Deep Pose Estimation, Path Planning, and Inverse Kinematics for Visual Upright Pick-and-place of the YCB Soup Can

William Chen
Dept. of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA
verityw@mit.edu

Alex Cuellar
Dept. of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA
alexcuel@mit.edu

***Abstract*—Modern robot platforms must integrate many subfields in order to robustly perform manipulation tasks. Systems designed to perform upright pick-and-place must be able to perceive and localize an object of interest, find feasible paths to grab and place the object in a desired pose, compute control commands to follow said trajectories, and finally execute the calculated joint angles. We thus investigate and implement several components of this manipulation framework in simulation. Starting with perception subsystems, we test two pose estimation networks – a purely visual feed-forward approach called the pose interpreter network and the Mask R-CNN and ICP approach, which connects visual and geometric methods. Subsequently, we design a system to grasp the YCB soup can and place it upright on a table, formulating the path-planning and inverse kinematics problem as interpolation and non-linear optimization respectively. Finally, we integrate the deep pose estimators with the upright pick-and-place algorithm, thus exploring all these systems' performance and flaws in the visual upright pick-and-place task.**

***Index Terms*—pose estimation, robot manipulation, nonlinear optimization, pick-and-place, deep convolutional neural networks**

I. Introduction

As the applications of robots expand to more diverse and complex realms, robots require systems to intelligently make conclusions about the surrounding environment. Object pose estimation remains one of the central strategies for informing robot task and motion planning. In this project, we focus on pose estimation in the context of bin-picking, a situation in which a robot is presented with a bin of various objects which must be removed from the bin and placed elsewhere. Bin-picking allows us to explore a context analogous to many real-life applications such as clearing a dishwasher while working within a simple and well-defined setting. To create an even more constrained and well-defined environment, we decided to only focus on recognizing and interacting with one object: the soup can from the YCB dataset.

In this paper, we implement and analyze two methods of pose estimation: the recreation of the pose interpreter architecture [1] and the Iterative Closest Point algorithm with MaskCNN for segmentation. For analysis of these methods, we both use direct metrics of pose accuracy presented in [2] and the more use-oriented metric of success rate in picking up the soup can from a bin. In order to test this pick-and-place success rate, we use a PyDrake simulation of the iiwa robot arm to reach, grab, and set down the can using interpolation between key-points and inverse kinematics via non-linear optimization.

II. Related Works

This project's primary purpose is to explore existing methods for object pose estimation and manipulation. Therefore, we draw heavily upon past work in these fields, both to replicate their results and to apply them to our specific problem. Several of the key papers we considered are presented below.

A. Mask R-CNN, ICP, and Pointcloud-based Pose Estimation

[3] describes Mask R-CNN, which is the most widely used instance segmentation mask algorithm. However, other similar mask-focused CNNs have influenced the field, such as [4] which introduced U-Net, a network that achieves semantic segmentation rather than instance segmentation. Since we only focus on one object, U-Net could have been applicable to our specific problem statement. However, in the world of bin-picking where duplicate items are possible, instance level mask segmentation is the more natural choice.

[5] introduced the ICP algorithm, the standard pointcloud based pose estimation system. However, non-leaning based point-cloud pose estimation often attempts to augment the basic algorithm. [6] for example proposes a version of ICP which uses RGB color data to achieve more accurate pose estimations. [4] uses features based on surface creature to attain more accurate correspondences between points in the ground truth pointcloud and the observed pointcloud.

Lastly, deep learning based pose estimators utilizing point-cloud data has been studied by several groups. Most closely related to the project here, [7] introduced a CNN architecture to directly estimate poses from RBGD images rather than infer segmentation from an RBG image and use non-learning based methods to estimate pose from depth data.

