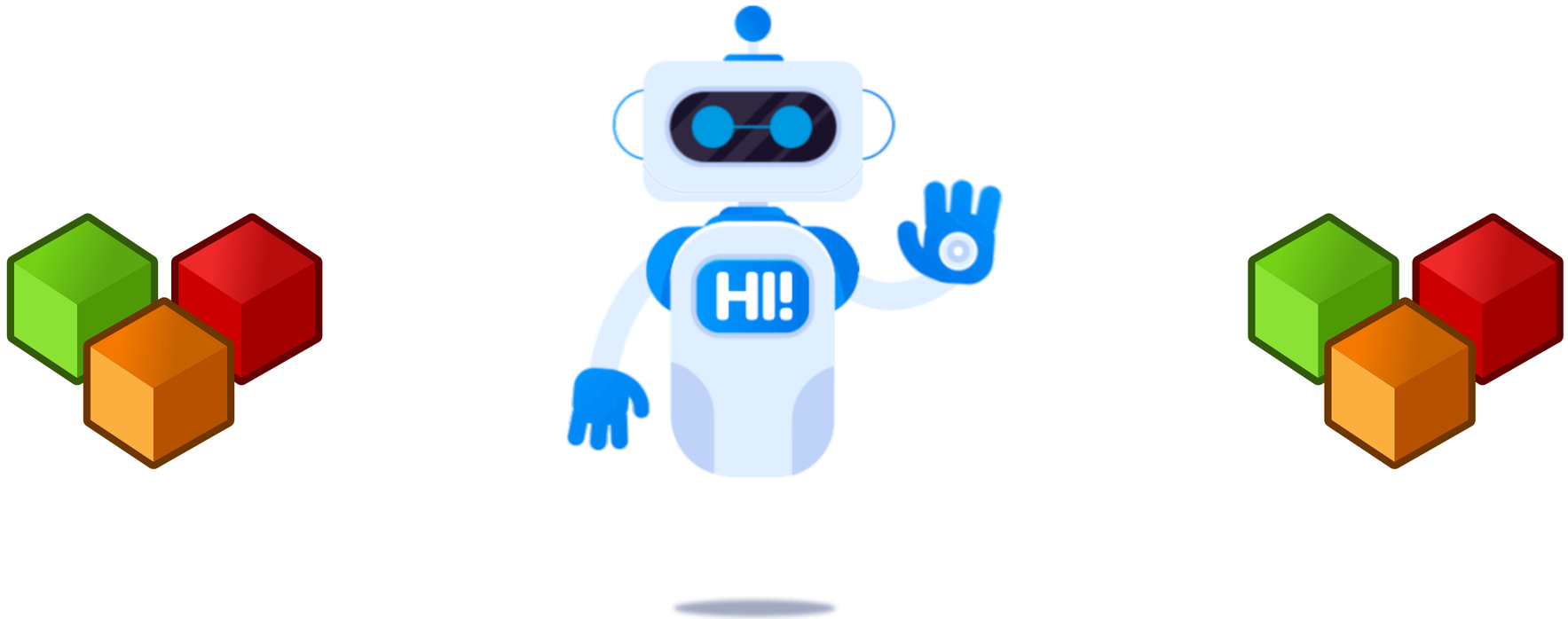
BELLEI | DE VITIS | SEVERI | VIGLIANISI



VISIO.CO.S.M.

Vision Controlled Smart Manipulator

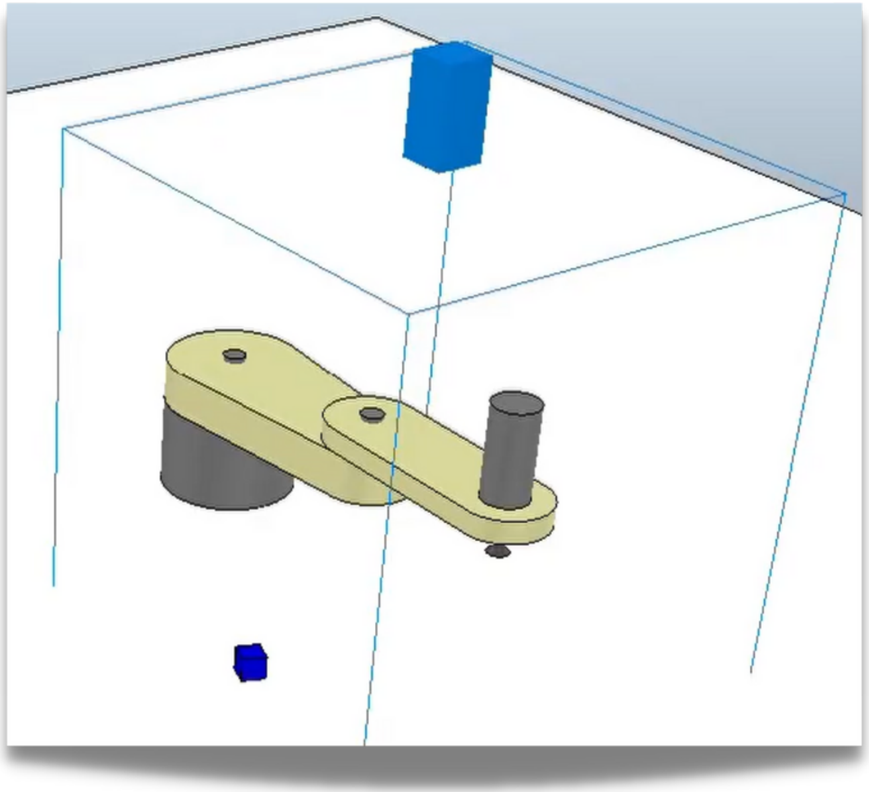
A project developed for the course Smart Robotics - Unimore 2022

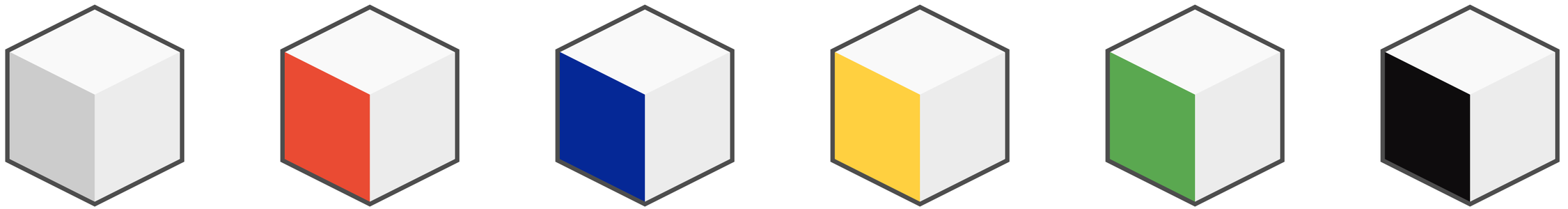
Program

- The problem
- Tools used
- Possible approaches
- Possible solutions
- Implementation
- Tests
- Results
- Conclusions

