

دانشگاه صنعتی امیرکبیر **(پلی تکنیک تهران)**

کنترل هوشمند در رباتیک

تکلیف اول کارگاه

استاد: دکتر ایمان شریفی

اعضای گروه:

درسا رحمتی ۹۹۲۳۱۱۰

تنیا تبشیری نمین ۹۹۲۳۱۰۰

امیرحسین دژدار ۹۹۲۳۰۲۶

محدثه رضایی حدادی ۹۹۲۳۰۳۴

سینا فاضل ۹۹۲۳۰۵۸

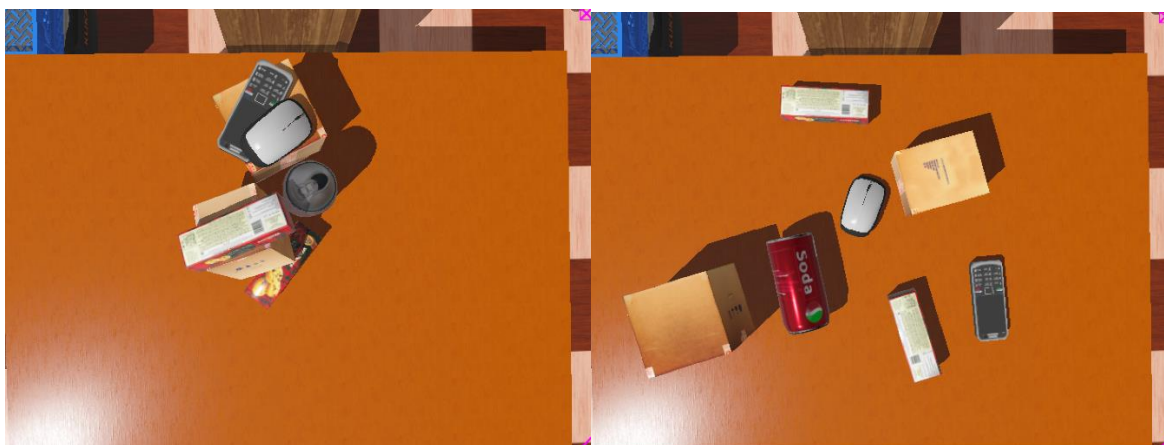
فهرست

3.....	شبکه عصبی
3.....	DATASET
6.....	TRAIN کردن شبکه عصبی (ایجاد مدل و آموزش آن)
12.....	اجرا و ارزیابی مدل
23.....	INVERSE KINEMATIC
28.....	منابع

شبکه عصبی

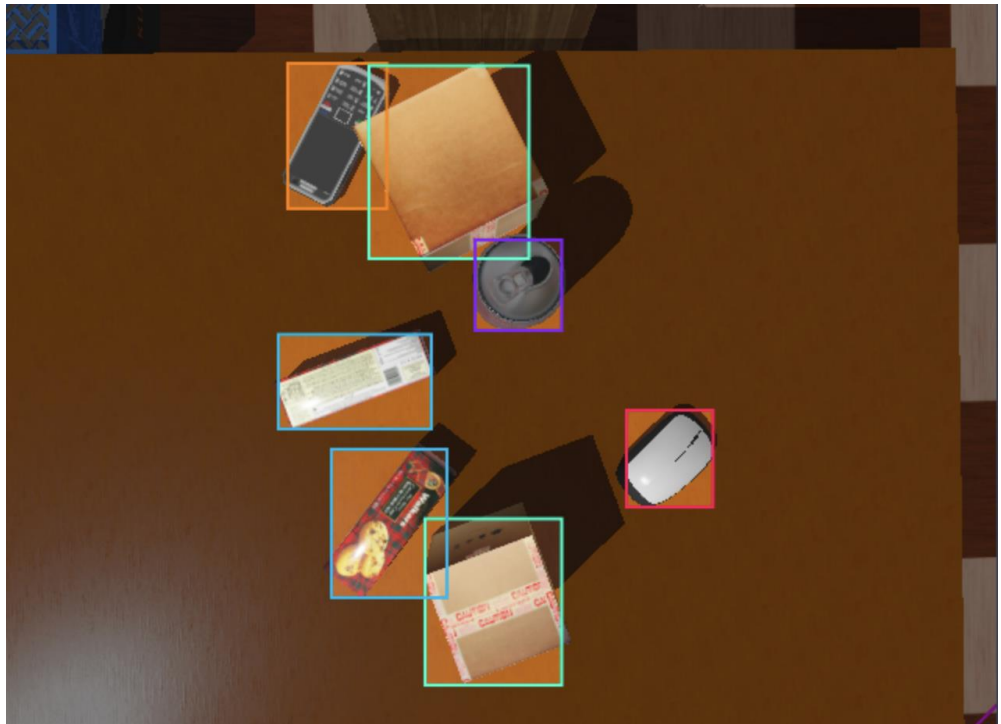
Dataset

برای train کردن شبکه CNN، ابتدا از اشیا مورد نظر در شرایط و جایگشت های متفاوت از دید دوربین ربات عکس گرفتیم. حدود ۱۷۷ عکس جمع آوری شد.



با کمک سایت Roboflow، برای این دیتاها لیبل زدیم. این لیبل ها شامل 'mobile'، 'biscuit_box'، 'soda'، 'mouse'، 'box' میشوند. همچنین تغییراتی اعم از Flip، Blur، Noise و Cutout دادیم و دیتاست مورد نیاز را ایجاد کردیم. در نهایت ۵۳۱ داده در اختیار ما قرار گرفت.

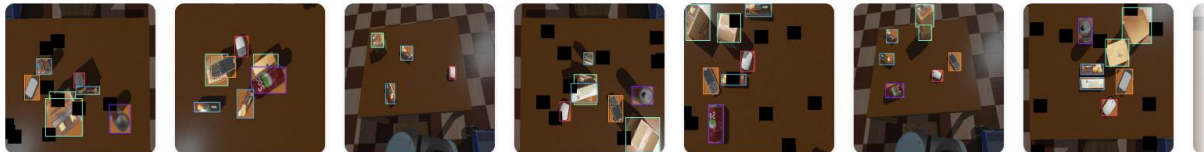
api_key="1dfLRDwtJldjEs4MgRXS"



توزیع این داده ها در قسمت های train، validation و test به شرح زیر است:

531 Total Images

[View All Images →](#)



Dataset Split

TRAIN SET

88%

465 Images

VALID SET

8%

44 Images

TEST SET

4%

22 Images

Preprocessing

Auto-Orient: Applied

Resize: Stretch to 640x640

Augmentations

Outputs per training example: 3

Flip: Horizontal, Vertical

Blur: Up to 1.8px

Noise: Up to 1.6% of pixels

Cutout: 10 boxes with 9% size each

Train کردن شبکه عصبی (ایجاد مدل و آموزش آن)

در ادامه به کد شبکه می پردازیم (این کد در Google Colab نوشته شده است):

```
!pip install roboflow
!pip install ultralytics
!pip install torch torchvision timm
```

در این بخش، کتابخانه های مورد نیاز یعنی roboflow (برای استفاده از دیتاست ایجاد شده) ، ultralytics (دارای ابزار هایی برای مدل YOLO) و torch torchvision timm (برای استفاده از ابزار ها و مدل های pytorch) را نصب میکنیم.

```
# get images for train from roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="1df1RDwtJJIdjEs4MgRXS")
project = rf.workspace("robsharifi-mhjuq").project("5label")
version = project.version(5)
dataset = version.download("yolov8-obb")
```

در این بخش، میخواهیم عکس ها و دیتا ها را برای train کردن شبکه دریافت کنیم. ابتدا از کتابخانه ، کلاس Roboflow را برای تعامل با سایت roboflow ، فراهم میکنیم.

در ادامه، برای دسترسی به داده ها و احراز هویت، از API key خود استفاده میکنیم.

در خط بعد به یک فضای کاری خاص (robsharifi-mhjuq) و پروژه (label5) در آن فضای کاری دسترسی پیدا میکنیم.

پروژه ما در roboflow چندین نسخه متفاوت داشت که ما به نسخه ۵ احتیاج داشتیم و در ادامه آن را انتخاب میکنیم.

در خط آخر، داده ها را دانلود میکنیم. این داده ها در فرمتی دانلود میشوند که با (oriented obb bounding boxes) در YOLOv8 سازگار باشند.

```
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8s.pt') # or 'yolov8n.pt', 'yolov8m.pt', etc.

# Train the model
model.train(data='/content/5label-5/data.yaml', epochs=100, imgsz=640, batch=16)
```

در ابتدا، کلاس YOLO را از کتابخانه ultralytics میگیریم.

در ادامه، یک مدل pre-train شده را load میکنیم. این مدل ها در سایز و کارکرد تفاوتهایی دارند، که ما با توجه به ابعاد و سایز پروژه ، از small (yolov8s.pt) استفاده میکنیم.