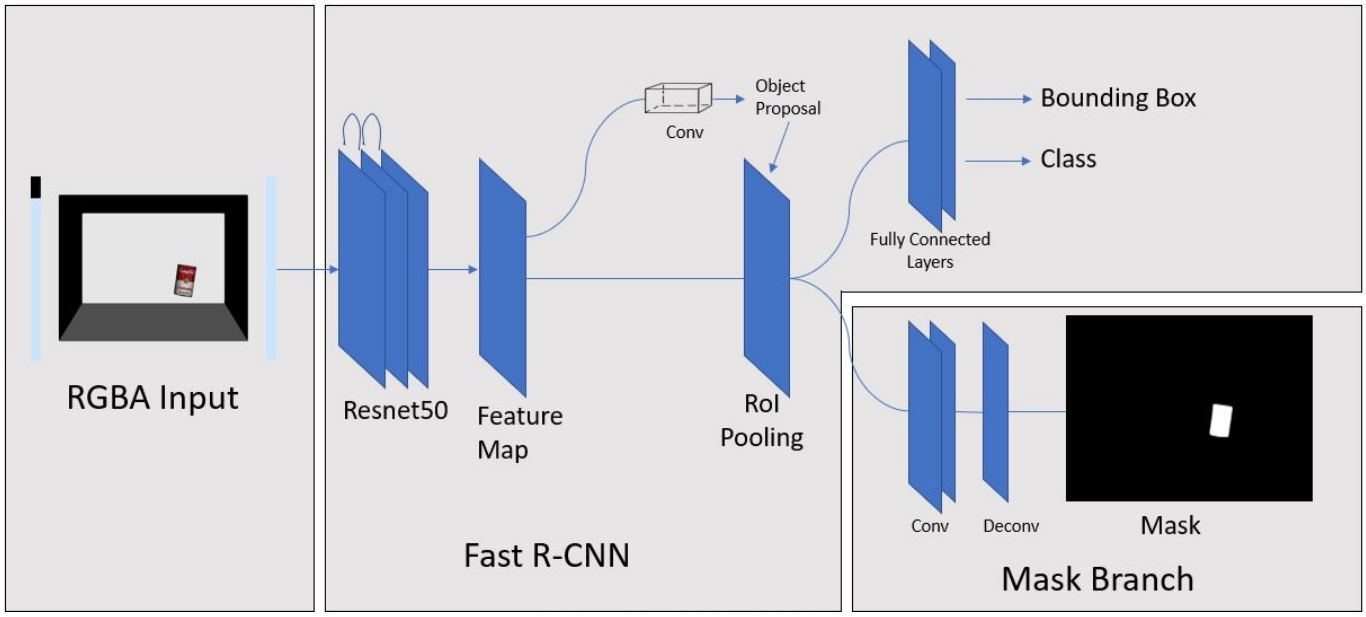

Fig. 1. Architecture for the Mask R-CNN implementation for PyTorch. The Archetecture is largely based on Faster R-CNN with a new RoI head that predicts a mask.

B. Pose Interpreter Networks

[1] presents an alternative to the neural network architectures discussed above. The author presents two networks – one that takes in RGB data and outputs semantic segmentation masks for the objects in the image and one that takes in these masks and outputs 6 degree-of-freedom (DoF) poses for each one. The latter network extends the ResNet18 architecture with fully-connected layers to estimate positions and (quaternion) rotations [8].

Three loss metrics are also provided to compare the predicted poses with the ground truth. The first is L_1 distance. The second, which was also used in PoseCNN, is a weighted combination of L_1 distance for position and negative cosine loss for the quaternion rotation. Finally, novel to this paper is the point cloud L_1 loss, which is the sum L_1 distance between corresponding points in two point clouds of the object, one transformed according to the predicted pose and the other according to the ground truth. This metric is similar to the metric used for ICP convergence, although it uses L_1 distance instead of Euclidean.

The paper shows that such a network can generate accurate real-time pose estimates of objects in real-world video feeds, even when the networks are trained on synthetic data. For these reasons, we largely replicated this pose estimation network for this project, training it on our own synthetic dataset of YCB objects. The specifics of this network will be discussed in the Methodology and Results section.

C. Pick-and-place

[9] provides a full framework for pick-and-place systems, presenting object localization, grasping, and motion planning techniques that can be used on objects being moved on a conveyor belt. For grasping, the authors describe the basic pre-grasp, grasp, post-grasp sequence. However, their methods differ from the ones presented below in that they generate the grasp poses manually – for each object type, the system records a family of user-picked grasp poses, which the robot then picks from at runtime. This affords some degree of flexibility when it comes to grasping, though not as much as the continuous range of possible solutions when generating grasp poses via a constrained optimization problem.

III. Methodology and Results

A. Data Generation

We adapted the data generation code from the online class textbook for this project. The object of interest, the YCB soup can, was given a random pose above the empty simulated bin. Then, we ran the simulator physics for 1 second so that the can could fall and land in a physically plausible pose. Finally, we retrieved an RGBA from the virtual camera and extracted the can's ground truth final pose as an input/output pair for our neural networks.

The strategy of data generation for Mask R-CNN is largely similar. However, despite only working with the soup can, we trained a network on both the soup can and the potted meat can, generating up to three objects in order to make the scene more interesting for image detection. Otherwise, the process of simulation was identical – generating the objects at a random pose above the bin and allowing the simulation to continue for 1 second.

B. Mask R-CNN and ICP Approach

Our first method for pose estimation uses the basic ICP algorithm in tandem with Mask R-CNN as segmentation of the

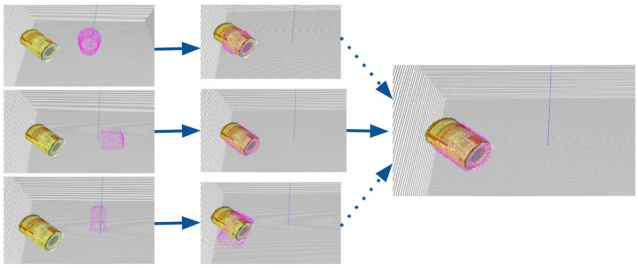

Fig. 2. Process for choosing final pose from ICP. We start with 3 initial guesses aligned with the world frame axes and extract a prediction. We then choose the pose with the minimum RMSE correspondence distances between the known pointcloud and the segmented depth image.

pointcloud. First, we pass an RGBD image of the simulated soup can through the Mask R-CNN architecture shown in Fig 1 to segment the can from the background. We then send a 3D point-cloud of the filtered depth image to the ICP algorithm and match to a known point-cloud of the soup can.

We utilized the PyTroch implementation of Mask R-CNN in order to train our network. This implementation uses a ResNet50-FPM backbone with two RoI heads – one defining bounding boxes around detected objects and one to create the mask. The head defining bounding boxes includes fully connected layers , while the mask head uses four Conv2D layers and a deconvolutional layer to produce the mask.

