Project 1: Search and Sample return

Explanation of functions

color_thresh - Threshold of RGB > 160 does a nice job of identifying ground pixels only

color_obstacle - Threshold of RGB < 160 does a nice job of identifying ground pixels only

color_rock – identifies rocks in golden color

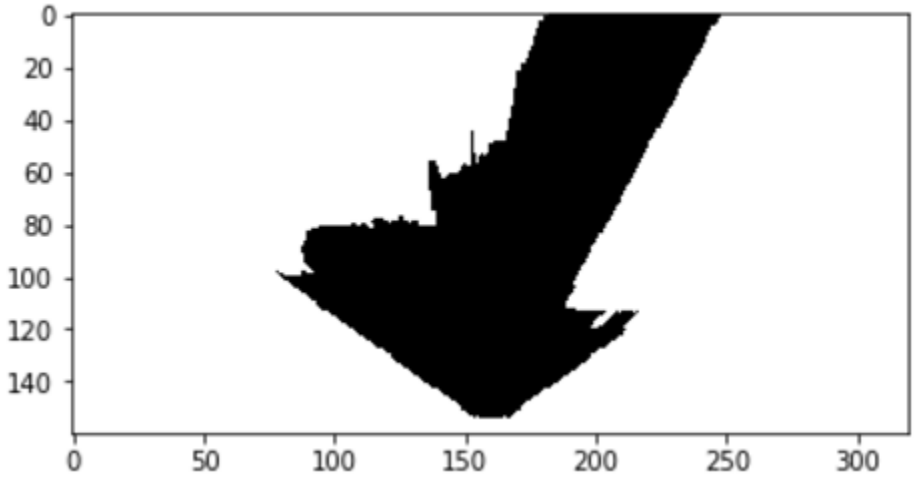
rover_coords – Find coordinates in terms of rover

to_ polar_coords – Converts radiant to polar coordinates

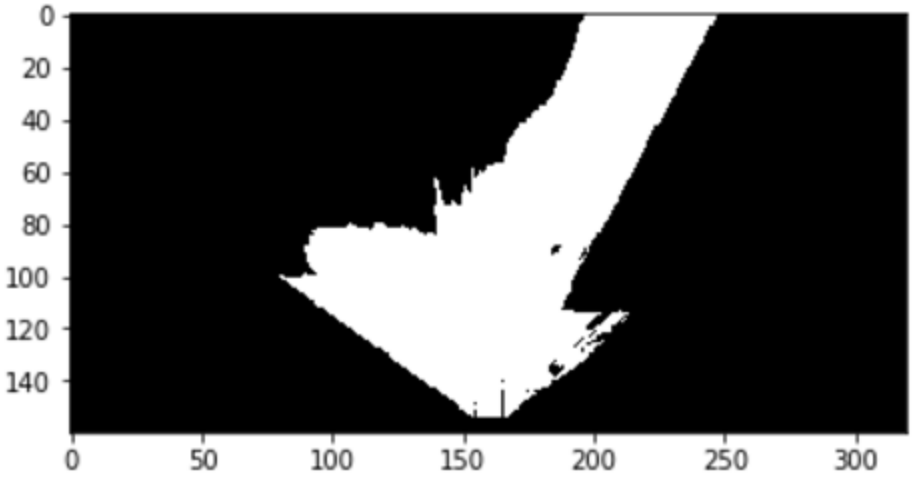
rotate_pix – Rotational with respect to Rover coordinates from world

