# Introduction to Cognitive Robotics

## Module 11: Cognition-enabled Robot Manipulation with CRAM

## Lecture 2: Error handling and recovery looking in different places

David Vernon
Carnegie Mellon University Africa

www.vernon.eu

## 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

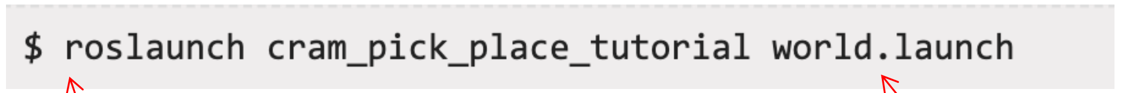
- Pick-and-place
    - Pick an object from one position
    - Place it in another position in the world.
- Error handling
- Recovery behaviors

To look in different places for the object

# 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)

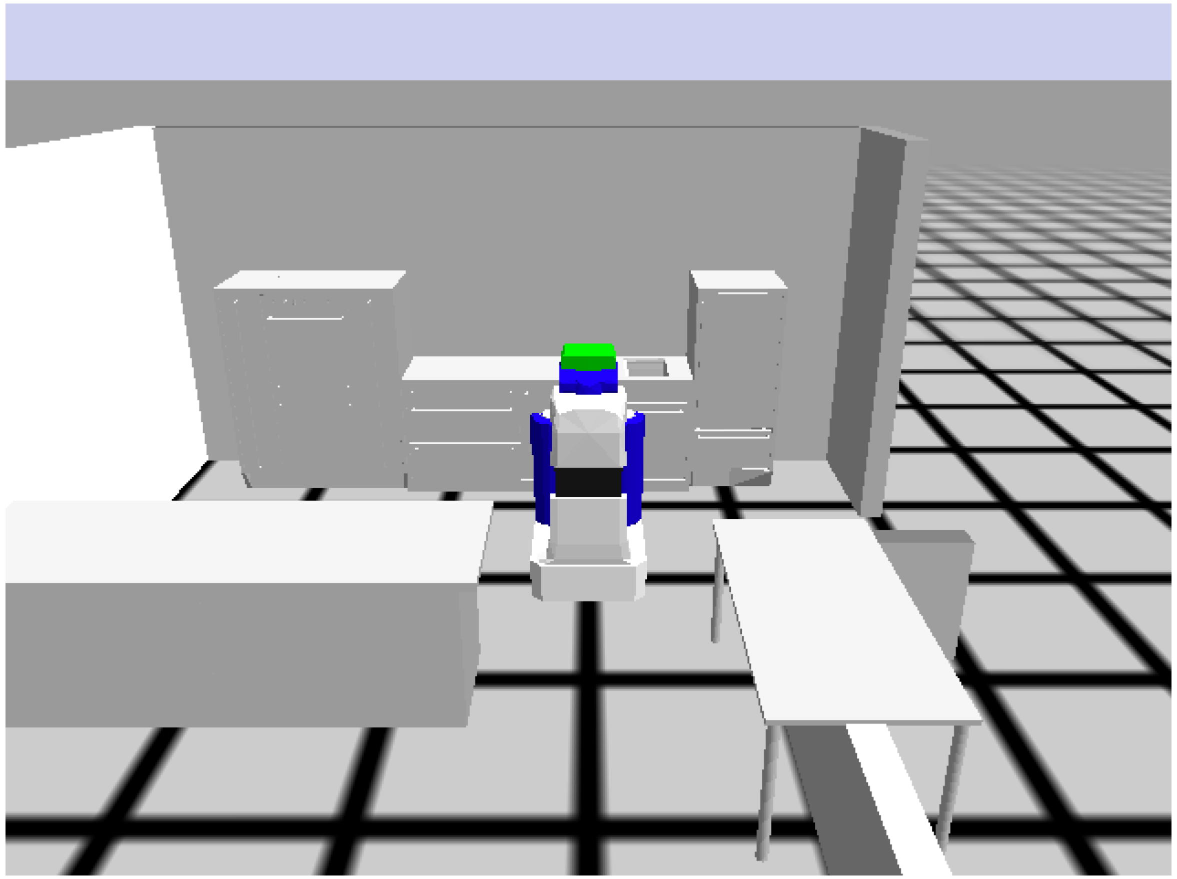
```
CL-USER> (ros-load:load-system "cram_pick_place_tutorial" :cram-pick-place-tutorial)
CL-USER> (in-package :cram-pick-place-tutorial)
```

- Start everything up

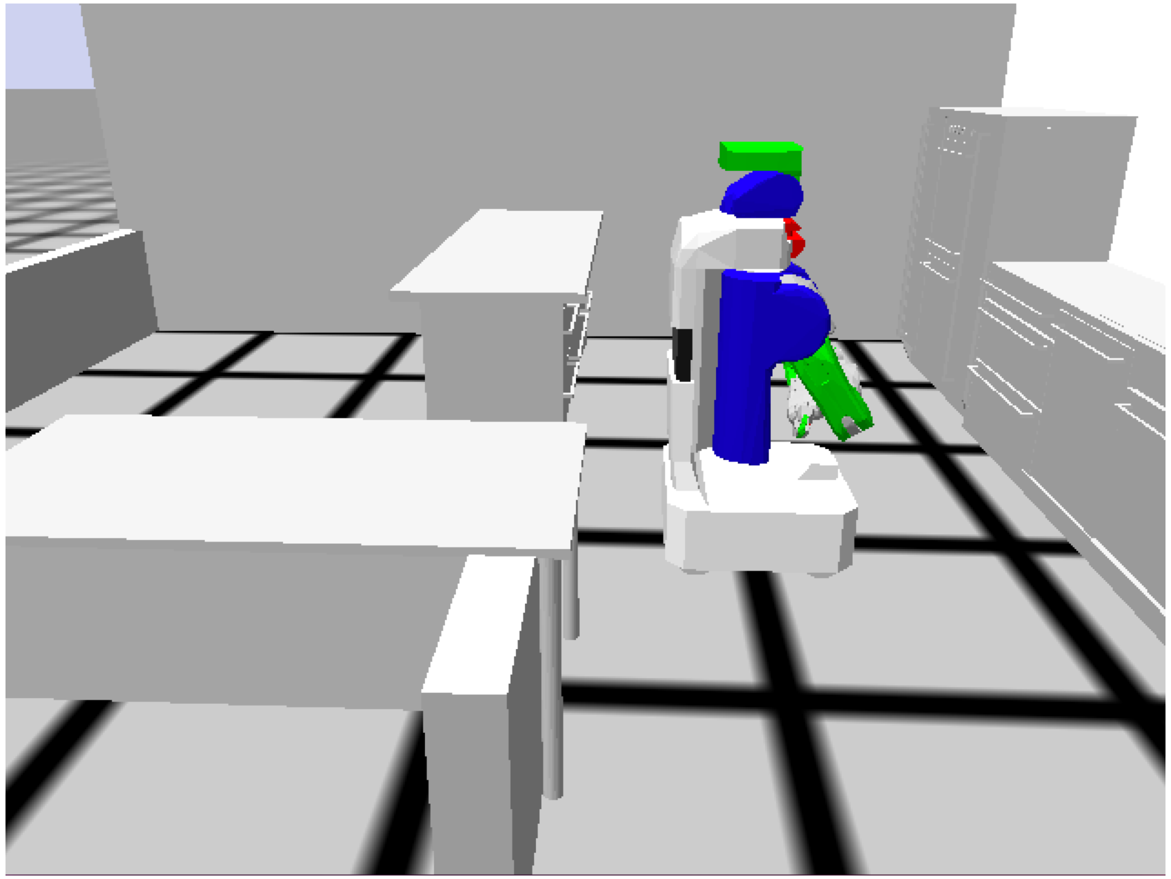
```
PP-TUT> (roslisp-utilities:startup-ros)
```

  - 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

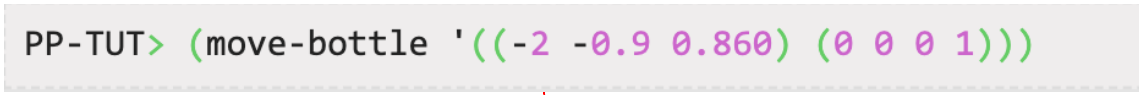


## Bullet Real-Time Physics Simulation



## Simple Fetch and Place Plan

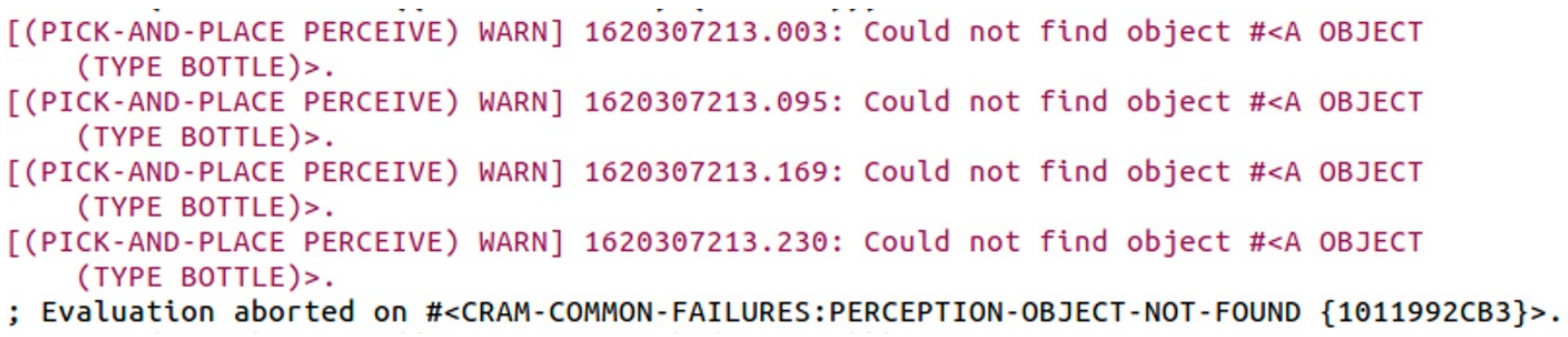
- The previous plan works because we instructed the robot to look in the right place for the bottle
- What happens if the bottle is placed somewhere else …



```
PP-TUT> (move-bottle '((-2 -0.9 0.860) (0 0 0 1)))
```

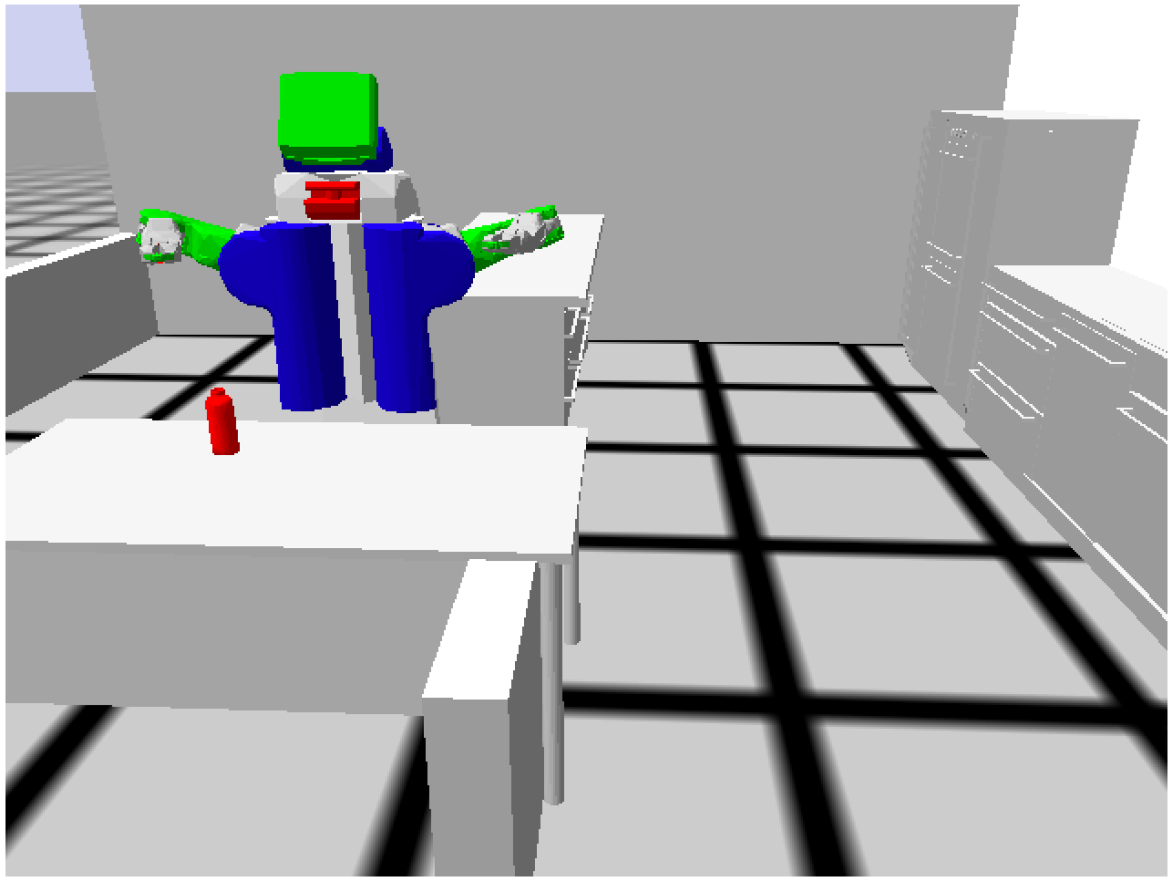
Was -1.6 in the previous example)

- We get an error:

