6.4212 Final Project: TetrisBot

Pranav Arunandhi
MIT
apranav@mit.edu

Ashley Ke
MIT
ashleyke@mit.edu

Sadhana Lolla
MIT
sadhana@mit.edu

Abstract

In this project, we explore the granularity of pick-and-place algorithms when combined with gameplay algorithms and complex simulation machinery. To that end, we develop a robot that can play the puzzle video game Tetris from end to end. We randomly generate pieces at a specific location, develop a perception system to detect the pieces, develop an algorithm for generating legal moves given a board, rotate and place the pieces at the desired location, and clear rows once they are completed. We show that using Drake and an iiwa arm, we can successfully leverage simulation software to mimic the unrealistic aspects of Tetris, such as teleportation and line clears, in combination with more traditional pick-and-place workflows to play a game of Tetris successfully. To our knowledge, this is the first work that explores simulating Tetris using a robotic arm.

1. Introduction

Pick-and-place tasks have long been an important part of manipulation, as they often require fine-grained perception and control. Examples such as the Toyota Research Institute's dishwasher loader show that in the real world, robots are able to achieve such granularity and can complete these tasks. In this project, we combine these advances with added complexity resulting from playing a game requiring extremely high-fidelity manipulation. We choose Tetris, a puzzle game that is played by moving seven different pieces that fall onto a playing field. Players can choose the rotations and locations for the pieces. Completed lines on the board clear automatically, and players lose when the uncleared lines reach the top of the board.

Tetris is an interesting problem to solve through manipulation because it is inherently unrealistic: teleporting blocks and clearing rows do not exist in the real world, so this project is an investigation into not only the actual mechanics of solving the manipulation aspects of this problem, but also how much of this unrealistic behavior we can mimic using simulation. Through creating a robotic version of Tetris, we can pave the way for more interactive simulators and robotic games in the real world, such as Connect 4, Jenga, and chess, which also require precise pick-and-place combined with gameplay algorithms and complicated rules.

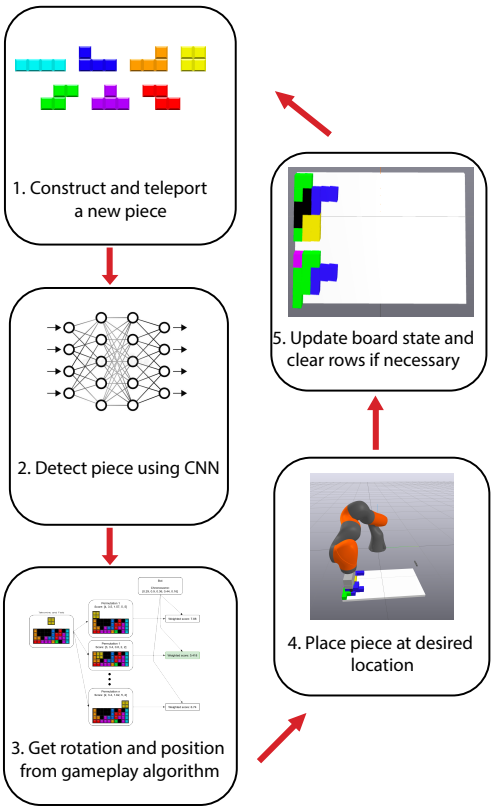

Figure 1. Workflow of TetrisBot. We first teleport a random piece to a fixed location, and then use object detection to determine which piece was generated. Then, our gameplay algorithm assigns a rotation and drop position, which we translate into a final pose on the game board. The iiwa places the piece, and we update our board state.

In summary, our novel contributions are as follows:

- An end-to-end system that can play Tetris automatically, as shown in Figure 1.
- A highly precise pick-and-place workflow that

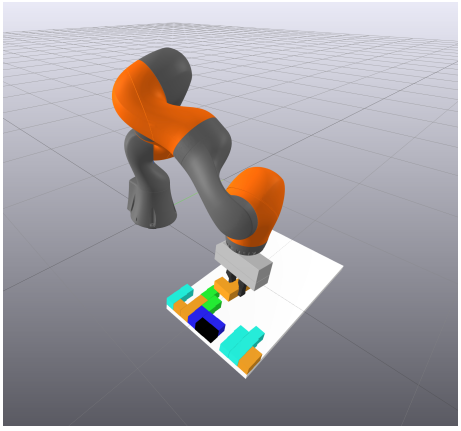

Figure 2. TetrisBot preparing to drop a Tetris piece onto the game board.

achieves realistic grasping and dropping of pieces, as shown in Figure 2.

