Baxter the Builder: Final Report

Emily Goldberg, Ashwad Pandit, Avneet Saluja, Claire Yang

Abstract

Automation of industrial tasks inherently limits human collaboration due to increased inflexibility of robot operation; however, this can be mitigated by including mechanisms for human-robot collaboration in ongoing task modulation. We discuss our implementation of a robot-human collaborative process using a Baxter manufacturing robot to carry out an object-selection pick-and-place task by recognizing handwritten text input.

Introduction

Automated manipulation tasks are foundational in many applications, such as identifying and harvesting ripe fruit in agricultural settings (Billingsley 2019), final assembly of passenger cars in the automotive industry (IFR 2018), and handling hazardous material or waste (Villani 2018). Currently, however, the vast majority of these robots operate “blind” and are non-collaborative. These robots do not perceive their workspace, and instead rely on the predictability of a predefined and carefully calibrated environment. They are also often locked behind physical or sensory barriers to ensure the operator's safety and may require a specialist to program and control the robot. Such robots are highly capable at accurate, precise, and repetitive tasks, but perform poorly when required to react to changes in the environment (Villani 2018). Such robots are becoming increasingly prominent in the industrial workplace: annual sales volume of industrial robots has doubled within the last five years (IFR 2018), rendering the challenge of human-robot collaboration especially important in the next few years. Without proper investigation and preemptive action, it has the potential to introduce greater hazards to the increasing number of human collaborators who are working alongside these robots every day.

Robots that cannot adjust their behavior via intelligent sensor usage are fundamentally limited in their ability to respond to new stimulus; however, these limits can be surpassed with the addition of human cognitive skills and flexibility via collaborative robots. Safe human-robot collaboration can be difficult to implement; for example, the United States Department of Labor Occupational Safety And Health Administration currently requires an industrial robot to have one or more safeguarding devices, such as fixed barriers or limiting devices, in order to protect nearby personnel (OSHA). Such restrictions fundamentally shape the requirements of industrial workflow, limiting the accessibility of the workspace and requiring that operations halt whenever human involvement is required in robot-dominated areas.

To some extent, these limitations can be moderated when robots are equipped with the ability to adjust their behavior based on operator input. However, while all robots are fundamentally

programmable, most industrial robots are programmed prior to use (OSHA). This limits the flexibility of human-directed behavior adjustment and results in more dangerous work environments, since a primary source of human error (and therefore hazard) derives from lack familiarity with the robot's repetitive movements (OSHA). Any adjustments made to a robot's behavior (e.g. to correct for such error, or to adapt to a particular workspace) therefore require a specialist, which prevents natural human-robot collaboration and decreases the overall efficiency of the system.

In our paper, we focus on investigating intuitive, multimodal user interfaces for controlling a robot's actions. We present a framework for the ongoing modification of an industrial robot pick-and-place task via recognition of text in the workspace. The Baxter manufacturing robot made by Rethink Robotics was programmed to carry out an object selection task where object criteria were defined by handwritten text instructions. Further development of similar forms of flexible and human-intuitive interfacing have the potential to ease robot-human collaboration and thereby increase safety and efficiency in a variety of applications.

Methodology

Our Task

We programmed a Baxter robot to identify blocks on the tabletop surface, then sort the correct blocks into a target region based on color. The desired block color is communicated to Baxter via handwritten instructions on the surface of the table. All programming was done using Robot Operating System (ROS) via its Python interface (Quigley et al. 2016). Packages used include baxter_interface (robot management), MoveIt! (inverse kinematics), and OpenCV (perception).

Workspace configuration

All tasks are achieved using Baxter (Rethink Robotics), an approximately-humanoid industrial robot equipped with grippers and cameras on each arm. After Baxter is enabled, one launch file is run to enable the left-hand camera and move both arms into a starting position. The left arm is positioned directly above the workspace; the right arm occupies a "ready" position to the right and above the workspace (**Fig. 1**). Once Baxter has reached its ready posture, a horizontal whiteboard is placed in front of it in a fixed location relative to Baxter's base, such that the whiteboard fills the viewframe of the left-hand camera. On the whiteboard, a color name (chosen from red, green, and blue) is written in block text.

