Solving Simple Tiling Puzzles with an End-to-End Robotic System

Anubhav Guha
Massachusetts Institute of Technology

***Abstract*—In this project a robotic system capable of solving simple puzzles is proposed and tested in simulation. A basic puzzle task is introduced, in which a robotic arm is to reconstruct a user-provided template image from a number of auto-generated blocks which represent the puzzle pieces. A complete end-to-end system is developed to accomplish this task - consisting of a perception sub-system for pose estimation, a grasp generation heuristic, a template image identification algorithm, a motion planner, and a number of modular motion/behavior primitives. The robot is successfully tested and evaluated on a number of puzzle tasks of varying image content and complexity.**

***Index Terms*—Robotic Manipulation, Pick-and-Place, Pose Estimation**

I. Introduction

In recent years a number of robotic systems have been used to successfully solve various manipulation tasks. In this paper we pay special attention to autonomous robots that operate without human aid or intervention. General autonomous manipulation tasks often require a number of complex sub-systems to work in tandem, leading to an end-to-end solution that solves a given problem. More specifically, solving a manipulation challenge may require the solution to a large number of sub-problems, such as challenges in perception (pose estimation, localization, image identification), motion planning (trajectory generation, constraint satisfaction, objective optimization) and manipulation (grasp generation, dexterous manipulation). Often these problems are tightly coupled (e.g, a grasp generation task may be thought of as a pose estimation task, or may inform a motion planning decision) and can be difficult to solve individually. As a result, the development of practical and effective end-to-end robotic manipulators and systems can entail the simultaneous tackling of multiple technical frontiers. In order to explore and develop an understanding of some of these challenges, this work aims to solve a puzzle-solving manipulation task. While the proposed task represents a highly constrained and simplified variant of a "true" puzzle task (e.g, a 1000 piece jigsaw puzzle), an adequate solution requires the development of a number of components and the solutions to a number of non-trivial sub-problems. As such, this admittedly simple task poses some interesting challenges and provides insight into the study of robotic manipulation.

A. Related Work

Robotic manipulation tasks in constrained settings have been the subject of much attention - presumably due to the relative tractability of the problems (in contrast to tasks that need to be completed in unstructured or novel environments) and their powerful commercial and economic incentives (e.g warehouse and industrial robotics). In this project, inspiration is drawn from the workflows and system-level philosophies of many of the participating teams in the Amazon Robotics Challenge (ARC), held from 2015 to 2017 [1] [2] [3] [4]. At a high level, the bin picking problems presented in these competitions are similar to the proposed puzzle-solving task in that a successful demonstration requires the generation of consistent and stable object grasps, accurate object pose estimations, and effective motion planning in somewhat tight/constrained spaces. In the winning entries to the ARC events [3] [4], full-stack systems were synthesized and designed so that the various sub-problems could be resolved in modular and occasionally independent manners. This is in contrast to methods that may model the task as an MDP for which a simple "universal" control policy is generated/learned (e.g, as in a full "pixels-to-torque" reinforcement learning approach) [5] [6]. Although many of the ARC entries utilized deep learning methods for perception/segmentation and other more complicated techniques that are beyond the scope of this project, we employ the same types of discrete behaviors and primitives in the puzzle task.

We additionally consider some more specific technical influences and related works. In [7], a combined RANSAC-ICP algorithm was used for 3D object pose estimation. Here, RANSAC was utilized to initialize an appropriate guess for the ICP routine, in order to satisfy ICP's strong dependence on a good initial guess [8] . In [9], environmental supports were used in combination with motion primitives to achieve superlative object manipulation. Although both the objects and the required tasks were more complex in [9] than in the puzzle task, similar concepts are considered and employed in developing the manipulation primitives in this work.

Lastly, we note that this project is very much focused on the integration and implementation of relatively well studied and known techniques. As such, we make prolific use of methods and techniques described in the course notes for the Intelligent Robotic Manipulation class taught at MIT [10].

II. Problem Statement

In this work we consider a simplified puzzle-solving task. Specifically, given an image "template" (e.g, Figure 1), a number of block shaped "pieces" are generated so that, when arranged and concatenated appropriately, the blocks recreate the template image (up to a scaling factor).

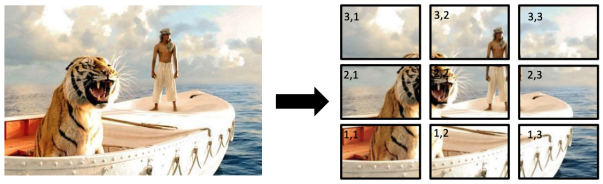

Figure 1. Left: example of a viable template image. This image is then turned into puzzle "pieces" (right), which may be indexed, depending on the desired puzzle dimension (3x3 in this example)

The blocks are to be picked from a bin, and may be initially arranged in arbitrary and "messy" configurations. The task is to have a robotic arm pick the puzzle pieces out of the bin, and to then move these pieces to a flat area so that by the end of operation the desired template image is accurately reconstructed. This flat rectangular region, termed the "construction area", is fenced off on two sides (refer to Figure 5 in Section III-C for a visual example). The robotic solution to this puzzle task should be general, scalable and autonomous.

General: The system should not require any information about the blocks or template image beyond the raw RGB data that is present in the image. Concretely, the labels or poses of the blocks in the bin are not known *a priori*, and we do not inject exogenous features or information (e.g, applying AprilTags to the blocks) into the problem.

Scalable: A compelling solution to this problem will be able to solve a broad range of puzzles in the constrained form described. Specifically, the arm should be able to reliably and effectively reconstruct an arbitrary template image that it is seeing for the first time. Additionally, the arm should effectively complete the task for puzzles of arbitrary size and dimension.

Autonomous: The resulting approach and algorithms should constitute an end-to-end solution that can complete the task autonomously. The inputs to the system are a user-provided template image, the number of associated puzzle pieces, and the physical environment, or scene. The output is then the solved puzzle, which should be solved without any human intervention (e.g, providing directives or block labels in real-time).

A. Assumptions, Simplifications & Implementations

Here we discuss some preliminaries related to the problem formulation and the proposed solution.