The problem

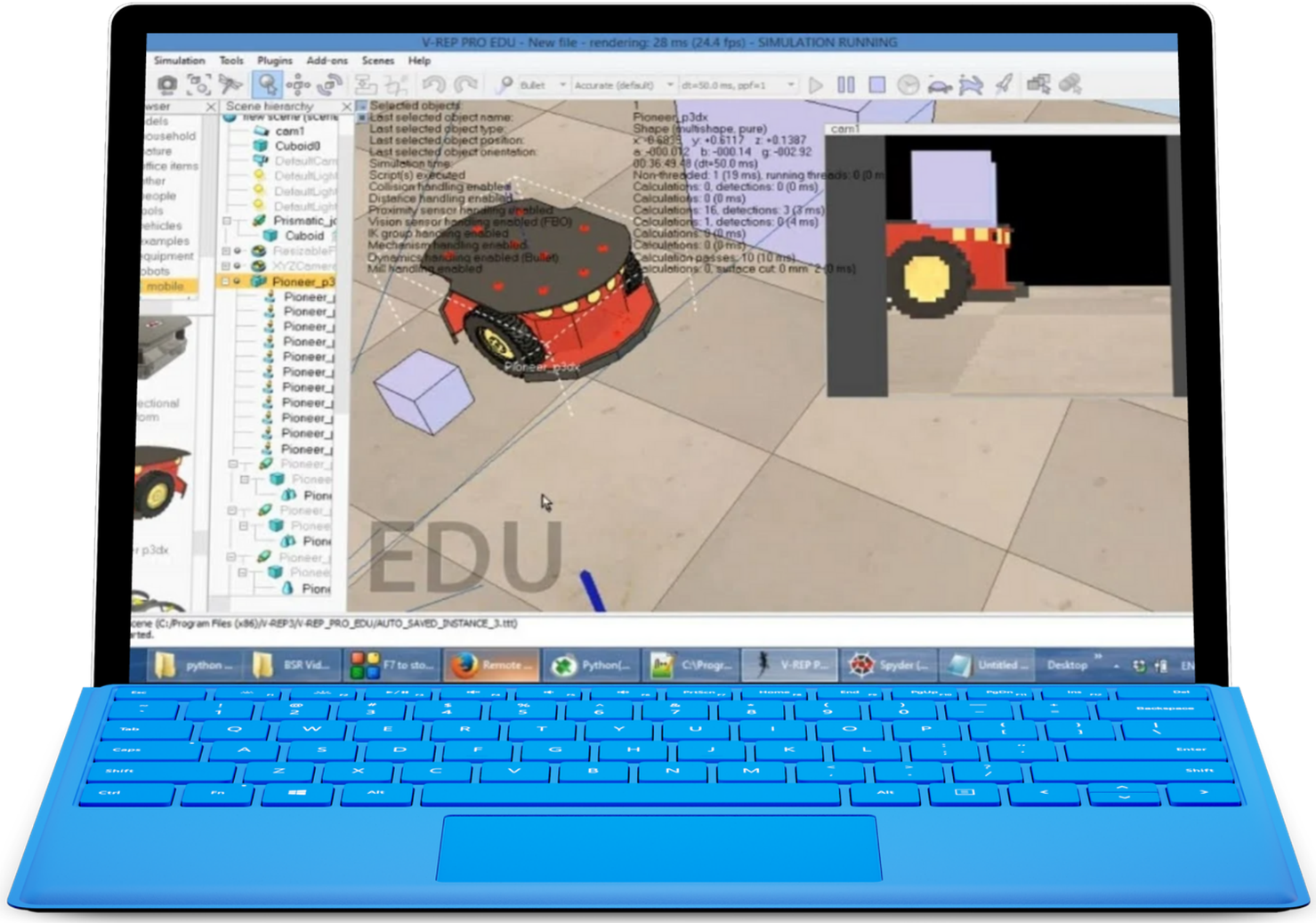
Pick and place with human avoidance





Tools used

CoppeliaSim



Jupyter Python Notebook

python™

Simulation and code interaction

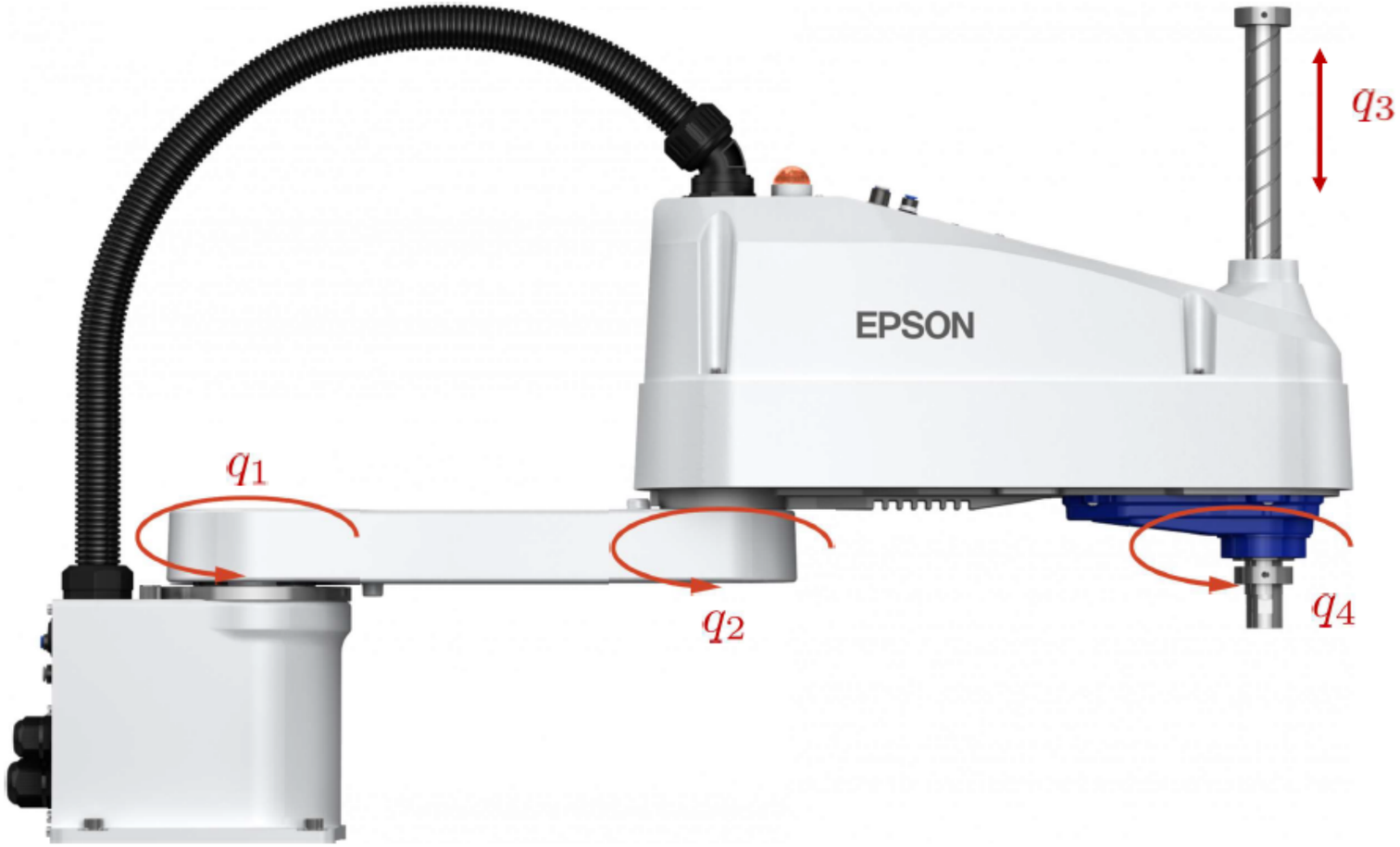
CoppeliaSim

An application programming interface (API) is a way for two or more computer programs to communicate with each other. It is a type of software interface, offering a service to other pieces of software

The manipulator

Selective Compliance Assembly Robot Arm

SCARA joints

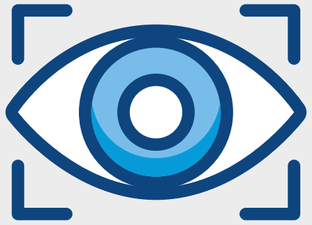


Possible approaches

The alternatives

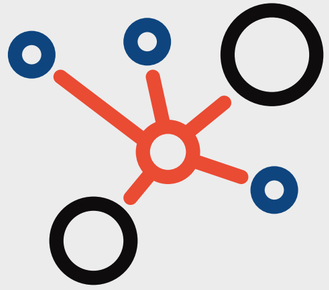
1

CLASSICAL APPROACH



2

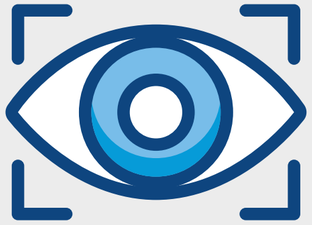
DEEP APPROACH



Pros and cons of classical approach

1

CLASSICAL APPROACH

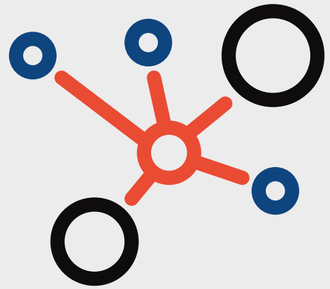


- **Explainable**
- **Predictable**
- **Easy to train and test**
- **Limited power**
- **Works better under hard constraints**

Pros and cons of deep approach

2

DEEP APPROACH



- **Really powerful**
- **Not fully explainable**
- **Needs a lot of data**
- **Needs powerful hardware**

Why we chose a classical approach

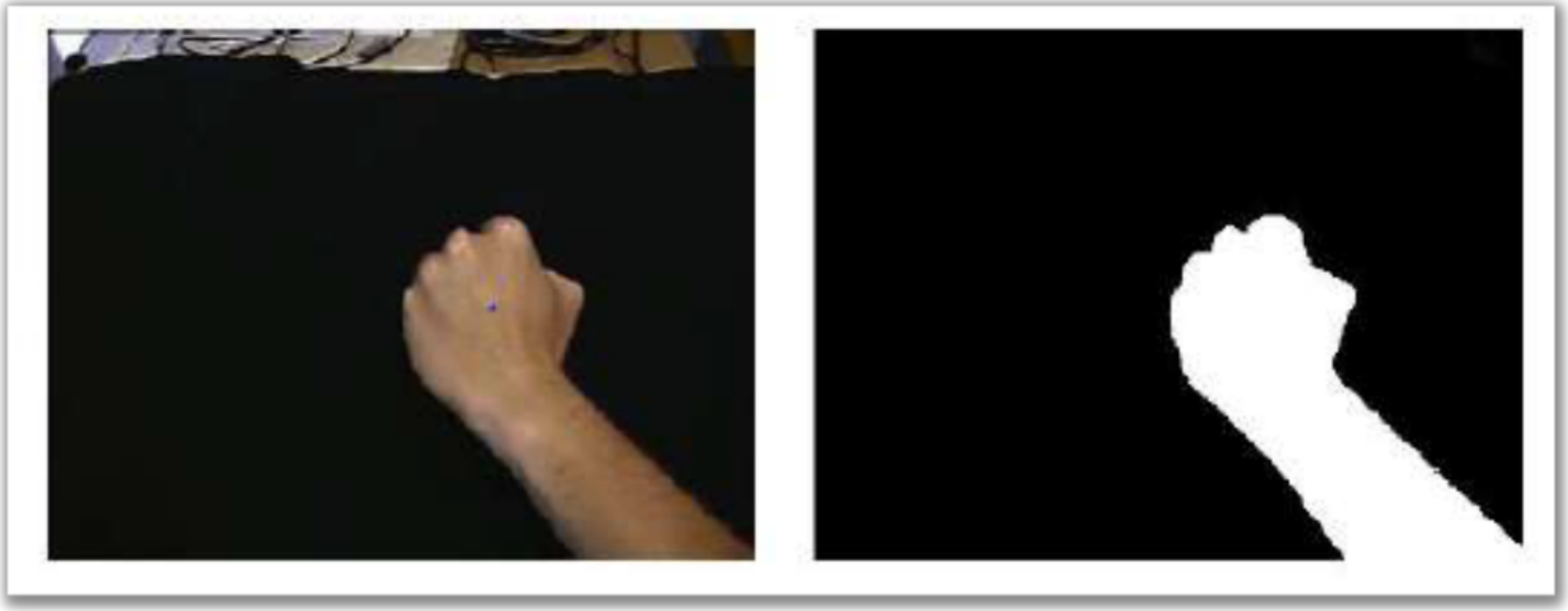
- We can control the environment
- We do not want to create a labelled dataset
- Classic Vision Algorithm are enough for the task
- AI is not always the answer