در آخر، مدل را با توجه به پارامتر های مورد نظرمان Train میکنیم؛ که در آن مسیر فایل (در colab)، epoch ها، سایز(تصاویر به سایز ۶۴۰x۶۴۰ پیکسل تغییر پیدا میکنند) ، و سایز batch مشخص شده است.

این مدل ذخیره میشود.

```
from ultralytics import YOLO
model = YOLO("/content/best(1).pt")
```

فایل best(1).pt حاوی وزن ها و پارامترهای مدلی است که قبلا آن را train کرده ایم که معمولا بهترین مدلی است که در طول یک فرآیند آموزشی ذخیره می شود.

در مراحل بعد با استفاده از مدل، تشخیص شی (object detection) را بر روی تصویر انجام میدهیم و کادر های مرزی و تگ را برای شیء مشخص میکنیم. در ادامه توضیحات دقیق تر آماده است:

```

import torch
from torchvision import transforms
from PIL import Image
import cv2
import numpy as np
from matplotlib import pyplot as plt
import numpy as np

# Load an example image
img_path = '/content/5label-5/test/images/Screenshot-2024-07-01-at-12-38-09-PM_png.rf.9a88eb089a0e7c7dd263b80b930a73d4.jpg'
img = Image.open(img_path).convert('RGB')

transform = transforms.ToTensor()
img_tensor = transform(img).unsqueeze(0)

```

در این بخش کتابخانه های مورد نیاز را دریافت میکنیم. عکس را از مسیر مورد نظر گرفته و به فرمت RGB تبدیل میکنیم. سپس آن را به یک تانسور تبدیل میکنیم.

```

# Perform object detection
with torch.no_grad():
    results = model(img_tensor)

for result in results:
    # Get bounding boxes in [xmin, ymin, xmax, ymax] format
    boxes = result.bboxes.xyxy.cpu().numpy()

    # Get class labels
    labels = [model.names[int(cls)] for cls in result.bboxes.cls]

    # Get confidence scores
    confidences = result.bboxes.conf.cpu().numpy()

```

حال از مدل برای تشخیص شیء در در تانسور استفاده میکنیم. از `torch.no_grad()` برای غیرفعال کردن محاسبه گرادیان استفاده می شود که برای استنتاج غیر ضروری است.


```
# Convert PIL image to OpenCV format
img_cv2 = np.array(img)
img_cv2 = cv2.cvtColor(img_cv2, cv2.COLOR_RGB2BGR)
```

این قسمت از کد، تصویر PIL را به آرایه NumPy تبدیل و سپس فضای رنگ را از RGB (استفاده شده توسط PIL) به BGR (مورد استفاده توسط OpenCV) تغییر میدهد. در واقع یک تصویر PIL را به فرمت تصویر OpenCV تبدیل می کند.

```
# Draw bounding boxes
for box, label, confidence in zip(boxes, labels, confidences):
    xmin, ymin, xmax, ymax = map(int, box[:4])
    cv2.rectangle(img_cv2, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)
    # Draw rectangle
    cv2.rectangle(img_cv2, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)

    # Draw label and confidence
    text = f'{label}: {confidence:.2f}'
    cv2.putText(img_cv2, text, (xmin, ymin - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

این کد با استفاده از OpenCV، جعبه های مرزی (bounding boxes) با برچسب ها (labels) و امتیازات اطمینان (confidence) را روی یک تصویر ترسیم می کند.

بخش (xmin, ymin), (xmax, ymax) مشخص کننده پایین-راست و بالا-چپ هستند.

همچنین رنگ سبز جعبه با (۰ و ۲۵۵ و ۰) مشخص شده است. در نهایت ضخامت این جعبه با عدد ۲ مشخص شده است.

در بخش آخر کد، نوشتن لیبل و اطمینان انجام شده که در آن فونت (۰.۵) و رنگ (سبز) و مکان قرار گیری متن (کمی بالاتر از بخش xmin,ymin)

```
# Save the image with bounding boxes
output_path = 'output_image.jpg'
cv2.imwrite(output_path, img_cv2)

# display the image with bounding boxes
plt.imshow(cv2.cvtColor(img_cv2, cv2.COLOR_BGR2RGB))
plt.show()
```

در این بخش تصویر با bounding box در فایل ذخیره شده و با کمک matplotlib نمایش داده میشود.

```
arr = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=np.float32)
arr = arr.reshape(7, 4)
# write the image bounding boxes
for i,(box, label, confidence) in enumerate(zip(boxes, labels, confidences)):
    if label == "box":
        label = 1
        z_real = 0.1
    elif label == "soda":
        label = 2
        z_real = 0.11
    elif label == "mobile":
        label = 3
        z_real = 0.03
    elif label == "biscuite_box":
        label = 4
        z_real = 0.07
    elif label == "mouse":
        label = 5
        z_real = 0.05
x, y, w, h = box[:4]
x = float(x)
y = float(y)
w = float(w)
h = float(h)
```

در این بخش ابتدا یک آرایه ۷ در ۴ با درایه های صفر تعریف میکنیم.

بخش for ، یک مقدار عددی و z_real بر اساس لیبل اختصاص می دهیم. سپس مختصات و ابعاد جعبه مرزی را استخراج کرده و آنها را به شناور (float) تبدیل می کنیم.

```

# Calculate center of bounding box
center_x = x + w / 2
center_y = y + h / 2

a = (center_x/640)-1/2
b = (center_y/480)-1/2

x_real = -(b*0.577)-0.45
y_real = a*0.778

arr[i][0] = label
arr[i][1] = float(x_real)
arr[i][2] = float(y_real)
arr[i][3] = float(z_real)

print(f"Box: {box}, Label: {label}, Confidence: {confidence},center_x: {center_x}, center_y: {center_y}")

