6.111 Final Project Checklist

Gian Delfin, Vivian Huang, Luis Terrones-Verastegui

Commitment

Maze Solving

- ❏ Skeletonize input maze (to reduce the number of possible paths we must explore); display on monitor via VGA
  - ❏ Python prototyping on binary array mazes
  - ❏ Verilog conversion
- ❏ Implement wall-following algorithm to solve maze; display solution on monitor via VGA
  - ❏ Python prototyping on binary array mazes; solution is list of (delta_x, delta_y) from start to end
  - ❏ Verilog conversion, store solution in BRAM

Goal

- ❏ Implement Shortest Path Algorithm (Lee's algorithm) on binary array mazes; solution of the same format as with wall-following

Interface to External Devices

- ❏ Construct mount for camera and projector so that they sit above the maze, facing down
- ❏ Module to take image of maze with camera
- ❏ Project path onto maze using projector
  - ❏ Module to convert solved path to actual path and store actual path in BRAM
  - ❏ Module to create pixels for showing car moving along the path

Perform Image Processing on Maze Image

- ❏ Convert RGB image to HSV
- ❏ Threshold HSV pixels to obtain walls and paths in the maze as a binary image
- ❏ Smooth binary image to remove any noise using erosion/dilation
- ❏ Skeletonize binary image to isolate the paths through the maze in as few pixels as possible. This will make the maze less computationally expensive to solve

Stretch Goal

Real time maze solving as maze changes

- ❏ Implement pruning of maze dead-ends to reduce maze solving algorithm runtime
- ❏ Use a simpler filter to reduce image processing time while maintaining/improving maze solving runtime

Project a virtual object moving through the maze

- ❏ Account for widths of object and path when skeletonizing and/or during maze solving
- ❏ Consider using diagonal moves to mirror real-world movement
    - ❏ Adjust skeletonization minimum width and/or maze-solving valid moves

Vary path weights; i.e., place colored “snacks” along certain paths such that they are worth more