Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Intern Work Progress
RoLand

Joaquin Arias

July 18, 2023

DFKI Labor Bremen
Robert-Hooke-Str. 5
28359 Bremen

Table of Contents

1 Dataset Creation

1.1 Rosbag to Video

The images used for training, validating and testing were obtained from rosbag files from the robot's webcam. To convert this into a video, the file **ros2bag2video.py** was modified in the following way:

Line 234 :

```
elif msg_encoding.find('rgb8') != -1 or
      msg_encoding.find('8UC3') != -1:
```

Line 241 :

```
print('Unsupported encoding:', msg_encoding)
```

Line 303 and 304 : Delete

Line 306 :

```
img = self.bridge.imgmsg_to_cv2(msg)
img = cv2.rotate(img, cv2.ROTATE_180)
```

Then the following command was run and the images were extracted:

```
python3 ros2bag2video.py --topic /video_frames /path/to/bag/
```

The bag files and the python script must be in the same directory. The used videos for the datasets creation were: **rosbag2_2022_05_11-15_-10_47** and **exp5/rosbag2_2022_05_11-15_39_43**.

In order to use only good quality images, a laplacian filter was applied to the images to determine which ones were blurry. For this, run the **detect.py** script:

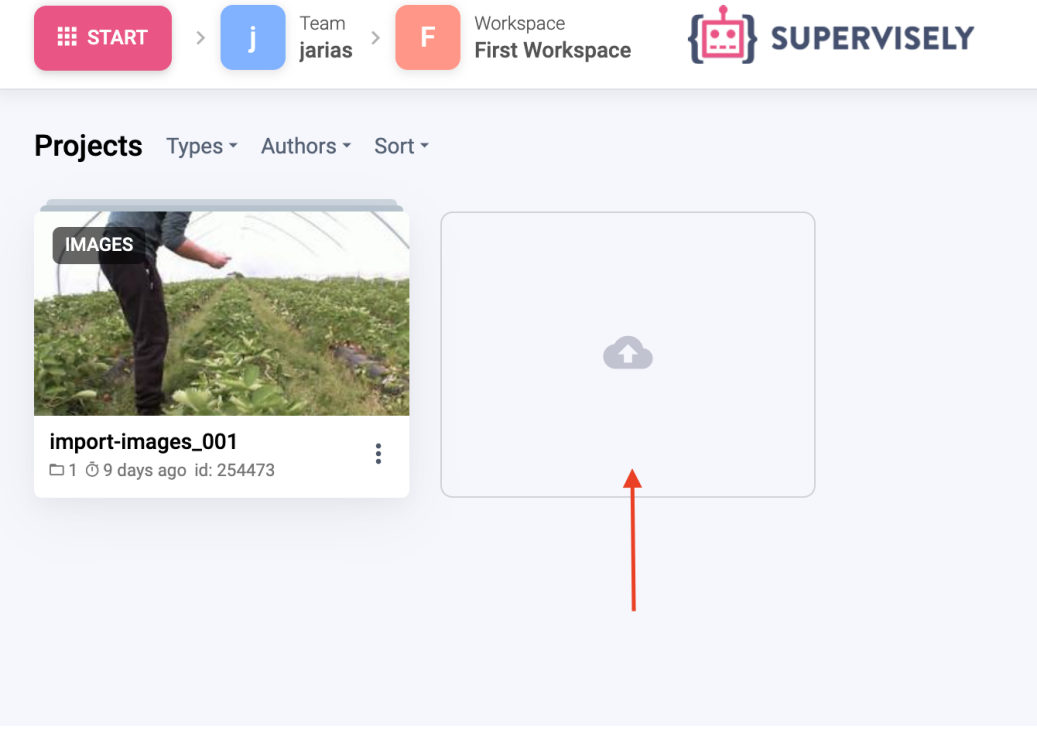
```
python detect.py /path/to/blurry/folder /path/to/output/folder
--threshold 500
```

1.2 Labeling

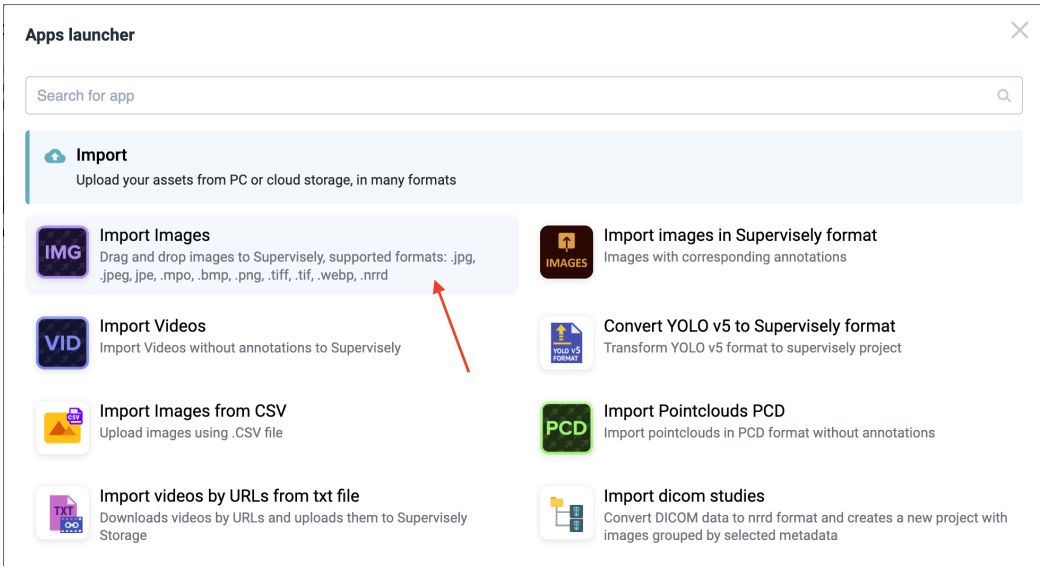
There exist several software that can be used for labeling. However, in this case, the Supervisely Platform was used.

