# UFACTORY LITE 6 OBB INTEGRATION

Cobot model: Ufactory lite 6

Cobot Software Development Kit: xArm-Python-SDK

Computer vision package: Ultralytics

Computer vision type of detection: Oriented Bounding Boxes (OBB)

Computer vision framework: PyTorch

Computer vision dataset and annotation: Roboflow

## Computer Vision setup for OBB detection.

- **Create python virtual environment (Optional):**

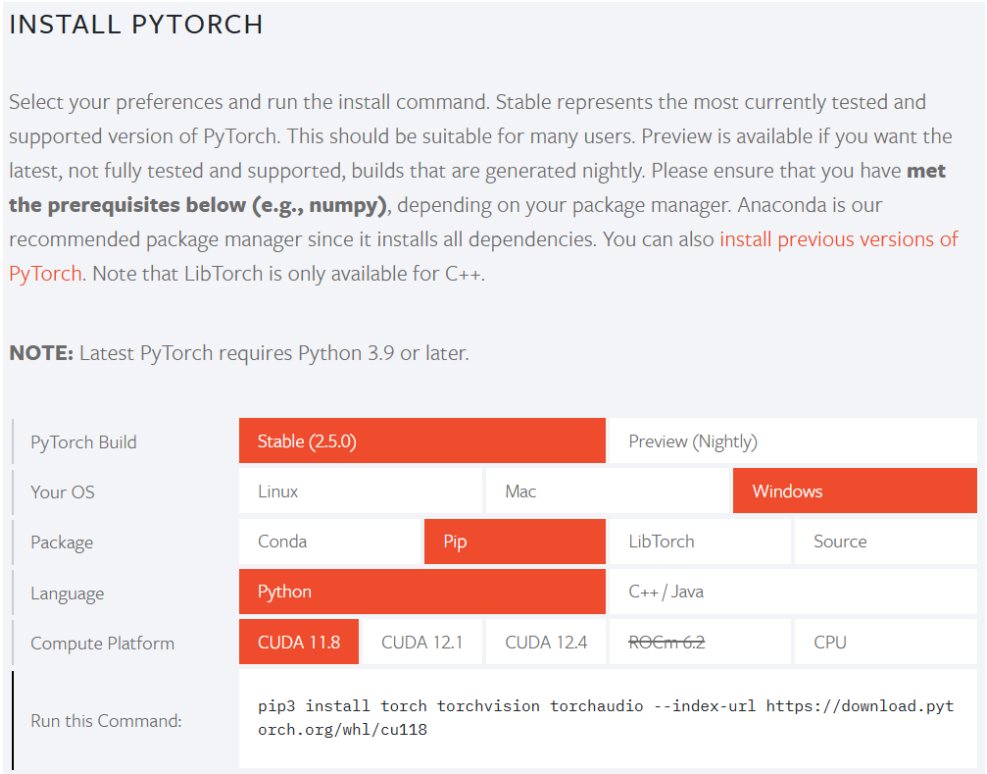
  python -m venv C:\path\to\new\virtual\environment

