HSR Grasping

Orlando Fraser

Summer 2020

1 Introduction

The aim of the project was to design an algorithm that will enable the HSR robot to execute grasps on fine objects such as pens or cutlery. This algorithm could then be used for a range of tasks that the robot needs to carry out. The novel contribution of this work is to use images from the RGB hand camera as input to the grasping algorithm rather than the main depth camera on the robot's head. The benefit of the hand camera is its ability to get closer to the desired objects and is less prone to obstructions to its field of view.

2 Background and Literature Review

the research area of robotic grasping has seen a range of work in recent years. Most approaches use depth images as input to the algorithm, using this to predict an optimal grasp of the object in the field of view. This grasp is defined by a vector that contains a position, an orientation, and a gripper width. The grasp is in 2D image space, so it then converted to 3D world space using a known set of transforms (refer to appendix for a description of the transforms). The grasp vector can then be used to find an error between the current position of the gripper/hand and the desired position (the grasp vector itself). This error can then be minimized in order to successfully execute the grasp.

RGB images could be used as input to a grasping algorithm and they would (in theory!) be able to

generate a grasp from the data. However depth data is required to convert this grasp from 2D image coordinates to 3D world coordinates. For this reason, you may as well use depth for the input and so, to our knowledge, there are no current methods that use RGB alone at this time.

We therefore can separate, the grasping algorithm into three distinct sections: depth estimation from RGB, grasp prediction from the depth map, and finally, the controller that will provide the necessary instructions to execute the grasp in a efficient manner. We elaborate on the design choices made for each of these sections below.

2.1 Depth Estimation

The hand camera is just RGB, however we need a depth map in order to generate a grasp. The problem of estimating depth from RGB images is an active research area called Monocular depth estimation. This enables a rough depth map to be produced from an RGB image, this depth map can then be used with the grasping algorithm.

The majority of current algorithms in this area use a CNN based approach trained with a dataset made up of RGB images along with the ground truth depth maps. We tested some popular algorithms but they produced very poor quality depth maps on our data. This was because most standard methods were geared towards wide angle outdoor scenes (eg for self driving cars) and therefore they didn't generalize well to our application, which consisted of more close up indoor images.

The recent approach from Ranftl et al [2] was designed to address the poor generalizabilty of previous methods by utilizing data from a range of datasets.

2.2 Grasp Prediction

The 2018 paper from Morrison et al. [1] detailed an algorithm that used convolutional neural networks to predict grasps (GG-CNN). It took took a 300x300 depth image as input and predicts a grasp for each pixel in this input, this output is called the grasp map. The grasp map consists of a quality, angle, and width map, each of the same dimension as the input. One can then find the optimal grasp by find the

Figure 1: The depth estimation algorithm tested on a range of different scenarios. Exhibits robust performance across the range of scales and lighting conditions . From the paper [2]

max point of the quality map. The estimated depth maps were 640x480 images, we therefore took a 300x300 crop from the center of the image to use as input to this stage.

2.3 Control

Both open loop and closed loop approaches can be used for the controller. In open loop, the grasp is calculated once and the error is minimized in a single step. Whereas in a closed loop approach, after the grasp is calculated, a small step is taken in a direction that minimizes the error, after which the grasp is

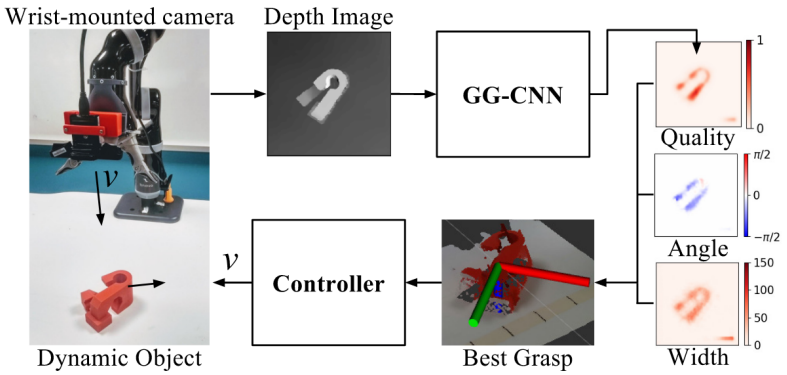

Figure 2: Pipeline showing the GG-CNN algorithm. From the paper [1].

recalculated from the new position and the process repeats until convergence.

The benefit of open loop systems is that the grasping algorithm only needs to be run once per grasp, therefore reducing the total amount of computation required. However, the lack of feedback means that they have no way of correcting the movement if an error arises in any part of the system. Closed loop systems have feedback which enables errors to be corrected quickly, leading to a much more robust algorithm. However they require an algorithm that can run at a speed of at least 30Hz or else the movement will be too slow to be viable.

Executing on the CPU of the computer we were using for the tests, the grasp prediction algorithm takes roughly 5 seconds while the depth prediction algorithm takes roughly 10 seconds. Unfortunately, this is unfeasibly slow for a closed loop controller so we were forced to use an open loop approach for grasping despite its lack of robustness. The HSR robot normally has access to a GPU so using this could reduce the runtime significantly.

3 Testing with the HSR simuation

The algorithm was tested with the HSR simulation on Gazebo. The HSR python API was used in conjunction with a Jupyter notebook to send and receive commands/data from the simulation. This enabled us to run the various python algorithms on the simulation in a simple way.

We first generated a basic environment that consisted of a low table and a generic cylindrical object that had similar dimensions to the objects one would be grasping in the real world. A point light source in the simulation was also required in order for the depth estimation to work. This was because without a light source there were no shadows, which were necessary to make the scene lifelike.

We set the initial position of the arm to be between 0.2-0.3m above the table with the hand pointing downwards so that the object was in frame.