- A method of simulating unrealistic aspects of puzzle video games by introducing breaks in simulation loops.

2. Related Work

At a high level, pick-and-place is a highly studied field with a variety of applications. We consider the aforementioned dishwasher loader developed by TRI, which tackles a very similar problem as is presented by Tetris - both involve object identification, optimal placement determination of an object while taking into account future objects that may interact with it, grasp determination to interact with the objects, and trajectory optimization. Despite this, the problems are still distinct - the TRI robots have much more advanced perception using principles such as segmentation to properly identify the dishes, while a robot playing Tetris would have to keep track of a world state that can evolve in very different ways based on each successive choice.

Another similar robotic system we can consider is the shopping cart robot, also from TRI, tasked with traversing a supermarket-style area and selecting the relevant items off of the shelves. Having the robot traverse and search a new environment introduces the need for SLAM to the problem, but the core pick-and-place pipeline is still present - identifying objects, grasps, and trajectories to successfully complete the task. Placing objects into a small shopping cart problem is a very similar problem as piece placement in Tetris, where each placement needs to be made while attempting to optimize over potential future states.

Within the realm of games specifically, Tetris can be contextualized as both a single-player game with the objective of surviving for as long as possible, or alternatively as a two-player game with an adversary selecting pieces to limit the player's survival. This ties in with similar long-term strategy games with uncertainty, such as Jenga. Jenga has been used as a target of research for various manipulation tasks, such as visual object identification and meta-learning [3, 4]. This paper, while taking a more cursory approach on the whole, will show that Tetris is a suitable game of study as well for similarly complicated problems in games.

3. Methods

In this section, we describe our approach to building TetrisBot. First, we give a brief overview of the entire simulation. The simulation consists of three fixed objects: (1) an iiwa arm with a wsg gripper placed at the origin, (2) a game board, and (3) a camera placed at (0, 0.5, -0.25) facing in the positive z direction.

In one iteration of the game loop, a Tetris piece is randomly generated and placed at (0, 0.5, 0). We use the camera to take a picture from below, and then feed the image through a CNN to determine the piece type. Then, we use the gameplay algorithm to determine a rotation and drop location. Finally, we construct a trajectory for the iiwa given these drop parameters, and the iiwa then executes the drop trajectory. The gameplay algorithm updates the game state, which is used to clear rows that are complete and update the board in the simulation.

3.1. Setup

We first describe the creation of of simulation components. We use the default iiwa, wsg, and camera SDFs as provided in the `drake` library. We create the game board and Tetris pieces using custom SDFs, as described in the following subsections. The unit block size in our simulation is 0.03.

3.1.1 Board creation

We create the board in an SDF file to be a 0.3×0.6 flat horizontal plane, which is the standard Tetris board size of 10 blocks wide by 20 blocks tall. The corners of the board are located at (-0.4, -0.3), (-0.4, 0.3), (0.7, -0.3), (0.7, 0.3) in the xy-plane. We chose a horizontal representation for the board after experimenting with both vertical and slanted boards. We found that for the vertical and slanted boards, the effect of gravity in the physics simulator caused many imprecise drops, resulting in blocks colliding.

3.1.2 Piece creation

We create the pieces in 14 different SDF files.

First, we create 7 different SDFs for each Tetris piece: I, J, L, O, S, T, and Z. Each piece has its own corresponding

shape as defined by the Tetris game and a unique color. We set the collision geometry of each piece to be 0.01 smaller in each of the x and y dimensions. This is because the pieces collide frequently when placed adjacent to each other, so reducing the collision geometry allows the pieces to visually lock into place next to each other with fewer collisions.

Then, we create 7 more SDFs of unit cubes to represent pieces after they have been dropped. Each cube corresponds to exactly one of the Tetris pieces and takes on that color. In the game loop, once a piece is dropped, it is turned into 4 distinct cubes that now act individually in the next simulation update. To prevent issues with these cubes bumping each other out of place, we reduce the collision geometry by 0.01 in the y dimension.

