CSCI 5551 - Intelligent Robotics Systems

Final Project Report (Turtle BotROS - The Joy of Painting)

Alex Overman (ayres036)

Nadya Postolaki (posto018)

Sam Williams (will6673)

Abstract

In this project, we propose an algorithm that simplifies images in such a way that a TurtleBot would be able to easily follow along and draw its contents. To this end, we used common edge detection algorithms as well as bitmap images in order to encode the shape of the image, as well as distance formula and inverse kinematics to push the TurtleBot along the edges of the image itself. This algorithm is self-correcting and produces a replica copy of the image.

Introduction

Goals Given a TurtleBot 3, which will be called BotROS, placed in a flat environment, the goal is to process any given image that has been uploaded to BotROS, then replicating that image by traveling on the flat plane of the environment and dragging a pen tool along the plane.

Purpose The idea of BotROS recreating images and artwork stems from the age-old quote said by the beloved Bob Ross, "I think there's an artist hidden at the bottom of every single one of us." Within every human exists some form of artistic talent, and quite possibly the same may be said for every robot too. To allow for creativity to flow from a computer, it must first learn and master the technique of replication, then developing new skills to combine to its previous knowledge will introduce a possibility of unique and creative work. In addition to creative endeavors, BotROS can also be used for the purpose of replicating images such as traffic signs and paths for roadways.

Assumptions The simulation of BotROS is done using ROS and RVis with the burger model of TurtleBot 3. The processing of the images for BotRos to replicate is done via use of existing python3 code, where the image color is turned to grayscale and dodged to produce a picture that will be easier for BotROS to read and replicate on the plane.

Related work A lot of work has been done with recreating images with a robotic arm, such as in [1], however, it appears that not as much work has been done with mobile robots for the purposes of recreating images on a larger scale. One example of a somewhat mobile robot recreating images on a larger (but limited) scale can be found in [2], which features a robot

drawing onto a particular canvas using a sort of pulley system to elevate it and move it around the canvas.

Approach

The project approach began with three main challenges to solve; setting up the environment where BotROS can travel along and "paint" the images that are provided to him, figuring out how to attain an image easy enough for BotROS to be able to process and replicate, and lastly, how to get BotRos to translate the provided image onto the plane he is painting on.

Environment For this project, we ended up using RViz and ROS Noetic. In order to track both the position of BotROS but also the velocity of BotROS, we decided to use RViz in order to ensure that he was making his way to the correct points. We used an empty plane for our RViz map, due to the fact that the applications that we intended for this project would already have the area clear of any debris or obstacles. BotROS starts at (0, 0, 0) on the plane, and would consequently need to be placed at the appropriate spot during real world applications in order to recreate the image to the user's specifications.

"Painting" an image In our initial approach to the BotROS algorithm we decided to use black and white bitmap images due to the fact that numpy, a common library for Python, very readily translates these images into a matrix containing zeroes and ones. This allowed us to identify the next closest pixel to BotROS at any given time using the distance formula while iterating through this matrix.

In order to travel to the next point, we used inverse kinematics in order to determine the difference between our current orientation and the heading that we would need to be going in in order to face the next point. Early attempts at this were successful but took a lot of time due to the fact that our original algorithm would turn left 350 degrees instead of right 10 degrees, so we needed to normalize the angle to avoid going the long way around and wasting precious time. Once the correct heading was established and BotROS had rotated accordingly, it was simply a matter of going forward in the x-direction relative to BotROS's current orientation. Semi-frequently, BotROS's angle will need to adjust slightly to continue in the correct heading,

resulting in a somewhat sinusoidal pattern when approaching certain points. This is likely due to residual angular velocity from the original turn towards the correct heading.

At present, we do not have a pen attached to BotROS, but we have an idea of how we would approach the problem once a pen was successfully attached. Observations can be made from the following diagram.

