PICASSO

Beatrice Liang (bsl2127), Jake Kwon(jk3655), Tonye Brown(tb2553)

Project description

Our goal was to use the PR2 to produce drawings on an arbitrary plane within the reach of the right arm of a stationary PR2 robot. Our main challenge was to produce an image that is drawn in a convincing artistic style. Moreover, it must be generalizable to a wide variety of images. Since we used MoveIt! to manipulate the PR2 by inputting cartesian coordinates, we had to ensure that we could appropriately convert image points that are two dimensional to three dimensional points within the bounds of a given piece of paper that is placed within the reach of the stationary PR2. Finally, we wanted to improve the style of the image by segmenting the input image into different color channels and allowing the robot to draw using the three primary colors in the subtractive color model: cyan, magenta, and yellow.

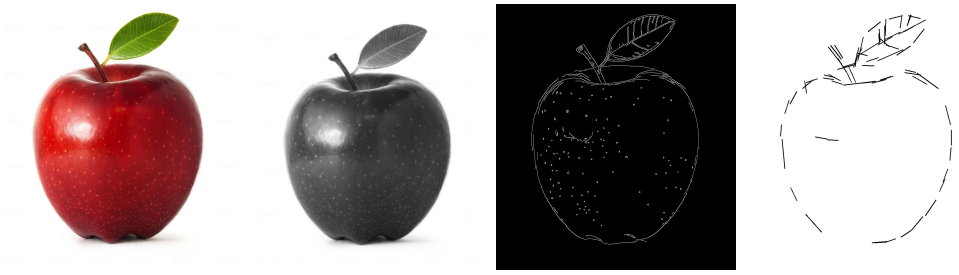
Implementation details

Extracting Lines from an Image

First, we must extract lines from an input image. To do so, we first convert the image to grayscale, perform Canny Edge Detection to extract edges from the image, and then transform those edges into lines using the Hough Line Transform. An example of the result can be seen below in Fig 1.

Fig 1. Original Pipeline

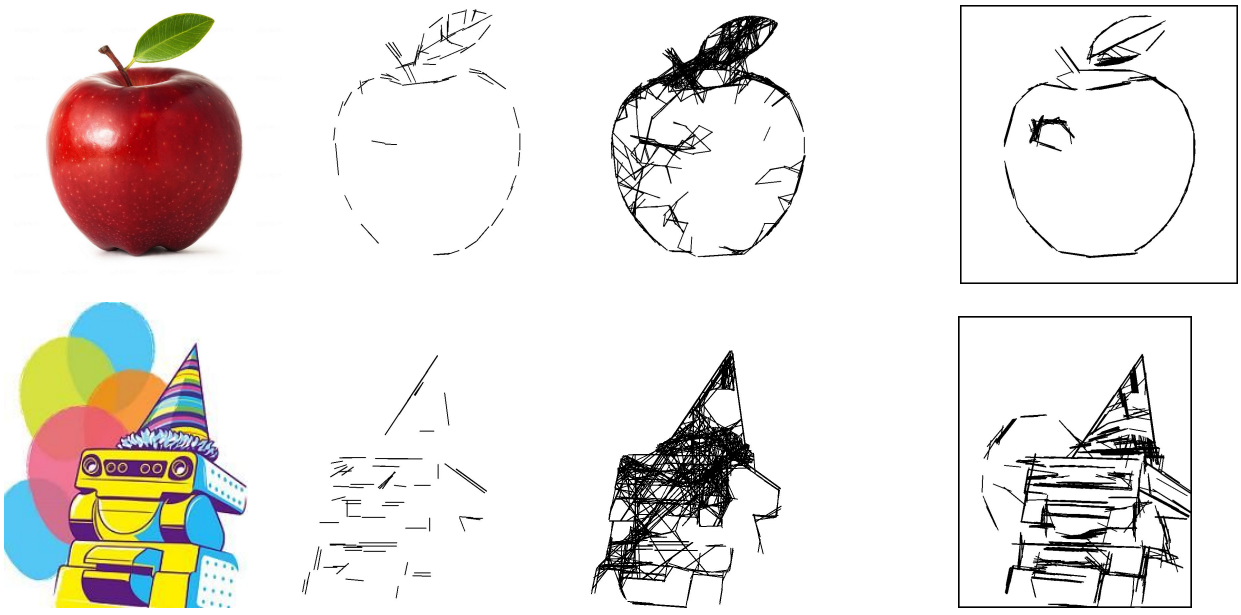
Original Image | Grayscale | Canny Edge Detection | Hough Line Transform



