2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)

A Case Study of Cyber-Physical System Design: Autonomous Pick-and-Place Robot

Pei-Chi Huang
Department of Computer Science
University of Nebraska at Omaha
Omaha, Nebraska, US
peggy@cs.utexas.edu

Aloysius K. Mok
Department of Computer Science
University of Texas at Austin
Austin, TX, US
mok@cs.utexas.edu

***Abstract*—Although modern robots in warehousing systems can perform adequately in a goods-to-person model using hand-designed algorithms that are specialized to a particular environment, developing a robotic system that is capable of handling new products at an inexpensive cost remains a challenge. A conspicuous example of this challenge is seen in Amazon's use of autonomous robots to fetch customers' orders in their massive warehouses. To encourage advance in this technology, Amazon organized the competition, Amazon Picking Challenge that asked participants to develop their own hardware and software for the general task of picking a designated set of products from inventory shelves and then placing them at a target location (called a pick-and-place task). Current technology for pick-and-place tasks is still insufficient to meet the demand for low-cost automation. Handling awkward or oddly shaped object must still depend on hand-programming or specialized robotic systems, making manufacturing automation less flexible and expensive. In this paper, we shall present the design and implementation of a software system that is a step in advancing the technology toward full automation at reasonable costs. Our system integrates a set of state-of-the-art techniques in computer vision, deep-learning, trajectory optimization, visual servoing to create a library of skills that can be composed to perform a variety of robotic tasks. We demonstrate the capability of our system for performing autonomous pick-and-place tasks with an implementation using *Hoppy*, an industrial robotic arm in an environment similar to the Amazon Picking Challenge.**

***Keywords*-Robotics, Deep Learning, Factory Automation**

I. INTRODUCTION

E-commerce is a competitive business area that depends on cost-efficient product loading system to speed up the shipping process, and to reduce cost. In traditional warehousing processes, workers search for product items in a warehouse, retrieve the items, and then deliver them for packing. Workers spend half of their time traveling back and forth (55% according to [1]) which amounts to a major part of the business cost. To improve the efficiency of the warehousing process, modern E-commerce companies invest enormous amounts of time and effort on automation and robotics research. A well known example is Amazon's use of autonomous robots to fetch customers' orders in the company's massive warehouses. Amazon has been successful in reducing the travel time of workers and the amount of labor [2]. Although modern robots have been successfully deployed in warehousing systems and can perform well in a goods-to-person transfer model using hand-designed robotic algorithms, current systems are specialized to work in specific work environments and to handle specific objects like boxes. Creating robots that are capable of being programmed to handle different products at a reasonable cost and in a short time remains a challenge.

To advance the warehousing automation technology, Amazon organized the competition, Amazon Picking Challenge (APC) [3] that challenged participants to develop their own hardware and software for the general task of picking a subset of products from inventory shelves and then placing them on a nearby table (called a *pick-and-place* task). Motivated by APC, we have developed a pick-and-place, 6 degree-of-freedom (DOF) single arm robot system that is built on our previous work [4]–[6]. Our system is able to accurately and robustly recognize a target object among a cluster of objects that are placed in front of the robot, pick up the object and place it in a designated location with a two-finger gripper (or a suction cup) and complete the task fast enough to satisfy realistic real-time constraints in a factory warehouse. We suggest the reader to view the video accessible via the following link: **http://www.cs.utexas.edu/∼peggy/apc.html** for a demonstration of the real-time performance of our system.

Our system consists of two modules: perception and manipulation. For the perception module, we used the information of the target items to be manipulated by the robot to customize software. Since we knew the items in the Amazon Contest, we used this information to build a set of models. First, the depth information from the Xtion camera sensor for each of the 18 items was gathered and then reconstructed into a 3D untextured mesh through Object Recognition Kitchen (ORK) [7]. Then, all of the objects' meshes and ideal grasp positions were stored in the CouchDB [8] database for query later. To recognize the target item among a cluster of objects, we adopted YOLO [9] to train a deep learning classifier that recognizes the objects and yields the desired Region Of Interests (ROIs) in a camera view in real time. Then, we used LineMOD [10], [11] for template matching to first find out a coarse object 6D pose and finally to refine the pose. For the manipulation

2325-1301/18/$31.00 ©2018 IEEE
DOI 10.1109/RTCSA.2018.00012

module, based on the recognition of the object, the robot chooses either a gripper or a suction device for picking up the object. The gripper was used for most of the objects, and the suction device was chosen for objects that have horizontal or vertical flat surfaces. In the final stage, we applied MoveIt! [12] for the purpose of motion planning and collision checking. We utilized a guided-policy search learning method [13], [14] and visual servoing [15], [16] for trajectory optimization.

The major contributions of this paper are as follows: (i) We have designed and implemented a pick-and-place robot system that exceeds or compares well with the real-time performance of extant systems in the Amazon Contest. (ii) We validated and demonstrated the performance of our software by using the robot *Hoppy* [17] which is an industrial-strength robot for factory automation. (iii) We have built a library of software skills that can be used to compose robotic tasks for various pick-and-place applications.

The rest of the paper is organized as follows: Section II first describes system integration and its architecture. Section III and IV present the perception pipeline and robot manipulation methodology, including object detection, recognition and its region-of-interest, 3D object model reconstruction and target 6D objects pose estimation. Section IV presents the robot manipulation methodology - gripper and suction, and trajectory optimization. Section V demonstrates the synergy of the methods. Section VI concludes the paper and discusses future work.

II. OVERVIEW OF THE ROBOTIC SYSTEM

This section presents the architecture of our autonomous pick-and-place robotic system and its flowchart.

The pick-and-place system in our study is the robotic arm *Hoppy* with its control system that is developed by *Robotics Robotics* (*RR*) Ltd. [17]. *Hoppy*'s end-effector is a specialized gripper with a suction function. The system has a real-time and 3D vision hardware system that we have programmed with various computer vision algorithms, including object recognition and localization, 3D model reconstructions, and integrated with a communication network layer. The goal of the robotic task investigated here is to instruct the robotic arm to recognize and pick up different types of objects and place them into a box. The physical realization of the pick-and-place system is housed in the Cyber-Physical System Laboratory at UT Austin. [18].