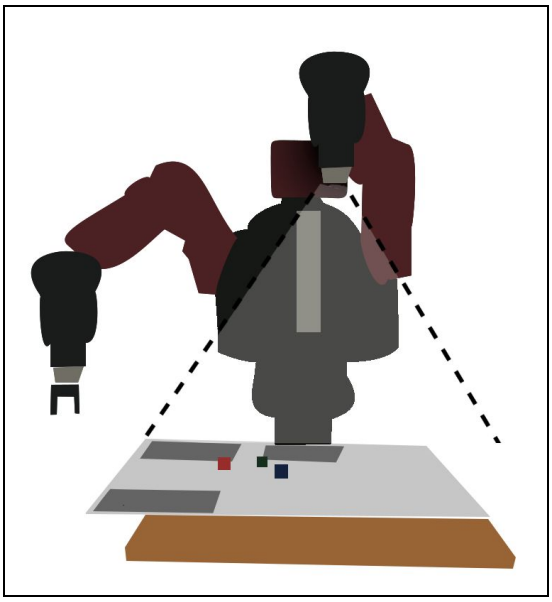


Figure 1. Baxter positioned with left arm ready to provide images for text identification and block localization; right arm in "ready" position. Dashed lines show camera sightlines.

Text recognition and target identification

The *TargetIdentifier* node is responsible for identifying the handwritten text on the whiteboard which corresponds to the color of the block to be picked using image published by Baxter's left-hand camera. First, the region containing the text is cropped. Then various noise in the image, whiteboard, and text is filtered using *OpenCV* and *Textcleaner* by ImageMagick. The preprocessed image containing the text is finally passed to the tesseract library, which interprets the written text. The identified text is then published to the 'target_color' topic to be received by the *BlockIdentifier*.

Object detection and centroid/color identification

The *BlockIdentifier* node identifies the blocks and their locations. It uses OpenCV to convert the camera image to HSV space, and then threshold on red, green, and blue colors in the image; it also subscribes to the target_color topic that is published by the *TargetIdentifier* node to determine which color to threshold on for the identified target. Once the binary image has been created, it creates a list of contours and filters the list for the largest contour. Finally, it calculates the mean of that contour, which is equivalent to the xy coordinates of the block object with regard to the image. The image coordinates are translated into Baxter's xyz coordinate frame via the visual servoing function described below.