The output and loss function PyTorch uses for the classification/bounding box head is identical to that from the Faster R-CNN [10] architecture (the architecture which functions as a base for Mask R-CNN). The classification outputs a probability distribution $p = (p_0, ...p_K)$ where K is the number of catagories (in our case 2), and p_0 is the background. p is computed via softmax and uses the log loss function:

$$L_{cls}(p, u) = -log(p_u) \tag{1}$$

where u is the true class. The bounding box outputs a tuple $t^k = (t^k_x, t^k_y, t^k_w, t^k_h)$ for each class K. The loss function applied to the bounding box is:

$$L_{loc}(t^u, v) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L_1}(t^u_i - v_i) \tag{2}$$

where $v = (v_x, v_y, v_w, v_h)$ is the true bounding box tuple and

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & |x| \leq 1 \\ |x| - .5 & o.w. \end{cases} \tag{3}$$

is the computed loss.

The output and loss of the mask is identical to that form [3], with a $K \times h \times w$ dimensional sigmoid output representing K masks of size $w \times h$. The loss function L_{mask} is defined only over positive RoIs with a target being the intersection of the RoI and ground-truth. This loss strategy allows the network to generate masks for every class without competition and decouples mask and class prediction.

The network's overall loss is then simply the sum of these 3 individual loss functions:

$$L = L_{cls} + L_{loc} + L_{mask} \tag{4}$$

The implementation of the Mask R-CNN for PyTorch is pre-trained on the COCO dataset. In order to train our own dataset, we keep the pre-trained ResNet50-FPM backbone, but attach new heads to the backbone with the correct number of classes for outputs.

The ICP algorithm we use is identical to the one described in [11]. The algorithm iteratively improves the match between our segmented point-cloud of the soup can and the can's known point-cloud by repeating 2 steps. We start with an initial guess of the object pose, and find the closest point to the segmented pointcloud. Then, we update the pose of the ground-truth pointcloud to minimize the least-square distances between each point and the corresponding point in our segmented ponitcloud. By iteratively choosing minimum distance correspondences in object points and choosing the best pose, our guess theoretically converges on the correct object pose.

C. Mask R-CNN and ICP Results

For the Mask R-CNN, we train on 1000 RGBA images using the data generation methodology in *3.A* for 10 epochs. After the 10 epochs, the total loss is .064 and the loss incurred by the mask is .047. With the resulting model, we find that the best cutoff to filter the mask's sigmoid-defined pixels is .6.

When performing ICP on the segmented image, we found that the performance was largely dependent on the initial guess. Originally, We used an initial guess at the origin with no rotation. However, while the translation was relatively accurate from this initial guess, the rotation would frequently fall into local minima, resulting in estimated poses perpendicular to the real pose.

To mitigate this shortfall, we run the ICP algorithm three times with the same translation and different rotations: no rotation, a $\frac{\pi}{2}$ rotation about the x axis, and a $\frac{\pi}{2}$ rotation about the y axis. Given the symmetries of the soup can, these 3 rotations cover all of the major right-angle realizations of the point-cloud. We then choose the final proposed pose by taking the resulting ICP estimation with the minimum RMSE of inlier correspondence distances as shown in Fig 2.

Note that for other objects, we would likely need to consider more initial rotations in order to lower the likelihood of ending in a local minima. For example, with the soup can, we can ignore the differences in the shape of the pointcloud for any rotation of π radians along any axis.

D. Pose Interpreter Network Approach

We implement a pose interpreter network similar to the one presented in [1]. Its architecture consists of a ResNet18 sequence followed by a multilayer perceptron. The 1000-class output of ResNet is connected to a 256-node fully connected layer, which then connects to two parallel fully connected layers, thus outputting the position and quaternion rotation of