Possible solutions

for the dectections of cubes and human avoidance with a vision approach

Segmentation

Perform the segmentation of the environment to ease the detection task



Two possible ways to solve the task

1

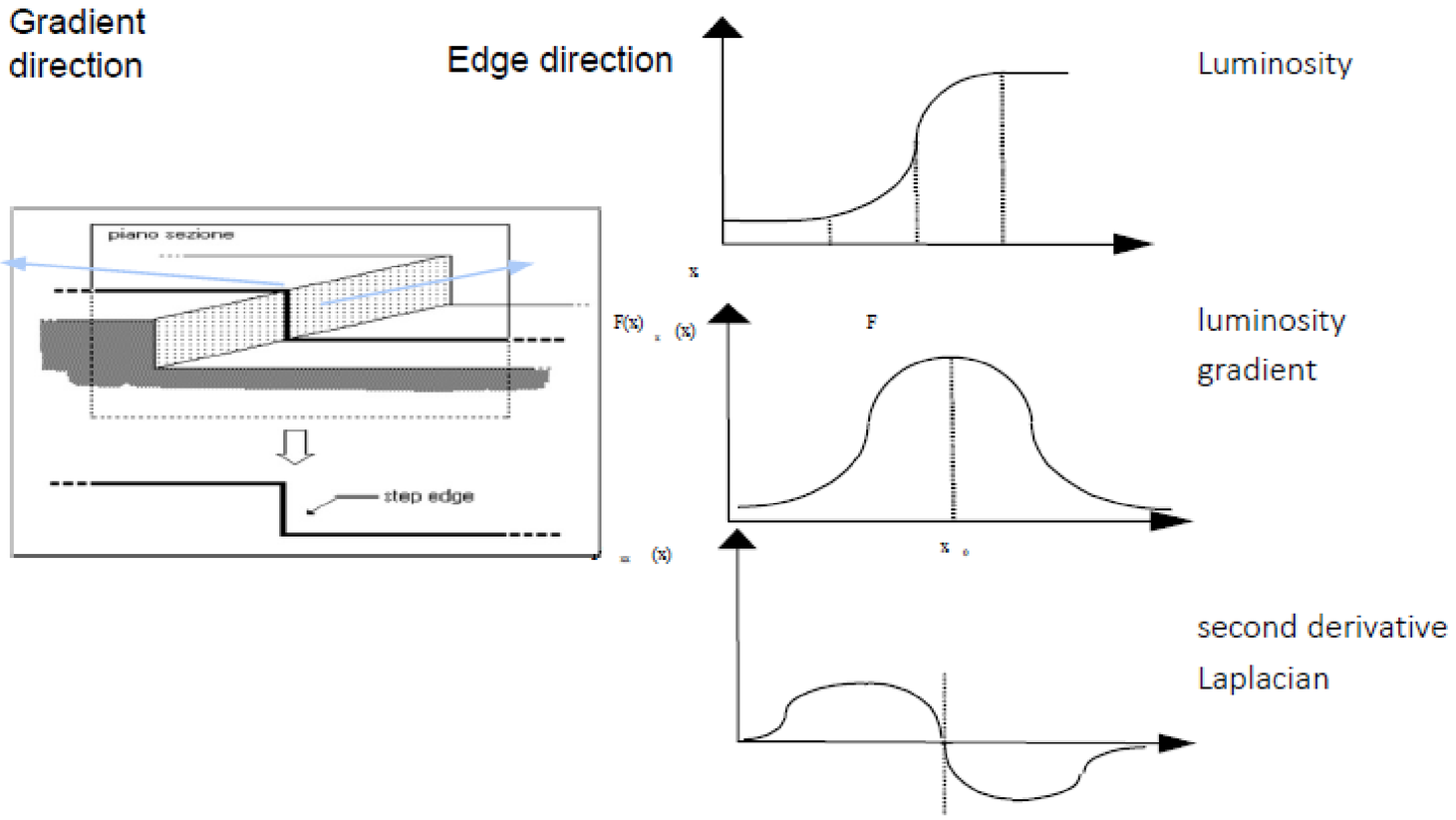
Detect the contour of the cubes

2

Detect the corners of the cubes

The role of the gradient in Computer Vision

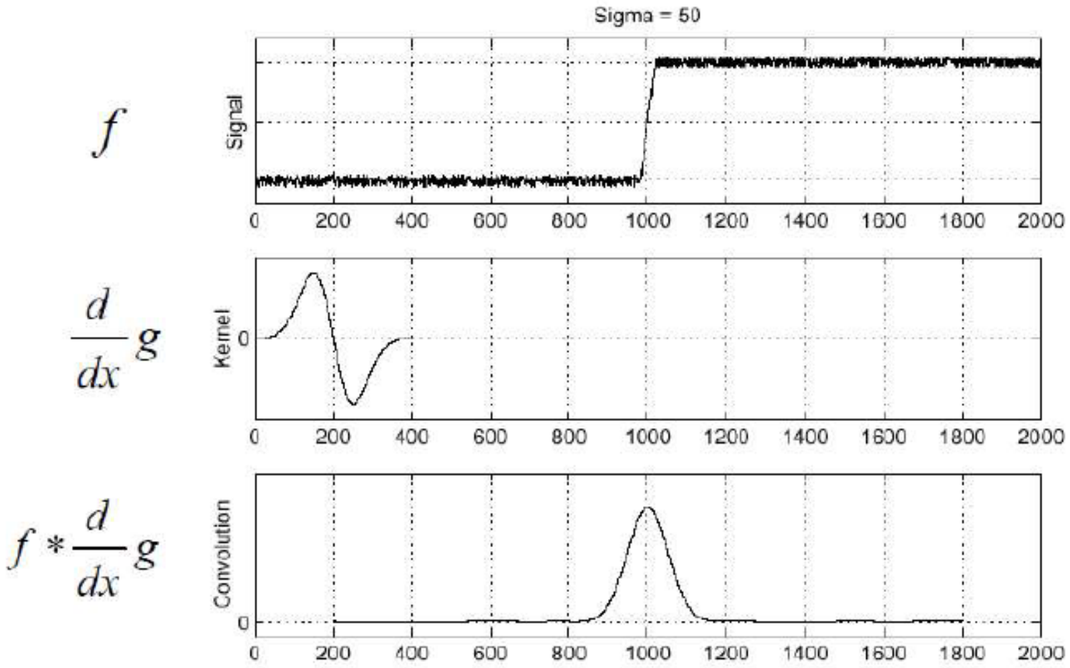
It is a measure of the variation of the intensity of the pixels in an image. It is use to find the edges.

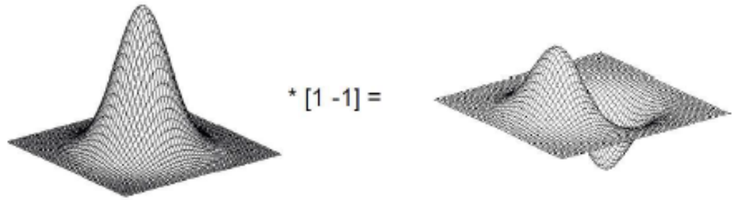


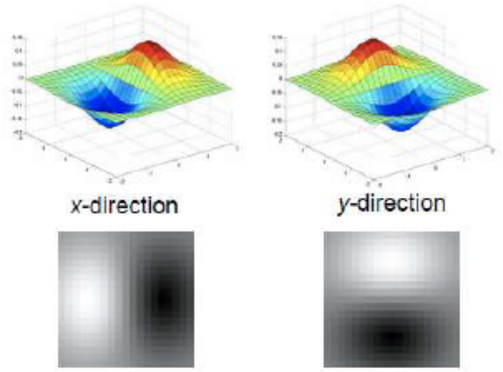
Canny contour detection (1)

Derivative of gaussian (DOG)

Appling the DoG filter is the equivalent of applying the gaussian filter and then comute the gradient of the image.





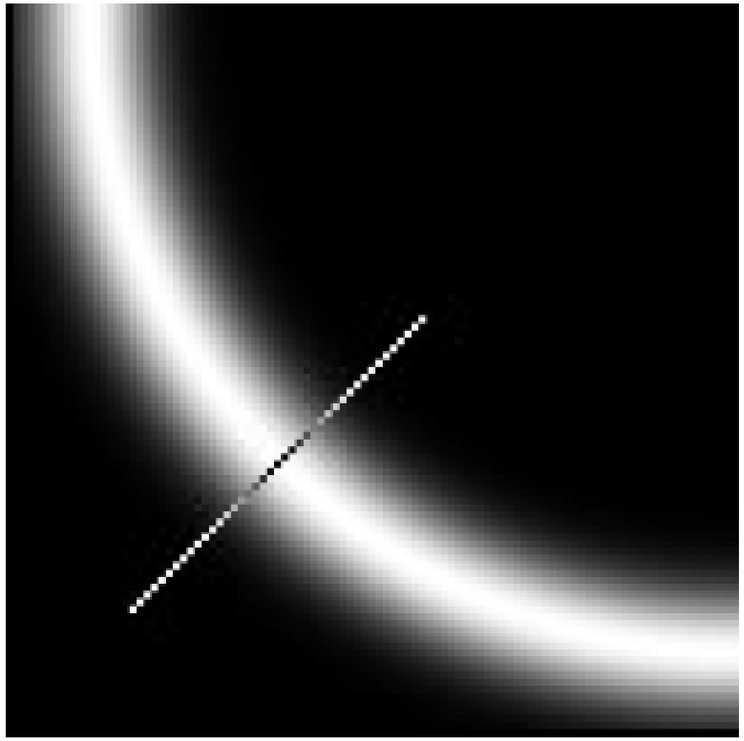


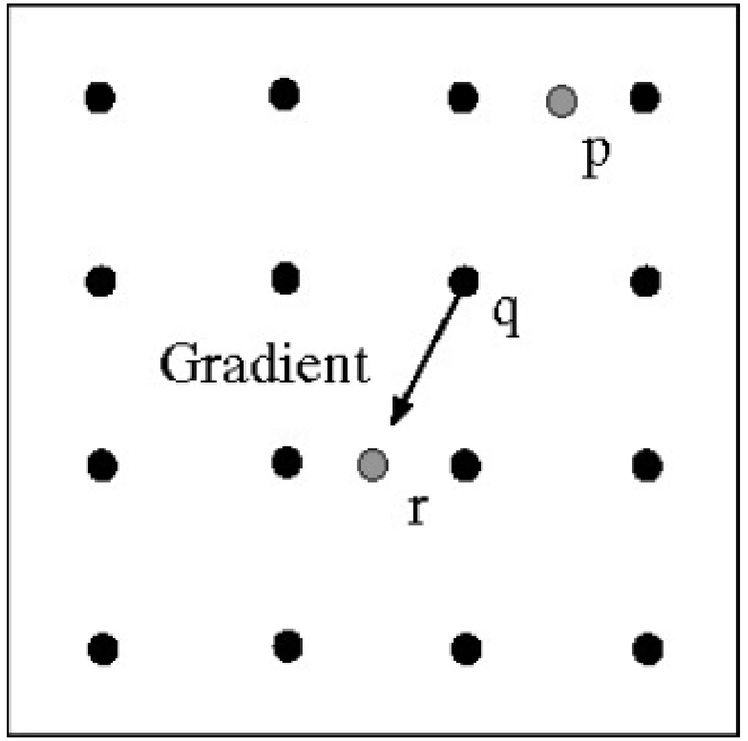
$$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$$

Canny contour detection (2)

Non maxima suppression

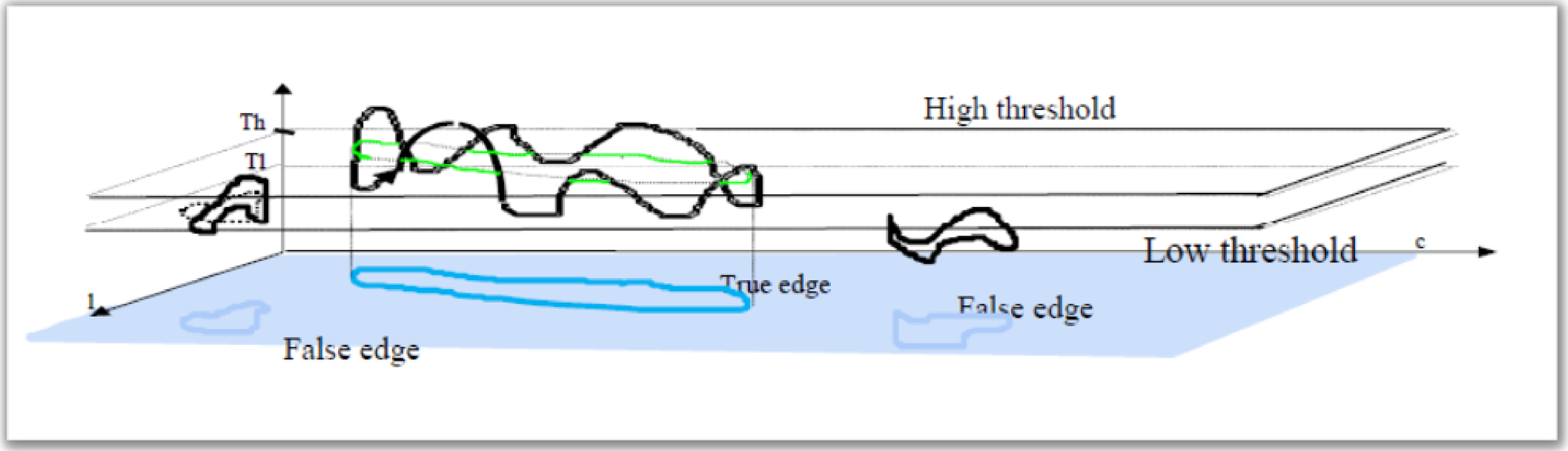
Considering the gradient direction, we compute the second derivative in order to find the local mximum. It corresponds to the right contour pixel





Canny contour detection (3)

Thresholding and hysteresis



Harris corner detection

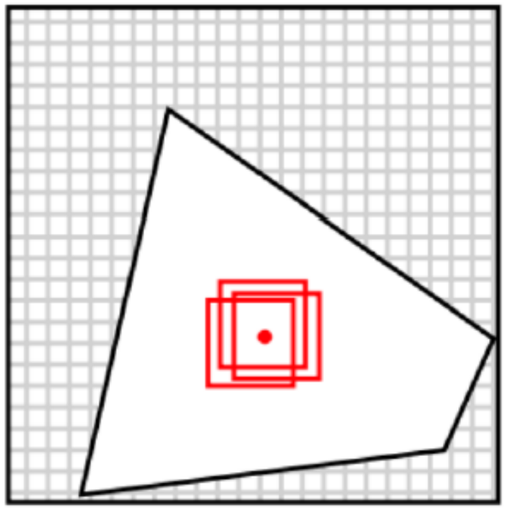
We use autocorrelation in order to understand if there are local changes and in how many directions

