Simple Fetch and Place Plan

Outline of a pick-and-place plan to pick a bottle from the table and place it on the counter

- Move the robot near the table.
- Move the arms out of the perception range, so that the robot can see the bottle without obstruction.
- Look towards the object area.
- Detect the object that has to be picked.
- Pick up the object and once again, keep the arms away from the front.
- Move the robot near the counter.
- Place the bottle on the destination.

Simple Fetch and Place Plan

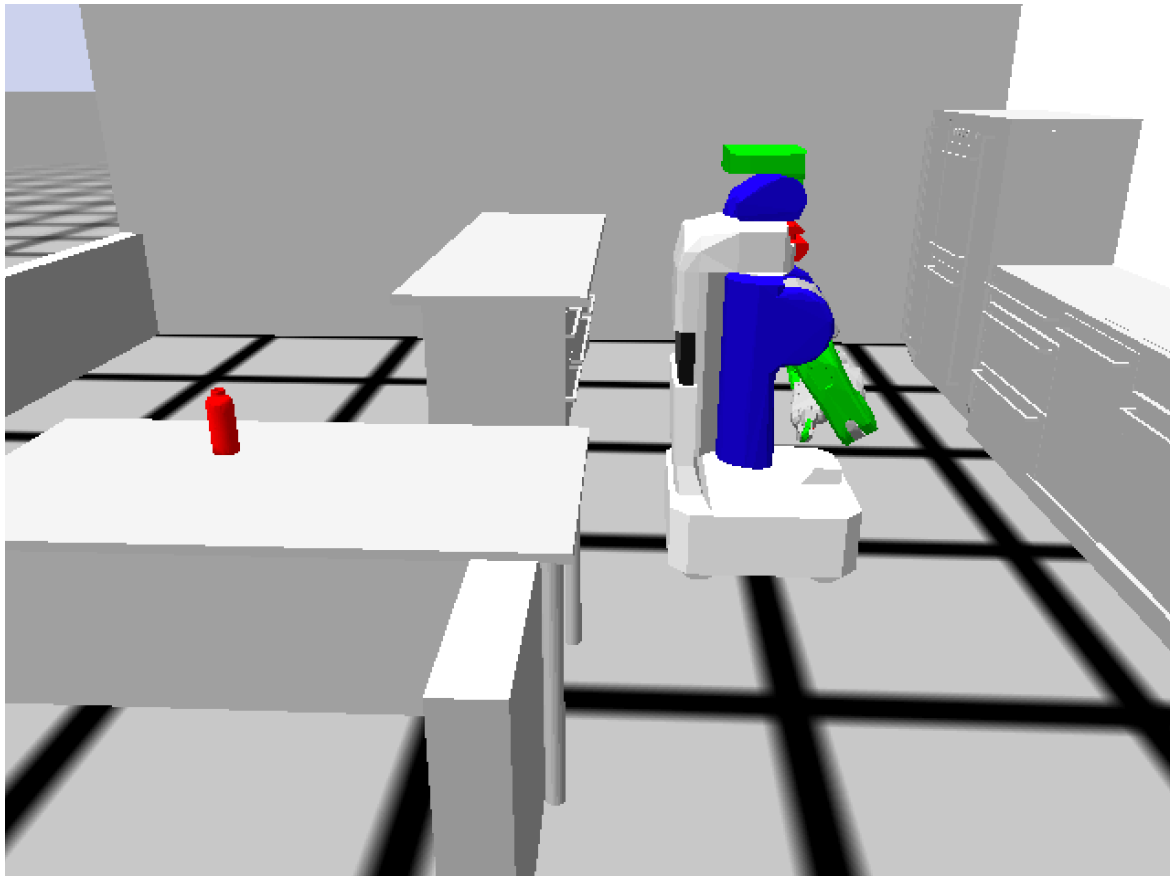
First, we need to create the bottle

General purpose function: default object is a red bottle

```
PP-TUT> (spawn-object '((-1.6 -0.9 0.82) (0 0 0 1)))
```

The pose is define using a vector and a quaternion
See Module 4, Lecture 2, Slides 9 – 17

Spawn the Bottle



Simple Fetch and Place Plan

For convenience, we specify some poses (position and orientation) *a priori*

```
(defparameter *base-pose-near-table*
  (make-pose "map" '((-1.447 -0.15 0.0) (0.0 0.0 -0.7071 0.7071))))

(defparameter *downward-look-coordinate*
  (make-pose "base_footprint" '((0.65335 0.076 0.758) (0 0 0 1))))

(defparameter *base-pose-near-counter*
  (make-pose "map" '((-0.15 2 0) (0 0 -1 0))))

(defparameter *final-object-destination*
  (make-pose "map" '((-0.8 2 0.9) (0 0 0 1))))
```

Robot position near the table

Position to look at for bottle

Robot position near the counter

Final destination of the bottle

Simple Fetch and Place Plan

Next, write the pick-and-place plan.

We write this in a function `move-bottle`

- Move the robot near the table.
- Move the arms out of the perception range, so that the robot can see the bottle without obstruction.
- Look towards the object area.
- Detect the object that has to be picked.
- Pick up the object and once again, keep the arms away from the front.
- Move the robot near the counter.
- Place the bottle on the destination.

```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                    (type moving-torso)
                    (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction))))))
    ;; Detect the bottle on the table.
    (let ((?grasping-arm :right)
          (?perceived-bottle (perform (an action
                                          (type detecting)
                                          (object (an object
                                                      (type bottle)))))))
      ;; Pick up the bottle
      (perform (an action
                   (type picking-up)
                   (arm ?grasping-arm)
                   (grasp left-side)
                   (object ?perceived-bottle)))
      (park-arm ?grasping-arm)
      ;; Moving the robot near the counter.
      (let ((?nav-goal *base-pose-near-counter*))
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?nav-goal))))))
      ;; Setting the bottle down on the counter
      (let ((?drop-pose *final-object-destination*))
        (perform (an action
                     (type placing)
                     (arm ?grasping-arm)
                     (object ?perceived-bottle)
                     (target (a location
                                 (pose ?drop-pose))))))
      (park-arm ?grasping-arm))))
```

Function to move the bottle
(Note indentation of everything below)

```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                    (type moving-torso)
                    (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction))))))
    ;; Detect the bottle on the table.
    (let ((?grasping-arm :right)
          (?perceived-bottle (perform (an action
                                          (type detecting)
                                          (object (an object
                                                      (type bottle)))))))
      ;; Pick up the bottle
      (perform (an action
                   (type picking-up)
                   (arm ?grasping-arm)
                   (grasp left-side)
                   (object ?perceived-bottle)))
      (park-arm ?grasping-arm)
      ;; Moving the robot near the counter.
      (let ((?nav-goal *base-pose-near-counter*))
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?nav-goal))))))
      ;; Setting the bottle down on the counter
      (let ((?drop-pose *final-object-destination*))
        (perform (an action
                     (type placing)
                     (arm ?grasping-arm)
                     (object ?perceived-bottle)
                     (target (a location
                                 (pose ?drop-pose))))))
      (park-arm ?grasping-arm))))
```

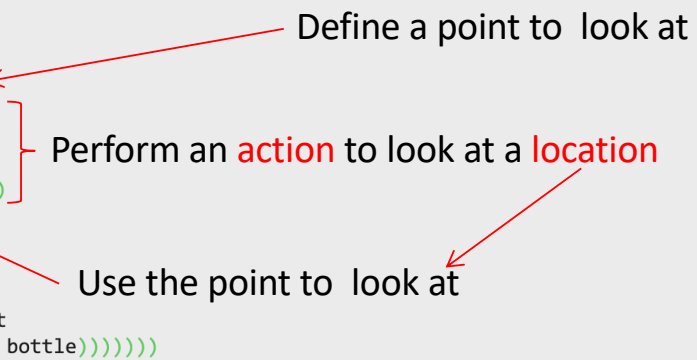
Create the bottle

```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                     (type moving-torso)
                     (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction))))))
    ;; Detect the bottle on the table.
    (let ((?grasping-arm :right)
          (?perceived-bottle (perform (an action
                                          (type detecting)
                                          (object (an object
                                                      (type bottle)))))))
      ;; Pick up the bottle
      (perform (an action
                   (type picking-up)
                   (arm ?grasping-arm)
                   (grasp left-side)
                   (object ?perceived-bottle)))
      (park-arm ?grasping-arm)
      ;; Moving the robot near the counter.
      (let ((?nav-goal *base-pose-near-counter*))
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?nav-goal))))))
      ;; Setting the bottle down on the counter
      (let ((?drop-pose *final-object-destination*))
        (perform (an action
                     (type placing)
                     (arm ?grasping-arm)
                     (object ?perceived-bottle)
                     (target (a location
                                 (pose ?drop-pose))))))
      (park-arm ?grasping-arm))))
```

Define a navigation goal
(note the indentation of what follows)
Then, perform three movements in parallel:

1. An action to the location near the table
   - Use the navigation goal
2. A motion to raise the robot torso
3. Park the arms

```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                     (type moving-torso)
                     (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction))))))
    ;; Detect the bottle on the table.
    (let ((?grasping-arm :right)
          (?perceived-bottle (perform (an action
                                          (type detecting)
                                          (object (an object
                                                      (type bottle)))))))
      ;; Pick up the bottle
      (perform (an action
                   (type picking-up)
                   (arm ?grasping-arm)
                   (grasp left-side)
                   (object ?perceived-bottle)))
      (park-arm ?grasping-arm)
      ;; Moving the robot near the counter.
      (let ((?nav-goal *base-pose-near-counter*))
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?nav-goal))))))
      ;; Setting the bottle down on the counter
      (let ((?drop-pose *final-object-destination*))
        (perform (an action
                     (type placing)
                     (arm ?grasping-arm)
                     (object ?perceived-bottle)
                     (target (a location
                                 (pose ?drop-pose))))))
      (park-arm ?grasping-arm))))
```



```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                    (type moving-torso)
                    (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction)))))
      ;; Detect the bottle on the table.
      (let ((?grasping-arm :right)
            (?perceived-bottle (perform (an action
                                            (type detecting)
                                            (object (an object
                                                        (type bottle)))))))
        ;; Pick up the bottle
        (perform (an action
                     (type picking-up)
                     (arm ?grasping-arm)
                     (grasp left-side)
                     (object ?perceived-bottle)))
        (park-arm ?grasping-arm)
        ;; Moving the robot near the counter.
        (let ((?nav-goal *base-pose-near-counter*))
          (perform (an action
                       (type going)
                       (target (a location
                                   (pose ?nav-goal)))))
          ;; Setting the bottle down on the counter
          (let ((?drop-pose *final-object-destination*))
            (perform (an action
                         (type placing)
                         (arm ?grasping-arm)
                         (object ?perceived-bottle)
                         (target (a location
                                     (pose ?drop-pose)))))
            (park-arm ?grasping-arm))))))
```