A. The Architecture of the Pick-and-Place System

Figure 1 illustrates the architecture of the pick-and-place system. *Hoppy* consists of a six-degree-of-freedoms (DOFs) robotic arm, a desktop PC (*controller PC*) running Ubuntu 14.04 with the RTAI patched real-time Linux kernel and a *vision PC* running on top of Ubuntu 14.04 distribution with Linux kernel version 3.13.0. A controller PC implements the control system model to direct *Hoppy*'s behavior,

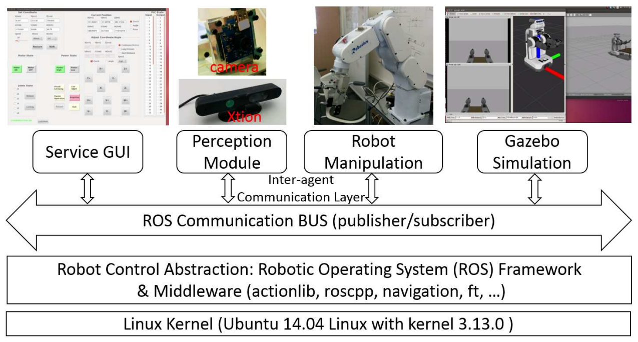

camera
Xtion
Service GUI
Perception Module
Robot Manipulation
Gazebo Simulation
Inter-agent Communication Layer
ROS Communication BUS (publisher/subscriber)
Robot Control Abstraction: Robotic Operating System (ROS) Framework & Middleware (actionlib, roscpp, navigation, ft, …)
Linux Kernel (Ubuntu 14.04 Linux with kernel 3.13.0)

Figure 1. The architecture of the pick-and-place system. All the messages are communicated and transferred through the ROS topic system upon Ubuntu Linux 14.04. At a high-level layer, the autonomy of the system is governed by a service graphical user interface (GUI) which directs perception and robotic manipulation. The robotic behavior was simulated in the Gazebo environment before transferring to the real world. A service GUI acting as the brain of the system decides which actions can be executed based on the sensors and control feedback.

and a *vision PC* implements machine learning and vision algorithms. The robotic control framework is based on Robotic Operating System (ROS) Indigo Igloo (the eighth ROS distribution) with the Gazebo simulator [19] 2.0 and 6.0. Two types of cameras are installed in the system. An Xtion Pro sensor is connected to the *vision PC* and is installed in front of the gripper to capture color images and depth information and generates 3D point cloud data. The other camera is installed in the back of the robotic arm to capture *Hopper*'s surrounding environment. The *vision PC* communicates with the *controller PC* through the ROS communication bus and a ROS master server. The *vision PC* yields 6D poses to the *controller PC*, and the *controller PC* plans the coordinates of the desired trajectory to reach the object and guides *Hoppy* to grasp it.

ROS [20], [21] is an open-source middleware framework for robotics research, and it has a flexible and extensible library for writing components-based robotic software. ROS provides a communication layer on top of host operating systems to support large-scale system integration and modularity for developing robotic applications. In ROS, the abstract layer provides *nodes* and *topics* to facilitate coordination among software modules by passing *messages*; *nodes* represent component modules to perform computations, operating at a fine-grained scale, and *topics* facilitate input and output between nodes by exchanging messages. *Messages* are an ROS data type, stored in .msg files, and are used to subscribe/publish to a topic. For example, our robotic control system has the "Grasp" topic which mixes use of three nodes; one node executing control actions to move the robotic arm, one node performing trajectory planning, and one node performing object detection and recognition. These nodes publish/subscribe messages to the "Grasp" topic. Our ROS-based system uses reusable libraries that are wrapped with a message-passing layer, and can be independently designed to apply to similar robots.

B. The Perception and Manipulation Modules

The two main modules in our system are the perception and manipulation modules. The perception module is used to run learning and vision algorithms; the robot manipulation is used to control *Hoppy*'s arm. To realize the control algorithms, a 3D model of *Hoppy* with a PID controller was implemented on the open-source simulator Gazebo. The model was programmed in ROS format to interface with the Gazebo simulator to simulate trajectory planning and object grasping. Also, a graphical user interface (GUI) service was provided for the human supervisor to issue the high-level commands for the robotic hand to perform the pick-and-place tasks, as well as to monitor the system status. All the messages are communicated and exchanged through the ROS topic system, which is a publish-subscribe system that can efficiently multiplex numerous messages between the modules. In this way, our design system decouples the modules so that each module can be independently implemented for reusability and maintainability.

Figure 2 depicts the flowchart of one pick-and-place experiment. The task is achieved as follows. First, a GUI service keeps polling the status from either the perception module or the robot manipulation module. If a human supervisor adds a pick-and-place task, the perception module will start consuming the video streaming input from the camera sensor, and fills the vision processing pipeline to perform object recognition and to propose an appropriate pose for grasping the desired object. The pipeline can be broken down into four steps, including (1) reading video streams from the depth sensors; (2) predicting and marking the Region-Of-Interest (ROI) of an object with its label, location, and the object recognition probability; (3) estimating the best pose to pick up the object by using LineMOD [10] and 3D models; and (4) transforming the object pose into the robotic arm's poses. The manipulation module is decomposed to execute in four steps: (1) performing trajectory planning, (2) moving the physical robotic hand to approach to the object, (3) grasping (suctioning) the object to move it to the requested position, and (4) releasing the object. The pick-and-place task was first operated in the Gazebo simulation environment, and the resulting controller was transferred and tested in the real-world environment.

Next, we discuss each module in more details.

III. PERCEPTION METHODOLOGY

To perform the pick-and-place task, a fast and accurate perception module is essential for detecting and recognizing the target items on a shelf, and to determine a pose for performing the grasp from a set of feasible grasp candidates. Figure 3 illustrates the procedure. Four NVIDIA Tesla GPU K80 cards running on Ubuntu 14.04 are used to accelerate

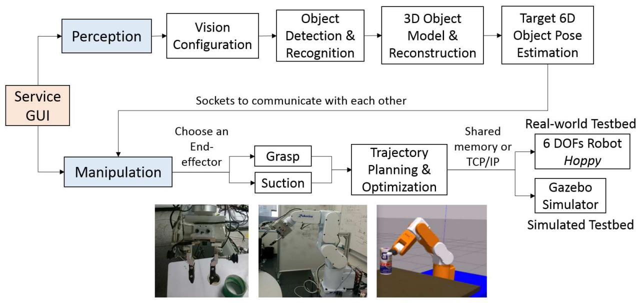


Figure 2. The flowchart of the pick-and-place system. A service GUI for the supervisor to issue the high-level commands to perform the pick-and-place tasks and monitor the system status. The functionalities of perception module are to perform the object recognition and to suggest the best poses for object grasp/suction; the manipulation task is to take the poses and object position output from the perception module to perform trajectory planning and then control/move the physical robotic hand. We implemented the experiments in the Gazebo simulation and then applied the results to the physical robot *Hoppy*.

the grasp training process. We used the CUDA driver and the CUDA 7.5 SDK toolkit [22]. The Xtion driver and the camera driver were installed in OpenNI2 SDK [23]. The following subsections give more details.