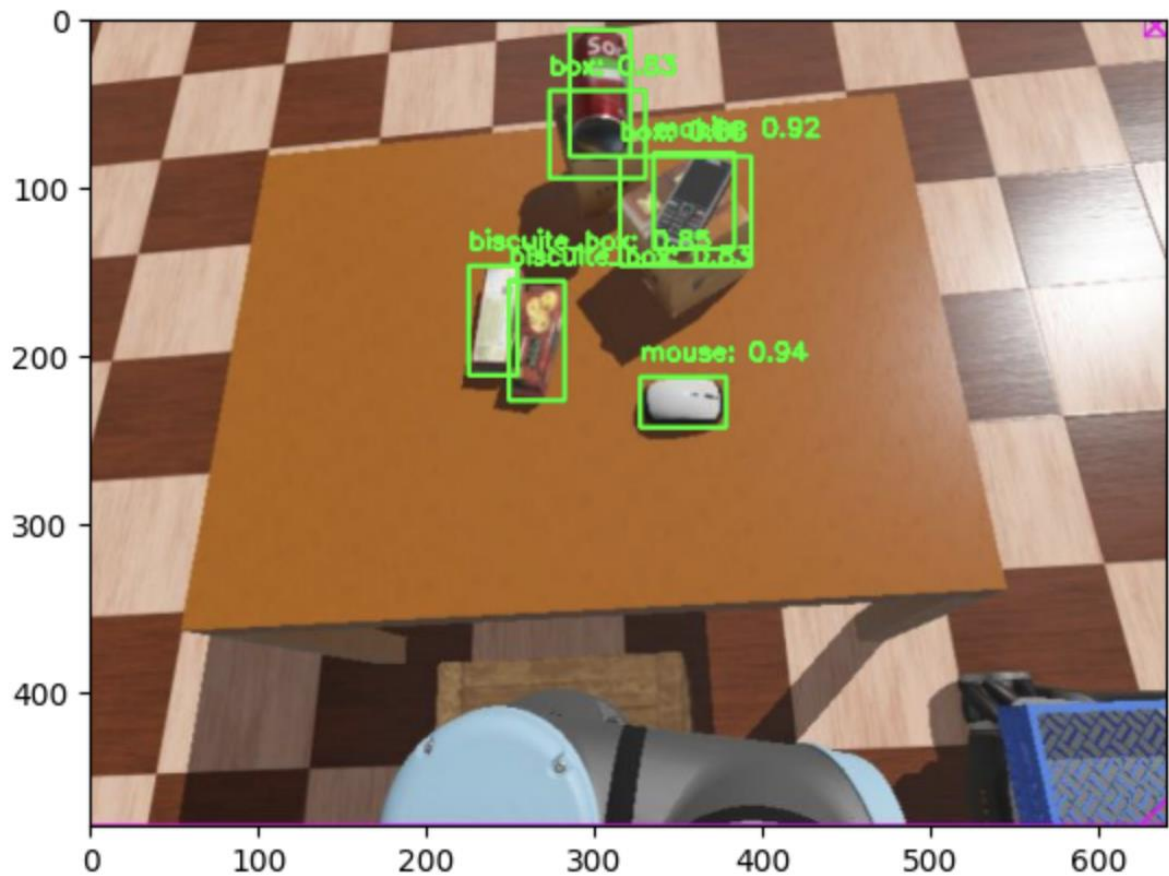
print(arr)

```

در این بخش کد مرکز کادرهای محدود کننده را محاسبه کرده و آنها را به مختصات دنیای واقعی تبدیل می کنیم و نتایج را در یک آرایه NumPy ذخیره می کنیم.

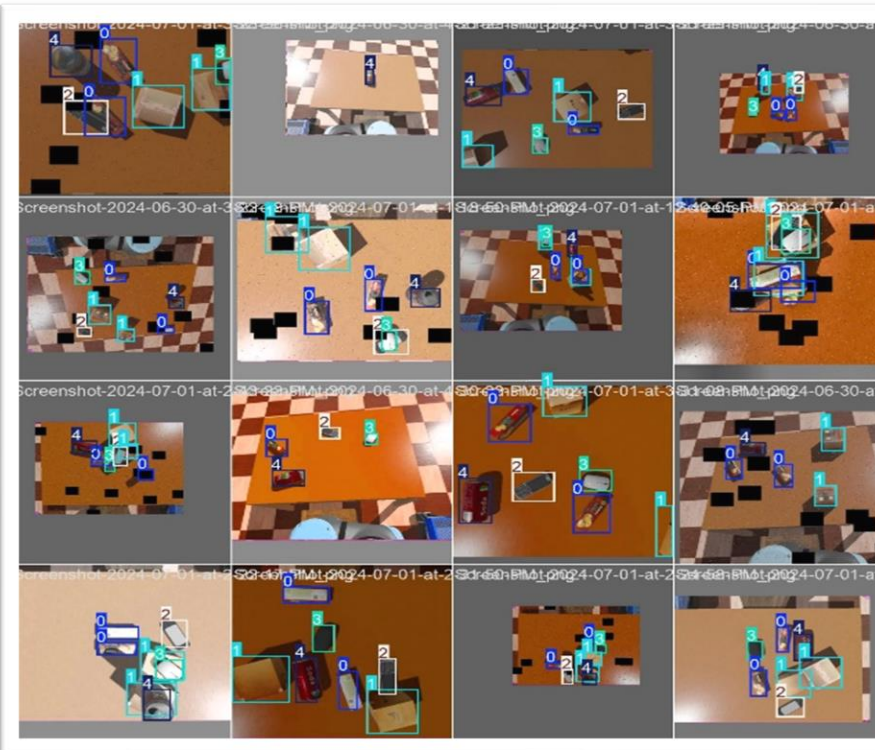
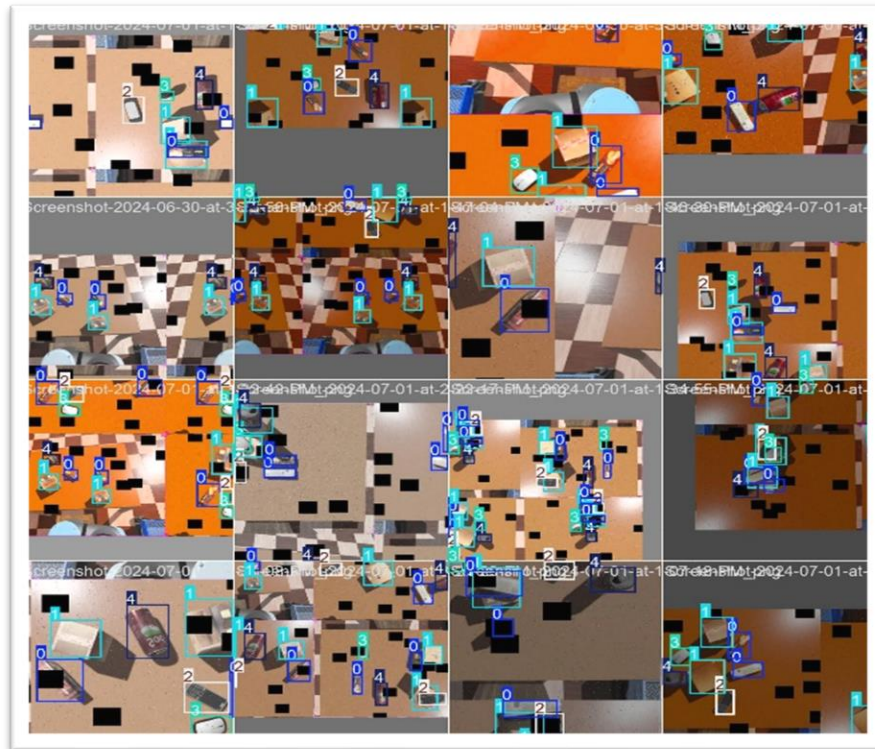
center_y و Center_x نشان دهنده مختصات مرکز جعبه ها هستند. a و b نرمالایز شده مراکز هستند. x_real و y_real این مقادیر نرمالایز شده را به مقادیر واقعی تبدیل میکنند. حال مقادیر بدست آمده را ذخیره میکنیم و در نهایت پرینت میکنیم.

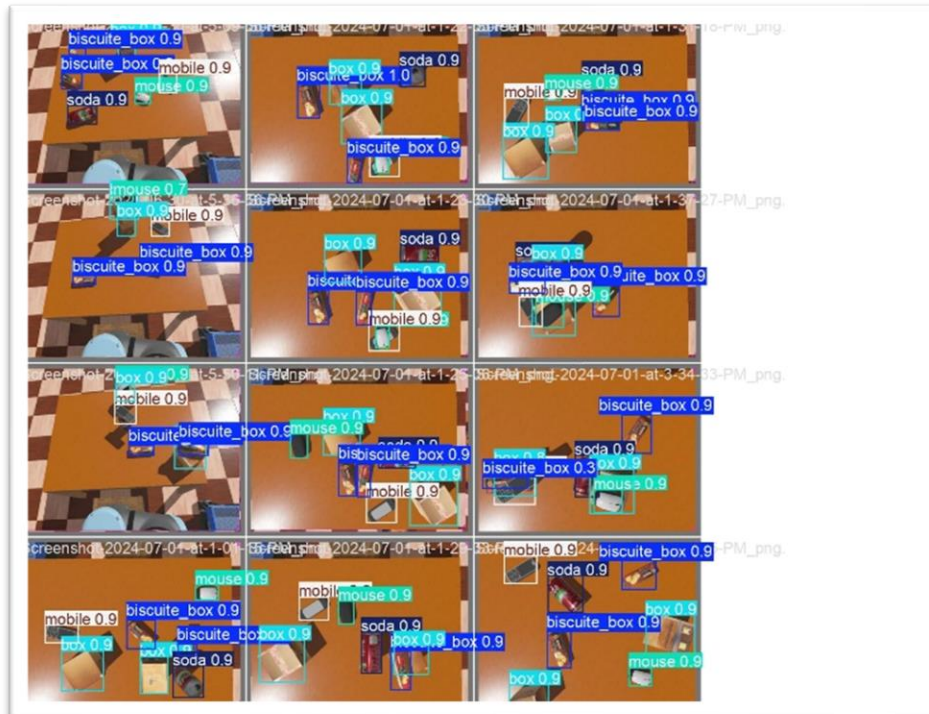
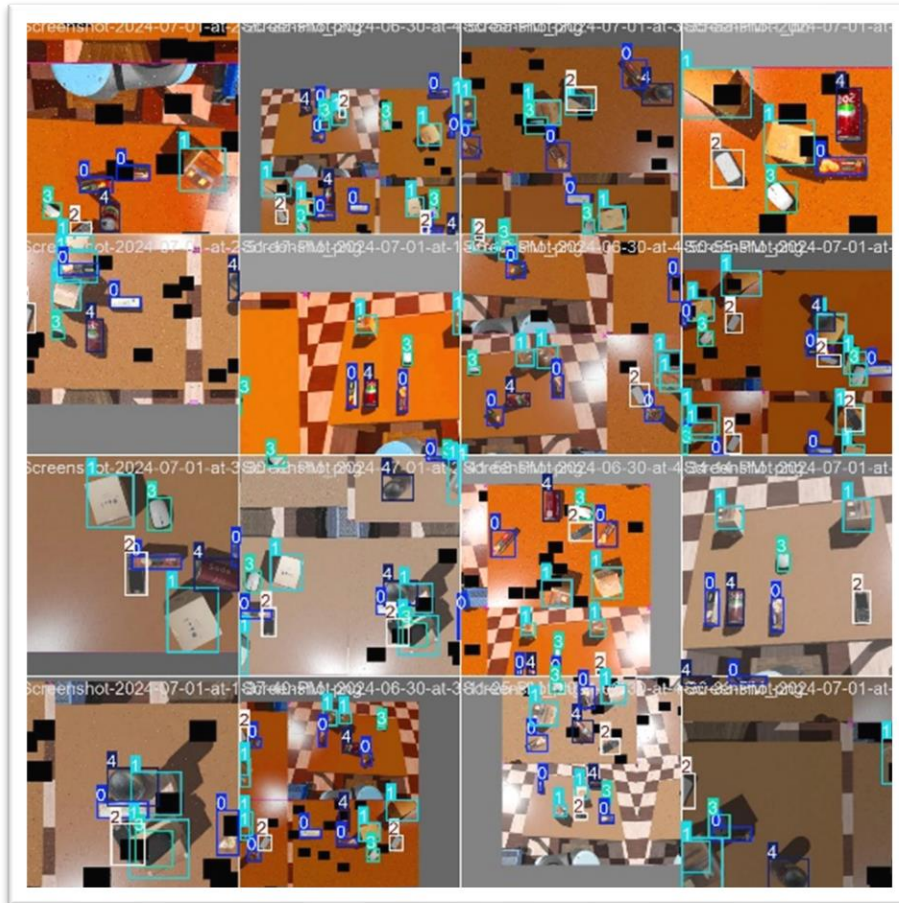
اجرا و ارزیابی مدل

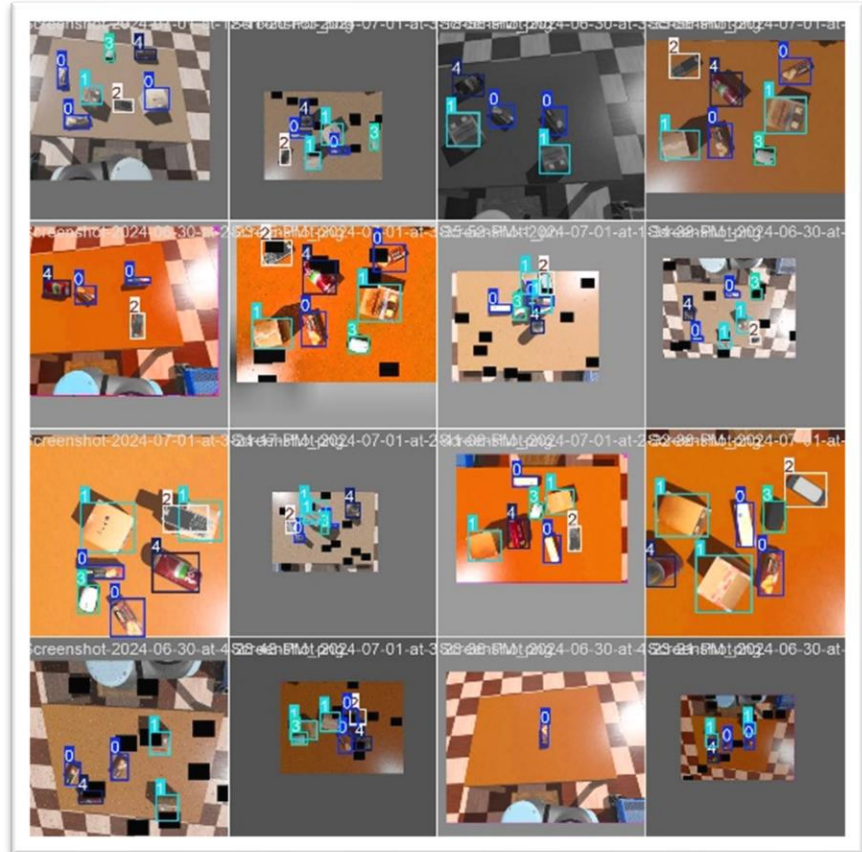


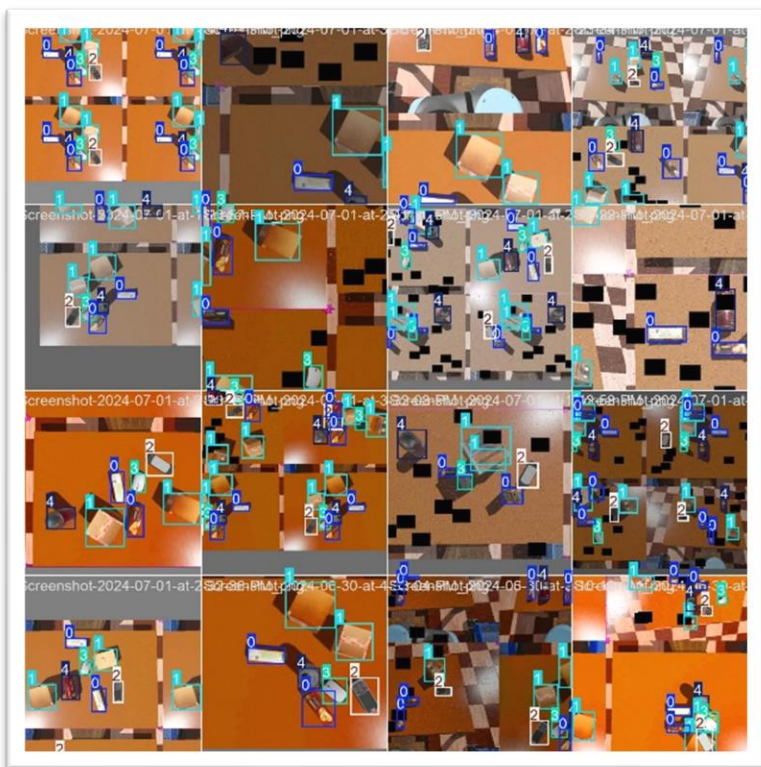
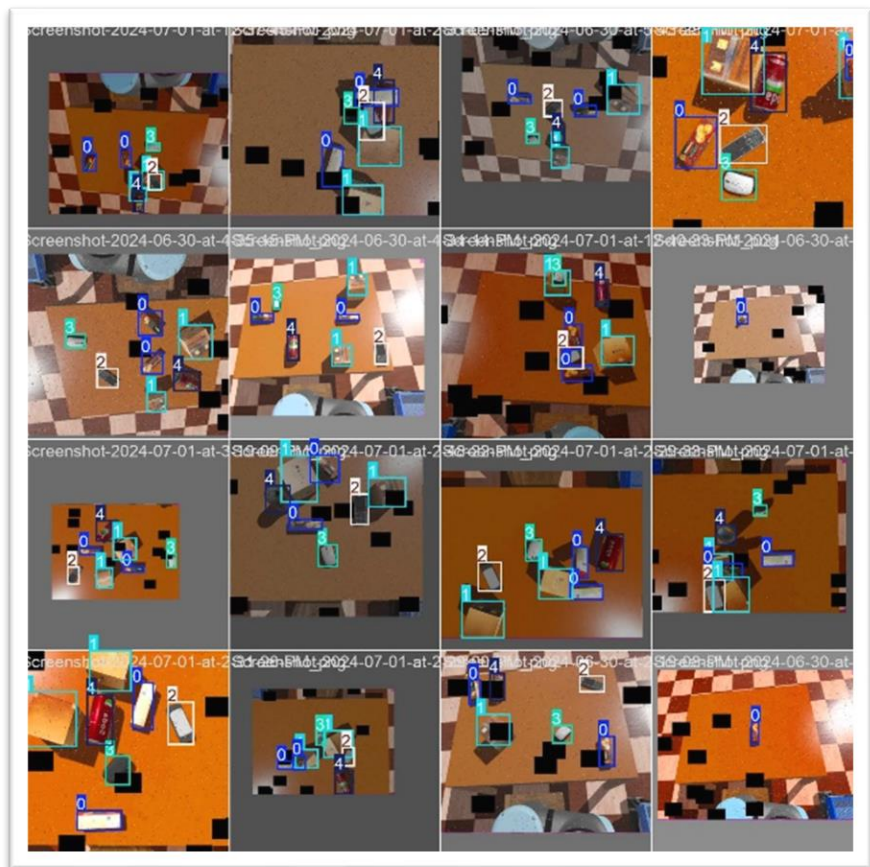
