OpenCV for Robotics

Unit 5: ARTags (Augmented Reality)

Introduction

- Summary -

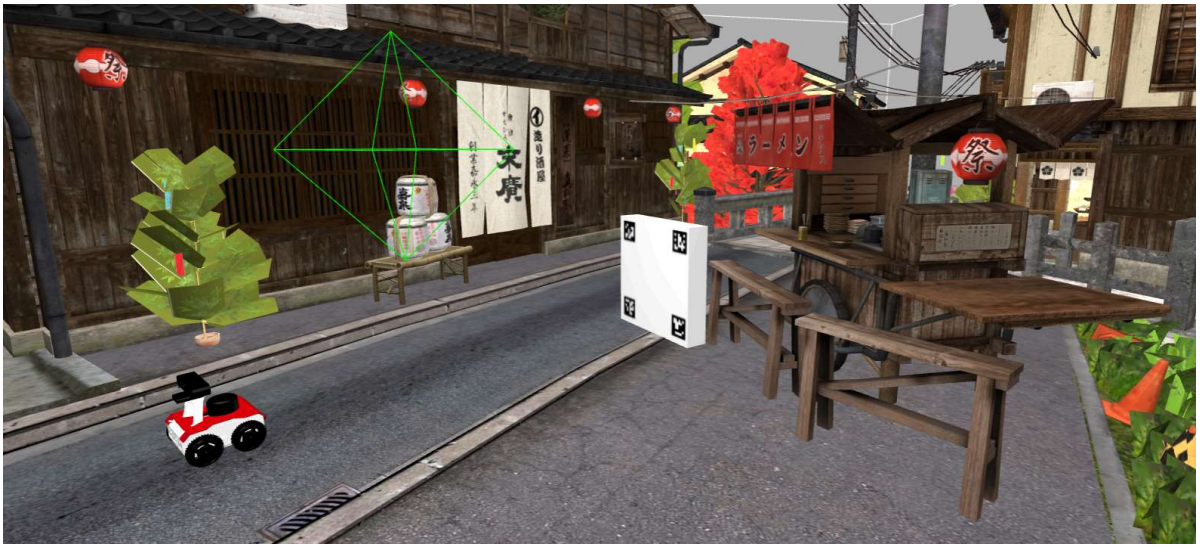
Estimated time to completion: **3-4 hours**

In this course, we will talk about:

- The ARtags, Augmented Reality, that is very useful in robotics, and how you can use it with a robot in ROS.

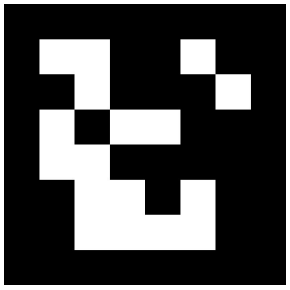
- End of Summary -

So, Augmented Reality Tags, as its name implies, are square fiducial markers designed to support augmented reality. They can be used to the apparition of virtual objects in real images; also, they give us a good approximation of the position and orientation of objects for robotics. For example, if it is a big area to explore and play with. For this module we are going to use the ArUco Library with opencv, developed by Rafael Muñoz and Sergio Garrido (https://www.sciencedirect.com/science/article/abs/pii/S0031320314000235).



5.0 ArUco Library

First, we are going to the creation of the tags. *An AruCo marker is a synthetic square marker composed of a wide black border and an inner binary matrix, which determines its identifier (id)* -Opencv docs (https://docs.opencv.org/trunk/d5/dae/tutorial_aruco_detection.html).



Opencv has some predefined dictionaries of AR markers that can be used for specific tasks. They consist simply of a list of binary codifications, whose main characteristics are the size of the marker and the length of the dictionary.

For the creation of the Tags, we are going to use a dictionary of length 250 and markers with size of 6x6 (DICT_6x6_250).

- Notes -

Don't forget to pay attention to the comments in the examples, you will find important info there.

- End of Notes -

5.1 ArUco Dictionary

- Example 5.1 -

Here you have the code in python.

In [ ]:
```
import cv2
import numpy as np
from cv2 import aruco



#Initialize the dictionary
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
for i in range (1, 5):

    size = 700
    img = aruco.drawMarker(aruco_dict, i, size)

    cv2.imwrite('/home/user/catkin_ws/src/unit5/Tags/image_'+str(i)+".jpg",img

    cv2.imshow('artag',img)
    cv2.waitKey(0)
    cv2.destroyAllWindows
```

The parameters of this function are pretty straightforward. The first one is the dictionary we are going to use. This one was defined at the begining of the code *(DICT_6X6_250)*. The second parameter is the id that will be assigned. In this case, we have a loop so we will create 4 markers with the ids 0,1,2, and 3, respectively. Finally, the Size makes reference to the size of the output image, in this case, it will be 700x700 pixels.

Once you have this code running, you have to press the spacebar as many times as the number of artags you said to write. In this case, you will press it 4 times until the code stops.

- End of Example 5.1 -

But you are here to apply this code with ros, and you need a place where you can do all this unit work.

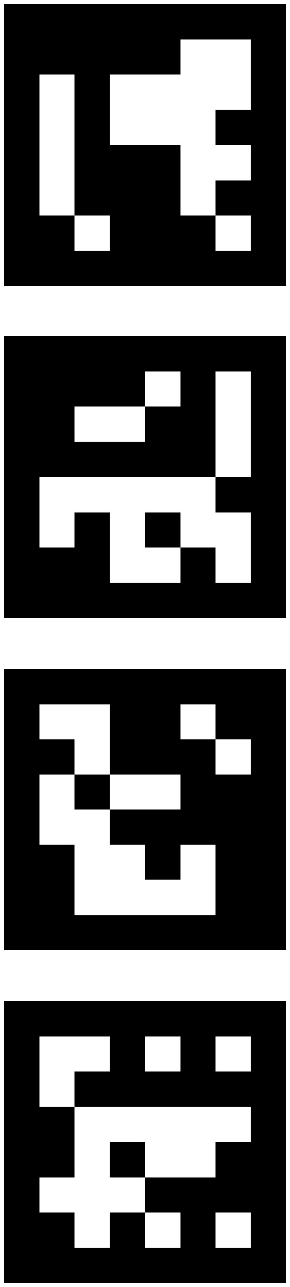
- Exercise 5.1 -

- Create a package inside *catkin_ws/src* and call it **unit5** with rospy as a dependecy
- Create a folder inside the *unit5* package and call it **Tags**
- Create a file inside *catkins_ws/src/unit5/src* and call it artag_generator.py
- Inside this file, run the code you saw before and create your artags.

- End of Exercise 5.1 -

- Expected Behavior for Exercise 5.1 -

Here are the tags that were created.



- End of Expected Behavior for Exercise 3.1 -

Once you create the tags, you can paste them to a document and print them, so you can use them with real images. Also, you can create more than the 3 markers depending on the application you want.

For this course, we are going to do a little example about augmented reality, replacing an area defined by 4 ArUco tags (the coordinates of a rectangle being the center of each one) with any image, so when the camera detects the markers, an image will appear in our camera.

5.2 Artags detection

- Notes -

Don't forget to pay attention to the comments in the examples, you will find important info there.