Procedure:

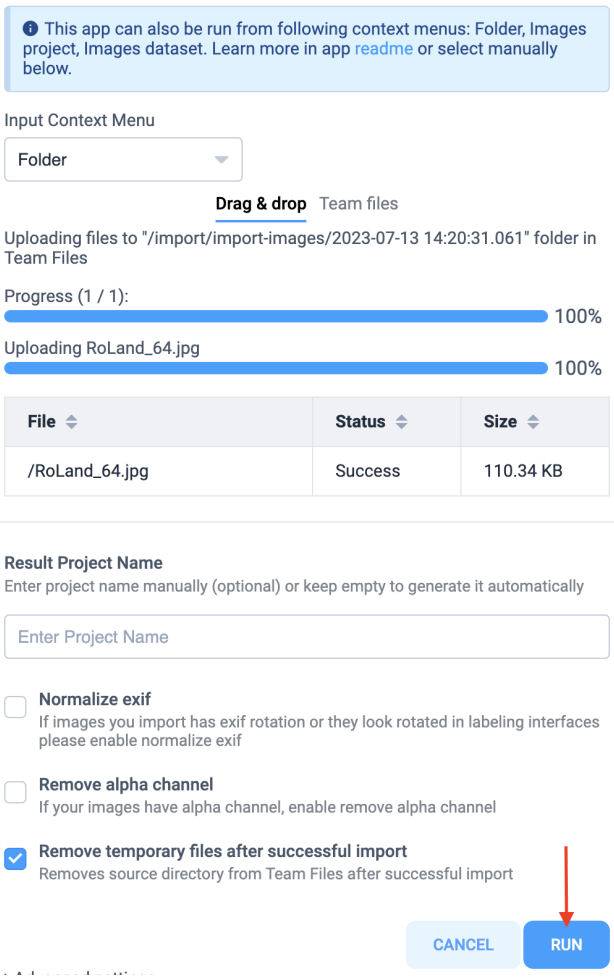
First, create an account in Supervisely. Then, to upload the images, click on the cloud icon.