Puzzle Discretization: All puzzle pieces are represented as rectangular cuboids, with length, height and width of $0.075m$, $0.05m$, $0.05m$ respectively. As a result, each block has four rectangular faces and two square faces. The oblong faces (with dimension $0.075m \times 0.05m$) are used to reconstruct the template image. The provided template is then re-scaled so that an $n \times m$ (with $n, m \in \mathbb{N}$) grid of $0.075m \times 0.05m$ rectangles *exactly* covers (no overlaps or extra space) the desired template. Note that this approach imposes mild restrictions on the relative length and width of the template. As an example, an image with aspect ratio $2:3$ (such as Figure 1) can be represented by a 3×3 grid of these blocks.

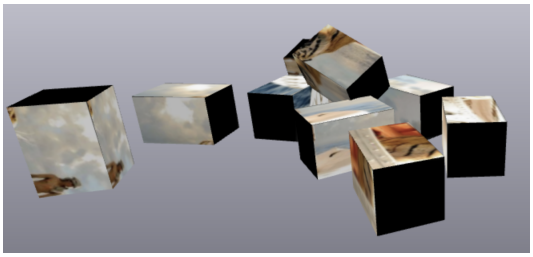

Figure 2. Example of the 3D blocks generate as puzzle pieces for the template image in Figure 1. Note that the sub-image is pasted along each of the four oblong block faces. The two "small" faces are given some constant RGB assignment

Puzzle Blocks: Next, the appropriate "mini-template" image is applied to the blocks - that is, the rectangular portion of the template image corresponding to a given block must be pasted onto the block face(s) (see Figure 2). Here we make a major simplification: the associated mini-template image is applied to all four rectangular block faces. Furthermore, the image is applied so that each block has a visual 4-fold symmetry with respect to the axis corresponding to the longest dimension of the block, henceforth referred to as the longitudinal axis. As a result, viewing a block from a single face gives perfect information about the occluded faces. For a human solving the puzzle task, this would be an unnecessary step - in a standard puzzle only one face of a piece contains the sub-image. After picking a piece, a human solver can simply rotate and flip the object until a non-occluded view of the image is achieved. Our ability to easily manipulate the piece in the required manner is largely due to the dexterous and flexible nature of the wrist and finger joints. For this work we use a manipulator (described below) that does not have access to these degrees of freedom. As a result, in certain scenarios and positions it may be impractical or infeasible to perform certain manipulations (such as a $180°$ block rotation) without violating joint or collision constraints. The use of 4-fold symmetry mitigates this deficiency by ensuring non-occluded views of at least 1 image-containing block face. For example, we can infer the content and orientation of the top image of a block by viewing the bottom face.

Robustness & Noise: In this work we assume nearly ideal conditions. RGB-D readings are taken to be accurate and reliable, and are used for image and pose detection with minimal filtering or correction (for example, we do not have to pay attention to the point cloud merging problem in this simulation). Additionally, no noise or random artifacts (defective pixels, background noise, calibration error, etc) are injected into the simulation. We also assume near-perfect knowledge of the puzzle piece and environment geometries. For example, knowledge of the pick bin's dimensions are used to constrain the search for grasp candidates. Lastly, a well calibrated model of the KUKA iiwa and it's associated controllers is assumed. Position control is used for open-loop directives, and the

iiwa state estimation is assumed to be accurate enough for feasible closed-loop control. Furthermore, the geometries of the manipulator body and end-effector are assumed to be known (for example, the dimensions of the Schunk gripper are used to bound and segment search spaces). In practice these are all unrealistic and restrictive assumptions that would likely limit the application of the puzzle robot to "real-world" puzzle tasks. As such, many of the immediate next steps following this work are to test, evaluate and improve (where necessary) the robustness properties of the total end-to-end system, and ultimately deliver a solution that can solve a real puzzle task outside of simulation.

Implementation Details: The Drake robotics toolbox (and it's associated Python bindings) is used to construct and operate the simulation. Libraries such as Open3D and OpenCV are used extensively for image processing. The manipulator is chosen to be a simulated KUKA LBR iiwa (7 degrees of freedom), and a Schunk WSG 050 gripper (1 degree of freedom) is chosen as the end effector. While there are other types of end effectors (suction-based, dexterous, etc.) that could be used to more easily solve the puzzle task (as proposed in [11] [12]), the simple 1 DoF gripper is a general and simple tool that can be used to solve a broad range of tasks. As a result, this work demonstrates that the puzzle task can be tackled without the use of specialized equipment. In order to map a given template image onto discrete puzzle blocks, a script is used to auto-generate .mtl and .obj files via Blender. These files are then loaded into custom SDF models.

III. Methods

We now describe in detail the overall end-to-end manipulation solution for the puzzle task introduced in Section II

A. Pipeline

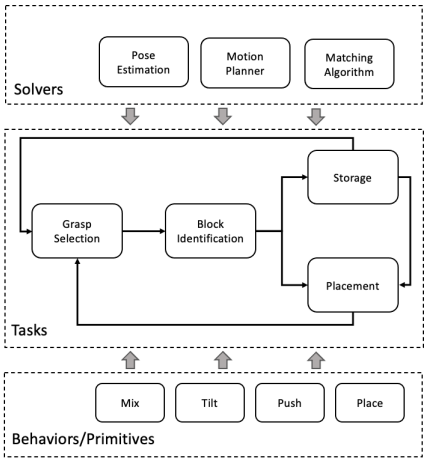

Figure 3. Design of the end-to-end robotic system. A number of low-level primitives and behaviors are defined, along with various "solvers" that are capable of solving specific problems (object pose estimation given a point cloud, generation of a joint angle trajectory, etc). These behaviors and solvers are utilized to varying degrees by the different tasks that need to be accomplished. This structure enables efficient re-use of algorithms and motions that are used often.

At a high level, there are a number of basic tasks in perception, motion planning and manipulation that need to be solved for adequate performance in the task. First, a block in the pick bin must be grasped. Ideally this grasp is stable enough to allow for basic manipulation and re-orientation operations. Next, the cameras are used to align the picked block into a "neat" orientation. This serves two purposes. First, it enables easy and efficient execution of the placing and pushing motion primitives later in the process. Second, it allows the lateral RGB-D cameras to take a neat and non-sheared snapshot of the block face. This snapshot is used to identify the puzzle piece, so that the block may be placed appropriately in the construction area. After identification, the puzzle piece is either placed into the puzzle immediately, or is stored in an overflow area. This choice is determined by the placement strategy/algorithm, as well as the identity of the grasped puzzle piece. Eventually the robot will need to return to the overflow area in order to place any remaining pieces to complete the puzzle. If the robot decides to place a puzzle piece, a series of basic motion primitives are used to robustly and efficiently push the piece into the proper position within the puzzle. After placing the piece, the process repeats until all the blocks have been picked and placed properly. In general, the robot will operate between picking, identification, storing, placing, re-picking and pushing until the puzzle is complete.