Use right arm to pick up the bottle

Perform an action to detect an object

The object should be a bottle

and store the detected object
(which includes the object pose)

```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                    (type moving-torso)
                    (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction))))))
    ;; Detect the bottle on the table.
    (let ((?grasping-arm :right)
          (?perceived-bottle (perform (an action
                                          (type detecting)
                                          (object (an object
                                                      (type bottle)))))))
      ;; Pick up the bottle
      (perform (an action
                   (type picking-up)
                   (arm ?grasping-arm)
                   (grasp left-side)
                   (object ?perceived-bottle)))
      (park-arm ?grasping-arm)
      ;; Moving the robot near the counter.
      (let ((?nav-goal *base-pose-near-counter*))
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?nav-goal))))))
      ;; Setting the bottle down on the counter
      (let ((?drop-pose *final-object-destination*))
        (perform (an action
                     (type placing)
                     (arm ?grasping-arm)
                     (object ?perceived-bottle)
                     (target (a location
                                 (pose ?drop-pose))))))
      (park-arm ?grasping-arm))))
```

Perform an action (of type picking-up) to pick up an object grasping from the left-side (more on grasp poses later) and the object is the bottle

```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                    (type moving-torso)
                    (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction))))))
    ;; Detect the bottle on the table.
    (let ((?grasping-arm :right)
          (?perceived-bottle (perform (an action
                                          (type detecting)
                                          (object (an object
                                                      (type bottle)))))))
      ;; Pick up the bottle
      (perform (an action
                   (type picking-up)
                   (arm ?grasping-arm)
                   (grasp left-side)
                   (object ?perceived-bottle)))
      (park-arm ?grasping-arm)
      ;; Moving the robot near the counter.
      (let ((?nav-goal *base-pose-near-counter*))
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?nav-goal))))))
      ;; Setting the bottle down on the counter
      (let ((?drop-pose *final-object-destination*))
        (perform (an action
                     (type placing)
                     (arm ?grasping-arm)
                     (object ?perceived-bottle)
                     (target (a location
                                 (pose ?drop-pose))))))
      (park-arm ?grasping-arm))))
```

The picking-up action designator is resolved by executing four atomic action designators (see runtime messages on later slide):

opening gripper

reaching

gripping

lifting

```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                    (type moving-torso)
                    (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction))))))
    ;; Detect the bottle on the table.
    (let ((?grasping-arm :right)
          (?perceived-bottle (perform (an action
                                          (type detecting)
                                          (object (an object
                                                      (type bottle)))))))
      ;; Pick up the bottle
      (perform (an action
                   (type picking-up)
                   (arm ?grasping-arm)
                   (grasp left-side)
                   (object ?perceived-bottle)))
      (park-arm ?grasping-arm)
      ;; Moving the robot near the counter.
      (let ((?nav-goal *base-pose-near-counter*))
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?nav-goal))))))
      ;; Setting the bottle down on the counter
      (let ((?drop-pose *final-object-destination*))
        (perform (an action
                     (type placing)
                     (arm ?grasping-arm)
                     (object ?perceived-bottle)
                     (target (a location
                                 (pose ?drop-pose))))))
      (park-arm ?grasping-arm))))
```

Parking the arm that grasped it

```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                    (type moving-torso)
                    (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction))))))
    ;; Detect the bottle on the table.
    (let ((?grasping-arm :right)
          (?perceived-bottle (perform (an action
                                          (type detecting)
                                          (object (an object
                                                      (type bottle)))))))
      ;; Pick up the bottle
      (perform (an action
                   (type picking-up)
                   (arm ?grasping-arm)
                   (grasp left-side)
                   (object ?perceived-bottle)))
      (park-arm ?grasping-arm)
      ;; Moving the robot near the counter.
      (let ((?nav-goal *base-pose-near-counter*))
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?nav-goal))))))
      ;; Setting the bottle down on the counter
      (let ((?drop-pose *final-object-destination*))
        (perform (an action
                     (type placing)
                     (arm ?grasping-arm)
                     (object ?perceived-bottle)
                     (target (a location
                                 (pose ?drop-pose))))))
      (park-arm ?grasping-arm))))
```

Define a navigation goal

Perform an action of type going to the location near the counter

Use the navigation goal

```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                    (type moving-torso)
                    (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction))))))
    ;; Detect the bottle on the table.
    (let ((?grasping-arm :right)
          (?perceived-bottle (perform (an action
                                          (type detecting)
                                          (object (an object
                                                      (type bottle)))))))
      ;; Pick up the bottle
      (perform (an action
                   (type picking-up)
                   (arm ?grasping-arm)
                   (grasp left-side)
                   (object ?perceived-bottle)))
      (park-arm ?grasping-arm)
      ;; Moving the robot near the counter.
      (let ((?nav-goal *base-pose-near-counter*))
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?nav-goal))))))
      ;; Setting the bottle down on the counter
      (let ((?drop-pose *final-object-destination*))
        (perform (an action
                     (type placing)
                     (arm ?grasping-arm)
                     (object ?perceived-bottle)
                     (target (a location
                                 (pose ?drop-pose))))))
      (park-arm ?grasping-arm))))
```

Define the final pose of the bottle on the counter

Perform an action of type placing to place
an object (the bottle)
at a location
which is the final pose

```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                    (type moving-torso)
                    (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction))))))
    ;; Detect the bottle on the table.
    (let ((?grasping-arm :right)
          (?perceived-bottle (perform (an action
                                          (type detecting)
                                          (object (an object
                                                      (type bottle)))))))
      ;; Pick up the bottle
      (perform (an action
                   (type picking-up)
                   (arm ?grasping-arm)
                   (grasp left-side)
                   (object ?perceived-bottle)))
      (park-arm ?grasping-arm)
      ;; Moving the robot near the counter.
      (let ((?nav-goal *base-pose-near-counter*))
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?nav-goal))))))
      ;; Setting the bottle down on the counter
      (let ((?drop-pose *final-object-destination*))
        (perform (an action
                     (type placing)
                     (arm ?grasping-arm)
                     (object ?perceived-bottle)
                     (target (a location
                                 (pose ?drop-pose))))))
      (park-arm ?grasping-arm))))
```

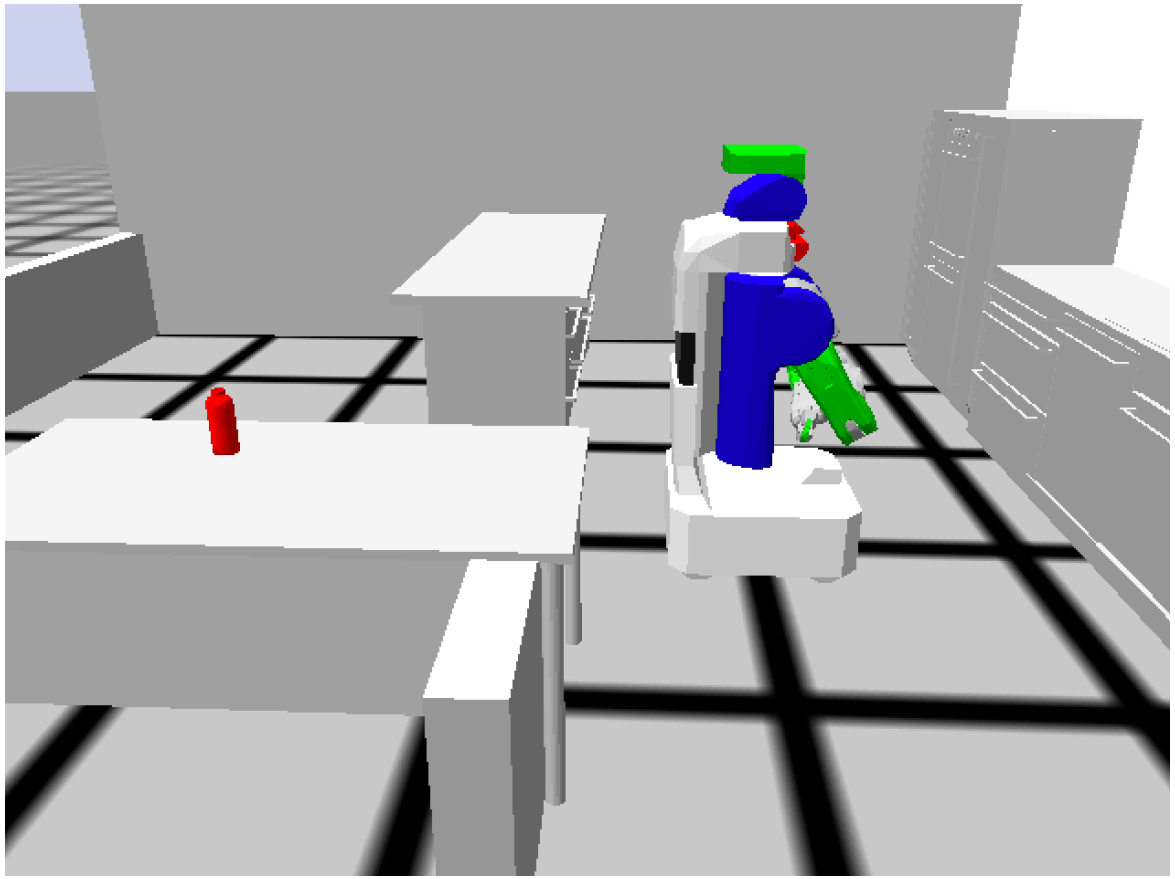
The placing action designator is resolved by executing four atomic action designators (see runtime messages on later slide):