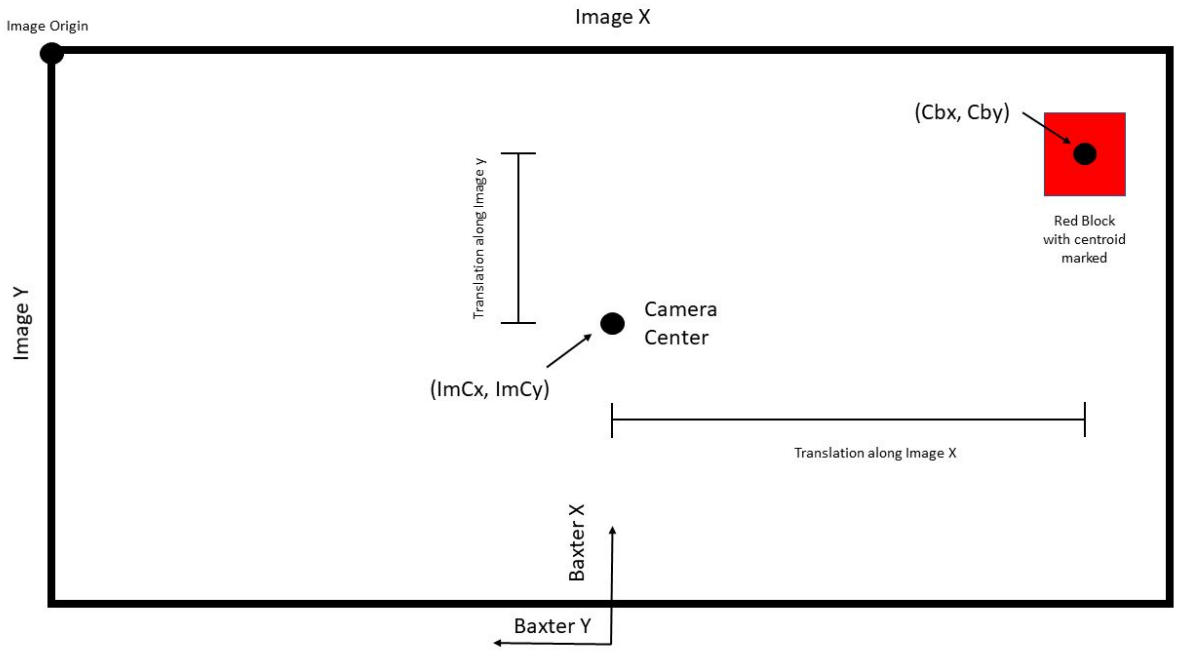
Visual Servoing

For Baxter to move to this location and pick up the correct block, the image coordinates need to be translated to Baxter's coordinate frame. This calculation is performed using the derived formula:

$$X_b = [(C_{by} - ImC_y) * (-CC_x) * H] + baxterLeftArmPose_x + gripperOffset$$
$$Y_b = [(C_{bx} - ImC_x) * (-CC_y) * H] + baxterLeftArmPose_y + gripperOffset$$

Where X_b and Y_b give Baxter's X and Y coordinate; C_{bx} and C_{by} give the block centroid's X and Y coordinates in the image frame; ImC_x and ImC_y give the image center's X and Y coordinates in the image frame (**Fig. 2**); CC_x and CC_y are two camera calibration factors (determined empirically); H gives the height of the camera above the workspace; $baxterLeftArmPose_x$ and $baxterLeftArmPose_y$ give the coordinates of the left arm camera in Baxter's frame; and *gripperOffset* is a factor that adjusts for the difference between the right arm's commanded location and the final gripper location.

Figure 2. Layout of the whiteboard that serves as Baxter's workspace.



The camera calibration factors CC_x and CC_y were calculated by placing an object of known dimensions in the image, and then dividing its dimensions by the length and width in pixels at a given distance from the camera.

Pick and Place

Our system uses a single node to carry out the pick-and-place movement required to move the block from the selection area to the target point. This node, titled *PickPlace*, establishes a service proxy for the service *ObjectLocation* (Miranda et al. 2015), which when called, returns the centroid location of a block of the specified color. This xy location is pre-transformed into Baxter's coordinate frame, and the z-coordinate is assigned to a fixed value (-0.245 m), since the system is only presented with blocks resting flat on a surface of constant height and therefore does not need to estimate the z position of the block and target.

Once *PickPlace* has obtained a block location from the desired block, it moves the block to a hardcoded target location using a combination of MoveIt! and baxter_interface package functions. It makes use of the MoveIt! planning scene to avoid self-collisions; the MoveIt! MoveGroupCommander class for motion planning and execution; and the baxter_interface Gripper class to control the right-hand gripper. In order to avoid any other blocks in the workspace and prevent collisions with the table, waypoints are specified that force Baxter to move directly above the target block before descending to pick it up; similarly, after depositing the block it is forced to raise its open gripper directly upward to avoid knocking the block out of position. When the movement is finished, Baxter returns to its ready position.

Results

Object detection reliability

The object detection and location identification is very accurate under certain conditions and not as robust under others. It is able to identify each colored block sequentially, if there are multiple of the same colored blocks on the table, as well as multiple other colored blocks. However, the color thresholding fails if the blocks are different shades of red, green, or blue, since the ranges on which the node is thresholding were customized for the particular lighting in the lab and the shade of colors. If those variables were to be controlled, the object detection node would be able to detect not only blocks that were of those colors, but other objects of interest as well. Because it is not realistic to always control these environmental factors, a further iteration of this node would identify contours and then average out the colors within the contours to identify whole objects that aren't dependent on pre-specified ranges.

Text recognition accuracy

The accuracy of the text detection algorithm used depends on many extraneous factors including lighting conditions, handwriting used, and boldness of characters. However, when used with the environmental configuration described above, text was successfully identified in approximately 90% of trials. Offline options for improving upon this are limited in scope, but possibilities

include training our handwriting detection algorithm separately or using an online API provided by Google's Vision library.

