Robotics Nanodegree

P1: Search and Sample Return

Markus Buchholz

1. Introduction

The goal for this project was to design Rover autonomous system which maps a simulated environment and searches for samples of interest (golden rocks). The code was written in Python with support of OpenCV library.

In order to detect and then collect the samples, the rover by use of camera explores the environment and navigate potential (defined) hazards obstacles.

Collecting samples and rover stuck avoiding requires special attention, therefore proper functions preventing such state occurrence were implemented.

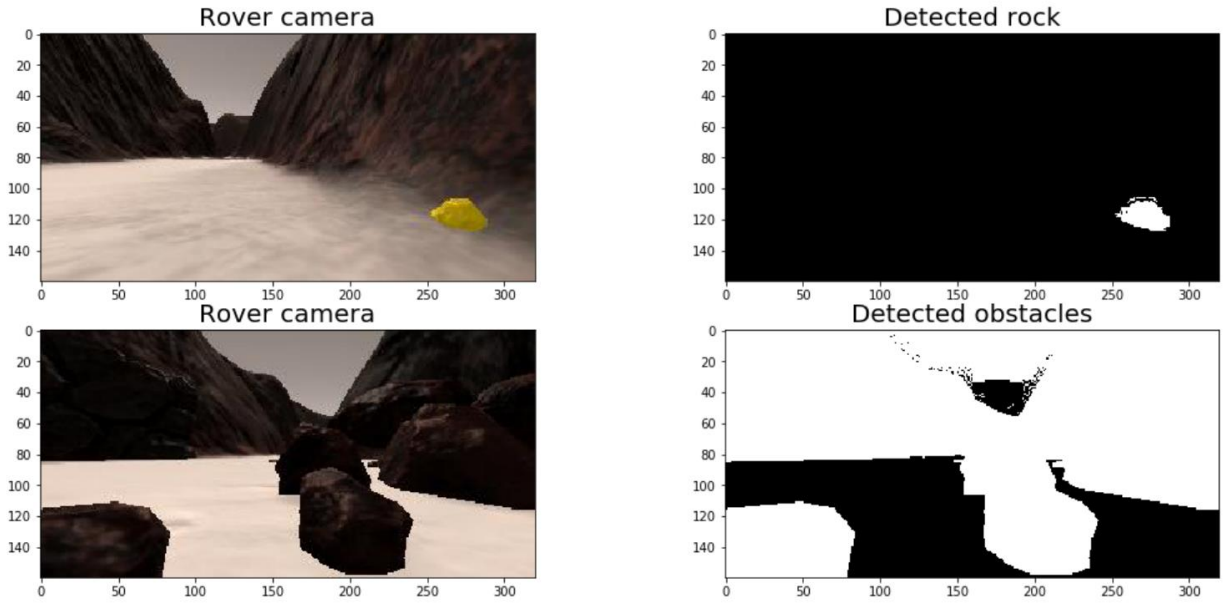
2. Notebook Analysis

2.1 obstacle and rock sample identification

In order to identify **obstacles** the image was analyzed similar to rover movable path detection. In this case for the coming image the three channel threshold (detecting black color) is applied and the same empty array is created. However in this case the output image is constructed in opposite way to the path detection. In this case the pixels below the threshold are classified as Boolean 1.

```
def color_thresh_obstacle(img):
    color_select = np.zeros_like(img[:,:,0])
    below_thresh = (img[:,:,0] < 160) \
                & (img[:,:,1] < 160) \
                & (img[:,:,2] < 160)

    color_select[below_thresh] = 1
    return color_select
```