- End of Notes -

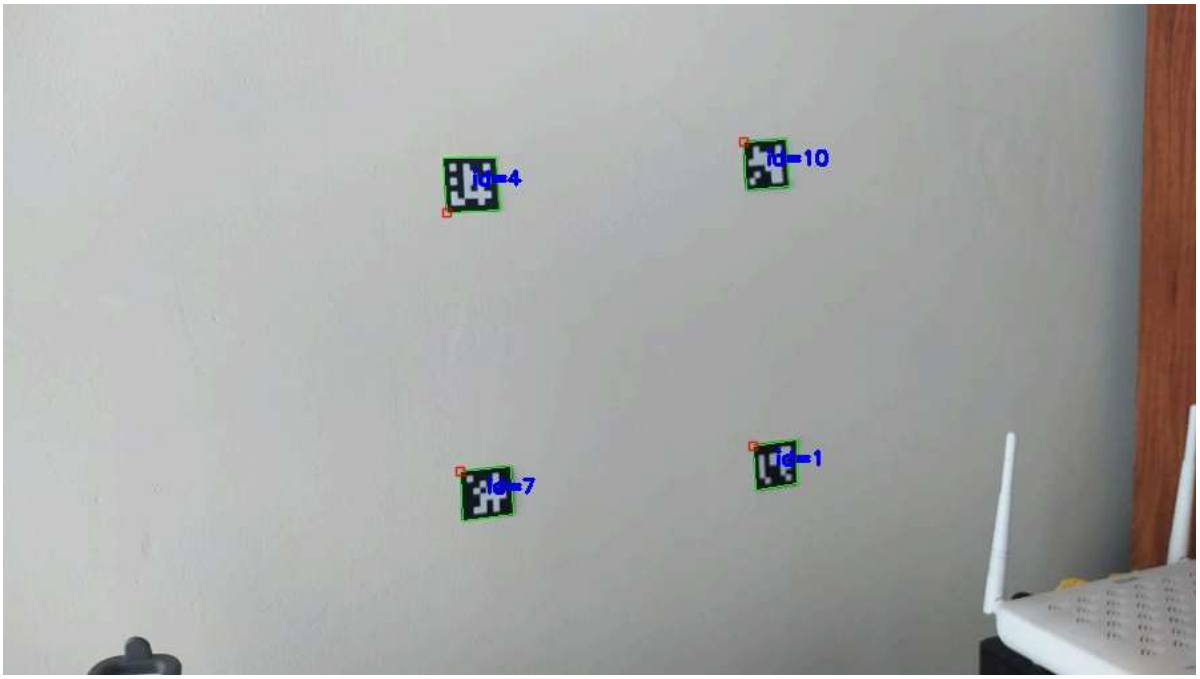
- Example 5.2 -

First of all, we need to recognize the id of the artag that we are using, which is the identity of the artag and that will allow us to work with it.

Then we are going to do an approximation through Polylines. What I mean with this is that we are going to extract the centers of every marker and join them with lines to form a rectangle and that specific area, but we are going to see that later.

