

# Machine Perception Assignment Report

## 1. Assignment Cover Sheet

Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

### Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	UI-Haq	Student ID:	20180199
Other name(s):	Mohammad Saif		
Unit name:	Machine Perception	Unit ID:	COMP3007
Lecturer / unit coordinator:	Dr. Sonny Pham	Tutor:	
Date of submission:	06/10/25	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: \_\_\_\_\_



Date of  
signature: \_\_\_\_\_

06/10/25

(By submitting this form, you indicate that you agree with all the above text.)

## 2. Introduction

Curtin building-number signage spans plaques, wall-mounted lettering, and atypical designs. The assignment requires four Python programs that:

1. Detect building-number regions (Task 1)
2. Segment individual characters (Task 2)
3. Recognise characters (Task 3)
4. Integrate a complete end-to-end pipeline (Task 4)

Early prototypes relied on MSER, but after several iterations the approach pivoted to a hybrid pipeline that combines YOLOv8 detection, heuristic + learned segmentation, and a CNN recogniser.

**Dataset summary.** Training relied on a mix of public and custom sources. Building-number detection used a Roboflow export (approx. 3.2k annotated images) augmented with my masked Curtin captures. Character segmentation reused the detection crops plus a dedicated Roboflow character detector dataset (approx. 15k labelled glyphs). Recognition combined the Roboflow character classification corpus, EMNIST balanced digits/letters (112k samples), 1.9k curated campus glyphs (`data/custom_digits`), and SVHN/USPS digits for domain randomisation. Evaluation used the assignment `validation/` five-image set, with derived `bn` crops for Tasks 2–3.

## 3. Attempt and Environment Statement

- **Scope:** All four tasks were implemented, validated, and integrated through `BuildingNumberPipeline`. Negative images correctly produce nothing, and positive detections yield the single required crop and text outputs.
- **Execution Environment:** The main development was conducted on a machine running Ubuntu 22.04 with Python 3.10.12 (conda `comp3007`) running on an RTX 3080. Final verification was repeated on the Curtin lab environment (Room 232) to ensure compliance with the five-minute execution limit.