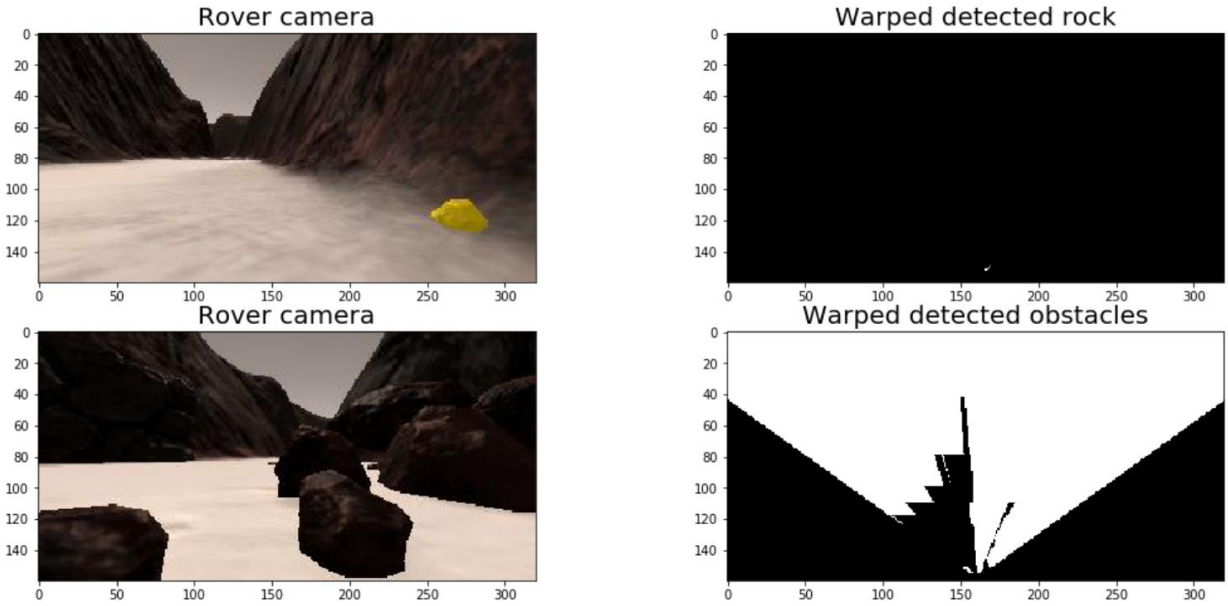


For the **rock sample** identification the OpenCV was applied. Here the yellow color from the coming image is separated. Function cv2.inrange identifies the pixels (in HSV format) which values are within defined range (for yellow color). Output of the function is the gray image containing the detected yellow objects. In order to send back image the main (pipeline) the conversion to RGB format has to be applied.

```
def color_thresh_rock(img):
    #inrang function requires HSV format
    rgb = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    hsv = cv2.cvtColor(rgb, cv2.COLOR_BGR2HSV)

    # threshold for yellow objects
    yellow_min = np.array([20, 190, 20], np.uint8)
    yellow_max = np.array([30, 255, 255], np.uint8)
    threshold_yellow_img = cv2.inRange(hsv, yellow_min, yellow_max)
    # output images should be in RGB format
    threshold_img = cv2.cvtColor(threshold_yellow_img, cv2.COLOR_GRAY2RGB)


    threshold_img= threshold_img[:,:,0]
    return threshold_img
```

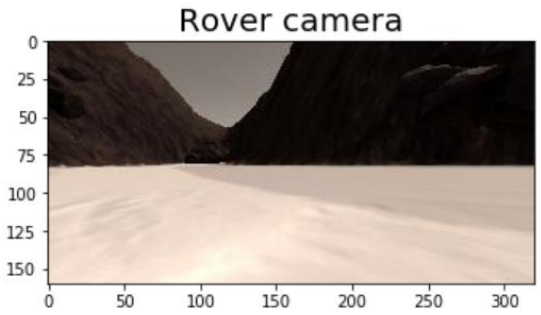


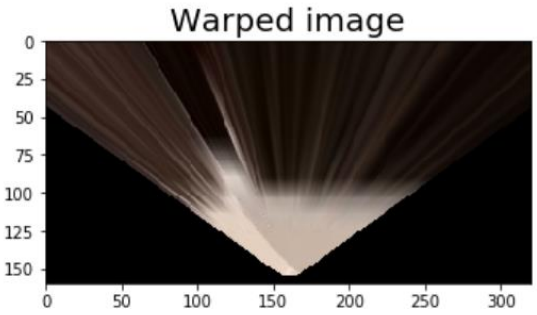
2.2 Worldmap creation – process_image() function (pipeline)

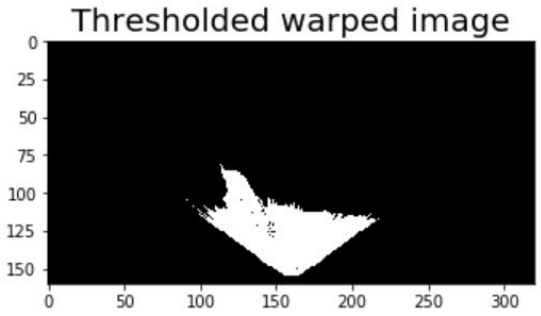
The main idea for the process_image function (pipeline) is to perform the image analysis in order to extract the information about the actual Rover state. Base on the analyzed Rover state the correct decision (in decision_step function) is performed. Beside the core information for the rover autonomous control system the world map is created.

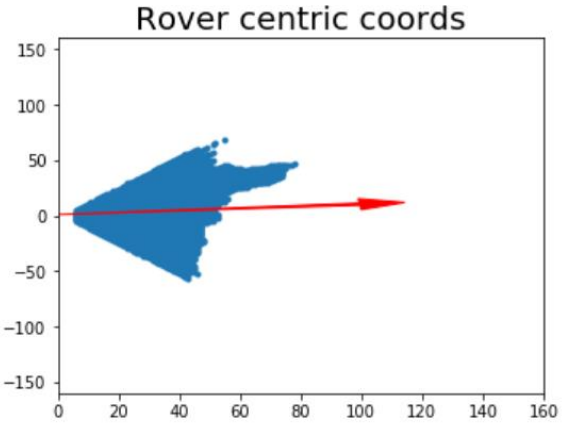
The pipeline can be defined (described) in following steps:

a. By use of defined source (image) and destination points the perspective transform is applied. Output from the function is warped image.
warped = perspect_transform(imgage, source, destination)
b. The color threshold for identification navigable terrain, obstacles and rock samples is applied. Output are thresholded images.
c. Thresholded images are converted to rover centric coordinates.
d. Rover centric coordinates pixels are converted to world coordinates
e. World coordinated pixels are used to update the World map.
f. World map, original image and warped image create the video frame. Video can be seen: test_mapping.mp4