- **Install the ultralytics package from PyPI**

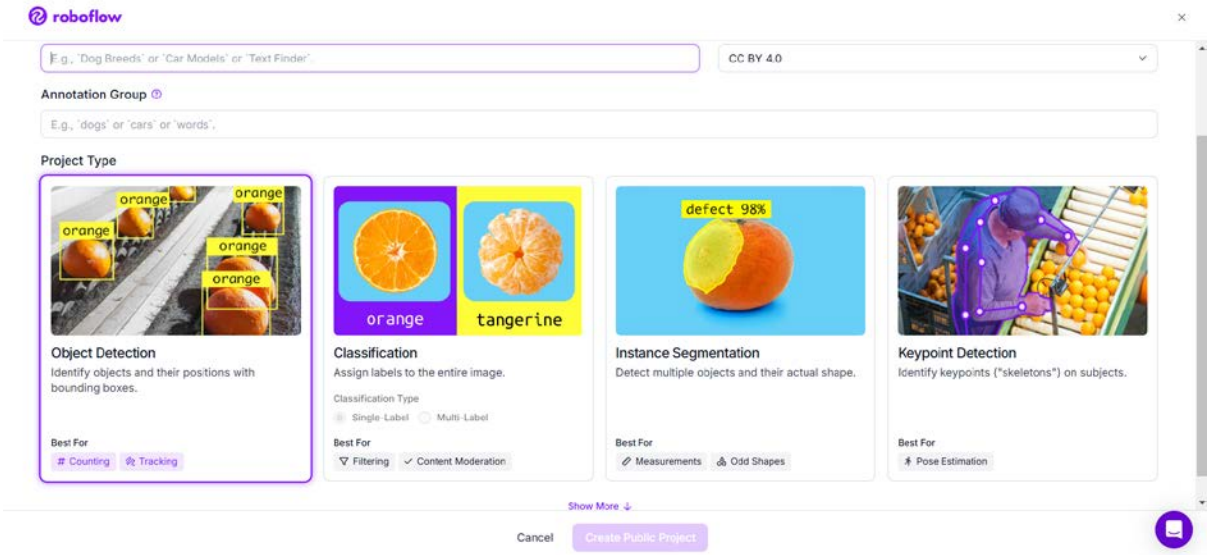
  pip install ultralytics

  **Other installation is also supported (Conda, Git clone and Docker)*

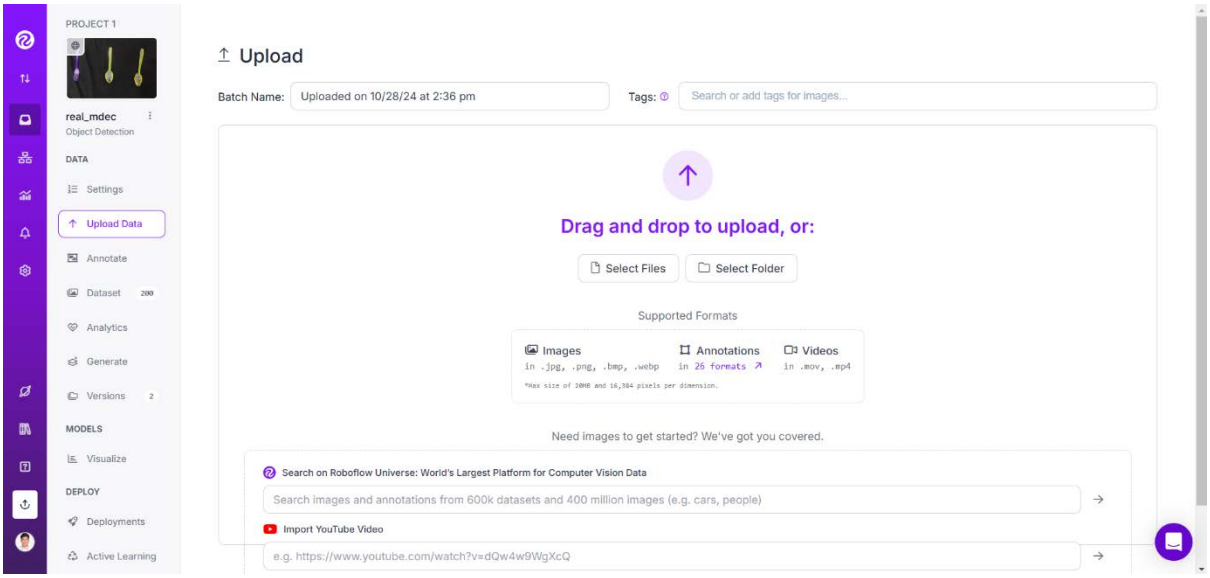
- **Install PyTorch**
  Choose the correct configuration before installing.
  **All the dependencies should already be installed when installing ultraytics.*
  **Understand the Compute Platform CUDA here [https://en.wikipedia.org/wiki/CUDA](https://en.wikipedia.org/wiki/CUDA)*



- **Creating a new project on Roboflow**
  Choose the Object Detection as the project type for Object Bounding Boxes