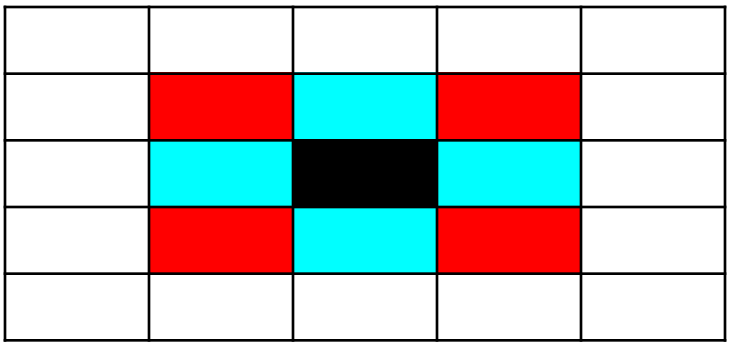

Fig. 1: Pixel distances in our bitmap image

In the above diagram, the black pixel is the current pixel that BotROS is sitting at. The cyan pixels that lie in the direction north, south, east, and west of it would therefore be exactly one unit away from our pixel of origin. Similarly, the red pixels in the image would be a unit of square root two away from our pixel of origin. When determining whether to bring our pen 'up' or 'down', it is as simple as putting the pen down whenever the next pixel is less than or equal to square root two units away from the current one. Otherwise, we would need to bring the pen up to avoid adding unnecessary and inaccurate lines to the drawn image.

Simplifying the image The initial idea was to take an image and vectorize it for ease of translation onto a cartesian plane. A vector image is simply a mathematical representation of an image on a cartesian plane, using formulas to plot coordinates on a map. The formulas from the vector would have then been given to BotROS and he would be able to recreate any image to any size plane. The issue with simplifying the image into a vector is that there could be hundreds, or thousands of different complex formulas for BotRos to follow and recreate. [9] As previously stated when "painting an image", a simple cross can take upwards to 4 minutes to travel through in the simulated environment.

The next solution was to transform the image into a grayscale version and dodge it to a cohesive picture, shown in the figures below. BotROS can then describe a path using black and white bitmap files and identifies the closest pixel using a distance formula- essentially "tracing" the image. Transformations of the images were done via use of pre-existing code shared in a teaching tutorial for python programmers. [11]

Fig. 2 BotROS Sample Image

Fig. 3 BotROS Post-Processing

Other possible solutions included creating an algorithm that replicated the abilities of an Etch-A-Sketch, where an image can be drawn using one single line. The idea behind such an algorithm would be to further simplify an image so that BotRos will no longer have to calculate the distance from the closest pixel, and rather travel along a single path, thus "painting" the image. This solution, however, was introduced fairly late to the project and not enough time was allotted to alter the current solution and guarantee that the Etch-A-Sketch algorithm would work. Another problem that could occur with this algorithm is it's interaction with the way an image is "painted". Since the next closest pixel is calculated for the route of BotROS, complex intersections pose the possibility of having "unchecked" sections that may never be drawn.

Simulating The Pen Mechanism Although not successfully implemented, attempts were made to modify the existing turtlebot to include a pen tool that could be moved up and down to simulate BotROS engaging and disengaging a writing utensil from the surface it's drawing on. This was to be implemented as a prismatic joint attached to the fixed frame of a turtlebot, which would have two orientations it would travel between based on a boolean value tracked via ROS.

The pen tool would be oriented at the origin of the fixed frame, to reduce the number of necessary transformations. If the project were to be expanded to include additional writing utensils for more complex colors and textures, a second concept was devised that would expand on the first device. In addition to the prismatic joint controlling the position of the pen up and down, a rotary joint would be added to spin a disc containing spring held carriages for additional writing utensils. A third position would need to be known for the prismatic joint, which would correspond to the pen engagement mechanism being lifted safely above the writing utensils to let them rotate freely. This, combined with a mechanism for maintaining a grip on the writing utensil (e.g. a pneumatic lock mechanism) would all the engagement mechanism gripping the pen and travelling in the two original positions to start and stop writing, and releasing the pen and retracting to allow the rotary joint to rotate the disc freely between known angular positions containing the additional writing utensils. Concept sketches for both designs have been made and are displayed Figure 4 below.