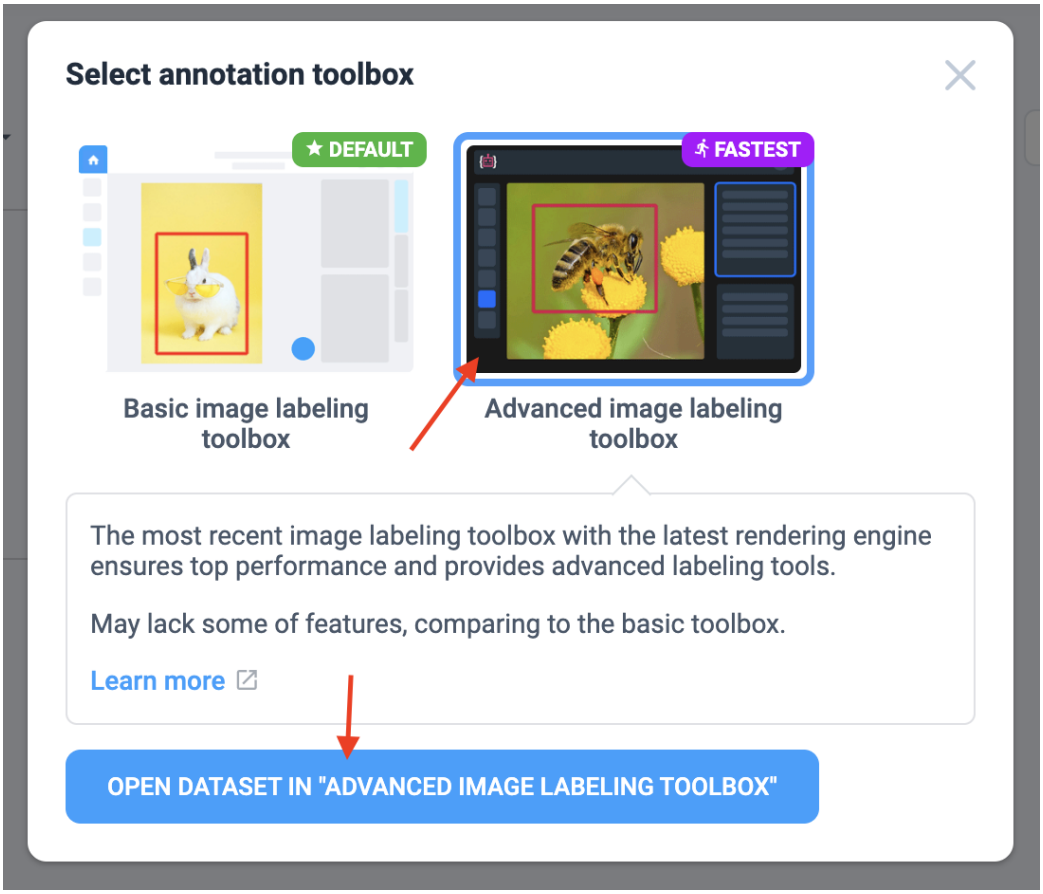
Choose **Import Images**



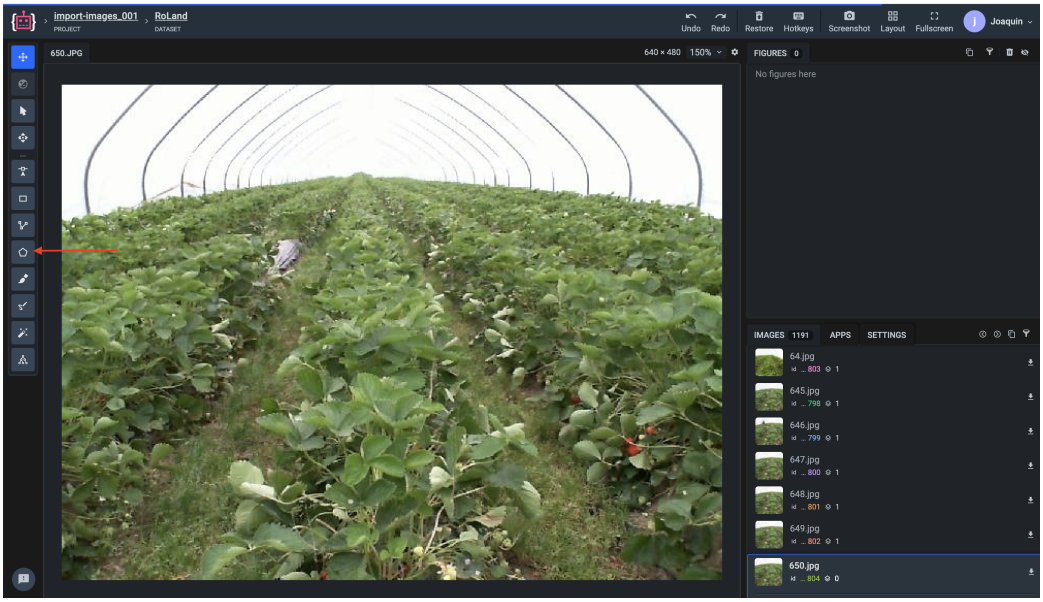
Add all the non-blurry images previously filtered and click **RUN**.