Unfortunately when we try to use the same pipeline for other images besides the apple, especially images with more complexity, the resulting line drawing was not recognizable. This is particularly noticeable in Fig 2, for the image of the PR2 in a party hat. While there are enough lines to represent the apple, there are too few lines to represent the PR2 in a party hat. We also tried tuning the parameters to allow for more lines in the line drawing. However, it was difficult to extract a meaningful image with a minimal number of line segments. We decided to extract contours for the image instead, and the result was more representative of each of the image we extracted lines from.

Fig 2. Comparing Canny Edge Detection

Original Image | Canny Edge Detection | Larger max distance between points | Contours



In choosing to use contours instead of edges, we had to add another step to the pipeline. Contours are curves that join continuous points along the boundary of an object which have the same color or intensity. We want the contours of the image to be drawn, so after converting the image to grayscale, we threshold the grayscale image to achieve a binary image. On this binary image, we can find the contours of the image to be drawn. Additional preprocessing before thresholding, such as changing the contrast or brightness, can be applied depending on the artistic direction.

Fig 3. Final Line Extraction Pipeline

Grayscale | Threshold | Contour | Hough Line Transform

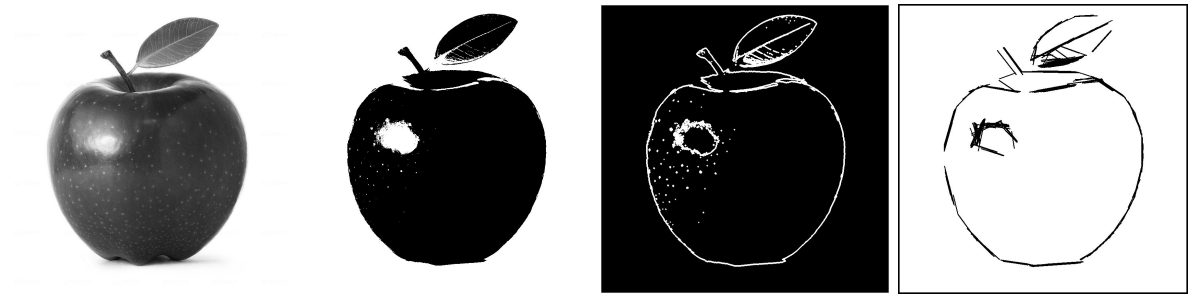


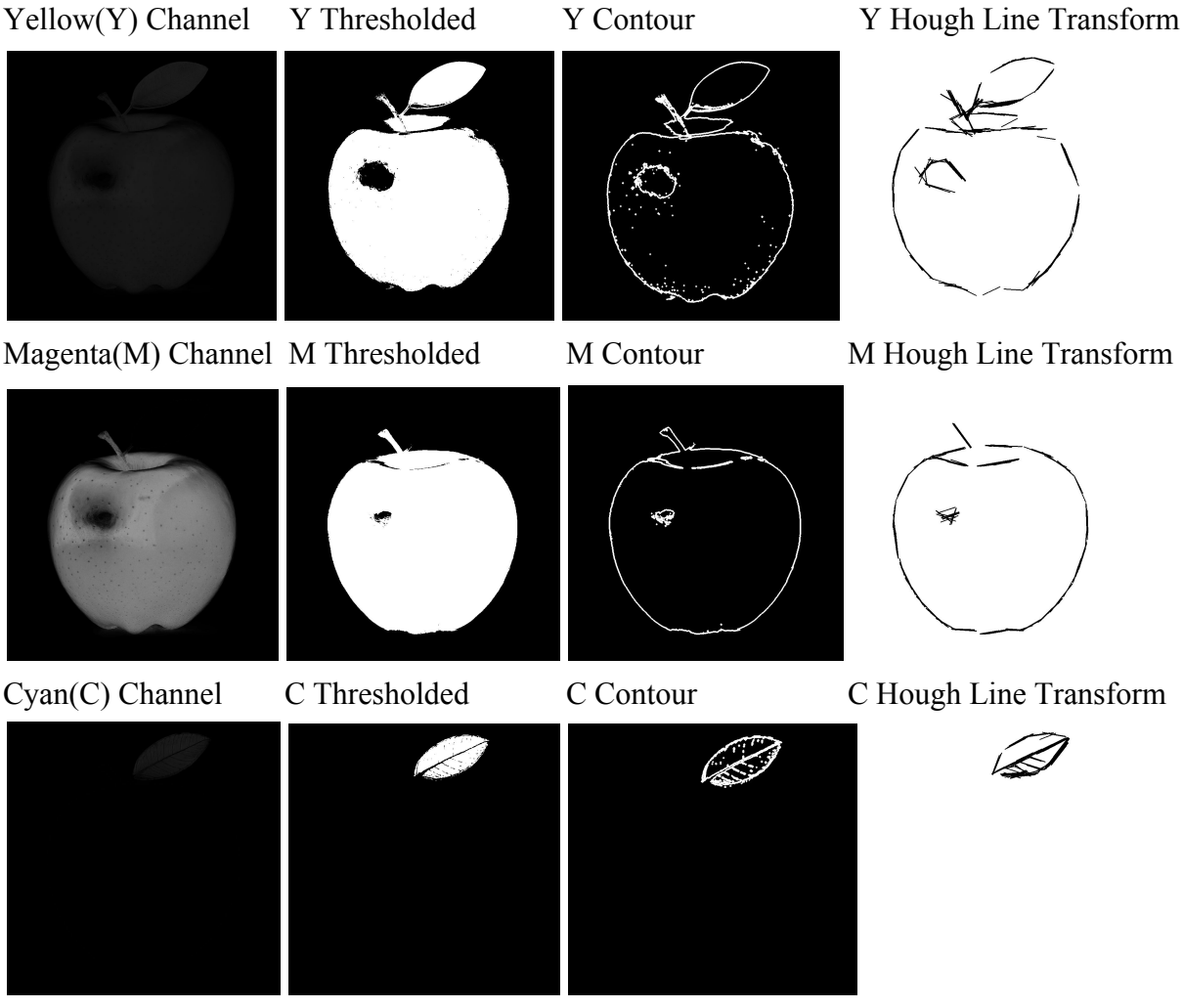
Image Segmentation by Color

Next, we want to segment the image into three primary colors of a subtractive color model: cyan, magenta, and yellow. We want the primary colors of the subtractive model, because we are drawing using ink on a piece of paper, much like a printer. Computer images use the additive model, since each pixel is a set of lights. As such, we need to convert the RGB channels to CMY channels.