3.1.3 Board representation

We internally represent the board with a 20 $\times$ 10 array `board_state` with one of the 8 following values: None, I, J, L, O, S, T. We set `board_state[r][c]` to None if no cube is present at row r, column c. We set `board_state[r][c]` to letter l if a cube from piece type l is present at row r, column c. The rows are enumerated from 0 to 20 in the direction of the positive y axis, and the columns are enumerated from 0 to 10 in the direction of the positive x axis.

Given a board representation, we generate the actual board in simulation by placing unit cubes at each row and column where `board_state[row][col]` is a letter l. We select the corresponding cube to piece type l and generate it at that location.

3.2. Simulation Loop

We now describe the simulation loop, which runs the game. The simulation loops through the following sequence of steps for each piece.

1. Piece teleportation: generate a random Tetris piece and teleport it to the starting location (0, 0.5, 0).

2. Object detection: use the camera to take a picture of the piece. Feed the image into a CNN to categorize it as one of the 7 Tetris pieces.

3. Gameplay algorithm: given the piece type and current board state, compute a rotation and drop column for the piece.

4. Pick and place: given the drop parameters, execute a pick and place trajectory that will grasp the piece, move it to the desired drop grasp, and drop the piece in place.

5. Board update: with the new piece dropped, compute the new internal board representation. Clear any full rows (according to Tetris rules), and regenerate the board in simulation from the internal representation.

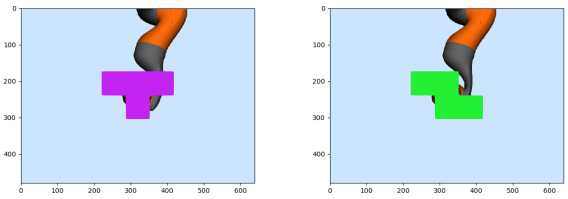

Figure 3. Two examples of inputs to the CNN. We note that since the camera is placed below the robot, the pieces appear reversed.

The approach to each of the these steps is described in further detail in the following subsections.

3.2.1 Piece Teleportation

We "teleport" a random piece to a set location to approximate a piece appearing for the user to place. To accomplish this, we choose a random piece, load the SDF for the piece, and set the free body pose to the default location (0, 0.5, 0).

3.2.2 Object Detection

The camera is placed at (0, 0.5, -0.25), directly below where pieces are placed. Immediately after the piece is teleported, the camera captures an image and runs a forward pass of our object detection neural network. Figure 3 shows example images taken from the camera that are used to identify pieces.

For the object detector, we use transfer learning with a frozen pre-trained MobileNet [1] and replace the last two layers of the network with a fully connected head to predict which one of the seven Tetris pieces has been generated. We create a small dataset using the camera images and perform data augmentation using random color jitters and crops. Since we will receive one of seven exact images from the camera every time, there is no change in rotation, position, or color, of the randomly generated pieces– so we do not need a large corpus of data, since we are guaranteed that the network will see an image that it has previously seen before. Once the image has been classified, we pass the piece information into the gameplay algorithm.

3.2.3 Gameplay Algorithm

The gameplay algorithm takes in the current piece as determined by our object detection algorithm, and computes a rotation and drop column according to the stored game board state. The goal of the algorithm is to allow the robot to survive in the game for as long as possible.

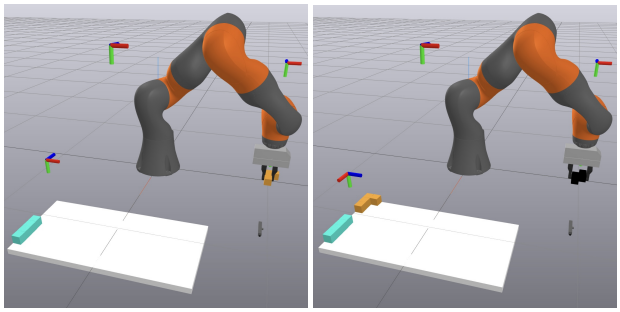

Figure 4. Two examples of the robot grasping the Tetris block, with the key frames of the pick and place trajectories visualized.

The algorithm as implemented is relatively naive and deterministic, and is based on [2]. It considers all potential rotations of the passed-in piece type, and finds the lowest possible row to place the piece in for every column on the board. For each of these potential drop locations, the algorithms selects a drop by minimizing three metrics, in order: the number of "gaps" generated by the drop, where a gap is defined as an empty board location with a filled in board location above it in the same column; the maximal height of the tower of pieces; and the row that the piece is being dropped in. If there are still multiple potential drops, one is selected arbitrarily.