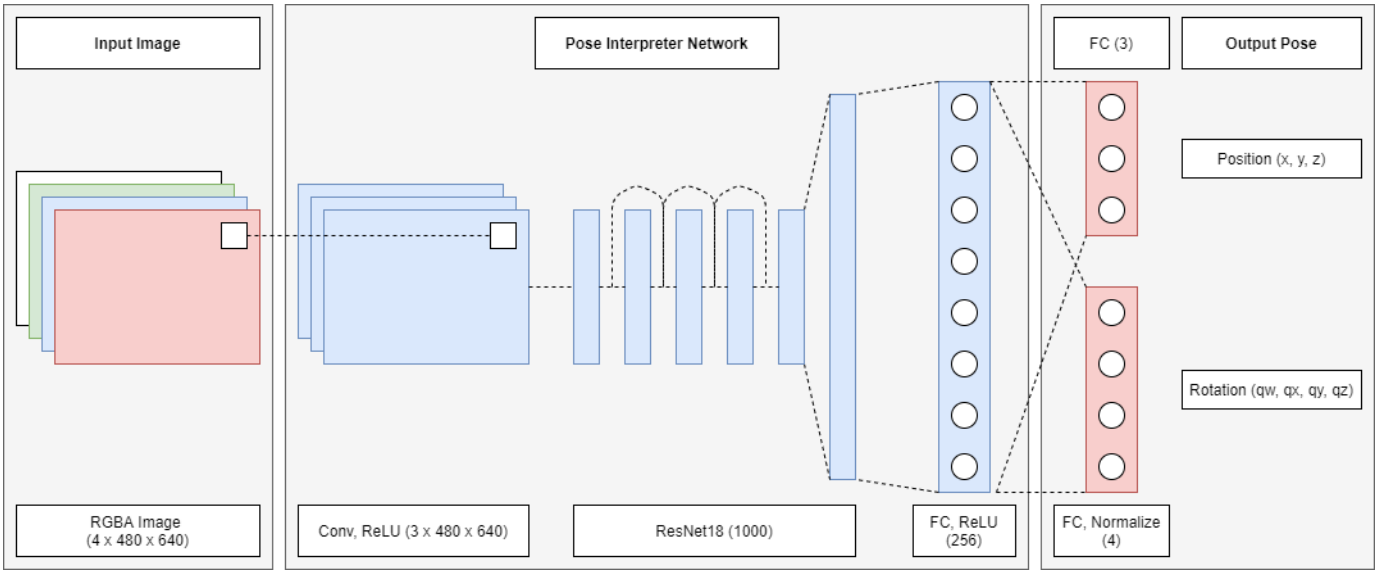

Fig. 3. Architecture for the pose interpreter neural network. The first convolutional layer (with three kernels of size 1×1 and stride 1) is *not* used when a mask is inputted – ResNet18 expects a 3-channel image, so this initial layer serves to lower the depth of the 4-channel RGBA input.

the considered object. As the quaternion represents a rotation, it passes through an additional normalization layer that makes it unit length. See Fig. 3.

We used the aforementioned PoseCNN loss as our distance metric:

$$L_{\text{PoseCNN}} = |\hat{\mathbf{p}} - \mathbf{p}| + \alpha(1 - \langle \hat{\mathbf{q}}, \mathbf{q} \rangle) \tag{5}$$

Where $\hat{\mathbf{p}}$, $\hat{\mathbf{q}}$, $\mathbf{p}$, and $\mathbf{q}$ are the ground truth position and rotation (as a quaternion) and predicted position and rotation respectively and α is a positive weight parameter (we initially set $\alpha = 1$). Moreover, $|\hat{\mathbf{p}} - \mathbf{p}|$ is the L_1 distance between $\hat{\mathbf{p}}$ and $\mathbf{p}$ and $\langle \hat{\mathbf{q}}, \mathbf{q} \rangle$ is the inner product of the two quaternions, expressed as 4-dimensional vectors. We can interpret the $1 - \langle \hat{\mathbf{q}}, \mathbf{q} \rangle$ term as effectively the cosine loss between the two rotations – since the quaternions are of unit length, their inner product is the cosine of the angle between them.

As quaternions $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation, we followed the convention in [1], which always chooses the quaternion with the *positive* real component, $\mathbf{q}_w$. Said paper introduced an additional loss term that drives the network towards predicting positive values of $\mathbf{q}_w$, which we adopted for our loss function as well. Our final loss function is thus:

$$L'_{\text{PoseCNN}} = |\hat{\mathbf{p}} - \mathbf{p}| + \alpha(1 - \langle \hat{\mathbf{q}}, \mathbf{q} \rangle) + \max(-\mathbf{q}_w, 0) \tag{6}$$

This objective is optimized with PyTorch's stochastic gradient descent (SGD) optimizer with momentum and learning rate values of 0.9 and 0.001 respectively. Note that since the RGBA input has 4 channels, we initially pass the image through a convolutional layer that reduces the number of channels to 3, but keeps the other dimensions' sizes the same.

E. Pose Interpreter Network Results

We initially train with 10000 training images and 2000 validation images of the can in random poses. This process yields poor performance and takes much training time, so we implement several modifications.

First, we change the data generation pipeline to spawn the soup can only with random yaws, just like it would in the upright pick-and-place code. Moreover, we switch to using PyTorch's Adam optimizer [12]. There is also a significant difference in magnitude between the position and rotation losses (the L_1 position loss is on the scale of $1e-2$ to $1e-1$ while the rotation loss is $\in [0, 1]$), so we reduce α to be of the same magnitude as the position loss, thus allowing the parameter updates to more equally depend on both losses. Finally, given that we are only working with the YCB soup can, we reduce the number of training/validation images to 4000 and 1000 respectively.

As expected, this fix lowers the training time for 5 epochs from approximately 1 hour to less than 30 minutes, while also lowering the error significantly. However, as we will discuss in III-I, this was ultimately often not enough accuracy for the integration pipeline, so we implement additional modifications to the network architecture, which we describe in the aforementioned subsection.

F. Upright Pick-and-place Approach

To achieve upright pick-and-place of the YCB soup can, we adapted the problem set code for opening a cabinet door [11]. The arm follows a simple trajectory defined by several hard-coded key frames. Starting in the default pose, the arm moves to an auxiliary pose above the initial one, then to the pre-grasp pose above the bin. From there, it moves to the grasp pose, grabs the object, and moves back up to the pre-grasp pose once again (we designate this the post-grasp pose). Finally, it moves to the table, releases the object, and moves to the post-place pose. Pictures of these poses can be seen in Fig. 4.

