PROJECT SPECIFICATION

Search and Sample Return

Writeup

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
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. Here is a template writeup for this project you can use as a guide and a starting point.	The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled.	This document outlines and addresses all the rubric points needed for the project. Each of the rubric points are addressed in the following tables against each rubric point.

Notebook Analysis

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
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.	Describe in your writeup (and identify where in your code) how you modified or added functions to add obstacle and rock sample identification.	To the function perspect_transform, added as mask to the pixels that are not in field of view of the camera. Can be seen in cell 6 of the notebook or line 77 of the perception.py Added a function to identify the rocks. The function is find_rocks (defined between lines 80-88 in perception.py or in cell 9 of the notebook.
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.	Describe in your writeup how you modified the `process_image()` to demonstrate your analysis and how you created a worldmap. Include your video output with your submission.	The following steps are performed in process_image: # 1) Define source and destination points for perspective transform # 2) Apply perspective transform # 3) Apply color threshold to identify navigable terrain/ obstacles/rock samples # 4) Convert thresholded image pixel values to rover-centric coords # 5) Convert rover-centric pixel values to world coords # 6) Update worldmap (to be displayed on right side of screen) # Example: data.worldmap[obstacle_y_world, obstacle_x_world, 0] += 1 # data.worldmap[rock_y_world, rock_x_world, 1] += 1 # data.worldmap[navigable_y_world, navigable_x_world, 2] += 1 # 7) Make a mosaic image, below is some example code

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		# 7.1) First create a blank image (can be whatever shape you like) # 7.2) Next you can populate regions of the image with various output # 7.3) Here I'm putting the original image in the upper left hand corner # 7.4) Add the warped image in the upper right hand corner # 7.5) Overlay worldmap with ground truth map # 7.6) Flip map overlay so y-axis points upward and add to output_image # 7.7) Then putting some text over the image

Autonomous Navigation and Mapping

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
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.	`perception_step()` and `decision_step()` functions have been filled in and their functionality explained in the writeup.	**Perception Step:** Step1: Perception points are defined Step2: based on percption points a perspect_transform has been performed. Step3: color threshold has been applied to identify navigable terrain/obstacles/rock samples 4. Rover.vision_image has been udpated based on the previous step. 5. Map pixel values are converted to rover-centrix co-ordinates 6. Rover-centrix pixel values are converted to world coordinates. 7. Rover worldmap is updated 8. rover-centrix pixel positions are converted in to polar coordinates **Decision Step:** The decision step is based on the decision tree below.

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
		 We always keep checking the state and take respective action. I have not made any changes to the base code for decision step.
Launching in autonomous mode your rover can navigate and map autonomously. Explain your results and how you might improve them in your writeup.	By running `drive_rover.py` and launching the simulator in autonomous mode, your rover does a reasonably good job at mapping the environment. The rover must map at least 40% of the environment with 60% fidelity (accuracy) against the ground truth. You must also find (map) the location of at least one rock sample. They don't need to pick any rocks up, just have them appear in the map (should happen automatically if their map pixels in `Rover.worldmap[:,:,1]` overlap with sample locations.) **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**	The rover is able map over 40% of the environment with 60%+ fidelity consistently. Please take a look at the rover.mp4 under videos folder. The setting I ran simulator was at 800x480 resolution and the FPS was around 23-25.

CRITERIA	MEETS SPECIFICATIONS	COMMENTS
	second (FPS output to terminal by `drive_rover.py`) in your writeup when you submit the project so your reviewer can reproduce your results.	