MOOC Autonomous Mobile Robots Week 4

Week 4 - Manipulation

(http://www.mobilerobots.com/Accessories/PioneerGripper.aspx) Robots (https://en.wikipedia.org/wiki/Robot) have been historically classified in mobile robots (https://en.wikipedia.org/wiki/Mobile_robot) and manipulators (https://en.wikipedia.org/wiki/Industrial_robot).

Nowadays, manipulation devices are available for many mobile platforms, including ground, underwater and aerial vehicles. The Pioneer is an indoor mobile platform that can be optionally equiped with a simple yet effective manipulator device.

In this module, you will learn to control the robot gripper, and program a task for autonomously grasping an object, an carrying it to a predefined destination.

- Gripper (Gripper.ipynb)
- Searching Ball (Searching.ipynb)
- Grasping (Grasping.ipynb)
- Searching Target (Searching%20Target.ipynb)
- Complete Manipulation Task (Complete%20Task.ipynb)

Try-a-Bot: an open source guide for robot programming

Developed by:

(http://robinlab.uji.es)

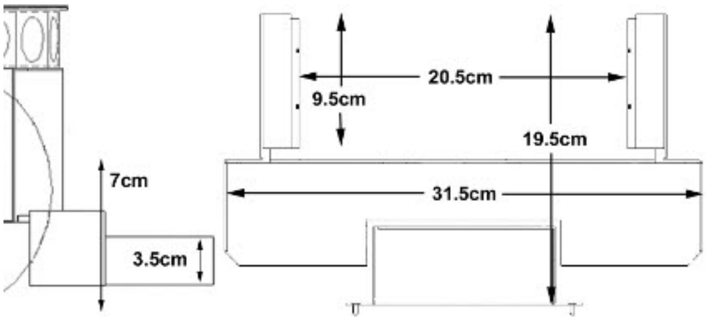
Sponsored by:

(http://www.ieee-ras.org) (http://www.cyberbotics.com) (http://www.theconstructsim.com)

Follow us:

(https://www.facebook.com/RobotProgrammingNetwork) (https://www.youtube.com/user/robotprogrammingnet)

The Gripper



(http://www.mobilerobots.com/Accessories/PioneerGripper.aspx) Grippers are robot end-effectors (https://en.wikipedia.org/wiki/Robot_end_effector) designed for grasping objects.

They are frequently used in robot manipulators, but they can nevertheless be attached to mobile platforms too, like the Pioneer robot.

In this case, the gripper is a 2-axis, 2 degree-of-freedom (dof) mechanism. It opens and closes horizontally, and raises up to carry the grasped object off the floor. The gripper paddles close together horizontally until they grasp an object or close on themselves.

In this notebook, we will use a GUI widget for manually controlling the gripper and getting familiar with it.

In [ ]:
```
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

First, the GUIs for moving the robot and the Kinect.

In [ ]:
```
import motion_widget
```

In [ ]:
```
import tilt_widget
```

Next, a new GUI widget for controlling the two degrees of freedom of the gripper: lifting and opening/closing the fingers.

In [ ]:
```
import gripper_widget
```

Alternatively, you can call the code function for setting the values for the gripper:

In [ ]:
```
lift = 0.0
fingers = 0.05
p3dx.gripper(lift,fingers)
```

where `lift` varies in an interval of -0.05 (up) to 0.05 (down) meters, and `fingers` varies in an interval of 0.0 (closed) to 0.1 (open) meters.

The task

You should teleoperate the robot with the previous GUI widgets for grasping the blue ball.

Use both the simulator window and the camera window for visual feedback.

In the following notebooks, the aim will be to program the robot for automatically doing this same task.

In [ ]:
```
%matplotlib inline
```

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
plt.imshow(p3dx.image);
# Click here and press Ctrl+Enter to refresh the image
```

Next: Searching the Ball (Searching.ipynb)

Searching

(http://www.hongkiat.com/blog/danbo-amazon-cardboard-robot-photos/) The first step of the manipulation task is searching for the object (the blue ball) and moving the robot near to it, so that the ball fits between the gripper fingers.

This task is very similar to the visually-guided line-following task, since the ball is brightly colored and it can be segmented (https://en.wikipedia.org/wiki/Image_segmentation) from the background with some simple image processing operations. Then, the robot can be controlled for approaching the ball.

A possible algorithm would be:

```
1. open the gripper and tilt the kinect for searching the ball
2. turn the robot until the ball is centered in the camera image
3. move the robot forward until the ball is near the bottom of the imag
e
```

First, let's initialize the robot.