In [ ]:

```
import cv2
import numpy as np
from cv2 import aruco

def order_coordinates(pts):

    #Initialize an empty array to save to next values
    coordinates = np.zeros((4, 2), dtype="int")

    s = pts.sum(axis=1)
    coordinates[0] = pts[np.argmin(s)]
    coordinates[2] = pts[np.argmax(s)]

    diff = np.diff(pts, axis=1)
    coordinates[1] = pts[np.argmin(diff)]
    coordinates[3] = pts[np.argmax(diff)]

    return coordinates

image = cv2.imread('/home/user/catkin_ws/src/opencv_for_robotics_images/Unit_5
h,w = image.shape[:2]

image = cv2.resize(image,(int(w*0.7), int(h*0.7)))
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Initialize the aruco Dictionary and its parameters
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()

#Detect the corners and id's in the examples
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict, parame

#First we need to detect the markers itself, so we can later work with the coo
frame_markers = aruco.drawDetectedMarkers(image.copy(), corners, ids)

#Show the markers detected
cv2.imshow('markers',frame_markers)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



As you can see, for this example, we used random id markers. For the detection, we can observe that we have 4 markers with Ids 1,4,7, and 10. For this specific example, the ids don´t really matter because what we really want is the position of them independently of their ID. But, in most cases, the Id provides the info necessary to proccess something.

- End of Example 5.2 -

Well let's see this with our robot.

- Exercise 5.2 -

Do the following steps in order to complete the exercise:

In the simulation we have already provided a wall with AR tags

So let's continue with these steps:

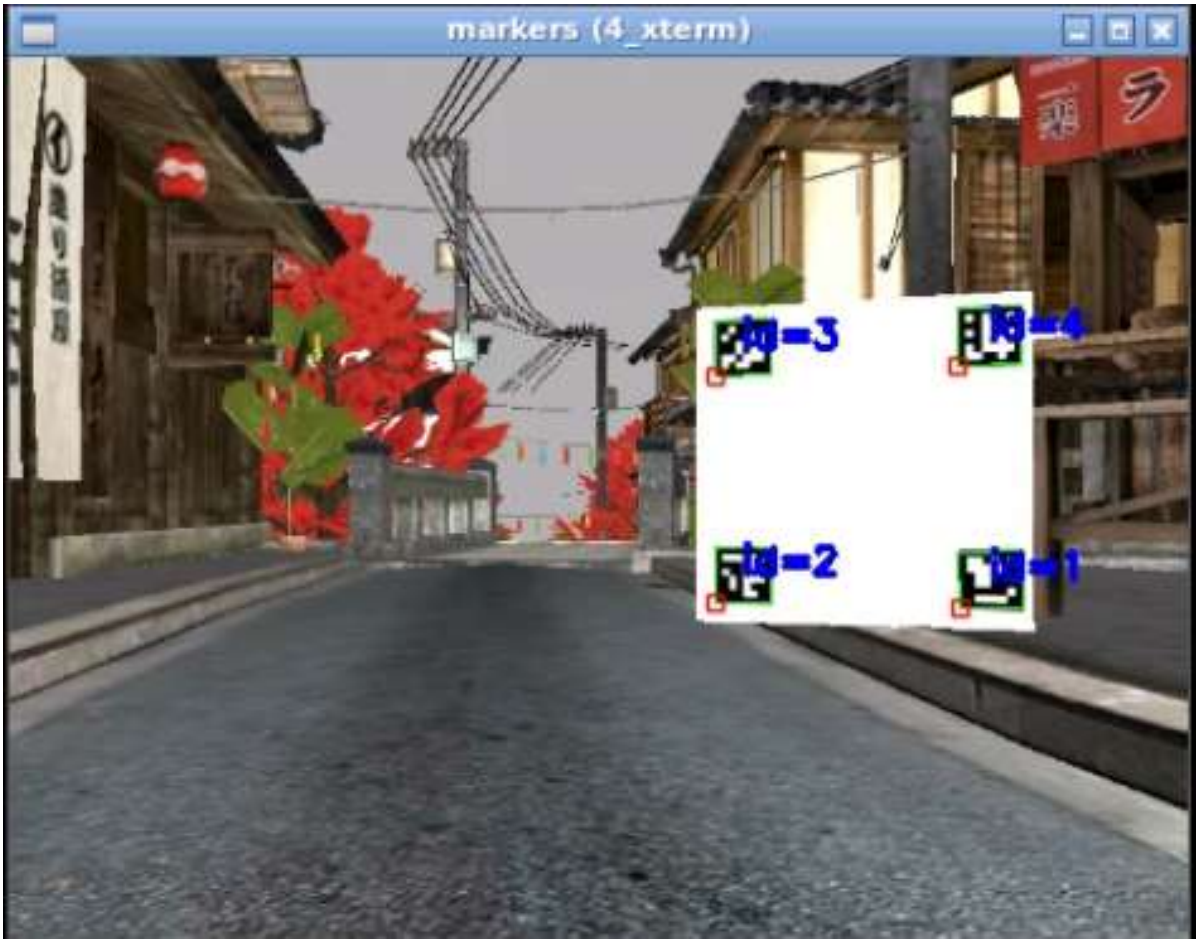
- Create a file inside `catkin_ws/src/unit5/src` and call it **exercise5_2.py**
- **Make it executable** with the command:

In [ ]:
```
chmod +x exercise5_2.py
```

- Using the structure of ROS that you already know, implement the code of the previous example in order to see the Ids of the artags in the simulation.
- Create a launch file inside `catkin_ws/src/unit5/launch` and call it exercise5_2.launch and launch this file.

- End of Exercise 5.2 -

- Expected Behavior for Exercise 5.2 -



- End of Expected Behavior for Exercise 5.2 -

- Notes -

- Remember that being a simulation, the results can get noisy and not be as accurate as in a real image
- If the Rosbot is not in the correct position, you can move it using this command: **roslaunch rosbot_navigation rosbot_teleop.launch**

- End of Notes -

5.3 Extract the center of artag

- Notes -

Don't forget to pay attention to the comments in the examples, you will find important info there.

- End of Notes -

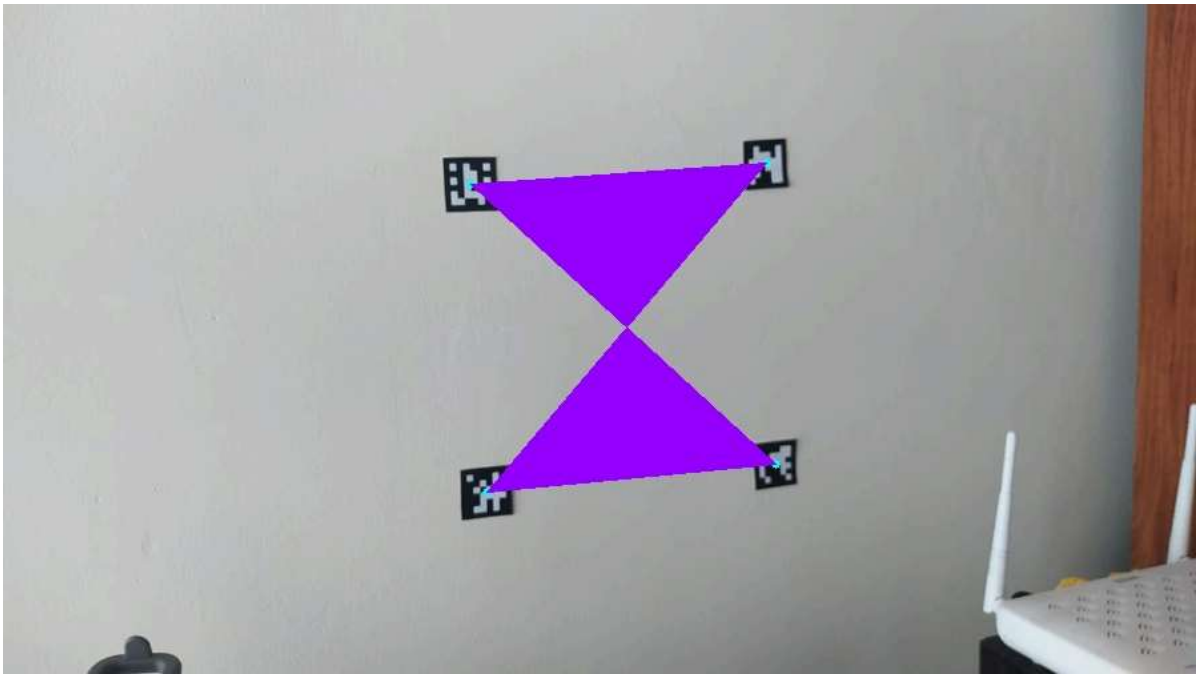
Once we get the detection, we will be able to calculate the center of them. With the algorithm, we created a corners array where the coordinates of the 16 corners in the image are stored.

- Example 5.3 -

In [ ]:

```
import cv2
import numpy as np
from cv2 import aruco

def order_coordinates(pts):

    #Initialize an empty array to save to next values
    coordinates = np.zeros((4, 2), dtype="int")

    s = pts.sum(axis=1)
    coordinates[0] = pts[np.argmin(s)]
    coordinates[2] = pts[np.argmax(s)]

    diff = np.diff(pts, axis=1)
    coordinates[1] = pts[np.argmin(diff)]
    coordinates[3] = pts[np.argmax(diff)]

    return coordinates

image = cv2.imread('/home/user/catkin_ws/src/opencv_for_robotics_images/Unit_5
h,w = image.shape[:2]

image = cv2.resize(image,(int(w*0.7), int(h*0.7)))
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Initialize the aruco Dictionary and its parameters
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()

#Detect the corners and id's in the examples
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict, parame

#First we need to detect the markers itself, so we can later work with the coo
frame_markers = aruco.drawDetectedMarkers(image.copy(), corners, ids)

#Show the markers detected
cv2.imshow('markers',frame_markers)
#Initialize an empty list for the coordinates
params = []

for i in range(len(ids)):

    #Catch the corners of each tag
    c = corners[i][0]
```

```
    #Draw a circle in the center of each detection
    cv2.circle(image,(int(c[:, 0].mean()), int(c[:, 1].mean())), 3, (255,255,0

    #Save thhe center coordinates for each tag
    params.append((int(c[:, 0].mean()), int(c[:, 1].mean())))

#Convert the coordinates list to an array
params = np.array(params)

#Draw a polygon with the coordinates
cv2.drawContours(image,[params],-1 ,(255,0,150),-1)

cv2.imshow('no_conversion',image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