A. Object Detection, Recognition and its Region-of-Interest

With images pixels, the object model can be represented by a *region-based convolutional neural network* (R-CNN) architecture. R-CNN has seen great advances in computer vision research and we used it to perform object detection and image classification. The architecture uses region proposal network [24] implemented by several fully connected CNNs to generate potential bounding boxes of objects in an image. To generate the bounding boxes, a selective search method with sliding windows is applied and each sliding window is a small network to slide over the conventional features map. We followed the sliding window approach [25] where individual objects are classified and evaluated in different locations in the image. We adopted an improved version of the algorithm by Girshick et al. [24] whereby region proposal networks (RPNs) are fed as input into a R-CNN to train a classifier from the proposed bounding boxes of each object. Each bounding box is called *Region-of-Interest* (ROI). However, the computation is time-consuming because each component must be trained individually. Better performance with fast R-CNN and faster R-CNN has been reported by Ren et al., [26], [27] who trained multiple objects with a single convolutional network simultaneously. Redmon et al., [9] has also proposed a similar idea that simplifies the computation to a single regression problem and they introduced *You Only Look Once* (YOLO) architecture which can predict each class probability and the coordinates of a bounding box from the extracted features of the images. Our implementation adopted a deep neural network based

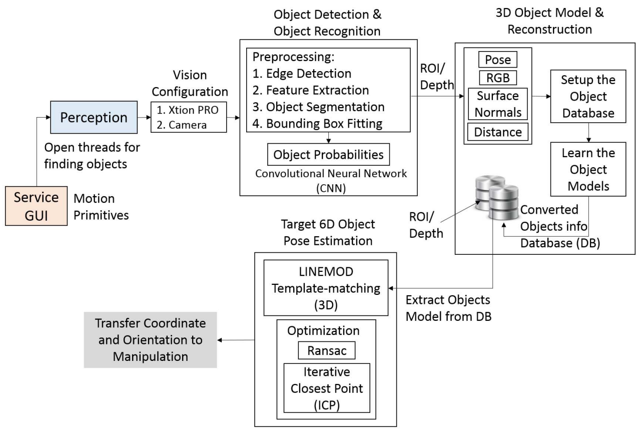


Figure 3. The flowchart of the perception module in the pick-and-place system. A camera and CUDA drivers and toolkits are installed for vision configuration; raw images are preprocessing and calculated each object's probability for object detection and recognition; 3D object models and information of the objects are reconstructed and stored in a database (DB) for easily fetching; LineMOD [10] was utilized to execute template matching and optimized coordinate/orientation for target 6D object pose estimation. Finally, 6D pose estimation was sent to the robot manipulation module. The perception module can predict object's orientation and coordinates in real time.

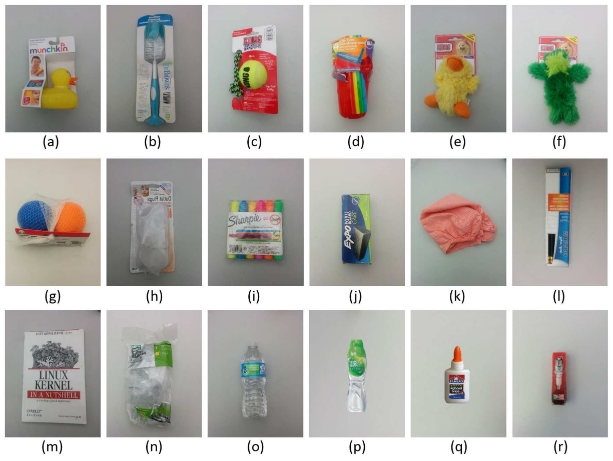


Figure 4. The dataset of example objects from Amazon Picking Challenge competition (APC): (a) a duck toy, (b) a brush, (c) a tennis ball, (d) a set of cups, (e) a yellow toy, (f) a green toy, (g) a set of balls, (h) wall plugs, (i) pens, (j) a box, (k) cloth, (l) a pencil box, (m) a book, (n) glasses, (o)(p) two bottles of water, (q) a glue tube, (r) a spark plug. The pictures from these objects have been semi-autonomously labelled and annotated with their bounding boxes and classifications. The dataset will be used to train our recognition model.

on YOLO [9] which is more robust against variation in brightness and bad angles of view.

Next we shall present the detailed experimental setup and results for the training and testing processes.

1) Training Process: We created a dataset that contains about 40,000 pictures of 18 objects in different orientations and against random backgrounds. Each image is semi-autonomously labeled and annotated with bounding boxes. We used 18 objects from the Amazon Picking Challenge competition (APC) and each object has 2,500 different images. Figure 4 shows that the dataset covers a large variety of object classifications. We implemented scripts to semi-automatically create the annotation to denote the type and coordinates of the bounding box for each object in each image which is then converted to YOLO input format for merging into a training list. Finally, the object detection results were used to train a single convolutional neural network, together with the extracted features of the images and the bounding boxes. The resulting trained neural network model was used to predict simultaneously the bounding boxes for the objects and the likelihood of their classifications.

Because the raw data from the Xtion sensor is 640×480, we followed Redmon et al., [9] to subsample the pixel array to form a reduced 448×448 image; each image with a bounding box was divided into an 7×7 grid cell. We have tried a larger cell, such as 14×14; however, the training process was so time-consuming that we decided to keep 7×7 as the default cell size. This grid cell is used in the detection of an object when its center falls inside one of the grid cell regions. The class prediction probability is defined as $P_r(Class_i) * \, IOU^{truth}_{predict}$ [9], where $P_r(Class_i)$ represents the probability of this class, and $IOU^{truth}_{predict}$ represents the intersection over union (IOU) between the predicted box and the ground truth. This pivot provides us a score for each box to predict which object class appears in this box as well as the accuracy of the predicted bounding box.

The neural network architecture is designed to acquire the class probabilities and coordinates of objects by image transformation to extract feature points. The architecture of the neural network is illustrated in Figure 5 which shows an input layer, followed by 9 convolutional layers and 5 maxpool layers alternately. These layers are followed by 3 fully connected layers, a dropout layer and an output layer [9], [28], [29]. Consider a volume of size width($\widehat{W}$) $\times$ height($\widehat{H}$) $\times$ depth($\widehat{D}$) to which we applied $\widehat{K}$ filters/kernels (called filters size). The spatial extent (called the receptive field of the neuron) of the volume is $\widehat{F}$, and the stride (the steps of the convolution operation) of the volume is $\widehat{S}$. In Figure 5, the first convolutional layer is 7×7, and has 16 filters (or kernels), and we denote it as $7 \times 7@16$. The stride $\widehat{S}$ and the size $\widehat{F}$ are set to 4. After the maxpool layer, the reduction layer volume is $56 \times 56 \times 16$. Note that the spatial extent always equals the depth of the input layer. We processed them in the same way to produce a $7 \times 7 \times 64$ image and followed up with the two fully-connected layers (256 inputs/256 outputs and 256 inputs/4096 outputs, respectively). To avoid overfitting, a dropout layer was added to temporarily remove the unit from the neural network such that the neural network is more general and accurate, and