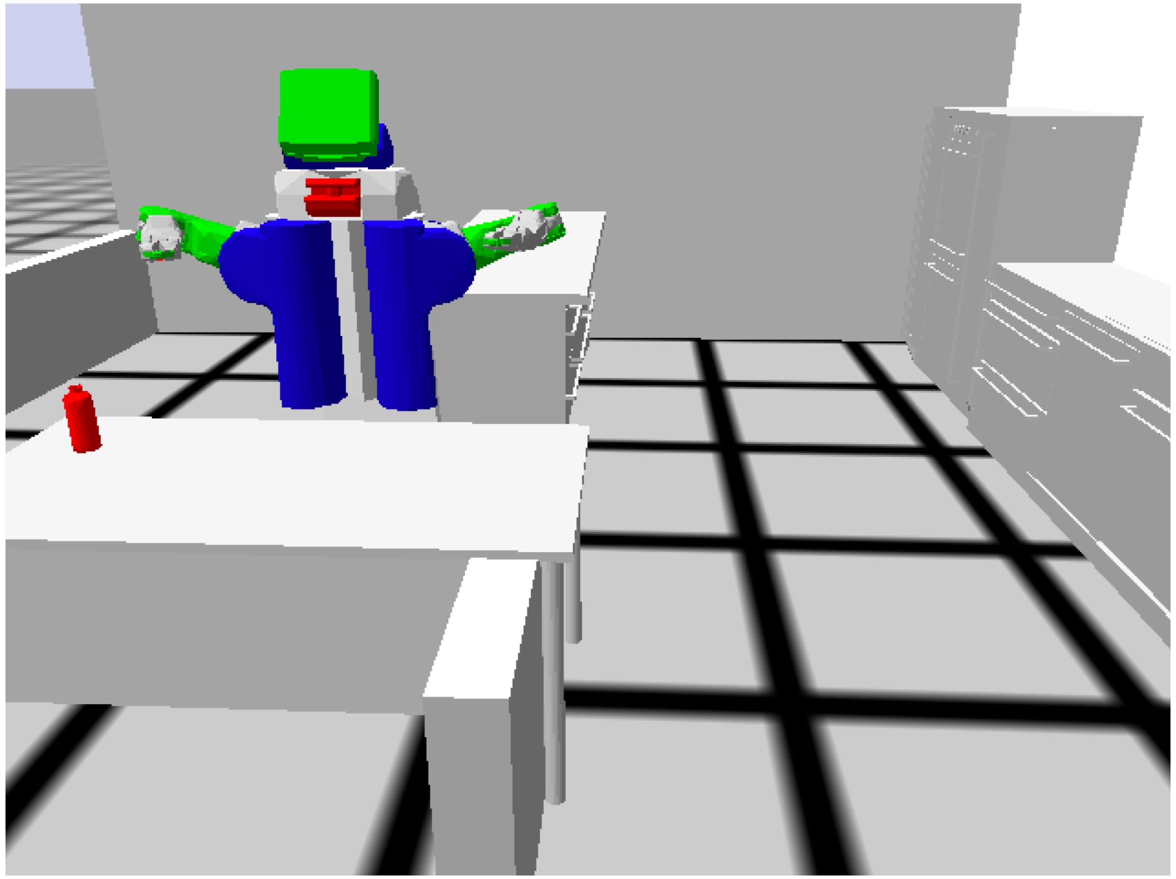


```
[(PICK-AND-PLACE PERCEIVE) WARN] 1620307213.003: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.
[(PICK-AND-PLACE PERCEIVE) WARN] 1620307213.095: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.
[(PICK-AND-PLACE PERCEIVE) WARN] 1620307213.169: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.
[(PICK-AND-PLACE PERCEIVE) WARN] 1620307213.230: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.
; Evaluation aborted on #<CRAM-COMMON-FAILURES:PERCEPTION-OBJECT-NOT-FOUND {1011992CB3}>.
```

## This bottle position is in the robot’s view

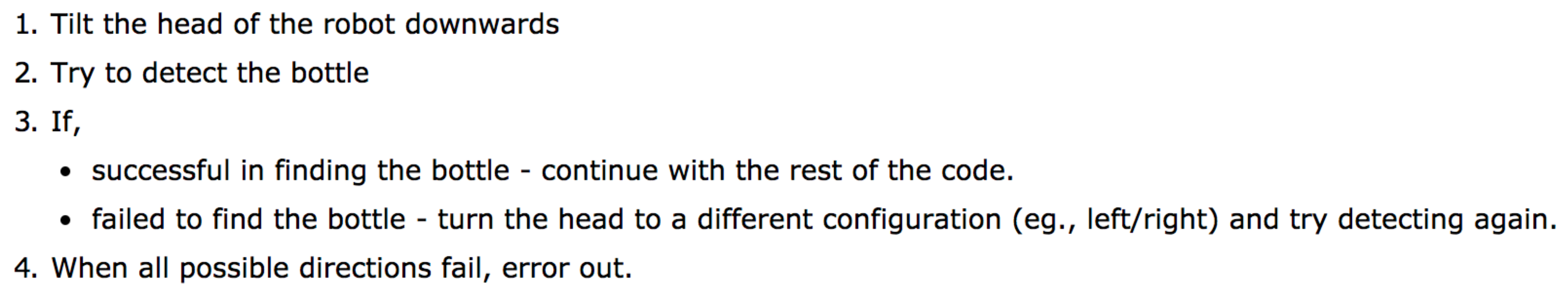


## This bottle position is not in the robot’s view



# Simple Fetch and Place Plan

We need a different strategy, i.e. a plan that can handle failures



1. Tilt the head of the robot downwards
2. Try to detect the bottle
3. If,
    - successful in finding the bottle - continue with the rest of the code.
    - failed to find the bottle - turn the head to a different configuration (eg., left/right) and try detecting again.
4. When all possible directions fail, error out.

## Simple Fetch and Place Plan

In general:

1. Perform some action

2. When a failure occurs
    - Execute a recovery strategy
    - Retry performing the action

```
(handle-failure <the error type that needs to be handled>

   (<all the actions to be performed under normal execution>)

   (<actions to be performed if there is an error of the declared type.
    Call retry to reevaluate the body if necessary>))
```

## Simple Fetch and Place Plan

Let’s define two additional directions in which to look

```
(defparameter *left-downward-look-coordinate*
  (make-pose "base_footprint" '((0.65335 0.76 0.758) (0 0 0 1))))

(defparameter *right-downward-look-coordinate*
  (make-pose "base_footprint" '((0.65335 -0.76 0.758) (0 0 0 1))))
```

## Simple Fetch and Place Plan

Recall that the original plan was

```
    Look towards the object

    Detect the object that has to be picked
```

With the first bottle pose, it didn't fail

- because we created the bottle in the same place
- as we defined the point to look at …

With the second bottle pose, it did fail

- because we created the bottle in a different place
- to the point to look at …

## Look towards the object

```
(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 point to look at

Perform an action to look at a location

Use the point to look at

```lisp
(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))))
```

## Detect the object that has to be picked

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)

## Simple Fetch and Place Plan

The new plan combines the looking and detecting:

```
Create a list of directions in which to look: downward, left, right
Set initial looking direction and remove it from the list

With failure handling enabled
   Perform an action to detect an object (but with failure handling)

   Execute this code when an object-not-found failure occurs:
      While there is still another viewing direction in the list
         Issue a failure warning
         Perform an action to look forward
         Set the looking direction to the first in the list
         Remove this direction from the list
         Perform an action to look in this direction
         Retry the code that failed
      Otherwise FAIL
```

Return the object detected

## Simple Fetch and Place Plan

We define a function that finds an object

- Combining the looking  and detecting
- Incorporating error handling

```
(defun find-object (?object-type)
  (let* ((possible-look-directions `(,*downward-look-coordinate*
                                      ,*left-downward-look-coordinate*
                                      ,*right-downward-look-coordinate*))
         (?looking-direction (first possible-look-directions)))
    (setf possible-look-directions (rest possible-look-directions))
    ;; Look towards the first direction
    (perform (an action
                 (type looking)
                 (target (a location
                            (pose ?looking-direction)))))

    ;; perception-object-not-found is the error that we get when the robot cannot find the object.
    ;; Now we're wrapping it in a failure handling clause to handle it
    (handle-failure perception-object-not-found
        ;; Try the action
        ((perform (an action
                      (type detecting)
                      (object (an object
                                  (type ?object-type))))))

      ;; If the action fails, try the following:
      ;; try different look directions until there is none left.
      (when possible-look-directions
        (print "Perception error happened! Turning head.")
        ;; Resetting the head to look forward before turning again
        (perform (an action
                     (type looking)
                     (direction forward)))
        (setf ?looking-direction (first possible-look-directions))
        (setf possible-look-directions (rest possible-look-directions))
        (perform (an action
                     (type looking)
                     (target (a location
                                (pose ?looking-direction)))))
        ;; This statement retries the action again
        (cpl:retry))
      ;; If everything else fails, error out
      ;; Reset the neck before erroring out
      (perform (an action
                   (type looking)
                   (direction forward)))
      (cpl:fail 'object-nowhere-to-be-found))))
```

New function to find the object

Create a list of directions in which to look

The initial direction is the first in the list

Remove the initial direction from the list by setting the list to everything after the initial direction

```
(defun find-object (?object-type)
  (let* ((possible-look-directions `(,*downward-look-coordinate*
                                      ,*left-downward-look-coordinate*
                                      ,*right-downward-look-coordinate*))
         (?looking-direction (first possible-look-directions)))
    (setf possible-look-directions (rest possible-look-directions))
    ;; Look towards the first direction
    (perform (an action
                 (type looking)
                 (target (a location
                            (pose ?looking-direction)))))

    ;; perception-object-not-found is the error that we get when the robot cannot find the object.
    ;; Now we're wrapping it in a failure handling clause to handle it
    (handle-failure perception-object-not-found
        ;; Try the action
        ((perform (an action
                      (type detecting)
                      (object (an object
                                  (type ?object-type))))))

      ;; If the action fails, try the following:
      ;; try different look directions until there is none left.
      (when possible-look-directions
        (print "Perception error happened! Turning head.")
        ;; Resetting the head to look forward before turning again
        (perform (an action
                     (type looking)
                     (direction forward)))
        (setf ?looking-direction (first possible-look-directions))
        (setf possible-look-directions (rest possible-look-directions))
        (perform (an action
                     (type looking)
                     (target (a location
                                (pose ?looking-direction)))))
        ;; This statement retries the action again
        (cpl:retry))
      ;; If everything else fails, error out
      ;; Reset the neck before erroring out
      (perform (an action
                   (type looking)
                   (direction forward)))
      (cpl:fail 'object-nowhere-to-be-found))))
```

Perform an action to look at a location

Use the initial point to  look at

```lisp
(defun find-object (?object-type)
  (let* ((possible-look-directions `(,*downward-look-coordinate*
                                     ,*left-downward-look-coordinate*
                                     ,*right-downward-look-coordinate*))
         (?looking-direction (first possible-look-directions)))
    (setf possible-look-directions (rest possible-look-directions))
    ;; Look towards the first direction
    (perform (an action
                 (type looking)
                 (target (a location
                            (pose ?looking-direction)))))

    ;; perception-object-not-found is the error that we get when the robot cannot find the object.
    ;; Now we're wrapping it in a failure handling clause to handle it
    (handle-failure perception-object-not-found
        ;; Try the action
        ((perform (an action
                      (type detecting)
                      (object (an object
                                  (type ?object-type))))))

      ;; If the action fails, try the following:
      ;; try different look directions until there is none left.
      (when possible-look-directions
        (print "Perception error happened! Turning head.")
        ;; Resetting the head to look forward before turning again
        (perform (an action
                     (type looking)
                     (direction forward)))
        (setf ?looking-direction (first possible-look-directions))
        (setf possible-look-directions (rest possible-look-directions))
        (perform (an action
                     (type looking)
                     (target (a location
                                (pose ?looking-direction)))))
        ;; This statement retries the action again
        (cpl:retry))
      ;; If everything else fails, error out
      ;; Reset the neck before erroring out
      (perform (an action
                   (type looking)
                   (direction forward)))
      (cpl:fail 'object-nowhere-to-be-found))))
```

With failure handling enabled

Execute this code to detect the object

```
(defun find-object (?object-type)
  (let* ((possible-look-directions `(,*downward-look-coordinate*
                                      ,*left-downward-look-coordinate*
                                      ,*right-downward-look-coordinate*))
         (?looking-direction (first possible-look-directions)))
    (setf possible-look-directions (rest possible-look-directions))
    ;; Look towards the first direction
    (perform (an action
                 (type looking)
                 (target (a location
                            (pose ?looking-direction)))))

    ;; perception-object-not-found is the error that we get when the robot cannot find the object.
    ;; Now we're wrapping it in a failure handling clause to handle it
    (handle-failure perception-object-not-found
        ;; Try the action
        ((perform (an action
                      (type detecting)
                      (object (an object
                                  (type ?object-type))))))

      ;; If the action fails, try the following:
      ;; try different look directions until there is none left.
      (when possible-look-directions
        (print "Perception error happened! Turning head.")
        ;; Resetting the head to look forward before turning again
        (perform (an action
                     (type looking)
                     (direction forward)))
        (setf ?looking-direction (first possible-look-directions))
        (setf possible-look-directions (rest possible-look-directions))
        (perform (an action
                     (type looking)
                     (target (a location
                                (pose ?looking-direction)))))
        ;; This statement retries the action again
        (cpl:retry))
      ;; If everything else fails, error out
      ;; Reset the neck before erroring out
      (perform (an action
                   (type looking)
                   (direction forward)))
      (cpl:fail 'object-nowhere-to-be-found))))
```

Execute this code in the case of a failure

```lisp
(defun find-object (?object-type)
  (let* ((possible-look-directions `(,*downward-look-coordinate*
                                     ,*left-downward-look-coordinate*
                                     ,*right-downward-look-coordinate*))
         (?looking-direction (first possible-look-directions)))
    (setf possible-look-directions (rest possible-look-directions))
    ;; Look towards the first direction
    (perform (an action
                 (type looking)
                 (target (a location
                            (pose ?looking-direction)))))

    ;; perception-object-not-found is the error that we get when the robot cannot find the object.
    ;; Now we're wrapping it in a failure handling clause to handle it
    (handle-failure perception-object-not-found
        ;; Try the action
        ((perform (an action
                      (type detecting)
                      (object (an object
                                  (type ?object-type))))))

      ;; If the action fails, try the following:
      ;; try different look directions until there is none left.
      (when possible-look-directions
        (print "Perception error happened! Turning head.")
        ;; Resetting the head to look forward before turning again
        (perform (an action
                     (type looking)
                     (direction forward)))
        (setf ?looking-direction (first possible-look-directions))
        (setf possible-look-directions (rest possible-look-directions))
        (perform (an action
                     (type looking)
                     (target (a location
                                (pose ?looking-direction)))))
        ;; This statement retries the action again
        (cpl:retry))
      ;; If everything else fails, error out
      ;; Reset the neck before erroring out
      (perform (an action
                   (type looking)
                   (direction forward)))
      (cpl:fail 'object-nowhere-to-be-found))))
```

If there is still another viewing direction in the list

Issue a failure message

Perform an action to look forward

Set looking direction to first in the list

Remove this direction from the list

```lisp
(defun find-object (?object-type)
  (let* ((possible-look-directions `(,*downward-look-coordinate*
                                     ,*left-downward-look-coordinate*
                                     ,*right-downward-look-coordinate*))
         (?looking-direction (first possible-look-directions)))
    (setf possible-look-directions (rest possible-look-directions))
    ;; Look towards the first direction
    (perform (an action
                 (type looking)
                 (target (a location
                            (pose ?looking-direction)))))

    ;; perception-object-not-found is the error that we get when the robot cannot find the object.
    ;; Now we're wrapping it in a failure handling clause to handle it
    (handle-failure perception-object-not-found
        ;; Try the action
        ((perform (an action
                      (type detecting)
                      (object (an object
                                  (type ?object-type))))))