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

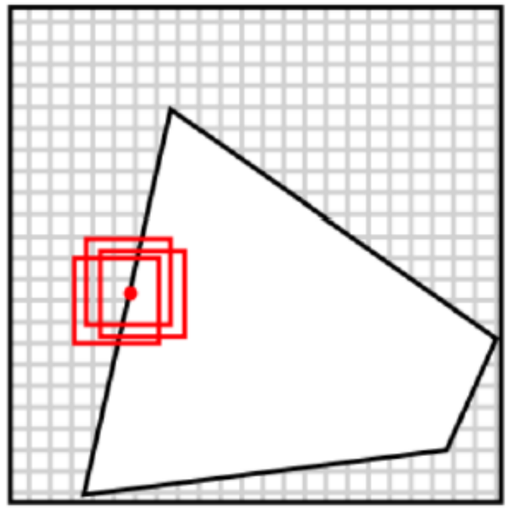
Window function

Shifted intensity

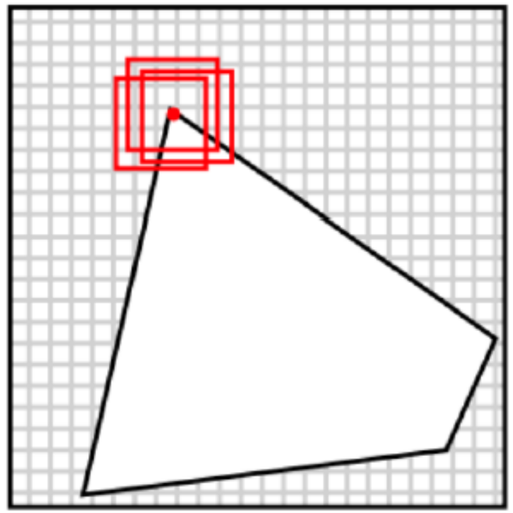
Intensity



Flat region
No local changes

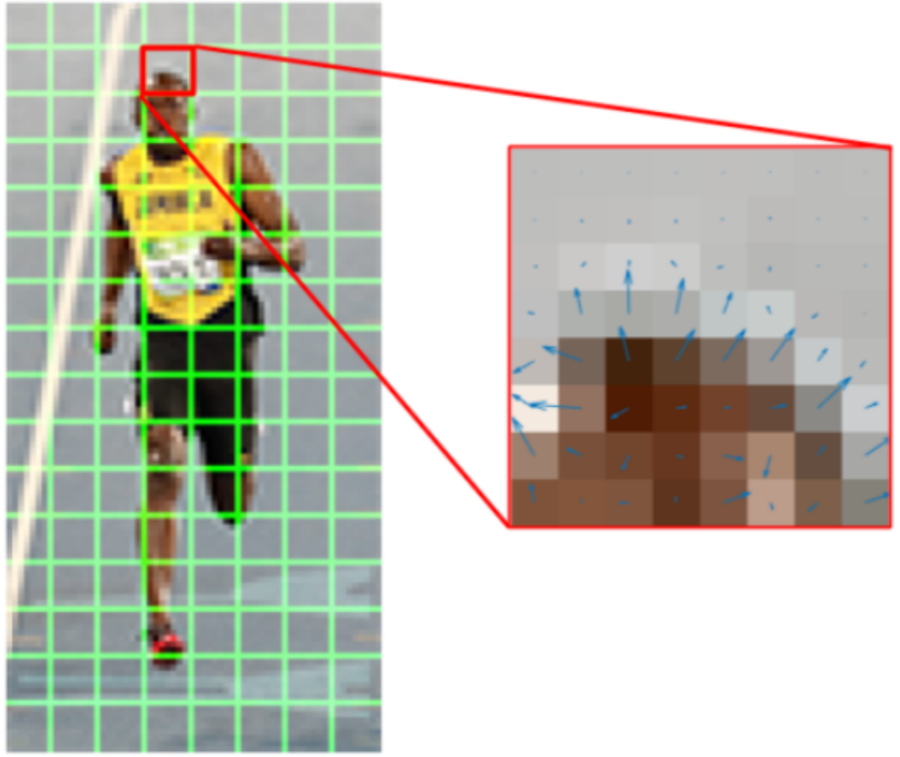


border
No local changes
In the border direction



Keypoint
local changes
In all directions

Histogram of oriented gradients



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

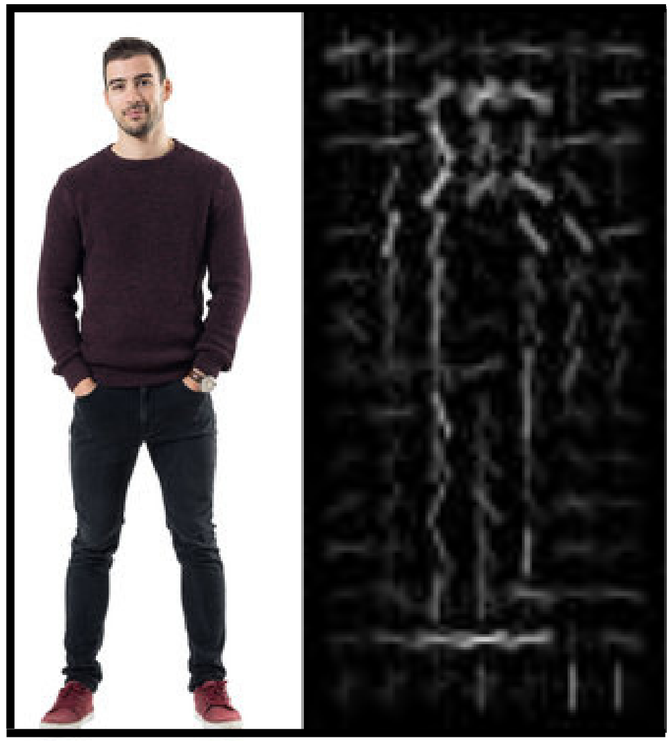
80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

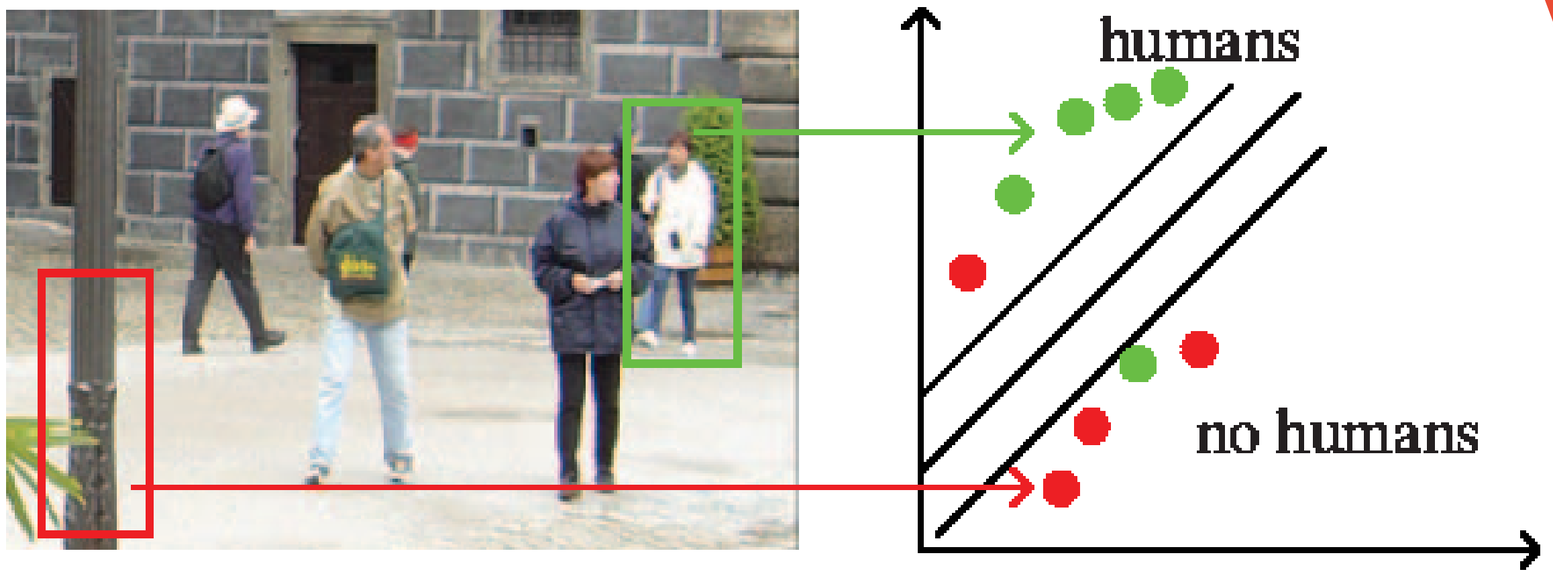
Gradient Direction

Divide the image in cells and describe them with an histogram of oriented gradientes

Histogram of oriented gradients

Then use this descriptor to detect shapes in the images by passing it throught an SVM classifier





Implementation

The acquisition of the image

Vision sensor in *CoppeliaSim*: a viewable object that renders what's in its field of view.

Why not a camera sensor?

Because a vision sensor:

- **has a fixed resolution**
- **is used mainly to display shapes**
- **works with the API**

The interface

The legacy remote API allows to control a simulation from an external application.

Interacts with CoppeliaSim via socket communication:

- **the client side: Pyhton**
- **the server side: via a CoppeliaSim plugin**

For instructions refer to:

Client: https://www.coppeliarobotics.com/helpFiles/en/remoteApiClientSide.htm

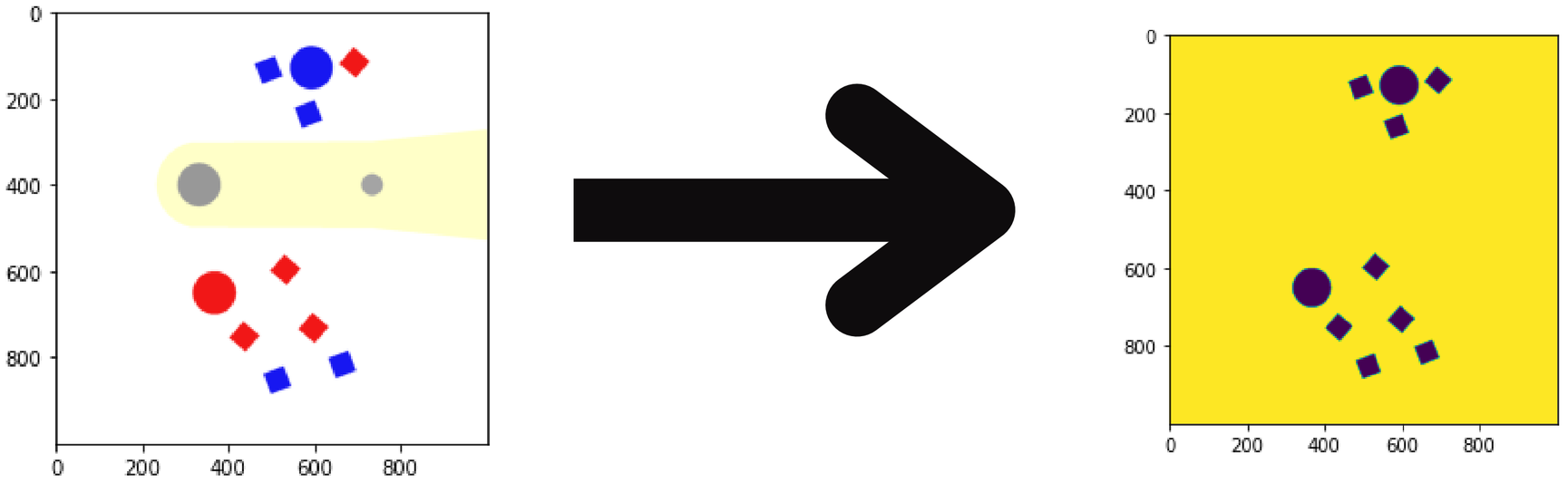
Server: https://www.coppeliarobotics.com/helpFiles/en/remoteApiServerSide.htm

The implemented vision pipeline – 1

```
# Otteniamo l'immagine
retCode, resolution, img=sim.simxGetVisionSensorImage(clientID,sensorHandle,0,sim.simx_opmode_oneshot_wait)
```

```
#convert image into greyscale mode
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#find threshold of the image
_, thrash = cv2.threshold(gray_image, 100, 250, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(thrash, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
```