reaching
putting
opening gripper
retracting

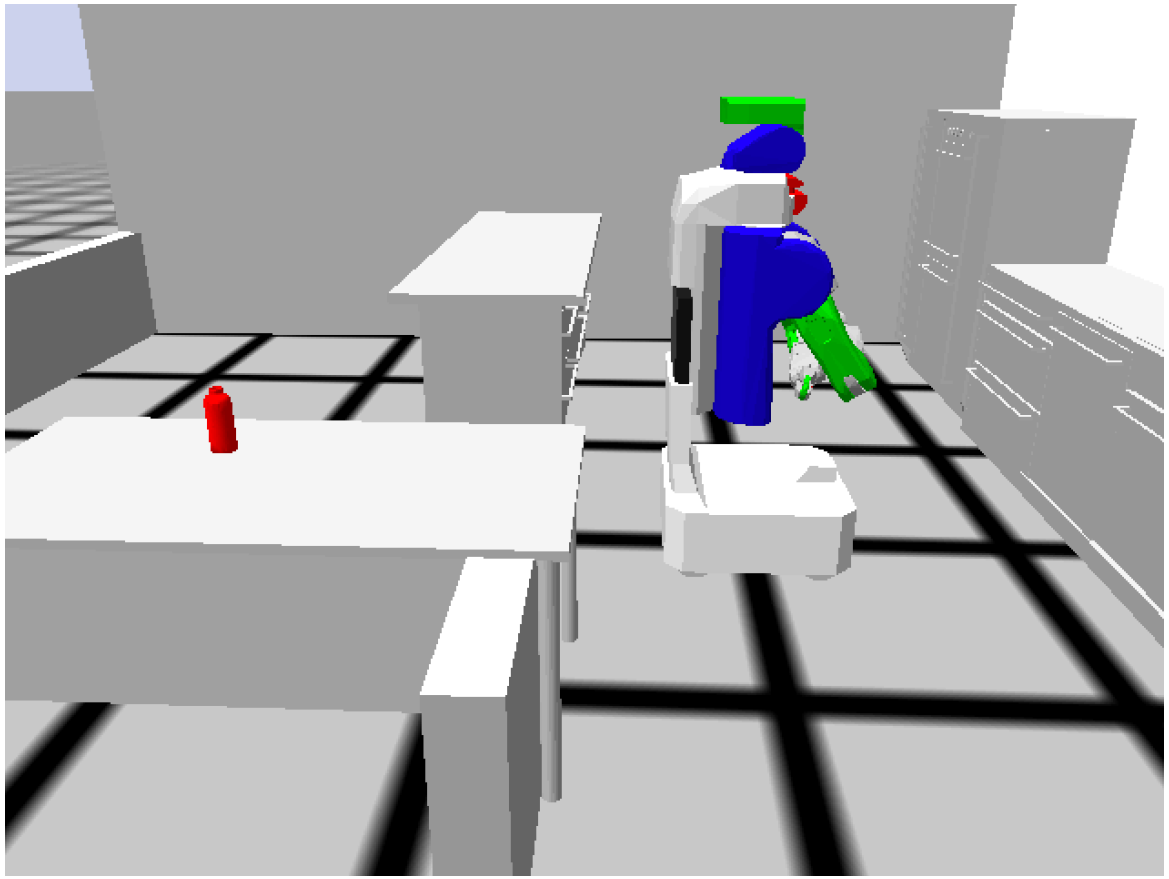
```
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?navigation-goal)))))
        (perform (a motion
                    (type moving-torso)
                    (joint-angle 0.3)))
        (park-arms)))
    ;; Looking towards the bottle before perceiving.
    (let ((?looking-direction *downward-look-coordinate*))
      (perform (an action
                   (type looking)
                   (target (a location
                               (pose ?looking-direction))))))
    ;; Detect the bottle on the table.
    (let ((?grasping-arm :right)
          (?perceived-bottle (perform (an action
                                          (type detecting)
                                          (object (an object
                                                      (type bottle)))))))
      ;; Pick up the bottle
      (perform (an action
                   (type picking-up)
                   (arm ?grasping-arm)
                   (grasp left-side)
                   (object ?perceived-bottle)))
      (park-arm ?grasping-arm)
      ;; Moving the robot near the counter.
      (let ((?nav-goal *base-pose-near-counter*))
        (perform (an action
                     (type going)
                     (target (a location
                                 (pose ?nav-goal))))))
      ;; Setting the bottle down on the counter
      (let ((?drop-pose *final-object-destination*))
        (perform (an action
                     (type placing)
                     (arm ?grasping-arm)
                     (object ?perceived-bottle)
                     (target (a location
                                 (pose ?drop-pose))))))
      (park-arm ?grasping-arm))))
```

