Lab04 - YOLO

MTRN4231 - UNSW School of Mechanical and Manufacturing Engineering

Preliminaries

Introduction

This lab covers YOLO and rosbags. YOLO is a powerful machine-learning architecture for item classification and segmentation. Once an item is detected from a camera feed depth data could be used to estimate its pose. Following this lab you should have all the tools required to start developing closed loop arm applications.

Task 1 - Run the camera driver

Note that this task has already been done for you but it's important to understand that a ROS2 camera package is required to subscribe to the camera feed.

```
source install/setup.bash
ros2 run v4l2_camera v4l2_camera_node
```

You should see a new topic called `/image_raw`.

Task 2 - Debugging

`rqt` can be used for debugging your ROS2 environment. Run the following command:

`rqt`

You can display images by going to `Plugins > Visualisation > Image View` and then changing the topic to subscribe.

Task 3 - YOLO

YOLO (You only look once) is a machine-learning image detection, classification, and segmentation model for computer vision. In its simplest form, the model receives an image and an input and classifies items by drawing a bounding box around them. While satisfactory for item tracking, when interacting with smaller items it is better to segment them and receive a binary mask of the pixels containing the item. Luckily YOLO has this capability all be it at a processing cost. The example code demonstrates both segmentation and bounding box classification, when working on your project you could use either.

First download YOLO if you don't have it, and make sure it is upgraded to the latest version.

```
pip install ultralytics
pip install --upgrade ultralytics
```

An example YOLO implementation has been provided in the starter repository. First resolve dependencies, build and source your workspace. Then run the command:

```
ros2 run lab4_perception_example perception_example
```

Task 4 - Extracting data from YOLO

Pick a random object that the default model can detect and which you have easy access to, ie: a cup, chair, a phone. Modify the provided code to publish a `pose` with the centroid of the segmented area to the topic `item_location`. Use CLI or RQT to verify your implementation.

Task 5 - Rosbag

Rosbags allow data capture from sensors for later use. The rosbag function listens to specific topics and saves them to a '.db3' file, allowing replay on the same topic.

First, list the topics available from your nodes:

```
ros2 topic list
```

To capture a rosbag, run the following command and specify the topics to record:

```
ros2 bag record -o <rosbag_name> <topic1> <topic2> <topic...>
```

The rosbag folder can be moved to different devices or shared, and played using:

```
ros2 bag play <rosbag_name>
```

Task: Record and play a rosbag of the raw image stream. Visualise this using RViz or rqt.

Task 6 - Apply Yolo on Rosbag

We can use the recorded image stream to test image recognition without a live camera feed.

Task: Run the Yolo node to infer the rosbag data.

Task 7 - (Extension) Item location

This task is an extension activity to bridge the gap between YOLO identification, and the perception node in the individual assignment. Now that an item can be detected and located in a 2D image, It is important to find its relative location in 3D space. Luckily, this can be done through the use of a depth camera. Using the provided sample code from lab3 look up the depth of the centroid and publish a transformation frame extending from the depth camera to the estimated item location. For this, you may choose to subscribe to the depth camera image feed rather than the v4l2_camera feed.

Task 8 - (Extension) Training Bounding Box Model

NOTE: This extension task walks you through how to train and classify a bounding box YOLO model.

The default training model may not be robust enough for your use case of object detection.

1. Capture images from the camera using `cheese`:
   - Capture a variety of lighting conditions and angles.
   - Move the images into `my_dataset/train/images`.
2. Annotate using a labelling tool: MakeSense.ai in object detection mode. Import your images into the website and set up your labels by importing. Add more classes if it fits your project, but don't renumber existing classes if using the pre-trained model.

   When done, export annotations in YOLO format and move them to `my_dataset/train/labels`.
3. Set up YAML config file:

```
path: <path_to_your_dataset>
train: train
val: val

names:
0: blue_cone
1: large_orange_cone
2: orange_cone
```

```
3: yellow_cone
4: unknown_cone
```

4. Train the model using CLI:

   ```
   yolo task=detect mode=train model=<source_model> imgsz=640 data=<dataset_yaml> epochs=10 batch=1
   ```

   Understand what these flags do:

   - `task`
   - `mode`
   - `model`
   - `imgsz`
   - `data`
   - `epochs`
   - `batch`
   - `name`

5. Load and test your model. The final model is located in:

   ```
   cd runs/detect/<name>/weights
   ```

   Move this model into your ROS package and configure it to use the new weights.

Resources

YOLO Documentation