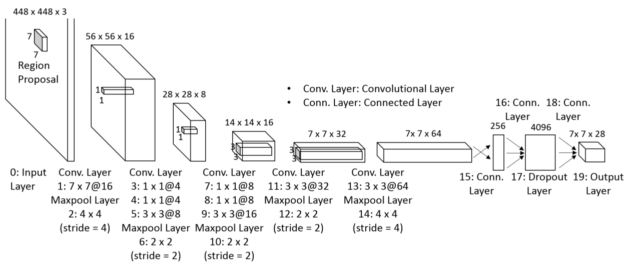

448 x 448 x 3
7
7
Region Proposal
56 x 56 x 16
1
1
28 x 28 x 8
1
1
14 x 14 x 16
3
3
7 x 7 x 32
3
3
7x 7 x 64
256
4096
7x 7 x 28
• Conv. Layer: Convolutional Layer
• Conn. Layer: Connected Layer
16: Conn. Layer
18: Conn. Layer
0: Input Layer
Conv. Layer 1: 7 x 7@16 Maxpool Layer 2: 4 x 4 (stride = 4)
Conv. Layer 3: 1 x 1@4 4: 1 x 1@4 5: 3 x 3@8 Maxpool Layer 6: 2 x 2 (stride = 2)
Conv. Layer 7: 1 x 1@8 8: 1 x 1@8 9: 3 x 3@16 Maxpool Layer 10: 2 x 2 (stride = 2)
Conv. Layer 11: 3 x 3@32 Maxpool Layer 12: 2 x 2 (stride = 2)
Conv. Layer 13: 3 x 3@64 Maxpool Layer 14: 4 x 4 (stride = 4)
15: Conn. Layer
17: Dropout Layer
19: Output Layer

Figure 5. The architecture of the neural network for training detection objects. The network has 20 layers, including the input layer, 9 convolutional layers, 5 maxpooling layers, 3 fully connected layers, 1 dropout layer, and the output layer. Based on previous work [9], [28], the input resolution for classification task is $448 \times 448 \times 3$ (width $\times$ height $\times$ depth), and followed by a series of convolution layers and maxpool layers. The trained neural network model can be used to predict objects and its coordinates of bounding boxes.

the probability $\widehat{p}$ is set to 0.5.

Because the trained model is for large-scale visual recognition, the results could take months of work if the implementation is not computationally efficient. Our implementation uses an open-source deep learning framework called Caffe [30]. Caffe provides an extensible toolkit for state-of-the-art deep learning algorithms and can support CUDA code to exploit parallelism on the GPU architecture. Our neural network was implemented to run on a NVIDIA Titan GPU with 10 cores, with 4 of them allocated and fully utilized to conduct the experiment. Our system was compiled with CUDA 7.5 and CUDNN 3, and the batch size setup was 32 for all neural networks. With the setup, the training time was shortened by a factor of two.

2) Testing Process: For generality, the position and orientation of the objects were randomized. The experimental parameters were the same as the training setting. The trained model was tested within $100cm$ for all the objects placed in different places on the shelf, as shown in Figure 6. A bounding box was automatically generated and centered on the object, and we also predicted the presence of multiple objects at the same time. The results show that the trained deep R-CNNs model can correctly predict what objects are present and where they are, even for a bottle of water or a drinking glass, despite their optical transparency.

3) 3D Object Coordinates: The CNNs through training can detect the ROIs and label objects with probabilities by integrating information from a high-dimensional image provided by the Xtion sensor. Because a bounding box is automatically generated and centered on the desired object, the boundary can be mapped to four coordinates. To locate the object in 3D positions, the center point of a 2D plane for each object (x, y) is determined, and (x, y) coordinates of each object can be mapped to the normalized depth raw data z to determine the 3D location.

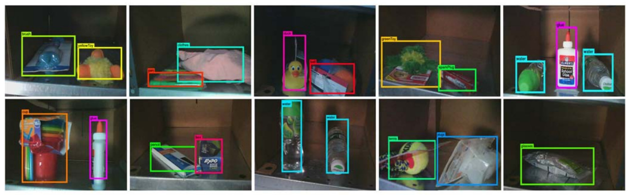


Figure 6. The testing results of the region-based convolutional neural networks (R-CNNs) model. An encompassing bounding box is automatically generated and centered on the desired object through the trained CNNs model, and also shows each object name and its width, length and coordinates. The figure shows that the model can precisely detect the object and its locations.

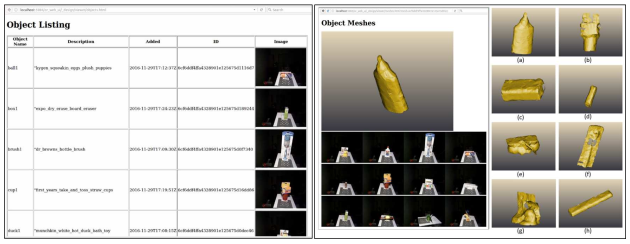
B. 3D Object Model and Reconstruction

Built on work from the open-source community, Willow Garage, the developer of the Personal Robot (PR2) robots, built the Object Recognition Kitchen (ORK) [7] for object recognition. ORK covers input/output handling, robot/ROS integration, and database management. We took advantage of ORK and a database (DB) was constructed to store all the object models with the help of a point cloud library (PCL) [31]. Due to space limitation, we only show here the creation of 3D object meshes in Figure 7 and the visualization of a set of balls, a box, a brush, a cup, a duck, a glue tube, a pencil box and a spark plug together with the associated information in the database.

C. Target 6D Objects Pose Estimation

This subsection describes the procedure to determine 6 DOFs' pose estimations for a robot to grasp the target object.

1) Generate all Templates: The library *object_recognition_linemod* [7] was utilized to generate point-cloud templates for each object from the Xtion sensor data. This library implements the LineMOD framework [10], [11] with an open-source computer vision library – the OpenCV library [32], [33]. OpenCV aims at providing efficient computer vision algorithms that can be applied to generate random views around an object. The different views of the 3D mesh object model can be obtained and stored as templates (point clouds) in the DB, indexed by the viewing angle (i.e., in-place rotation of the camera) and the scale (ratio of image size in pixels to actual object size in meters). The indexing information can then be used to compute (1) depth, (2) RGB, (3) object size, and (4) distance between the object and the camera. This process was repeated until enough coverage of the object was obtained from various viewpoints. The variety of viewpoints is aimed at covering the scales and points of view corresponding to the in-plane rotations of the cameras, given that the object is within a small range of distances from the camera, as is the case in the Amazon Contest. Figure 8 shows the process of generating the templates, as the object was rotated in-plane for each 20 degrees and



localhost:5984/or_web_ui/_design/viewer/objects.html

Object Listing

Object Name	Description	Added	ID	Image
ball1	"kygen_squeakin_eggs_plush_puppies	2016-11-29T17:12:37Z	6cf6ddf4ffa4328901e125675d1116d7	
box1	"expo_dry_erase_board_eraser	2016-11-29T17:24:23Z	6cf6ddf4ffa4328901e125675d189244	
brush1	"dr_browns_bottle_brush	2016-11-29T17:09:30Z	6cf6ddf4ffa4328901e125675d0f7340	
cup1	"first_years_take_and_toss_straw_cups	2016-11-29T17:19:51Z	6cf6ddf4ffa4328901e125675d16dd86	
duck1	"munchkin_white_hot_duck_bath_toy	2016-11-29T17:08:15Z	6cf6ddf4ffa4328901e125675d0dec46	

Object Meshes

Figure 7. Screenshot capture of the creation of 3D meshes of objects and objects listing. Left: Store objects 3D models in the database. Right: Objects' meshes contain (a) a glue tube, (b) a cup, (c) a box, (d) a spark plug, (e) a set of balls, (f) a brush, (g) a duck and (h)a pencil box.