B. Placement Strategy

We use a simple but effective heuristic to determine a puzzle piece placing strategy. Viewing the puzzle assembly problem from a birds-eye view, the task can be described in 2D. The strategy is then to build the puzzle from the left-to-right, and from bottom-to-top. As an example, we start by placing the bottom-left corner piece, and then building up the left-most column from the bottom up.

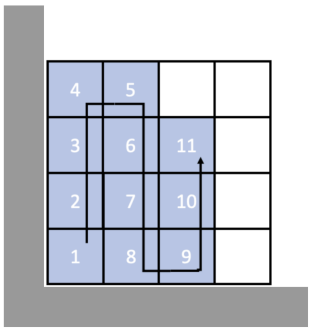

Figure 4. Placement strategy for puzzle pieces. Blocks are numbered in the order in which they would be placed. The grey bars represent the walls of the construction area

We then build the column directly to the right of the previous column, in the same bottom-up manner. This strategy ensures that a block will always be placed into a "nook" - where the bottom and left edges of the piece are constrained by rigid and immovable wall-like entities. In discussing the pushing motion primitives, we will see how this can be valuable.

C. Grasp Selection

The first task in solving the puzzle is to pick up the puzzle pieces (also referred to as blocks). In general, the pieces are initially presented in a cluttered manner inside a bin. RGB-D cameras are placed strategically on the perimeter of the bin. Due to the disordered nature of the items in the bin, it is difficult to spatially separate the distinct objects. While deep learning based approaches (such as Mask R-CNN) have been successfully used to solve similar segmentation tasks, we consider purely geometric approaches in this project. Further, the RGB information in the scene point cloud is not very informative for the grasp selection task. If we knew beforehand that the different puzzle pieces were distinctly and uniquely colored (as in the simplified puzzle task shown in Figure 5), we could perform a simple HSV thresholding procedure to mask out the point clouds corresponding to each block. A model alignment procedure such as iterative closest point (ICP) could then be employed. In solving the puzzle for arbitrary template image content, however, we have no such guarantee on the nature of the image. As such, we use only the depth information from the point clouds.

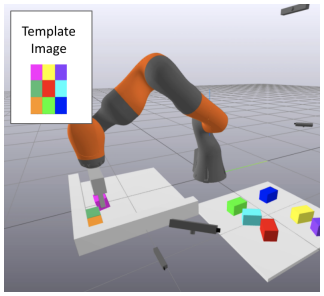

Figure 5. A simplified variant of the proposed task. Blocks may be segmented on the basis of color and then subjected to a pose detection procedure.

Prior to merging the camera point clouds, the depth readings are down sampled and approximate surface normals are calculated. In the standard grasp setting, we may then formulate an optimization problem that promotes grasps which align the fingers in an anti-podal manner with the point cloud normals. In this task, however, we can use additional prior knowledge about the object geometries to first constrain the search and more effectively find good grasps. Specifically, we know that all the objects are cuboids - which means that 1) each object is formed by six intersecting planes (which are either parallel or orthogonal) and 2) the centroids of each face are aligned with the center of mass of the object. As a result, in order to grasp the object in a manner that balances torque due to gravity (and therefore minimize swinging of the object) we can align the finger contacts with the centroids of the cuboid faces. Therefore, if we can find distinct planar faces in the point cloud and identify the centers, we can improve the quality of the generated grasp candidates. Algorithms 1 and 2 are used to estimate the centroids and geometries of planar faces in the observed point cloud.

Algorithm 1 Find Cuboid Faces

Input: points, normals
Initialize : face_list
1: **while** points.Length() > 1 **do**
2: point $\leftarrow$ randPoint(points)
3: plane $\leftarrow$ findCoplanar(point, points, normals)
4: face $\leftarrow$ searchFace(point, plane)
5: face_list.Append(face), points.Remove(face)
6: **end while**
7: **return** face_list

Algorithm 2 Find Face Center

Input: face
Initialize : min_area $= \infty$, center $= \varnothing$
1: face_2D, reverseProject2D$(\cdot)$ $\leftarrow$ Project2D(face)
2: hull $\leftarrow$ convexHull(face_2D)
3: **for** edge **in** hull **do**
4: $\theta \leftarrow \angle$ (edge, $\hat{i}$)
5: hull$_\theta$ $\leftarrow$ rotate(hull, θ)
6: $x_{min},\ x_{max},\ y_{min},\ y_{max} \leftarrow$ bounds(hull$_\theta$)
7: area $\leftarrow (x_{max} - x_{min})(y_{max} - y_{min})$
8: **if** area $<$ min_area **then**
9: min_area $\leftarrow$ area
10: center2D$_\theta$ $\leftarrow [x_{max} - x_{min}, y_{max} - y_{min}]$
11: center2D $\leftarrow$ rotate(center2D$_\theta$, -θ)
12: center $\leftarrow$ reverseProject2D(center2D)
13: **end if**
14: **end for**
15: **return** center, min_area

Given a point $\boldsymbol{p}$ with associated normal vector $\boldsymbol{n}$, Step 3 of Algorithm 1 determines the set of co-planar points as $\{\boldsymbol{x}_i \in$ points $s.t\ ||\angle(\boldsymbol{n}, \boldsymbol{x}_i - \boldsymbol{p})| - \frac{\pi}{2}| < \delta,\ |\angle(\boldsymbol{n}, \boldsymbol{n}_i)| < \delta\}$, where $\boldsymbol{n}_i$ is the normal vector corresponding to point $\boldsymbol{x}_i$, $\delta > 0$ is a threshold, and $\angle(\boldsymbol{a}, \boldsymbol{b})$ is used as a shorthand to denote the angle between vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. Just determining the co-planar points does not approximate the block face well, as in seen in Figure 6:

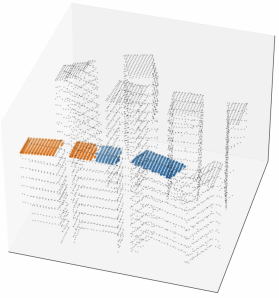

Figure 6. Two co-planar sets determined by Step 3 of Algorithm 1 (orange and blue). The black points represent the downsampled point cloud points. The scene in question is of multiple upright blocks of equal size sitting in a bin. Because the blocks have equal height, the findCoplanar routine finds points from distinct blocks as being in the same plane

We then apply the searchFace routine in Step 4 to more accurately segment the planes into faces. We form a graph, with the points $\boldsymbol{x}_i$ in the plane found in Step 3 representing vertices. Pairs of points are connected by an edge if their Euclidean distance is under a threshold. We then determine the face containing $\boldsymbol{p}$ as all points in the graph component in which $\boldsymbol{p}$ is located. At a high level, this procedure is just further segmenting the plane into groups of points which are "close" to each other.