- End of Example 5.3 -

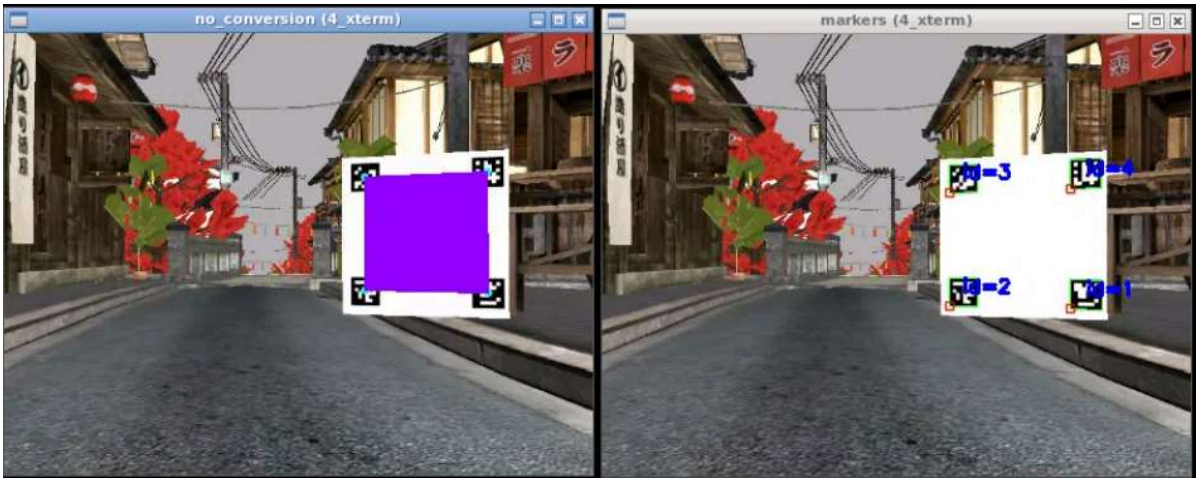
Let's see if you can identify with the artags in the simulation.

- Exercise 5.3 -

- Create a file inside catkin_ws/src/unit5/src and call it exercise5_3.py
- Using the structure of ROS that you already know, implement the code of the prior example in order to see how it interacts with the simulation
- Create a launch file inside catkin_ws/src/unit5/launch and call it exercise5_3.launch and launch this file.

- End of Exercise 5.3 -

- Expected Behavior for Exercise 5.3 -



- End of Expected Behavior for Exercise 5.3 -

- Notes -

- Remember that being a simulation, the results can get noisy and not be as accurate as in a real image
- If the Rosbot is not in a correct position, you can move it using this command: **roslaunch rosbot_navigation rosbot_teleop.launch**

- End of Notes -

5.4 Order the centers correctly

- Notes -

Don't forget to pay attention to the comments in the examples, you will find important info there.

- End of Notes -

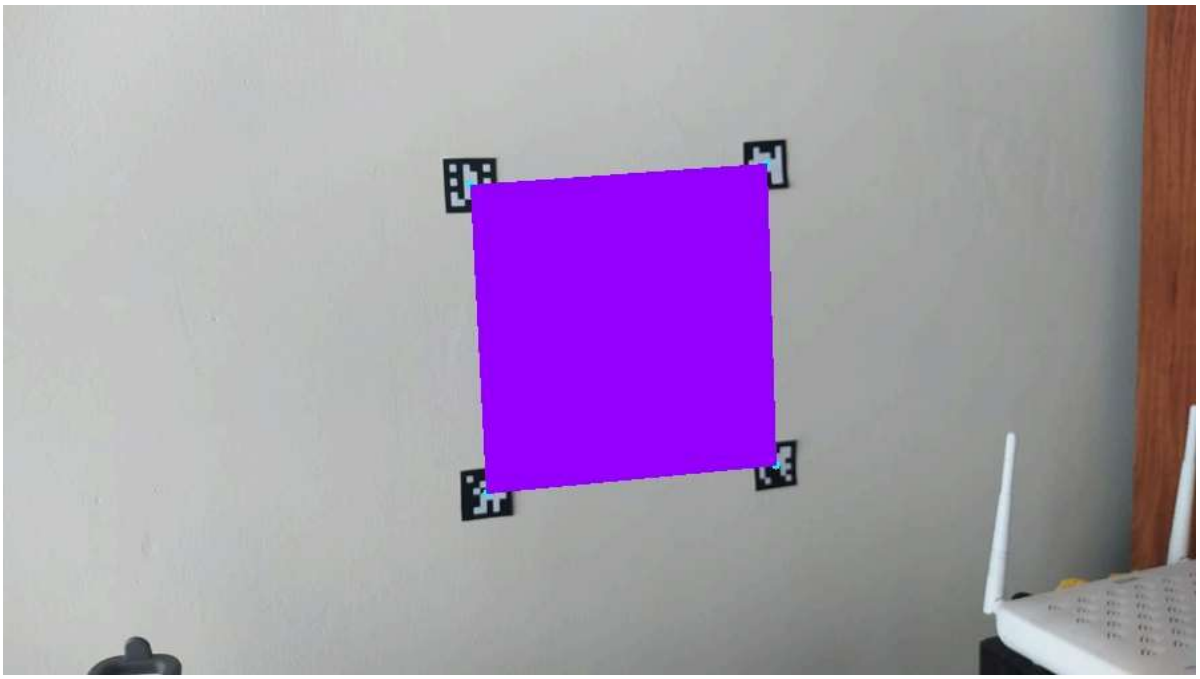
As can you see in the image in example 5.3, the polygon drawn in the image is very sensitive to the order of the parameters given. In this moment, our parameters list is in the order 7, 1, 4, 10, so it will draw this strange figure. So we need to order our array so the points are correlative, like for example 4, 10, 1, 7.