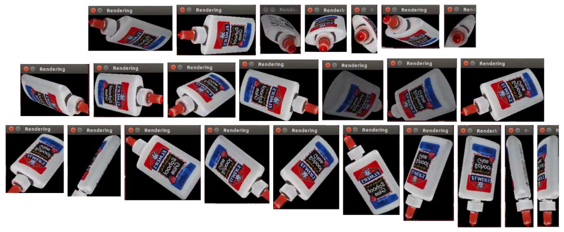
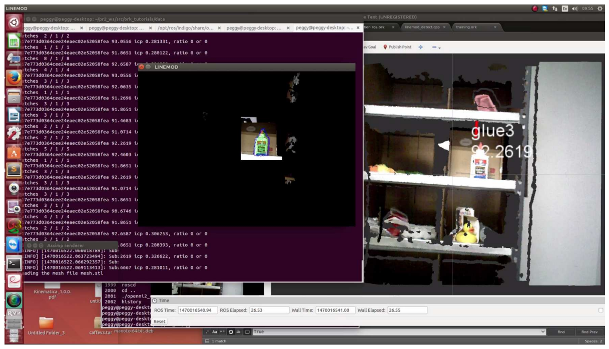


Figure 8. Snapshots are shown the generation of all templates for a glue tube. The input is a mesh; the outputs are depth, RGB, and the distance between the object and the camera from different scales and viewpoints. All the potential templates are stored in the database. The number of templates is below 100. Because the templates are used to match, if the cases are too many, it is difficult to judge. The conclusion is that to generate all templates can be used for templates matching.

also adjusted different scales to produce many samples. As far as we know, there are no known analytical solution on how many templates are enough for matching. Based on our experiments, we observed that too many samples could lead to failure. This is because too many similar samples make it difficult to determine which template is the best. Also, computation time for each template is expensive. Therefore, for each object, we decided to generate no more than 100 templates.

2) Template Matching: Given a set of point-cloud templates covering different views of an object, the 6D pose of an object can be coarsely estimated from the DB. However, we observed from our experiments that the classification results are not very precise because of a huge range of templates in the LineMOD pipeline. Therefore, to improve the accuracy of pose estimation and to reduce false positives, we leverage the output as described in the previous Subsection III-A: a bounding box containing the target object. Since the type of object has already been determined and RGB-D information already narrowed down, the search space of candidate templates is much reduced, and this allows us to use a point-cloud matching algorithm [34] to compute a 6D pose estimation; this algorithm makes use of the surface normals and RGB gradient features of the object silhouette to select templates that score above some threshold value as determined by experiment.



```
peggy@peggy-desktop: ~/pr2_ws/src/ork_tutorials/data
tches  2 / 1 / 2
7e773d0364cee24eaec02e52058fea 93.0556 icp 0.281331, ratio 0 or 0
tches  1 / 1 / 1
7e773d0364cee24eaec02e52058fea 91.8651 icp 0.280122, ratio 0 or 0
tches  8 / 1 / 8
7e773d0364cee24eaec02e52058fea 92.6587
tches  4 / 1 / 4
7e773d0364cee24eaec02e52058fea 93.0556
tches  3 / 1 / 3
7e773d0364cee24eaec02e52058fea 92.0635
tches  1 / 1 / 1
7e773d0364cee24eaec02e52058fea 91.2698
tches  3 / 1 / 3
7e773d0364cee24eaec02e52058fea 91.8651
tches  3 / 1 / 3
7e773d0364cee24eaec02e52058fea 91.4683
tches  2 / 1 / 2
7e773d0364cee24eaec02e52058fea 91.0714
tches  2 / 1 / 2
7e773d0364cee24eaec02e52058fea 92.2619
tches  5 / 1 / 5
7e773d0364cee24eaec02e52058fea 92.4603
tches  1 / 1 / 1
7e773d0364cee24eaec02e52058fea 91.8651
tches  3 / 1 / 3
7e773d0364cee24eaec02e52058fea 92.2619
tches  3 / 1 / 3
7e773d0364cee24eaec02e52058fea 91.0714
tches  3 / 1 / 3
7e773d0364cee24eaec02e52058fea 91.8651
tches  3 / 1 / 3
7e773d0364cee24eaec02e52058fea 90.6746
tches  4 / 1 / 4
7e773d0364cee24eaec02e52058fea 91.8651
tches  2 / 1 / 2
7e773d0364cee24eaec02e52058fea 92.6587 icp 0.306253, ratio 0 or 0
tches  2 / 1 / 2
                                .8651 icp 0.280393, ratio 0 or 0
INFO] [1470016522.063723494]: Sub:2619 icp 0.326622, ratio 0 or 0
INFO] [1470016522.066292357]: Sub:
INFO] [1470016522.069113413]: Sub:6667 icp 0.281011, ratio 0 or 0
ading the mesh file mesh.stl
```

```
1999  roscd
2000  cd ..
2001  ./openni2_
2002  history
```

ROS Time: 1470016540.94

Figure 9. Snapshots of template matching. The glue tube was detected by using R-CNNs and circularized with a bounding box. Since the templates have been constructed, this object can be detected from the input point clouds by executing the LineMOD with ICP and RANSAC. The ROS-RVIZ plugin display window visualized the glue tube's Id, name, and confidence.

3) Optimization of Pose Estimation: To enhance the accuracy of pose estimation, the resulting templates were registered by using the Point Cloud Library (PCL) registration API [35], and the best match is obtained by applying the Iterative Closest Point (ICP) algorithm [36], [37] to iteratively minimize the difference between two point clouds, and by using the random sample consensus (RANSAC) algorithm [7] to eliminate the mismatched points that do not belong to the target area. The snapshots, as shown in Figure 9, are an example of detecting a glue tube, and the outcome is visualized in a 3D visualization tool for ROS – RVIZ that displays the recognized object's Id, name and confidence of the match. However, the results were not as good as we would like due to noise from the environment; our eventual solution is to also combine the technique of visual servoing to lessen the need for very high precision.

Each object's coordinates and orientation is communicated to the manipulation module to guide the robotic arm in executing the pick-and-place tasks.

IV. MANIPULATION METHODOLOGY

Now that we have the 6D end-effector poses of specific objects from RGB-D information, we address in this section the problem of planning a path between the initial pose of the robotic arm and the desired pose for the gripper (suction cup). Figure 10 illustrates manipulation procedure. Each action step in the manipulation is decided for achieving a specific goal along a motion trajectory. The capability of *Hoppy* can be reduced to a set of pre-defined motion actions for controlling the robotic arm.