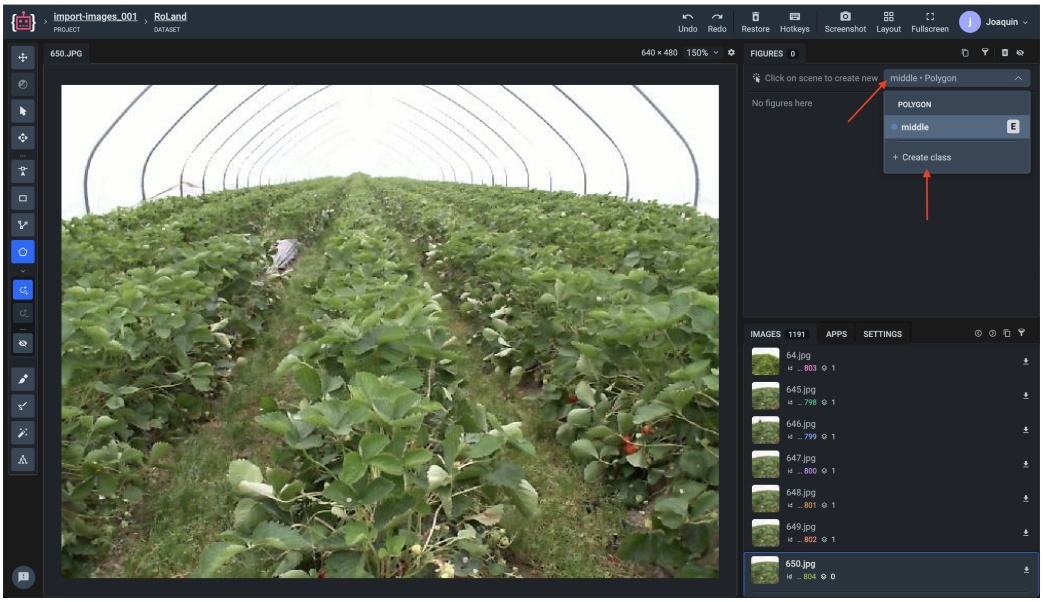


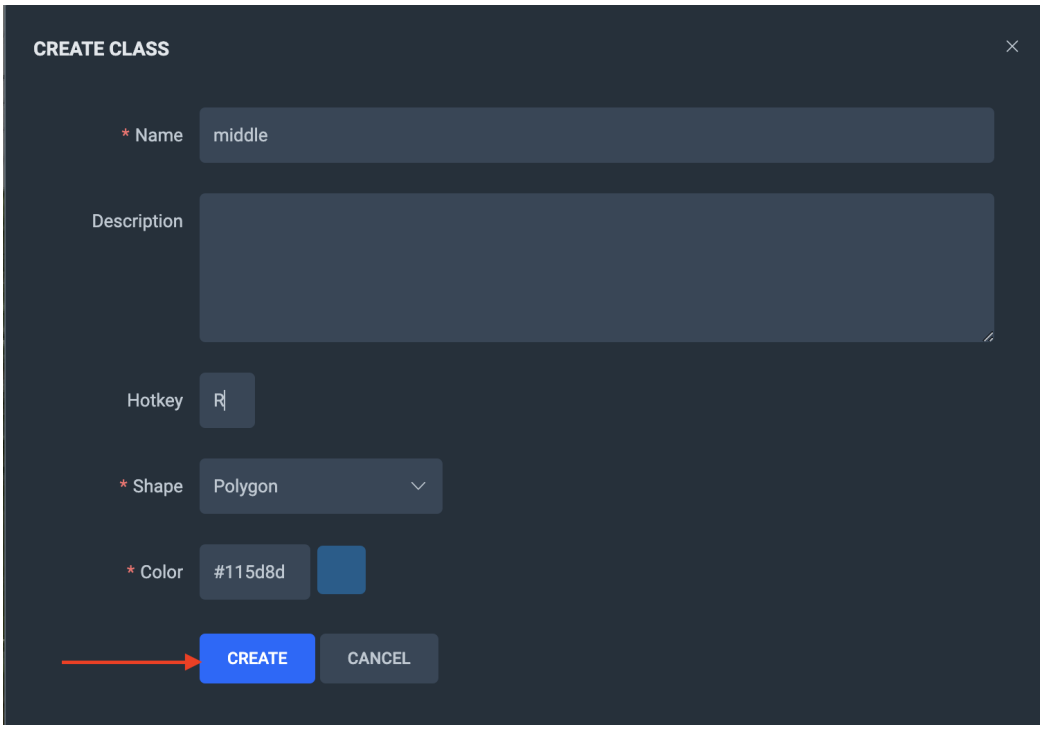
Once it finished uploading, click on the project and choose **Advanced image labeling toolbox**.



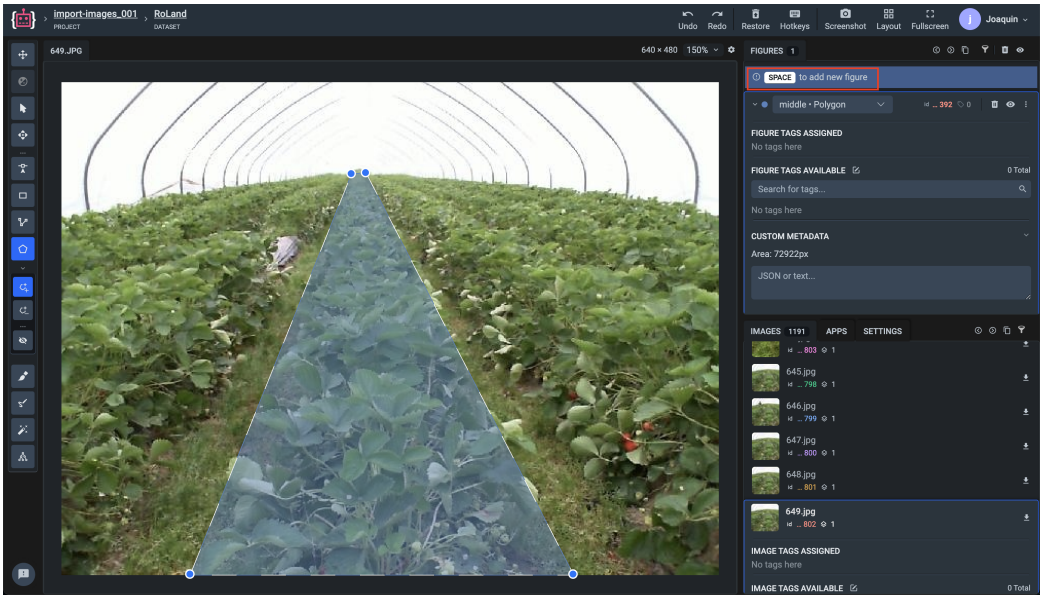
Click on the polygon option and create a class.