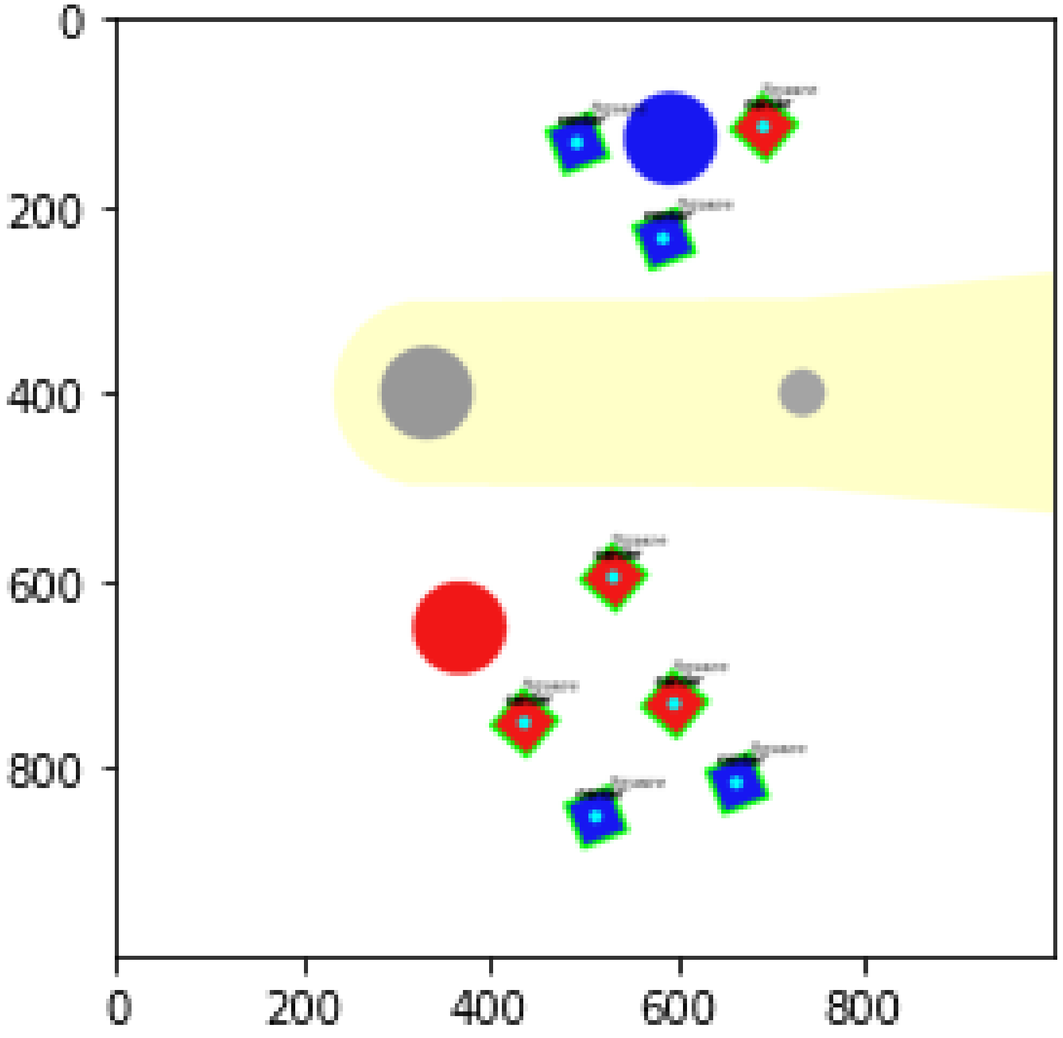
The implemented vision pipeline – 2

```
centers=[]
colori=[]
for contour in contours:
    shape = cv2.approxPolyDP(contour, 0.01*cv2.arcLength(contour, True), True)
    if shape[0][0][0] != 0:
        x_cor = shape.ravel()[0]
        y_cor = shape.ravel()[1]

        if len(shape) ==4:
            #shape cordinates
            x,y,w,h = cv2.boundingRect(shape)

            #width:height
            aspectRatio = float(w)/h
            cv2.drawContours(image, [shape], 0, (0,255,0), 4)
            if aspectRatio >= 0.9 and aspectRatio <=1.1:
                cv2.putText(image, "Square", (x_cor, y_cor), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0,0,0))
                M = cv2.moments(contour)
                if M['m00'] != 0:
                    cx = int(M['m10']/M['m00'])
                    cy = int(M['m01']/M['m00'])
                    retCode, resolution, img2=sim.simxGetVisionSensorImage(clientID,sensorHandle,0,sim.simx_opmode_oneshot_wait)
                    image2 = np.array(img2, dtype=np.uint8)
                    image2.resize([resolution[1],resolution[0],3])
                    (b, g, r) = image2[cy,cx]
                    colori.append([b, g, r])
                    cv2.drawContours(image, [contour], -1, (0, 255, 0), 2)
                    cv2.circle(image, (cx, cy), 7, (0, 255, 255), -1)
                    cv2.putText(image, "center", (cx - 20, cy - 20),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)
                    centers.append([cx,cy])
            else:
                cv2.putText(image, "Rectangle", (x_cor, y_cor), cv2.FONT_HERSHEY_COMPLEX, 0.5, (255,0,0))
```

The implemented vision pipeline – 3



The implemented vision pipeline – 4

```
for c in centers:
    if colori[m][0]>200:
        immXdiff = ((resolution[0]/2) - c[0]) * 0.001
        immYdiff = ((resolution[1]/2) - c[1]) * 0.001
        print("diff calculated from the image")
        print(immXdiff)
        print(immYdiff)
        posizioneTarget=[0,0,0]
      # Sommando la differenza trovata prima alla posizione della
      # camera che é nota si trova la posizione target
        posizioneTarget[0]=posizioneCamera[0] + immXdiff
        posizioneTarget[1]=posizioneCamera[1] + immYdiff
        posizioneTarget[2]=0.05
        eulC=[0,0,0]
        targetP = posizioneTarget + eulC
        print(targetP)
        sq = sp.symbols(['q1','q2','q3','q4'])
        T2 = sp.Matrix([[sp.cos(sq[0] + sq[1] + sq[3]), -sp.sin(sq[0] + sq[1] + sq[3]), 0, 0.467*sp.cos(sq[0]) + 0.4005*sp.cos(sq[0] + sq[1])],
                        [sp.sin(sq[0] + sq[1] + sq[3]), sp.cos(sq[0] + sq[1] + sq[3]), 0, 0.467*sp.sin(sq[0]) + 0.4005*sp.sin(sq[0]+ sq[1])],
                        [0, 0, 1, 0.234 - sq[2]], [0, 0, 0, 1]])
        # inviamo alla posa di destinazione
        d2 = targetP
        D2 = matrixFromPose(d2)
        q = sp.nsolve(T2-D2, sq, [0.1, 0.1, 0.1, 0.1], prec=6)
```

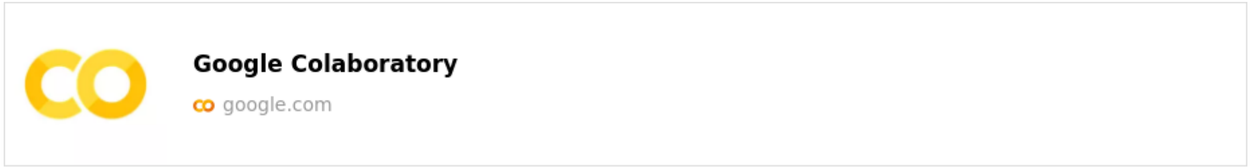
The implemented vision pipeline – 5

```
# spostiamo il robot nella posizione
retCode = sim.simxSetJointTargetPosition(clientID, joint1, q[0], sim.simx_opmode_blocking)
retCode = sim.simxSetJointTargetPosition(clientID, joint2, q[1], sim.simx_opmode_blocking)
retCode = sim.simxSetJointTargetPosition(clientID, joint3, 0, sim.simx_opmode_blocking)
retCode = sim.simxSetJointTargetPosition(clientID, joint4, q[3], sim.simx_opmode_blocking)
time.sleep(1)
# abbassiamo l'attuatore
retCode = sim.simxSetJointTargetPosition(clientID, joint3, q[2], sim.simx_opmode_blocking)
time.sleep(1)
# attiviamo l'end effector
setEffector(1)
time.sleep(1)
# alziamo l'attuatore
retCode = sim.simxSetJointTargetPosition(clientID, joint3, 0, sim.simx_opmode_blocking)
time.sleep(1)
# andiamo alla posizione per i cubi rossi
retCode = sim.simxSetJointTargetPosition(clientID, joint1, 0, sim.simx_opmode_blocking)
retCode = sim.simxSetJointTargetPosition(clientID, joint2, 0.5, sim.simx_opmode_blocking)
retCode = sim.simxSetJointTargetPosition(clientID, joint3, 0, sim.simx_opmode_blocking)
time.sleep(1)
# abbassiamo il cubo
retCode = sim.simxSetJointTargetPosition(clientID, joint3, q[2] - (n1*0.05), sim.simx_opmode_blocking)
time.sleep(1)
# Disattiviamo l'attuatore
time.sleep(1)
setEffector(0)
# e lo alziamo
retCode = sim.simxSetJointTargetPosition(clientID, joint3, 0, sim.simx_opmode_blocking)
n1 = n1+1
print("Ho posato un cubo rosso nella sua posizione")
```

Human avoidance code

```python
def Anomaly_detection():
    result = False
    #Otteniamo handle del sensore di visione 2
    retCodeA,sensorHandleA=sim.simxGetObjectHandle(clientID,'Vision_sensor2',sim.simx_opmode_blocking)
    # Otteniamo l'immagine
    retCodeA, resolutionA, imgA=sim.simxGetVisionSensorImage(clientID,sensorHandleA,0,sim.simx_opmode_oneshot_wait)
    # Initializing the HOG person detector
    hog = cv2.HOGDescriptor()
    hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
    # Detecting all the regions in the image that has a pedestrians inside it
    (regions, _) = hog.detectMultiScale(image, winStride=(4, 4), padding=(4, 4), scale=1.05)
    # Drawing the regions in the Image
    for (x, y, w, h) in regions:
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
        result = True
    # Showing the output Image
    plt.imshow(image)
    plt.show()
    return result
```