A. Gripper or Suction Cup

The goal is for the gripper to grasp the object between two fingers that form a vertical parallel jaw. Because the target object's location and orientation is known as input information, the best pose of the gripper for grasping the target object and the force to be applied by the end-effector to achieve the grasp can be computed before actual

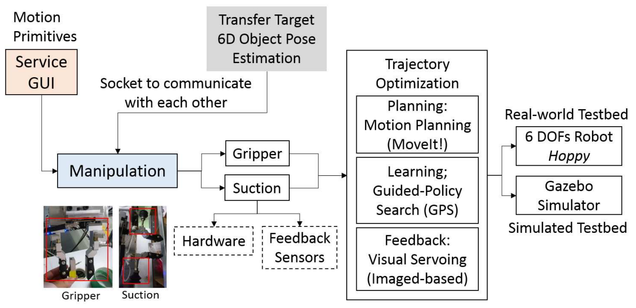

Motion Primitives
Service GUI
Transfer Target 6D Object Pose Estimation
Socket to communicate with each other
Manipulation
Gripper
Suction
Hardware
Feedback Sensors
Gripper
Suction
Trajectory Optimization
Planning: Motion Planning (MoveIt!)
Learning; Guided-Policy Search (GPS)
Feedback: Visual Servoing (Imaged-based)
Real-world Testbed
6 DOFs Robot Hoppy
Gazebo Simulator
Simulated Testbed

Figure 10. The flowchart of manipulation module in the pick-and-place system. The first procedure of this module is a gripper with two fingers and a suction design. Then, three approaches for trajectory optimization are motion planning (MoveIt!), learning algorithm (Guided-policy searching), and visual servoing (Imaged-based). The process is implemented in the Gazebo simulation and then applied to a 6-DOFs robot *Hoppy* in real world.

manipulation begins. However, the gripper still has many restrictions; for example, if the objects are placed on the bottom of the shelf, it is too close to grasp. Therefore, the suction cup was designed to make our system more robust for grasping the objects with a horizontal or vertical flat surface. In general, a suction cup is easier than a gripper to pick up an object. Figure 10 shows the customized gripper with two fingers and the suction cup.

B. Trajectory Optimization

This Subsection focuses on the planning, learning, and feedback approaches to compute a trajectory or produce a general strategy/model.

1) Motion Planning: Robotic motion planning is a maturing field in robotics. Once a valid 6D pose target is determined, a variety of known motion planning algorithms for 6 DOFs can be used to generate a trajectory which is passed to the robotic controller to execute the movement in real time. A widely-used software framework, called MoveIt! [12] has been successfully integrated with many robots, such as Willow garage's PR2 and DARPA's Atlas. We used MoveIt! to construct a collision-free path for *Hoppy*'s gripper or suction cup from a start configuration to a target configuration while taking geometric constraints into consideration. Although this approach can successfully compute collision-free paths, the computation is too expensive for use in real-time applications where it is not practical to wait for the production of a path for the controller to execute. We solve this problem by using guided-policy search [13], [14], [38].

2) Guided-policy search: Assume an *agent* with Markovian system state x_t and action (control) u_t at time step t, $x_t, u_t \mid t \in [1, T]$. *Cost function* $l(x_t, u_t)$ defines the goal of task (i. e., closing the distance between the gripper and the target). The total cost is $\sum_{i=1}^{i=T} l_i(x_t, u_t)$. The current state x_t and the dynamics of the system are used to compute the next state x_{t+1}. A sequence of steps of (x_t, u_t) forms a trajectory= $x_1, u_1, x_2, u_2, \ldots, x_T, u_T$. A *policy* is a mapping (to be learned by the agent) from an agent's state to an action to be performed by the agent. Let $\pi(u_t \mid o_t)$ be a learned nonlinear global policy parametrized by weights θ, where o_t is the observation at time step $t \in [1, T]$. Let $\pi(u_t \mid x_t)$ be a learned time-varying linear Gaussian mixture model (GMM) controller for initial state x_1^i [13], [14], [38].

In the pick-and-place task, our goal is to find a policy $\pi(u_t \mid o_t)$ that an agent can use to choose actions u_t to follow a trajectory for grasping and moving the object to the target destination. Each task is associated with a cost function $l(x_t, u_t)$, and the objective is to minimize the expected cost $E_{\pi_\theta}[\sum_{i=1}^{i=T} l_i(x_t, u_t)]$ over trajectories governed by the policy.

3) Visual Servoing: Because OpenCV has no API to control focal lengths, we first used an open-source software webcam application for the Linux desktop, GUVCview to perform the calibration for the stereo cameras. We modified the application software to provide the functionality for controlling the camera's focus, exposure and resolution in Ubuntu 14.04. We took 30 photos for each time step for the calibration task by exercising the OpenCV sampling code so as to acquire the mapping relationship between the original images and the rectified images. Second, we repeated the same process in the previous step to obtain the relationship between the robot and the camera for the hand-eye calibration. After obtaining the intrinsic parameters (camera model) and extrinsic parameters (rotations and translations), the Scale-Invariant Feature Transform (SIFT) algorithm [39] can then be applied to capture features on the images that are used as input to the visual servoing algorithm. Finally, the imported input data and features were sent to the visual servoing program to move the robotic arm to the desired destination. We also implemented visual servoing in the Gazebo simulation and real-world environment through ROS and then transferred it to *Hoppy*.

Figure 11 shows how visual servoing was implemented. Figure 11(a) illustrates an architecture of visual servoing and how data flows through it, and Figure 11(b) illustrates how to convert OpenCV images to ROS format to be published over ROS. The Gazebo box is responsible for robot control and image fetching. In the Gazebo simulation, *Hoppy* can use the cameras to capture RGB and depth streams, and calculate hand positions to publish as ROS topics. The visual servoing module box took these topics as input by subscribing to them through the ROS master service. After applying the aforementioned steps, the component will generate the next robotic hand position/orientation and then send it back to *Hoppy* to move the end-effector to the next position.

V. SYNERGY OF METHODS

This section describes the integration of the software modules in our system to perform robotic tasks under various scenarios. The experimental setting is described in V-A, and

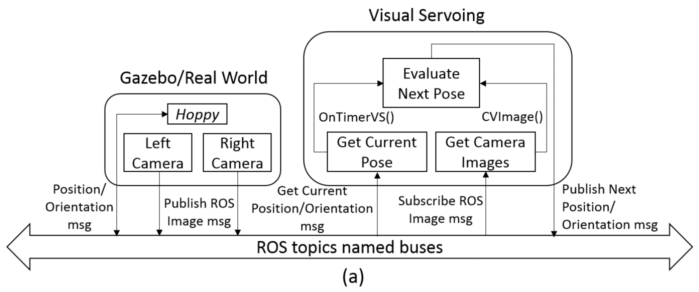


Figure 11. How visual servoing works. (a) An architecture of visual servoing and how data flow through it. (b). Convert OpenCV images to ROS format to be published over ROS.