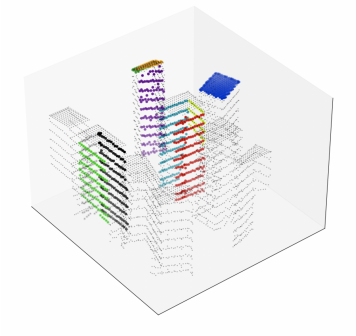

Figure 7. The figure depicts a small number of faces found using Algorithm 1. The searchFace routine is able to effectively approximate the cuboid faces from the pointcloud data, and is less error prone than just searching for planes alone.

After segmenting the scene into distinct faces, we use Algorithm 2 to find the center of each face. First, the 3D face points are projected into two dimensions. Because the points are approximately co-planar, this projection should leave a relatively small residual. Algorithm 2 then implements a basic rotating calipers subroutine to determine the minimum area bounding rectangle of the convex hull. After determining the rectangle, we can easily find the centroid, and project this point back into the world frame 3D coordinates. In the case where we have a full image of a cuboid face (with uniformly distributed points), we can bypass Algorithm 2 entirely, and just calculate the average of all points associated with the face. In practice, however, the faces are often partially occluded. In these cases, Algorithm 2 and the rotating calipers routine offer a method of guessing what the full face looks like.

Figure 8 demonstrates how Algorithms 1 and 2 can be used to find accurate estimates of the cuboid face centers. In particular, Algorithm 2 is used often in the pipeline. In order to generate anti-podal grasp candidates, we first use Algorithms 1 and 2 to find a number of high confidence face centroids in the point cloud. These centroids are sampled, and a large number of gripper configurations are generated so that at least one finger makes contact with a face center. An objective is then formulated, which positively weights alignment of the finger normals with surface normal vectors. Gripper poses are then checked for collisions and ranked based on their fitness as measured by the objective function. In practice, only sampling centroid points for gripper pose generation can be quite restrictive, especially when the scene is sparse and there aren't many identified centroids in the point

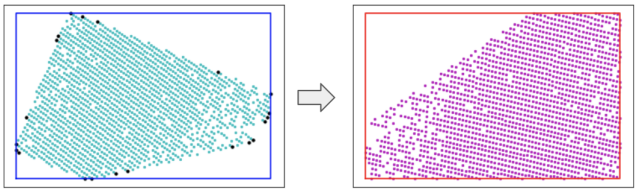

Figure 8. The left figure depicts point cloud points (cyan) taken from a camera view of a block's oblong face (which is known to have dimensions of $.075m \times .05m$). These points have been grouped together using Algorithm 1 and projected into 2D. The black points on the perimeter represent the convex hull, while the blue lines represent the axis-aligned bounding rectangle. The area of this rectangle is calculated as 4.4e-3m^2, which significantly overestimates the true area of the face (3.75e-3m^2). In the right figure, the points (purple) have been rotated so that the enclosing rectangle (red lines) has minimal area. This rectangle has an area of 3.6e-3m^2, which is much closer to the true area. We then determine the centroid of the red rectangle, and reverse the rotations/projections to arrive at a robust estimate of the cuboid facet center

cloud. Instead of solely sampling centroids, we may instead choose to sample points uniformly from the point cloud, and incorporate distance to the co-facet centroid into the objective function, so that points far from the corresponding face center are penalized in the optimization but not excluded entirely. After determining and ranking a number of grasp candidates, the best pose is picked, and the robot picks a block. This block is then moved to the next region of the workspace: the identification station.

D. Pose Estimation

Before we can use RGB image processing techniques to determine the appropriate location of the puzzle piece, the block pose is first estimated. This allows the arm to then re-orient the block so that one of the four oblong faces is directly in view of a camera. To estimate the pose, we use a multi-step approach, as outlined in Algorithm 3:

Step 1: The known geometries of the gripper and block are used to segment the point cloud. Specifically, using the assumption that some point of the block is in contact with a gripper finger, and that the block is a rigid body, we can easily bound the space, as in Figure 9:

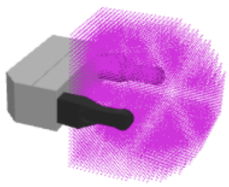

Figure 9. Pink region represent the space in which a block could be, in the gripper frame.

Steps 3-4: A sequential form of RANSAC is used. Roughly, RANSAC is used to fit a single plane to the segmented points. Points that are associated with the fitted plane are removed from the point population, and RANSAC is applied to the remaining points. In practice, this loop is only run 2-3 times

Algorithm 3 Pose Estimate

Input: points

Initialize : min_error $= \infty$, planes $= \varnothing$

1: points $\leftarrow$ segment(points)
2: population $\leftarrow$ copy(points)
3: **for** i **in** range(6) **do**
4: plane, normal $\leftarrow$ RANSAC(population)
5: center_plane, area $\leftarrow$ findFaceCenter(plane) (Alg. 2)
6: area_error $\leftarrow$ min(|area - $.05 \times .05$|, |area - $.075 \times .05$|)
7: signed_normal $\leftarrow$ f(population, plane, normal, area)
8: **if** area_error $<$ min_error **then**
9: min_error $\leftarrow$ area_error
10: translation $\leftarrow$ center_plane + signed_normal
11: best_plane $\leftarrow$ plane
12: **end if**
13: planes.Append(plane)
14: population.Remove(plane)
15: **end for**
16: rotation $\leftarrow$ getRotation(best_plane)
17: translation, rotation $\leftarrow$ ICP(translation, rotation, points)
18: **return** translation, rotation

(instead of 6). Fundamentally, RANSAC is being used to find faces of the cuboid, so we could have instead used Algorithm 1. RANSAC is used here as it utilizes a consensus based approach, and therefore might be a more robust routine when outliers are present. In principle both algorithms could be used in a synergistic manner.