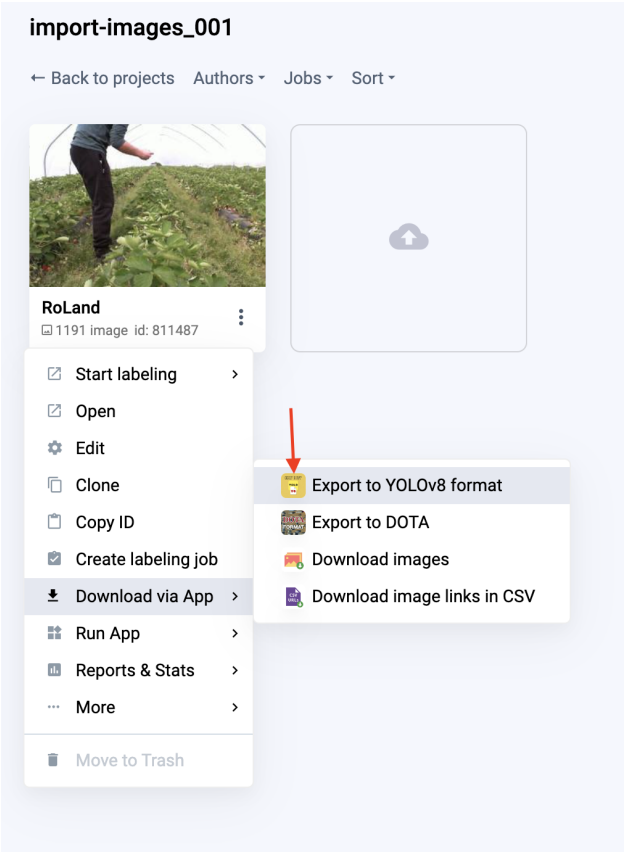




Use the mouse to create the labels around the object to label (in this case, middle lane). Press SPACE to finish the label.



Once all the images have been labeled, go back to the project page and click on **Export to YOLOv8 format**:



Download the generated file and it should contain the images, labels, and the **data_config.yaml** file.

1.3 Dataset Processing

Removing non-labeled images:

If not all the images were labeled, run the script **delete_non_labeled_imgs.py** replacing **path/to/folder** with the path to the exported folder.

Creating Datasets:

To create the training, validation and testing datasets, run the **create_datasets.py** script:

```
python create_datasets.py /path/to/exported/folder
/path/to/output/dataset/folder 90
```

With this command, 90% of the data is going to be for training, 5% for validating, and 5% for testing.

2 YOLO

YOLO (You Only Look Once) is a real-time object detection algorithm that predicts bounding boxes and class probabilities directly from images or videos. In this case, YOLOv8 was used. This is the latest version of the model for real-time object detection and image segmentation. Built upon advancements in deep learning and computer vision, YOLOv8 offers unparalleled speed and accuracy. YOLOv8 brings new features and improvements, enhancing its performance, flexibility, and efficiency.

2.1 Training

Configuration File

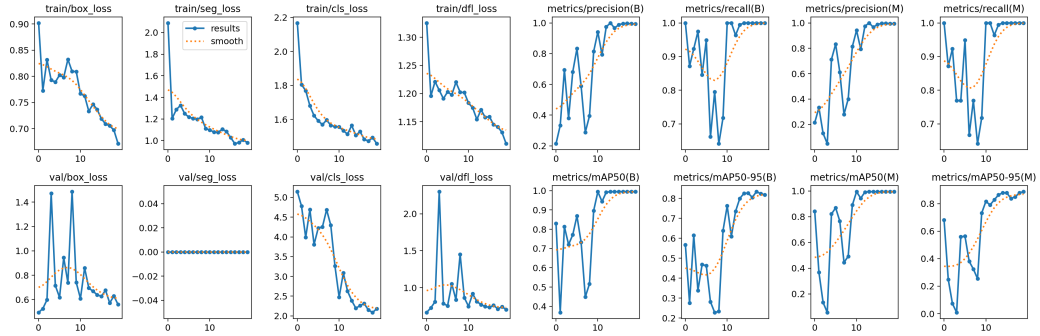
Open the **data_config.yaml** and change it to:

```
path: ../datasets
train: images/train
val: images/val
test: images/test

names:
  0: middle
```

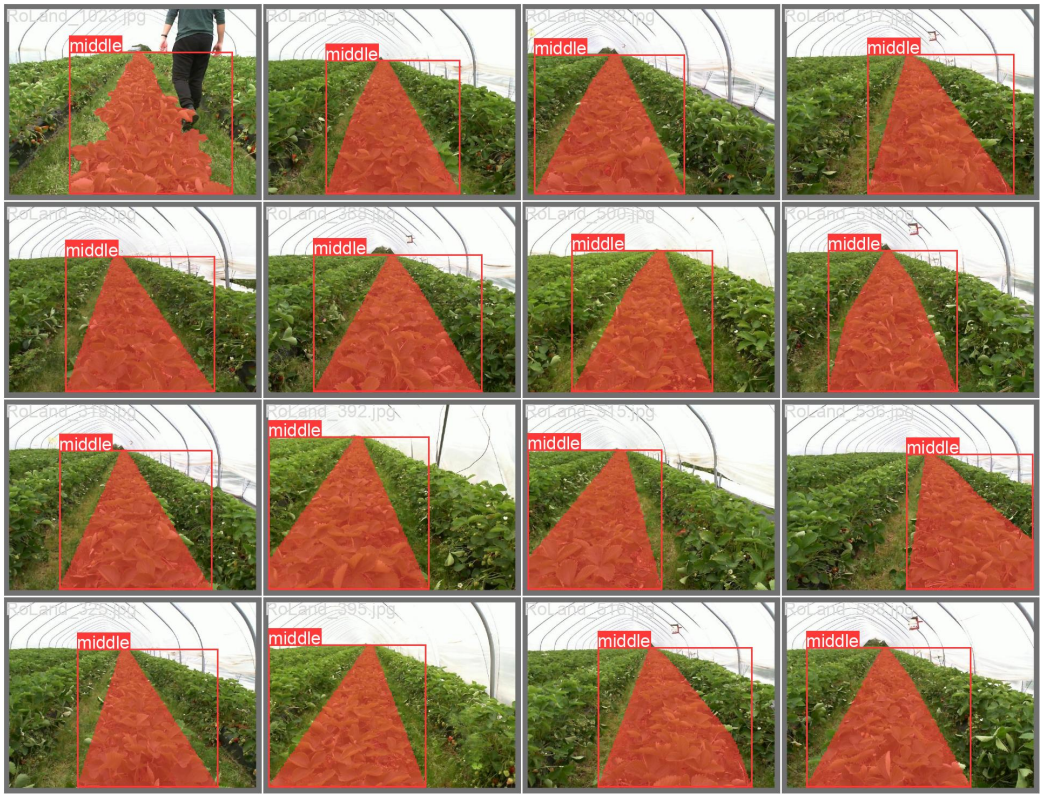
The pre-trained model **yolov8n-seg.pt** was used for training. Just replace the data path to the path of the config file. There were several training tests with different amounts of data and they can be found under training versions in GitHub. However, the best performance was the model trained with 716 images and the following hyperparameters:

```
epochs: 20
patience: 30
batch: 16
imgsz: 640
pretrained: true
lr0: 0.001
lrf: 0.001
weight_decay: 0.0005
```