      ;; If the action fails, try the following:
      ;; try different look directions until there is none left.
      (when possible-look-directions
        (print "Perception error happened! Turning head.")
        ;; Resetting the head to look forward before turning again
        (perform (an action
                     (type looking)
                     (direction forward)))
        (setf ?looking-direction (first possible-look-directions))
        (setf possible-look-directions (rest possible-look-directions))
        (perform (an action
                     (type looking)
                     (target (a location
                                (pose ?looking-direction)))))
        ;; This statement retries the action again
        (cpl:retry))
      ;; If everything else fails, error out
      ;; Reset the neck before erroring out
      (perform (an action
                   (type looking)
                   (direction forward)))
      (cpl:fail 'object-nowhere-to-be-found))))
```

Perform an action to look in this direction

```lisp
(defun find-object (?object-type)
  (let* ((possible-look-directions `(,*downward-look-coordinate*
                                     ,*left-downward-look-coordinate*
                                     ,*right-downward-look-coordinate*))
         (?looking-direction (first possible-look-directions)))
    (setf possible-look-directions (rest possible-look-directions))
    ;; Look towards the first direction
    (perform (an action
                 (type looking)
                 (target (a location
                            (pose ?looking-direction)))))

    ;; perception-object-not-found is the error that we get when the robot cannot find the object.
    ;; Now we're wrapping it in a failure handling clause to handle it
    (handle-failure perception-object-not-found
        ;; Try the action
        ((perform (an action
                      (type detecting)
                      (object (an object
                                  (type ?object-type))))))

      ;; If the action fails, try the following:
      ;; try different look directions until there is none left.
      (when possible-look-directions
        (print "Perception error happened! Turning head.")
        ;; Resetting the head to look forward before turning again
        (perform (an action
                     (type looking)
                     (direction forward)))
        (setf ?looking-direction (first possible-look-directions))
        (setf possible-look-directions (rest possible-look-directions))
        (perform (an action
                     (type looking)
                     (target (a location
                                (pose ?looking-direction)))))
        ;; This statement retries the action again
        (cpl:retry))
      ;; If everything else fails, error out
      ;; Reset the neck before erroring out
      (perform (an action
                   (type looking)
                   (direction forward)))
      (cpl:fail 'object-nowhere-to-be-found))))
```

Retry the code that failed

```
(defun find-object (?object-type)
  (let* ((possible-look-directions `(,*downward-look-coordinate*
                                     ,*left-downward-look-coordinate*
                                     ,*right-downward-look-coordinate*))
         (?looking-direction (first possible-look-directions)))
    (setf possible-look-directions (rest possible-look-directions))
    ;; Look towards the first direction
    (perform (an action
                 (type looking)
                 (target (a location
                            (pose ?looking-direction)))))

    ;; perception-object-not-found is the error that we get when the robot cannot find the object.
    ;; Now we're wrapping it in a failure handling clause to handle it
    (handle-failure perception-object-not-found
        ;; Try the action
        ((perform (an action
                      (type detecting)
                      (object (an object
                                  (type ?object-type))))))

      ;; If the action fails, try the following:
      ;; try different look directions until there is none left.
      (when possible-look-directions
        (print "Perception error happened! Turning head.")
        ;; Resetting the head to look forward before turning again
        (perform (an action
                     (type looking)
                     (direction forward)))
        (setf ?looking-direction (first possible-look-directions))
        (setf possible-look-directions (rest possible-look-directions))
        (perform (an action
                     (type looking)
                     (target (a location
                                (pose ?looking-direction)))))
        ;; This statement retries the action again
        (cpl:retry))
      ;; If everything else fails, error out
      ;; Reset the neck before erroring out
      (perform (an action
                   (type looking)
                   (direction forward)))
      (cpl:fail 'object-nowhere-to-be-found))))
```

Perform an action to look forward

```
(defun find-object (?object-type)
  (let* ((possible-look-directions `(,*downward-look-coordinate*
                                     ,*left-downward-look-coordinate*
                                     ,*right-downward-look-coordinate*))
         (?looking-direction (first possible-look-directions)))
    (setf possible-look-directions (rest possible-look-directions))
    ;; Look towards the first direction
    (perform (an action
                 (type looking)
                 (target (a location
                            (pose ?looking-direction)))))

    ;; perception-object-not-found is the error that we get when the robot cannot find the object.
    ;; Now we're wrapping it in a failure handling clause to handle it
    (handle-failure perception-object-not-found
        ;; Try the action
        ((perform (an action
                      (type detecting)
                      (object (an object
                                  (type ?object-type))))))