$$cyan = 1 - \frac{red}{255},\ yellow = 1 - \frac{green}{255},\ magenta = 1 - \frac{blue}{255}$$

We create separate images for each color channel and extract lines from them using the method outlined in the previous section. When the PR2 draws the lines, the three images are superimposed to create the full drawing.

Fig 4.



Note: Some color channels appear very faint before thresholding.

Drawing Path

We want to minimize the time spent drawing this image, so we should draw the lines in order of how close the next line is to the current line. We choose an arbitrary first line and then find an undrawn line with the closest midpoint to the current line's midpoint and draw that line. Next, we find the an undrawn line with the closest midpoint to the current line and continue until all lines have been drawn.

Transforming the 2D Image onto a Given Plane

One of the most challenging components of this project for us was to determine how to draw on an arbitrary plane within reach of the stationary PR2. We accomplished this by putting the PR2 in mannequin mode, so that we can manipulate the robot by hand. We then place the end effector on the top left (*point a*), bottom left (*point b*), and bottom right (*point c*) corners of plane we want to draw on.

First, we scale the image by $\frac{min(\|c-b\|,\|a-b\|)}{max(image_w, image_l)}$, so that the image stays within the bounds of defined plane. Although we had originally planned on doing a projective transformation we decided that given the images we would be feeding in an orthographic projection suited our needs well so we used the scaling method.

