A Rapid Development Methodology for an Autonomous Warehouse Picking Robot

Miles C. Aubert[1], Anne W. Draelos[2], Mark Draelos[3], Yihui Feng[1], Humin He[4], Brenton Keller[3], Jianqiao Li[4], Branch Vincent[1], Fan Wang[4], Shengbin Wu[1], Kevin Zhou[4], Ted Zhu[4], and Kris Hauser[4]

***Abstract*— Developing a successful warehouse picking robot requires an integrative approach across multiple robotics disciplines, including hardware design, distributed systems integration, machine learning, grasp planning, and motion planning. Like other robot systems projects, it also requires frequent testing and debugging. Our university team is currently addressing this problem for the Amazon Robotics Challenge 2017 by emphasizing the minimization of design-debug-test cycle times. This is achieved via the use of rapid hardware prototyping, robot simulation tools, and a software infrastructure built around centralized persistent state and pervasive visualization.**

I. Introduction

The Amazon Robotics Challenge (ARC) is a recurring international competition, started in 2015, that asks teams to build autonomous warehouse picking robots that can recognize, pick, and pack a variety of items. Each year's Challenge poses progressively harder problems, and the most recent competition, to take place in July of 2017, requires teams to push the boundaries of manipulation and computer vision research in several ways. First, it requires manipulation of extremely cluttered piles of items packed together in tight spaces. Second, these items are highly variable in size, weight, material, and appearance characteristics. Third, only half of the item set is made available to teams; the other half will be revealed to each team 30 minutes before competition.

This paper outlines Team Duke's current progress toward addressing the problems posed by the ARC. Team Duke is a university team with 14 members, primarily graduate students, with background in many disciplines. The level of involvement of individual students also fluctuates with their course loads and the university's academic schedule. As a result, the team must be managed in an agile, adaptive manner. The barriers of entry to making contributions to the system ought to be extremely low, since some students may be only available for a few months or even weeks. Hence, the team has adopted a software and hardware development infrastructure that facilitates rapid prototyping, testing, integration, and debugging of complex robot systems.

[1]M. Aubert, Y. Feng, B. Vincent, S. Wu, and T. Zhu are with the Mechanical Engineering and Materials Science Department, Duke University, Durham, NC 27708, USA {miles.aubert, yihui.feng,branch.vincent,shengbin.wu}@duke.edu

[2]A. Draelos is with the Physics Department, Duke University, Durham, NC 27708, USA amw73@duke.edu

[3]M. Draelos and B. Keller are with the Biomedical Engineering Department, Duke University, Durham, NC 27708, USA {mark.draelos,brenton.keller}@duke.edu

[4]H. He, J. Li, F. Wang, K. Zhou, T. Zhu, and K. Hauser are with the Electrical and Computer Engineering Department, Duke University, Durham, NC 27708, USA {humin.he,jianqiao.li,fan.wang,kevin.zhou, ted.zhu,kris.hauser}@duke.edu

II. Workcell Layout and Hardware Design

The workcell consists of a robot, a storage system (aka. shelf), and a staging area for the input tote or order boxes. We use a Staubli TX90L 6DOF industrial robot as the base platform. A hybrid vacuum and mechanical gripper is attached as the end effector, and is extended on a long, thin arm to reach into deep corners of the shelf. The vision system consists of multiple Intel RealSense SR300 RGB-D cameras. Two cameras look at the shelf and one looks at the order tote.

The mechanical sub-team was split into two design areas: the storage solution and gripper design. The sub-team chose an agile design and development approach, which allowed it to adapt design requirements as team members became more familiar with the problem posed by the competition and the requirements of the other sub-teams. Also, early prototypes allow the software sub-teams to work with tangible solutions that will be similar to the final products.

A. Storage Solution

The originally proposed shelf design was vertical, made of three cylindrical layers of 40 cm diameter, which would rotate on a fixture at its base. However, analyses by the planning sub-team (Sec. V-A) found this orientation did not admit optimal reachability for TX90L. This led us to choose a horizontal shelf design of 105 cm x 30 cm x 30 cm. The two side compartments have width 30 cm, and the middle compartment has a width of 45 cm. This partitioning allows the shelf to accommodate the items of max dimension (42 cm x 27 cm x 14 cm) in the middle compartment, while preventing items from shifting around too much.