Rover camera


Warped image


Thresholded warped image


Rover centric coords


Rover Camera
Thresholded view
World map

3. Autonomous Navigation and Mapping

Deployed in notebook pipeline function has been incorporated into perception step function which is called when the rover explore the environment.
Similar steps as in process_image function have been utilized. However, rover autonomous control system (rover pipeline = perception_step) has been developed by adding extra functionality. Outputs from allocated additional functionality are used by decision_step function in order to perform rover control commands.

The ***perception_step function*** can be presented as follows:

a. By use of defined source (image) and destination points the perspective transform is applied. Output from the function is warped image.
   warped = perspect_transform(imgage, source, destination)
b. The color threshold for identification navigable terrain, obstacles and rock samples is applied. Output are thresholded images.
c. Perfom the update of Rover.vision_image, which will be displayed on left side of the screen
d. Thresholded images are converted to rover centric coordinates.
e. Rover centric coordinates pixels are converted to world coordinates
f. World coordinated pixels are used to update the Rover world map.
g. Perform the conversion of rover-centric pixel positions to polar coordinates for rover, rock samples and obstacle positions.

Decision_step function is used to send correct command in order to perform correct movement by rover.

The function was bevided by two parts (modes) which which take a decision what to do if rover moves or is stopped. In this project, it was noticed that in order to succeed the extended goal (to collect sample rock) it was necessary to secure that the rover could manage to overcome the stuck position (when rover crashes with obstacle or cannot move out from blind narrow path).

In **forward mode** (while robot moves forward) the following actions/decisions can be performed:

a. Update of the rover position if the terrain is proper for the rover movement. Reduce throttle is there is not.
b. Perform action while the rover stuck (rover velocity is 0 while the throttle is adjusted to the command level). In this situation the throttle is cut, brake activated and rover turns. In order to secure loop avoidance (rover has to know when to stop rotate) the counter is implemented. After proper rotation rover receive command to follow proper angle with commanded throttle.
c. Perform the picking the sample rock if the rock is detected. In this case the rover receive command to slow down and follow the angle which points the rock sample. When the goal (rock) was approach activate the picking command. Mounted on the rover robot picks the rock.
d. When the rover approaches the obstacle the modus forward changes to stop modus

In **stop mode** following functions have been implements:

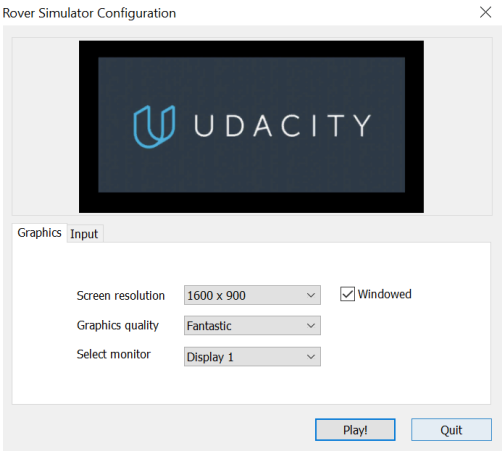
a. If the rover does not stop completely, so the brakes are still active.
b. When the rover is stopped, verify the terrain and start procedure to move the rover (set the flag to forward).
c. In stop mode the rover stuck state is also checked and proper steps (identical to forward mode) are performed.

4. Summary:

a. In this project, it was solved the mapping problem for rover control system. Rover explores environment autonomously and collects depending on position, 3 to 5 rock samples. The simulation can take between 15-20 minutes.

b. Simulator settings (resolution and graphics quality set on launch) and frames per second used in this project were as follows:

Screen resolution: 1600x900
Graphics quality : Fantastic
FPS: 17 – 26



c. Rover maps over 95% of the simulated environment with fidelity around 43%
d. Rover can encounter stuck position (if the rover crashes with central positioned obstacles or after picking up rock samples) and it take a while (around 60 – 120 seconds) to regain the forward movement.


Rover Simulator
Throttle: 0.2
Brake: 0.0
Steer angle: -10.8674
Ground speed: 0.2m/s
Position: (96.2, 78.6)
Pitch angle: 359.25
Yaw angle: 106.41
Roll angle: 359.97
Is near objective: No
Is picking up:No
Autonomous Mode!
Time: 1317.4 s
Mapped: 98.2%
Fidelity: 43.5%
Rocks
Located: 6
Collected: 5