Dominoes in the World

A unified pipeline for detection, orientation, semantic labeling, and world-map integration

Henry Zhu and Tiger Zou
15-494 Cognitive Robotics Final Project

May 1, 2026

Project Goal

- Detect flat dominoes from camera images.
- Recover each domino's world **position** and **orientation**.
- Recognize **pip counts** and assign semantic face labels.
- Maintain persistent `DominoObj` instances in the world map.
- Use mapped dominoes for visualization and obstacle-aware planning.

Design principle

Conservative robustness: prefer ? over an unstable pose or wrong face label.

System Contributions

Perception

YOLO segmentation, contour extraction, duplicate suppression, and coarse geometry.

Geometry

Divider-line detection recovers orientation under foreshortening and short-side views.

Semantics

Pip counting converts detections into labels such as 4–5 with confidence gates.

End result

A complete perception-to-world-model pipeline rather than a lab-only detector overlay.

Overall Runtime Pipeline

1. Receive a camera frame and run YOLO segmentation.
2. Map each mask back to the original image and extract contours.
3. Compute a coarse rotated rectangle and suppress duplicates.
4. Detect the divider line inside the domino patch.
5. If divider exists, set long axis perpendicular to divider; otherwise fall back to rectangle geometry.
6. Project center and axis endpoints to the ground plane.
7. Create or update a world-map `DominoObj`.
8. If divider exists, rectify the domino, split into halves, count pips, and assign a face label.
9. Render results in robot view, debug view, and 3D world map.

Main Code Structure

Lab-side files

- `lab8/domino_world_detector.py`: detector wrapper, geometry, divider logic, label-provider interface
- `lab8/domino_half_face_pipeline.py`: rectified pip counting pipeline
- `lab8/DominoWorldMap.py`: demo FSM and overlay integration
- `lab8/test_domino_world_detector.py`: geometry sanity checks

Shared framework changes

- `aim_fsm/program.py`: domino runtime options and detector setup
- `aim_fsm/worldmap.py`: `DominoObj`, candidate creation, association, lifecycle integration
- viewer updates for world-map rendering and debug display

World-Map Integration

What a `DominoObj` stores

- persistent id and lifecycle state
- world pose (x, y, θ)
- physical dimensions and obstacle semantics
- image debug metadata: contour, quadrilateral, divider, axis, mask area, confidence
- optional face label and half-count metadata

Association logic

- compare projected center distance
- compare long-axis orientation modulo 180°
- promote after repeated observations
- reuse standard visible / missing / reclaimed states

Projection and Planning Integration

Projection pipeline

- image center projected to the ground plane gives world (x, y)
- image long-axis endpoints are projected to estimate θ
- world pose becomes part of candidate insertion into the map

Planning semantics

- each domino is treated as an oriented rectangle
- promoted objects become path-planning obstacles
- main practical issue is conservative size calibration

Key insight

Geometry alone grounds the object in the world, but divider detection and pip counting are required to make that geometry stable and semantically meaningful.

Orientation Problem and Failed Attempts

Core problem

`minAreaRect` works only when the domino silhouette is visibly elongated. Under perspective, short-edge views look almost square and cause 90° flips.

Abandoned approaches

- rectangle long-axis only
- raw-image Hough detection
- constrained Hough search near the center
- adaptive position constraints in raw image space

Why they failed

All of them remained too sensitive to pips, shadows, highlights, blur, and perspective distortion in the original image.

Final Divider-Line Method

1. Extract the segmentation contour and obtain a coarse quadrilateral with `minAreaRect`.
2. Warp the domino into canonical coordinates (120×60).
3. Search divider angles over $\pm 35^\circ$ with a small step size.
4. For each candidate, project valid pixels onto the normal direction and build a 1D grayscale profile.
5. Score a true divider as a **dark center band** with lighter side bands.
6. Map the best stripe back to image coordinates.
7. Set the domino long axis perpendicular to the recovered divider.