      ;; If the action fails, try the following:
      ;; try different look directions until there is none left.
      (when possible-look-directions
        (print "Perception error happened! Turning head.")
        ;; Resetting the head to look forward before turning again
        (perform (an action
                     (type looking)
                     (direction forward)))
        (setf ?looking-direction (first possible-look-directions))
        (setf possible-look-directions (rest possible-look-directions))
        (perform (an action
                     (type looking)
                     (target (a location
                                (pose ?looking-direction)))))
        ;; This statement retries the action again
        (cpl:retry))
      ;; If everything else fails, error out
      ;; Reset the neck before erroring out
      (perform (an action
                   (type looking)
                   (direction forward)))
      (cpl:fail 'object-nowhere-to-be-found))))
```

And FAIL (no directions left in list)

## Simple Fetch and Place Plan

Finally, we use this function in a revised `move-bottle`

It is shorter than the previous version

because we moved the looking and detecting parts

into the function to find the object (with error handling)

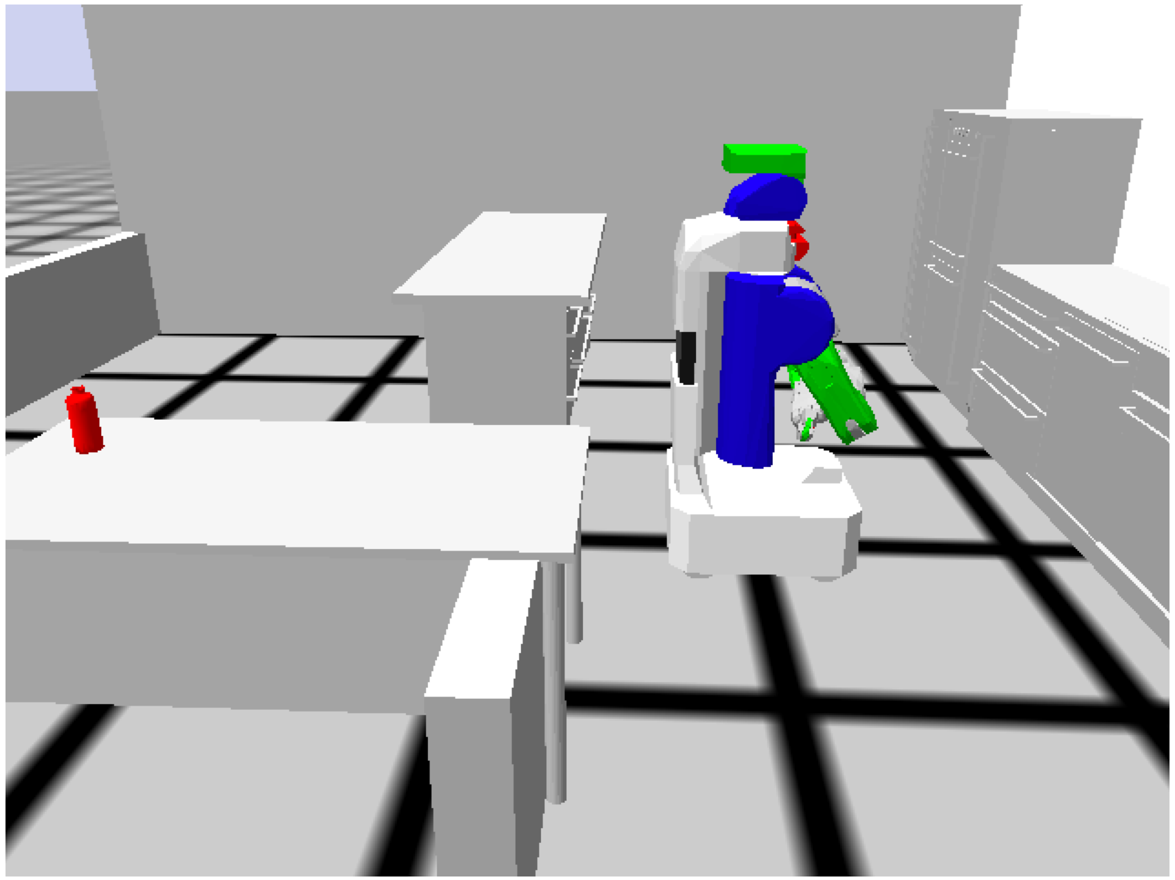
```lisp
(defun move-bottle (bottle-spawn-pose)
  (spawn-object bottle-spawn-pose)
  (with-simulated-robot
    (let ((?navigation-goal *base-pose-near-table*))
      (cpl:par
        ;; Moving the robot near the table.
        (perform (an action
                     (type going)
                     (target (a location
                                (pose ?navigation-goal)))))
        (perform (a motion
                    (type moving-torso)
                    (joint-angle 0.3)))
        (park-arms)))
    ;; Find and detect the bottle on the table. We use the new method here
    (let ((?perceived-bottle (find-object :bottle))
          (?grasping-arm :right))