- Example 5.4 -

In [ ]:

```
import cv2
import numpy as np
from cv2 import aruco

def order_coordinates(pts):

    #Initialize an empty array to save to next values
    coordinates = np.zeros((4, 2), dtype="int")

    s = pts.sum(axis=1)
    coordinates[0] = pts[np.argmin(s)]
    coordinates[2] = pts[np.argmax(s)]

    diff = np.diff(pts, axis=1)
    coordinates[1] = pts[np.argmin(diff)]
    coordinates[3] = pts[np.argmax(diff)]

    return coordinates

image = cv2.imread('/home/user/catkin_ws/src/opencv_for_robotics_images/Unit_5
h,w = image.shape[:2]

image = cv2.resize(image,(int(w*0.7), int(h*0.7)))
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Initialize the aruco Dictionary and its parameters
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()

#Detect the corners and id's in the examples
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict, parame

#First we need to detect the markers itself, so we can later work with the coo
frame_markers = aruco.drawDetectedMarkers(image.copy(), corners, ids)

#Show the markers detected
cv2.imshow('markers',frame_markers)

#Initialize an empty list for the coordinates
params = []

for i in range(len(ids)):

    #Catch the corners of each tag
    c = corners[i][0]
```

```
    #Draw a circle in the center of each detection
    cv2.circle(image,(int(c[:, 0].mean()), int(c[:, 1].mean())), 3, (255,255,0

    #Save thhe center coordinates for each tag
    params.append((int(c[:, 0].mean()), int(c[:, 1].mean())))

#Convert the coordinates list to an array
params = np.array(params)

#Draw a polygon with the coordinates
cv2.drawContours(image,[params],-1 ,(255,0,150),-1)


if(len(params)>=4):
    #Sort the coordinates
    params = order_coordinates(params)

#Draw the polygon with the sorted coordinates
cv2.drawContours(image,[params],-1 ,(255,0,150),-1)

cv2.imshow('detection',image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```



- End of Example 5.4 -

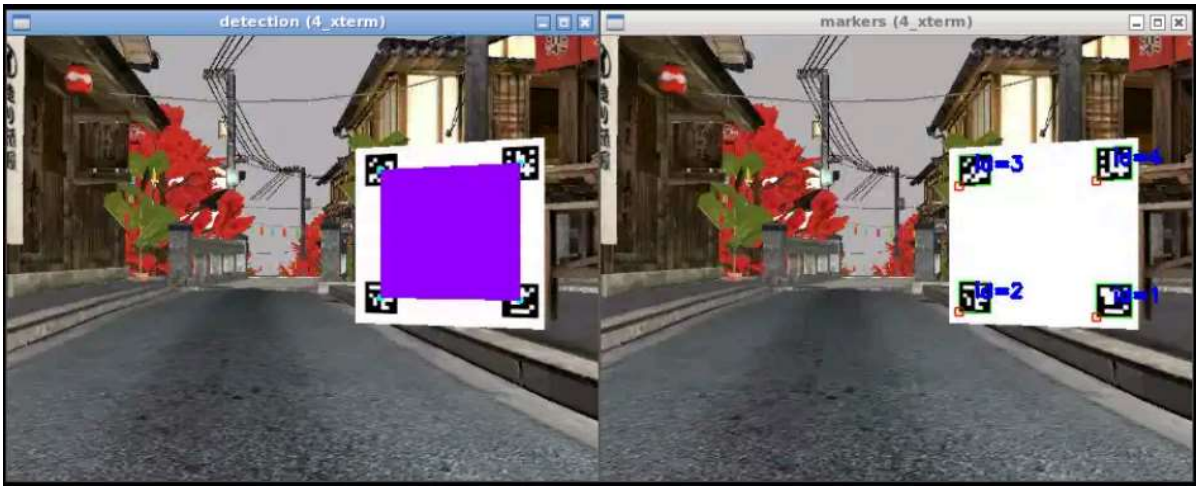
Now it's time to order that in our simulation, so let's do it.

- Exercise 5.4 -

- Create a file inside catkin_ws/src/unit5/src and call it exercise5_4.py
- Using the structure of ROS that you already know, implement the code of the previous example in order to see how this interacts with the simulation
- Create a launch file inside catkin_ws/src/unit5/launch and call it exercise5_4.launch and launch this file.

- End of Exercise 5.4 -

- Expected Behavior for Exercise 5.4 -



- End of Expected Behavior for Exercise 5.4 -

- Notes -

- Remember that being a simulation, the results can get noisy and not be as accurate as in a real image
- If the Rosbot is not in the correct position, you can move it using this command:

In [ ]:
```
roslaunch rosbot_navigation rosbot_teleop.launch
```

- End of Notes -

5.5 Area substraction

- Notes -

Don't forget to pay attention to the comments in the examples, you will find important info there.

- End of Notes -

Now that we have an idea of the extraction of the coordinates to draw a rectangle in a specific position, we can use this approximation to replace this purple area with a desired virtual media that we want, like a photo of the earth! And how can we do that? Well, we zre going to substract this purple area in order to paint an image in this area. Let's see.

- Example 5.5 -

```
import cv2
import numpy as np
from cv2 import aruco

def order_coordinates(pts, var):
    coordinates = np.zeros((4,2),dtype="int")

    if(var):
        #Parameters sort model 1
        s = pts.sum(axis=1)
        coordinates[0] = pts[np.argmin(s)]
        coordinates[3] = pts[np.argmax(s)]

        diff = np.diff(pts, axis=1)
        coordinates[1] = pts[np.argmin(diff)]
        coordinates[2] = pts[np.argmax(diff)]

    else:
        #Parameters sort model 2
        s = pts.sum(axis=1)
        coordinates[0] = pts[np.argmin(s)]
        coordinates[2] = pts[np.argmax(s)]

        diff = np.diff(pts, axis=1)
        coordinates[1] = pts[np.argmin(diff)]
        coordinates[3] = pts[np.argmax(diff)]

    return coordinates

image = cv2.imread('./Examples/a1.jpg')
h, w = image.shape[:2]

image = cv2.resize(image,(int(w*0.7), int(h*0.7)))
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Initialize the aruco Dictionary and its parameters
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()

#Detect the corners and ids in the images
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict, parameters=
parameters)

#Initialize an empty list for the coordinates
params = []

for i in range(len(ids)):
```

```
        #Catch the corners of each tag
        c = corners[i][0]

        #Draw a circle in the center of each detection
        cv2.circle(image,(int(c[:, 0].mean()), int(c[:, 1].mean())), 3, (255,255,0), -
    1)

        #Save the coordinates of the center of each tag
        params.append((int(c[:, 0].mean()), int(c[:, 1].mean())))

    #Transfom the coordinates list to an array
    params = np.array(params)
```

In the previous code, we sorted the coordinates to a specific order. For this part, we need this order and also a second sort changing the position 2 and 3 of the array. This is done just for the algorithm to work well. When we work with the convexPoly, it works with a different order than the warped image with the homography. This variation can be seen in the function "order coordinates".

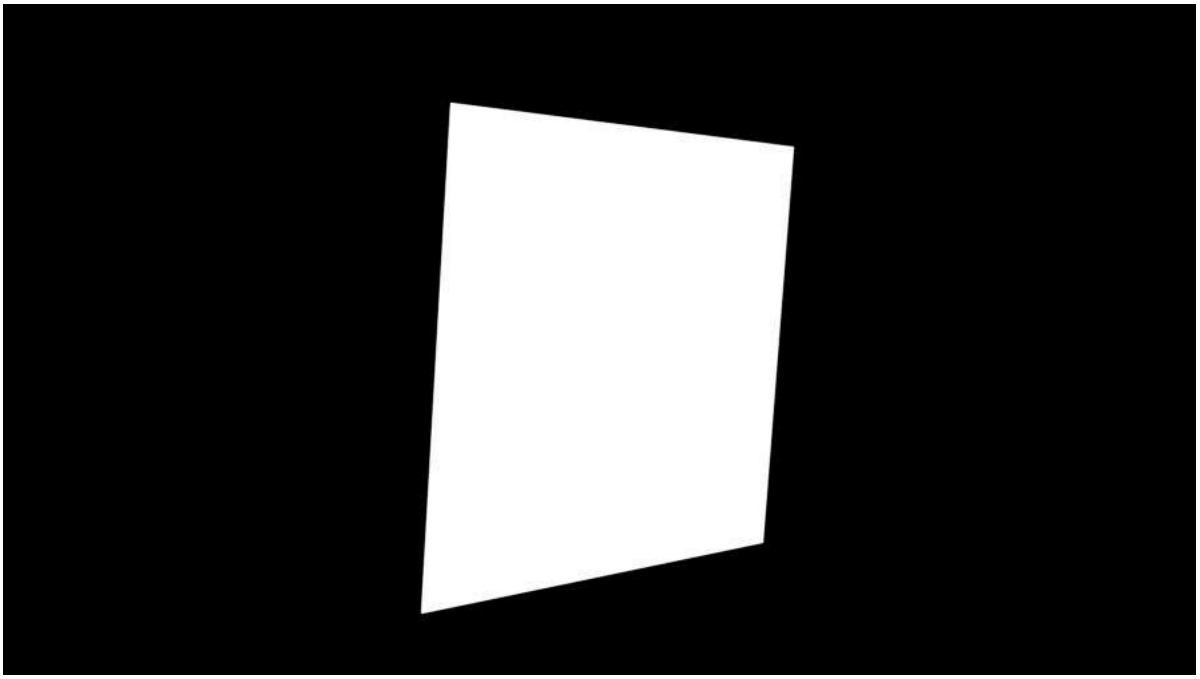
```
    if(len(params)>=4):
        #Sort model 1
        params = order_coordinates(params,False)