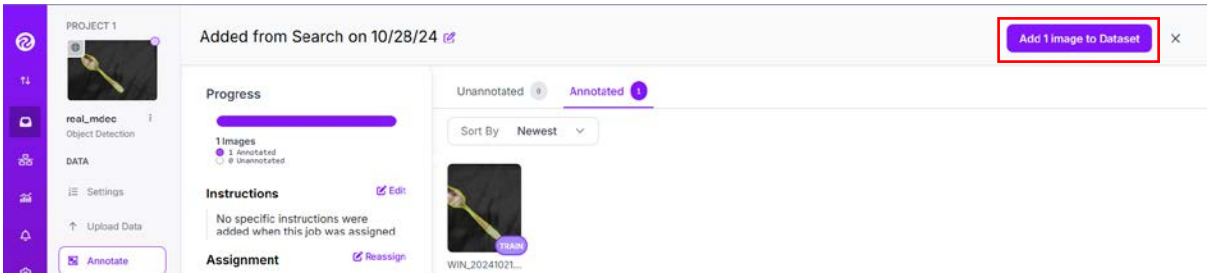



- **Prepare data for annotation in Roboflow**
  Upload the data to roboflow

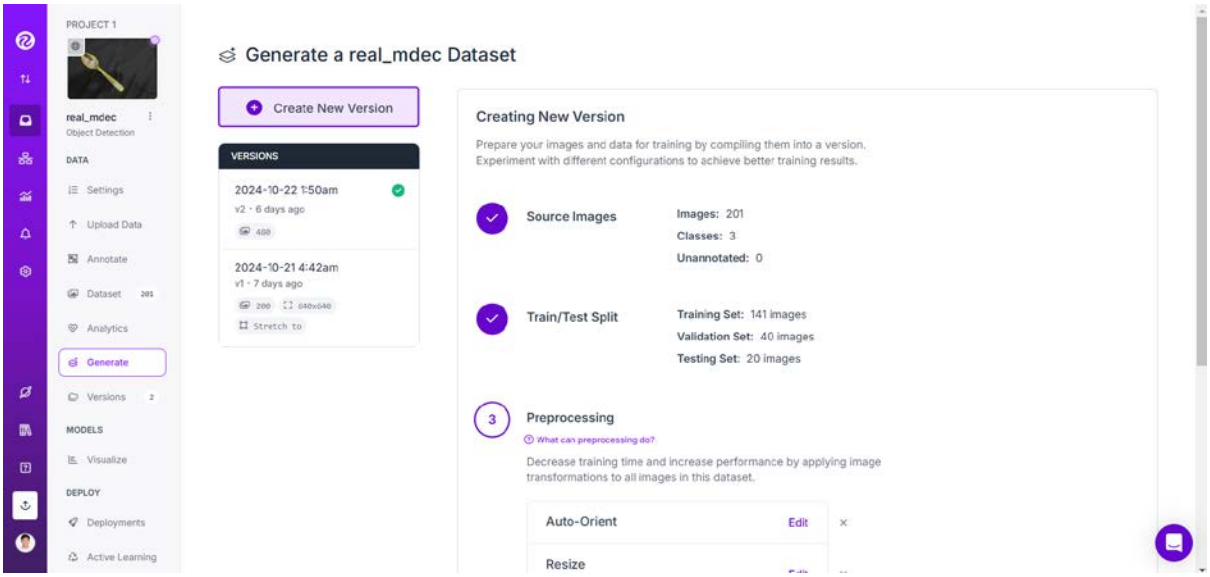


- **Generate the dataset**

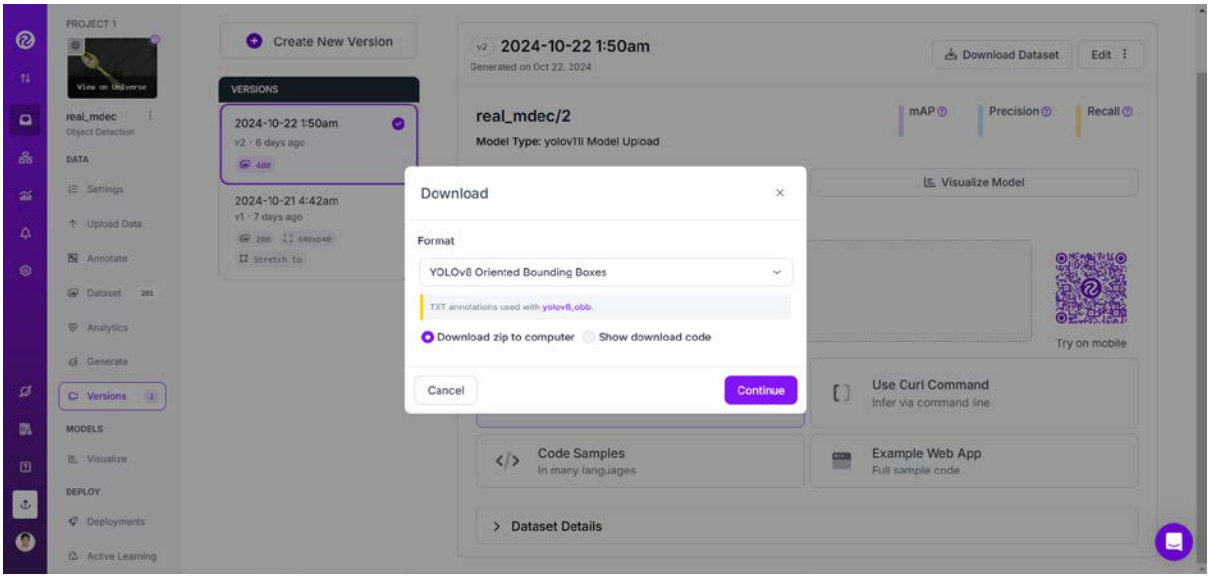
  After completing the annotation, add it all to the dataset.




  Configure the dataset to create the new version of the dataset




  Download the dataset and choose the correct format (as for this project, select YOLOv8 Oriented Bounding Boxes) and download zip to computer.



- **Train the dataset using ultralytics**
  Extract the dataset zip file and copy the dataset file into ultralyics datasets folder from the workspace directory.
  ** "datasets" folder will be available inside workspace directory when follow ultralytics quickstart python guide or it can be manually created.*

  Create a python script to train the dataset.

```python
from ultralytics import YOLO

# Load a pretrained model
model = YOLO("yolo11n-obb.pt")

# Train the model
results = model.train(data="path/to/custom_dataset.yaml", epochs=100, imgsz=640)
```