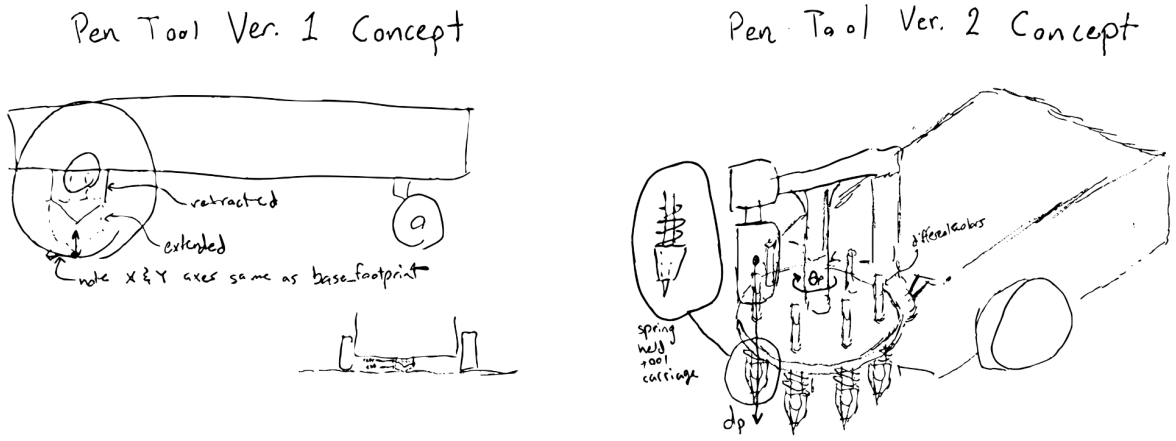

Fig. 4: Pen Tool Concept Sketches

Results

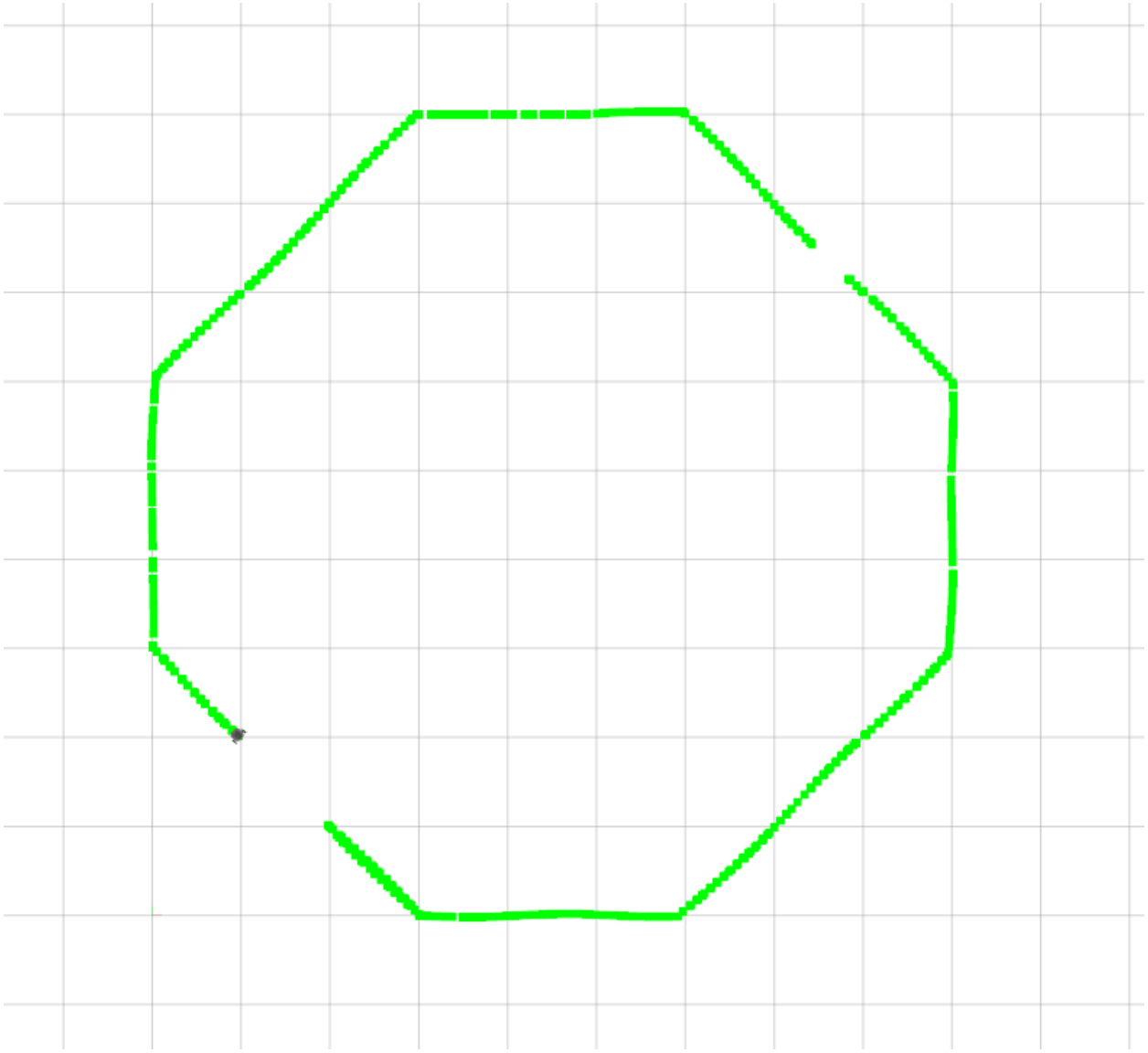

Fig. 5: A ‘circle’ as drawn by BotROS

Though we have made a lot of progress on BotROS, there are still some quirks that need to be worked out. As mentioned previously, some of the lines have a sort of sinusoidal quality to them due to residual angular velocity, though it should be noted that this makes for perhaps a more ‘creative’ output than a direct replica. BotROS is able to successfully follow along with the black pixels in a bitmap image, essentially ‘drawing’ the image therein. Although we do not yet have a pen-attachment, we plan to incorporate that in future designs.

Conclusion

In this project, we proposed an algorithm that simplifies images in such a way that a TurtleBot would be able to easily follow along and draw its contents. An algorithm for a BotROS package was developed that was able to traverse a bitmap image and allow a TurtleBot to navigate according to the pixels in the image. Bitmap images of various shapes were tested, yielding varying results. In order to expand the complexity of the input images, methods for simplifying input images were analyzed, such as image vectorizing and later converting the image to grayscale and dodging it to improve cohesiveness. Finally, concepts for simulating a retractable writing tool were discussed but were met with difficulty when implementing in ROS. Ideal functionality would include generating a variable to control the state of the pen in order to allow the BotROS algorithm to make jumps when necessary. In future, this functionality would be expanded to allow the BotROS algorithm to utilize additional colors. Other planned future work includes improving the efficiency, speed, and precision of the path-finding algorithm for BotROS, implementing the examined methods for generating simplified images for BotROS to draw, and considering difficulties that could appear when applying BotROS to a real TurtleBot device. Overall, this concludes the story of BotROS, and in the words of the late Bob Ross, “until next time, happy painting and God bless.”