Steps 5-11: Given a plane (and corresponding normal vector) from RANSAC, we use Algorithm 2 to approximate the center and area of the face. We then estimate which "type" of face the plane corresponds to (e.g, long or short face) by comparing expected areas and calculated areas. Note that for a short face the cuboid center will be $.075/2m$ from the plane, and for a long face the cuboid center will be $.05/2m$ from the plane. We can use either the point cloud normal or another simple technique to determine the appropriate direction and magnitude of the normal vector (denote as some arbitrary map f in Step 7). This information is then used to approximate the cuboid center, which provides an estimation of the block's translation.

Step 16: Here we determine the rotation of the block. Note that only two types of faces exist - long and short. We exploit the symmetries of the object by assuming that any observed long face corresponds to the blue face in Figure 10 and any short face corresponds to the red face. We also associate to each of these faces a normal vector and an orthogonal vector within the plane of the face. These define a local reference frame stitched onto the block face. We next determine the face-bounding rectangle using Algorithm 2. From this rectangular representation and the RANSAC plane fit, we can easily determine the face normal and orthogonal vectors in the world frame. We now need to solve the problem of finding a rotation that transforms two vectors in one frame to another. An optimal rotation matrix (which minimizes the distance error after rotation) is found using the Kabsch algorithm [13]

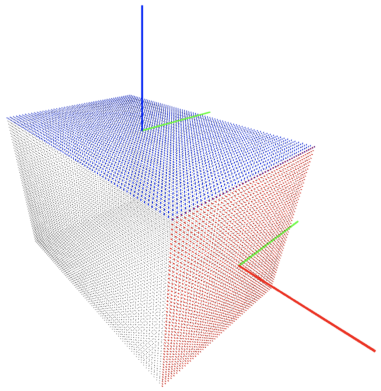

Figure 10. Canonical block with no translation or rotation.

Step 17: Steps 1-16 were used to provide a decent initial estimate of the block pose. We then use ICP to improve this estimate. In practice, ICP worked extremely poorly if the initial pose estimate was not already very good - as a result, the steps leading up to Step 17 are crucial.

E. Pose Alignment

After estimating the block pose, we reorient the block so that an image of the face can extract useful and rich information. In the environment, there is one camera underneath the identification area and another camera off to the side. We want to align the block so that at least one of it's faces is parallel to a camera plane. Based off the object pose estimate, we determine if the block is "mostly" upright or flat. It was found through testing that if the block was initially upright and the robot was asked to reorient the block to be flat, the lack of flexibility and degrees of freedom at the wrist/fingers would lead to collisions, motion planning failures and other catastrophic failures. As a result, if the block is deemed to be upright, we instead orient the block to be "perfectly" upright, and use the side camera to take a picture of the presented face.

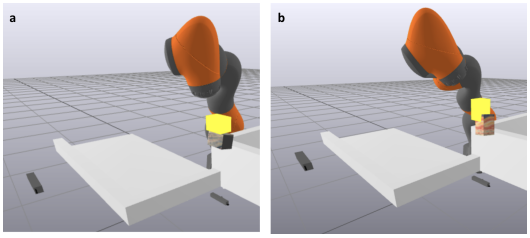

Figure 11. In (a), the block is deemed "flat" and the robot is commanded to reorient the block into the pose given by the yellow point cloud. In (b), the block is instead determined to be upright, and the desired yellow pose is upright. In addition to modifying the block rotation, the desired pose (yellow) is translated so that the block lies in the center of the camera field of view. Note that the gripper is not shown, so as not to occlude the yellow point cloud in the visualization

In determining the desired block pose we first note that, from a purely geometric perspective, the cuboids are 4-fold symmetric about the "long" axis, and 2-fold symmetric about

each of the "shorter" axes. This indicates that there are 16 geometrically equivalent block rotations. We need to orient the block to just one of these configurations. As a result, after determining if the desired orientation is upright or flat, the 16 equivalent rotations are generated. The rotation with the smallest distance to the current block rotation is picked as the target rotation. In general, distance between rotations can be defined in a number of ways. [14] provides a few interesting and valid distance metrics; for this work we take one of the simpler approaches and define the distance between two quaternions as:

$$d(q_1, q_2) = \min\{||q_1 - q_2||, ||q_1 + q_2||\} \tag{1}$$

with $d : S^3 \times S^3 \Rightarrow \mathbb{R}^+$. The target pose translation is a constant value that is the same for all blocks, and which is a function of the camera locations in the environment. The target pose is then fully defined. Denoting the target pose as O_t, the current block pose as O_c, and the gripper pose as G_c the target gripper pose (in the world frame) may then be easily determined as $G_t = (O_t O_c^{-1})G_c$. Note that the gripper pose is found (generally accurately) from the KUKA iiwa.

F. Image Identification

Upon successful execution of the pose estimation and pose alignment sequences, the block face(s) are now "ideally" situated in the field of view of at least one camera. For example, if the block was oriented to be flat, the bottom camera may record an image like Figure 12

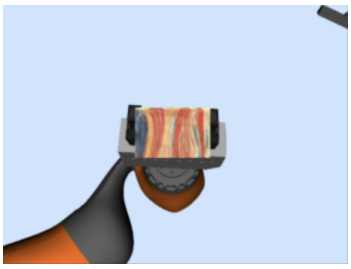

Figure 12. Bottom camera view of an oblong block face after alignment

The first step in the identification procedure is to re-estimate the object pose. We apply the same process described in the previous sections. Next, a block model (with some appropriately fine mesh) is placed at this pose. The next step is to identify the portion of the RGB image that corresponds to the block face. Here we use an assumption that only a single face is presented to the camera (which is why the accurate estimation and alignment in the previous sections is so necessary). All points in the captured point cloud that are less than some threshold distance to any point in the transformed model point cloud are considered to be associated to the block. We then consider only the RGB data corresponding to these points. Experimentally, this process accurately parses out the block image from the RGB capture. Next, the image is re-scaled so that the dimensions match the known size of the block faces. We additionally note that, while from a purely geometric perspective the blocks are highly symmetrical, the image on the block face is not invariant with respect to certain rotations. Indeed, while the block is still visually 4-fold symmetric about it's longitudinal axis, the introduction of surface images causes the block to become (visually) 2-fold anti-symmetric about it's non-longitudinal ("shorter") axes. Specifically, considering the axis extending from the camera (going into the page in Figure 12), the image will flip if the block is rotated $180°$. Identifying this rotation is important, because it informs the robot if the block needs to be rotated prior to placement in the puzzle. Additionally, the existence of this 2-fold anti-symmetry implies that the observed image needs to be compared against two versions of each "sub-image" in the puzzle - one rotated and one not rotated. The sub-image which minimizes some objective metric will then tell us (a) where in the puzzle the block should go and (b) how the block should be rotated before placing. This performance metric is described below.