Return the grasping arm to the parked position

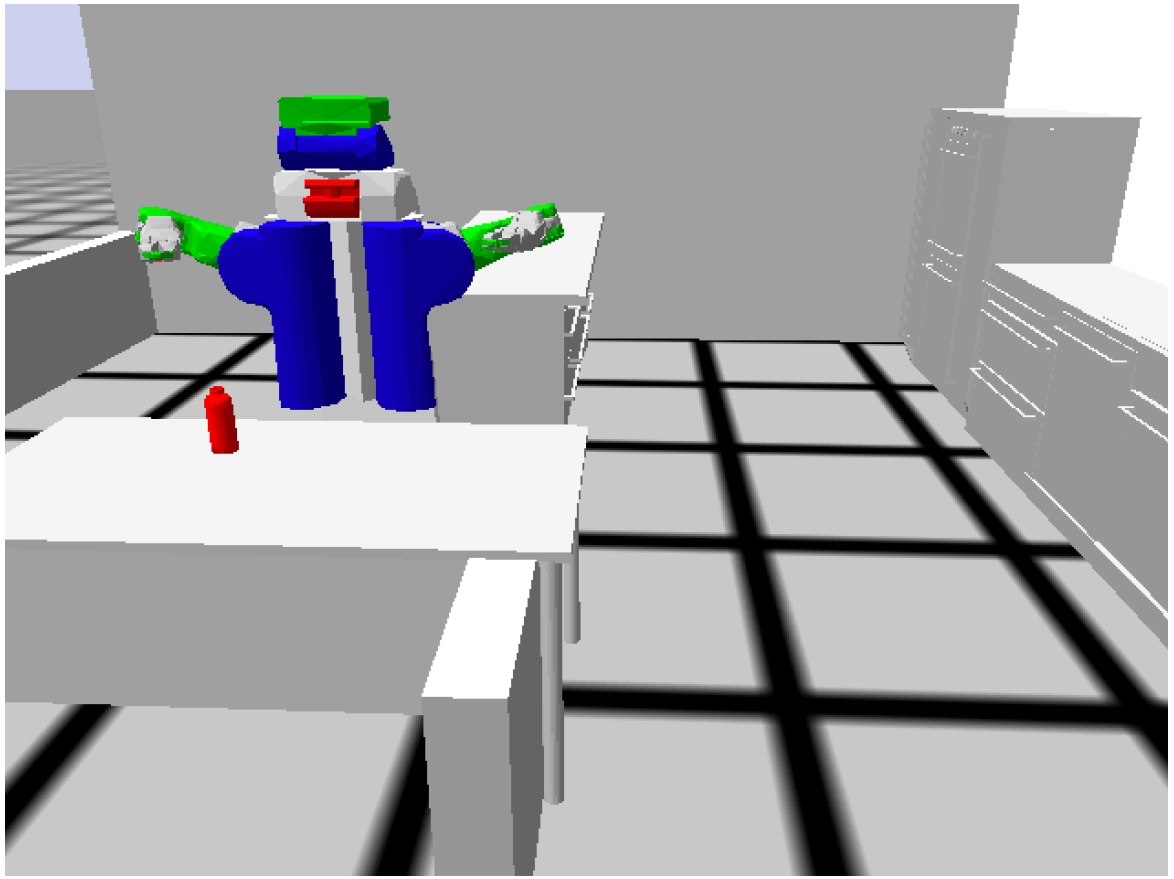
Spawn the bottle



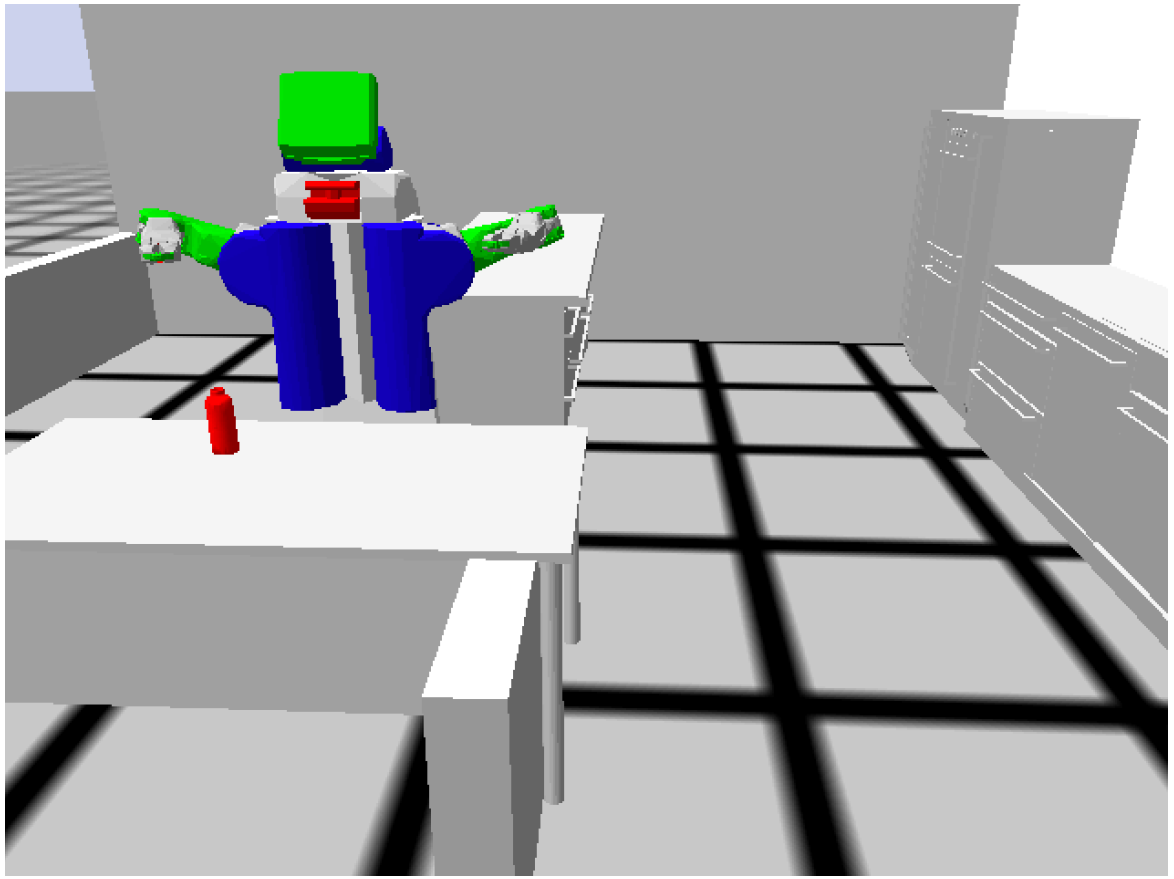
Perform motion
adjust torso



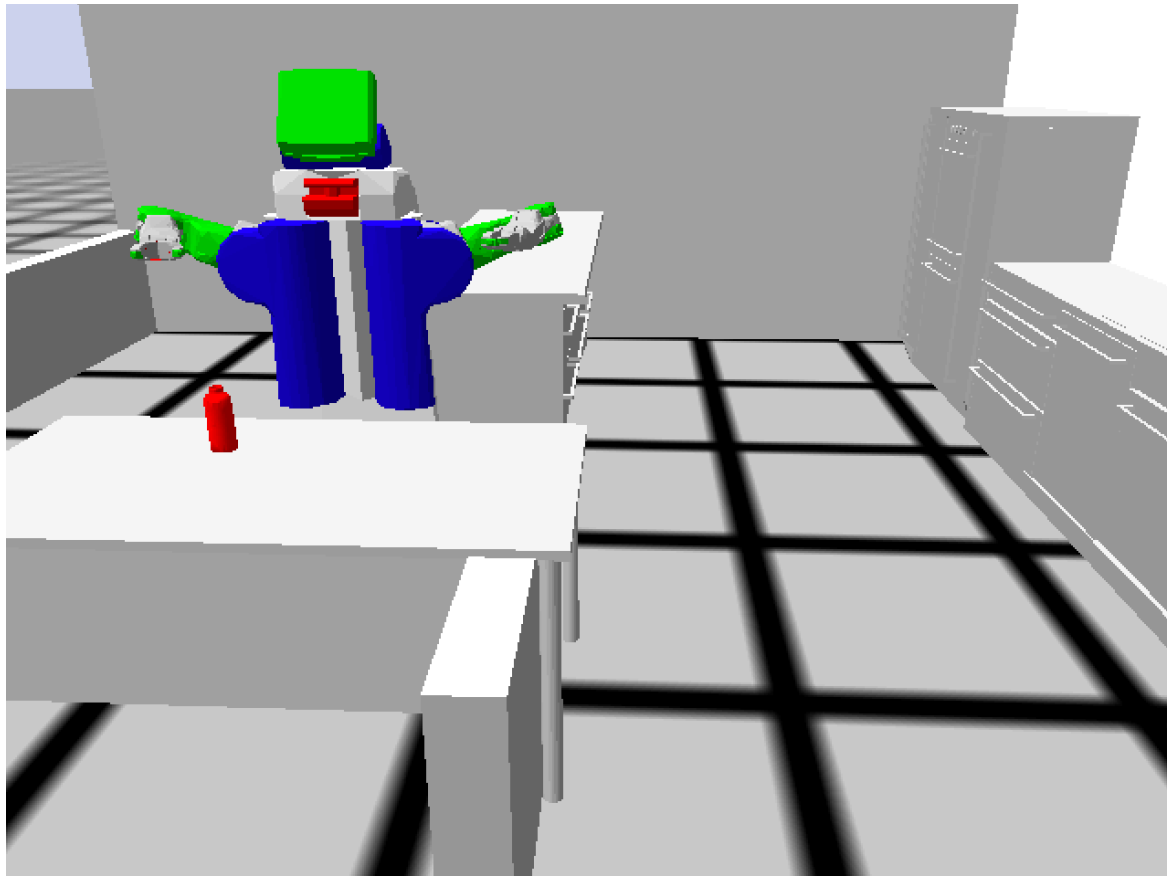
Perform action
move near the table



Perform action
look towards the bottle

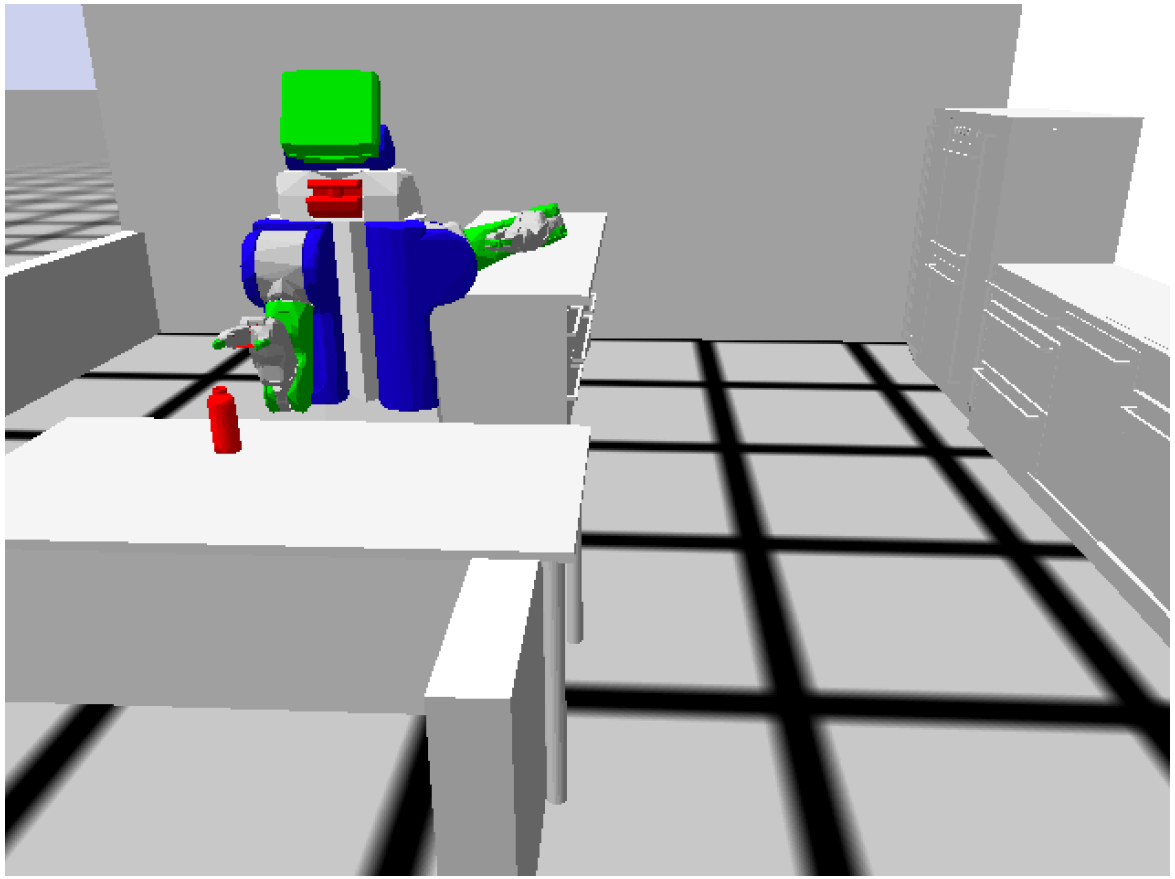