        #Sort Model 2
        params_2 = order_coordinates(params,True)

    #Here we are going to read the image we want to overlap
    paint = cv2.imread('./Examples/earth.jpg')
    height, width = paint.shape[:2]

    #We extract the coordinates of this new image which are basically the full sized im
    age
    coordinates = np.array([[0,0],[width,0],[0,height],[width,height]])

    #Just like in chapter 3 we will find a perspective between the planes
    #Homography will help us with the image transformations
    hom, status = cv2.findHomography(coordinates, params_2)

    #We will save the warped image in a dark space same with the same size as the main
    image
    warped_image = cv2.warpPerspective(paint, hom, (int(w*0.7), int(h*0.7)))

    #We create a black mask to do the image operations
    mask = np.zeros([int(h*0.7), int(w*0.7),3], dtype=np.uint8)

    #To the black mask we will replace the area described by the ar tags with white
    cv2.fillConvexPoly(mask, np.int32([params]), (255, 255, 255), cv2.LINE_AA)
    cv2.imshow('black mask',mask)
```

Al the code until this part should look like this one.

In [ ]:

```
import cv2
import numpy as np
from cv2 import aruco

def order_coordinates(pts, var):
    coordinates = np.zeros((4,2),dtype="int")

    if(var):
        #Parameters sort model 1
        s = pts.sum(axis=1)
        coordinates[0] = pts[np.argmin(s)]
        coordinates[3] = pts[np.argmax(s)]

        diff = np.diff(pts, axis=1)
        coordinates[1] = pts[np.argmin(diff)]
        coordinates[2] = pts[np.argmax(diff)]

    else:
        #Parameters sort model 2
        s = pts.sum(axis=1)
        coordinates[0] = pts[np.argmin(s)]
        coordinates[2] = pts[np.argmax(s)]

        diff = np.diff(pts, axis=1)
        coordinates[1] = pts[np.argmin(diff)]
        coordinates[3] = pts[np.argmax(diff)]

    return coordinates

image = cv2.imread('/home/user/catkin_ws/src/opencv_for_robotics_images/Unit_5
h, w = image.shape[:2]

image = cv2.resize(image,(int(w*0.7), int(h*0.7)))
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Initialize the aruco Dictionary and its parameters
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()

#Detect the corners and ids in the images
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict, parame

#Initialize an empty list for the coordinates
params = []
```

```
for i in range(len(ids)):

    #Catch the corners of each tag
    c = corners[i][0]

    #Draw a circle in the center of each detection
    cv2.circle(image,(int(c[:, 0].mean()), int(c[:, 1].mean())), 3, (255,255,0

    #Save the coordinates of the center of each tag
    params.append((int(c[:, 0].mean()), int(c[:, 1].mean())))

#Transfom the coordinates list to an array
params = np.array(params)
if(len(params)>=4):
    #Sort model 1
    params = order_coordinates(params,False)

    #Sort Model 2
    params_2 = order_coordinates(params,True)

#Her we are going to read the image we want to overlap
paint = cv2.imread('/home/user/catkin_ws/src/opencv_for_robotics_images/Unit_5
height, width = paint.shape[:2]

#We extract the coordinates of this new image which are basically the full siz
coordinates = np.array([[0,0],[width,0],[0,height],[width,height]])

#Just like in chapter 3 we will find a perspective between the planes
#Homography will help us with the image transformations
hom, status = cv2.findHomography(coordinates, params_2)

#We will save the warped image in a dark space same with the same size as the
warped_image = cv2.warpPerspective(paint, hom, (int(w*0.7), int(h*0.7)))

#We create a black mask to do the image operations
mask = np.zeros([int(h*0.7), int(w*0.7),3], dtype=np.uint8)

#To the black mask we will replace the area described by the ar tags with whit
cv2.fillConvexPoly(mask, np.int32([params]), (255, 255, 255), cv2.LINE_AA)
cv2.imshow('black mask',mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



We will calculate the difference between the original image and the mask to obtain a black space (no color) in the desired area, adding the following lines to substract and show the result

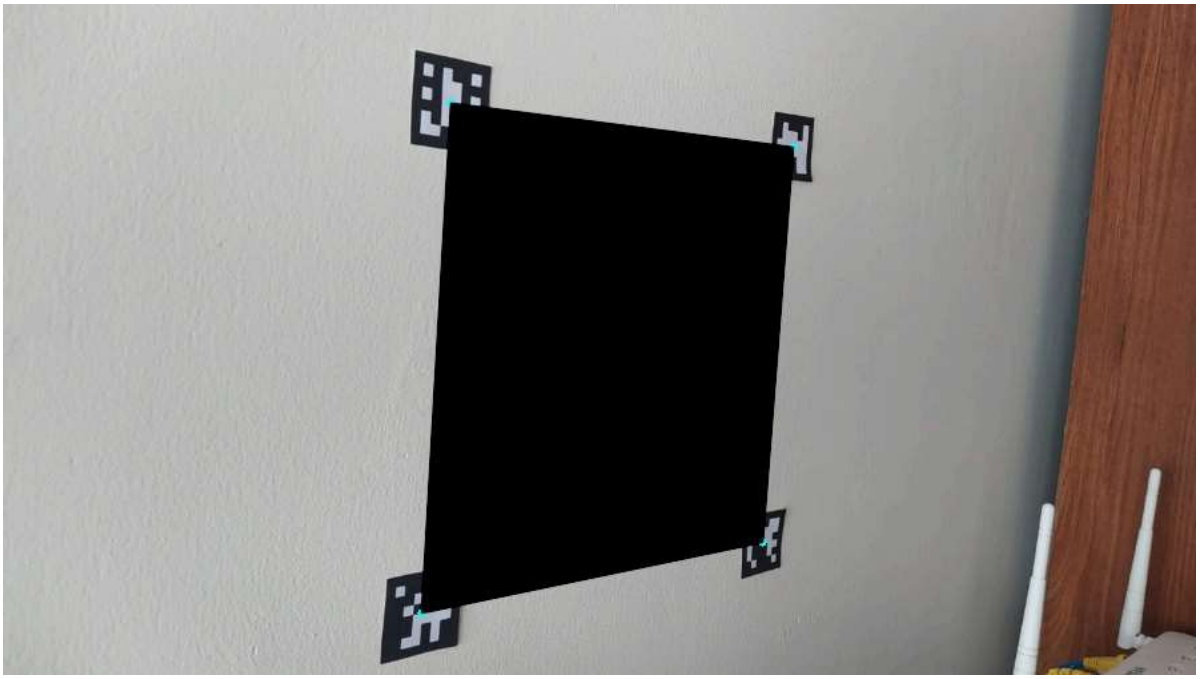
```
substraction = cv2.subtract(image,mask)
cv2.imshow('substraction',substraction)
```

But where will you add it? Well, at the end, like in this code you have below.