Template Matching We use a simple but generally effective template matching algorithm. First, we discuss the straightforward and naive approach of comparing each pixel. Because we have resized the segmented RGB image to be of the same shape as the template sub-images, there is a one-to-one correspondence of template pixels to observed pixels. As a result, we could consider minimizing the following:

$$J = \sqrt{\sum_{c\in[r,g,b]} \sum_{i=1}^{W} \sum_{j=1}^{H} (T_{ij}^c - O_{ij}^c)^2} \tag{2}$$

Where T^c, O^c are the matrices corresponding to color channel c of the template sub-image and the observed image, respectively. W, H are the image dimensions, in pixels. Under unrealistic assumptions and perfect conditions, measuring the pairwise Euclidean distance in RGB space might be a viable approach - however the approach is decidedly non-robust to minuscule errors. A more effective approach could be to discretize the images into grids with cells of size n pixels $\times m$ pixels. We could then associate to each cell the average RGB value of all pixels contained in the cell, and order the cells: $C_1, C_2, C_3 \ldots, C_N$, with $N = (W/n)(H/m)$ and C_i returning the mean RGB (vector) value of all pixels it contains. The matched template would then minimize:

$$J = \sqrt{\sum_{i=1}^{N} ||C_i^T - C_i^O||} \tag{3}$$

Where the superscripts indicate that the cell belongs to the template or observed images. While this approach is more robust to minor fluctuations, we can refine it slightly. The approach described above "throws" away much of the information in the cell, reducing the representation to just the mean value. A more complex approach could be to fit a multi-variate normal distribution to the RGB data in a given cell and to then minimize some appropriate metric (such as KL divergence). However even this approach assumes that the distribution of colors is well-modeled by the second-order statistics. Instead, we construct a color histogram for each channel. If each

histogram utilizes l buckets, then a given cell will have three histograms with l buckets. Alternatively, the cell can be represented as a length $3l$ vector. This representation is essentially modeling the (unordered) distribution of color values in a given cell, and for sufficiently large l can capture more relevant information than a basic Gaussian would. We then minimize the metric given in Equation 3, where C_i represent these length $3l$ vectors. This histogram method generally succeeds in returning an accurate identification of the block image, along with the proper rotation determination.

G. Motion Planning

In order to string the results from the previous subsections together, the robot needs to be able to effectively move between the current and commanded poses. In general, all poses provided from the sub-systems (such as the grasp selection module, the alignment routine, etc) are provided in task space in the world frame. For commanded poses that are somewhat distant we first create a pose trajectory. These are generated via quaternion slerp (for rotations) and linear interpolations (translations). We then generate a joint angle trajectory that can be followed by the position-controlled iiwa. Each knot on the joint angle trajectory is determined as the solution to a inverse kinematics (IK) optimization problem. This approach to the joint angle calculation is desirable for a few reasons:

1) Task specific joint angle constraints and environmental constraints are easily incorporated as hard constraints
2) Joint preferences are encoded via a weighting matrix in the quadratic objective
3) Translational slack can be incorporated into the optimization

In (1), we found, experimentally, that certain regimes of operation only needed control authority in a few joints. Where possible, it was desirable to limit joint mobility as the IK optimization would sometimes lead to wild and unpredictable motion (such a phenomenon was observed, for example, in WPI-CMU's DARPA Robotics Challenge entry [15]). In contrast, other primitives (such as object pose alignment) required as many degrees of freedom as were available. These different requirements could be easily incorporated into the joint angle trajectory generator for each sub-task. In (2), certain joints are more "expensive" to utilize than others. For example, the rotational joint that attaches to the end-effector is very valuable (especially when rotating a block) and cheap to use (even large $\pm 180^\circ$ changes are not likely to bring the arm into environment collision) when compared to the base joint. In (3), certain tasks require more precision than others, and this can be easily accounted for. For example, in the image alignment procedure described previously it is very important that the desired rotation and XY position is achieved, however significant slack in the Z direction can be tolerated. In the placing primitives (to be described), extreme precision is required, so very little slack is afforded to any of the six degrees of freedom.

In general, the IK optimization approach works very well - however it was noticed that occasionally the procedure would fail to converge, or that the resulting joint angles would lead to destructive or non-ideal behavior. These problems were solved by conservatively constraining the joint angles (where feasible) and generating more finely distributed pose knots. The generated joint angle knots are then linearly interpolated to produce a continuous time trajectory which is sent to the iiwa controller. A PD controller is used: because certain primitives involve the explicit "over-commanding" of target poses (as will be seen below), an integral term may promote undesirable behavior and issues with windup.

H. Motion Primitives

Finally, we discuss the motion primitives that are used throughout the end-to-end system. There are a number of "types" of motions that are used on an ad hoc basis. Simple ones include the transfer of a picked block from the bin to the identification area, and the transfer of the block from identification area to placement area. These are often just sequences of two or three poses and will not be discussed here. Other primitives are more complex and interesting: in this section we will describe the mixing, tilting and place & push behaviors.

Mixing: The mixing behavior is a sequence of hard-coded gripper poses. These poses are then transformed into a continuous time joint angle trajectory via methods described in Section III-G. The sequence is executed when the robot has attempted and failed, three times consecutively, to grasp and lift a block from the pick bin. Recall that the grasp generator returns a number of grasp candidates - these are tried in turn until the failure condition is hit. The motivation for the mixing behavior is the assumption that if the failure condition has been hit, then the blocks are arranged in the bin in a manner that makes picking or grasping difficult. Moreover, after a few grasp attempts it is likely that the point cloud over which the grasp candidates were calculated is no longer representative of the scene, so it does not make sense to continue attempting picks with the old grasp candidates. As a result, the mixing behavior is initiated, which sweeps the end-effector in a hard-coded pattern around the interior of the bin. The hope is that vigorous motion inside the bin will disturb the block positions enough that viable anti-podal grasps may emerge - however there is no guarantee that the mixing sequence will improve the situation. To prevent the robot arm and gripper from occluding the cameras, the robot is then directed to lift the arm vertically out of the bin so that the suite of cameras may be used to reevaluate the scene. The methods described in Section III-C are then employed to re-attempt a pick.