Perform action
pick-up: opening gripper

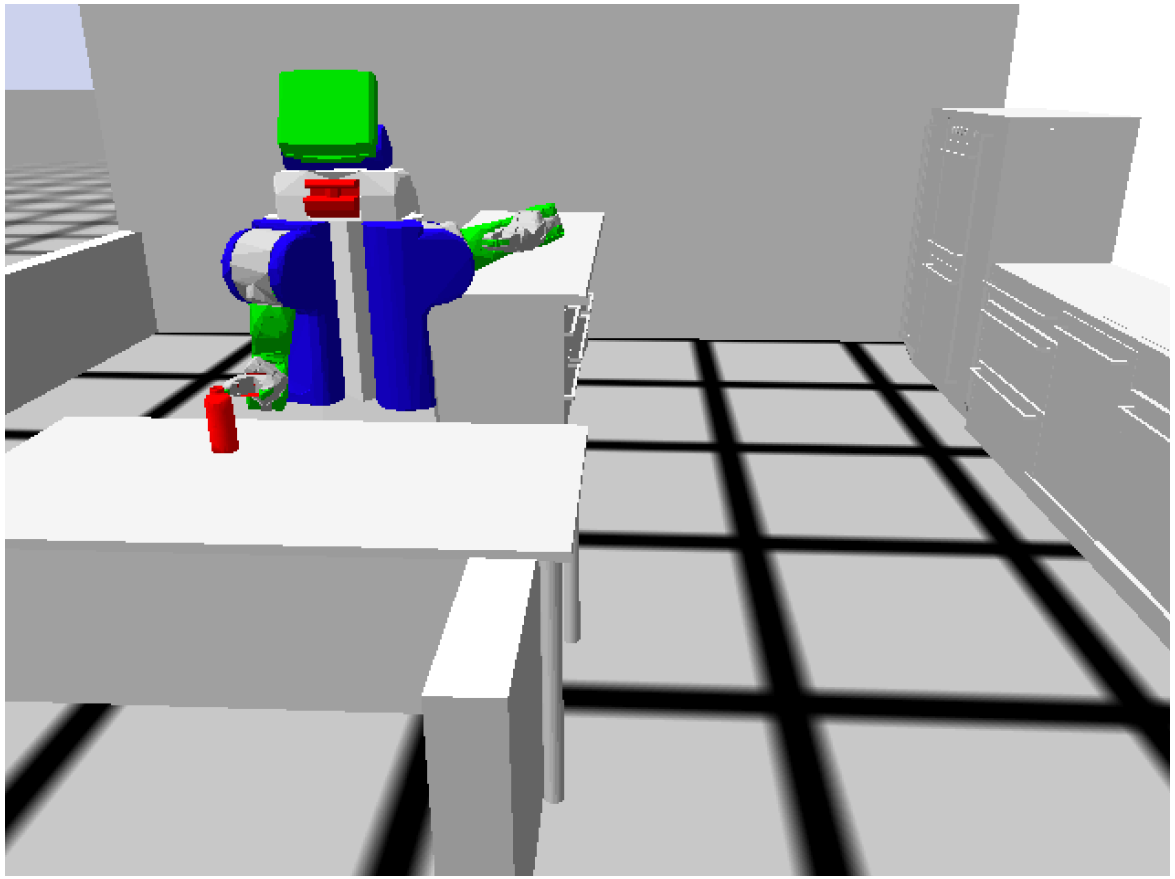


Perform action

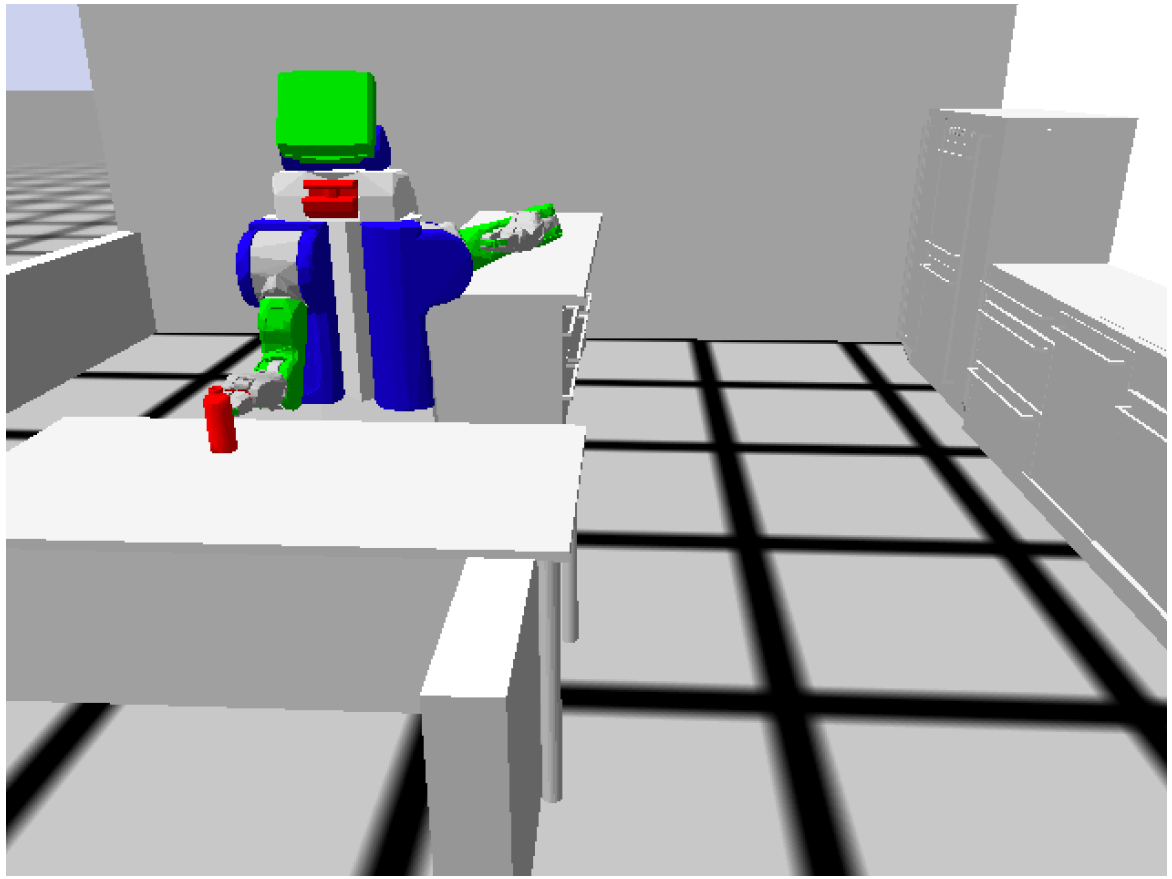
pick-up: reaching



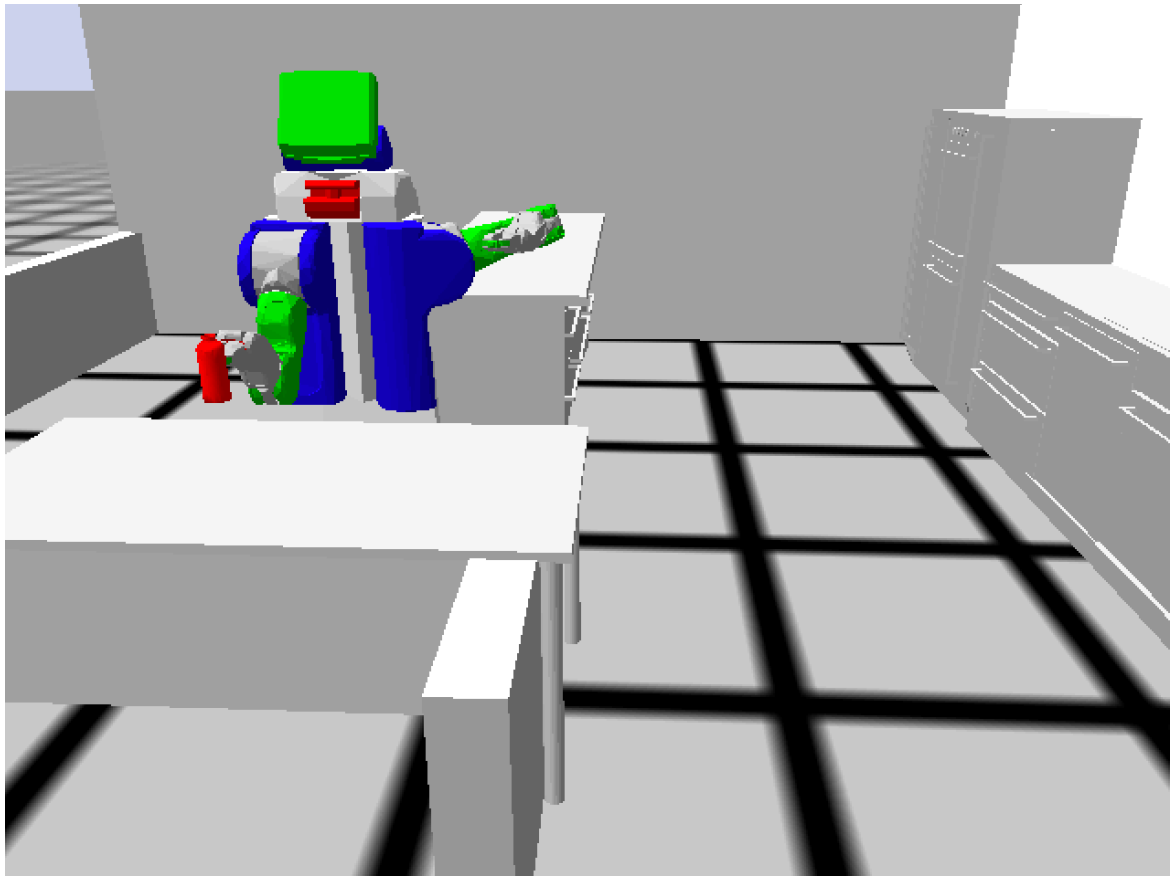
Perform action
pick-up: reaching



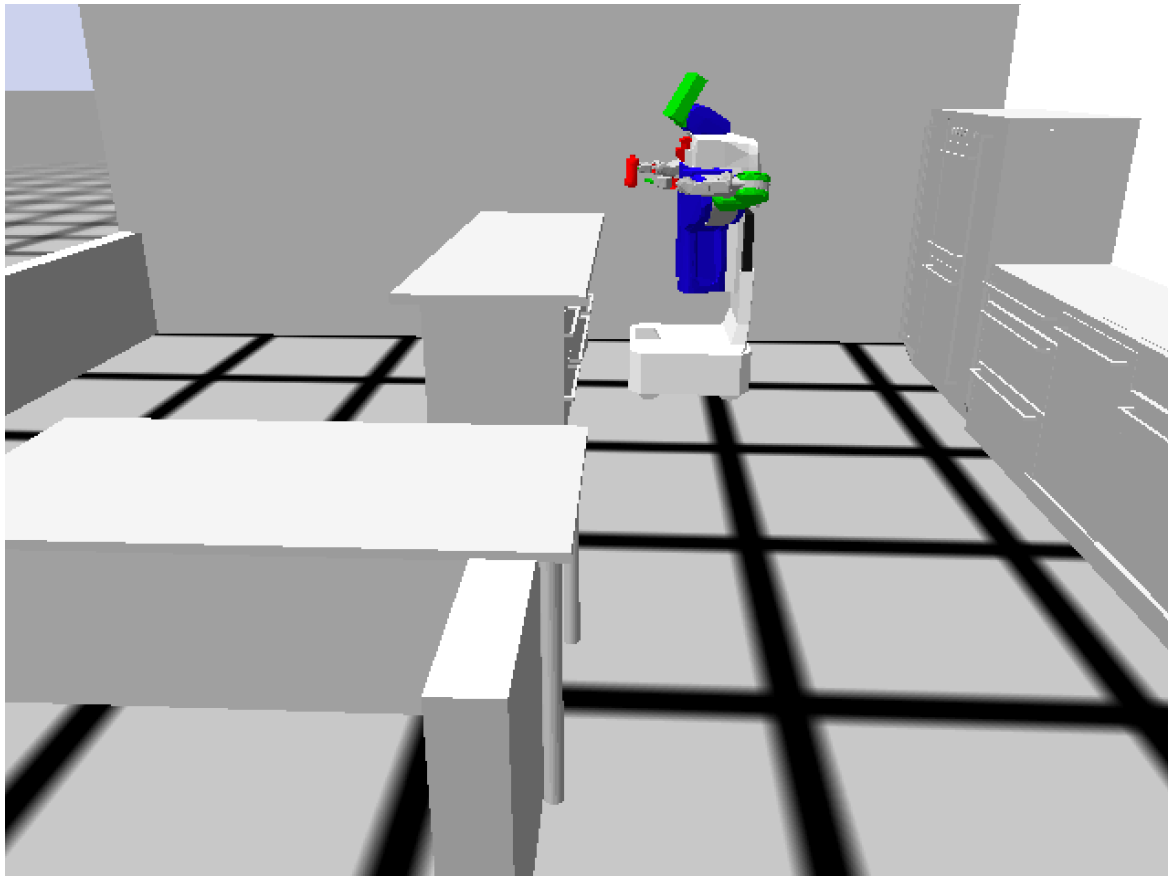
Perform action
pick-up: grasping