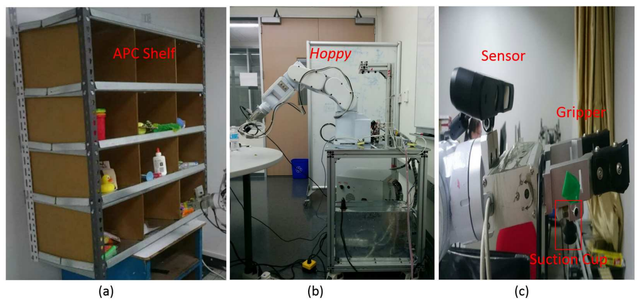


Figure 12. The experimental setting for the Amazon Picking Challenge competition (APC). (a) the shelf with 18 objects in Figure 4, (b) *Hoppy*, (c) the sensor mounted on the end-effector for the pick-and-place task. In the default setting, *Hoppy* executes the gripper. However, if the PLC state input is marked, *Hoppy* works the suction. After the Visual Servo button is pressed, *Hoppy* is directed to approach the object and the two finger motors are synchronized or the suction cup to perform the grasp. This panel can automate the pick-and-place experiment.

an automated platform was displayed in V-B, and the pick-and-place task experiments were performed on a shelf with 18 objects as described in V-C.

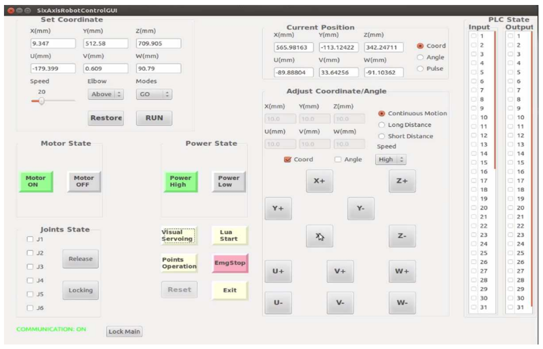
A. Experimental setting

We followed the rules of APC to build the experimental environment. Given a 2 × 2-meter work cell in front of a standard Kiva pad shelving unit with 4×3 bins that are used in the Amazon warehouse, the goal is to autonomously pick up these items and place them in the box. Each bin was loaded with multiple items in different orientations. The 18 objects we used are shown in Figure 4. Figure 12 illustrates the environment, including the 18 objects on the shelf, and a box placed near the shelf and *Hoppy* on the top of the desk. There are some practical difficulties as described below: bins are not of the same size; some objects are too deep inside a bin to fetch; each bin has a "lip" on the bottom edge and also on the top edge to prevent items from slipping off, but this makes it difficult to grasp an object inside; the metallic bottom of the shelf produces optical reflections that create problems for the vision system. The solution to these problems will be reported in future work.

As mentioned in Section III, the perception module is the crucial component to identify the object and to obtain 6D pose estimation with the Xtion camera and the CMOS web camera. We used the Xtion to get point cloud data (by measuring the time of flight), including RGB-D ranging from 0.4 to 1.8 meters, and the resolution is 640 × 480. The experimental hardware consists of (1) a manipulation controller PC that controls *Hoppy* (2) a perception auxiliary computer that runs the operation for computer vision and learning algorithms. We used TCP/IP sockets to communicate over WiFi. The perception software design was based on the middleware framework ROS, and was exercised in the Gazebo simulation first. RVIZ was used to provide visualization capability to observe the robot's behavior running a physics engine. Finally, all the components were migrated to the real robot *Hoppy*. To get RGB and depth information, the Xtion camera was mounted on the end-effector of *Hoppy*, on the pillar at the right-hand side. The RGB sensor rate was set at 30fps and depth rate at 60fps. The depth sensor has $0.2 - 1.2$ meters range for the perception module.

B. An automated platform

An automated platform was built to demonstrate the pick-and-place process. First, a ROS cameras package with an open-source driver was developed. This package was applied to generate point clouds from raw depth map and RGB image utilizing existing UV map in real time. Next, a calibration package was developed to compute the location relationship between the camera and the robot. The trained model could then automatically recognize the object and outputted the object's coordinates and orientation to the manipulation module. Several commands were implemented on the control panel to control *Hoppy*, as shown in Figure 13. The $X+$, $X-$, $Y+$, $Y-$, $Z+$, $Z-$ buttons control the position of *Hoppy*'s end-effector; the $U+$, $U-$, $V+$, $V-$, $W+$, $W-$ buttons can manipulate its orientation. In Figure 13 which shows the control panel, the PLC state input button is used to activate controller, commanding the suction cup to hold onto the objects when the button is pressed. After the Visual Servo button is pressed, *Hoppy* is directed to approach the object and the two finger motors are synchronized to perform the grasp. Figure 14 illustrates the implementation



Set Coordinate

X(mm)	Y(mm)	Z(mm)
9.347	512.58	709.905
U(mm)	V(mm)	W(mm)
-179.399	0.609	90.79

Speed 20 · Elbow Above · Modes GO

Current Position

X(mm)	Y(mm)	Z(mm)
565.98163	-113.12422	342.24711
U(mm)	V(mm)	W(mm)
-89.88804	33.64256	-91.10362

[✓ Coord] Angle Pulse

Adjust Coordinate/Angle

X(mm)	Y(mm)	Z(mm)
10.0	10.0	10.0
U(mm)	V(mm)	W(mm)
10.0	10.0	10.0

[✓ Continuous Motion] Long Distance Short Distance

[✓ Coord] ☐ Angle · Speed High

COMMUNICATION: ON

Figure 13. A screen capture of the panel of control software [17] for the pick-and-place task. In the default setting, *Hoppy* executes the gripper. However, if the PLC state input is marked, *Hoppy* works the suction. After the Visual Servo button is pressed, *Hoppy* is directed to approach the object and the two finger motors are synchronized or the suction cup to perform the grasp. This panel can automate the pick-and-place experiment.

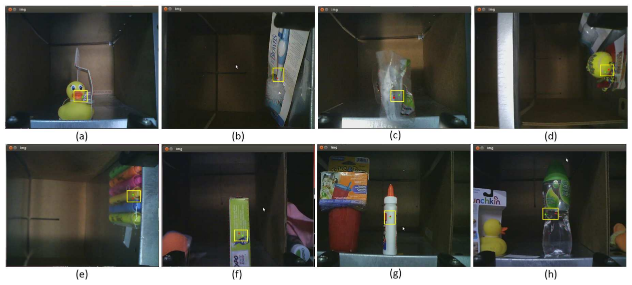


Figure 14. Implementation of visual servoing for 8 objects, (a) a duck toy, (b) a brush, (c) a pair of glasses, (d) a tennis ball, (e) a set of pens, (f) a box, (g) a glue tube, (h) a water bottle. The yellow bounding box indicates the red dot and the green dot; the red dot was returned by the result of visual servoing and the green dot points out the middle of objects. After iteratively executing the visual servoing approach, the two dots are overlapped, which means the robotic arm reaches the target. Then, the end-effector will perform the pick-and-place task for the desired object.

of visual servoing for different objects, including a duck toy, a brush, a pair of glasses, a tennis ball, a set of pens, a box, a glue tube, and a water bottle. The bounding box in yellow represents the output from the visual servo program with a red dot in the middle. Another bounding box gives the target position with a green dot in the middle. The two dots are eventually merged into one, as *Hoppy*'s arm moves toward the final grasping position.

