15-494: Cognitive Robotics

Mingjun Feng/Josh Basu

Project Overview

Our Goal: Create a VEX-AIM robot that can recognize and manipulate dominoes for a simplified domino-playing task.

- Detect dominoes on the table and estimate their positions/orientations
- Represent detected dominoes on the world map as DominoObjs
- Select a target domino and a valid placement location
- Push and align the domino accurately
- Simulate a simplified domino game by making a legal move

Our Approach

Our Approach Can be Split into 3 Distinct Categories

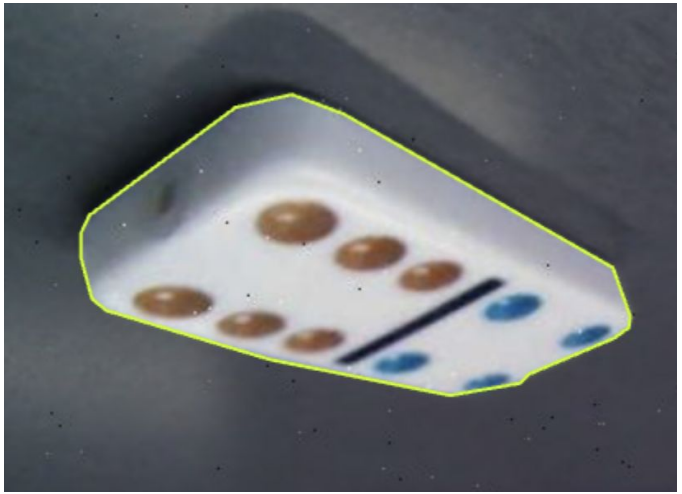
Perception

Motion

Optimization

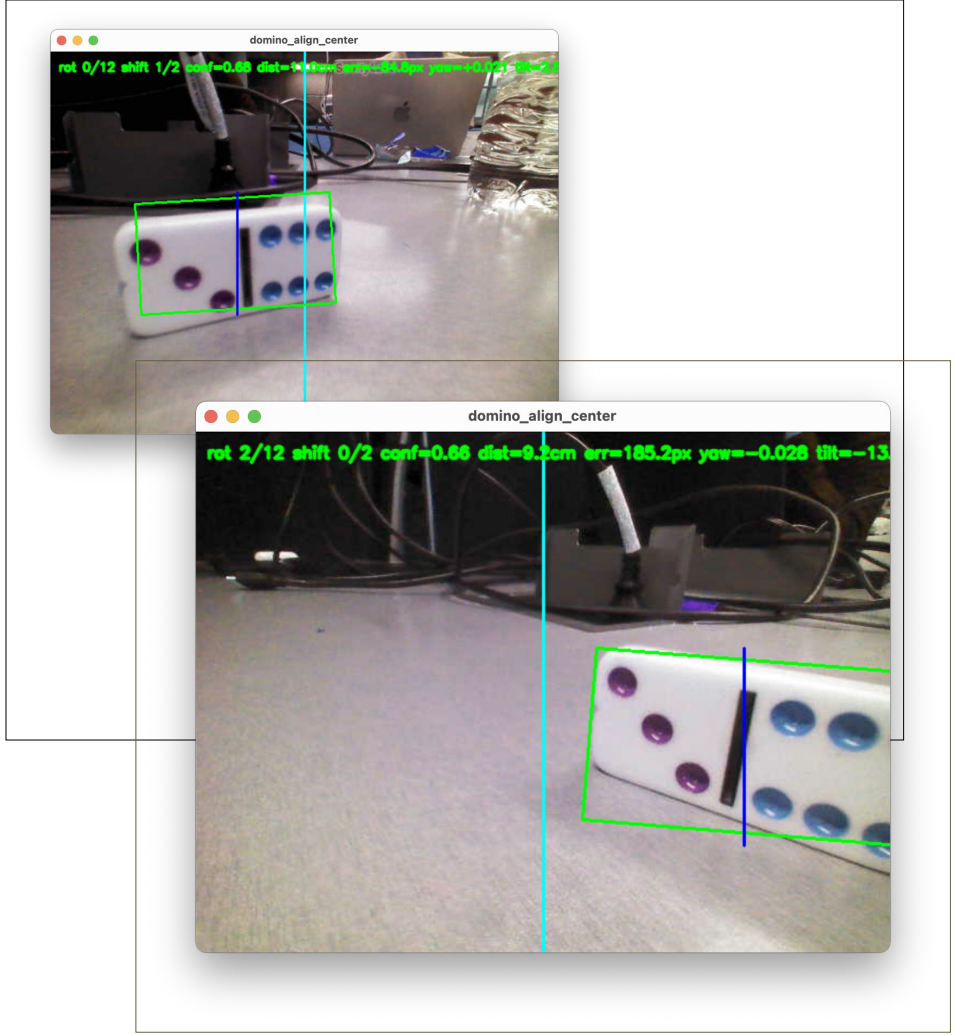
Perception

- Used Ultralytics YOLO for perception: first for standing domino detection, then trained a separate segmentation model for fallen dominoes from robot-camera snapshots (Roboflow labeling + light augmentation).