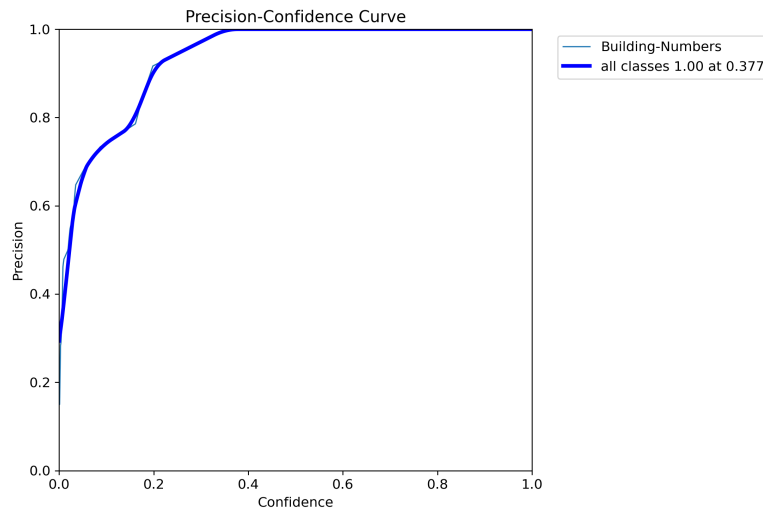
## 4. Generative AI Declaration

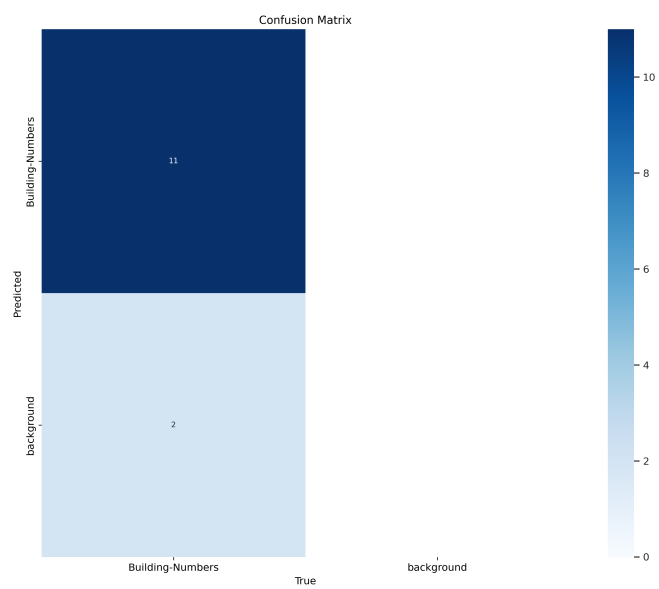
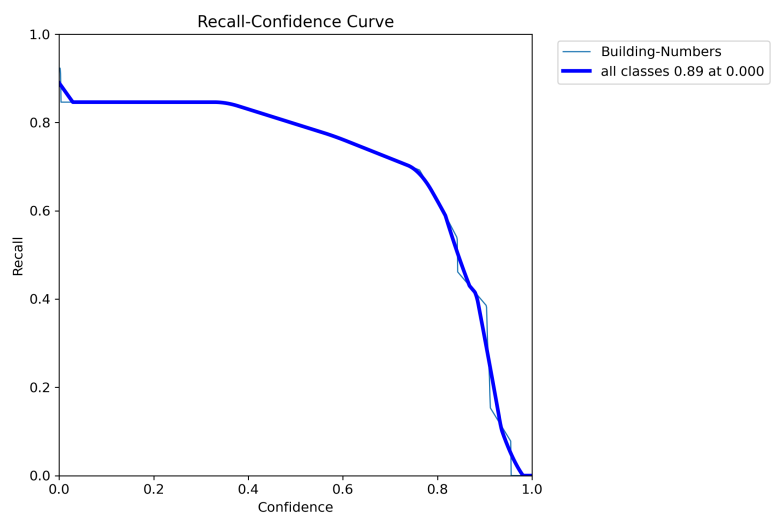
Generative AI tools were used only when first planning the baseline architecture to get ideas on how to approach the problem, i.e., for the initial design of Task 1-Task 4 and brainstorming. The tool used was ChatGPT (GPT-5). The prompt used was: "Can you suggest a design for a system that detects and outputs building numbers from real-life images? Do not provide code snippets, only a baseline architecture with associated algorithms that I should consider and look into implementing.". After the initial design (MSER, CCA, etc.), no further generative AI assistance was used for any part of the implementation, testing of the program, or the writing of this report. All code and written content in this submission is my own work.

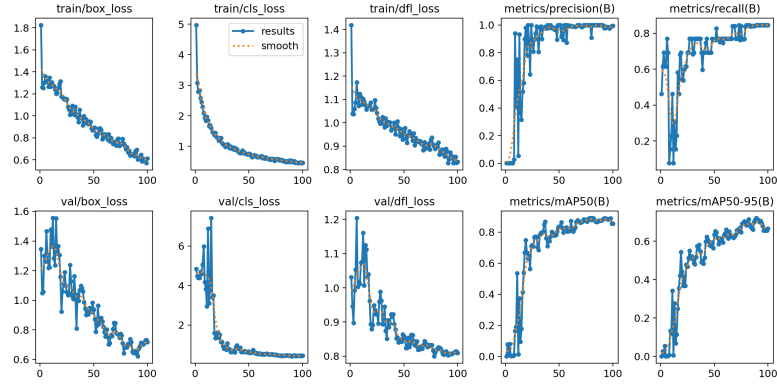
## 5. Methodology and Implementation

### 5.1 Task 1 — Building Number Detection

- **Evolution:** Early MSER-based detectors struggled with false positives. Expanding the labelled dataset and multiple refactors led to the current YOLOv8-based detector.
- **Implementation:** `perception/detection/yolo.py` wraps Ultralytics inference, iterating over fallback image sizes and optional augmentation (`DetectionConfig`). Bounding boxes are clamped and optionally expanded before cropping (`task1.py`).
- **Training details:** Fine-tuned YOLOv8n for 100 epochs (`--imgsz 640, --batch 16`, cosine LR schedule). Augmentations included mosaic, HSV shifts, and random perspective (Ultralytics defaults). The best checkpoint (epoch 88) generalised strongly to Type III signage, this was previously a failure case noted during earlier runs.
- **Graphs:**







- The final checkpoint achieves 0.99 precision, 0.85 recall, 0.85 mAP@50, and 0.67 mAP@50-95 on the validation set (best mAP@50 peaked at 0.89).