LINKS TO THE CODE

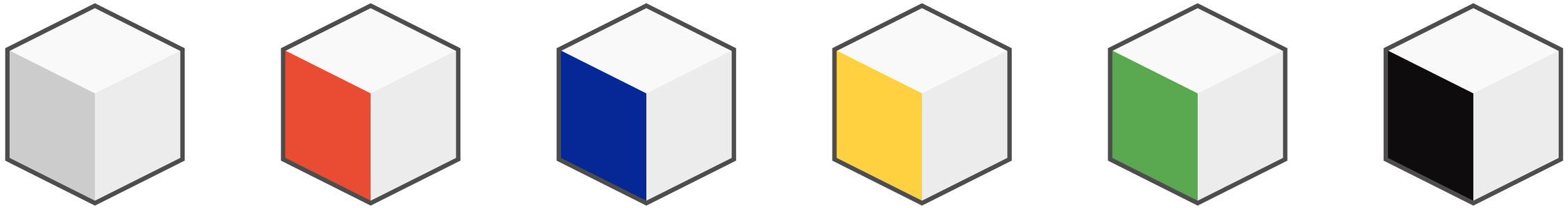


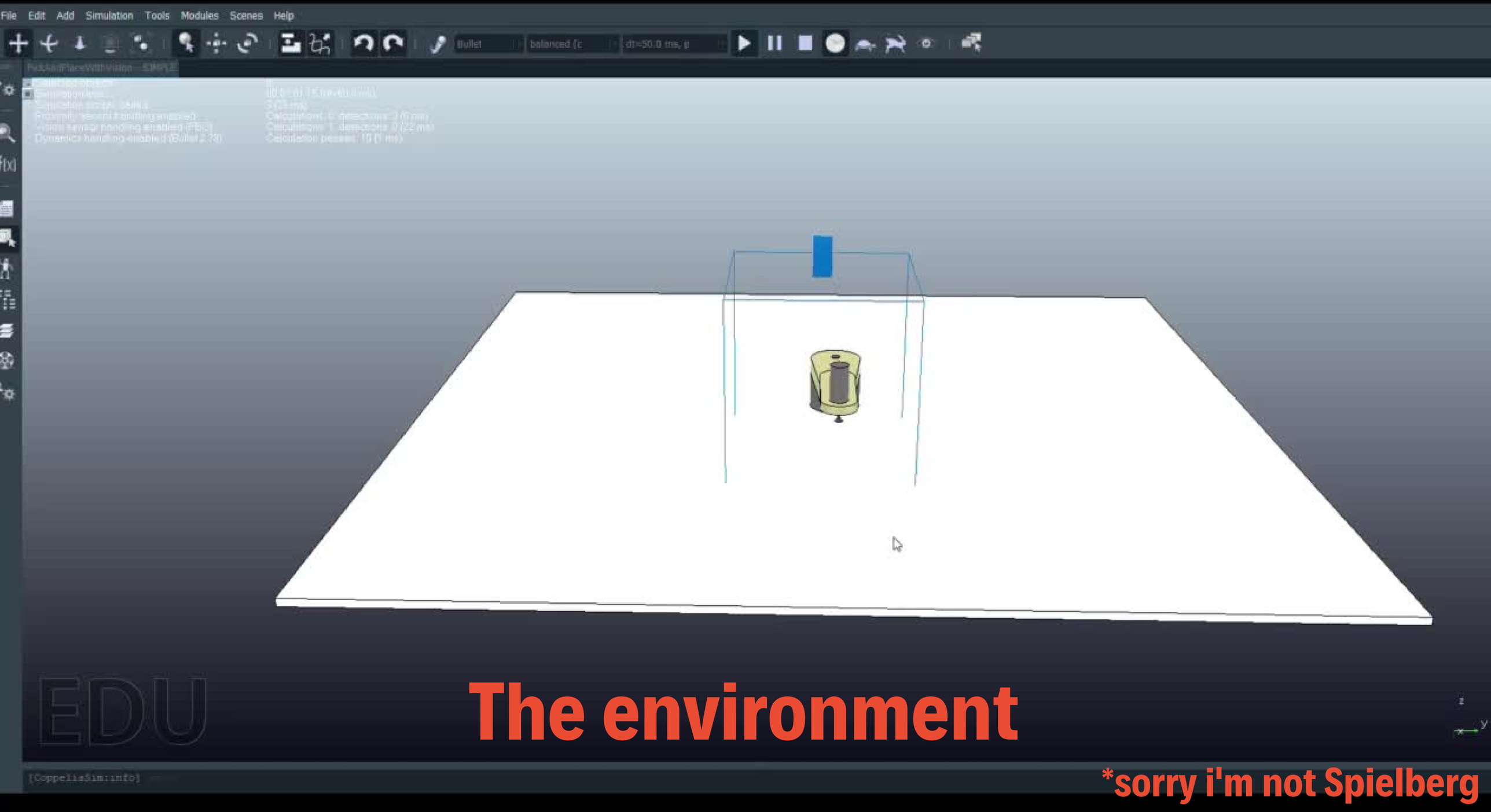
https://github.com/Eronion/VisioCoSM

Tests

Main objectives of the tests

- Pick a cube using a vision sensor
- Distinguish shapes and color
- Separete the object of interest w.r.t. the color
- Have the less a priori notions about the settings
- Stop the work when a human approach the robot and resume it when safe




File Edit Add Simulation Tools Modules Scenes Help
EDU
The environment
*sorry i'm not Spielberg

The experiments assumption

Neither the position or the orientation of the cubes were known a priori

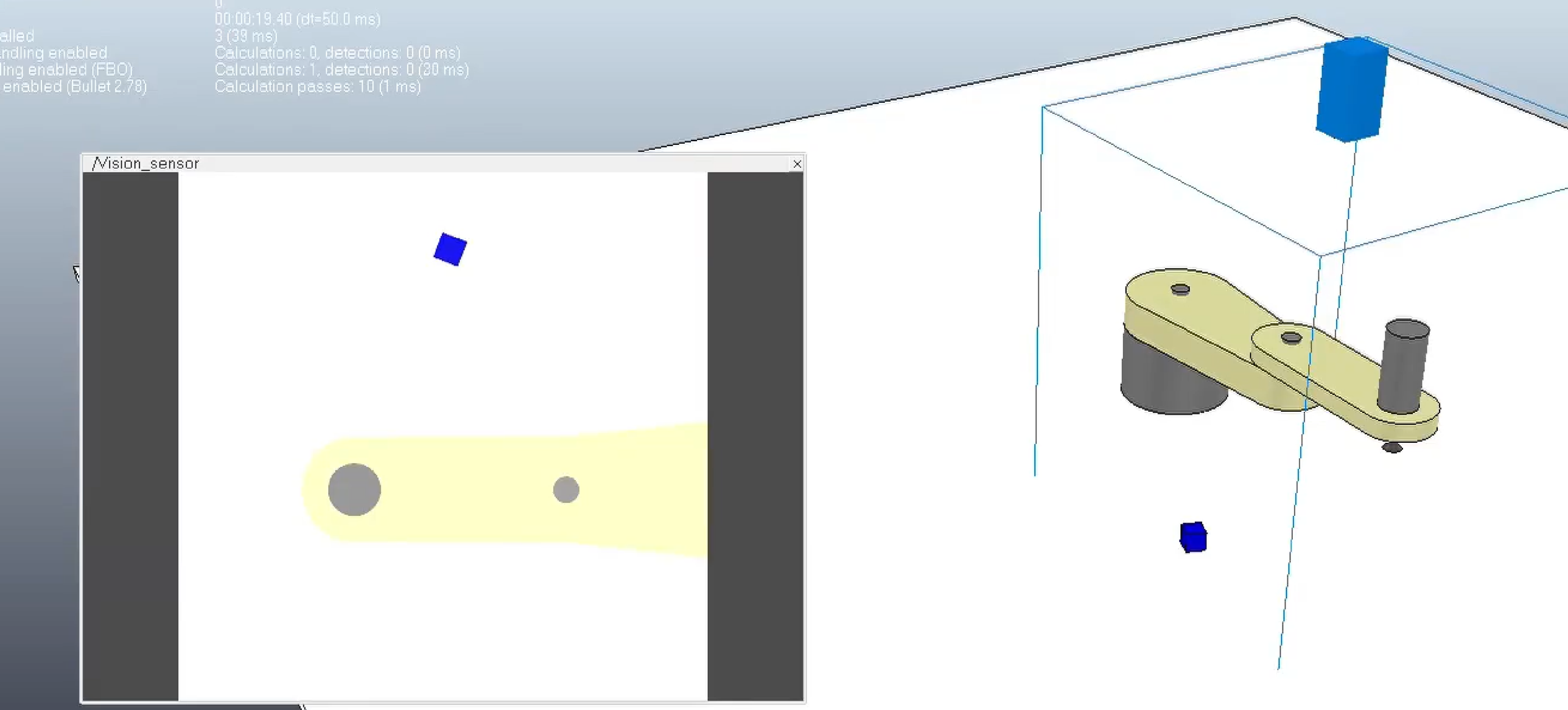
Only the color, the shape and the height of the cubes were known

All the vision tasks are done online via the image obtained from the vision sensor

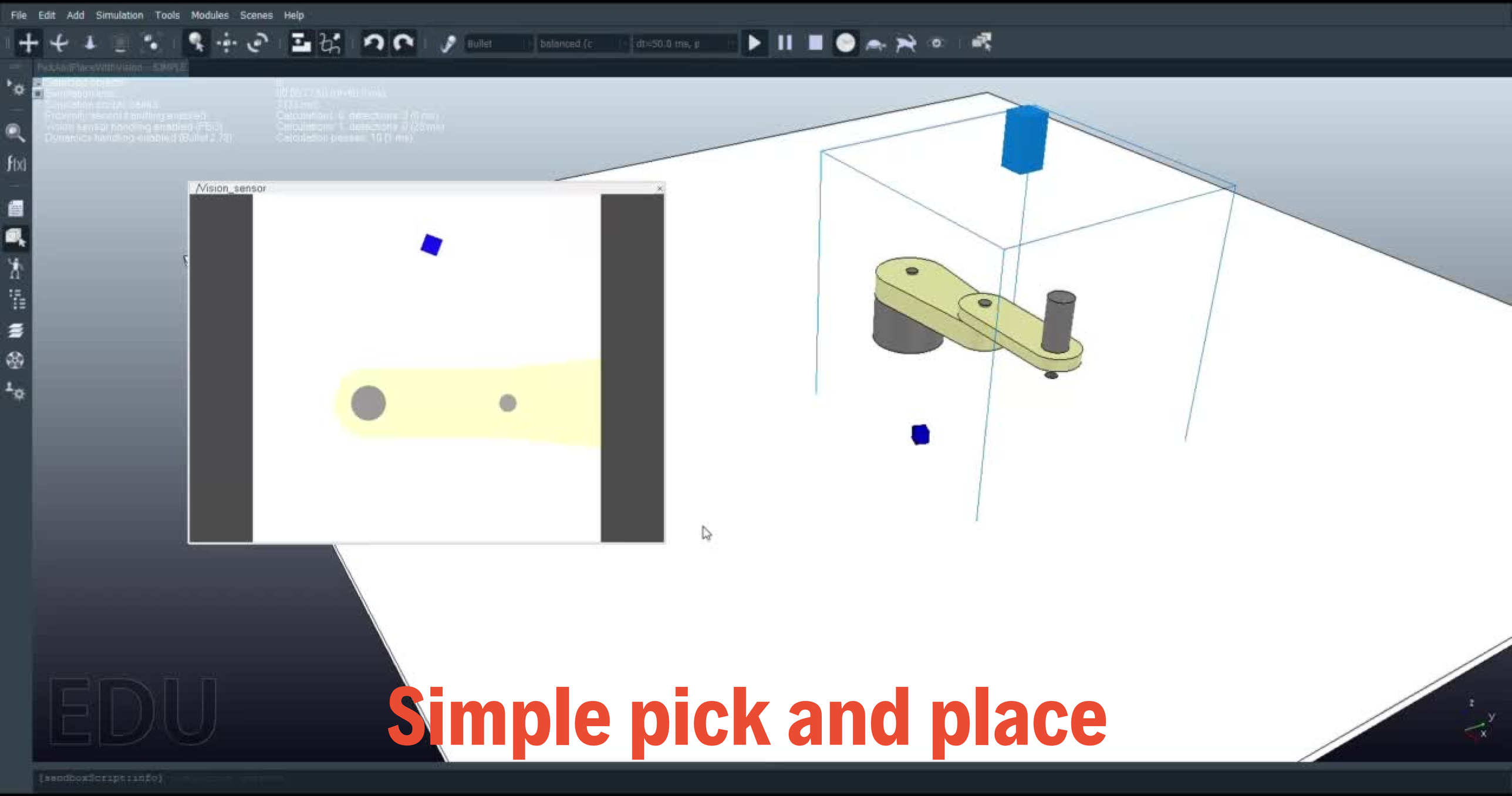
The vision sensor position and its parameters are considered known and fixed