این مدل توانایی تشخیص و شناسایی دقیق اشیاء را دارد، در زوایای مختلف کار میکند و در حالت عمودی بهترین نتیجه را میدهد، اما هر چه این زاویه از حالت ۹۰ درجه خارج شود، از عملکرد مناسب آن کاسته میشود. همچنین بدلیل استفاده از دیتاست متنوع، این مدل توانایی تشخیص حالت های خاص را دارد.

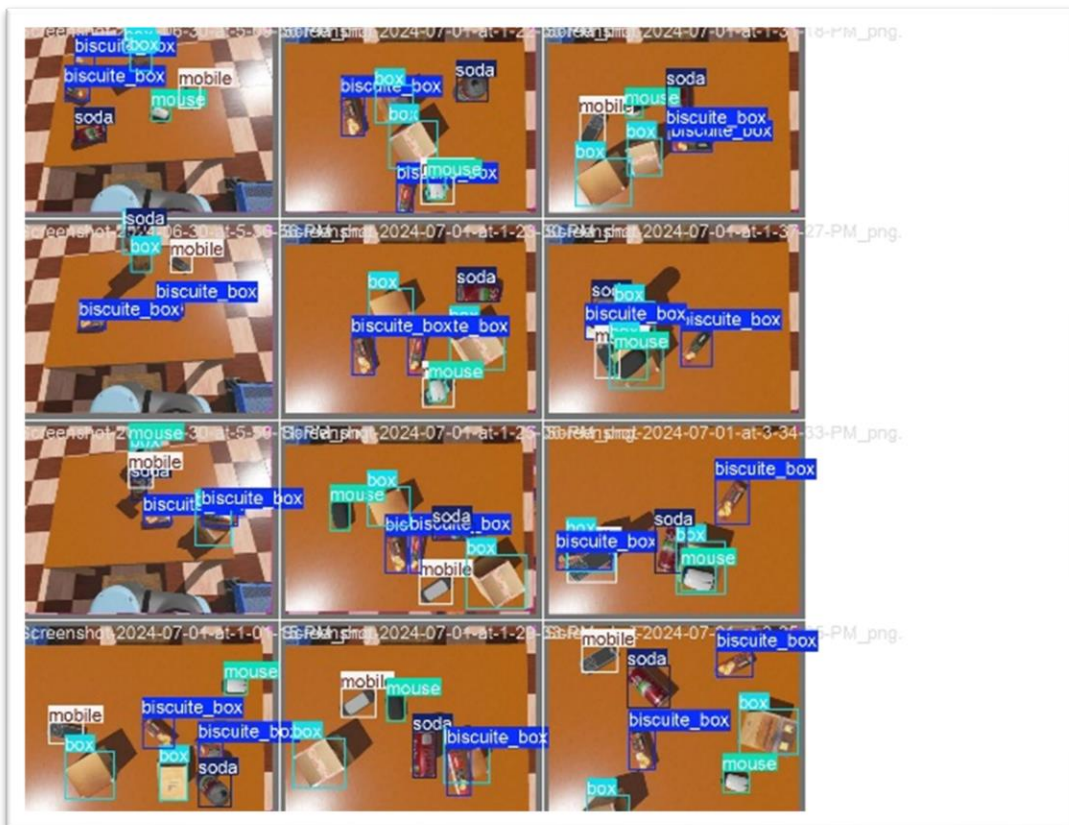
Train and Val batches:



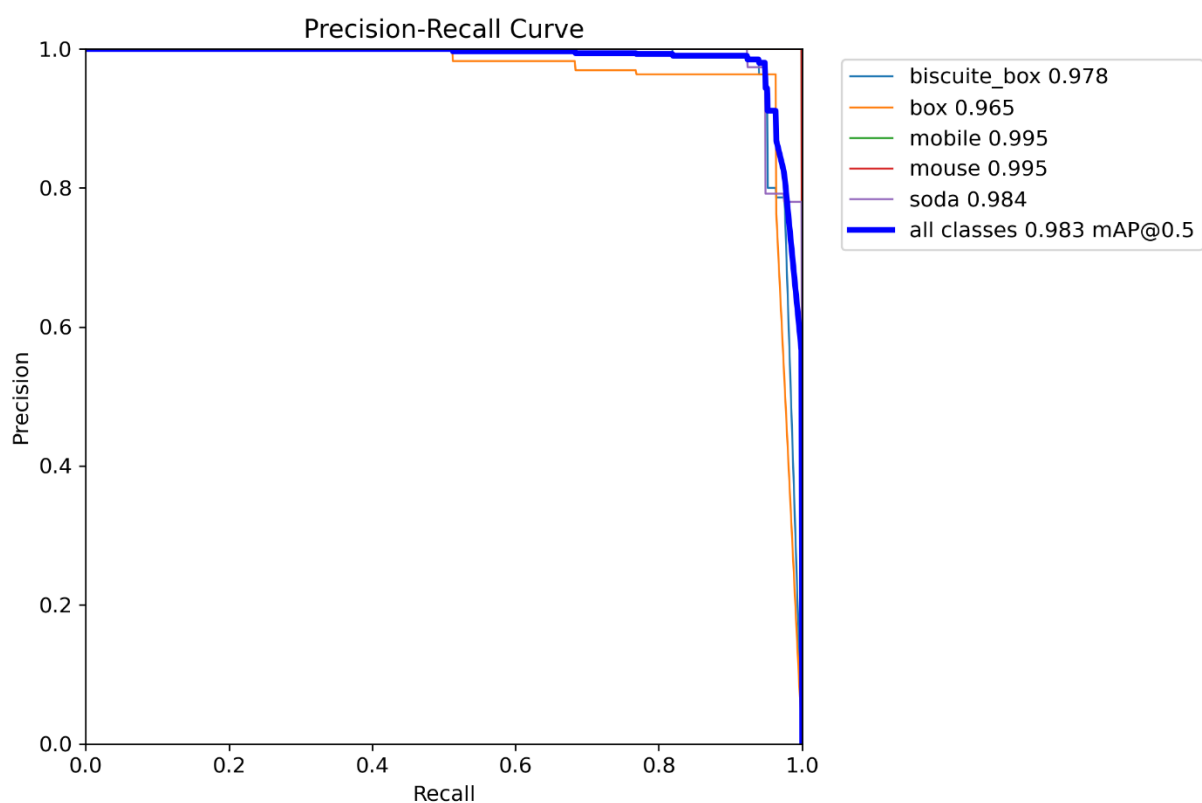
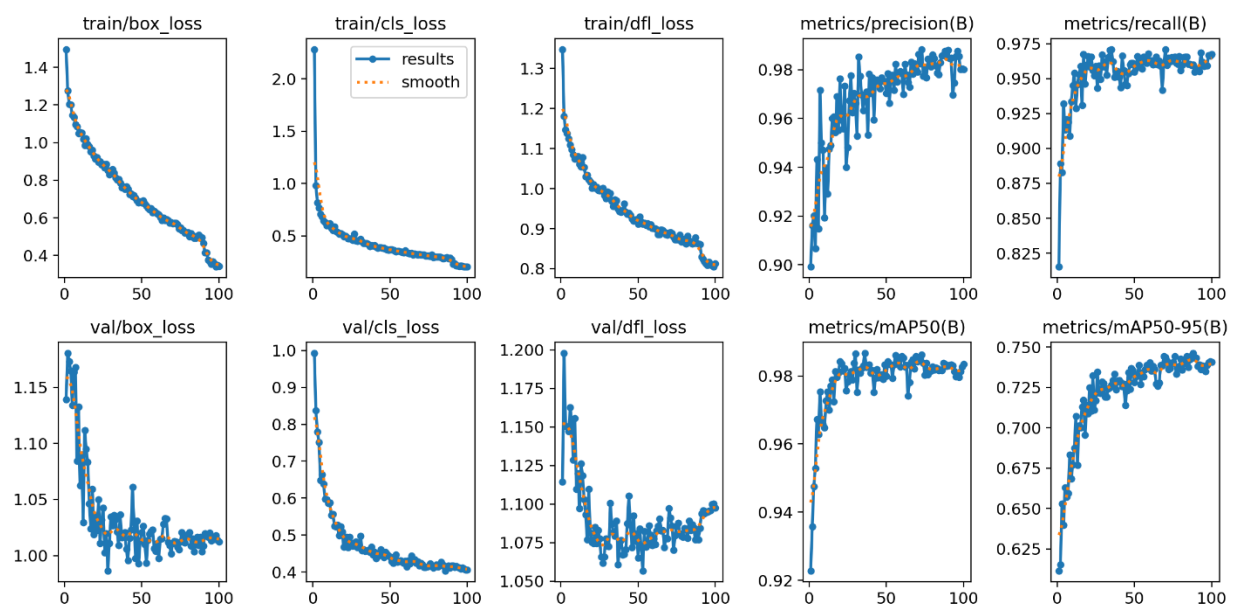


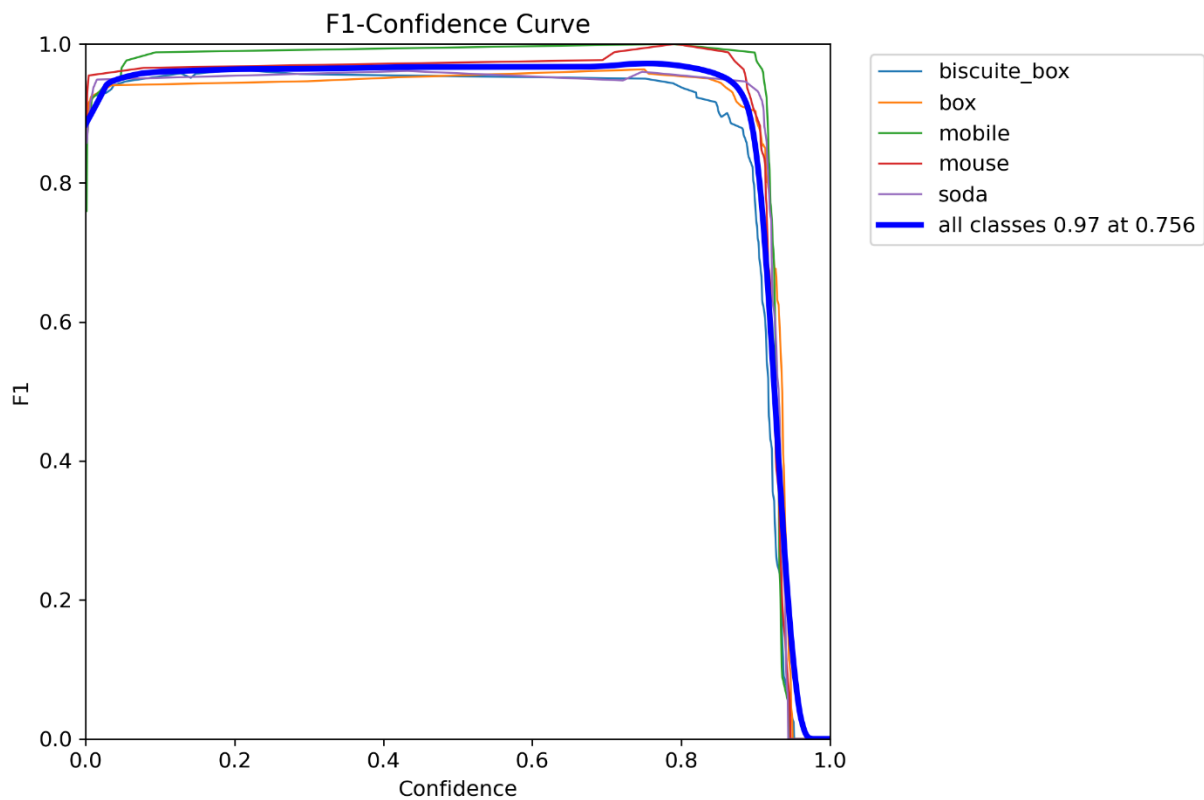
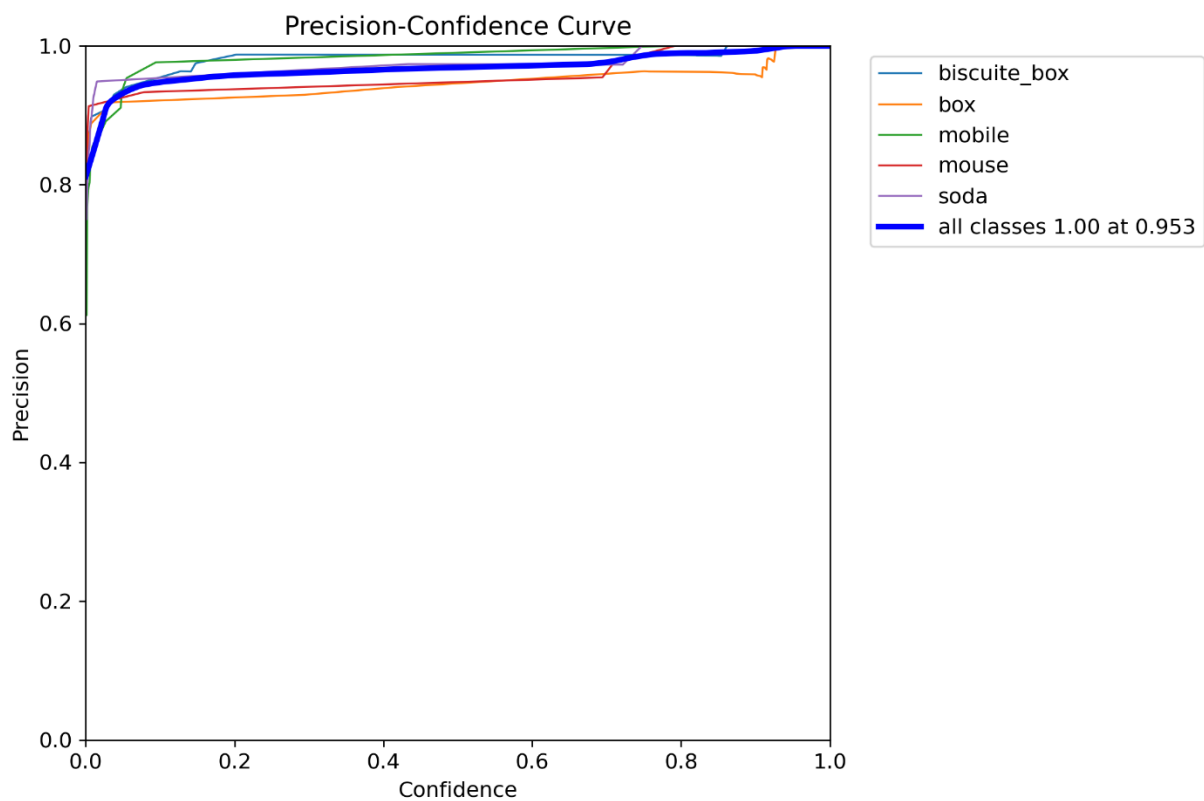


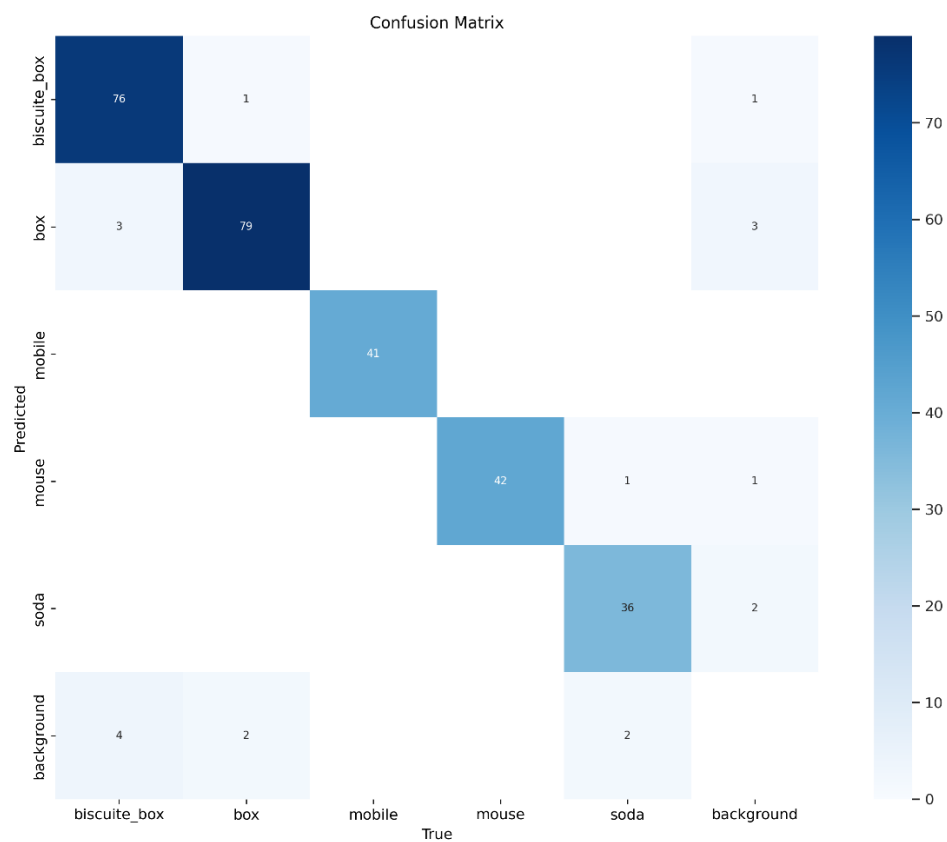
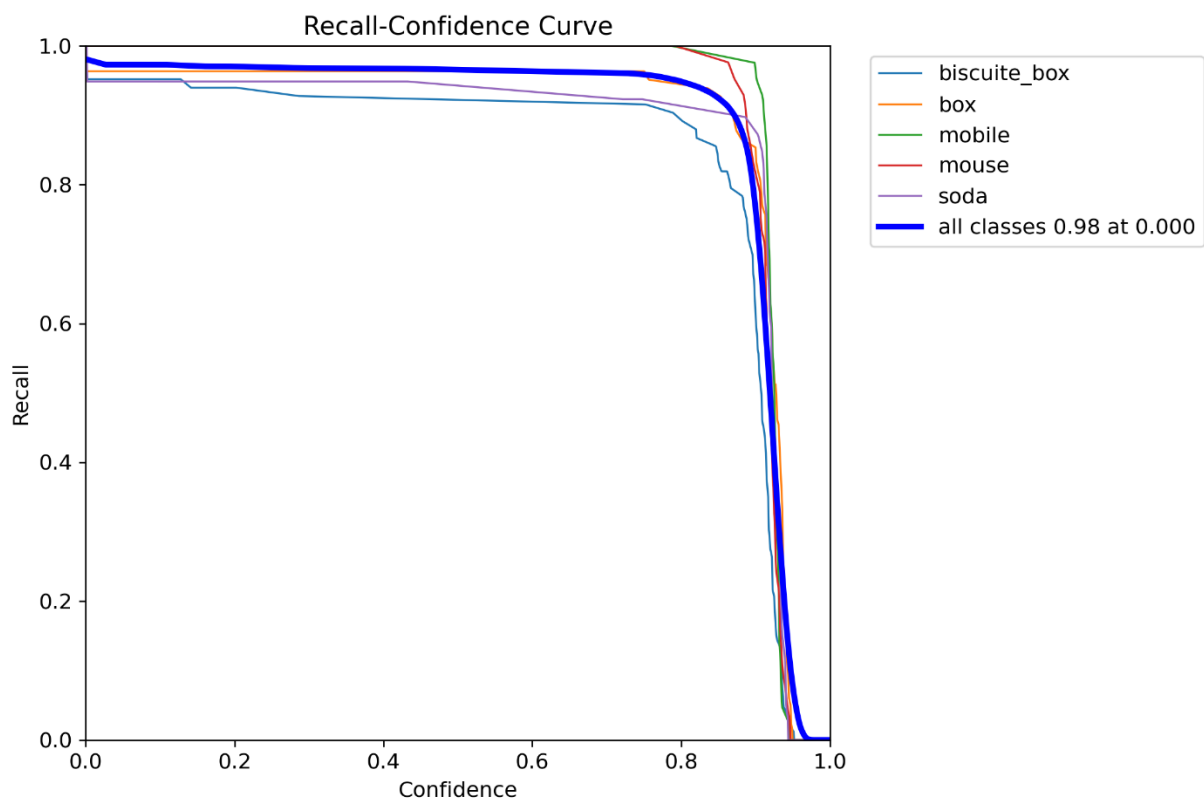


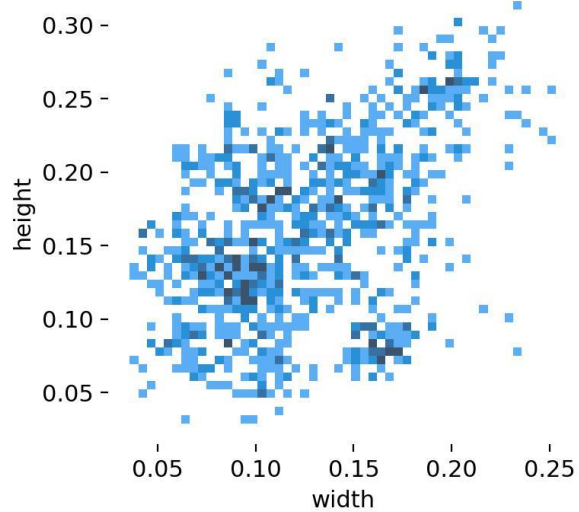
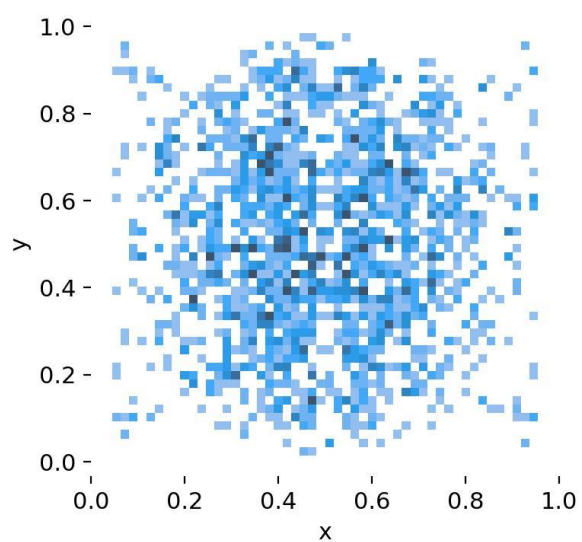
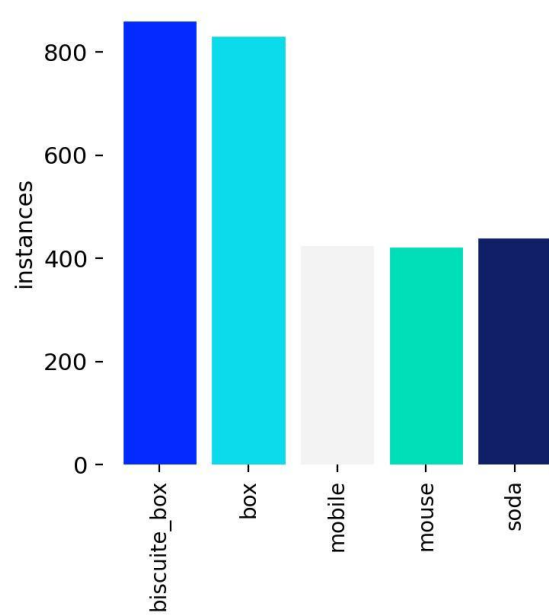


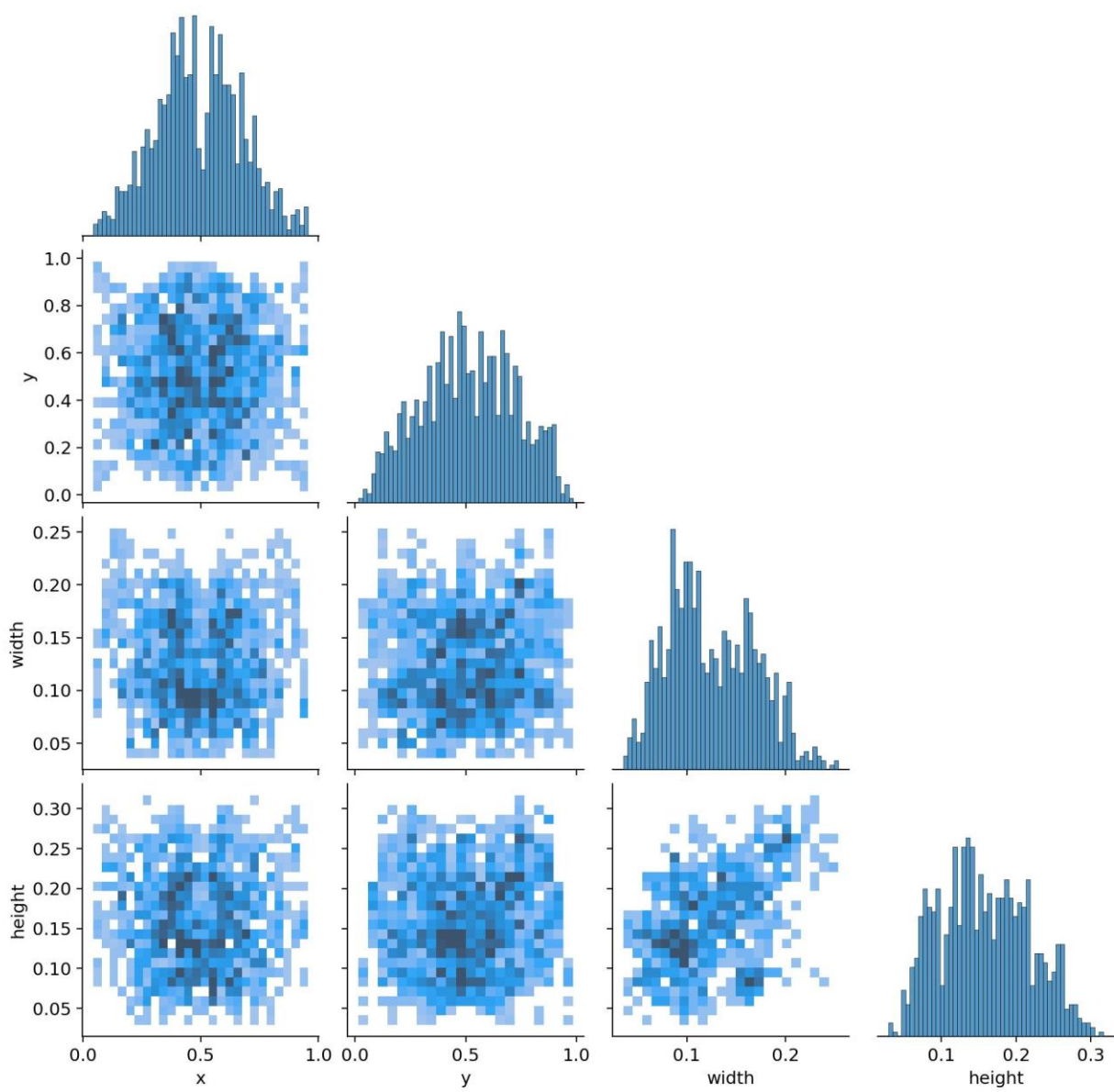
نمودار ها:











Inverse Kinematic