Why Divider Detection Works

What changed

- search happens after rectification
- background pixels are removed by the warped mask
- detector looks for stripe structure, not arbitrary dark segments
- multiple candidate angles make it tolerant to imperfect warps

Behavior

Robust orientation for:

- long-side-facing views
- short-side-facing views
- strong foreshortening
- noisy pips and imperfect segmentation

Pip-Counting Goal and Early Failures

Goal

Count pips on each half, return labels like 4–5, and output ? whenever visual evidence is weak or inconsistent.

Abandoned approaches

- whole-half color blob counting
- hard top-face ROI filtering
- adaptive top-surface support mask only
- hardcoded template masks

Lesson

Pip counting in raw perspective images is too brittle; normalization must happen first.

Final Pip-Counting Pipeline

1. Rectify the full domino and split it into two halves using the detected divider.
2. Crop each half to non-white bounds and letterbox to a canonical 64×64 image.
3. Build an active mask from visible non-white structure.
4. Estimate the local domino-body background in HSV and Lab.
5. Compute pip response from saturation and Lab chroma contrast.
6. Clean the candidate mask morphologically and extract connected components.
7. Score blobs by size, shape, circularity, extent, saturation, and chroma.

Blob Filtering, Repair, and Count Selection

Repair logic

- keep soft rejects as repair candidates
- promote them only when they match strong accepted blobs in hue, chroma, saturation, and size
- use count-specific color families only as secondary tie-breakers

Count selection

Candidate sets are scored using:

- relative layout prior
- average blob quality
- color-family support
- edit penalty for adding or dropping blobs

Soft priors

Examples: 2 = diagonal pair, 3 = diagonal pair + center, 5 = corners + center, 6 = two columns by three rows with foreshortening tolerance.

Stability Gates and Special Handling for Sixes

Failure gates

- zero-conflict
- weak-blobs
- too-many
- count-jump
- ambiguous
- low-confidence

If either half fails, the whole face becomes ?.

Why sixes are hard

- far-row pips are often faint or compressed
- amber color support helps recover missing evidence
- temporal stabilization resists flicker from 6 to 4 or 5

Hard divider gate

If no divider is detected, pip counting is skipped entirely and the face label is forced to ?.

Rendering and Debug Visualization

Camera overlay

Raw observations, promoted objects, ids, quadrilaterals, long axes, divider lines, and optional labels.

Label debug view

Rectified crops, response maps, active masks, candidate masks, rejected blobs, accepted blobs, and per-half scores.

3D world map

Flat cuboids with divider lines, colored pip discs, and ? for unknown halves.

Why this mattered

Most improvements came from visually inspecting exactly which geometric cues and blob candidates the pipeline trusted.

Duplicate Suppression and Validation

Duplicate suppression

- sort valid observations by confidence
- keep the strongest observation first
- suppress later observations if mask overlap exceeds 50% of the smaller mask area

Validation modes

- synthetic geometry tests
- static compilation checks
- real snapshot inspection
- end-to-end runtime checks in the demo FSM

Remaining Risks and Assumptions

Perception risks

- heavy blur or glare can weaken divider signals
- poor segmentation can corrupt both geometry and pip counting
- image-border projections can destabilize axis endpoints

Mapping risks

- hand-tuned association thresholds may swap nearby objects
- lifecycle promotion can be slow under intermittent detections
- obstacle dimensions need real-world calibration

Work After Presentation

Temporal belief voting

- Stabilize `half_counts` with a short rolling belief window instead of trusting only the current frame.
- Replace the current special-case 6 memory in `lab8/domino_world_detector.py` with a general per-domino belief tracker.
- Store recent valid samples: center, (L, R) counts, per-half confidences, face confidence, and frame id / tick.
- Expire old samples after about 8 valid detections or roughly 1 second.

Belief resolution

- Resolve left and right halves independently with confidence-weighted categorical voting.
- Vote weight: `half_confidence * recency_weight`.
- Require a minimum winning belief and a margin over the runner-up; otherwise return unknown for that half.
- Build final outputs from belief winners: `half_counts`, `face_label`, and `face_confidence`.

Interface + validation

Keep public fields unchanged, preserve current `half_image_centers` for rendering, add debug text like `raw 4 conf .55 -> belief 6 .81`, and extend `lab8/test_domino_world_detector.py` to cover flicker, replacement, low-confidence rejection, and nearby-but-distinct domino histories.

Thank you