In [2]:
```
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

Colored blob detection

We already know how to locate a colored blob (the line in the previous week, the ball now) in the image, thanks to its **centroid**, which is computed from the image moments. But we should also consider the case that the ball is *not visible* in the image. One solution is checking the **area** of the blob, which is given by M_{00}, and returning the centroid values only if the area is greater than zero.

Let's define a function named `color_blob` for computing the area and centroid of a colored blob. If the blob is not detected, the area will be zero, and the centroid coordinates will be `None` (the Python value for null). This function is an improved version of the code used for the line following task.

In [1]:
```
import cv2
def color_blob():
    hsv = cv2.cvtColor(p3dx.image, cv2.COLOR_RGB2HSV)
    mask = cv2.inRange(hsv, lower, upper)
    M = cv2.moments(mask)
    area = M['m00']
    if area > 0:
        cx = int(M['m10']/area)
        cy = int(M['m01']/area)
    else:
        cx = None
        cy = None
    return area, cx, cy
```

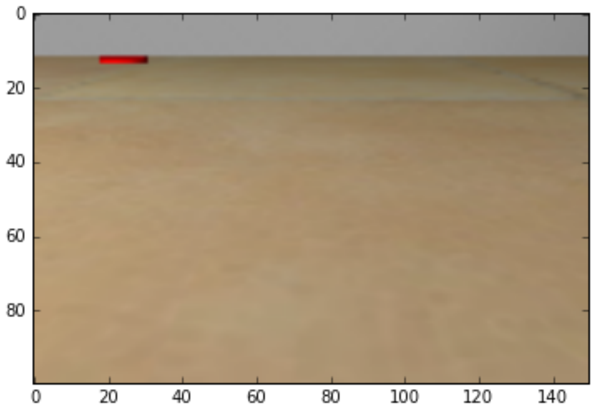
Step 1: initial position

Open the gripper and tilt the kinect for searching the ball: the fingers should be wide open, the gripper down close to the ground, and the kinect should be tilted properly for searching throughout the room.

In [16]:
```
p3dx.gripper(0.05,0.1)
p3dx.tilt(-0.35)
```

In a good configuration, the top plate of the robot would be only slightly visible at the bottom of the image, and the walls of the room should be visible too (that is the robot should not be neither looking too much to the floor, nor to the ceiling). You may check the image in the next cell, and change the above parameters if necessary, until the result is satisfactory.

In [17]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.imshow(p3dx.image);
```



Step 2: turning

We are going to use a `while` loop for turning the robot, which will stop when the blob is detected and its coordinates are approximately in the center of the image. Most of the code is given in the next cell, but you must figure out some values.

- First, since the color of the ball is blue, you need to find out its proper **hue** value (please remember that the hue range in OpenCV scales from 0 to 180).

In [21]:
```
import numpy
lower = numpy.array([110, 100, 100])
upper = numpy.array([130, 255, 255])
```

- Next, you should choose the interval for considering the blob as centered.

In [31]:
```
def is_blob_centered():
    area, cx, cy = color_blob()
    if area > 0 and cx >= 70 and cx < 80:
        return True
    else:
        return False
```

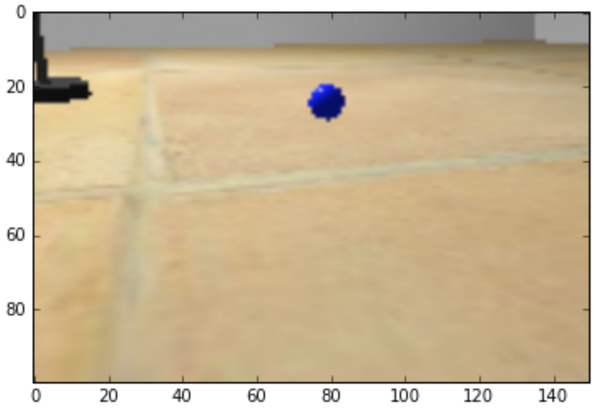
- Finally, you must provide the velocity values for turning.

In [19]:
```
while not is_blob_centered():
    p3dx.move(-0.5,0.5)
p3dx.stop()
```

Again, you can check the result:

```
In [20]: plt.imshow(p3dx.image)
         print('Area: %d, cx: %d, cy: %d' % color_blob())
```

```
Area: 17850, cx: 78, cy: 24
```



Step 3: approaching

As the robot moves forward and approaches to the ball, the position of the ball in the image will go down.