در این بخش ما از مقاله UR5 Inverse Kinematics بهره گرفتیم (لینک آن در بخش منابع موجود است). پارامترهای DH را با توجه به ربات و پروژه تغییر دادیم. برای این مقاله در گیت هاب پیاده سازی انجام شده بود و با یکسری از تغییرات توانستیم از آن کد بهره بگیریم (لینک در بخش منابع). با کمک این دو منبع مشکلات بوجود آمده برطرف شد و توانستیم از آن استفاده کنیم.

$$\vec{P}_5^0 = T_6^0 \begin{bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\theta_1 = \psi + \phi + \frac{\pi}{2}$$

$$\psi = \text{atan2}((P_5^0)_y, (P_5^0)_x)$$

$$\phi = \pm \arccos\left(\frac{d_4}{(P_5^0)_{xy}}\right) = \pm \arccos\left(\frac{d_4}{\sqrt{(P_5^0)_x^2 + (P_5^0)_y^2}}\right)$$

$$\theta_5 = \pm \arccos\left(\frac{(P_6^1)_z - d_4}{d_6}\right)$$

$$T_1^6 = ((T_1^0)^{-1} T_6^0)^{-1}$$

$$-\sin(\theta_6)\sin(\theta_5) = z_y$$

$$\cos(\theta_6)\sin(\theta_5) = z_x$$

$$\theta_6 = \text{atan2}\left(\frac{-z_y}{\sin(\theta_5)}, \frac{z_x}{\sin(\theta_5)}\right)$$