Next, from the collected points, we can construct a rotation matrix as follows:

$$i = \frac{c-b}{\|c-b\|},\ j = \frac{a-b}{\|a-b\|},\ k = i \times j \ \Rightarrow rotationMatrix = [\ i\ j\ k\ ]$$

where:

i is the unit vector along the x-axis composed of the bottom left and bottom right points,

j is the unit vector along the y-axis composed of the top left and bottom left points, and

k is the unit vector along the z-axis, the normal to the plane

We rotate the input image, which is a matrix of points, by multiplying the input image matrix by the rotation matrix.

$$rotatedMatrix = inputImage \cdot rotationMatrix$$

After rotating the matrix, we need to translate the image from the origin of the PR2 to the plane by adding the origin of the plane to each column of the rotated matrix.

$$final = rotatedMatrix + [\ b\ ...\ b\ ]$$

We then add the final transformed image points to a list of waypoints that the MoveIt! cartesian path planner must plan a path through. MoveIt! then uses Inverse Kinematics to solve for joint angles such that the end effector moves through the waypoints and it returns which percentage of the waypoints are reached in the resulting path. As the plane is already within the robot's reach with a set end effector orientation and the waypoints are fed in by nearest neighbor, if the coordinate system is correct, we get 100% reachable.

Robot Pipeline

First, we open the right gripper, give the PR2 the yellow brush pen, and close the gripper. We close the gripper and place the PR2 into mannequin mode. In mannequin mode, we move the arms to touch three corners of the paper(top left, bottom left, and bottom right) the robot will draw on. Next, we exit mannequin mode and have the robot draw the yellow lines. After the robot completes drawing the yellow lines, we release the pen, give the PR2 the magenta pen, and close the gripper. The robot then draws the magenta lines. We repeat the same process for cyan.

Results

Without lifting the pen, without colors, using Canny Edge Detection

https://youtu.be/HkANLR1iUf8

Lifting pen, without colors, using Canny Edge Detection

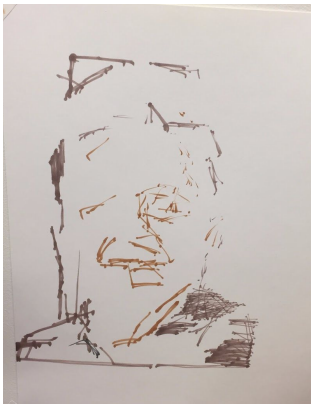
https://youtu.be/TpESsMnF3bc

Lifting pen, with colors, using Contours

https://www.youtube.com/watch?v=y2d0L4QZN1o

Lifting pen, with colors, using Contours

https://www.youtube.com/watch?v=OvfEbuyI69I

Lifting pen, with colors, using Contours, allow for shorter lines
https://www.youtube.com/watch?v=5LxJoHq1EQQ


Lifting pen, with colors, using Contours, allowing for even shorter lines
https://www.youtube.com/watch?v=hyQL34nPw84


Division of labor

While each of us contributed to every aspect of the project, we divided the work so that Beatrice was in charge of image processing, namely converting a given image into three series of line segments in three dimensional space. Jake and Tonye largely focused on moving from simulation space to the real world and manipulating PR2 to produce straight lines onto mounted paper. They accomplished this by creating a method to extract the plane on which the robot draws and to ensure that the coordinate systems were aligned.

Related previous work

We used work from the following papers to get a sense of possible challenges while developing our methods.

Calinon, S., Epiney, J., & Billard, A. (2005, December). A humanoid robot drawing human portraits. In *Humanoid Robots, 2005 5th IEEE-RAS International Conference on* (pp. 161-166). IEEE.
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1573562

Agarwal, S., Rawat, S. S., & Sumathi, V. (2014, February). A drawing robotic hand based on inverse kinematics. In *Information Communication and Embedded Systems (ICICES), 2014 International Conference on* (pp. 1-5). IEEE.
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7034005

Chang, H. H., & Yan, H. (1998). Vectorization of hand-drawn image using piecewise cubic Bezier curves fitting. *Pattern recognition*, *31*(11), 1747-1755.
http://www.sciencedirect.com/science/article/pii/S0031320398000454

Aguilar, C., & Lipson, H. (2008, December). A robotic system for interpreting images into painted artwork. In *International conference on generative art* (Vol. 11).
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.531.8792&rep=rep1&type=pdf

Tresset, P., & Leymarie, F. F. (2013). Portrait drawing by Paul the robot. *Computers & Graphics*, *37*(5), 348-363.
http://doc.gold.ac.uk/~ma701pt/patricktresset/wp-content/uploads/2015/03/computerandgraphics_tresset.pdf

Lau M., Baltes J. , Anderson J. & Durocher S. (2012). A Portrait Drawing Robot Using a Geometric Graph Approach: Furthest Neighbour Theta-Graphs. In *International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE.
http://www.cs.umanitoba.ca/~durocher/research/pubs/lbadAIM2012.pdf