      (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 object 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))))
```

Same as before

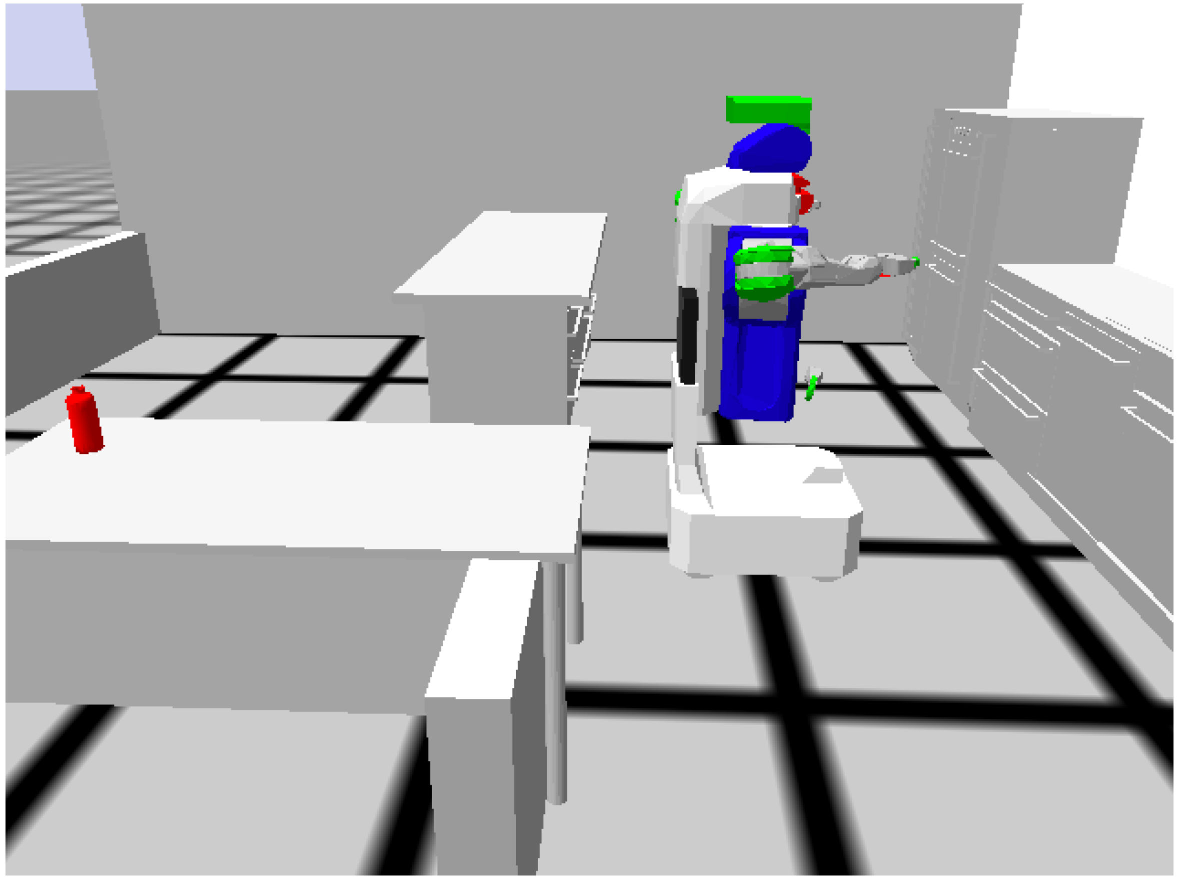
Call the function to find the object, returning the object

Same as before

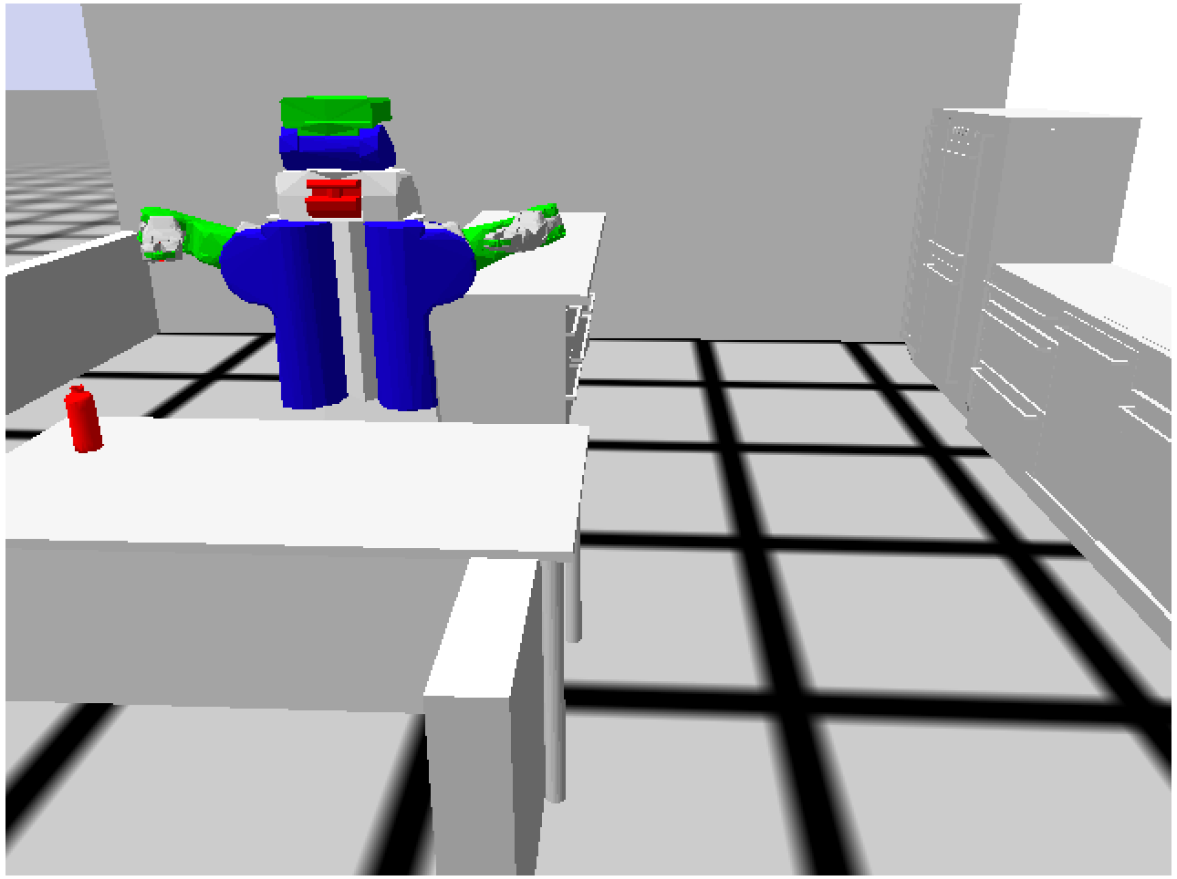
## Spawn the bottle



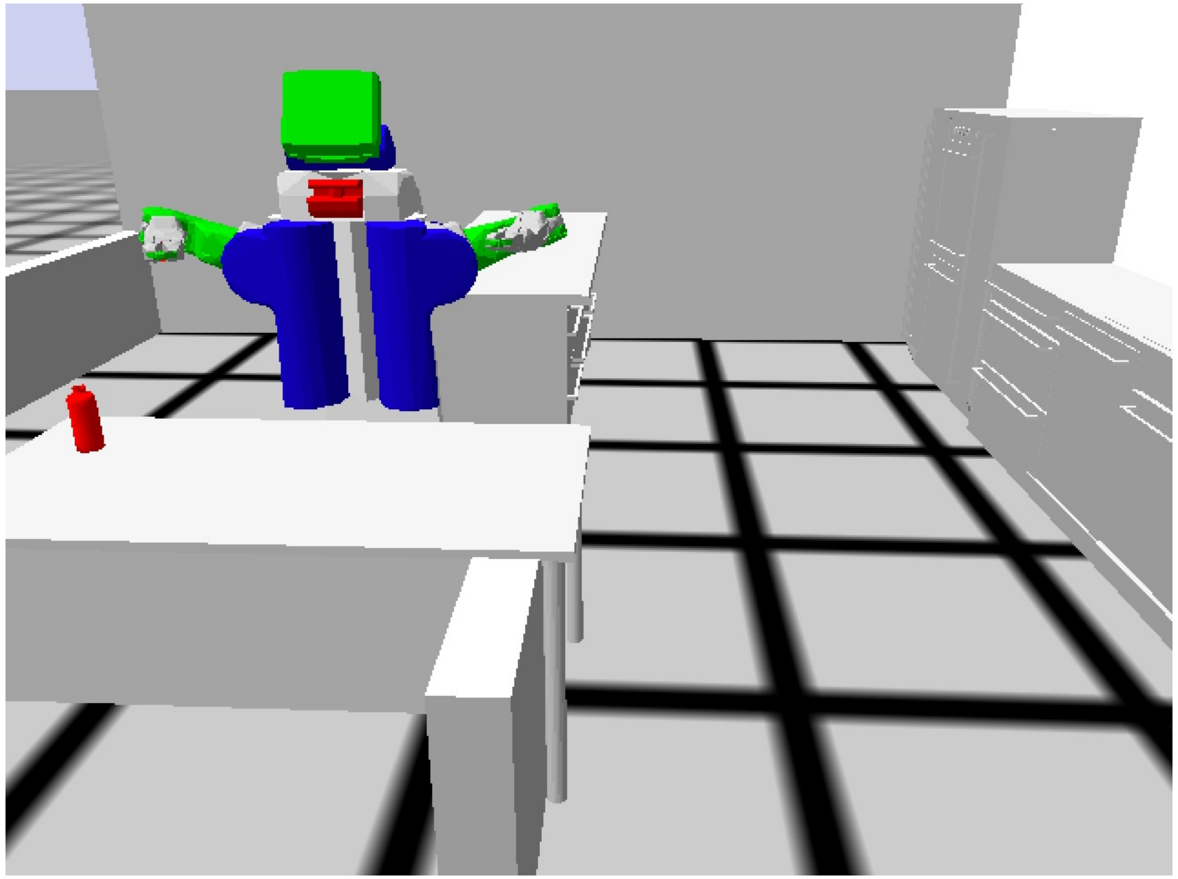
## Perform motion
## adjust torso



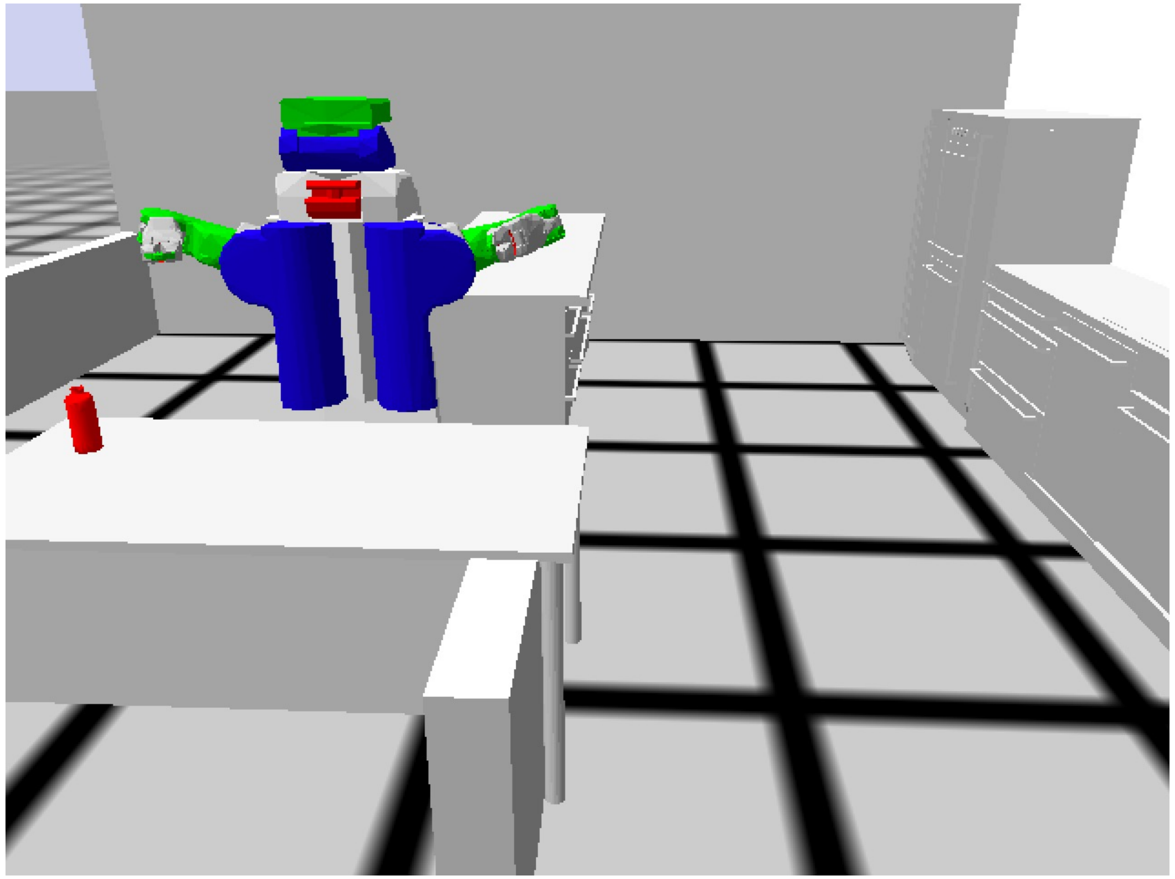
## Perform action
## move near the table



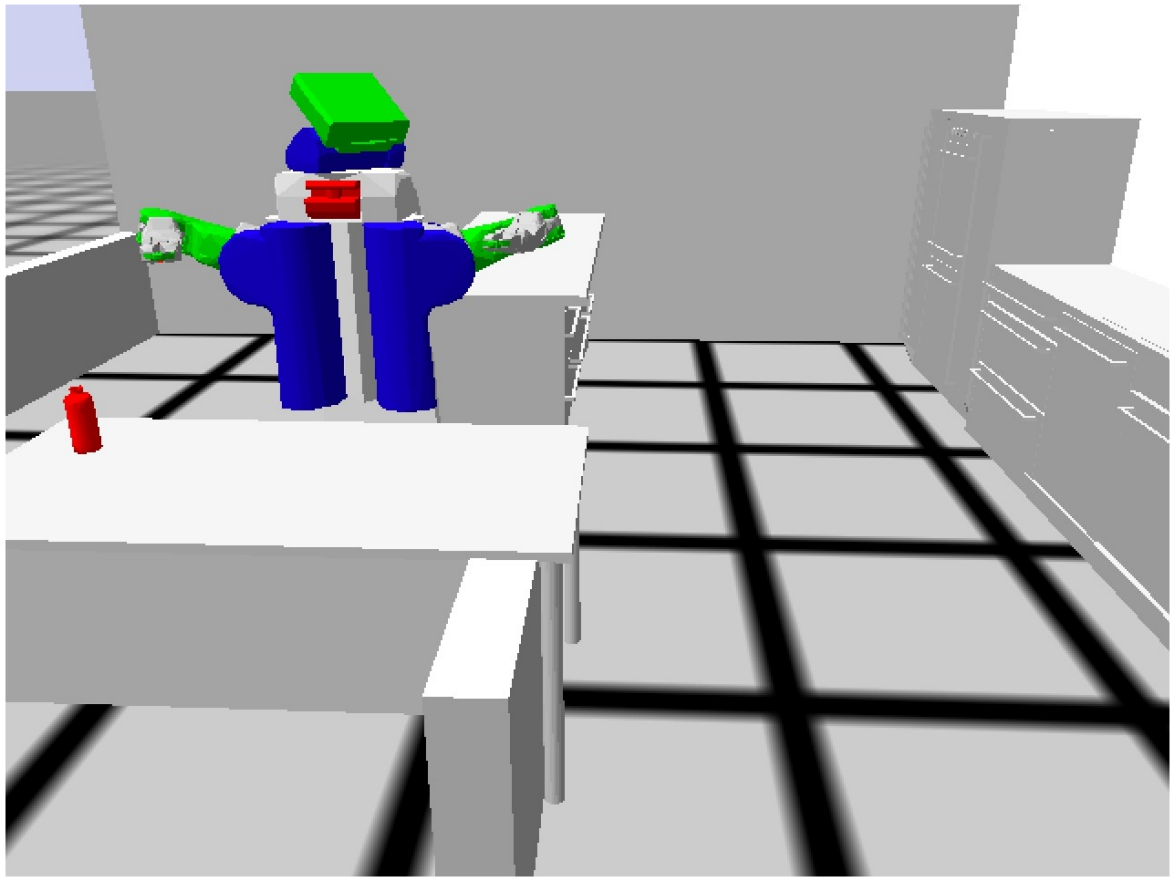
## Perform action
## look downwards



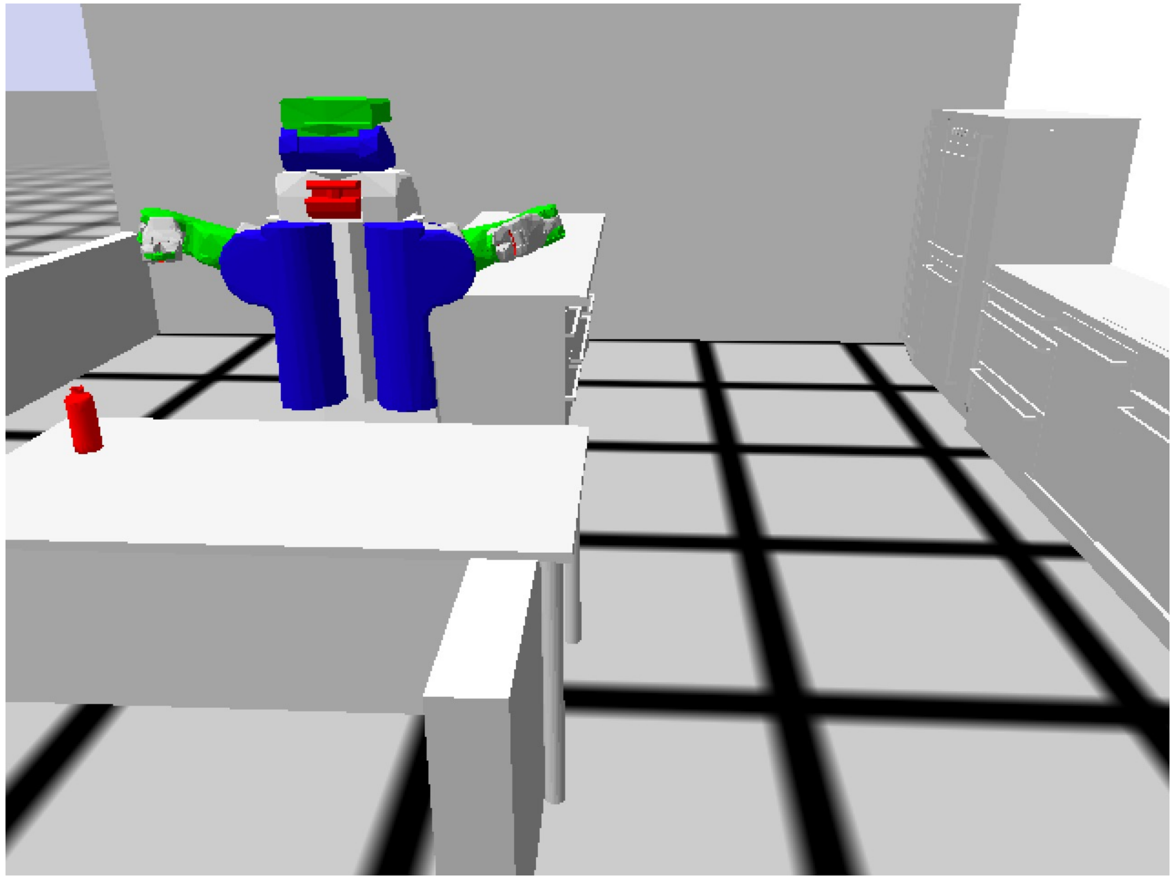
## Perform action look forwards



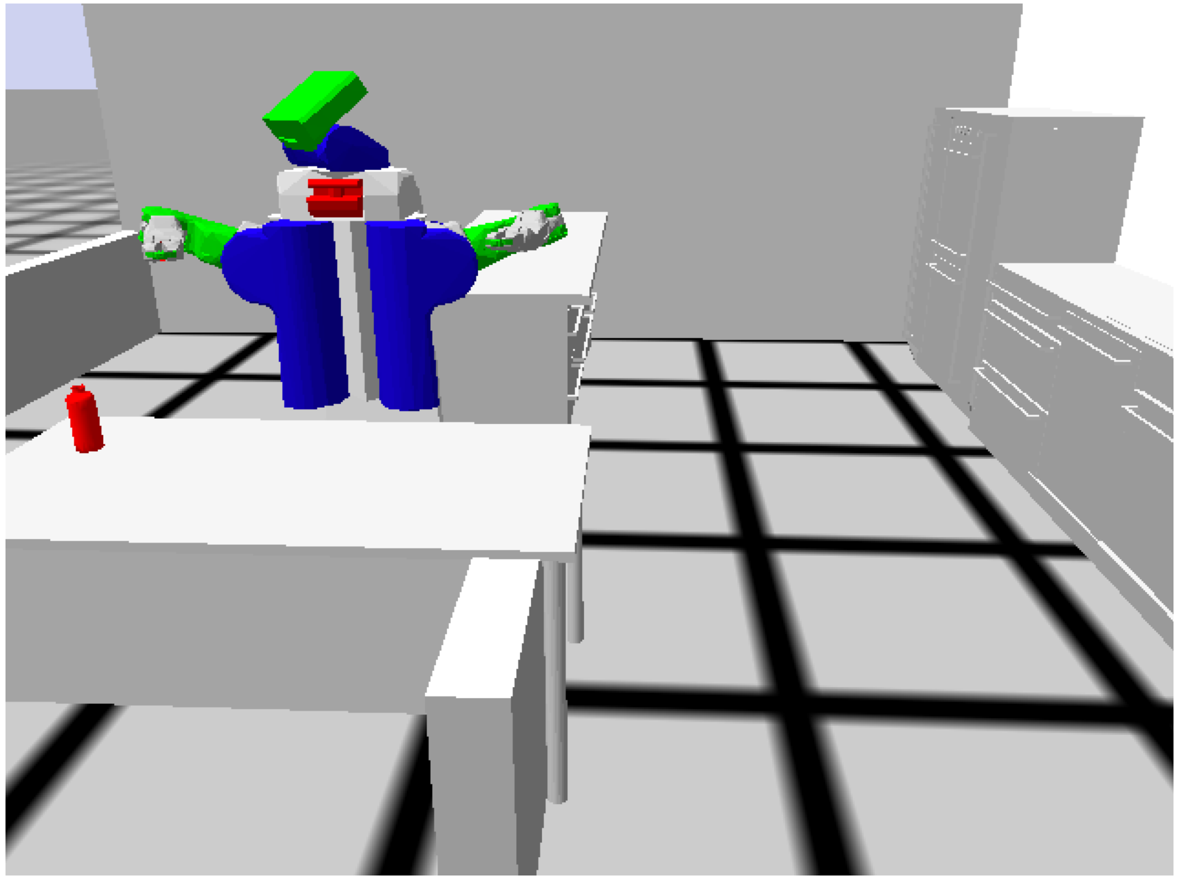
## Perform action
## look left



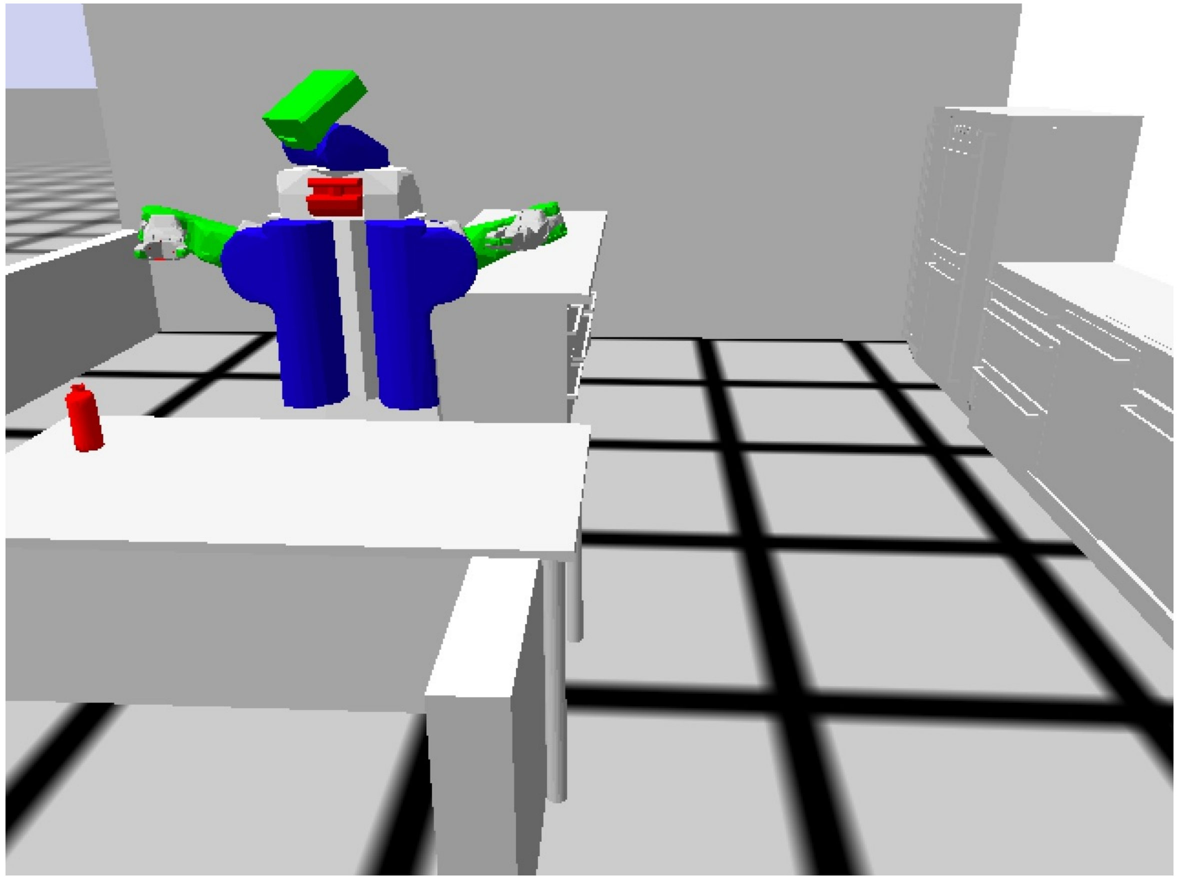
## Perform action
## look forwards



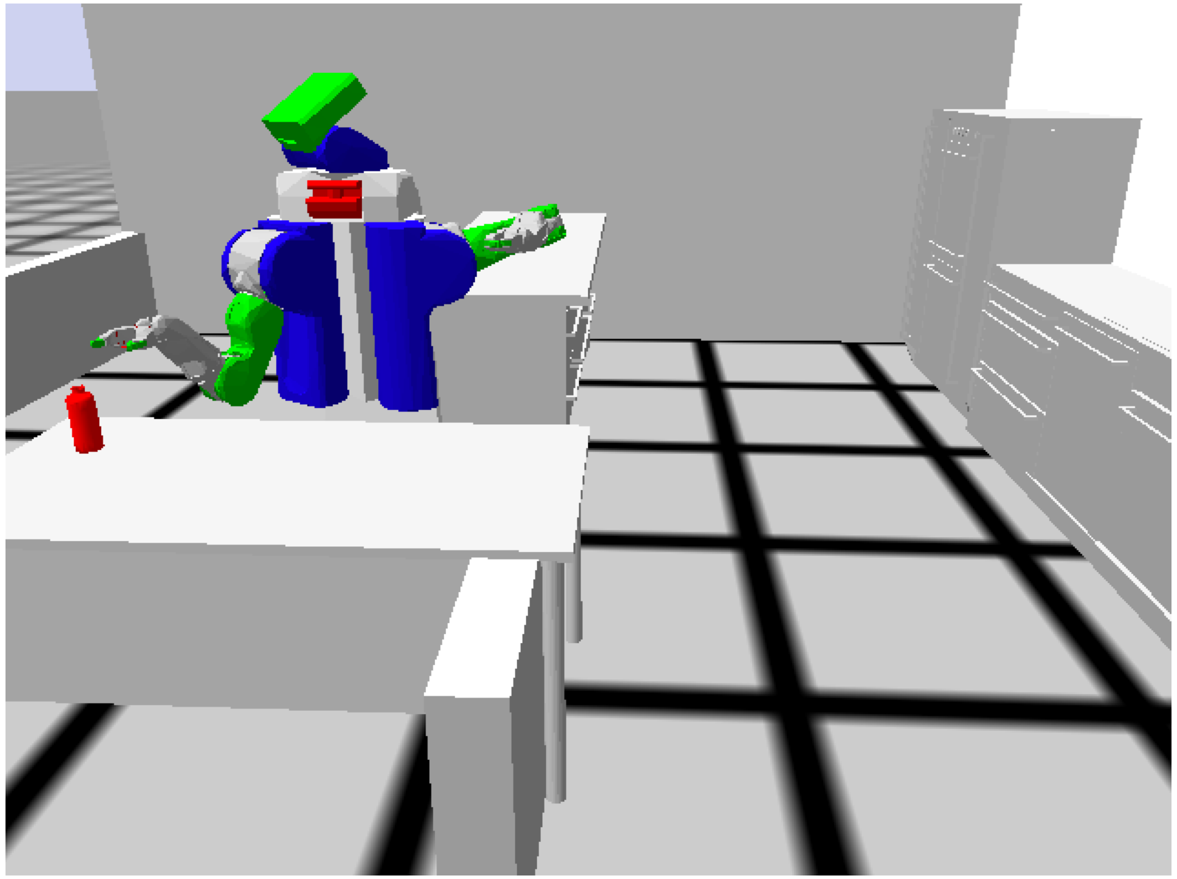
## Perform action
## look right



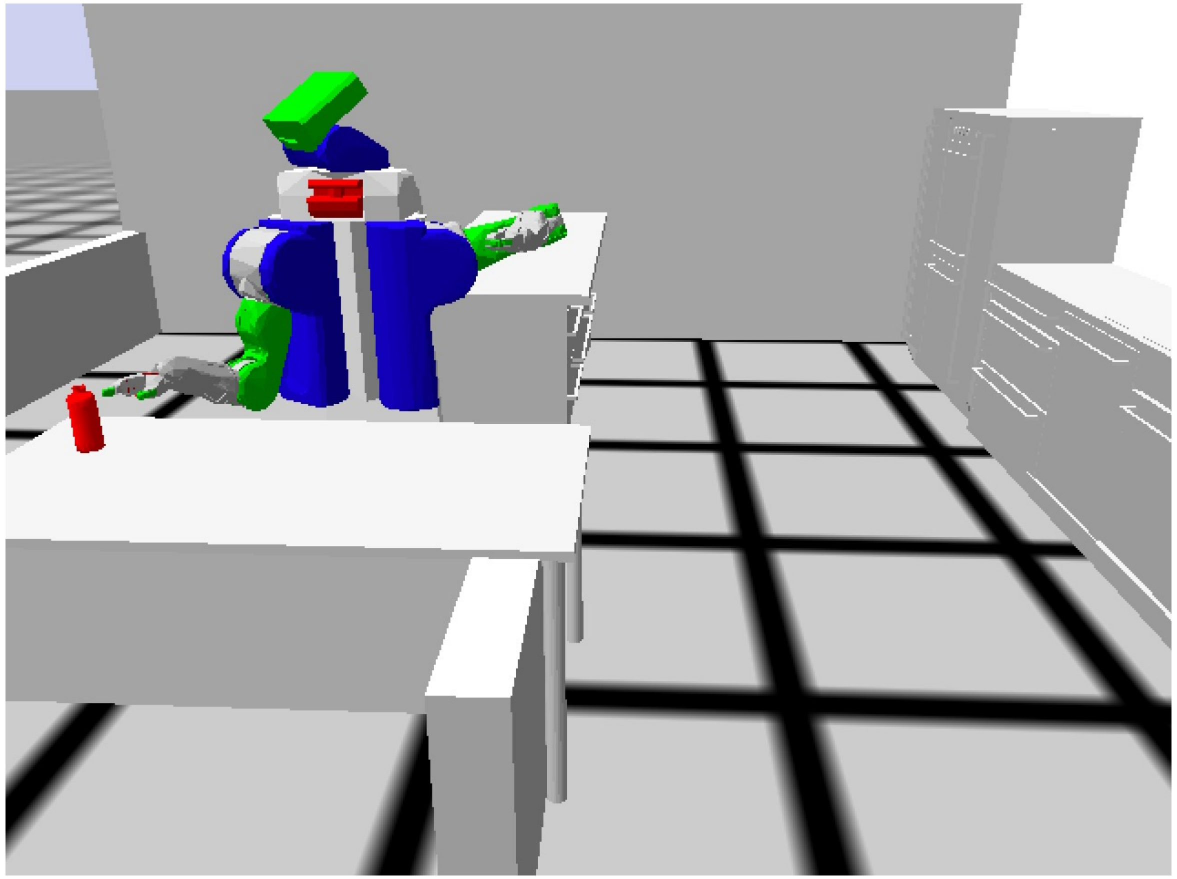
## Perform action
## pick-up: opening gripper



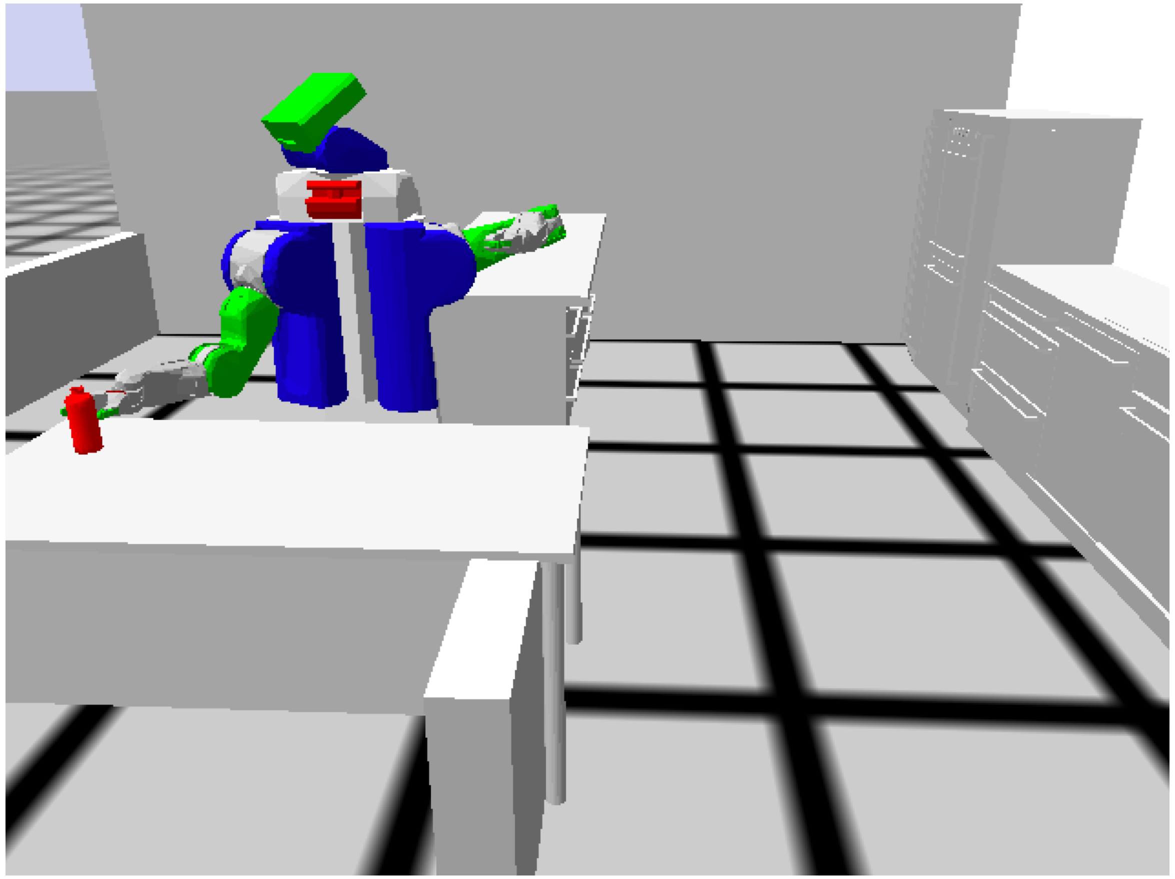