  **Edit the .yaml file of the dataset to the correct path. To understand more, read the code comment from*
  *https://github.com/ultralytics/ultralytics/blob/main/ultralytics/cfg/datasets/DOTAv1.yaml*
  *https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#option-2-create-a-manual-dataset*

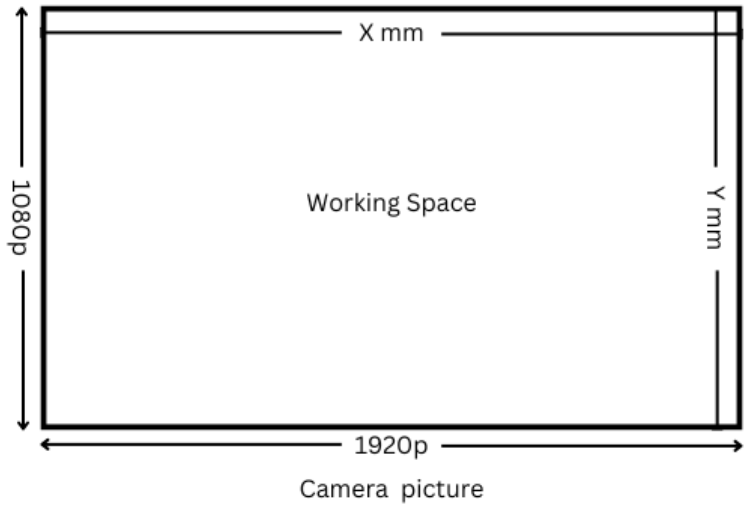
- **Using custom model to predict**
  After training is completed, get the weight usually named "best.pt" and "last.pt". That is the custom trained model.
  **Recommended to use best.pt*

```python
from ultralytics import YOLO

# Define path to the model file
model = YOLO("patch/to/custom.pt") # Load a custom model

# Define path to the image file
source = "path/to/image.jpg"

# Run inference on the source
results = model(source)  # list of Results objects
```

## Cobot setup.

- **Install the robot.**
  Follow the instructions from ufactory lite 6 user manuals.
  **For this documentation, robot installation is on the table.*

- **Set the working space.**
  To set up the working space, attach the camera to the robot end factor facing downward. The image of the camera is the working space for the robot. After that, get the measurement of the working space.