Tilting: The tilting procedure is used to place blocks that are oriented vertically in the gripper's fingers (see Figure 11b). Recall that the oblong face of the block must be placed face-down/face-up. As a result, the block must be rotated by 90° prior to entering the puzzle. In some joint configurations it can be destructive or impossible to achieve this rotation. Additionally, if the robot was to successfully rotate the block by a full 90°, it is likely that during placement the gripper fingers will come into contact with the "floor" - making it

difficult to release the object in a clean way. As a result, if a block is determined to be upright the robot will tilt the piece 45° prior to placement. Because the block in the upright position is taller than it is wide, this will cause the block to fall into the desired flat position when the gripper opens the fingers. The newly placed block is then amenable to the pushing primitives that will be discussed next (although the center of the block will be slightly translated compared to a flat brick placed in the normal way - this can easily be incorporated into the primitive by exploiting knowledge of the block geometry).

Place & Push: The behaviors that tie the whole end-to-end system together are the placing and pushing primitives. There are two types of placement - placing into the puzzle, and placing for storage. Recalling the strategy given in Section III-B, the blocks are placed in a specified order. As such, after identification many blocks are moved to a storage region, where they can be retrieved at a later time for puzzle placement. The identification procedure establishes the position of the block in the puzzle. These positions are mapped to positions in the storage area. For example, the top right block of the puzzle will be stored in the top right corner of the storage area. If a block is identified for storage, the storage placement sequence is executed to place the block cleanly onto the floor. Note that the ability to place "cleanly" (e.g, the bottom-facing face of the block is parallel with the floor) is predicated on the performance of the pose alignment algorithms given in Section III-E. After placement, the RANSAC-ICP method described in Section III-D is used to determine the pose of the placed block. This will be necessary information when the robot comes back to eventually pick and place the stored block.

We now describe the puzzle placement primitives. Note that prior to executing this behavior we have an estimate of the end-effector pose in the block frame (either from the pose alignment sequence, or from the storage placement sequence, discussed above). This relative pose is then used to place the block accurately in a desired position. Many of the low-level behaviors and commands are re-used from the storage placement primitive - the main difference is that the target block pose is different. This target pose is indexed by the block identity. As illustrated in Figure 13, the block is strategically placed to be offset from it's appropriate position in the puzzle. This will then enable the pushing primitives to cleanly and robustly correct the block position.

The push primitives are now described. We use the left and bottom walls of the placement area to constrain the block motion. For example, to move the block in Figure 13 into the bottom left corner of the placement area, the robot applies a negative force in the x direction until contact with the left wall is achieved, and then applies a negative force in the y direction until contact with the bottom wall is achieved. As long as the robot pushes with some minimum force over some minimum time (in both directions), the desired block position will be achieved. Indeed, pushing with a greater force over a larger time period than required will still result in the block reaching the desired position (complicated contact interactions notwithstanding). As a result, the pushing primitive commands the position-controlled iiwa to a pose that would result in the block being pushed *beyond* the walls. This pushing strategy is *robust* in the sense that the walls and rigid bodies in the construction area are used to achieve desirable and accurate positioning, even if the initial block pose is not perfect, or if environmental factors such as friction coefficients are not known and cannot be used to perfectly estimate the required pushing force/time. The pushing primitives work hand-in-hand with the placement strategy - when the robot is placing any given block, there will always be immovable "walls" to the left and bottom of the desired pose (see Figure 4). The pushing primitive can be neatly described by the offset pose, the "overpush" poses, and the gripper movements between these poses. Because these target poses can all be represented as relative to the block pose in the puzzle, these variables are hand-tuned, hard-coded and used for every block in the puzzle. The push trajectories are then determined once the block is identified.

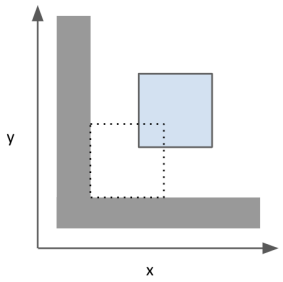

Figure 13. Target block placement position. The place & push strategy purposefully drops the block (blue) so that it is offset in both directions from the final desired resting point (dotted). The grey segments represent the placement area walls.

IV. Results

The results of this project are now presented. Due to the nature of the work, videos of the end-to-end solution in operation are likely at least as informative as numerical/written results. Refer to [16] for a video summary of the project.

While exhaustive simulations and evaluations were out of the scope of this project, the following table provides some interesting success and failure measurements over a small number of runs (n=10):

Puzzle Size	Blocks Before Failure
2 x 2	[4, 4]
3 x 2	[6]
3 x 3	[9, 9, 9 , 7]
4 x 4	[16 , 5, 12]

The number of entries in the rightmost column represent the number of times a given puzzle size was tested. Each entry denotes the number of blocks that were placed before failure. If the task was successful, the number of blocks in the right column is simply the total number of puzzle blocks (successes are in green, failures are in red). To verify the

ability of the robot to solve puzzles with arbitrary image content, 10 unique images were tested (one for each trial). There is a trend towards more failures for larger and more complex puzzles, which is to be expected - larger puzzles require the robot to execute a larger number of behaviors and primitives, which creates a higher potential for failure. Moreover, the area required for the storage and construction regions scales with the number of puzzle blocks. As the puzzle size becomes larger, it becomes more difficult for the robot to cleanly execute some of the required behaviors, as the target poses might be on the periphery of the robot's effective workspace.

V. Conclusions & Future Efforts

In this work we presented a complete end-to-end solution to the simplified puzzle task presented in Section II. A diverse set of techniques were utilized to solve the manipulation, perception and planning problems posed by the task. Image processing and pose estimation methods were used to localize the puzzle blocks, optimization techniques were used to generate grasp candidates and plan effective trajectories, motion primitives were designed to achieve effective and modular behaviors, and template matching algorithms were considered for the identification problem. It was demonstrated through this report and the associated video summary that this approach could viably reconstruct a broad range of images, with reasonable success rates on puzzles composed of a small number of blocks. While this work delivers a solution to the problem proposed in this paper, there are a number of ways the work can be improved, expanded upon and made more applicable to a realistic scenario.