## Perform action
## pick-up: reaching



## Perform action
## pick-up: reaching

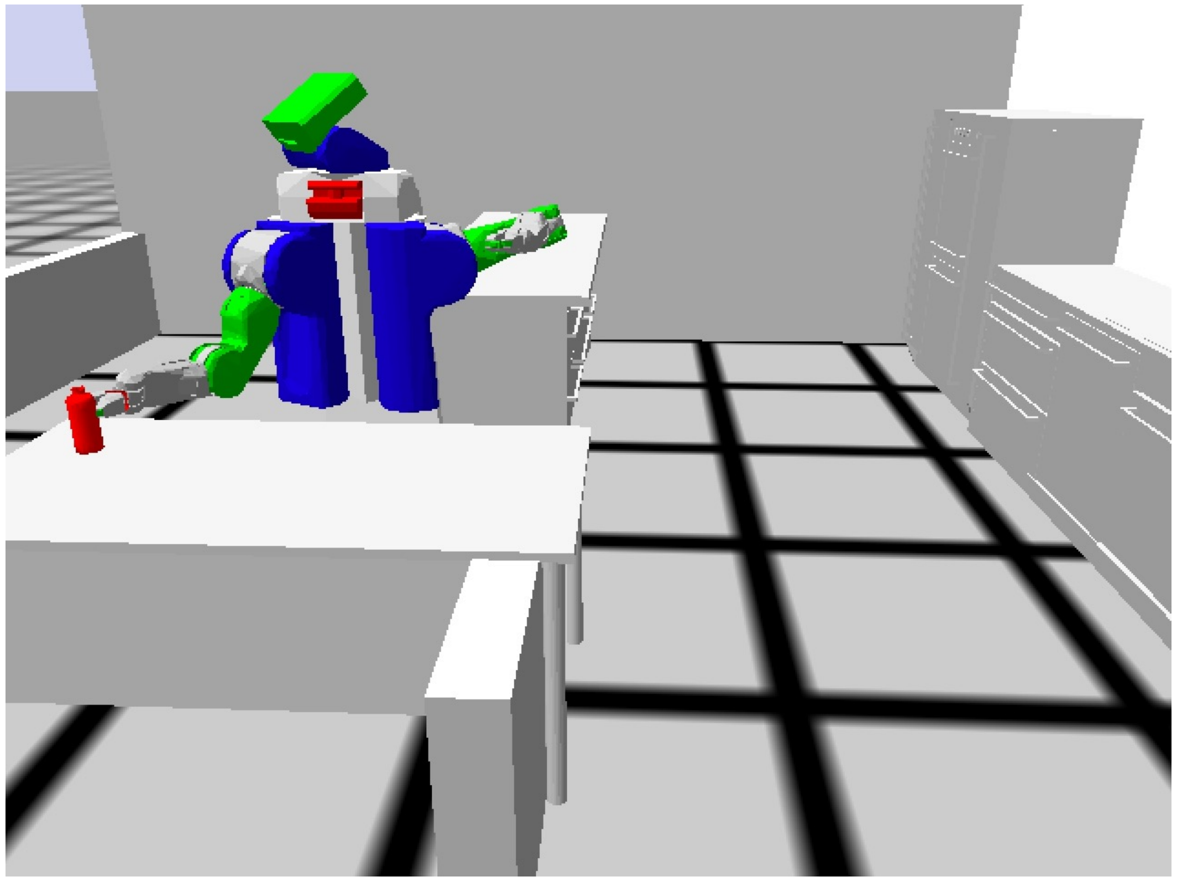


## Perform action
## pick-up: grasping

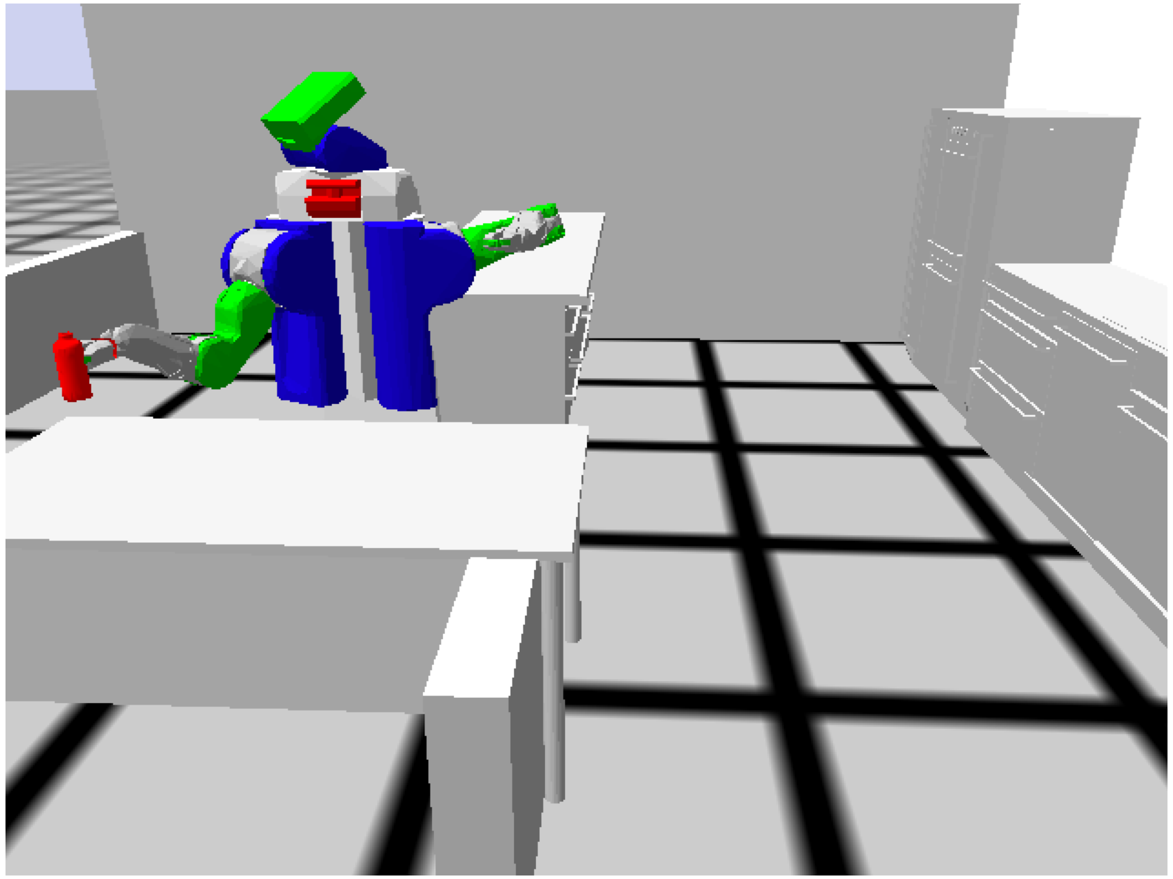


## Perform action

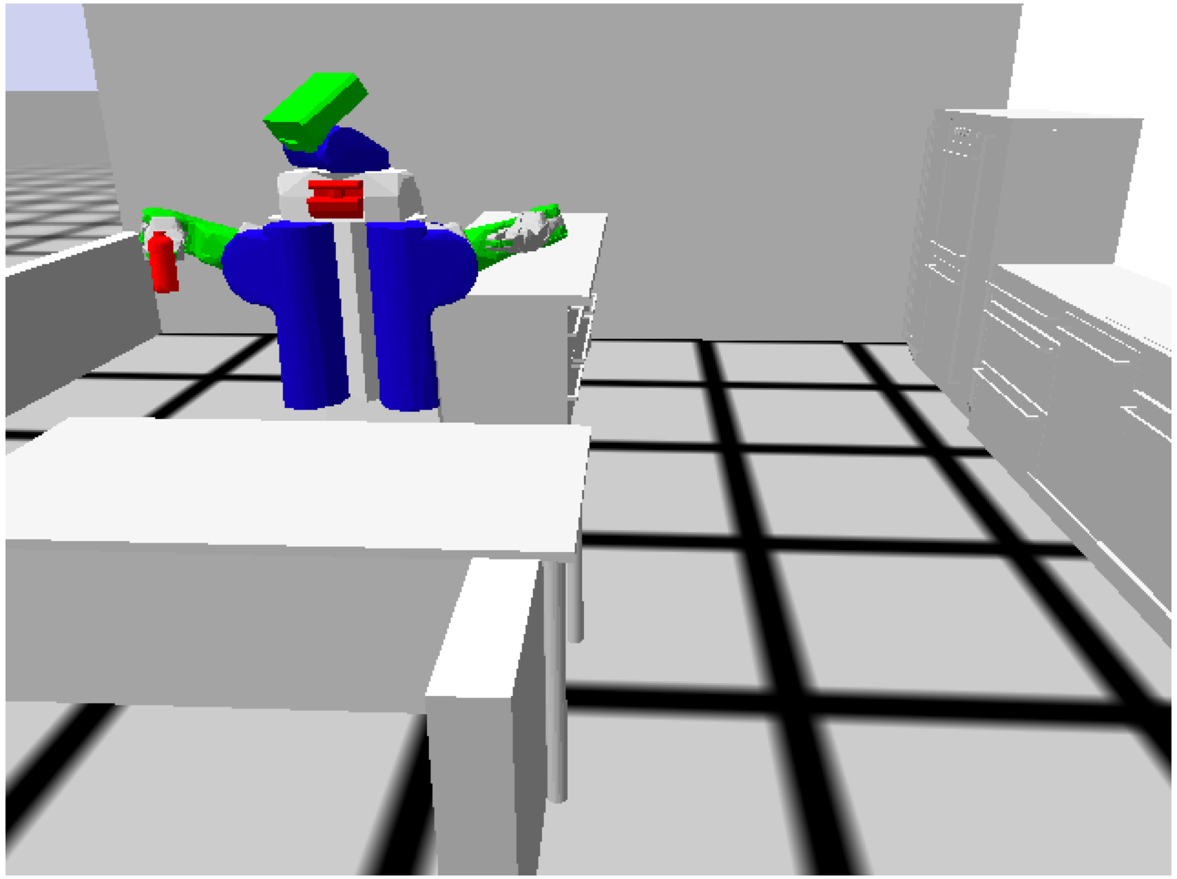
## pick-up: grasping



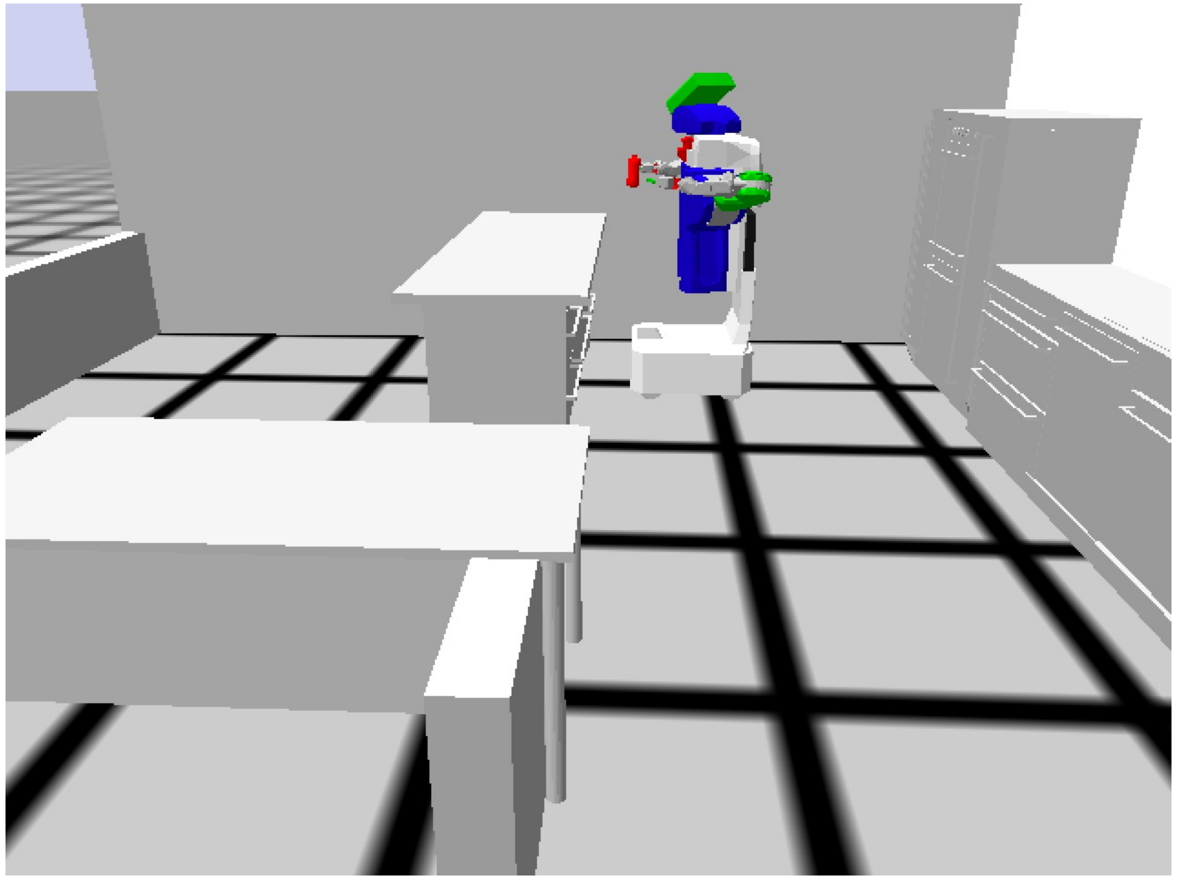
## Perform action
## pick-up: lifting



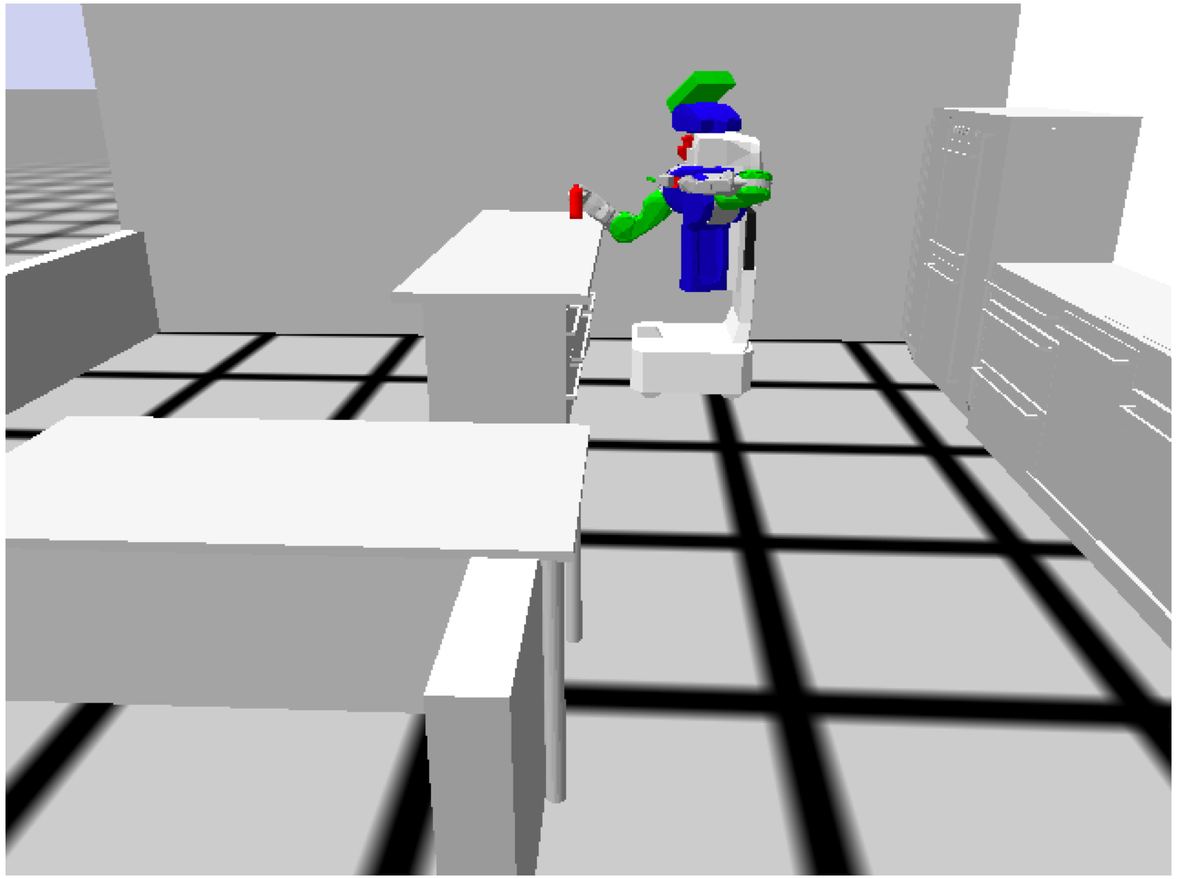
## Perform action
## pick-up: lifting



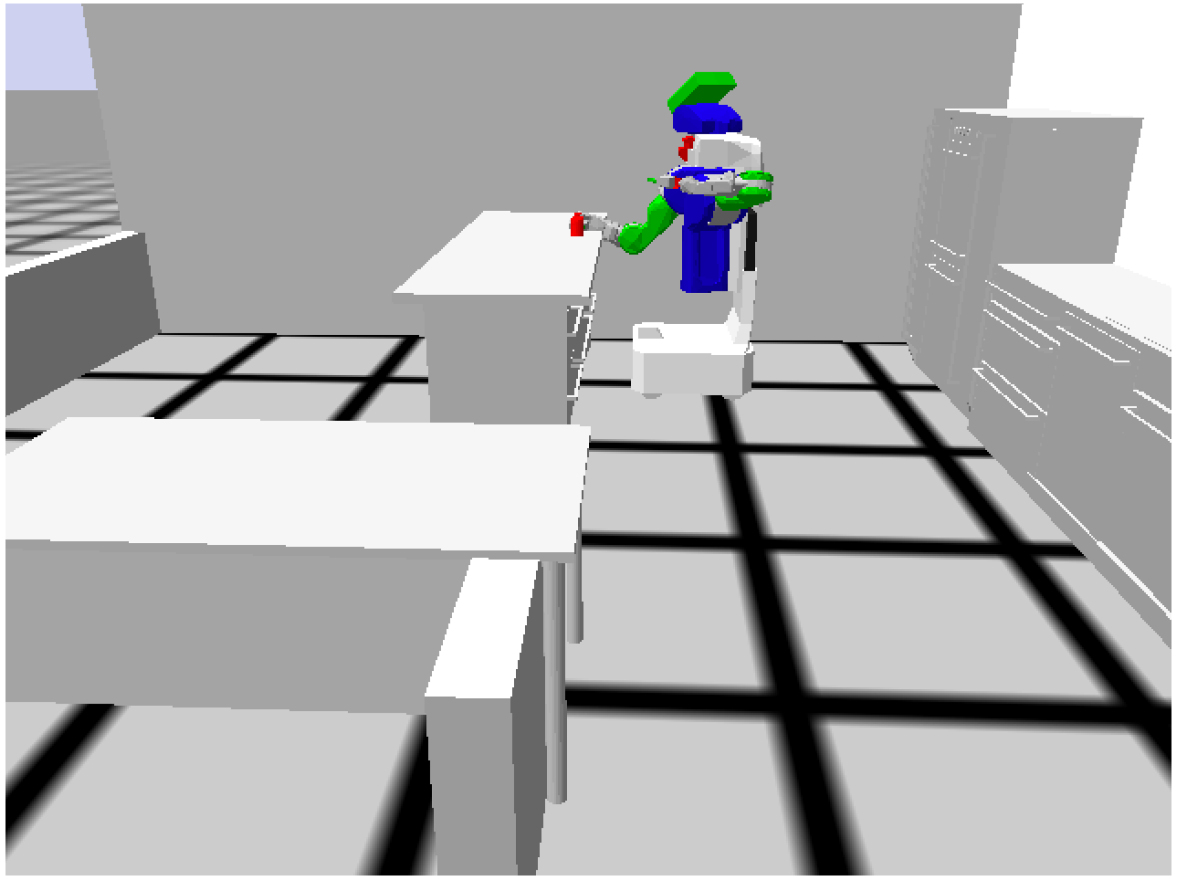
## Perform action
## move near the counter



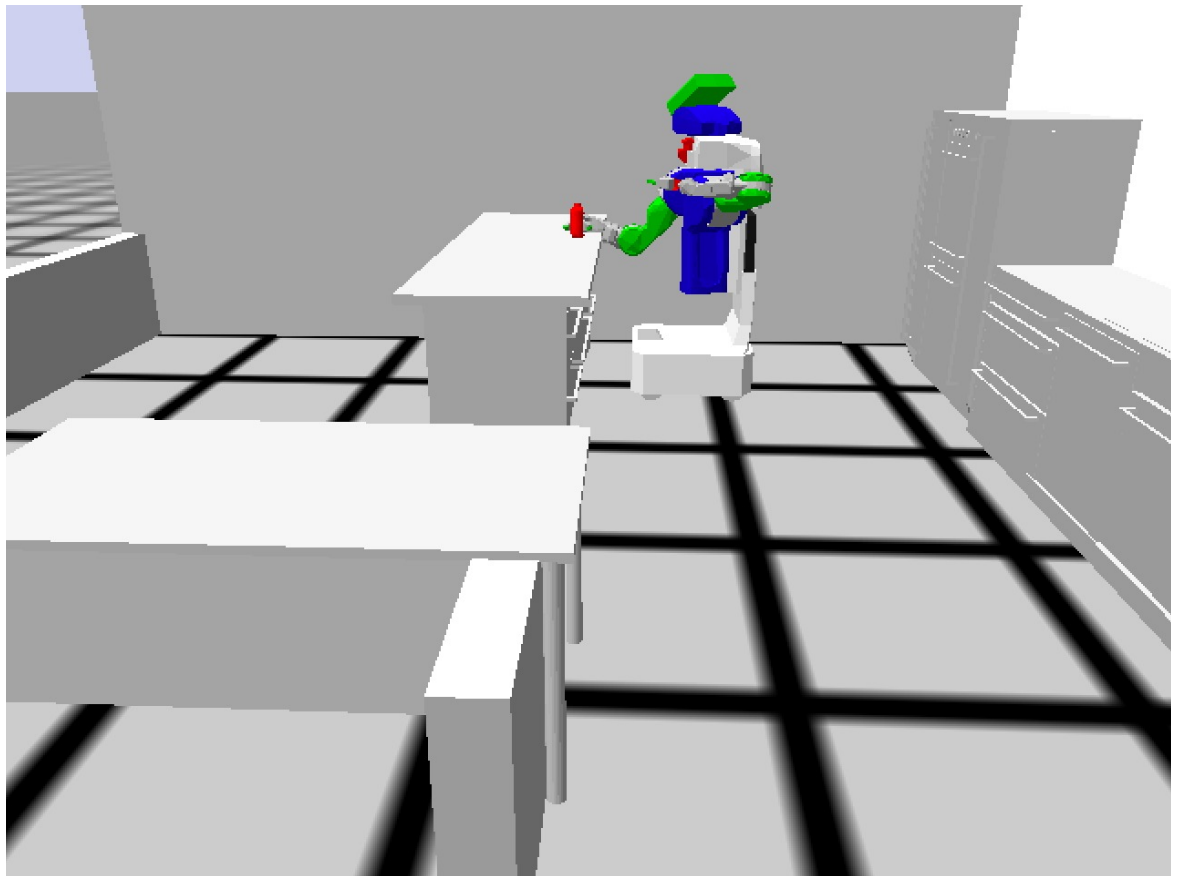
## Perform action place: reaching



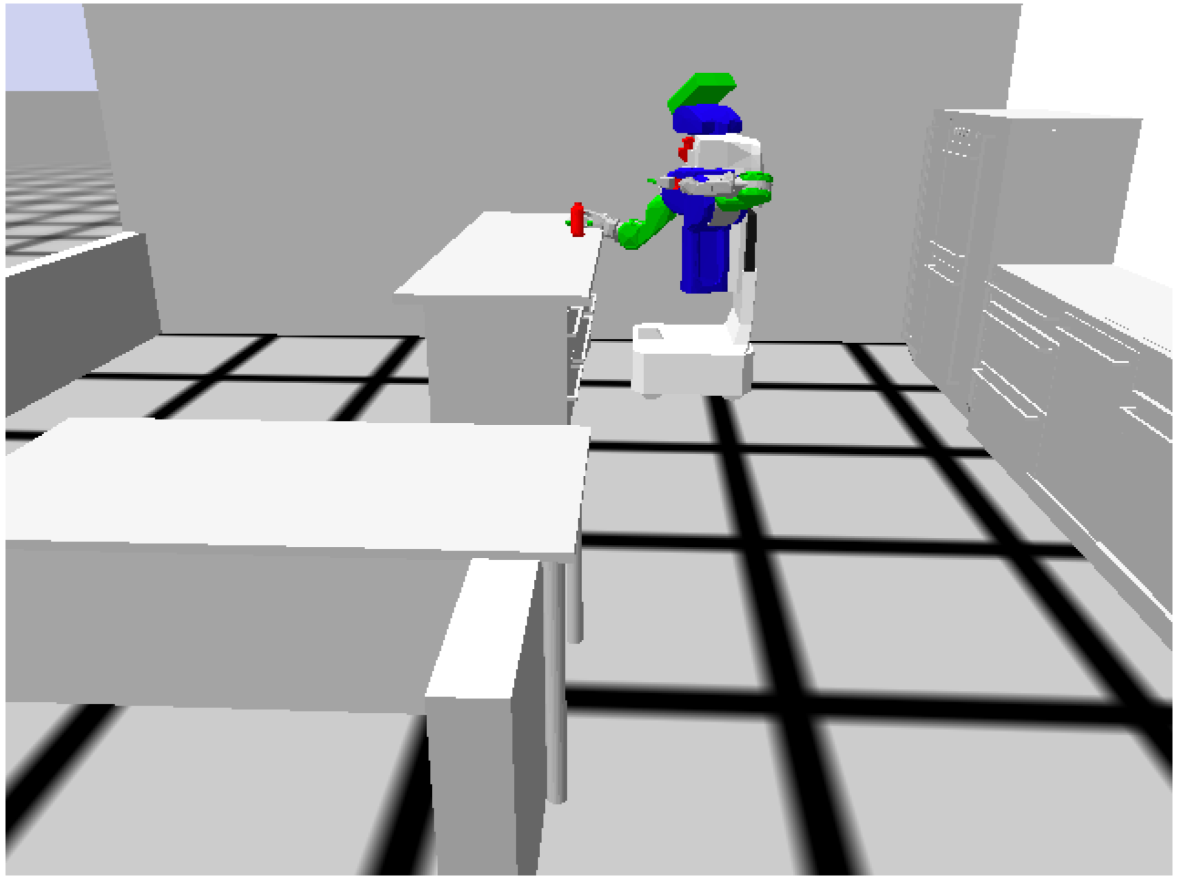
## Perform action
## place: putting



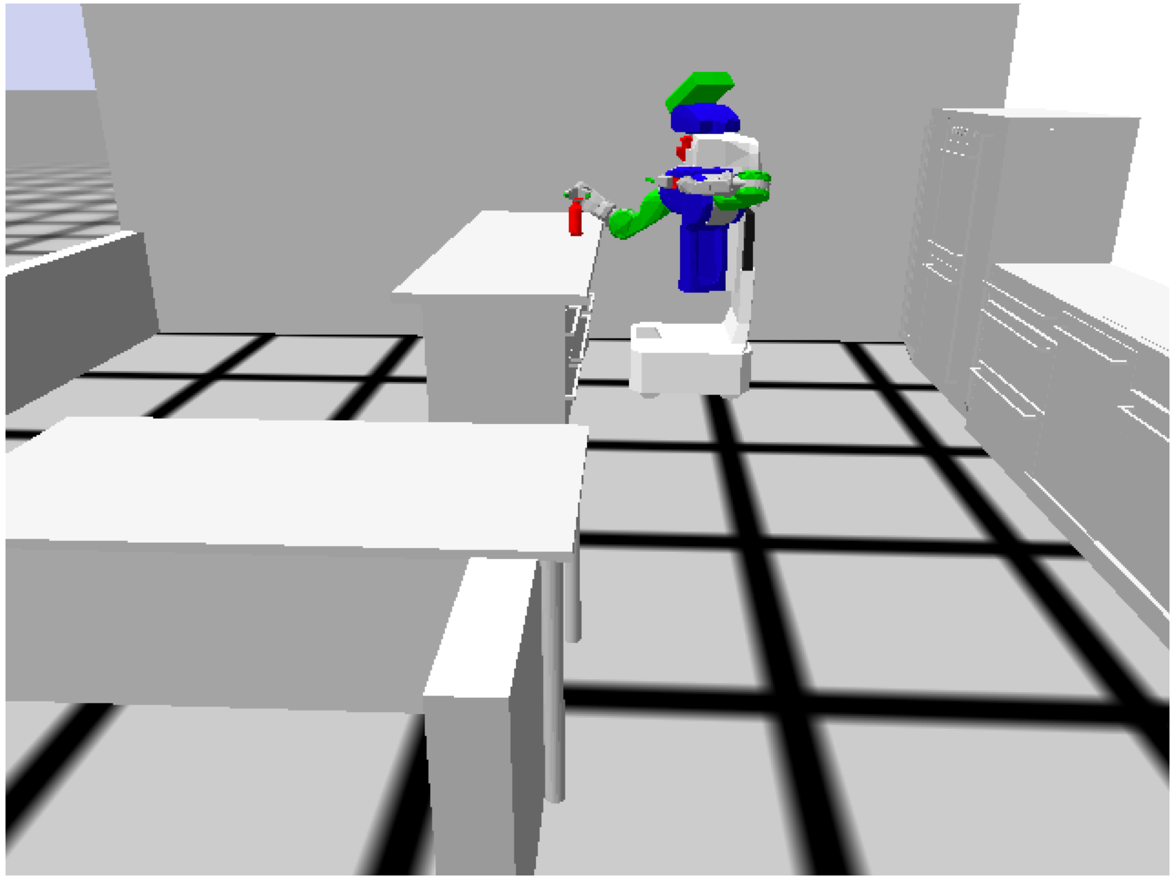
## Perform action
## place: opening gripper



## Perform action place: retracting

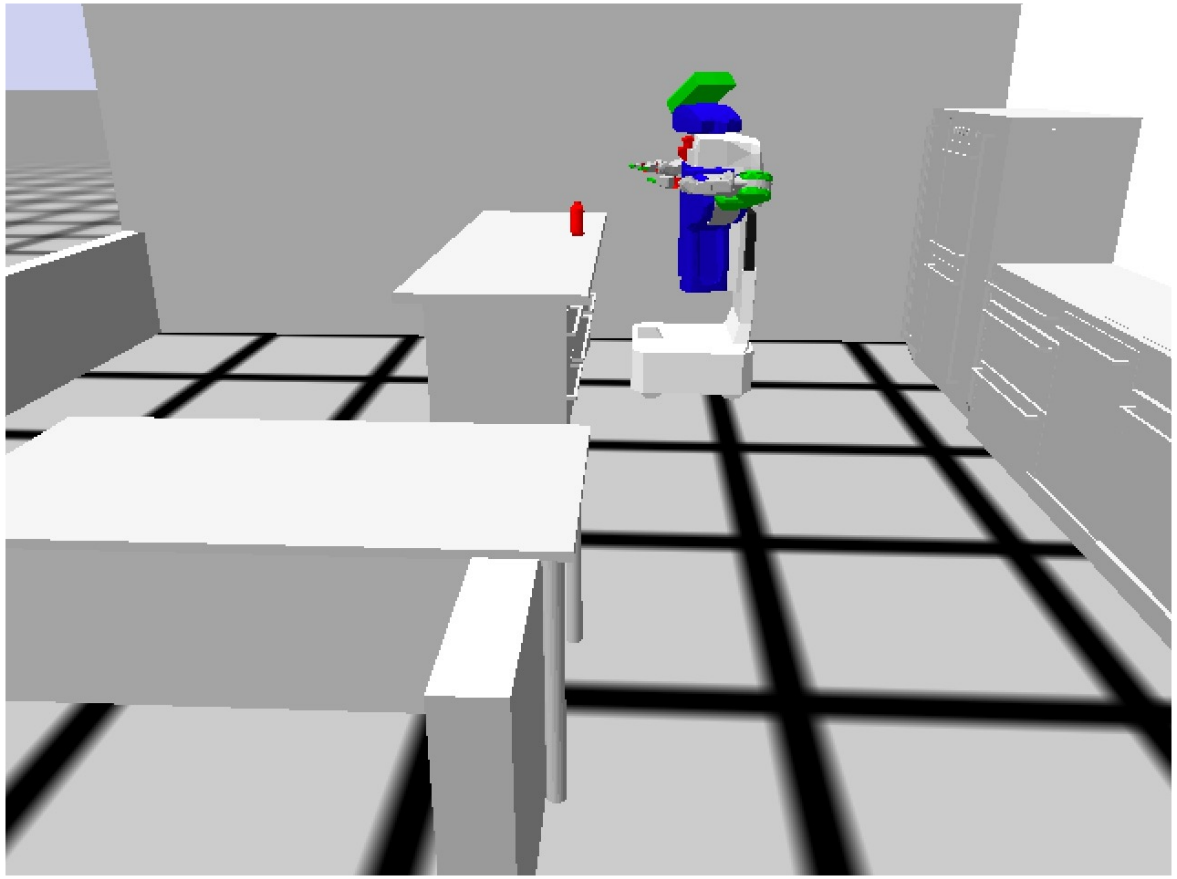


## Perform action place: retracting

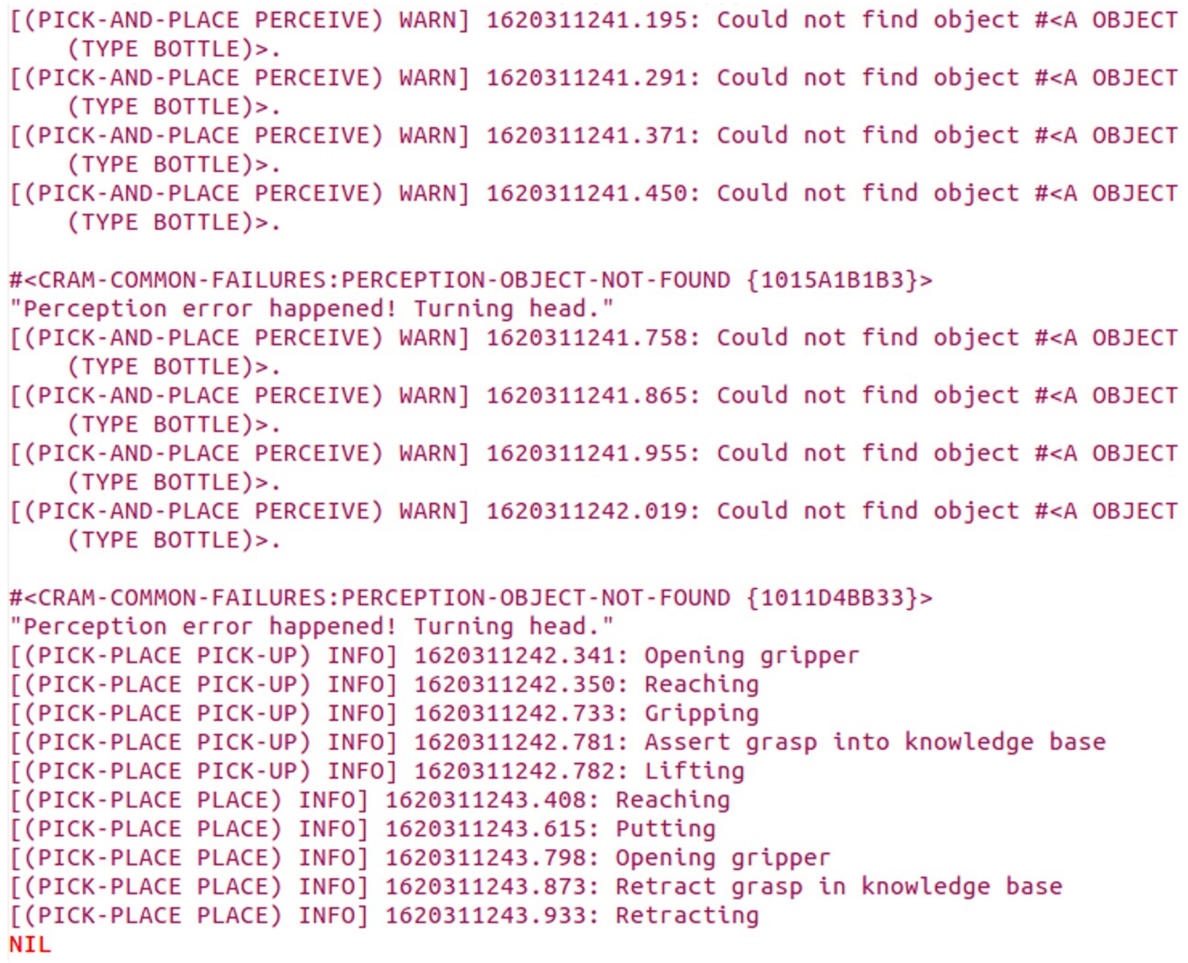


Perform action

place: retracting



## Simple Fetch and Place