In [ ]:

```
import cv2
import numpy as np
from cv2 import aruco

def order_coordinates(pts, var):
    coordinates = np.zeros((4,2),dtype="int")

    if(var):
        #Parameters sort model 1
        s = pts.sum(axis=1)
        coordinates[0] = pts[np.argmin(s)]
        coordinates[3] = pts[np.argmax(s)]

        diff = np.diff(pts, axis=1)
        coordinates[1] = pts[np.argmin(diff)]
        coordinates[2] = pts[np.argmax(diff)]

    else:
        #Parameters sort model 2
        s = pts.sum(axis=1)
        coordinates[0] = pts[np.argmin(s)]
        coordinates[2] = pts[np.argmax(s)]

        diff = np.diff(pts, axis=1)
        coordinates[1] = pts[np.argmin(diff)]
        coordinates[3] = pts[np.argmax(diff)]

    return coordinates

image = cv2.imread('/home/user/catkin_ws/src/opencv_for_robotics_images/Unit_5
h, w = image.shape[:2]

image = cv2.resize(image,(int(w*0.7), int(h*0.7)))
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Initialize the aruco Dictionary and its parameters
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()

#Detect the corners and ids in the images
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict, parame

#Initialize an empty list for the coordinates
params = []
```

```
for i in range(len(ids)):

    #Catch the corners of each tag
    c = corners[i][0]

    #Draw a circle in the center of each detection
    cv2.circle(image,(int(c[:, 0].mean()), int(c[:, 1].mean())), 3, (255,255,0

    #Save the coordinates of the center of each tag
    params.append((int(c[:, 0].mean()), int(c[:, 1].mean())))

#Transfom the coordinates list to an array
params = np.array(params)
if(len(params)>=4):
    #Sort model 1
    params = order_coordinates(params,False)

    #Sort Model 2
    params_2 = order_coordinates(params,True)

#Her we are going to read the image we want to overlap
paint = cv2.imread('/home/user/catkin_ws/src/opencv_for_robotics_images/Unit_5
height, width = paint.shape[:2]

#We extract the coordinates of this new image which are basically the full siz
coordinates = np.array([[0,0],[width,0],[0,height],[width,height]])

#Just like in chapter 3 we will find a perspective between the planes
#Homography will help us with the image transformations
hom, status = cv2.findHomography(coordinates, params_2)

#We will save the warped image in a dark space same with the same size as the
warped_image = cv2.warpPerspective(paint, hom, (int(w*0.7), int(h*0.7)))

#We create a black mask to do the image operations
mask = np.zeros([int(h*0.7), int(w*0.7),3], dtype=np.uint8)

#To the black mask we will replace the area described by the ar tags with whit
cv2.fillConvexPoly(mask, np.int32([params]), (255, 255, 255), cv2.LINE_AA)
cv2.imshow('black mask',mask)
substraction = cv2.subtract(image,mask)
cv2.imshow('substraction',substraction)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



- End of Example 5.5 -

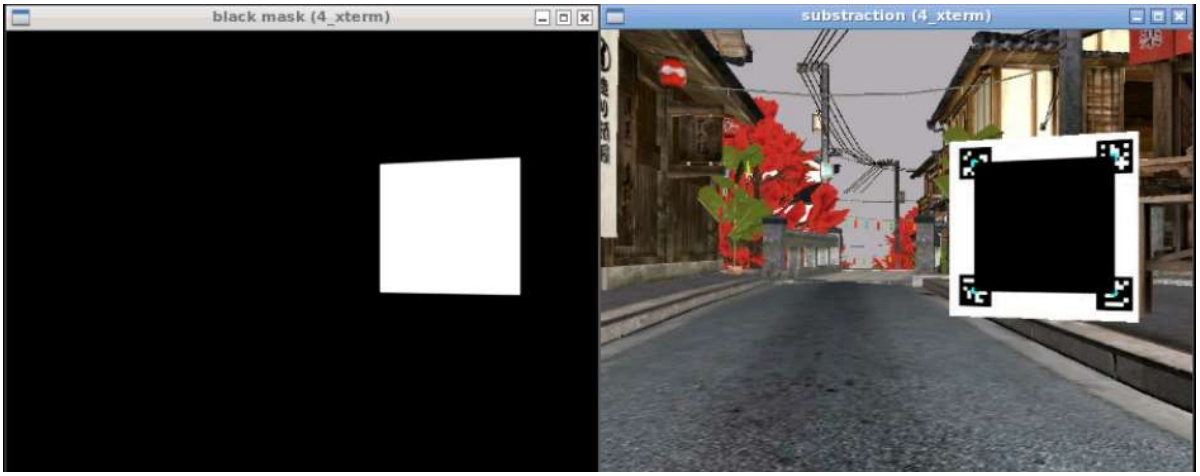
Now apply both things and see how you can do it with the simulation.

- Exercise 5.5 -

- Create a file inside catkin_ws/src/unit5/src and call it exercise5_5.py
- Using the structure of ROS that you already know, implement the code of the previous example in order to see how this interacts with the simulation
- Create a launch file inside catkin_ws/src/unit5/launch and call it exercise5_5.launch and launch this file.

- End of Exercise 5.5 -

- Expected Behavior for Exercise 5.5 -



- End of Expected Behavior for Exercise 3.1 -

- Notes -

- Remember that being a simulation, the results can get noisy and not be as accurate as in a real image
- If the Rosbot is not in the correct position, you can move it using this command: **roslaunch rosbot_navigation rosbot_teleop.launch**

- End of Notes -

5.6 Painting in Area

Once we have the area color-free, we can add the warped image through image addition, just adding the following lines to the code.

```
addition = cv2.add(warped_image,substraction)