translate_pix - Transformation with respect to Rover coordinates from world

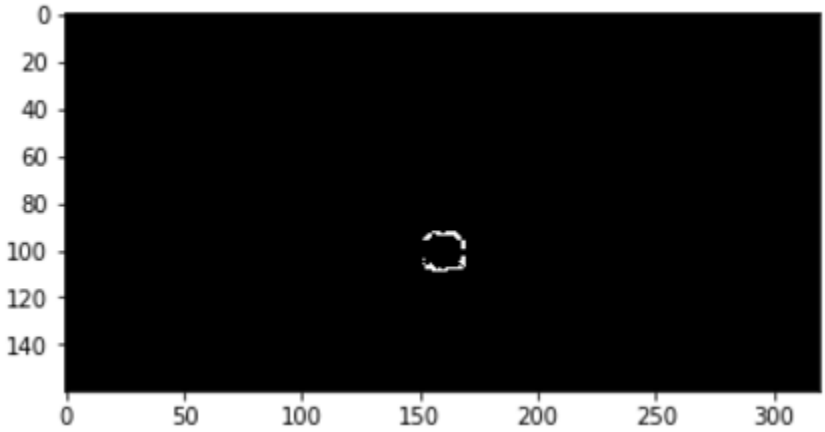
Identifying Terrain



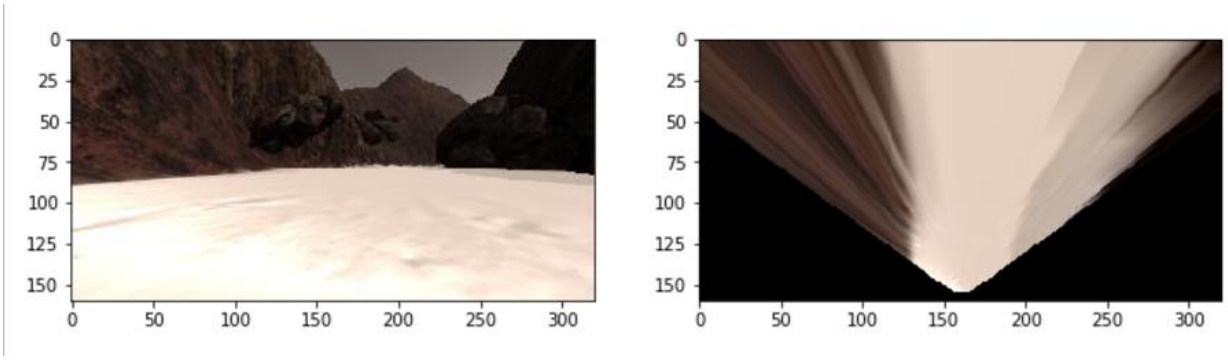

Identifying Obstacles

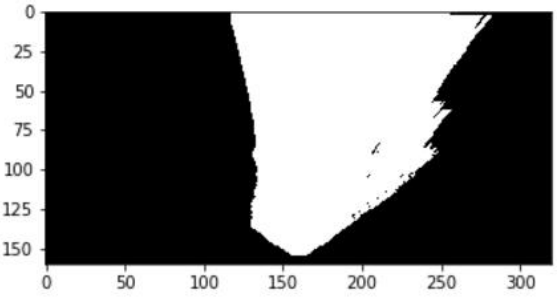


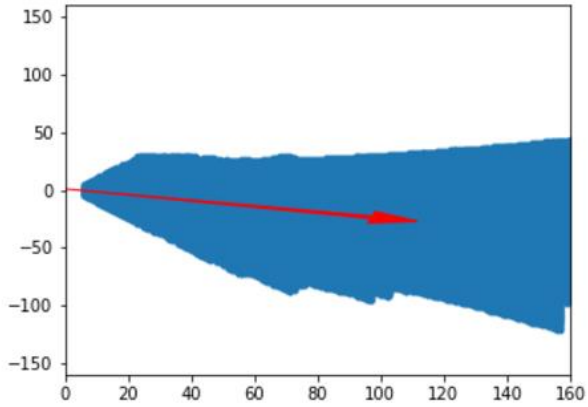

Identifying Rock




Terrain images






Function explanations

process_image()

There were three objects that needs to be identified. Namely, Terrain, Obstacle and rock. These were fetched from every video image as the rover proceeds.

As these objects are in different colors, I used RGB values of them to differentiate. As Obstacles are darker in color, RBG threshold of less than 160, 160, 160 was considered as Obstacles.

Terrain was lighter in color, RGB thread hold more than 160, 160, 160 is considered as terrain.

Identifying the rocks was done providing the upper and lower range of gold color.

This was calculated using providing the range for rock color (100, 100, 20) to (255, 255, 80)

perception_step()

Initially the coordinates for the grid just in front of the rover was identified from the map which was marked as source.

The relative coordinates was calculated in terms of the total image and marked that as destination

Long view transformation of this was found in perspect_transform.

For the given frame, the terrain, obstacles and rocks were identified passing them to their respective threshold functions and they were updated to Rover.vission_image triple in different indices.

Rover view relative pix cells of these values were calculated passing the threshold values of terrains, obstacles and rocks.

For doing this a scale of 10 was used. The image was rotated, translated and reduced in scale using pix_to_world

These values were updated in the World map

Rover centric values were converted from radians to polar coordinates uses to_polar_coords

decision_step()

Decistion.py controls the decision step of the rover and decides its next step with respect to its current position and state

This was the most exploratory and interesting part of the project.

This is how it works.

Rover than looks to move forward.

Its mode is set 'forward'. If the navigable angels are sufficient it looks if is struck. It move forward if not at a mean of navigable angle.

If the rover doesn't have sufficient navigable angle it comes to a stop mode.

Actions Under 'stop' mode

If velocity is not zero it apply brake. If it has sufficient navigable angle forward at its mean angel. If navigable angles are less than forward angles it decides to turn setting its steer to right.

Actions Under 'stuck' mode

Its brake is released with full acceleration and it steers complete right and proceeds further

Areas of improvements

- Pick up rocks
- Rover still needs further intelligence on which direction to turn
- Needs algorithm to get back to origin.

Screen size

Screen 1366 x 768

Graphics quality Fantastic