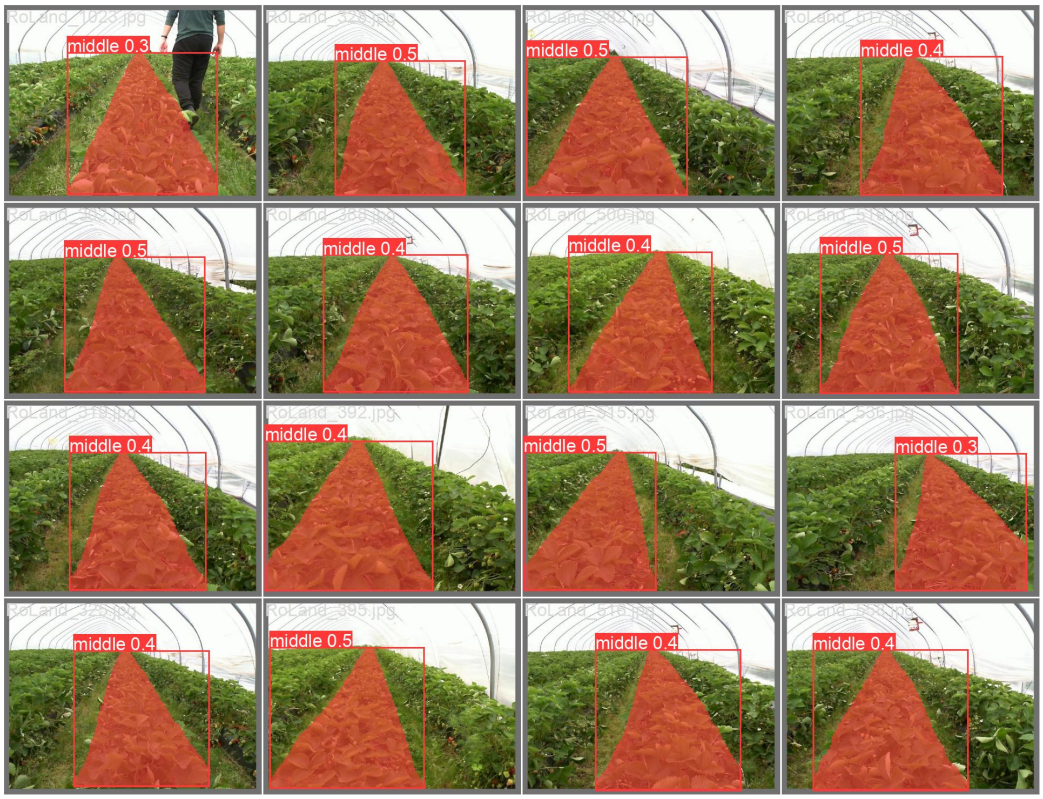
Training Results



Batch 0 labels:



Batch 0 predictions:



The best model is saved in **'runs/segment/train/weights/best.pt'**

2.2 Prediction

To predict the, use the **predict.py** file and replace the directory path with the path to the test folder from the datasets.

2.3 Validation

For validation, use the **val.py** script and load the trained model.

3 Orientation and Position Detection

Now that the lane is being detected, it is necessary to calculate its position and orientation with respect to the camera.