All of these metrics are used in optimal human Tetris play as well, amongst others that are more difficult to quantify, such as "potential for a multi-line clear", since these are worth more points in the game; however, the metrics used are empirically more effective for surviving as long as possible.

In order to communicate the drop decided on by the algorithm to the robot, the algorithm provides a tuple consisting of the following data for the desired drop: the number of clockwise rotations; the row and column locations of the true center of the piece; and the updated board state.

3.2.4 Pick and Place

Once the desired drop rotation and locations are determined, the iiwa constructs a trajectory to perform a pick and place.

First, we construct the iiwa trajectory, which consists of four poses: `X_W_grasp`, `X_W_intermediate`, `X_W_drop`, and `X_W_init`. The first pose `X_W_grasp` is the pose for picking up the piece, which is fixed for all pieces since each piece is generated in the same starting orientation. The second grasp `X_W_intermediate` is also constant, and it rotates the iiwa arm to move to the center of the board. `X_W_drop` is calculated depending on the drop parameters. The RotationMatrix is set corresponding to the desired rotation of the piece. For position, x and y are computed from the row and column parameters given for the center of the piece, and z is set to a constant drop height of 0.2. Finally, `X_W_init` is constant and resets the iiwa arm to the initial state of the trajectory at the end of each pick and place. These poses are constructed into a trajectory using PiecewisePolynomial.MakeLinear.

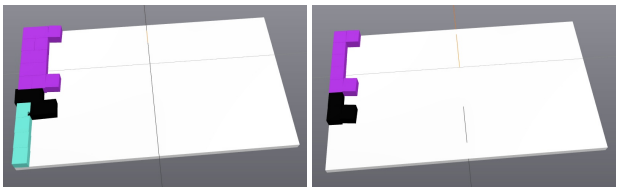

Figure 5. Tetris game board before and after clearing a full row.

Note that while `X_W_intermediate` and `X_W_init` may seem unnecessary, they are important for resetting the iiwa arm to the original state after each pick and place. Without these intermediate poses, the iiwa arm takes a suboptimal route between the grasp and drop poses, preventing it from fully reaching many of the grasps. The four key poses described in this section are visible in two example trajectories in Figure 4.

We also construct a corresponding gripper trajectory. We set the gripper to close at `X_W_grasp` to pick up the piece, we set it to open at `X_W_drop` to drop the piece. We construct the final gripper trajectory using PiecewisePolynomial.FirstOrderHold.

Finally, the iiwa and gripper trajectories are passed to the output ports of the iiwa arm and wsg gripper, simulating the pick and place.

3.2.5 Board Update

Once the piece has been placed, we update the board to contain the new piece. The gameplay algorithm executes the logic of computing the new `board_state` by placing the piece and clearing fully completed rows. As soon as the new piece is placed, we reassign the new value of `board_state`, and the board is then regenerated with cubes in simulation. This also corrects any misplaced blocks, since when the board is regenerated, dropped Tetris pieces are removed and corresponding colored cubes are generated in their intended positions. Figure 5 shows an example game update, where the bottom row is cleared after the drop of the black Z-piece.

4. Results and Discussion

The result was a simulation that successfully simulates the described components of Tetris. We successfully teleport a random piece and detect it, then the iiwa successfully picks and places pieces according to the rules of Tetris, and finally the game board updates and clears rows as expected.

4.1. Evaluation

We evaluate performance on the following metrics:

1. Correct identification of teleported Tetris block.
2. Legal placement of each Tetris block as according to gameplay.
3. Minimized difference between intended block position and actual block position.
4. Correct update of `board_state` following each piece drop.

Metric 1 is essential because it evaluates whether the implementation of the teleportation and object detection are both correct. We measure these using the accuracy metrics of the CNN, and by visual inspection. Metric 2 evaluates the gameplay algorithm, ensuring that we do not request an illegal move. We can evaluate this by visual inspection, or by ensuring that the simulation can solve the system (i.e. it will fail if we try to place two blocks in the same place). Metric 3 evaluates the grasping mechanism: once we have a correct position identified, we want to ensure that we achieve that position. We evaluate this by visual inspection, checking the difference between the center of mass of the dropped block and the intended center of mass.