$$T_4^1 = T_6^1 \ T_4^6 = T_6^1 \ (T_5^4 \ T_6^5)^{-1}$$

$$\vec{P}_3^1 = T_4^1 \begin{bmatrix} 0 \\ -d_4 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\cos(\xi) = \frac{||\vec{P}_3^1||^2 - a_2^2 - a_3^2}{2a_2a_3}$$

$$\theta_3 = \pm \arccos\left(\frac{||\vec{P}_3^1||^2 - a_2^2 - a_3^2}{2a_2a_3}\right)$$

$$\theta_2 = -\text{atan2}((P_3^1)_y, -(P_3^1)_x) + \arcsin\left(\frac{a_3\sin(\theta_3)}{||\vec{P}_3^1||}\right)$$

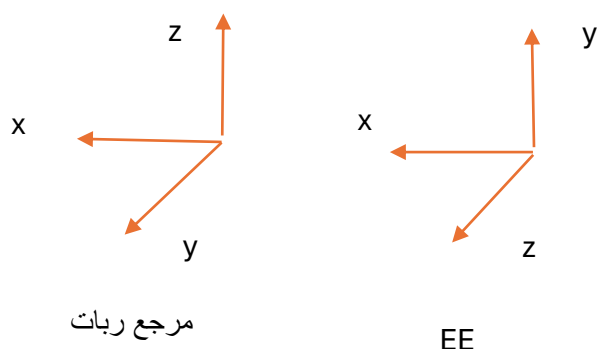
$$T_4^3 = T_1^3 T_4^1 = (T_2^1 T_3^2)^{-1} T_4^1$$

$$\theta_4 = \text{atan2}(x_y, x_x)$$

یکی از مشکلات ما به این شرح بود که رباتی که در مقاله است و مقادیر $q_1=0, q_2=0, q_3=0$ برای آن تعریف شده است با webots تفاوت دارد؛ در ویبات ربات ما مقادیر مفصل اول و دوم جابجا هستند و همچنین مقدار مفصل دوم ویبات باید + یا - مقدار ۳.۱۴ شود تا مانند ربات در مقاله شود.