3.1 Lane Borders Detection

Mask

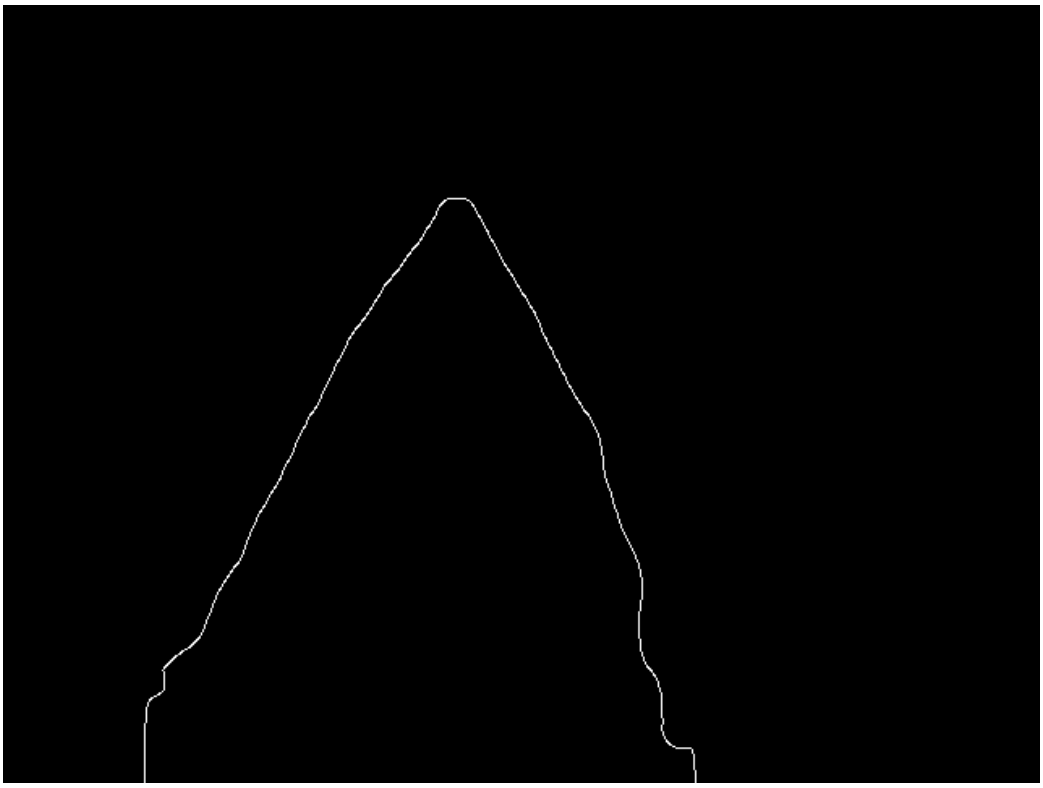
First, the detection mask has to be extracted:

```
# Predict with the model
model.predict(directory, max_det=1, boxes=False)

# Extract the lane region

# Predict on an image
results = model(directory)
# Get the masks object from the results
masks = results[0].masks
 # Get the mask corresponding to the first detected object (lane)
mask = masks.data[0].numpy()

# Create a binary image from the mask
binary_mask = (mask > 0).astype(np.uint8) * 255
```

Canny Edge Detection

Now the Canny edge detector is applied to the mask:

```
# Apply Canny edge detection
edges = cv2.Canny(binary_mask, 50, 150)
```



To detect the lines, the Hough Transform is applied:

```
# Apply Hough transform to detect lines
 lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 50, None, 50, 10)
```

And assign each detected line to either left or right boundaries:

```
# Filter and separate lines as left and right lane boundaries
left_lines = []
right_lines = []

if lines is not None:
    for line in lines:
        x1, y1, x2, y2 = line[0]
        slope = (y2 - y1) / (x2 - x1)
        if slope < -0.5:
            left_lines.append(line[0])
```

```
        elif slope > 0.5:
            right_lines.append(line[0])

# Fit lines to left and right lane boundaries if lines
    were detected
left_lane = np.mean(left_lines, axis=0, dtype=np.int32)
            if left_lines else None
right_lane = np.mean(right_lines, axis=0, dtype=np.int32)
            if right_lines else None
```

Then, the lines are extended and set to begin from the bottom of the image so the result looks like:

3.2 Middle of the Lane Determination

After the borders have been detected, the middle of the lane is calculated:

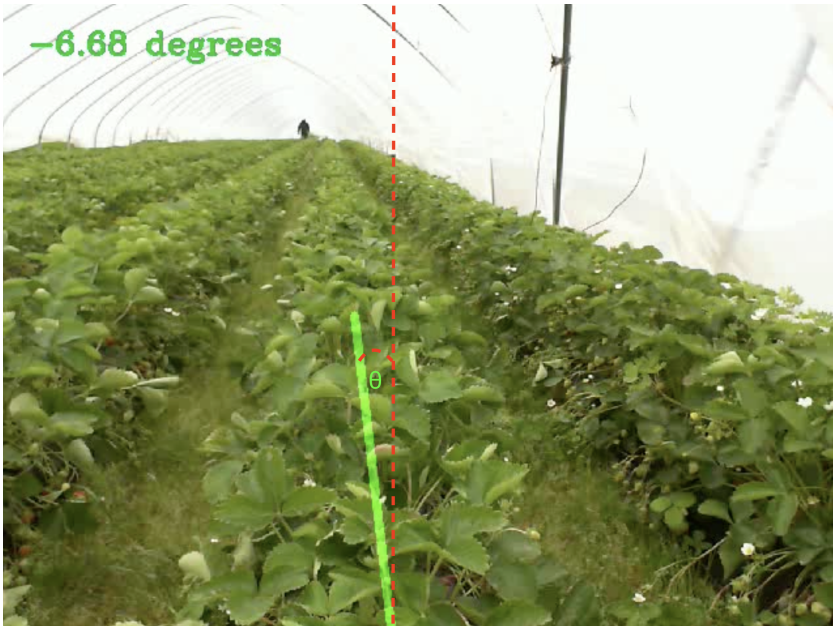
```
middle_x_bottom = int((left_lane_extended[0] +
                  right_lane_extended[0]) / 2)
middle_x_top = int((left_lane_extended[2] +
```

```
                    right_lane_extended[2]) / 2)
middle_y_bottom = y1
middle_y_top = y2
middle_slope = (middle_y_top - middle_y_bottom) /
                    (middle_x_top - middle_x_bottom)
```