To test these metrics, we run the game simulation 10 times, for 10 blocks each and observe progression of the simulation. We found that the CNN achieves 100% accuracy, as expected, since we only classify seven blocks. We also find that the Tetris algorithm does not generate blocks that violate game rules; however, it is difficult to measure the optimality of the gameplay algorithm. In addition, our grasping poses achieve very close actual position when compared with intended position; we find that the grasps fail mostly on edges; the first row had the most blocks fall of the board. We originally found that placing pieces directly next to each other resulted in collisions, and mitigated this by making collision geometry smaller than the visual geometry. We find that for Metric 4, the board is updated correctly: cleared rows disappear and the blocks in the rows above move down, and pieces remain intact and in their correct positions if there are no lines that clear.

4.2. Limitations

We faced several limitations when implementing the simulation.

There were several physical limitations of the simulation. The block size was restricted by the width of the gripper, limiting us to a relatively small board. We also ran into difficulties with our original idea of implementation on a vertical or slanted board. We found the effects of gravity too strong, leading to very imprecise drops that would sometimes crash the simulation.

Even with the horizontal board, we still found dropping to be a limitation. Precise drops required the robot to move extremely slowly, which reduced the quality of the game as a visual experience. When the pick and place trajectory was executed in too short of a time period, the pieces sometimes fell on their sides or collided into with blocks, preventing them from dropping in their precise locations. Sometimes imperfect collisions would also cause pieces to wobble in place after landing on the board, creating the illusion of a rotating piece until the simulation resets. These imperfect collisions also occasionally led the entire simulation to crash with convergence errors from MultibodyPlant's discrete update solver.

Another limitation comes with the simulation reset. After each piece drop update, new cubes are generated for the new game state. The old cubes are only deleted after the new ones are generated, creating a doubling effect in the time where the simulation contains both new and old blocks. When we tried deleting the old cubes before, it created a gap in the simulation, showing an empty board in the time it took to load and teleport the new cubes.

Finally, the algorithm we use for gameplay is naive and non-optimal, which creates suboptimal game states after a large pieces have been placed.

5. Conclusions

In this project, we demonstrate that we can combine pick-and-place, simulation loops, neural networks, and gameplay algorithms to create the first end-to-end Tetris-playing robot arm. We show that we can teleport pieces and clear rows using a simulation loop, and we can place pieces extremely precisely with error correction as necessary.

Tetris as a game presents an interesting and diverse series of problems that are relevant to the field of robotic manipulation, and we demonstrated its utility as a tool to study these problems. We could further extend TetrisBot by developing an optimal algorithm for gameplay: algorithms for Tetris typically involve genetic algorithms and reinforcement learning, which we do not use here. We could also further tune simulation parameters to achieve a more realistic board, such as a vertical board with a lower gravity. We could also increase the effect of gravity as time goes on in the case of a vertical board, which would result in blocks falling faster as the game progresses, which is representative of behavior in Tetris. In addition, we can implement multi-player or adversarial Tetris with another iiwa arm.

Our code is available at https://tinyurl.com/roboTetris.

6. Team Contributions

We split up project work to allow different team members to focus on different aspects of the game.

Sadhana created SDFs for the various boards tried, along with Tetris pieces and cubes, and designed the visual and collision geometry of the blocks necessary for simulation. She also implemented the camera system and the neural network for object detection. She also contributed to teleportation of pieces at the start of each move.

Pranav implemented the algorithm for the game. He implemented all logic for the internal representation `board_state`.

Ashley implemented the pick and place trajectories for each piece. She computed the poses required for grasping and placing each piece, and constructed the corresponding trajectories. She also constructed the game simulation loop, which runs the game and generates the game board from the internal representation.

All three members contributed to the writing of this report.

References

[1] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017. 3

[2] Alvin Lin. tetris-bot. https://github.com/omgimanerd/tetris-bot/tree/simple, 2016. 4

[3] Luca Marchionna, Giulio Pugliese, Mauro Martini, Simone Angarano, Francesco Salvetti, and Marcello Chiaberge. Deep instance segmentation and visual servoing to play jenga with a cost-effective robotic system. 2022. 2

[4] Corban G. Rivera and David A Handelman. Visual goal-directed meta-learning with contextual planning networks. 2021. 2