B. Gripper Design

The current working end-effector design combines two gripping mechanisms: a vacuum suction cup and a two-fingered parallel jaw. We found through early testing that the vacuum gripper was reliable at picking up most of the items from a large test pool of items provided by team members and sample items from the ARC organizers. However, the vacuum gripper failed to pick up items that were too heavy or had an irregular surface that prevented a proper suction

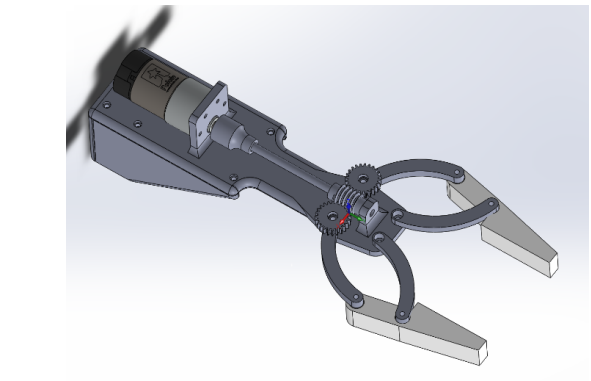

Fig. 1: The two-fingered mechanical gripper design.

cup seal. Back-of-the-envelope calculations concluded that the parallel jaw gripper would be able to pick up most of the items missed by the vacuum gripper.

An early design of a vacuum suction cup mechanism was implemented quickly to allow the other sub-teams to work with a simple gripping mechanism while the more complex mechanical gripper was developed and tested.

The fingers move via a 4-bar linkage, and are actuated by 12 VDC motor-with-encoder, that rotates a worm shaft (Fig. 1). The worm shaft was chosen because the worm drive mechanism is self-locking and confers a large speed reduction ratio. To save space, a long shaft is employed to add separation distance between the motor and gear system. This enabled the gripper to have a minimum width profile of 7.5 cm. 3D printing was used extensively during the development and testing of the gripper.

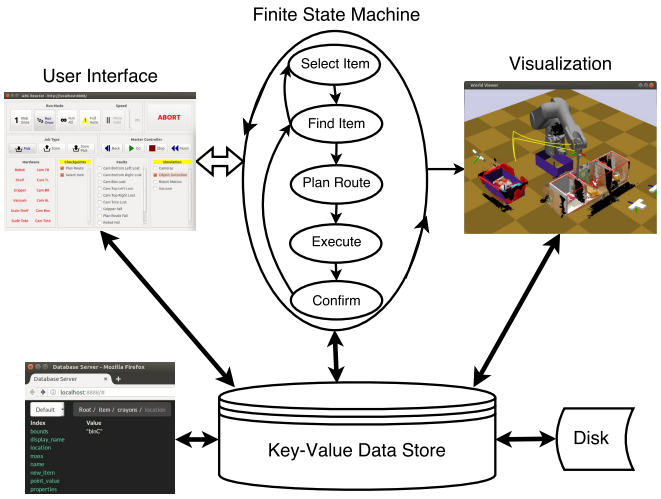

Fig. 2: High-level diagram showing the system's four primary components. Thick filled lines indicate data flow whereas other arrows indicate logical (i.e., control) flow.

III. Software Infrastructure

The software infrastructure is implemented using the four components as described below (Fig. 2). It is primarily written in the Python and C++ programming languages and allows cross-platform development across many machines.

A. Data Store

The data store subsystem provides a centralized mechanism for managing the system's *information state*. System components store their state in the data store, so that the entire system state can be saved and restored. Furthermore, the system's state is readily available for inspection, debugging, and generation of test datasets.

We developed a custom hierarchical key-value data store to support the needs of our of system. Specifically, it provides

- persistence across process lifetimes,
- concurrent access from multiple processes in a networked environment,
- partial and complete saving and loading in a human-readable format,
- copy-on-write snapshots,
- a stateless API readily consumed in Python, C/C++, and JavaScript.

The data store is implemented as a Python HTTP server that exposes database functionality through a JSON-based API to any client capable of generating HTTP requests, including a standard web browser. The server uses the Tornado Web Server (`http://www.tornadoweb.org/`) for handling asynchronous HTTP requests and can service several thousand requests per second.

B. Finite State Machine

The finite state machine (FSM) encodes the central logic of the system. Each state is an operation for the system to perform, and the FSM describes the sequence of states to run during routine and exceptional circumstances. States are run in separate processes to isolate the main FSM logic from unhandled exceptions. Information input and output from each state is controlled through the data store system, which also hosts the signaling needed to decide the specific transition to a subsequent state. The entire system is saved via database snapshot before each state is run and can therefore be restored for debugging purposes.