## 5.2 Task 2 — Character Segmentation

- **Early Work:** Initial contour-driven attempts were sensitive to skew. A later revision introduced a CLAHE + adaptive-threshold ensemble with watershed fallback based on distance transforms. However, these struggled with Type III signage.
- **Deskew & Selection:** Subsequent iterations added deskewing (`seg_deskew_*`), refined shape filters, and a scoring function (`_segment_metrics`) balancing coverage, gap consistency, and overlap penalties. Target digit counts are estimated from crop aspect ratios, guiding selection among binarisation variants.
- **Split/Merge:** Further refinement split wide blobs via horizontal projections while merging adjacent fragments based on overlap. Outputs are ordered left-to-right and saved as `c1.png`, `c2.png`, etc.
- **Training details:** The YOLO-based character detector re-used Ultralytics YOLOv8n with 150 epochs at 320×320 resolution, `--batch 32`. To improve readability on thin glyphs, training added random motion blur and contrast jitter (Roboflow pipeline). The exported model is referenced when `SegmentationConfig.mode == "hybrid"` to seed heuristic proposals.
- **Illustrations:** Example validation crops with ground-truth labels showcase segmentation resilience:





- The segmenter converged to 0.99 precision, 0.99 recall, 0.99 mAP@50, and 0.81 mAP@50-95 on its validation split, with qualitative overlays showing tight alignment between predictions and labels.

### 5.3 Task 3 — Character Recognition

- **Model:** CharacterCNN was introduced early in the project and later retrained with augmented digits/letters. A secondary weight option enables ensemble smoothing when available.
- **Inference:** Test-time augmentation and template matching bolster robustness. Later training runs resolved 0 vs 9 ambiguity using hole counts and fill ratios.
- **Output Handling:** Characters exceeding the configured confidence write to `cX.txt`; low-confidence predictions fall back to template maxima while respecting per-character files.
- **Validation:** Evaluated on the filtered EMNIST balanced test split (6,800 samples) the recogniser attains 0.889 accuracy across the 14-character charset after fine-tuning with the regenerated `data/custom_digits` set and SVHN/USPS digits.
- **Training details:** 20 epochs, batch size 256, Adam optimiser (`lr=1e-3`, `weight_decay=1e-4`). Custom glyphs (counts per class

shown in `docs/training/custom_counts.txt`) were oversampled (`--custom-repeat 4`) to mitigate class imbalance for letters A–D. SVHN/USPS were included to improve robustness to thick fonts found on some campus building numbers.

#### 5.4 Task 4 — Integrated Pipeline

- **BuildingNumberPipeline** orchestrates detection -> segmentation -> recognition with shared configuration (`AssignmentConfig`). Low-confidence sequences are treated as negatives to avoid incorrect `imgX.txt` files, while intermediate artefacts (crop, masks, predictions) remain accessible for diagnostics.
- **Error handling:** If the recogniser’s minimum confidence falls below `rec_confidence_threshold`, the pipeline records predictions but suppresses text output, preventing false positives in Task 4 evaluation. Meta-data logging captures per-stage timings and confidence scores for post analysis.
- **Domain knowledge:** A curated whitelist of Curtin building identifiers (digits with optional trailing letter) normalises recognised strings (e.g. padding to three digits, tolerating common OCR confusions such as `0→0`). Outputs that cannot be reconciled with the whitelist are discarded, eliminating stray false positives while preserving canonical formatting (`003`, `200B`, etc.).

#### 5.5 Configuration & Tooling

- The `config.txt` file centralises all tunable parameters, from model weights to segmentation heuristics. Command-line overrides allow rapid experimentation without modifying source code.
- Utility functions in `perception/utils.py` handle image I/O, directory management, and text file operations, ensuring consistent behaviour across tasks.