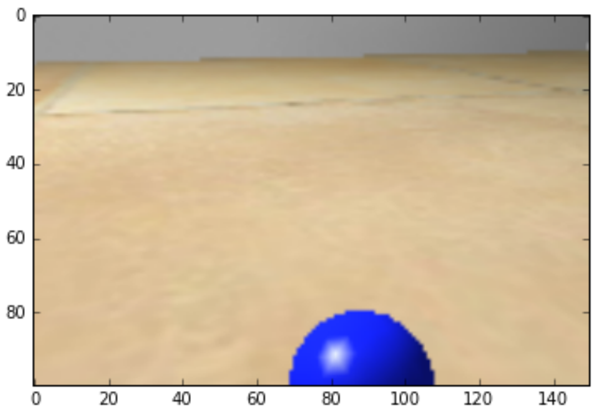
We can define a threshold for stopping the robot before the ball goes out of the image. The code is very similar to the previous step.

```
In [33]: def is_blob_close():
             area, cx, cy = color_blob()
             if area > 0 and cy >= 90:
                 return True
             else:
                 return False
```

```
In [25]: while not is_blob_close():
             p3dx.move(1.0,1.0)
         p3dx.stop()
```

Let's check the result:

```
In [27]: plt.imshow(p3dx.image);
```



Additional step: fine motion

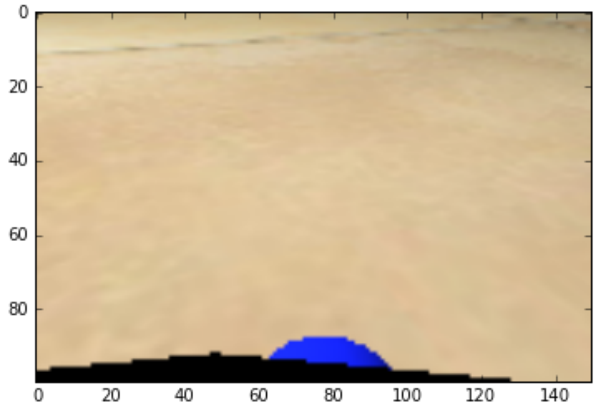
If the ball is not between the fingers yet, you need to move the robot closer. This can be done in open loop, but this is prone to errors. A better option is to tilt the kinect lower, and repeat a new iteration of the centering and approaching steps.

In [ ]:
```
p3dx.tilt(-0.47)
```

In [32]:
```
while not is_blob_centered():
    p3dx.move(-0.5,0.5)
p3dx.stop()
```

In [34]:
```
while not is_blob_close():
    p3dx.move(1.0,1.0)
p3dx.stop()
```

In [35]:
```
plt.imshow(p3dx.image);
```



Next: Grasping (Grasping.ipynb)

Grasping

Assuming that the robot is well positioned, the grasping task is straightforward:

1. `close fingers`
2. `lift gripper`

Not every task requires a lot of code! ;-)

In [1]:
```
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

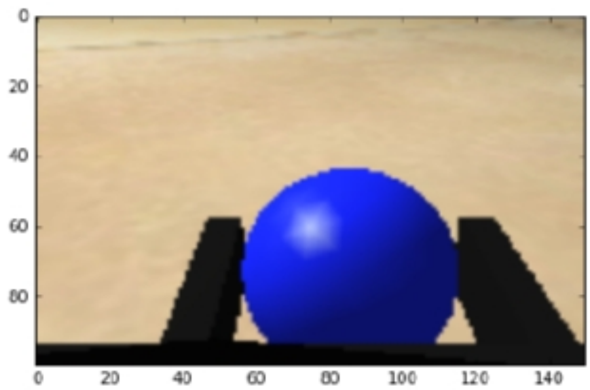
Close fingers

In [2]:
```
p3dx.gripper(0.05,0.0)
```

Lift gripper

In [3]:
```
p3dx.gripper(-0.05,0.0)
```

In [6]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.imshow(p3dx.image);
```



If the ball is not grasped, you should go back the previous notebook and refine the values for centering and approaching the ball.

Next: Searching Target (Searching%20Target.ipynb)

Searching the Target

The target is a red disk on the floor. The robot should approach the disk, and gently leave the ball on top of it.