Motion

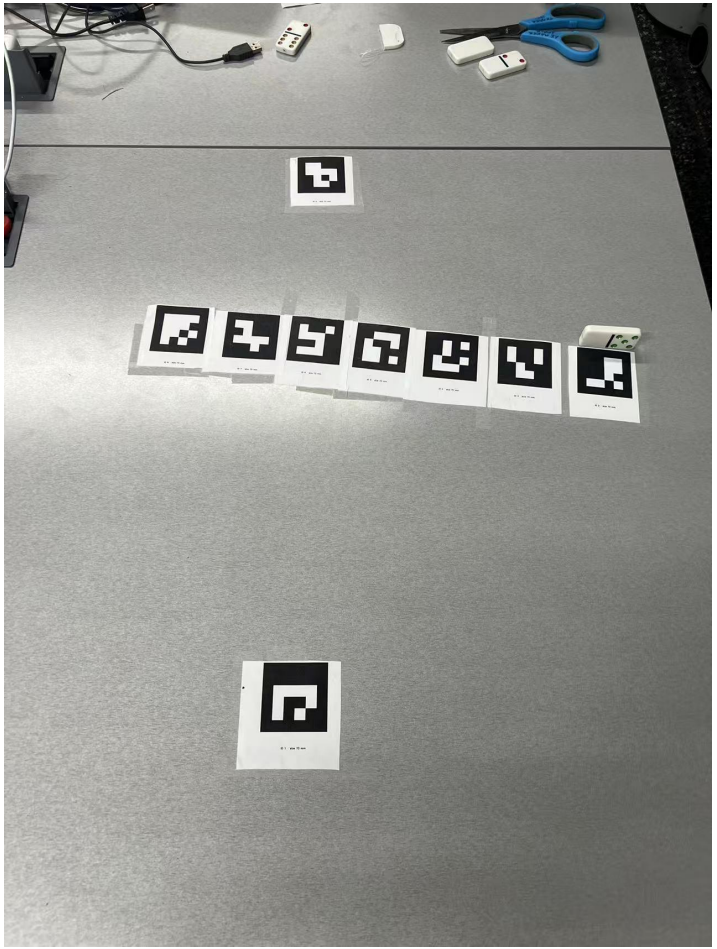
- Converted detections into **actionable geometry** with the box/mask as shown on the right
- Center + angle, and kicks off the dominos via rotating and backward motions
- Implemented motion **as closed-loop primitives:** turn to reduce heading error, sideways to reduce lateral error, then forward to reach standoff/contact, with conservative clamping and max-step safeguards.



Optimization

- Added a **marker-based “infrastructure” option** to improve accuracy: ArUco markers to define a repeatable table reference and coarse positioning when pure vision-based metric distance was inconsistent.

- Iterated by testing failure cases (occlusion, close-range crop, glare, angle), then **tuning relevant parameters** to optimize: confidence thresholds, ROIs, gains, and stop distances, keeping a working baseline while improving each stage.



What is the most interesting aspect of your solution?

- Combining learning-based perception (YOLO segmentation) with geometry-driven control: the model finds the domino; the robot uses the mask to compute a short-side contact point and physically act on it.

- The hybrid idea of using ArUco markers for repeatable coarse positioning while still relying on vision for final alignment, which reduces sensitivity to placement and distance calibration.

The results we obtained: What worked?

- YOLO reliably detected standing dominoes in the lab scene and drew accurate outlines at reasonable distances.
- A dedicated fallen-domino segmentation model could find a fallen domino and produce masks good enough to estimate pose (center/angle) and pick a short-side contact point.
- Marker checks with ArUco were repeatable at close range; markers helped create a more controlled demo setup for coarse positioning.
- Worked sometimes: Approach-and-kick/knock behaviors succeeded when alignment was good and the domino stayed in view; soft kick reduced random sliding vs hard kick.

The results we obtained: What Can Be Improved?

- Converting vision outputs into accurate millimeter distances for final contact; we often saw ~2–3 cm systematic misses due to frame/reference-point and projection errors.
- Closed-loop detection during motion could fail when turning/moving pushed the domino out of frame; open-loop plans reduced this but still accumulated motion error.
- Didn’t fully solve: Ensuring the domino falls in a perfectly predictable direction every time; small misalignment and magnet/friction effects still caused variability.

Future Work: General Ideas

- Improve metric accuracy with camera calibration + table-plane calibration (or a calibrated marker board) so pixel-to-mm estimates are trustworthy for final contact.
- Switch the final approach to visual servoing (image-space alignment) for the last few centimeters, instead of relying on absolute distance.
- Add multi-object reasoning: detect multiple fallen dominoes, track which have already been handled, and plan an order that avoids collisions.
- Add robustness features: automatic recovery behaviors (re-scan, back up, re-center), and better state estimation from wheel odometry/IMU to reduce drift.

Appendix

- Two videos provided in the .zip, One quick demo, and a second more expansive 13 minute video on ideals and how the code was implemented.
- Source code also provided in the .zip
- Instructions on how to run the Source code are present in the 13 minute video

Thank you for your time!