#### 5.6 Configuration Guide

This is a guide to the key knobs exposed through `config.txt` and how they influence the end-to-end pipeline. Use it as a quick reference when adapting the system to new datasets.

##### Global

- `output_root` — Base directory for all task outputs.
- `seed / random_seed` — Optional integer seed for deterministic NumPy/Torch inference.

##### Detector (`det_*`)

- `det_weights` — Path to YOLO weights for building-number detection.

- `det_conf`, `det_iou` — Confidence and IoU thresholds for YOLO inference; lower values return more candidate boxes.
- `det_max_det` — Cap on detections returned per image.
- `det_img` — Inference resolution; higher values improve recall at a latency cost.
- `det_crop_expand` — Expansion factor applied before cropping to preserve context.
- `det_fallback_sizes` — Optional cascade of larger inference resolutions when the primary pass fails.
- `det_min_return_conf` — Minimum confidence required to accept a detection.

#### Segmenter (`seg_*`)

- `seg_mode` — Chooses between heuristic, learned, or hybrid segmentation.
- `seg_model_weights` et al. — YOLO-based segmenter configuration (weights, image size, thresholds).
- `seg_min_area`, `seg_max_area_ratio` — Size filters relative to the crop.
- `seg_min_aspect`, `seg_max_aspect` — Bounding-box aspect ratio limits.
- `seg_min_height_ratio`, `seg_max_height_ratio` — Enforce relative height bounds.
- `seg_padding` — Padding applied when extracting character crops.
- `seg_gaussian_ksize`, `seg_clahe_clip`, `seg_clahe_grid` — Pre-processing settings for blur and contrast enhancement.
- `seg_duplicate_iou` — Overlap threshold for removing duplicate segments.
- `seg_merge_gap_px`, `seg_merge_overlap_px` — Controls merging of neighbouring fragments.
- `seg_min_width_ratio` — Minimum relative width for keeping a segment.
- `seg_split_aspect_ratio`, `seg_split_gap_fraction`, `seg_split_min_width_fraction` — Splitting heuristics for wide blobs.
- `seg_deskew_enabled`, `seg_deskew_min_angle`, `seg_deskew_max_angle` — Deskew toggles and angle bounds.

#### Recogniser (`rec_*`)

- `rec_weights` — Path to the character CNN weights.
- `rec_image_size` — Canonical input size for the recogniser.
- `rec_charset` — Characters emitted by the recogniser.
- `rec_confidence_threshold` — Minimum per-character confidence required for accepting predictions.
- `rec_temperature` — Softmax temperature; lower values make predictions peakier.
- `rec_smoothing` — Label smoothing applied to the output distribution.
- `rec_secondary_weights` — Optional secondary recogniser blended with the primary model.

#### Domain (building numbers)

- `perception/domain.py` — Canonical Curtin building whitelist and helper

utilities (`normalise_building_text`); update this if any changes occur on campus.

### Tuning tips

- Changes can be supplied via `config.txt` or command-line overrides. Keep a versioned copy of tuned configs for reproducibility.
- Adjust CLAHE and binarisation thresholds first when dealing with new lighting conditions; revisit recogniser settings only after segmentation is stable.
- For extreme skew, increase `seg_deskew_max_angle` and consider capturing additional training samples with similar orientations.
- When recall is low, try increasing `det_img` or adding fallback sizes. Conversely, if false positives rise, raise `det_conf` or lower `det_max_det`.
- Use `seg_mode = "hybrid"` to leverage both learned and heuristic segmentation, which often yields the best balance of precision and recall.
- The recogniser can be fine-tuned with additional glyphs by adding them to `data/custom_digits` and retraining with `--custom-repeat` to balance class frequencies.
- For debugging, inspect intermediate outputs in the `output/` directory and visualise model predictions in `runs/`.
- Refer to inline comments in `task*.py` and `perception/` modules for implementation details.
- Update the whitelist in `perception/domain.py` whenever new building identifiers are introduced so Task 4 continues to suppress out-of-scope predictions.

