In [1]:

```python
import csv
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
```

In [2]:

```python
def readImageData(rootpath):
    '''Reads data
    Arguments: path to the image, for example './Training'
    Returns:   list of images, list of corresponding outputs'''
    images = [] # images
    output_1 = [] # corresponding x index
    output_2 = [] # corresponding y index
    output_3 = [] # corresponding x width
    output_4 = [] # corresponding y width

    prefix = rootpath + '/'
    gtFile = open(prefix + 'myData'+ '.csv') # annotations file
    gtReader = csv.reader(gtFile, delimiter=';') # csv parser for annotations fi
    next(gtReader)
    # loop over all images in current annotations file
    for row in gtReader:
        img=Image.open(prefix + row[0])  # the 1th column is the filename
        # preprocesing image, here we resize the image into a smaller one
        img=img.resize((128,128), Image.BICUBIC)
        img=np.array(img)
        images.append(img)
        output_1.append(float(row[1])) # the 8th column is the label
        output_2.append(float(row[2]))
        output_3.append(float(row[3]))
        output_4.append(float(row[4]))

    gtFile.close()
    return images, output_1, output_2, output_3, output_4
```

In [3]:

```python
def display_images(images, rows, cols, titles=None):
    """
    Display a group of images in a grid.
    Arguments:
        images: List of images (each as a NumPy array).
        rows: Number of rows in the grid.
        cols: Number of columns in the grid.
        titles: Optional list of titles for each image.
    """
    fig, axes = plt.subplots(rows, cols, figsize=(15, 15))
    axes = axes.flatten()  # Flatten the grid for easy iteration
    for i, ax in enumerate(axes):
        if i < len(images):
            ax.imshow(images[i], cmap='gray')  # Display the image
            ax.axis('off')  # Hide axes
            if titles:
                ax.set_title(titles[i], fontsize=10)
        else:
            ax.axis('off')  # Hide extra axes
    plt.tight_layout()
    plt.show()
```

In [4]:

```
# Function to calculate the error between predicted and ground truth values
def calculate_error(pred, gt):
    # Calculate the Euclidean distance between predicted and ground truth (x, y,
    pred_x, pred_y, pred_xw, pred_yw = pred
    gt_x, gt_y, gt_xw, gt_yw = gt

    # Calculate absolute errors for each dimension
    error_x = abs(pred_x - gt_x)
    error_y = abs(pred_y - gt_y)
    error_xw = abs(pred_xw - gt_xw)
    error_yw = abs(pred_yw - gt_yw)

    # Sum of absolute errors (you can also use squared error or Euclidean distan
    total_error = error_x + error_y + error_xw + error_yw
    return total_error
```

In [5]:

```
def draw_rectangle(ax, center_x, center_y, x_width, y_width, color, scale_factor
    """
    Draw a rectangle on the image.
    Arguments:
    - ax: Matplotlib axis object to draw on
    - center_x, center_y: Center coordinates of the rectangle
    - x_width, y_width: Width and height of the rectangle
    - color: Rectangle border color
    - scale_factor: Scaling factor for resizing coordinates
    """
    # Scale the coordinates and dimensions
    center_x_scaled = center_x * scale_factor_x
    center_y_scaled = center_y * scale_factor_y
    x_width_scaled = x_width * scale_factor_x
    y_width_scaled = y_width * scale_factor_y

    # Calculate the top-left corner of the rectangle
    top_left_x = center_x_scaled - x_width_scaled / 2
    top_left_y = center_y_scaled - y_width_scaled / 2

    # Create a rectangle patch
    rectangle = patches.Rectangle(
        (top_left_x, top_left_y),  # Top-left corner
        x_width_scaled,           # Width
        y_width_scaled,           # Height
        linewidth=2, edgecolor=color, facecolor='none'
    )
    ax.add_patch(rectangle)
```