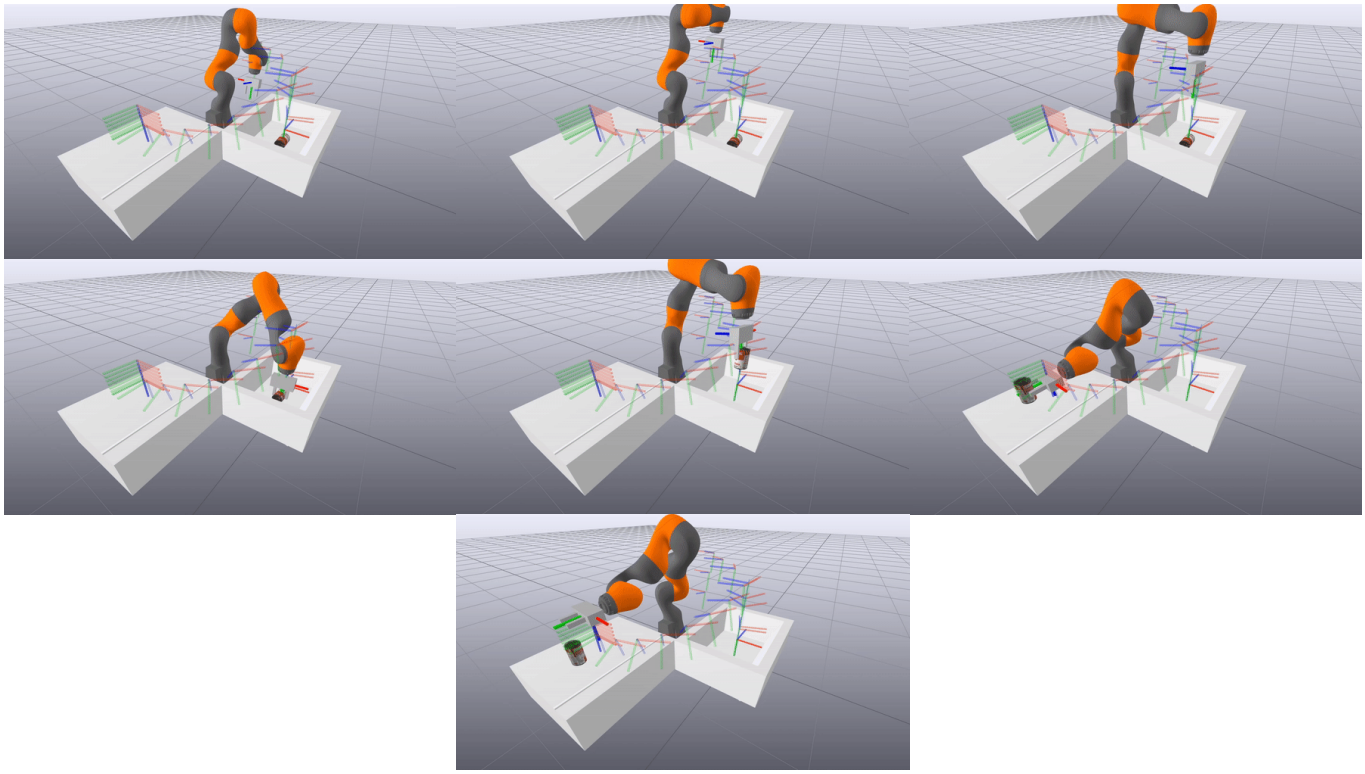

Fig. 4. End effector key frames for upright pick-and-place of the YCB soup can. In order, we have the initial pose, the auxiliary pose, the pre-grasp pose, the grasp pose, the post-grasp pose, the placement pose, and the post-placement pose.

The grasp pose is determined by the soup can's pose. As with the door handle case from the corresponding problem set, in order to grab the can, the end effector frame's z-axis must align with the length of the can. If the can stays aligned this way, then placing the can upright thus requires the placement pose have the effector's z-axis pointing directly downwards, such that the bottom of the can points towards the table when it is released.

Notably, unlike with the door handle, the soup can's default reference frame does not have its z-axis pointed along its length. Rather, the can's y-axis points in this desired direction. If the pose of the can in the world frame is known, then the y-axis is simply the second column of rotation matrix ${}^{W}R^{\text{Can}}$. We shall refer to the first, second, or third columns of a rotation matrix ${}^{A}R^{B}$ as ${}^{A}\mathbf{r}_{x}^{B}$, ${}^{A}\mathbf{r}_{y}^{B}$, and ${}^{A}\mathbf{r}_{z}^{B}$ respectively, as they represent the x, y, and z-axes of the corresponding reference frame. Since we want ${}^{W}\mathbf{r}_{y}^{\text{Can}}$ to be the same as the grasp pose's z-axis, we have that:

$$ {}^{W}\mathbf{r}_{z}^{\text{Grasp}} = {}^{W}\mathbf{r}_{y}^{\text{Can}} \tag{7} $$

We also want the end effector to point downwards when in the grasp pose – that is, its fingers (which are aligned with the y-axis in its own frame) should be pointing in the negative z-axis direction in the world frame. Therefore, we have that:

$$ {}^{W}\mathbf{r}_{y}^{\text{Grasp}} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{8} $$

With two of the three axes for the grasp rotation defined, we compute the final axes by taking the cross product of the known two:

$$ {}^{W}\mathbf{r}_{x}^{\text{Grasp}} = {}^{W}\mathbf{r}_{y}^{\text{Grasp}} \times {}^{W}\mathbf{r}_{z}^{\text{Grasp}} \tag{9} $$

Now, all three columns of the grasp pose rotation matrix are known.

To compute the position of the grasp frame, if we are given that the can's frame's position is at ${}^{W}\mathbf{p}^{\text{Can}} = [x, y, z]^{T}$, then we simply set the grasp pose position to ${}^{W}\mathbf{p}^{\text{Grasp}} = [x, y, z + \epsilon]^{T}$ for some small positive value of ϵ (we used $\epsilon = 0.09$). This allows the can to be slightly below the end effector, directly between the fingers. With this, the grasp pose is fully defined based on values of the can's pose.