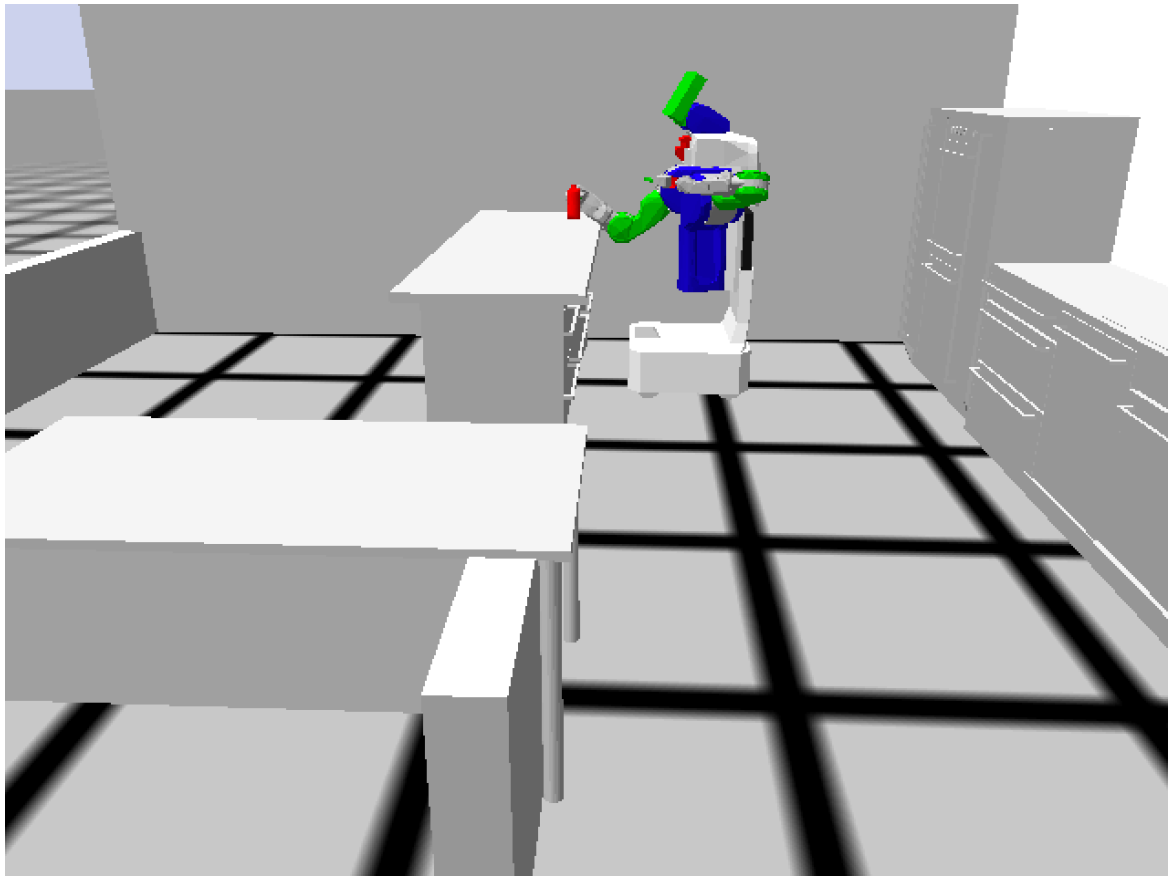
Perform action
pick-up: lifting



Perform action
move near the counter



Perform action
place: reaching

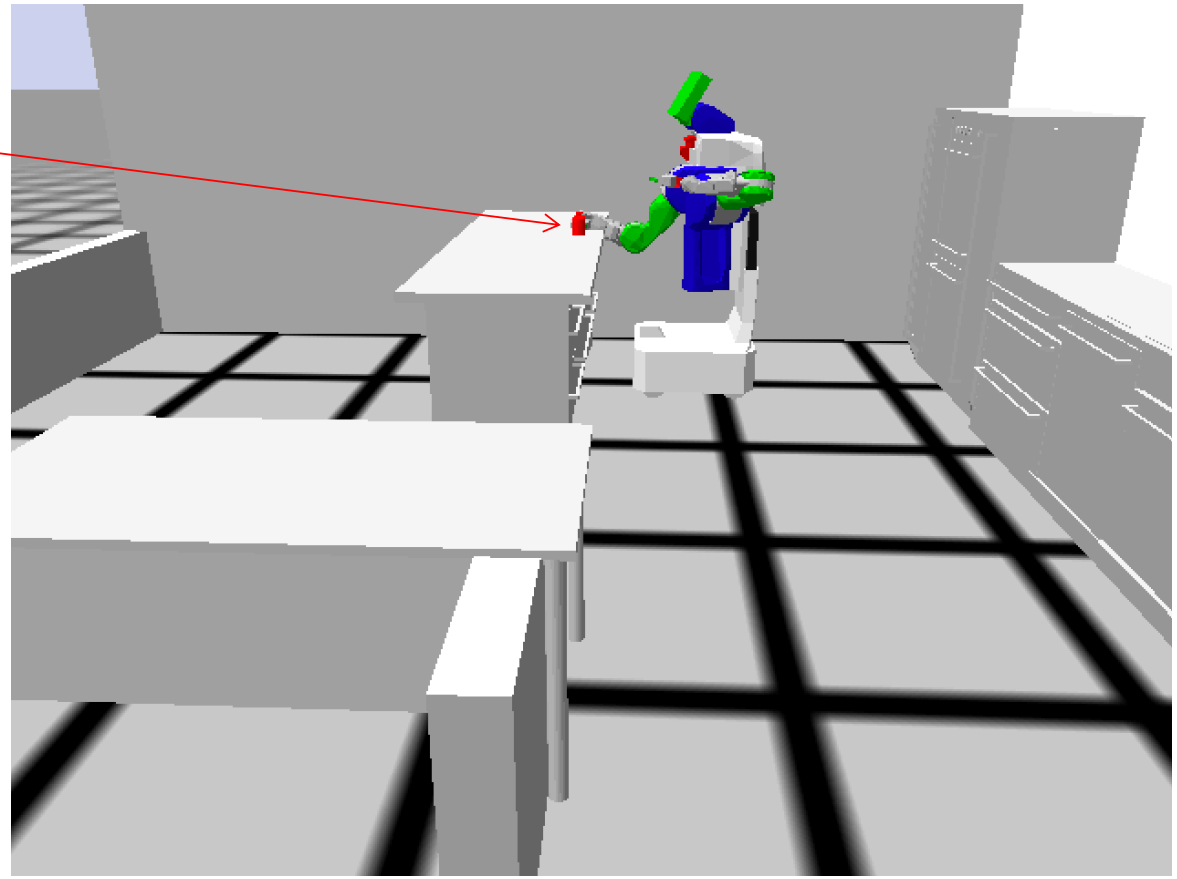


Perform action
place: putting

What's going on here?

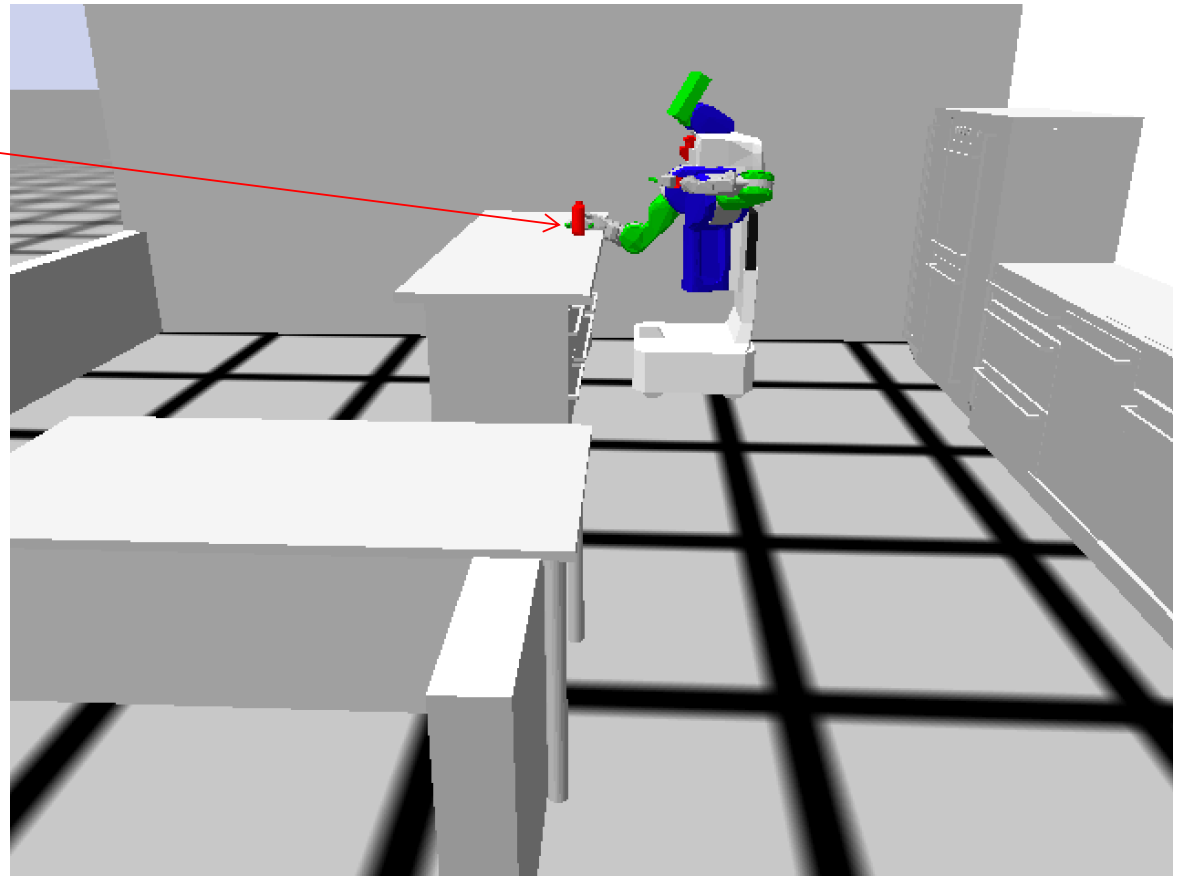
The bottle has been pushed through the counter-top.

