Project: Search and Sample Return

The goals / steps of this project are the following:

Training / Calibration

- Download the simulator and take data in “Training Mode”
- Test out the functions in the Jupyter Notebook provided
- Add functions to detect obstacles and samples of interest (golden rocks)
- Fill in the `process_image()` function with the appropriate image processing steps (perspective transform, color threshold etc.) to get from raw images to a map. The `output_image` you create in this step should demonstrate that your mapping pipeline works.
- Use `moviepy` to process the images in your saved dataset with the `process_image()` function. Include the video you produce as part of your submission.

Autonomous Navigation / Mapping

- Fill in the `perception_step()` function within the `perception.py` script with the appropriate image processing functions to create a map and update `Rover()` data (similar to what you did with `process_image()` in the notebook).
- Fill in the `decision_step()` function within the `decision.py`

script with conditional statements that take into consideration the outputs of the `perception_step()` in deciding how to issue throttle, brake and steering commands.
- Iterate on your perception and decision function until your rover does a reasonable (need to define metric) job of navigating and mapping.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf.

You're reading it!

Notebook Analysis

1. Run the functions provided in the

notebook on test images (first with the test data provided, next on data you have recorded). Add/modify functions to allow for color selection of obstacles and rock samples.

- first we identify the pixels of the road with the help of the **color_thresh function** with a paramater (160,160,160) which means that each pixel which has rgb values greater than this threshold is considered navigable terain , for the **obstacle identification** we take the pixels that dont meet this requirement and we also apply a mask to take into consideration only the pixels that the camera sees
- finally for the rock samples identification we make a simillar **function find rocks** with the following threshold R > 110 , G > 110 and B < 50 (the rocks are yellow)

1. Populate the `process_image()` function with the appropriate analysis steps to map pixels identifying navigable terrain, obstacles and rock samples into a worldmap. Run `process_image()` on your test data using the `moviepy` functions provided to create video output of your result.

- first we define the source and destination points that we took in the beggining (source from the rover camera and destination from the top down view of the world) which refer to the same location

- second we apply a perspect_transform to go from pixels in the ground to pixels in a top down view we also get our mask from this function
- third we apply **color threshold** to identify navigable terrain/obstacles and rock samples
- fourth we convert those image pixel values to rover-centric coords with the help of **rover_coords** function we also have a function **impose_range** in order to take into consideration only the pixels that are close to the rover for increased accuracy
- fifth we take this images that are rover centric and we superimpose them to the world map in order to find exactly where we are with the help of the **pix_to_world function** in two steps 1.(Rotate the rover-centric coordinates so that the x and y axes are parallel to the axes in world space.)
  2.(Translate the rotated positions by the x and y position values given by the rover’s location (position vector) in the world.)
- finally we update our worldmap with our navigable terain and our obstacles on top of the ground_truth_map in order to test our accuracy giving more weight (+10) to the navigable terain rather than the obstacle map (+1)

Autonomous Navigation and Mapping

1. Fill in the `perception_step()` (at the bottom of the `perception.py` script) and `decision_step()` (in `decision.py`) functions in the autonomous mapping

scripts and an explanation is provided in the writeup of how and why these functions were modified as they were.

- **The perception step does almost exactly the same as the process image function** but it also returns the most navigable terain for our rover to follow and also when we detect a rock we return the angle of that rock and it changes the Rover mode to **go get the rock**
- **the decision step** is the brain of our rover with simple artificial intelligence **(if-else statements)**
- it can detect when to go forward when it sees navigable terain ahead
- it can detect when to stop and turn when there is a wall in front
- it can detect when it is stuck **sees navigable terain ahead but it is not moving** and turns and speeds in an effort to get unstucked
- it can detect when it is looping out of control in which case it stops and goes in a straight line
- when it detects a rock goes straight to him **(if get stucks in the meanwhile it will rotate a bit)** and finally when it is in grabbing distance t will stop and grab it
- Remembers the start location and when it has collected all 6 samples and is less than 10 meters from the start location it will stop and finish his mission

2. Launching in autonomous mode your rover can navigate and map

autonomously. Explain your results and how you might improve them in your writeup.

Note: running the simulator with different choices of resolution and graphics quality may produce different results, particularly on different machines! Make a note of your simulator settings (resolution and graphics quality set on launch) and frames per second (FPS output to terminal by `drive_rover.py`) in your writeup when you submit the project so your reviewer can reproduce your results.

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.

- **the aproach I took was a wall crawler who navigates the whole map and locates the rock samples and picks them up**
- **the techniques i used were go forward with an offset in order to really follow the wall, get unstuck if we encounter a rock or stuck in wall and a looping breaker behaviour which might happen in an effort to follow the wall but the wall is far away**
- **the pipeline might fail if it gets really stuck or a rock is barely visible or rarely if it fails to pick up a rock because is never on the left side of the rover (the rover will pick only rocks on its left in order to follow the wall and not zig**

zag). Finally sometime i get a weird error when it is rotating in a wall continiously

- **Improvements might be a more intelligent route planning that we will learn in future projects, more speedy approach, detecting obstacles in order to avoid them in the first place and not having to get unstuck and finally a more intelligent way to get back to the start when we have collected all of the samples instead of following the wall untill we get within 10 meters of the start**