Once all the key frames are determined, we interpolate between them using first-order hold (FOH) for positions and quaternion spherical linear interpolation (slerp) for the rotations, thus allowing us to determine the desired end effector pose at any point in continuous time. Subsequently, we computed the interpolated pose at 30 uniform points in time along the full duration of the pick-and-place task.

Finally, we formulate nonlinear optimization problems to solve the inverse kinematics for each of these computed poses. The loss function we try to minimize is simply the squared L_2 distance from the variable joint angles to the nominal ones. That is,

$$ L = ||\mathbf{a}_{\text{nominal}} - \mathbf{a}||^{2} \tag{10} $$

Where $\mathbf{a}_{\text{nominal}} = [0, 0.6, 0, -1.75, 0, 1, 0, 0, 0]^T$ is a vector of the default nominal joint angles for the arm presented in [11] and $\mathbf{a}$ is a vector of the joint angle variables whose values are being computed by the nonlinear optimizer. Both these vectors are 9-dimensional, with the first seven being arm joint angles (which are used in the forward kinematics to determine the end effector pose) and the last two being gripper joint angles.

To ensure that the end effector pose was sufficiently close to the desired ones, we imposed relaxed constraints on the position and rotation of the effector, widening the range of viable solution joint angles. Formally, the full nonlinear optimization can be written as:

$$\begin{aligned} \text{minimize: } & L = ||\mathbf{a}_{\text{nominal}} - \mathbf{a}||^2 \\ \text{subject to: } & \mathbf{p}(\mathbf{a}) \leq \mathbf{p}_{\text{desired}} + \Delta\mathbf{p} \\ & \mathbf{p}(\mathbf{a}) \geq \mathbf{p}_{\text{desired}} - \Delta\mathbf{p} \\ & \Theta(\mathbf{q}(\mathbf{a}), \mathbf{q}_{\text{desired}}) \leq \Delta\theta \end{aligned} \tag{11}$$

Where $\mathbf{p}(\mathbf{a})$ and $\mathbf{q}(\mathbf{a})$ are the forward kinematics position and quaternion rotation of the end effector (as functions of joint angles) respectively and $\mathbf{p}_{\text{desired}}$ and $\mathbf{q}_{\text{desired}}$ are the desired end effector position and quaternion rotation respectively. $\Theta(\cdot, \cdot)$ is a function of two quaternion rotations, outputting the positive angle of the relative rotation transforming the first rotation to the other. Note that this is the same as the magnitude of the Lie algebra vector corresponding to said relative rotation, computed through the logarithm mapping from rotation matrices to vectors, as we mention later in the Discussions section [2]. Finally, $\Delta\mathbf{p}$ and $\Delta\theta$ are the (positive) tolerances for position and rotation. The former is 3-dimensional, forming a bounding box in space centered at the desired position, whereas the latter is a scalar.

We use Drake's default SNOPT optimizer to solve the posed constrained optimizations. For each optimization, we also set the initial angle guess to the solution to the previous problem. Once viable solutions have been found for all the poses, the robot executes the computed joint angles, picking up the soup can and placing it upright on the table.

G. Upright Pick-and-place Results

Initial tests involve the soup can in its default pose at the center of the bin, with the grasp key frame being hard-coded in. The simulated arm successfully completes the upright pick-and-place task. Notably, as the contact forces for the end effector are modeled as being applied by a single point on each finger, the soup can is free to swing when the rotation of the end effector changes. Nevertheless, this did not stop the arm from completing its task.

In subsequent experiments, we allow the can to start at random positions and rotations. However, to simplify the problem, we restrict its initial rotation to purely being a random yaw – that way, the soup can will still be lying on its side, so the grasp pose generation method discussed prior would still work. However, in running these new tests, several problems arise.

First, the can often spawns in close proximity to a wall of the bin, causing the end effector or a link of the arm to collide with said wall when attempting to reach the grasp pose. To prevent this, the viable spawn zone is made smaller while remaining at the center of the bin, making it impossible for the can to spawn prohibitively close to the wall.

Second, whereas in the initial test the pre-grasp pose is directly above the can, when the position of the can is varied, this is no longer the case. The end effector's fingers thus often collides with the can while trying to reach the grasp pose, knocking it away from its expected position.

We therefore also allow the pre-grasp pose to vary with can position and orientation – it is computed in the same way as the grasp pose, but with the translation's z-value set to 0.3 (i.e. directly above the grasp pose, at the same height as the initial, hard-coded pre-grasp pose). Thus, when moving from the pre-grasp pose to the grasp pose, the end effector moves straight down, limiting any unwanted contact between the fingers and the can prior to grasping. However, the "post-grab pose" (which the end effector moves to after successfully gripping the can) is the same as the initial hard-coded pre-grasp pose, as the trajectory to the table placement pose assumes that as the starting pose. Still, changing the pre-grasp pose to be directly above the grasp pose results in an increase in robustness.

Finally, by default, contact forces with the end effector fingers are approximated with two spherical contacts at the finger tips. When grabbing the soup can, the two spherical contacts act as a hinge; there is no rotational friction providing torque on the can, meaning that it swing about freely as the end effector's pose changes. Therefore, sometimes, the can will swing such that it is not upright upon reaching the table placement pose. In some test runs, the can is slanted (rather than being aligned with the z-axis), so when it is released onto the table, it falls on its side. One can add more contact points to the finger contact model to address this issue, but that is beyond the scope of this paper.