نکته بعدی این است که xyz که کد از ما میگیرد نسبت به base ربات هستند. در مختصات دستگاه base ربات ، x و y با x و y کل اتاق (وسایل و ...) تطابق دارند اما z آنها به اندازه ۰.۶ تفاوت دارد. پس ما میتوانیم از x و y که از شبکه عصبی میگیریم با اطمینان استفاده میکنیم و دقت میکنیم که z وسایل نسبت به base سنجیده شده و در ادامه کد، z را برای هر کدام از وسایل مشخص میکنیم.

برای استفاده از کد inverse kinematics ابتدا باید position و orientation ، EE را نسبت به بیس ربات پیدا کنیم.



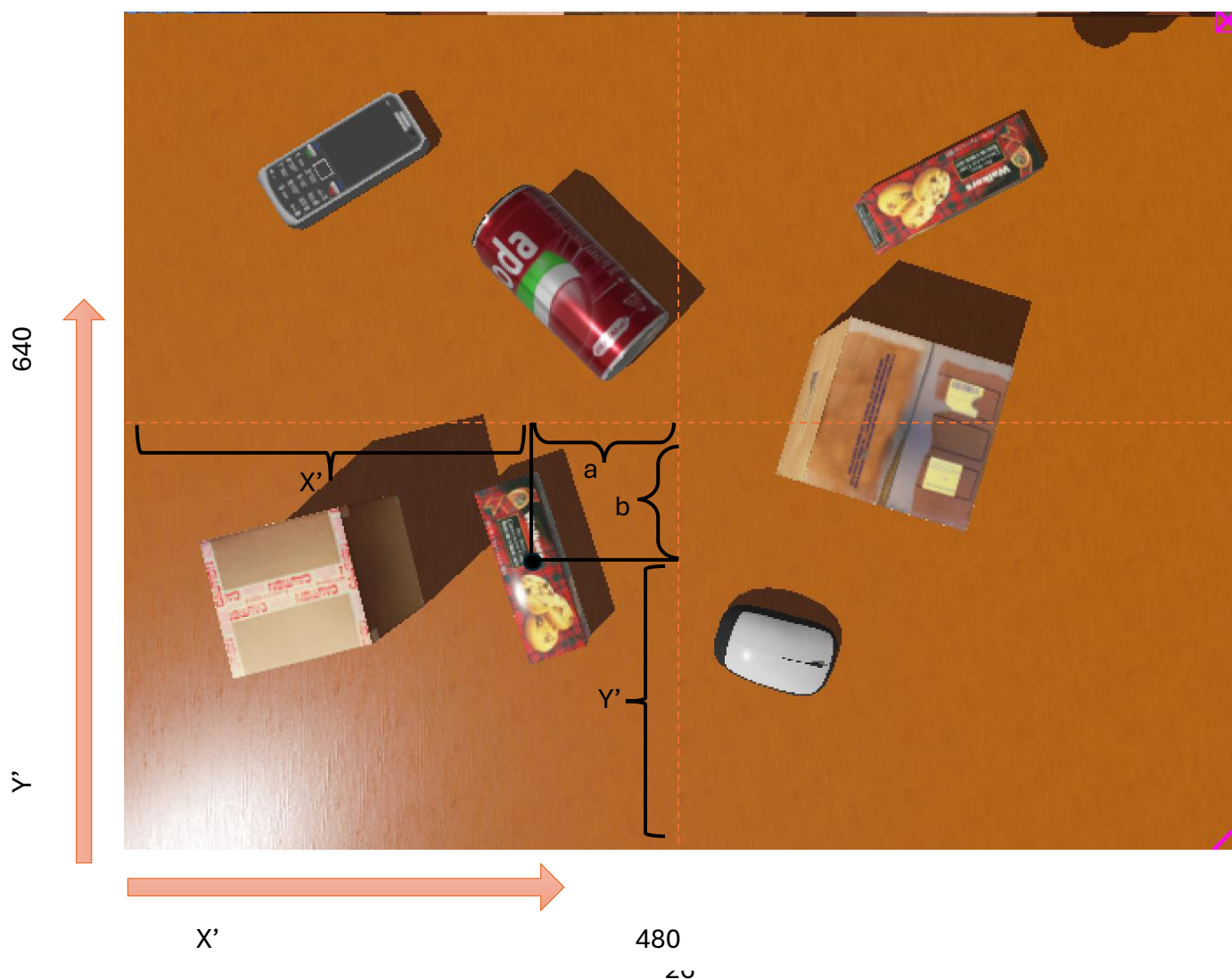
با توجه به اینکه در مختصات دوم حول x ، ۱۸۰ درجه دوران داشتیم ماتریس تبدیل به صورت زیر بدست می آید:

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & -1 & 0 & y \\ 0 & 0 & -1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
position = np.array([[1, 0, 0, x], [0, -1, 0, y], [0, 0, -1, z], [0, 0, 0, 1]], dtype = np.float32)
```