**When setting up the working space, make sure it is within the working range of the robotic arm.*
***For this documentation, the camera resolution and the image resolution width=1920p and height=1080p*

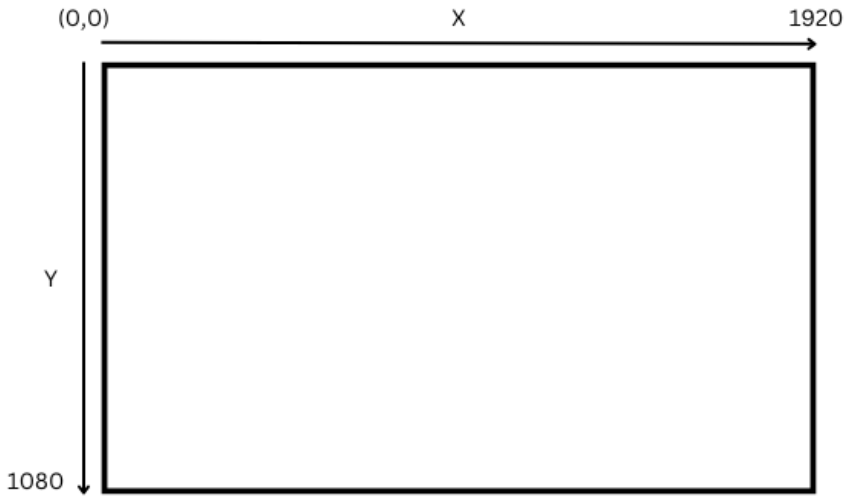
## Cobot and computer vision integration.

- **Extract the information from the OBB**
  Read the results reference on the ultralytics website for storing and manipulating Oriented Bounding Boxes (OBB).
  *[https://docs.ultralytics.com/reference/engine/results/#ultralytics.engine.results.OBB](https://docs.ultralytics.com/reference/engine/results/#ultralytics.engine.results.OBB)

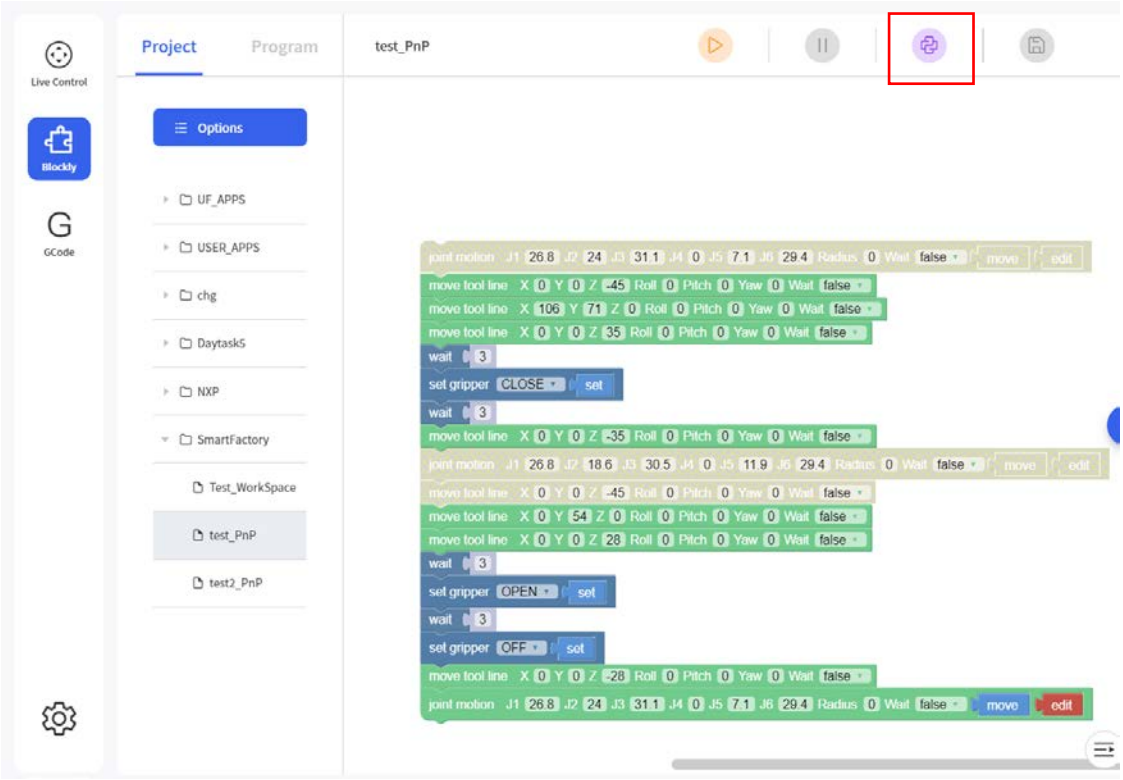
  This is the example of the code where it can detect multiple class id (object) center and rotation:

```python
# Iterate over each detected object
for i in range(len(obb.cls)):
    # Get the class name and confidence for each detected object
    det_class_id = obb.cls[i]  # Class ID of the i-th detection
    det_name = names[int(det_class_id)]  # Class name
    confidence = obb.conf[i]  # Confidence score of the i-th detection

    # Filter detections based on class name and confidence threshold
    if det_name in target_classes and confidence >= 0.6:
        # Get the x, y coordinates from the obb.xywhr (x_center, y_center, width, height, rotation)
        x = obb.xywhr[i][0]  # x_center
        y = obb.xywhr[i][1]  # y_center
        angle = obb.xywhr[i][4]
        angle_degrees = (angle * (180 / math.pi)) + 90
```