cv2.imshow('detection',addition)
```

like this

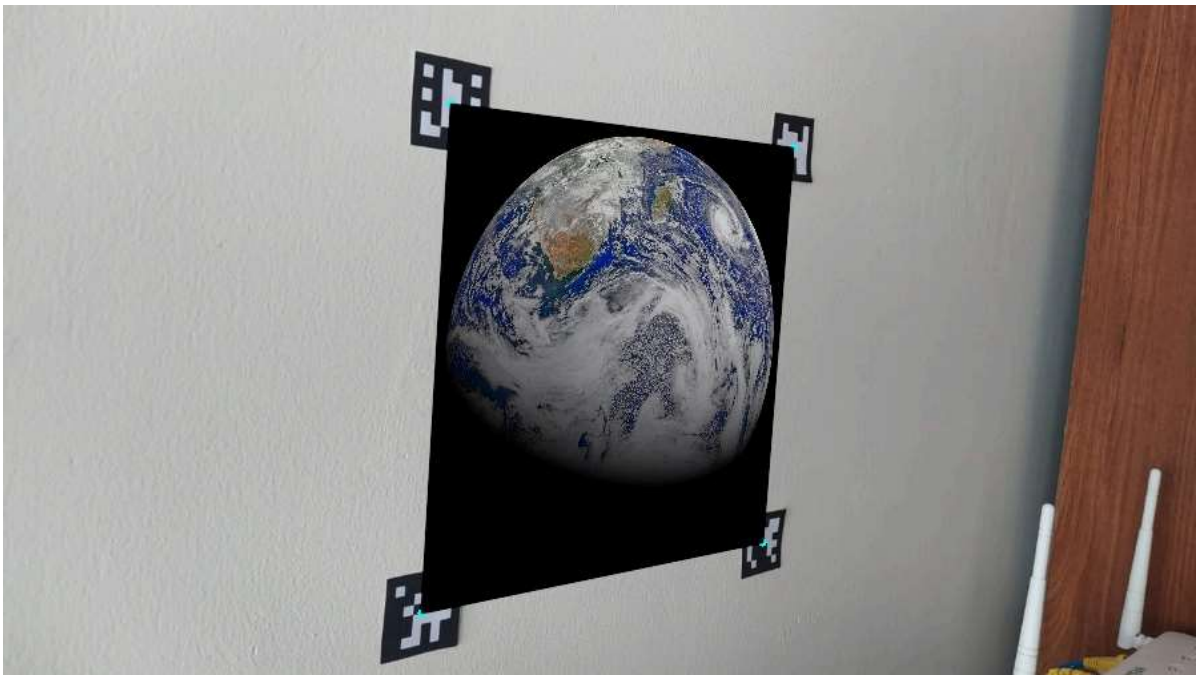
- Example 5.6 -

In [ ]:

```
import os
import cv2
import numpy as np
from cv2 import aruco
print(cv2.__file__)

def order_coordinates(pts, var):
    coordinates = np.zeros((4,2),dtype="int")

    if(var):
        #Parameters sort model 1
        s = pts.sum(axis=1)
        coordinates[0] = pts[np.argmin(s)]
        coordinates[3] = pts[np.argmax(s)]

        diff = np.diff(pts, axis=1)
        coordinates[1] = pts[np.argmin(diff)]
        coordinates[2] = pts[np.argmax(diff)]

    else:
        #Parameters sort model 2
        s = pts.sum(axis=1)
        coordinates[0] = pts[np.argmin(s)]
        coordinates[2] = pts[np.argmax(s)]

        diff = np.diff(pts, axis=1)
        coordinates[1] = pts[np.argmin(diff)]
        coordinates[3] = pts[np.argmax(diff)]

    return coordinates

image = cv2.imread('/home/user/catkin_ws/src/opencv_for_robotics_images/Unit_5
h, w = image.shape[:2]

image = cv2.resize(image,(int(w*0.7), int(h*0.7)))
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Initialize the aruco Dictionary and its parameters
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()

#Detect the corners and ids in the images
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict, parame

#Initialize an empty list for the coordinates
```

```
params = []

for i in range(len(ids)):

    #Catch the corners of each tag
    c = corners[i][0]

    #Draw a circle in the center of each detection
    cv2.circle(image,(int(c[:, 0].mean()), int(c[:, 1].mean())), 3, (255,255,0

    #Save the coordinates of the center of each tag
    params.append((int(c[:, 0].mean()), int(c[:, 1].mean())))

#Transfom the coordinates list to an array
params = np.array(params)
if(len(params)>=4):
    #Sort model 1
    params = order_coordinates(params,False)

    #Sort Model 2
    params_2 = order_coordinates(params,True)

#Her we are going to read the image we want to overlap
paint = cv2.imread('/home/user/catkin_ws/src/opencv_for_robotics_images/Unit_5
height, width = paint.shape[:2]

#We extract the coordinates of this new image which are basically the full siz
coordinates = np.array([[0,0],[width,0],[0,height],[width,height]])

#Just like in chapter 3 we will find a perspective between the planes
#Homography will help us with the image transformations
hom, status = cv2.findHomography(coordinates, params_2)

#We will save the warped image in a dark space same with the same size as the
warped_image = cv2.warpPerspective(paint, hom, (int(w*0.7), int(h*0.7)))

#We create a black mask to do the image operations
mask = np.zeros([int(h*0.7), int(w*0.7),3], dtype=np.uint8)

#To the black mask we will replace the area described by the ar tags with whit
cv2.fillConvexPoly(mask, np.int32([params]), (255, 255, 255), cv2.LINE_AA)
cv2.imshow('black mask',mask)
substraction = cv2.subtract(image,mask)
cv2.imshow('substraction',substraction)
addition = cv2.add(warped_image,substraction)
cv2.imshow('detection',addition)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```



- End of Example 5.6 -

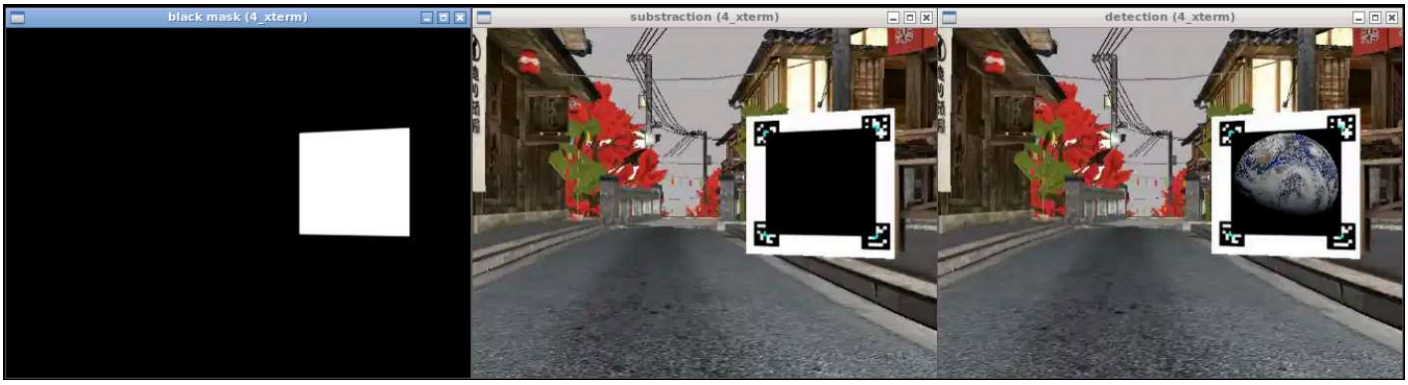
We want to see the earth in our simulation, too. Let's try it! It's gonna be awesome!!

- Exercise 5.6 -

- Create a file inside catkin_ws/src/unit5/src and call it exercise5_6.py
- Using the structure of ROS that you already know, implement the code of the previous example in order to see how this interacts with the simulation
- Create a launch file inside catkin_ws/src/unit5/launch and call it exercise5_6.launch and launch this file.

- End of Exercise 5.6 -

- Expected Behavior for Exercise 5.6 -



- End of Expected Behavior for Exercise 5.6 -

- Notes -

- Remember that being a simulation, the results can get noisy and not be as accurate as in a real image
- If the Rosbot is not in the correct position, you can move it using this command:

In [ ]:
```
roslaunch rosbot_navigation rosbot_teleop.launch
```

- End of Notes -

WE HOPE YOU LEARNED A LOT! YOU ARE READY FOR THE FINAL PROJECT OF THE COURSE. GOOD LUCK!