یکی دیگر از مشکلات تبدیل خروجی مرکز bounding box (که از عکس دوربین بدست می آید) به مختصات x و y که نسبت به دستگاه مرجع ربات است.

عکس از دوربین ربات:



```
a = (1/2)-(center_x/640)
b = (-1/2)+(center_y/480)
```

```
x_real = -(b*0.577)-0.45
y_real = a*0.788
```

برای تبدیل این مقادیر که اعدادی بین 1 تا 640 و 1 تا 480 هستند به x و y در مختصات مرجع ربات، ابتدا a و b که اعدادی بین 0 و 1 هستند را بدست می‌آوریم به همان شکلی که در شکل بالا نشان داده شده است. بعد از روی این مقادیر x و y چهار گوشه میز، x و y نقاط روی میز را بدست می‌آوریم.

مختصات چهارسر میز:

	x	y
d1	-0.16	0.385
d2	-0.16	-0.385
d3	-0.737	0.385
d4	-0.737	-0.385

منابع

→ کلاس های کارگاه

→ <https://roboflow.com/>

→ https://tianyusong.com/wp-content/uploads/2017/12/ur5_inverse_kinematics.pdf

→ https://github.com/mc-capolei/python-Universal-robot-kinematics/blob/master/universal_robot_kinematics.py