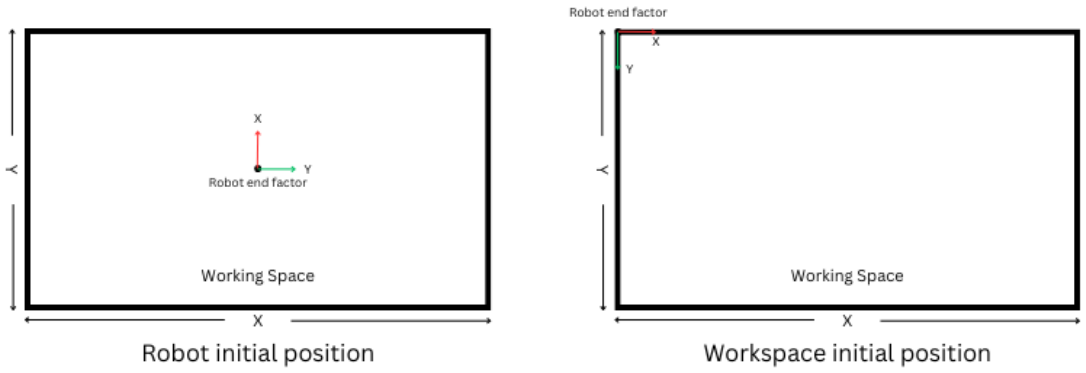
  When using the “xywhr” attribute, the predict format is in pixel and the rotation is in radian. The origin of X and Y is on the top left.



- **Create robot movement.**
  Create the robot movement using blocky. Can convert it to python code to make it easy to integrate with computer vision (OBB).




Since the information extracted is in X (width) and Y (height), it's recommended to set the robot initial position and configure it to new workspace initial position.
**For OBB, x and y refer to the x_center and y_center of detected object.*
***Make sure the robot end factor is offset from the camera*




The information extracted format is in pixel and the rotation in radian. To make it easy to integrate, convert the rotation from radian to degree and scale the workspace to represent how many millimeters correspond to one pixel in each dimension. The scaling factor for width is $\frac{Workspace\ width}{Pixel\ width}$ and scaling factor for height is $\frac{Workspace\ height}{Pixel\ height}$. To get the measurement in millimeters, the formula for width is

$W = x_{obb} \times$ width scaling factor, and height is $H = y_{obb} \times$ height scaling factor. Set the W, H and degree as the variable on the “linear motion [variable]” block for X, Y and Yaw. “linear motion [variable]” block will move to the set value in millimeter from the initial position.