The master cycle begins with the selection of an item (either from an order file for the pick task or a list of visible items for the stow task). It then proceeds to "find item," "route plan," or "declutter" as appropriate, with success / errors triggering various recovery actions. The FSM continues cycling until all orders are filled or all items are stowed.

C. User Interface

The user interface provides a combination of debugging and execution functionality. The FSM is instantiated from this interface and can be run in various modes for testing purposes. Instead of allowing the FSM to cycle autonomously, here each state can be sequentially advanced, stopped, reversed, or changed mid-operation to fine-tune individual state parameters. Additional diagnostic states can be inserted into the cycle to allow for user intervention. For example, a route checking state can be added to visualize the planned route before execution. Based on the user's assessment, the route can be executed, re-planned, or aborted. Simulation flags can also be set from the user interface, enabling testing

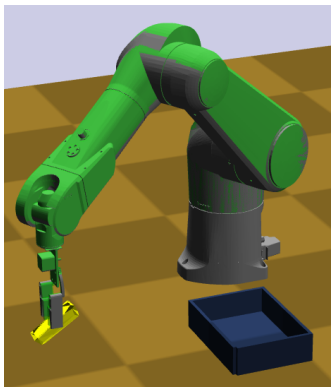

Fig. 3: Pick and place with parallel gripper in physics simulation

of certain system components without requiring the entire physical setup. This allows for modular and simultaneous development of multiple system components.

D. Visualization

The visualization is a 3D graphical view of the system state implemented with Klamp't (`http://klampt.org/`) and OpenGL. This view shows models of the robot and environment, end-effector traces from planned motions, point clouds for objects and cameras, bounding boxes for bins and totes, and key waypoints for motion plan debugging. The visualization synchronizes its display with the database at different rates depending upon the visualized data. For example, the robot and objects poses are updated every frame whereas point clouds are regenerated only when changed. Because the visualization is database-driven, it can run on any computer with network access to the data store.

E. Simulation environment

We also test our planners and different hardware designs in the Klampt simulation environment. It accepts crude 3D meshes constructed from the item point clouds provided by Amazon, and these can be tested in physical simulation quite readily (Fig. 3). Starting from raw 3D CAD files for a new gripper, it takes a couple hours to generate a robot model with the new gripper mounted. As the mechanical sub- team designs new prototype CAD models, the results from testing in simulation can give feedback to the designers quite rapidly.

IV. Perception

The goal of the perception module is to locate and identify objects in extreme clutter (Fig. 4). Perception must be able to execute quickly because the lifetime of our application is around 15 minutes and the environment will constantly be changing. Scenes will contain up to 32 objects from a pool of 40 objects. Half of the 40 objects are known in advance, but the other half will be revealed to us 30 minutes prior to the start of our trial. Because of this time constraint, we focused on segmentation and recognition methods that could be trained quickly, or are model free.

Fig. 4: A segmentation of a cluttered tote, labeled with most likely item identifications. The crayons, white facecloth, and tissue box are incorrectly identified.

We split our approach into two stages, localizing distinct objects in the scene and then identifying each of the objects. Other methods such as R-CNNs [6] are able to localize and identify object simultaneously, but require thousands of manually segmented images. This is impractical for our application as half of the items we are searching for are unknown until 30 minutes prior to the start of our trial.

A. Clutter Segmentation

The input to our localization algorithm is a depth image of the scene, a full color image of the scene (i.e. not just where the depth image is valid), and the transformation from the depth to the color image. We begin by filtering the depth image with a median filter to reduce noise and re-scaling it to be within 0-255 to match the range of the color image. We convert the RGB color image to the CIELAB color space and apply a mean shift filter [1] to remove texture in the image. The depth image is then aligned to the color image. We remove pixels in the color and aligned depth image outside the region of interest and combine them into a 4D array. Finally, we segment the 4D image using graph cuts [2]. A minimum bounding rectangle is fit to each segment. This rectangle is extracted from the full color image and is sent to our object identification method.

We compared our segmentation to a manual segmentation of 8 cluttered shelf images and determined the percent of correctly identified pixels. The percent of pixel-wise overlap between each ground truth segment and the automatically identified one averaged 48.1% with standard deviation 4.7%.

B. Object recognition

We are pursuing two object identification methods, a deep learning approach and a scale invariant feature transform (SIFT) approach. Using two approaches we hope to be more robust to high confidence, but incorrect predictions [5].