The value of ?drop-pose, based on global variable *final-object-destination*, specifies a height of 0.90 m but the counter is higher than this, so the robot pushes the bottle through the surface (it's a simulation after all)

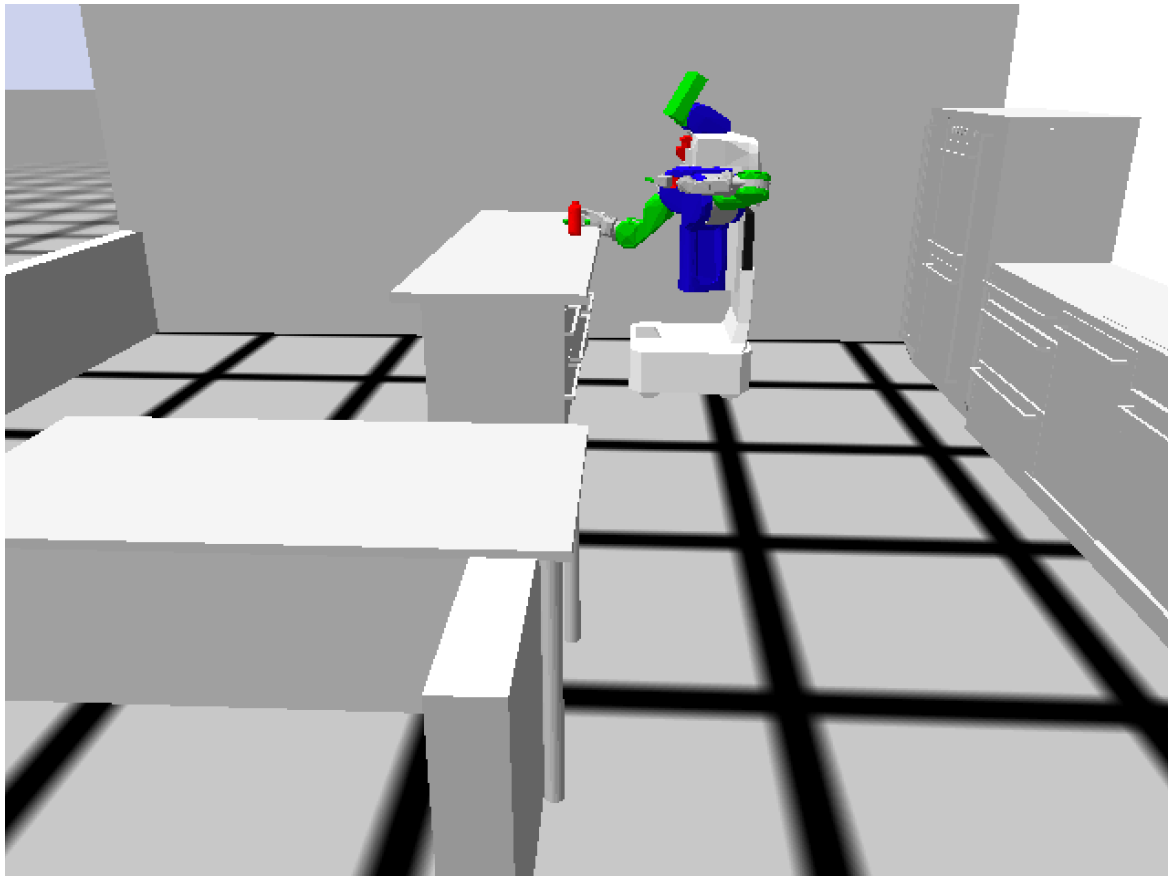


Perform action
place: opening gripper

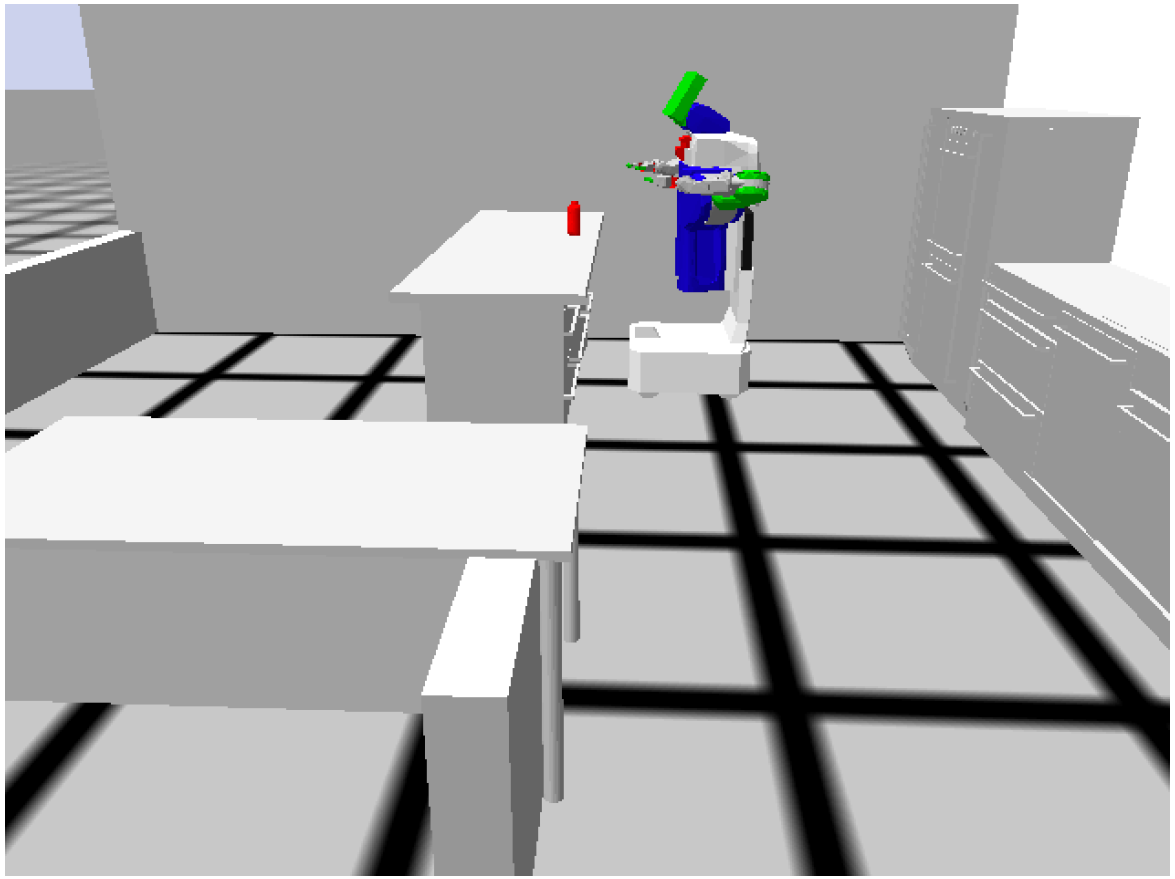
And then the physic engine takes over and pushes the bottle back up so that it is supported by, but not embedded in, the counter-top



Perform action
place: retracting



Perform action
place: retracting



Simple Fetch and Place Plan

```
PP-TUT> (move-bottle '((-1.6 -0.9 0.82) (0 0 0 1)))
[(PICK-PLACE PICK-UP) INFO] 1620307408.425: Opening gripper
[(PICK-PLACE PICK-UP) INFO] 1620307408.426: Reaching
[(PICK-PLACE PICK-UP) INFO] 1620307408.752: Gripping
[(PICK-PLACE PICK-UP) INFO] 1620307408.832: Assert grasp into knowledge base
[(PICK-PLACE PICK-UP) INFO] 1620307408.833: Lifting
[(PICK-PLACE PLACE) INFO] 1620307409.221: Reaching
[(PICK-PLACE PLACE) INFO] 1620307409.408: Putting
[(PICK-PLACE PLACE) INFO] 1620307409.513: Opening gripper
[(PICK-PLACE PLACE) INFO] 1620307409.559: Retract grasp in knowledge base
[(PICK-PLACE PLACE) INFO] 1620307409.586: Retracting
NIL
PP-TUT>
```

Grasp Poses

- A bottle has four pre-defined grasp poses associated with it

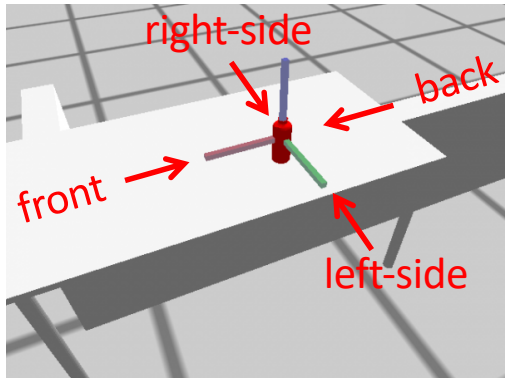
- Each defined with respect to the frame embedded in the bottle

  front — grasp frame is aligned with the positive X axis, directed towards the frame origin