```
[(PICK-AND-PLACE PERCEIVE) WARN] 1620311241.195: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.
[(PICK-AND-PLACE PERCEIVE) WARN] 1620311241.291: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.
[(PICK-AND-PLACE PERCEIVE) WARN] 1620311241.371: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.
[(PICK-AND-PLACE PERCEIVE) WARN] 1620311241.450: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.

#<CRAM-COMMON-FAILURES:PERCEPTION-OBJECT-NOT-FOUND {1015A1B1B3}>
"Perception error happened! Turning head."
[(PICK-AND-PLACE PERCEIVE) WARN] 1620311241.758: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.
[(PICK-AND-PLACE PERCEIVE) WARN] 1620311241.865: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.
[(PICK-AND-PLACE PERCEIVE) WARN] 1620311241.955: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.
[(PICK-AND-PLACE PERCEIVE) WARN] 1620311242.019: Could not find object #<A OBJECT
    (TYPE BOTTLE)>.

#<CRAM-COMMON-FAILURES:PERCEPTION-OBJECT-NOT-FOUND {1011D4BB33}>
"Perception error happened! Turning head."
[(PICK-PLACE PICK-UP) INFO] 1620311242.341: Opening gripper
[(PICK-PLACE PICK-UP) INFO] 1620311242.350: Reaching
[(PICK-PLACE PICK-UP) INFO] 1620311242.733: Gripping
[(PICK-PLACE PICK-UP) INFO] 1620311242.781: Assert grasp into knowledge base
[(PICK-PLACE PICK-UP) INFO] 1620311242.782: Lifting
[(PICK-PLACE PLACE) INFO] 1620311243.408: Reaching
[(PICK-PLACE PLACE) INFO] 1620311243.615: Putting
[(PICK-PLACE PLACE) INFO] 1620311243.798: Opening gripper
[(PICK-PLACE PLACE) INFO] 1620311243.873: Retract grasp in knowledge base
[(PICK-PLACE PLACE) INFO] 1620311243.933: Retracting
NIL
```

## Recommended Reading

CRAM zero prerequisites demo tutorial: simple fetch and place

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

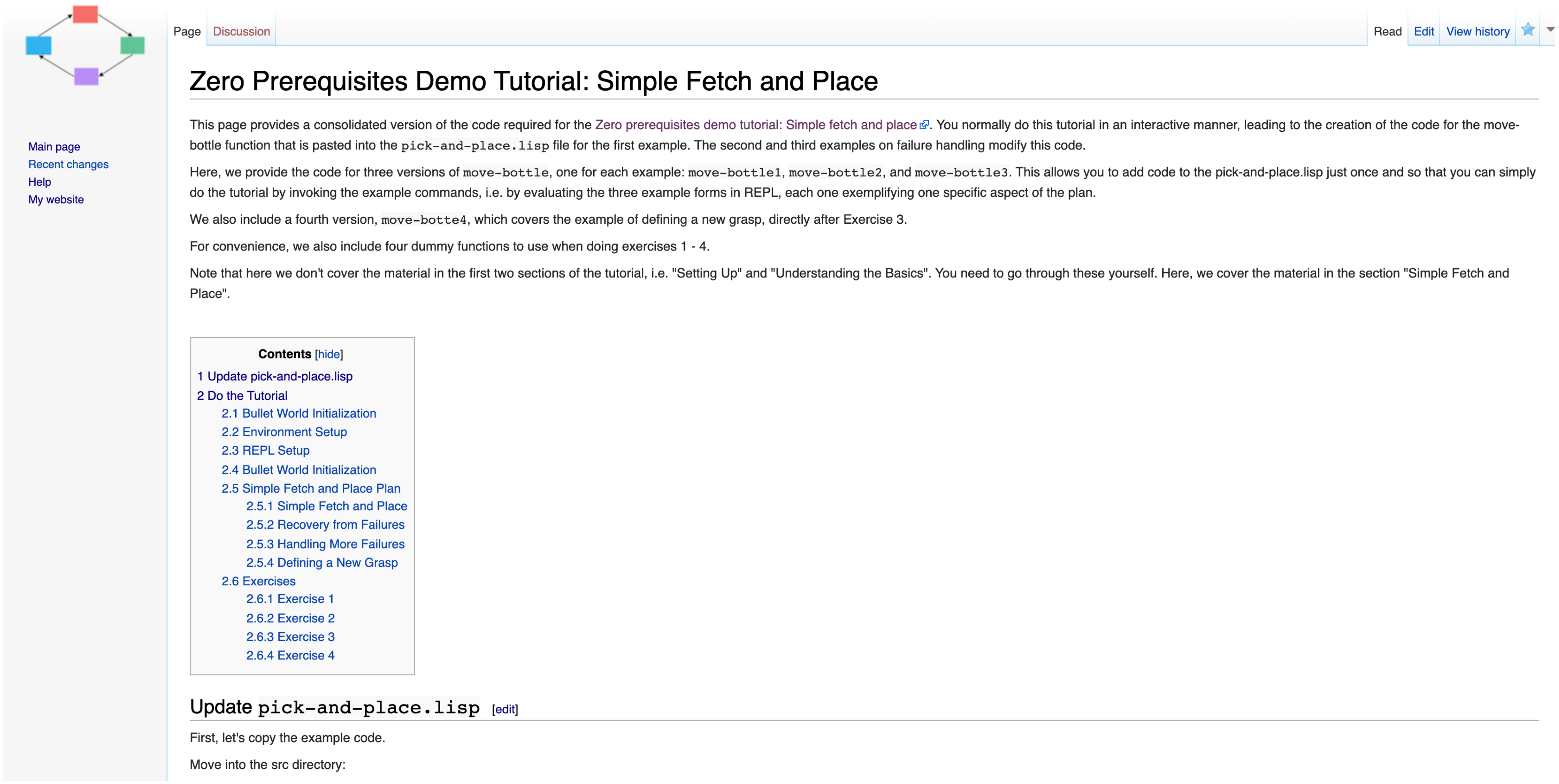
# Implementation of a pick-and-place CRAM plan

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]

## 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