References

[1] Markovska, Maria and Felicia Gihl Vieider. “Drawing robotic arm.” (2017).

[2] te Voortwis, Janwillem. BUILDING A GIANT DRAWING MACHINE. University of Twente, July 2017, essay.utwente.nl/73003/1/te_voortwis_janwillem_BA_EEMCS.pdf.

[3] “How to Launch the TurtleBot3 Simulation With ROS.” Automatic Addison, 7 Aug. 2020, automaticaddison.com/how-to-launch-the-turtlebot3-simulation-with-ros/.

[4] “ROBOTIS e.” Manual, emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/.

[5] “[ROS Q&A] Move a Certain Distance, Turn, Then Move (Using Odometry Topic).” The Construct, 28 Aug. 2017, www.theconstructsim.com/move-certain-distance-turn-move-using-odometry-topic/.

[6] Aderinola, Bayode. “How to Reset catkin_ws (ROS Catkin Workspace) and Start Afresh.” The Construct ROS Community, 4 Sept. 2019, get-help.robotigniteacademy.com/t/how-to-reset-catkin-ws-ros-catkin-workspace-and-start-afresh/630.

[7] harshitroy2605. “harshitroy2605/Imag-to-Sketch.” GitHub, github.com/harshitroy2605/imag-to-sketch/blob/master/1.py.

[8] “Image To Vector.” Deep Learning Studies, 11 Oct. 2018, necromuralist.github.io/neural_networks/posts/image-to-vector/.

[9] “How to Use the Linedraw.py Library to Vectorise Images.” How to Use the Linedraw.py Library to Vectorise Images - BrachioGraph 0.1 Documentation, brachiograph.readthedocs.io/en/latest/how-to/use-linedraw.html.

[10] Pirosuke. How To Convert Photo Images To Fantastic Line-Drawings Using Python And OpenCV, 20 Jan. 2019, blog.pyrospect.com/2019/01/how-to-convert-photo-images-to.html.

[11] randerson112358. “Convert A Photo To Pencil Sketch Using Python In 12 Lines Of Code.” Medium, Python in Plain English, 26 Sept. 2020, python.plainenglish.io/convert-a-photo-to-pencil-sketch-using-python-in-12-lines-of-code-4346426256d4.