Research Report

Problem overview

The task requires an application that can do object detection on a number of objects and from this determine some interactions between them. Given the time and resources available, the initial plan was to use a combination of classical and learnt detection methods, something to monitor the "world state" and keep track of objects even when detection fails, and some kind of heuristic to determine interactions. Ultimately, a fully learnt detector was used, but the initial classical methods explored are discussed here and the code is still present in the submission.

Initial Classical Methods

Initially classical computer vision methods were applied to the task of object detection.

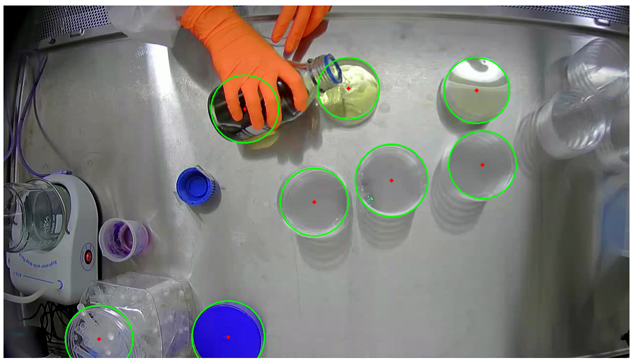
Colour Segmentation

The hands are the only orange thing in the frame, so these can be reasonably detected with colour space thresholding. Converting the image to HSV and then thresholding on min and max orange HSV values produces a mask, from which hands can be detected by finding the largest contiguous regions.

This is a comparatively cheap method although it does require making a whole image copy. Execution time is still in the order of a few milliseconds. However, this method cannot discern between left and right hands and is not robust to certain types of occlusion resulting in non-contiguous regions - i.e. holding the bottle palm up separates the wrist from the fingertips.

Hough Circles

Given that the petri dishes are a fairly fixed size and are circular, the circle Hough Transform can be used to identify them in the image. From a region of interest pixel sample about the centre of the circles, the petri dish state can be classified as well as filtering out false positives based on the colour. Low mean saturation corresponds to an empty dish whereas yellow hues of different saturations and intensities can be used to classify dishes that are being filled or are filled with a lid on. The main advantage of this method is that it's comparatively computationally cheap provided that the search space is appropriately constrained and it is reasonably robust to partial occlusion. Execution time on a M1 Macbook Pro is approximately 10ms.



Despite this, this method was still not as robust as initially hoped - colour space filtering is not robust to different positions in the hood and objects such as the lip of the plastic jar in the bottom right of the frame were often classified as an empty petri dish.

YOLO detector

Initially the plan was to use a pre-trined YOLO model to detect the bottle as this class is part of the MS COCO dataset. Given that the bottle changes appearance, becoming emptier throughout the video sequence, using a learned approach for detection seemed more likely to give better results over something like SIFT feature tracking.
However, given the failure modes of the classical methods for the other objects, it seemed prudent to try fine-tuning a YOLO model that could detect everything.

200 images were extracted from the sample video and labelled using Label Studio[1] with the labels:
1. bottle
2. bottle cap
3. petri dish empty
4. petri dish filled
5. petri dish filled with lid
6. left hand,
7. right hand

Then using the Ultralytics package, a YOLOv11[2] nano model was fine tuned with these new custom classes in a Google Colab notebook[3] for 60 epochs. Given the number of classes and constrained environment, the smallest "nano" model was sufficient to achieve accurate detection while keeping execution time low, achieving real time speeds on a CPU.

This model, while it works well on the sample video, is very likely overfit to this scene. It would probably perform worse on a completely new video. Furthermore, the detection isn't perfect and will occasionally misclassify an object, particularly if the frame is corrupt, so it's possible a larger model is more appropriate in future work.

Object Tracking

Object detection algorithms are imperfect and so a system is required that keeps track of objects that momentarily fail to be detected and that rejects spurious detections. The system implemented here does this with observation count thresholding and simple distance thresholding on the detected object centroids in order to associate current detections with previous ones. A more sophisticated system would use something like a Kalman filter to handle observation uncertainty and handle motion estimation, but for the current task the simple thresholding method seems to work well enough albeit with some shortcomings.