Results

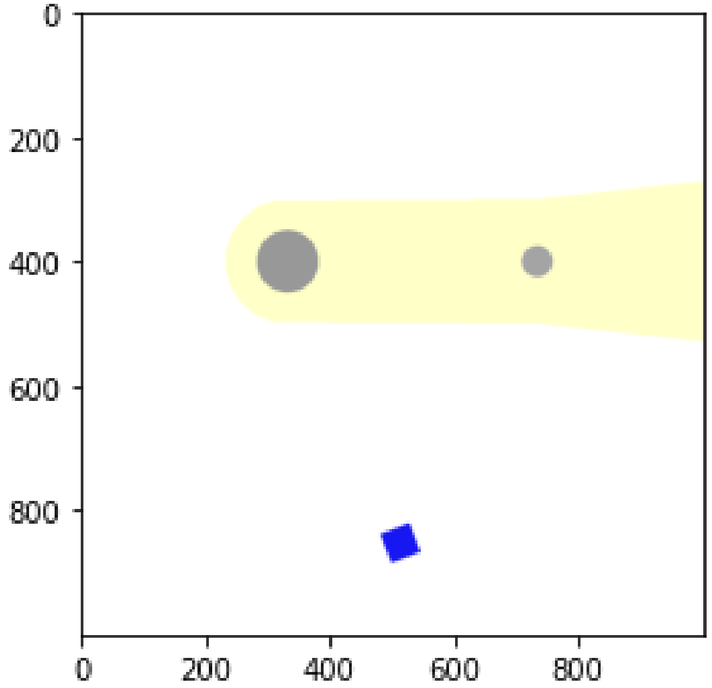
Videos of the simulations



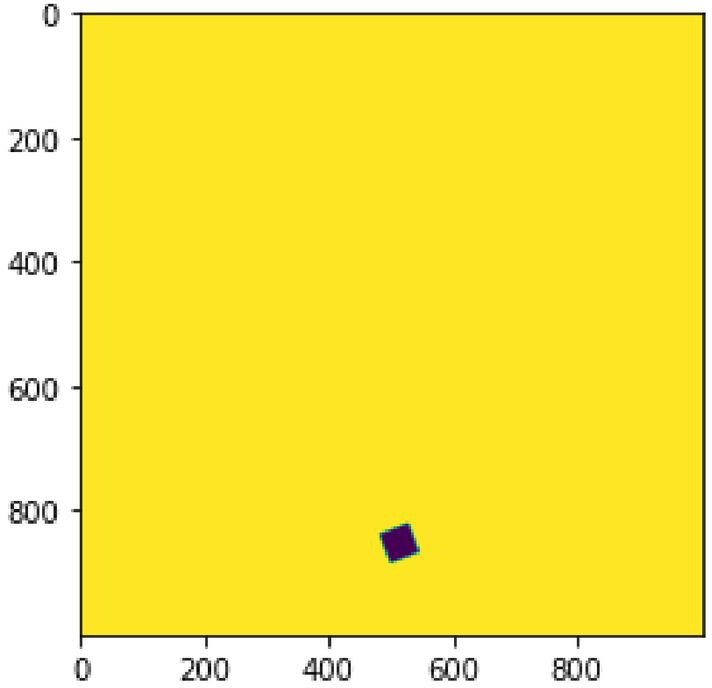
Simple pick and place


File Edit Add Simulation Tools Modules Scenes Help
Vision_sensor
EDU
Simple pick and place