3.3 Rotation

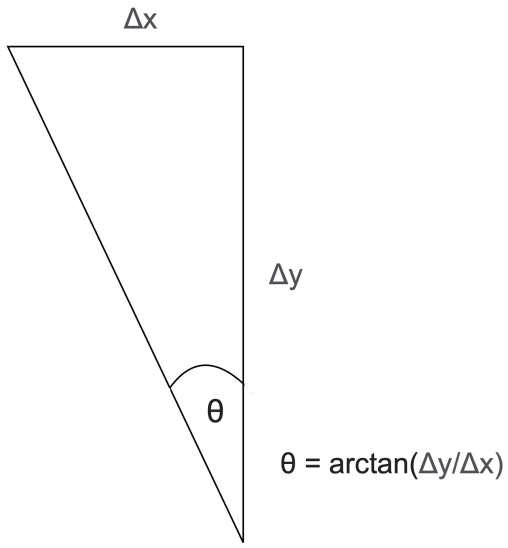
Description

In order to determine the rotation of the robot with respect to the lane, the angle between the lane and the base of the image was calculated. If the angle is more than 10° off from the $y - axis$, then the text showing the angle is going to turn red.



Angle Determination

The angle is calculated using trigonometry:




```
delta_x = middle_x_top - middle_x_bottom
delta_y = middle_y_top - middle_y_bottom
angle_rad = np.arctan2(delta_y, delta_x)
angle_deg = np.degrees(angle_rad)
```

3.4 Threshold

Description

The threshold is the region in which the middle of the lane can be before triggering a warning for lane correction. It stays static during the whole process. The threshold can be set automatically from the first frame in which the lane is detected or it can be manually set.

Automatically Set

```
threshold_from_frame = True
threshold = 50
```

When this variable is set to **True**, the line detection is performed until both lanes are detected. Then the threshold is set by translating the border lines along the x-axis by the threshold value:

```
l_threshold_x_bottom = left_lane[0] + threshold
l_threshold_x_top = left_lane[2] + threshold

r_threshold_x_bottom = right_lane[0] - threshold
r_threshold_x_top = right_lane[2] - threshold
```

Manually Set

```
threshold_from_frame = False
threshold = 50
threshold_x = 300
```

To manually set the threshold, set **threshold_from_frame** to **False**. The bottom of the threshold lines start at $x =$ **threshold_x**. Then they

are translated by the **threshold** value (to left and right respectively).
The top x values of the lines are set by using the slope formula: $y - y_1 = m(x - x_1)$ where x is the top of the line, so $x = x_1 + (y - y_1)/m$. Note that $m = tan(\theta)$, where θ is the angle between the $x - axis$ and the line.

```
angle_from_yaxis = 3
angle = np.radians(90 + angle_from_yaxis)
y1 = height - 1
y2 = int(height * 0.5)

l_threshold_x_bottom = threshold_x - threshold
l_threshold_x_top = l_threshold_x_bottom + (y2 - y1) / np.tan(angle)

r_threshold_x_bottom = threshold_x + threshold
r_threshold_x_top = r_threshold_x_bottom + (y2 - y1) / np.tan(-angle)
```

Therefore, with the variable **angle** it is possible to set the inclination of the threshold lines.

3.5 Displacement

Description

The displacement of the robot across the lane is also important, as it helps to determine if the robot is already off track. This cannot be always determined by the rotation, since the lane can still be parallel to the robot even if the robot is off the lane. The point of reference to measure de displacement can be either the center of the $x - axis$ of the image or it can be set to be the middle of the threshold. This reference is represented as a small black line at bottom of the video.

Center of the Image

```
center_of_cam = True
```

When this parameter is set to **True**, the center of the image is taken by dividing the frame width by 2.

```
cam_width = int(video_capture.get(cv2.CAP_PROP_FRAME_WIDTH))
cam_center_x = cam_width // 2
```

Center of Threshold

`center_of_cam = False`

After setting this parameter to **False**, the **cam_center_x** is set to the x coordinate of the bottom middle line (**middle_x_bottom**), or to **threshold_x**, depending on whether the threshold is set automatically or manually.

Displacement Determination

To calculate the displacement of the robot from the center of the lane, the distance along the $x - axis$ from the bottom of the center of the lane to the **cam_center_x** is calculated:

`distance = middle_x_bottom − cam_center_x`

Final Code

The code can be tested by running **lane_detection.py**.