We produced a visualization tool that showed the RGB image, the estimated depth map, and the quality map in a single image. This allows one to detect any errors that might arise in either the depth

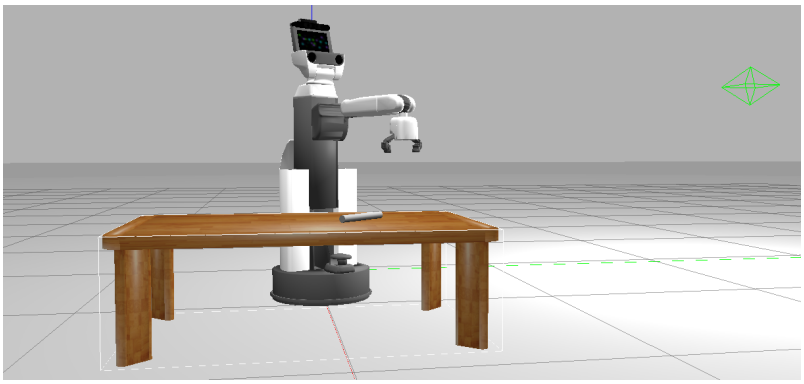

Figure 3: Simple Gazebo set up that was used to test the algorithm.

estimation or grasp prediction sections of the pipeline. The grasp vector is used to produce a grasp rectangle that is plotted on top images, this makes it very easy to see whether the predicted grasp is reasonable (similar to how a human would decide to pick up the object). An example output from this tool is displayed in Figure 4.

We tested a range of different control algorithms to execute the grasp once a grasp vector had been found. It seemed like the grasp prediction algorithm was most accurate when the object was near the center of the image. We therefore tried a two stage process where first the hand would be moved in the x-y plane only, to center the object. Then the grasp prediction would be run again on the new image and the grasp would be executed using this new grasp vector. Using this approach, the algorithm worked fairly robustly with the simulation and took approximately 30 seconds per grasp. One issue was that sometimes the edge of the table was mistaken for a potential grasp. We got around this issue by placing the object far enough into the table so that the table edge was not in the frame.

4 Conclusions and Future Work

Due to constraints imposed by the COVID-19 pandemic, we were unable to test the system on the real HSR robot. Therefore the main work to be done is to test the algorithm do these tests and make changes where necessary. Additionally, it would be a good idea to measure the speed increases that could be

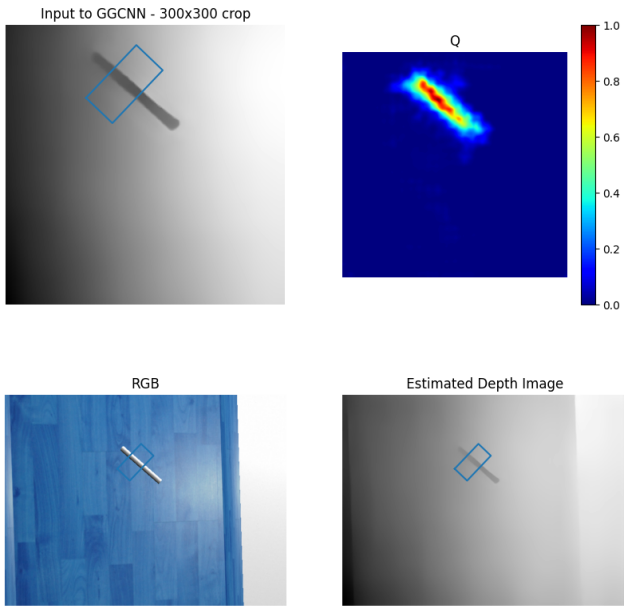

Figure 4: Visualization tool.

achieved by running the algorithm on a CPU rather than a GPU as this could make closed loop control approaches viable.

5 Appendix: Camera to World Frame Transform

The grasp prediction algorithm produces a grasp vector that consists of a position, an angle, and a width, all defined in the 2D image coordinate system of the camera. We can write this as $\mathbf{g} = [u, v, \theta, w]$.

We need the grasp in a 3D world coordinate system. We decided to use the palm of the hand coordinate system (hand palm link TF frame) so that we can move the robot hand in its own coordinate system to minimize the error. This 3D grasp can be written as $\mathbf{G} = [x, y, z, \theta_x, \theta_y, \theta_z, W]$.

The functional mapping $f : \mathbf{g} \longmapsto \mathbf{G}$ makes use of the intrinsic and extrinsic parameters for the camera as well as the depth of the object. The intrinsic parameters are the focal length $f = 205.47$ and principal point position $c_x = 240.5, c_y = 320.5$. We use these to convert from the 2D image frame to the

3D camera frame. Our algorithm can only estimate grasps perpendicular to the plane of the image and as a result we cannot determine an optimal angle of approach about the x or y axis'. We therefore set the angles about these axis' to zero. We use the subscript c to denote the camera frame.

$$x_c = \frac{u - c_x}{f} * depth \qquad y_c = \frac{v - c_y}{f} * depth \qquad z_c = depth$$

$$W = \frac{w}{f} * depth \qquad \theta_x = 0 \qquad \theta_y = 0 \qquad \theta_z = \theta$$

We then need to convert from the 3D camera frame to the hand palm frame using the rotation and translation parameters that define the extrinsic transformation. We use homogeneous coordinates so that the rotation and translation can be applied through a single matrix multiplication.

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -0.039 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.004 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix} \tag{1}$$

References

[1] Morrison, D., Corke, P. and Leitner, J., 2018. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. arXiv preprint arXiv:1804.05172.

[2] Lasinger, K., Ranftl, R., Schindler, K. and Koltun, V., 2019. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. arXiv preprint arXiv:1907.01341.