A particular shortcoming of the current object tracking is that the system has no understanding that "petri dish empty", "petri dish filled" and "petri dish filled with lid" are different states of the same object. As a result the system will consider the scenario where a hand is holding a petri dish as being filled as the hand no longer holding an empty petri dish, followed by the and holding a new filled petri dish. This also results in the system over-counting the number of filled petri dishes. This could be mitigated by implementing state machine behaviour for the petri dish.

[1] https://labelstud.io/

[2] https://docs.ultralytics.com/models/yolo11/

[3] https://colab.research.google.com/drive/1coXG7L99l5mkGVLame3yv7-ncjFHYRH8?usp=sharing

Interaction Heuristics

Events and interactions in the scene can be detected from heuristics based on the object detections.

Simple intersection area checks on detected object bounding boxes can be used to infer whether a hand is touching an object or whether a bottle has its cap removed or not. Two objects are said to be interacting if the intersection area of the bounding boxes is above a threshold.

Counting new instances of the “petri dish filled” class that aren’t associated with previous detections, can be used to keep track of how many petri dishes have been filled. This gets a number that’s in the right ballpark but has a tendency to over-count, either due to detection inaccuracies, or failure to account for more complex behaviour such as topping up an existing dish with more liquid.
A more sophisticated heuristic that leverages state-machine behaviour and takes in to account the bottle position (i.e. to check for pouring) could make this more robust. That said, it may be worth exploring a learned action recognition solution to help account for the more complex behaviour.

Further work

Detection
YOLO v11 has an AGPL license and it would be preferable not to have to pay for an enterprise license. The Ultralytics package was used here for its ease of use in getting a proof-of-concept model trained in a short time period. There exist other implementations online of older YOLO versions with more permissive licenses that would likely work well enough for this application[4,5]. Depending on future requirements, it might be prudent to produce our own network, whether that be a YOLO network or something DETR-like[6]

The network trained for this task is likely overfit to this specific video sequence and so further work would include creating a larger more diverse dataset from other videos taken from a lab setting. The Ultralytics train script does some image augmentation as part of its training procedure, though whether more sophisticated and varied augmentations could be applied ought to be investigated.

State Machine behaviour
As noted previously, the system has no understanding that “petri dish empty” and “petri dish filled” are two states of the same object and instead considers them as separate entities. Scene understanding would be improved with the implementation of some state machine behaviour for petri dishes and potentially other types of objects.

Hand pose and grasp classification
Simple intersections were used to determine if hands were holding or touching certain objects. This method is flawed in that it will consider hands moving over the top of objects the same as touching them. A network that can do hand pose estimation or can classify hand poses in grasping positions would make this functionality more accurate.

Action Recognition
Heuristic methods have their obvious shortcomings for detecting interactions and doing action recognition and as such, a learned a approach should be explored. A good place to start would be looking at winners of the EPIC Kitchens Action Recognition competition as this is a particularly

[4] https://github.com/Lornatang/YOLOv4-PyTorch

[5] https://github.com/MultimediaTechLab/YOLO

[6] Zhao et al. “DETRs Beat YOLOs on Real-time Object Detection”, 2024 IEEE/CVF Conference on Computer Vision, 2024

challenging dataset. Both AVION[7] and OpenTAD[8] look promising as network architectures and training schema to start from. However dataset curation for such tasks is difficult and time consuming and training can be expensive (although AVION claims SotA despite training on consumer hardware). More research needs to be done in to what would be needed to train an action recognition model to determine whether it's worthwhile over attempting to craft more sophisticated heuristics.

[7] Yue Zhao & Philipp Krähenbühl, "Training a Large Video Model on a Single Machine in a Day", CVPR 2023 https://github.com/zhaoyue-zephyrus/AVION

[8] Liu S et al. "Harnessing Temporal Causality for Advanced Temporal Action Detection", CVPR 2024 https://github.com/sming256/OpenTAD/