In this paper specific attention has been paid to problems related to pose estimation. In general, this is the task that was the hardest to design a solution for, and the task that had an outsized impact on the ultimate effectiveness of the complete system. For this project only geometric and "classical" image processing/computer vision techniques were considered. Deep learning based methods have demonstrated impressive performance on similar tasks. It is likely that the performance of the object pose detection, segmentation and grasp selection routines could be massively improved by replacing some of the geometric approaches with data-driven methods. For example, Mask R-CNN [17] could be used in the bin picking phase to segment the scene into distinct blocks. After segmentation, a technique such as ICP could be applied to directly extract block pose estimates. Viewers of the associated video will also notice that a large number of RGB-D cameras are utilized - it is possible that superior computer vision algorithms would lessen the required number of cameras. Future work could also explore the use of an end-effector mounted camera, in lieu of some of the fixed cameras. This would have the added advantage of providing access to useful RGB-D data throughout the operation, instead of only having cameras mounted near specific regions in the task space. This could be used for more fine-grained and higher frequency closed-loop control.

A number of the low-level motion primitives may also be improved with further attention. Specifically, the pushing primitives utilize position control of the robotic manipulator. While this rudimentary approach appears to work, the impetus behind the use of the push primitive likely justifies using force-control. For example, instead of hard-coding an "over-push" position, we could achieve robust pushing primitives by instead commanding forces that attenuate when a reaction force is presented by a wall/another block. Another primitive that may be improved is the mixing sequence. Currently, the mixing behavior is hard-coded and does not utilize information about the current bin state to inform the mixing protocol. It is actually possible that the mixing movement could make the grasp selection process *harder* (e.g, if the procedure pushed a previously centered block into a corner, so that no anti-podal grasp around the block exists). A more intelligent behavior would utilize the scene to inform the robot as to which mixing actions should be taken.

Lastly, we consider the problem of high-level task planning, a challenge that has largely been ignored in this project. We simply select the first "good" grasp that is generated, and then use a basic heuristic to decide if the block should be immediately placed. While this approach works in principle, it is possible that the strategy is overly slow and inefficient. For example, different placing strategies might require less unnecessary movement to and from the storage area. Additionally, the grasp selection procedure used in this work has ignored the coupled nature of the bin picking problem and the puzzle placement problem. That is, each time the robot picks a block the decision is being made to greedily use the best identified grasp. Although we are only using depth readings for bin picking, there is a wealth of information in the RGB channels. It might be preferable, for example, to pick a slightly worse grasp candidate if it means that an earlier block in the block order is picked. Although we could potentially hand-design a strategy/heuristic for this, it seems that a learning-based approach such as reinforcement learning could be used effectively to determine more efficient and performant strategies. The temporal correlations of placing/picking decisions and associated action-values can be naturally considered in the standard ADP/RL formulation in a way that is difficult to capture in a hard-coded rules based system or state machine.

Acknowledgment

I would like to thank Professor Russ Tedrake, Terry Suh and Meng Feng for a fantastic semester of 6.881, and for providing valuable guidance and advice throughout the class and this project.

References

[1] Eppner, Clemens, et al. "Lessons from the amazon picking challenge: Four aspects of building robotic systems." Robotics: science and systems. 2016.

[2] Correll, Nikolaus, et al. "Analysis and observations from the first amazon picking challenge." IEEE Transactions on Automation Science and Engineering 15.1 (2016): 172-188.

[3] D. Morrison et al., “Cartman: The Low-Cost Cartesian Manipulator that Won the Amazon Robotics Challenge,” 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, 2018, pp. 7757-7764, doi: 10.1109/ICRA.2018.8463191.
[4] C. H. Corbato, M. Bharatheesha, J. van Egmond, J. Ju and M. Wisse, “Integrating Different Levels of Automation: Lessons From Winning the Amazon Robotics Challenge 2016,” in IEEE Transactions on Industrial Informatics, vol. 14, no. 11, pp. 4916-4926, Nov. 2018, doi: 10.1109/TII.2018.2800744.
[5] Haarnoja, Tuomas, et al. “Composable deep reinforcement learning for robotic manipulation.” 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018.
[6] Gu, Shixiang, et al. “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates.” 2017 IEEE international conference on robotics and automation (ICRA). IEEE, 2017.
[7] H. Kuo, H. Su, S. Lai and C. Wu, “3D object detection and pose estimation from depth image for robotic bin picking,” 2014 IEEE International Conference on Automation Science and Engineering (CASE), Taipei, 2014, pp. 1264-1269, doi: 10.1109/CoASE.2014.6899489.
[8] Yang, Jiaolong, Hongdong Li, and Yunde Jia. “Go-icp: Solving 3d registration efficiently and globally optimally.” Proceedings of the IEEE International Conference on Computer Vision. 2013.
[9] Kazemi, Moslem, et al. “Robust object grasping using force compliant motion primitives.” Robotics: Science and Systems (RSS)(MIT Press, Sydney, Australia, 2012) (2012): 177-185.
[10] http://manipulation.csail.mit.edu/index.html
[11] Fisher, Benjamin. “Design of a Jigsaw-Puzzle-Solving Robot.” (2014).
[12] Saadat, Mozafar, et al. “An image processing approach for jigsaw puzzle assembly.” Assembly Automation (2007).
[13] https://en.wikipedia.org/wiki/Kabsch_algorithm
[14] D. Huynh, “Metrics for 3D Rotations: Comparison and Analysis,” J. Math Imaging Vis. (2009) 35: 155-164. DOI 10.1007/s10851-009-0161-2
[15] C. G. Atkeson et al., “No falls, no resets: Reliable humanoid behavior in the DARPA robotics challenge,” 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), Seoul, 2015, pp. 623-630, doi: 10.1109/HUMANOIDS.2015.7363436.
[16] https://www.youtube.com/watch?v=IRNPaNytWdMfeature=youtu.be
[17] He, Kaiming, et al. “Mask r-cnn.” Proceedings of the IEEE international conference on computer vision. 2017.