In [ ]:

```
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

You can use the same strategy for searching the disk, since it can also be detected by color (in this case, red).

In fact, the code for the segmentation function is exactly the same (only the `lower` and `upper` values will change).

In [ ]:

```
import cv2
def color_blob():
    hsv = cv2.cvtColor(p3dx.image, cv2.COLOR_RGB2HSV)
    mask = cv2.inRange(hsv, lower, upper)
    M = cv2.moments(mask)
    area = M['m00']
    if area > 0:
        cx = int(M['m10']/area)
        cy = int(M['m01']/area)
    else:
        cx = None
        cy = None
    return area, cx, cy
```

These are the values for segmenting red pixels in the image.

In [ ]:

```
import numpy
lower = numpy.array([ 0, 100, 100])
upper = numpy.array([ 10, 255, 255])
```

In [ ]:

```
p3dx.tilt(0.0)
```

In [ ]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.imshow(p3dx.image);
```

Turning

The following code is exactly the same as that used in the search for the ball. The robot will turn until the target is centered in the image.

In [ ]:

```
def is_blob_centered():
    area, cx, cy = color_blob()
    if area > 0 and cx >= 70 and cx < 80:
        return True
    else:
        return False
```

In [ ]:

```
while not is_blob_centered():
    p3dx.move(-0.5,0.5)
p3dx.stop()
```

In [ ]:

```
plt.imshow(p3dx.image);
```

Approaching

Copy and paste the code from the appraoch step of the ball-searching notebook.

In [ ]:

```
def ...
```

In [ ]:

```
while ...
```

In [ ]:

```
plt.imshow(p3dx.image);
```

Fine adjustment

Again, it might be necessary an additional step for finely reaching the final position. Usually, only translation is necessary, no need for rotation.

In [ ]:

```
p3dx.tilt(-0.47)
```

In [ ]:

```
while not is_blob_close():
    p3dx.move(1.0,1.0)
p3dx.stop()
```

In [ ]:

```
plt.imshow(p3dx.image);
```

Releasing the ball

Copy and paste the statements from the grasping notebook that lower the gripper and open the fingers. Finally move the robot backwards.

In [ ]:

```
p3dx.gripper(...)
```

In [ ]:

```
p3dx.gripper(...)
```

In [ ]:

```
p3dx.move(-1.0,-1.0)
p3dx.sleep(3)
p3dx.stop()
```

Next: Complete Manipulation Task (Complete%20Task.ipynb)

Try-a-Bot: an open source guide for robot programming

Developed by:
Robotic Intelligence Lab @ UJI
(http://robinlab.uji.es)

Sponsored by:

(http://www.ieee-ras.org) (http://www.cyberbotics.com) (http://www.theconstructsim.com)

Follow us:

(https://www.facebook.com/RobotProgrammingNetwork) (https://www.youtube.com/user/robotprogramminc

Complete Manipulation Task

Let's put everything together for the autonomous execution of the whole task.

In [ ]:

```
import packages.initialization
import pioneer3dx as p3dx
p3dx.init()
```

Helper functions

In [ ]:

```
import cv2
import numpy

def color_blob():
    hsv = cv2.cvtColor(p3dx.image, cv2.COLOR_RGB2HSV)
    mask = cv2.inRange(hsv, lower, upper)
    M = cv2.moments(mask)
    area = M['m00']
    if area > 0:
        cx = int(M['m10']/area)
        cy = int(M['m01']/area)
    else:
        cx = None
        cy = None
    return area, cx, cy
```

In [ ]:

```
def is_blob_centered():
    area, cx, cy = color_blob()
    if area > 0 and cx >= 70 and cx < 80:
        return True
    else:
        return False
```

In [ ]:

```
def is_blob_close():
    area, cx, cy = color_blob()
    if area > 0 and cy >= 90:
        return True
    else:
        return False
```

Search ball

Copy and paste the necessary code from previous notebooks for these subtasks:

- locate ball
- approach to the ball
- pick the ball

In [ ]:

```
lower = numpy.array([110, 100, 100])
upper = numpy.array([130, 255, 255])
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Search target

Same for the target:

- locate target
- approach to the target
- release the ball

In [ ]:

```
lower = numpy.array([ 0, 100, 100])
upper = numpy.array([ 10, 255, 255])
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Try-a-Bot: an open source guide for robot programming

Developed by:

(http://robinlab.uji.es)

Sponsored by:

(http://www.ieee-ras.org) (http://www.cyberbotics.com) (http://www.theconstructsim.com)

Follow us:

(https://www.facebook.com/RobotProgrammingNetwork)

(https://www.youtube.com/user/robotprogrammingnet)