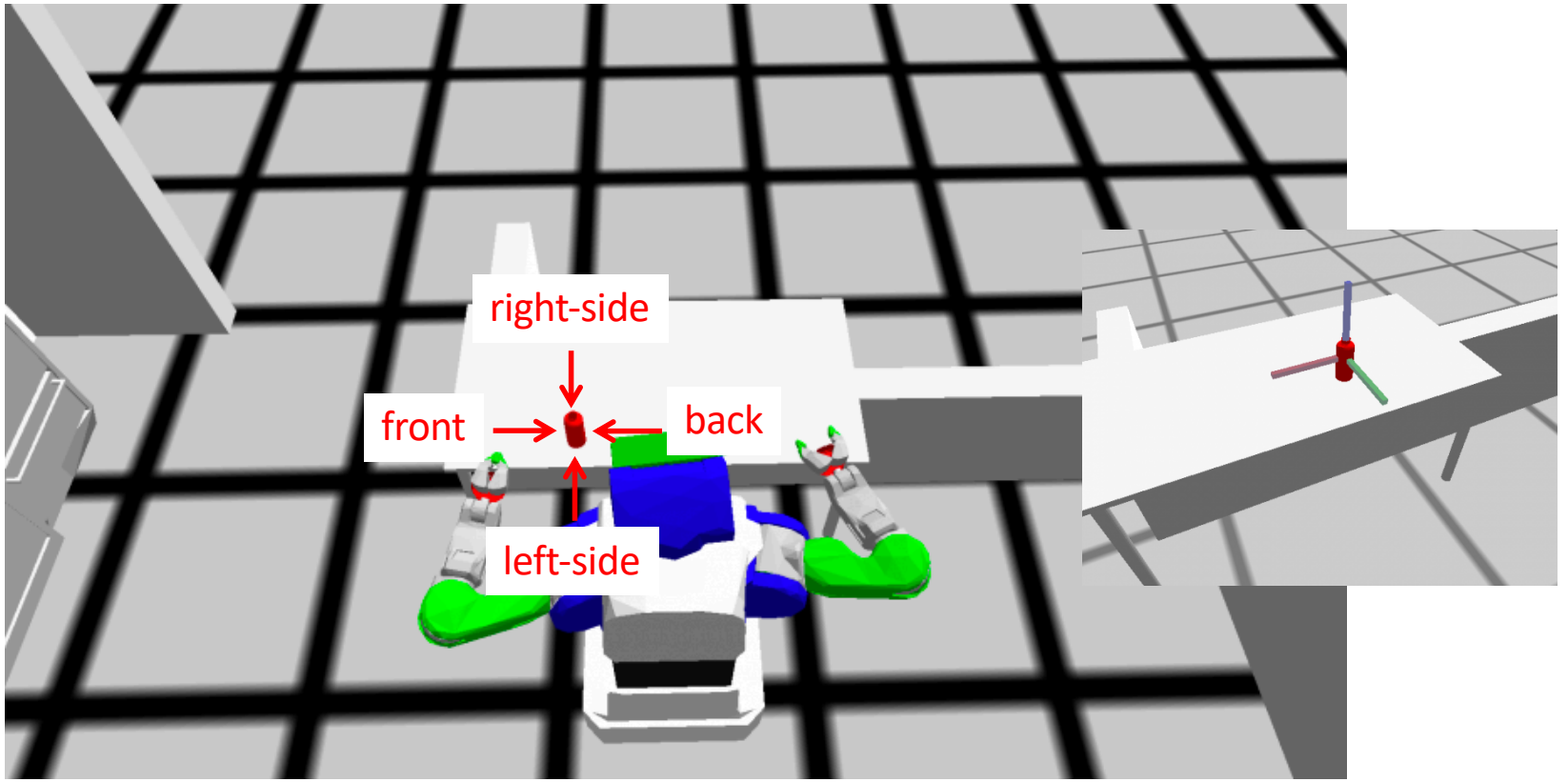
  back — grasp frame is aligned with the negative X axis, directed towards the frame origin

  left-side — grasp frame is aligned with the positive Y axis, directed towards the frame origin

  right-side — grasp frame is aligned with the negative Y axis, directed towards the frame origin

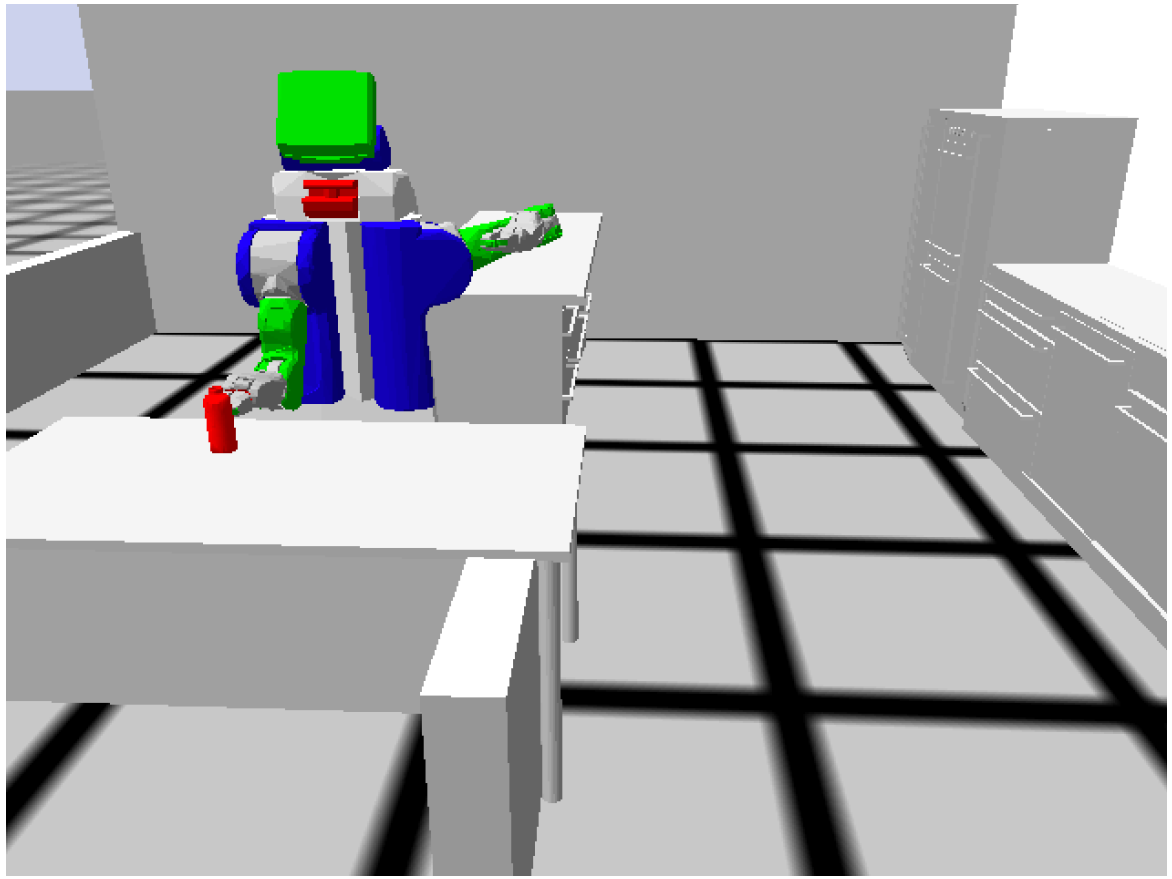


Grasp Poses



Grasp Poses

This is grasping from the left-side



Grasp Poses

We will explain how to define new grasp poses and approach poses in CR11-04

Recommended Reading

CRAM zero prerequisites demo tutorial: simple fetch and place

`http://cram-system.org/tutorials/demo/fetch_and_place`

Implementation

Follow these instructions

"Zero Prerequisites Demo Tutorial: Simple Fetch and Place"

http://www.vernon.eu/wiki/Zero_Prerequisites_Demo_Tutorial:_Simple_Fetch_and_Place

to implement the pick-and-place example

Page Discussion Read Edit View history

Main page
Recent changes
Help
My website

Zero Prerequisites Demo Tutorial: Simple Fetch and Place

This page provides a consolidated version of the code required for the Zero prerequisites demo tutorial: Simple fetch and place. You normally do this tutorial in an interactive manner, leading to the creation of the code for the move-bottle function that is pasted into the `pick-and-place.lisp` file for the first example. The second and third examples on failure handling modify this code.

Here, we provide the code for three versions of `move-bottle`, one for each example: `move-bottle1`, `move-bottle2`, and `move-bottle3`. This allows you to add code to the pick-and-place.lisp just once and so that you can simply do the tutorial by invoking the example commands, i.e. by evaluating the three example forms in REPL, each one exemplifying one specific aspect of the plan.

We also include a fourth version, `move-botte4`, which covers the example of defining a new grasp, directly after Exercise 3.

For convenience, we also include four dummy functions to use when doing exercises 1 - 4.

Note that here we don't cover the material in the first two sections of the tutorial, i.e. "Setting Up" and "Understanding the Basics". You need to go through these yourself. Here, we cover the material in the section "Simple Fetch and Place".

Contents [hide]

1 Update pick-and-place.lisp
2 Do the Tutorial
  2.1 Bullet World Initialization
  2.2 Environment Setup
  2.3 REPL Setup
  2.4 Bullet World Initialization
  2.5 Simple Fetch and Place Plan
    2.5.1 Simple Fetch and Place
    2.5.2 Recovery from Failures
    2.5.3 Handling More Failures
    2.5.4 Defining a New Grasp
  2.6 Exercises
    2.6.1 Exercise 1
    2.6.2 Exercise 2
    2.6.3 Exercise 3
    2.6.4 Exercise 4

Update `pick-and-place.lisp` [edit]

First, let's copy the example code.

Move into the src directory:

http://www.vernon.eu/wiki/Zero_Prerequisites_Demo_Tutorial:_Simple_Fetch_and_Place