For deep learning we are using the VGG-16 network [7], a deep convolutional neural network (CNN) trained on the

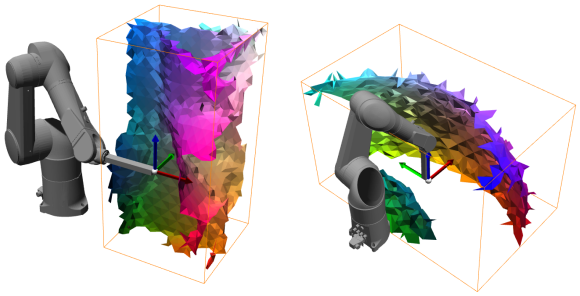

Fig. 5: Boundaries of the optimized redundancy map for two end-effector orientations using the method of [3]. Left: in a forward-facing orientation (i.e., vertical shelf) the singularities about the robot's center line make it challenging to execute continuous Cartesian motions. Right: in a downward-facing orientation (i.e., horizontal shelf) the reachable workspace is a clean torus.

ImageNet data set. We exploit the features learned by VGG-16 when training a CNN for the 40 object classes by fine-tuning the VGG-16 weights rather than starting from scratch. This procedure is known as transfer learning and can take minutes rather than days. To fine-tune VGG-16, we captured a couple thousand images of the items in the item set, which were then augmented with random rotations and rescalings.

Our second identification approach is based on SIFT detection[4]. We train it on six images from different viewpoints for each object. It detects key points on the surface of each object in three color channels and computes a position vector $[r, \theta]$ from the center of the object. When trying to identify an object, we match key points from the unknown object to those computed for the known objects using K-nearest neighbors. Because each key point stores a position vector, we can use key points assigned to an object to estimate the object's center. We find consensus clusters using the DBSCAN algorithm to find the object's center.

The SIFT training process takes approximately 2 minutes to train on 40 objects (6 images for each object). The detector extracts roughly 100,000 key points for all items. The matching and voting process takes about 20 seconds to identify a single object in a full 640x480 image, but is faster on smaller images returned from our segmentation.

V. Planning

A. Workspace reachability and continuity test

To choose the layout of the work cell, we analyzed the reachability and continuity of the TX90L with the tool developed by [3]. Given a specified Cartesian workspace and orientation of the end effector of the robot, this tool chooses among multiple redundant inverse kinematics solutions to minimize the number of discontinuities, and shows the discontinuity boundaries as colored surfaces (Fig. 5). Based on this analysis the downward orientation admits the largest contiguous free workspace. Moreover, the tool verifies that within this workspace, no motion planning besides inverse kinematics is needed to plan a collision-free path.

B. Grasp planning

The approach that we take for grasp planning is to rely almost entirely on the geometry of segmented point clouds. Using geometric features allows our robot to generalize grasps to novel objects more readily than approaches like a grasp database that requires accurate identification of object identity and 6D pose.

We do so by computing several features of the segmented point cloud geometries, including interference with the gripper geometry at some nominal grasp point. We then augment these features with various grasp-relevant attributes of the object, like heavy, slippery, porous, etc., which are estimated by the object identity predictor. The combined feature vector is then fed into a grasp scoring system that is currently hand-tuned but could in the future be learned from data.

A score is given for each grasp for each gripper action (vacuum or top-down parallel-jaw grasp). They are filtered by item relevance to a given action (pick, stow, or declutter), and then the top-ranking grasp is chosen.

C. Packing planning

Packing is an important new requirement for APC 2017, since the shelf is limited to about 25% the volume of previous years' shelves, and orders must be placed into small order boxes (with bonuses for being able to close each box). In physics simulation, we tested both the naive method of dropping an item in the center of the box, as well as a heuristic packing algorithm that uses a randomized search to find a collision-free location. Running 100 randomly-generated orders that are verified to be able to fit inside a given order box, we find that the heuristic boosts the expected number of points received by $\approx 20\%$ for 2-item small-box order and $\approx 10\%$ for 3- and 5-item large-box order.

References

[1] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE T. Pattern Anal. Machine Intel.*, 24(5):603–619, 2002.

[2] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comp. Vision*, 59(2):167–181, 2004.

[3] K. Hauser. Continuous pseudoinversion of a multivariate function: Application to global redundancy resolution. In *Workshop Alg. Found. Rob.*, San Francisco, USA, 2016.

[4] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[5] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, pages 427–436, 2015.

[6] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE T. Pattern Anal. Machine Intel.*, 39(4):640–651, 2017.

[7] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.