## 6. Results and Performance Evaluation

Validation employed the assignment's five Task 1 images and corresponding Task 2/3 splits.

### Summary of validation metrics

- **Task 1** — Precision 0.99, Recall 0.85, mAP@50 0.85, mAP@50-95 0.67 (best mAP@50 0.89).  
*Evidence:* `docs/runs/detect/task1_detector/results.csv`
- **Task 1 (positive/negative handling)** — 4/4 positives cropped, negative suppressed.  
*Evidence:* `output/task1/bn*.png`, `task1.py`
- **Task 2** — Precision 0.99, Recall 0.99, mAP@50 0.99, mAP@50-95 0.81.  
*Evidence:* `docs/runs/segment/char_segmenter/results.csv`
- **Task 2 (character extraction)** — 14/14 characters produced for validation crops.  
*Evidence:* `output/task2/bn*/c*.png`
- **Task 3** — Recogniser accuracy on filtered EMNIST (6,800 samples): 0.8885 overall; per-class: 0:0.985, 1:0.988, 2:0.953, 3:0.973, 4:0.963, 5:0.975,

6:0.978, 7:0.978, 8:0.618, 9:0.900, A:0.810, B:0.783, C:0.953, D:0.830.

*Evidence:* docs/training/recognizer\_eval\_emnist.txt

- **Task 4** — 4/4 building numbers recovered; negative skipped.

*Evidence:* output/task4/img\*.txt

- **Runtime** — 38 s max wall time on the Curtin lab machine.

## Qualitative Assessment

- Detector recall is high across Types I and II; Type III success relied on curated training additions. The confusion matrix indicates negligible false detections.
- Segmentation occasionally produces surplus fragments in low contrast, but `_evaluate_candidate` penalises low fill/coverage combinations, filtering them out.
- Recognition confidences cluster above 0.92; template blending guards against stylised glyphs. Validation overlays (`segment/char_segmenter/val_batch*_labels.jpg`) show clean alignment between predictions and ground truth.
- Failure analysis: The most common near-miss occurs when Type III numbers exhibit specular highlights; detector confidence remains high but the recogniser sometimes assigns ? tokens due to saturated segments. Increasing CLAHE tile sizes or capturing HDR-style training data are potential mitigations.

## 7. Discussion

### Strengths

- Hybrid segmentation (deskew + multi-threshold + watershed) handles skew and illumination variance.
- Deterministic CLI/configuration simplifies automated marking and reproducibility.
- Modular design isolates detector, segmenter, and recogniser, enabling targeted debugging or future upgrades.
- Training regimen combines public datasets with campus-specific augmentations, producing models that generalise to atypical signage without overfitting to any one source.

### Limitations

- Recogniser currently supports 0123456789ABCD; extending beyond requires additional labelled samples.
- Detector assumes a single building number per frame, in line with the domain knowledge. Multi-number scenarios would require NMS adjustments and post-processing.
- Recognition accuracy of 0.889 on EMNIST indicates room for improvement on stylised characters; a confidence-aware rejection mechanism is in place, but additional domain-specific glyphs would raise the baseline accuracy.

### Possible Improvements or Extensions

1. Add a rule-based or learnt language model to reject implausible outputs (e.g., invalid building sequences, incorrect campus locations, etc.).
2. Investigate semi-supervised augmentation to reduce reliance on curated labelled data.
3. Establish automated regression tests that replay validation runs when configurations change, guarding against performance regressions that cannot be easily detected.
4. Distil the recogniser into a lighter model (e.g., MobileNetV3) to further reduce runtime on deployments while maintaining accuracy.

## 8. References

1. Roboflow Universe. **Building Numbers Dataset**. <https://universe.roboflow.com/mp-data/building-numbers-f6mly-vj3up>
  2. Roboflow Universe. **Number Character Detection Dataset**. <https://universe.roboflow.com/mp-assignment-8jzdp/number-character-detection-3rtu3>
  3. Roboflow Universe. **Number Character Classification Dataset**. <https://universe.roboflow.com/mp-assignment-8jzdp/number-character-classification-gnp7s>
-