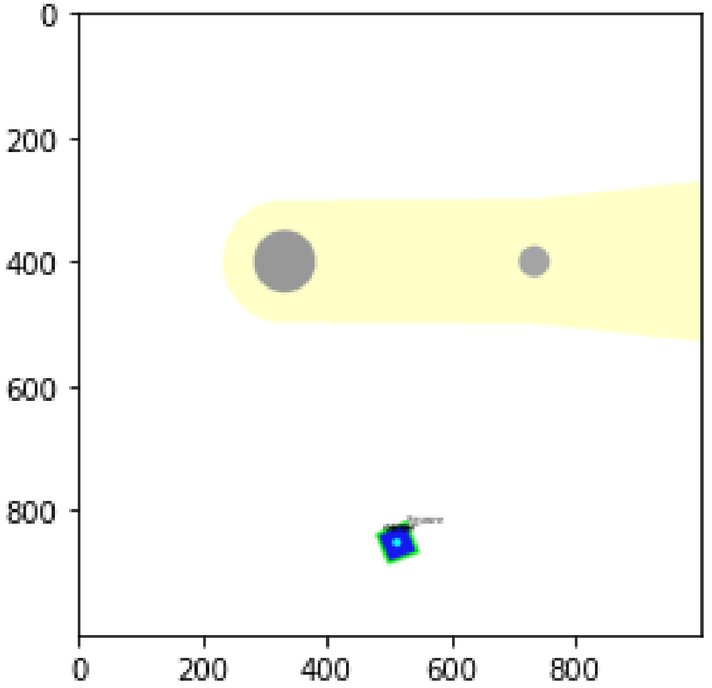
Simple pick and place vision pipeline



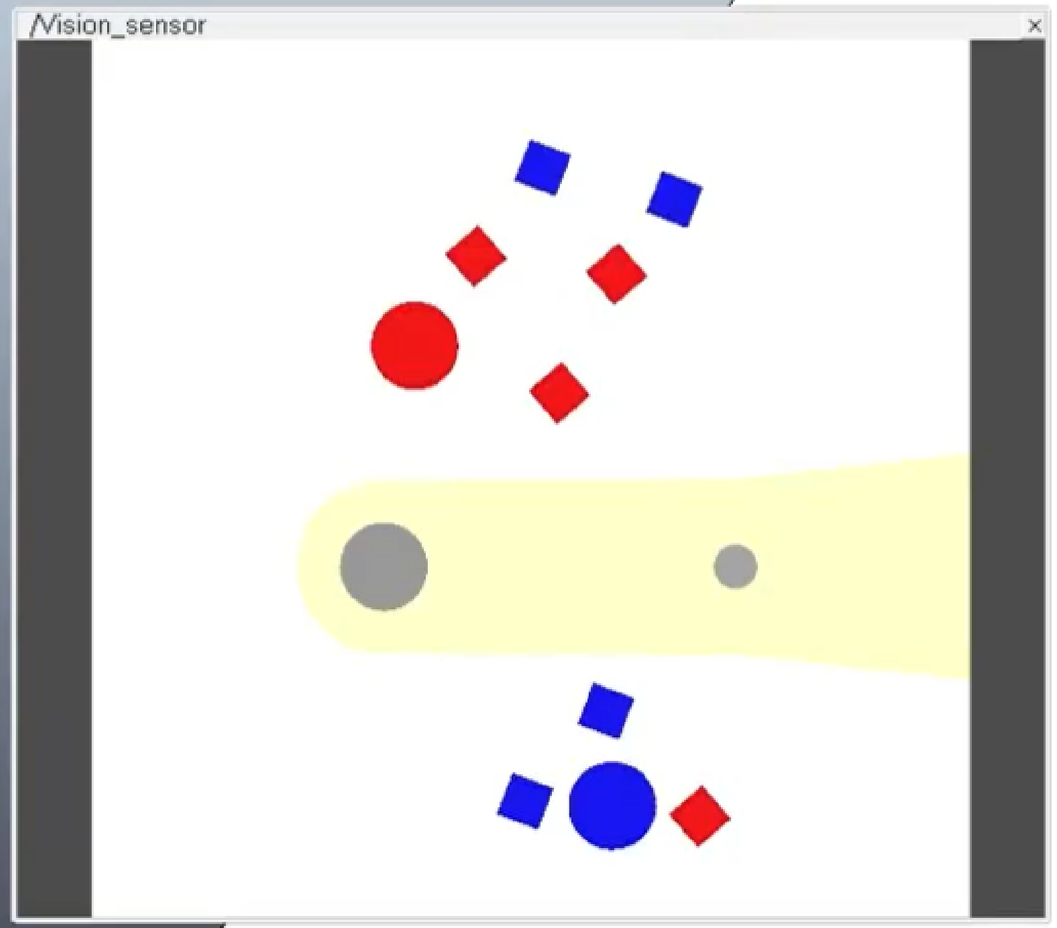
Observation

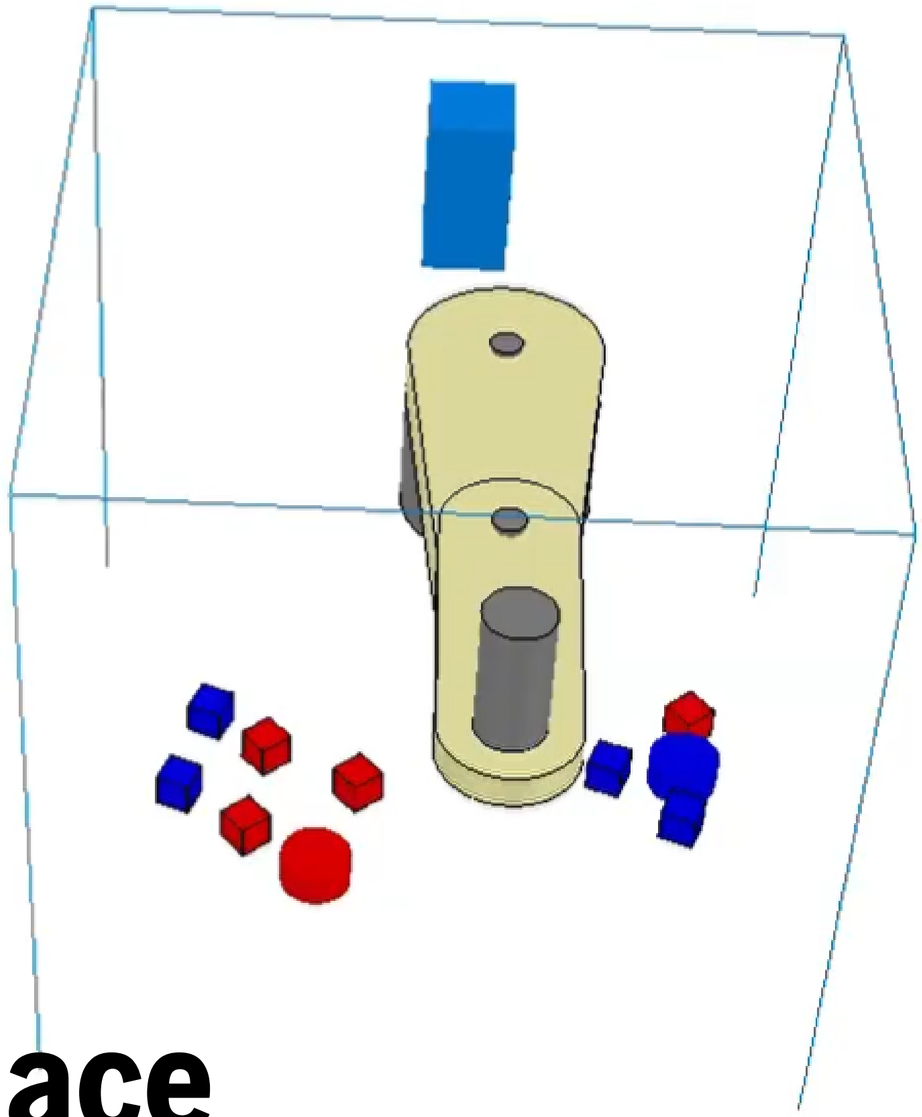


Segmentation

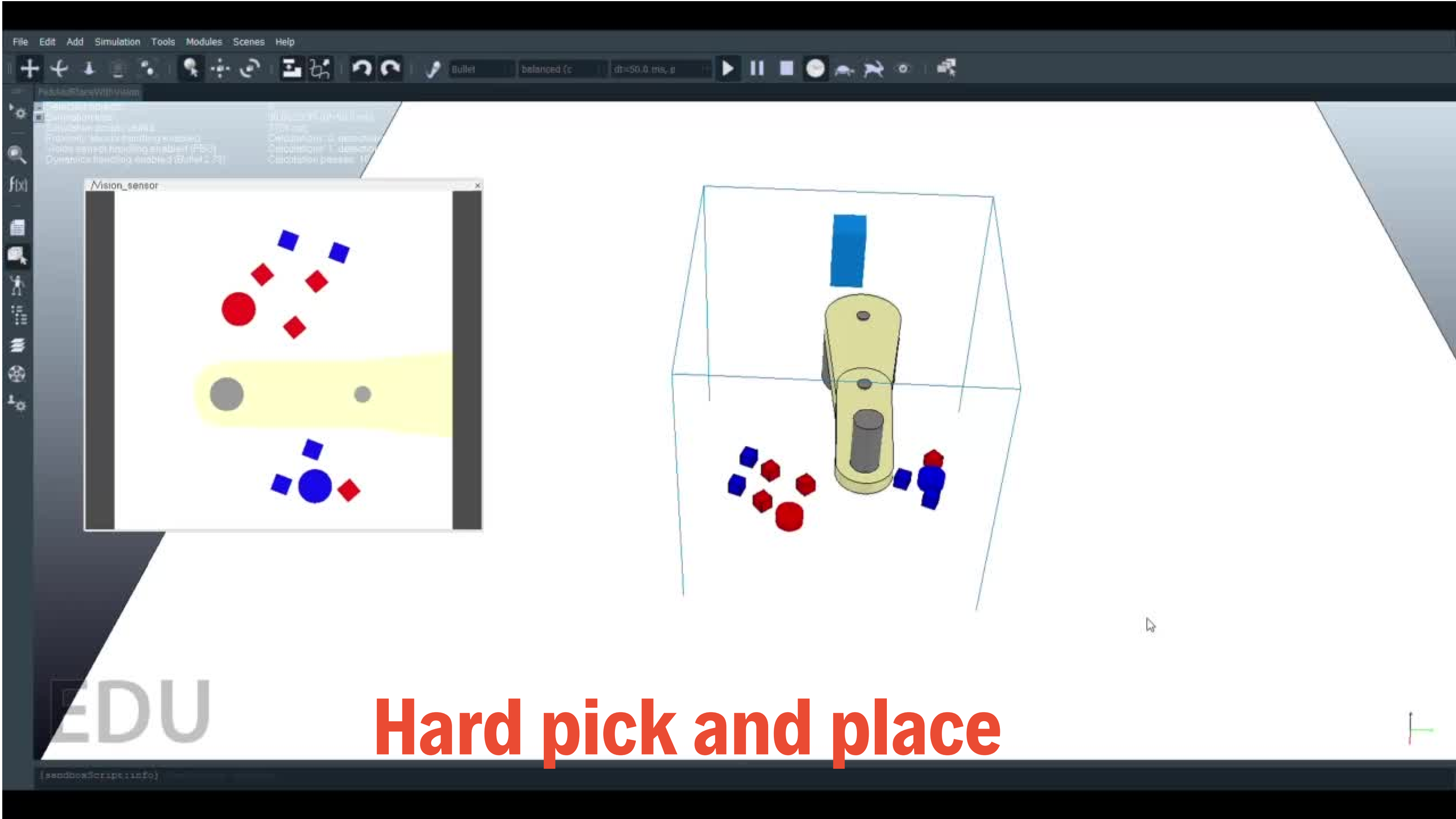


Discovery

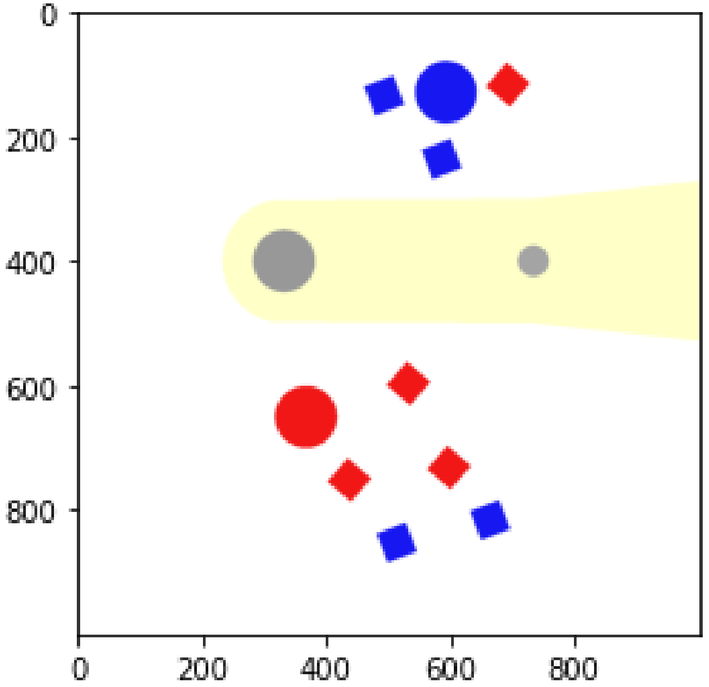




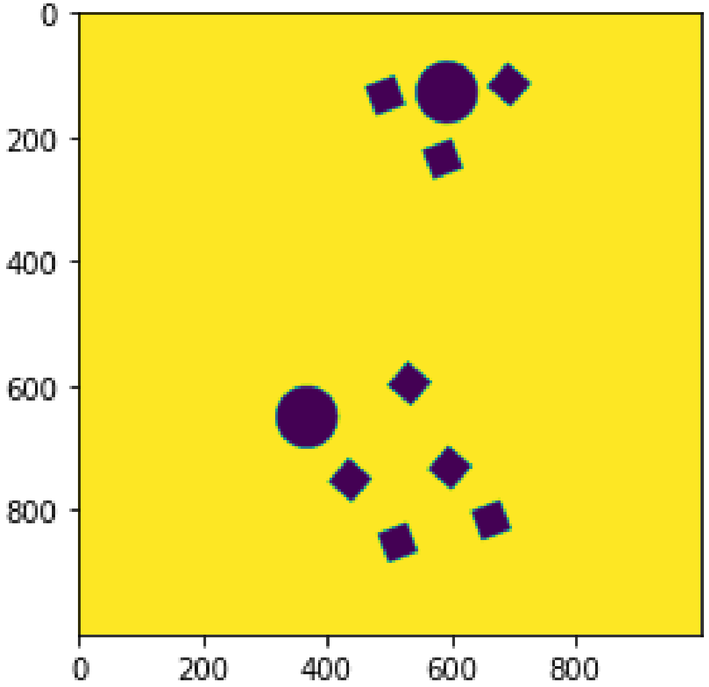
Hard pick and place


File Edit Add Simulation Tools Modules Scenes Help
Vision_sensor
EDU
Hard pick and place

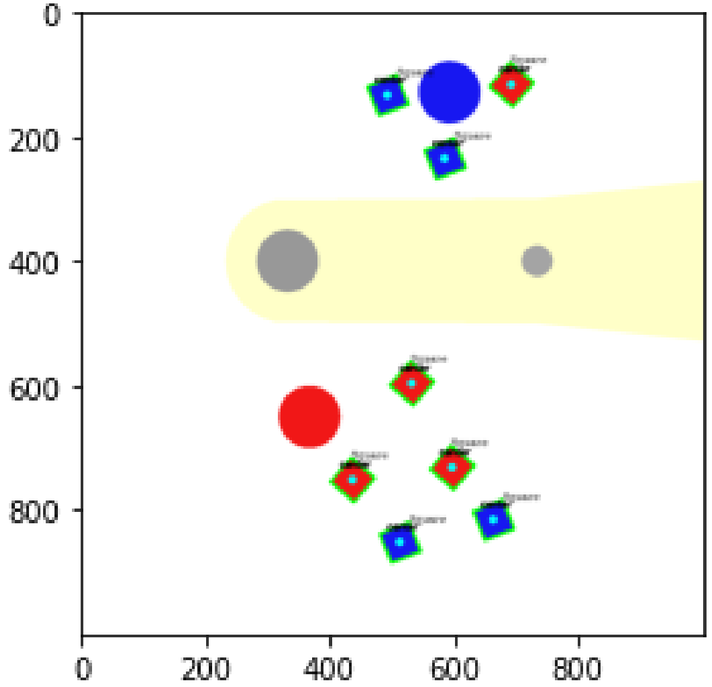
Hard pick and place vision pipeline



Observation

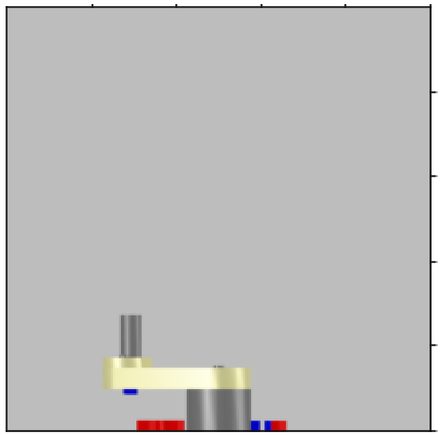


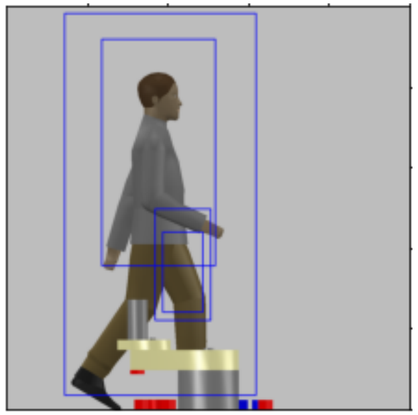
Segmentation

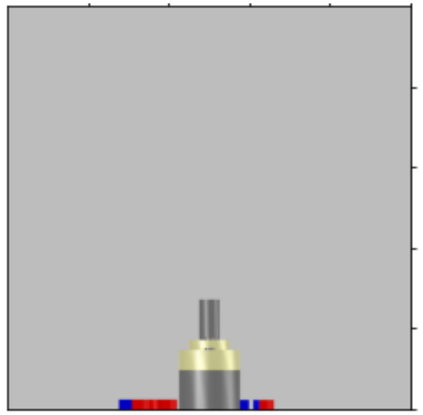


Discovery

Human avoidance system







H.O.G.

Conclusions

Major results

We learned the enormous potenciality of the Vision-Robotic duo

OUR MANIPULATOR WAS ABLE TO RESOLVE THE TASK IN A SMART WAY

The robot became aware of the environment

MAKE A "THING" BEEING ABLE TO DISTRINGUISH BETWEEN SHAPES AND COLORS WAS REALLY SATISFYING

Hard work pays

HOURS OF TESTING, DEVELOPMENT AND STUDY HAVE FINALLY PRODUCED A CONCRETE RESULT

Future improvements

Test other vision sensor locations

FOR SIMPLICITY WE OPTED FOR A FIXED LOCATION BUT TO AVOID OCCLUSION THE BEST THING IS TO PLACE THE CAMERA IN THE ROBOT END-EFFECTOR

Make the prototype usable in a real context

TRANSLATING THE PROJECT FROM PYTHON TO C/C++ ENABLES US THE USE OF HARDWARE ACCELERATION AND PARALLELIZATION TECHNIQUES THAT WOULD SIGNIFICANTLY SPEED UP THE ROBOT INFERENCES ABOUT THE ENVIRONMENT

Develop a full deep approach

THE ENORMOUS POTENCIALITY OF A PURE DEEP APPROCACH COULD BE A KEY STEP TO RELAX SOME OF THE ASSUMPTION MADE ON THE ENVIRONMENT AND COULD OPEN THE USE OF THE MANIPULATOR IN "OUT OF THE FACTORY" CONTEXTS

THANK YOU

A project developed for the course Smart Robotics - Unimore 2022

BELLEI | DE VITIS | SEVERI | VIGLIANISI