C. *Effectiveness of the Pick-and-Place system*

The video **http://www.cs.utexas.edu/∼peggy/apc.html** demonstrates the implementation of the perception and manipulation modules for the pick-and-place task. Figure 15 shows screen captures taken from a demonstration involving 15 grasping tasks. *Hoppy* was able to approach the target object by using vision and machine learning algorithms to recognize and obtain 3D location and 6D pose. The target objects were grasped by applying visual servoing in the final stage, and were placed in the box. Some of the objects in

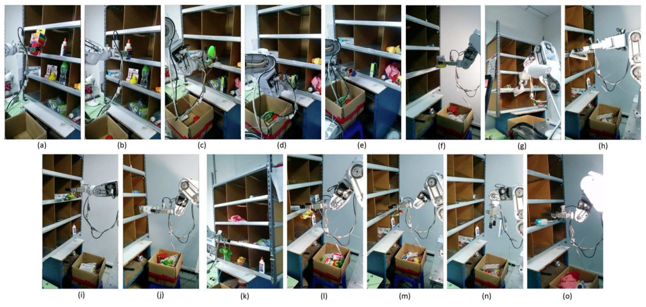


Figure 15. Our system picks and places several objects. The gripper grasped (a) a set of cups, (b) a glue tube, (c) a bottle of water, (d) a box, (e) a set of balls, (f) a duck toy, (g) wall plugs. The suction caught up (h) a book, (i) a tennis ball, (j) a pair of glasses, (k) a green toy, (l) a yellow toy, (m) pens, (n) pencils, (o) a brush. These figures confirm that our system can successfully accomplish the autonomous pick-and-place task.

APC are difficult to grasp, such as a book and a brush, so the suction cup was used. However, our approach does not work very well with two of the objects: plastic seal bag, clothes. Also, some objects are too small to grasp well by the suction cup, such as a spark plug, and some objects have smooth surfaces like a glue tube that is hard to grasp. Further work is needed to design more flexible end-effectors.

VI. Conclusion and Future Work

This paper describes the design and implementation of an autonomous pick-and-place system for an industrial robot *Hoppy*. The two major components of our system are (1) the perception module which performs object detection/recognition, 3D model reconstruction, and pose estimation, and (2) the manipulation module which includes the gripper/suction design and trajectory optimization. We demonstrated the real-time performance of our system through simulation and also with the physical robot *Hoppy* under a variety of scenarios. Our software is organized into a library of modules that can be used to compose robotic tasks to meet application needs. This library exploits known vision algorithms as well as algorithms of our own design.

The main human-machine interface of our system is based on a control panel. For future work, we shall investigate more flexible ways to program the robotic arm to perform tasks that require tight hand-eye coordination of various types. We also envision more powerful human-machine interface techniques that will allow the robot to function in work environments that can tolerate more uncertainty in sensor information and actuator precision. These future systems will have to make better use of real-time feedback techniques such as visual servoing, and also be easy to program in order to automate robotic tasks for fast turn-around, small-lot manufacturing.

Acknowledgment

The work reported herein is partially supported by the Office of Naval Research under ONR Award N00014-17-

2216. Also thanks to all gifts, support and help from the Robotics Robotics Ltd [17].

REFERENCES

[1] J. A. Tompkins, J. A. White, Y. A. Bozer, and J. M. A. Tanchoco, *Facilities planning*. John Wiley & Sons, 2010.

[2] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, p. 9, 2008.

[3] P. R. Wurman and J. M. Romano, "Amazon picking challenge 2015," *AI Magazine*, vol. 37, no. 2, pp. 97–99, 2016.

[4] P.-C. Huang, J. Lehman, A. K. Mok, R. Miikkulainen, and L. Sentis, "Grasping novel objects with a dexterous robotic hand through neuroevolution," in *Proceedings of the IEEE International Symposium on Computational Intelligence in Control and Automation (CICA)*, 2014, pp. 1–8.

[5] P.-C. Huang, L. Sentis, J. Lehman, C.-L. Fok, A. K. Mok, and R. Miikkulainen, "Tradeoffs in real-time robotic task design with neuroevolution learning for imprecise computation," in *Proceedings of the IEEE International Conference on Real-Time Systems Symposium (RTSS)*, 2015, pp. 206–215.

[6] P.-C. Huang, "Real-time robotic tasks for cyber-physical avatars," Ph.D. dissertation, 2017.

[7] Willow Garage, ROS community, "ORK: Object Recognition Kitchen," Available: https://github.com/wg-perception/object_recognition_core.

[8] T. A. S. Foundation, "CouchDB: Couch DataBase," Available: http://couchdb.apache.org/.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[10] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2011, pp. 858–865.

[11] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes," in *Asian Conference on Computer Vision*. Springer, 2012, pp. 548–562.

[12] I. A. Sucan and S. Chitta, "MoveIt!" Available: http://moveit.ros.org, 2013.

[13] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems*, 2014, pp. 1071–1079.

[14] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 156–163.

[15] X. Gratal, J. Romero, J. Bohg, and D. Kragic, "Visual servoing on unknown objects," *Mechatronics*, vol. 22, no. 4, pp. 423–435, 2012.

[16] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 651–670, 1996.

[17] Robotics Robotics Ltd., "The robotic arm *hoppy*," Available: http://www.robotics-robotics.com/.

[18] "The Real-Time Cyber-Pyhsical System Laboratory, Computer Science Department, The University of Texas at Austin," Available: http://www.cs.utexas.edu/users/cps/.

[19] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems (IROS). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, pp. 2149–2154.

[20] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on Open Source Software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[21] ROS.org, "Robot operating system robotics framework," Available: http://www.ros.org/.

[22] NVIDIA Corporation, "NVIDIA CUDA driver and toolkit," http://www.nvidia.com/.

[23] "OpenNI2," https://structure.io/openni.

[24] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[25] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[26] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[27] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 91–99.

[28] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun, "Object detection networks on convolutional feature maps," *arXiv preprint arXiv:1504.06066*, 2015.

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[30] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia*, 2014, pp. 675–678.

[31] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 1–4.

[32] "OpenCV," http://opencv.org/.

[33] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

[34] H. Zhang, P. Long, D. Zhou, Z. Qian, Z. Wang, W. Wan, D. Manocha, C. Park, T. Hu, C. Cao *et al.*, "Dorapicker: An autonomous picking system for general objects," *arXiv preprint arXiv:1603.06317*, 2016.

[35] "Point Cloud Library (PCL) Registration API," http://docs.pointclouds.org/trunk/group__registration.html.

[36] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, Feb. 1992. [Online]. Available: http://dx.doi.org/10.1109/34.121791

[37] K.-L. Low, "Linear least-squares optimization for point-to-plane icp surface registration," *Chapel Hill, University of North Carolina*, vol. 4, 2004.

[38] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[39] W. Burger and M. J. Burge, "Scale-invariant feature transform (sift)," in *Digital Image Processing*. Springer, 2016, pp. 609–664.