Movement planning

Movement planning was generally completed successfully, but suffered from calibration errors and some lack of flexibility. Pick-and-place tasks were easily completed using hard-coded block locations, and no paths caused collisions with the work surface. However, there remained some error (+- 3 cm) in the transformation from the block's location in a camera image to Baxter's coordinate frame; this rendered the system incapable of reliably grasping the block even when motion was otherwise executed successfully. Secondly, the system remains inflexible to unusual path planning challenges. When presented with an impossible plan or a system error that temporarily prevents path planning, the system simply skipped that step and proceeded through the remainder of the pick-and-place motion. It would be preferable to include more error-checking throughout planning and execution, such as making multiple attempts at a failed motion plan, and printing an informative error and returning to ready position if problems persist. Both this systemic robustness and block-picking precision could be improved through further time input to refine the code of *BlockIdentifier* and *PickPlace*.

Conclusions and future work

Combining human robot collaboration with the pick and place task requires many considerations, such as safety, intuitive user interfaces, and flexible programming. This project mostly focused on including an intuitive user interface, by having the collaborator write a desired block color at the target location, which the robot can identify and place all of the blocks of those colors at that location. As this sort of user interface may have applications in harvesting fruit or sorting hazardous waste, future work should look into including more identifying features of the objects to sort on. Examples of these could be the texture of the fruit, the hardness of the material, or the weight of the waste. The user interface would have to be extended to also accommodate the delineation of these extra features, which could create a multimodal sensory input. In industrial settings, it might be more appropriate to incorporate gesturing commands such as pointing to a specific location and telling it a string of description words for the types of objects that would be placed there. Additionally, there would need to be more rigorous studies on the robustness and safety of such a system. Although this paper did not focus on safety, the way the human collaborator interacts with the robot intrinsically affects their safety, and future work would need to be done to identify potential hazards and pitfalls of such an interface, so that it can be accounted for in the programming of the robot. This project was a small step in the direction of human robot collaboration, opening many other opportunities down the road to rethinking the way humans work alongside robots.

Literature Cited

Billingsley, Billingsley, J., & Taylor & Francis. (2019). *Robotics and automation for improving agriculture (Burleigh Dodds series in agricultural science)*. London, UK: Burleigh Dodds Science Publishing.

IFR. (2018). *Global industrial robot sales doubled over the past five years*. Retrieved December 16, 2019, from https://ifr.org/ifr-press-releases/news/global-industrial-robot-sales-doubled-over-the-past-five-years.

Miranda, J., Akhmametyeva, S., Nazon, Y., Choudhary, T. (2015). *Baxter Builder Project*. GitHub repository, https://github.com/opti545/baxter_builder.

Quigley, M., Gerkey, B., Smart, W. (2016). *Programming Robots With Ros*. Shroff Publishers & Distr.

Villani, V., Pini, F., Leali, F., & Secchi, C. (2018). *Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications*. Mechatronics, 55, 248–266. doi: https://doi.org/10.1016/j.mechatronics.2018.02.009

OSHA. Industrial Robots and Robot System Safety. (n.d.). Retrieved December 16, 2019, from https://www.osha.gov/dts/osta/otm/otm_iv/otm_iv_4.html.

Supplementary Materials

Service Setup

We created our own *ObjectLocation* service, which is included in the prototype_scripts folder of the code repository. This service takes in a string representing the target color as input, and outputs the x, y position of the block, as well as a boolean for the object's status, and an integer for the object's color (where the integers are defined beforehand for each color). Unlike the service used in the project that was created by Miranda et. al's (2015) Baxter project at Northwestern University, this service takes in the target color, so *BlockIdentifier* doesn't have to subscribe to the *target_color* topic, and *PickPlace* could fully control the input and timing of the service calls.

Launch Files

We also created our own launch files, one for the preliminary setup of Baxter's package dependencies and camera setup, and another one for the demo to start Baxter's pick and place task. These files are also included in the prototype_scripts folder of the code repository.