H. Integration Pipeline Approach

We add a camera to the manipulation station simulation environment with the same camera parameters and pose as the camera used in the data generation code. Once the can is spawned into a random pose, we retrieve an RGBA image from the camera. This is fed into either the pose interpreter neural network or the Mask R-CNN and ICP network, which produces a pose estimate. Finally, we use this pose estimate in place of the ground truth can pose for upright pick-and-place, computing the key frames based off of it and subsequently running the rest of the upright pick-and-place code.

I. Integration Pipeline Results

The pipeline above is tested in 10 randomly generated can upright pick-and-place scenarios. Performance metrics and overall success rates of the upright pick-and-place task are presented in Table I for each of the two tested networks. Specifically, the performance metrics we used were L_{dg} and L_{c}'s mean value and standard deviation over the ten trials. The equations for said metrics are presented below, in IV-C.

TABLE I
AVERAGE LOSS FOR POSE ESTIMATION METHODS OVER 10 TRIALS

Method	L_{dlg} Mean	L_c Mean	L_{dlg} Std Dev	L_c Std Dev	Success Rate
Pose Interpreter	0.35	0.47	0.51	0.67	0.7
Mask R-CNN + ICP	2.02	2.29	0.80	0.40	0.1

While using the original pose interpreter network, predicting 6D pose results in significant inaccuracy, especially in terms of the orientation. Thus, since we are spawning all cans in the bin with just a yaw, we modify the network slightly to have it output position and the 1D yaw of the can, changing the final fully-connected and normalization layers to a new fully-connected layer, outputting yaw values $\phi \in [0, 2\pi)$. In training this modified network, we thus change the loss of the network to:

$$L_{\text{new}} = |\hat{\mathbf{p}} - \mathbf{p}| + \alpha|\hat{\phi} - \phi| \tag{12}$$

where $|\hat{\phi} - \phi|$ is the minimum positive difference of yaw angle between the actual soup can and the predicted yaw. We reduce the training dataset size to 4000 images with 200 validation images. Otherwise, all other parameters are the same as when we trained the initial network.

After introducing this new network architecture, the pipeline succeeds in grasping and placing the can without the full ground truth pose, instead relying on the estimated can pose. Qualitatively, the system succeeds in the pick-and-place task, but frequently fails to place the can upright. The baseline upright pick-and-place system already occasionally fails the task (as discussed earlier), but the slight inaccuracies of the pose estimation network likely compound these problems, making the failure rate significantly higher. Nevertheless, the system is relatively consistent in being able to grasp and place the can purely through camera sensor data and outperforms the Mask R-CNN and ICP pipeline, both in most performance metrics and in the visual upright pick-and-place success rate.

J. Data Analysis Next Steps

As stated in the Results sections for Mask R-CNN + ICP, the loss incurred by the Mask R-CNN + ICP system is largely due to geometric symmetry in the soup can allowing pose estimation to be incorrect while the point-clouds generally align. We also suspect that success rate for the bin picking with the Mask R-CNN + ICP system is low because pointcloud alignment doesn't necessarily mean all axes are aligned.

If given more time, more trials with the integration pipeline would ideally be conducted and more insightful performance metrics measured. For example, it would be good to see how the two networks' position and orientation estimation abilities compared, and how each of those two affected the overall upright pick-and-place performance (i.e. if the upright pick-and-place system's performance were more sensitive/robust to positional or orientational estimation error). Moreover, we would have been curious to implement other grasp candidate algorithms directly dependant on the pointcloud (such as antipodal points) to see if such methods would work better with the axes-misaligned estimates of Mask R-CNN + ICP.

IV. DISCUSSION

A. Data Generation Discussion and Next Steps

While data generation is not central to our goal, the limitations of our project appear in the data our algorithms work with. The clearest example is with our pose interpreter, which relies on data from one YCB object and only varies the pose with respect to the yaw angle.

Since we use PyTorch's standard architecture for Mask R-CNN, we can be confident in the ability for the network to properly recognize various poses for many objects despite our limitations of one soup can. However, the possibility of more objects, overlap, and obstructions providing less clean data to ICP could severely limit it's effectiveness in pointcloud matching even if we assume perfect segmentation from Mask R-CNN.

B. Mask R-CNN and ICP Discussion and Next Steps

The positive results we found from Mask R-CNN are very unsurprising given we used a very standard package to train the network. We do we see that both general loss and loss for the mask specifically drop very quickly within 3 epochs and do not drop much farther for the following 7. This indicates that work in the future may involve gaining a better understanding of the relationship between the complexity of the data given to the network (especially with respect to the object variety) and the amount of training required both with respect to size of dataset and number of epochs trained.

Another extension of the Mask R-CNN could to help improve the versatility of our Pose Interpreter system. Specifically, we wanted to use the mask of an image along with the RGBA data as input to the pose interpreter. While it is very unclear whether this pipeline would improve results, it is easy to imagine how this extra piece of data could at least help the network learn the translation of objects in space.

As for ICP, our choice of the soup as the object of interest certainly limited the potential accuracy of our pose estimation. The soup can's inherent symmetry means that loss functions comparing a ground truth pose with an estimated pose do not well correlate with how successfully the two pointclouds aligned. In fact, while the losses for pose estimation with the Mask R-CNN + ICP system were very high, visual investigation of the alignment shows that the general shapes of the soup can cylinders generally match up. Given more time to integrate various objects into the pick and place pipeline, we are curious as to the improvement in loss metrics for our

Mask R-CNN + ICP pipeline given less symmetric objects. Additionally, we would have been curious to integrate ICP variants which take color data into account as a way to mitigate the symmetry problem of pose estimation loss for the soup can.

C. Pose Interpreter Network Discussion and Next Steps

Being similar to the network in [1], our modified pose interpreter network similarly yields good pose estimates of objects when trained on synthetic data, though testing and validating our network on real image feeds/in real time is beyond the scope of this paper. It succeeds for our purposes of localizing the soup can in the bin based on sensor data, which is what we hoped to explore in this project.

As a next step, one could explore different loss functions and their performance with the network. Beyond the ones presented in [1], another possible distance metrics for poses is the double geodesic distance in $SE(3)$, which is effectively the norm of a vector consisting of the relative displacement and rotational angle between two poses:

$$\begin{aligned} L_{\text{dg}} &= \sqrt{||\log((^{W}\hat{R}^{\text{Can}})^{TW}R^{\text{Can}})^{\vee}||^2 + ||\hat{p}-p||^2} \\ &= \sqrt{||\hat{p}-p||^2 + ||\theta||^2} \end{aligned} \tag{13}$$

where $\log((^{W}\hat{R}^{\text{Can}})^{TW}R^{\text{Can}})^{\vee} = \theta$ is the logarithm map from the Lie group set of $SO(3)$ to the corresponding Lie algebra $so(3)$, and can be interpreted as the angle between the predicted and actual rotations. A similar distance metric is the chordal distance in $SE(3)$:

$$L_{\text{c}} = \sqrt{||\hat{p}-p||^2 + ||2\sqrt{1-\cos(\theta)}||^2} \tag{14}$$

Both these losses were presented in [2], and can be further modified with a weighing hyperparameter for the positional/rotational components and a $\max(-\mathbf{q}_w, 0)$ term, similar to the modified PoseCNN loss above. While no network was trained with these loss functions, they are used in Table I as network performance metrics.

D. Upright Pick-and-place Discussion and Next Steps

Ultimately, the upright pick-and-place system was successful at grasping and placing the soup can upright in simulation. Still, there are several possible avenues of further exploration.

First, as mentioned earlier, we can add additional contact points to the fingers for a more robust facsimile of real contact mechanics. This would address many of the problems with the can rotating to an undesirable orientation while being carried to the final table, as the effector would be able to more firmly grasp it. This is a problem both in the baseline upright pick-and-place problem and especially in the full integration pipeline (as we use the non-ground truth pose estimate), so this change would likely increase robustness greatly.

To simplify the problem, we currently place very heavy limitations on the position and rotation of the can in the bin. This allows us to make assumptions about the directions of its reference frame, which is crucial in determining the grasp pose parameters. The grasp pose computation can be generalized significantly to also allow the arm to grab the soup can when it is standing directly or leaning diagonally on a wall of the bin.

In such cases, one would likely need to consider the geometry of the bin in order to determine a physically feasible grasp pose for the arm– i.e. one that is not colliding with any walls or require unreasonable joint angles. The table placement pose likewise assumes that the length of the can is perpendicular to the grasp plane. Different key frames would be needed if, for example, the end effector grabs the can from the top (e.g. if the can itself is standing upright in the bin).

Other possible channels to expand this work include adding a sensor feedback control loop, as right now, the control sequence is open loop and will execute even if the initial grasping was unsuccessful. Moreover, one can also attempt to replicate this work with different/more objects in the bin. This would also add an element of planning to the problem, as if objects occlude or are on top of each other, then the system cannot always pick and place objects on the table in a set order, but must determine the logical order by itself. Finally, executing this system on an actual robot arm platform would also likely yield additional insights and challenges.

E. Integration Discussion and Next Steps

The most important improvement to the integration pipeline's performance would likely be to increase the contact points, as the torsion experienced by the can is definitely the most noticeable in the pipeline. Otherwise, as the integration pipeline connects the pose interpreter network and upright pick-and-place algorithm, the improvements to both those subsystems discussed prior would likewise improve the overall system's performance.

REFERENCES

[1] J. Wu, B. Zhou, R. Russell, V. Kee, S. Wagner, M. Hebert, A. Torralba, and D. M. Johnson, "Real-time object pose estimation with pose interpreter networks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[2] L. Carlone, "Lie Groups and Distances (Course notes for MIT 16.485)," downloaded on Nov. 27, 2020.

[3] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2018.

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[5] 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, 1992.

[6] L. Liang, "Precise iterative closest point algorithm for rgb-d data registration with noise and outliers," *Neurocomputing*, vol. 399, pp. 361–368, 2020.

[7] W. Chen, X. Jia, H. J. Chang, J. Duan, and A. Leonardis, "G2l-net: Global to local network for real-time 6d pose estimation with embedding vector features," 2020.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[9] A. Cowley, B. Cohen, W. Marshall, C. J. Taylor, and M. Likhachev, "Perception and motion planning for pick-and-place of dynamic objects," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 816–823.

[10] R. Girshick, "Fast r-cnn," 2015.

[11] R. Tedrake, "Robot Manipulation: Perception, Planning, and Control (Course Notes for MIT 6.881)." downloaded on Nov. 27, 2020 from http://manipulation.csail.mit.edu/http://manipulation.csail.mit.edu/.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.