Rover Report

Course Code: CSE 374s

Course Name: Digital Image Processing

Dr. Mahmoud Khalil

Supervised by: Eng. Mahmoud Selim

Ain Shams University

Faculty of Engineering

Fall Semester – 2022

Team Members

Name	ID
Heba Mahmoud Abd Elhafez Bakry	1900402
Ahmed Elsayed Rashad Mohamed	1900730
Nada Amin Abdelsattar Ali	1900594
Mariam samir wassef	1900313
Jumana Emad Eldin Saleh Mohamed Elhak	1900980

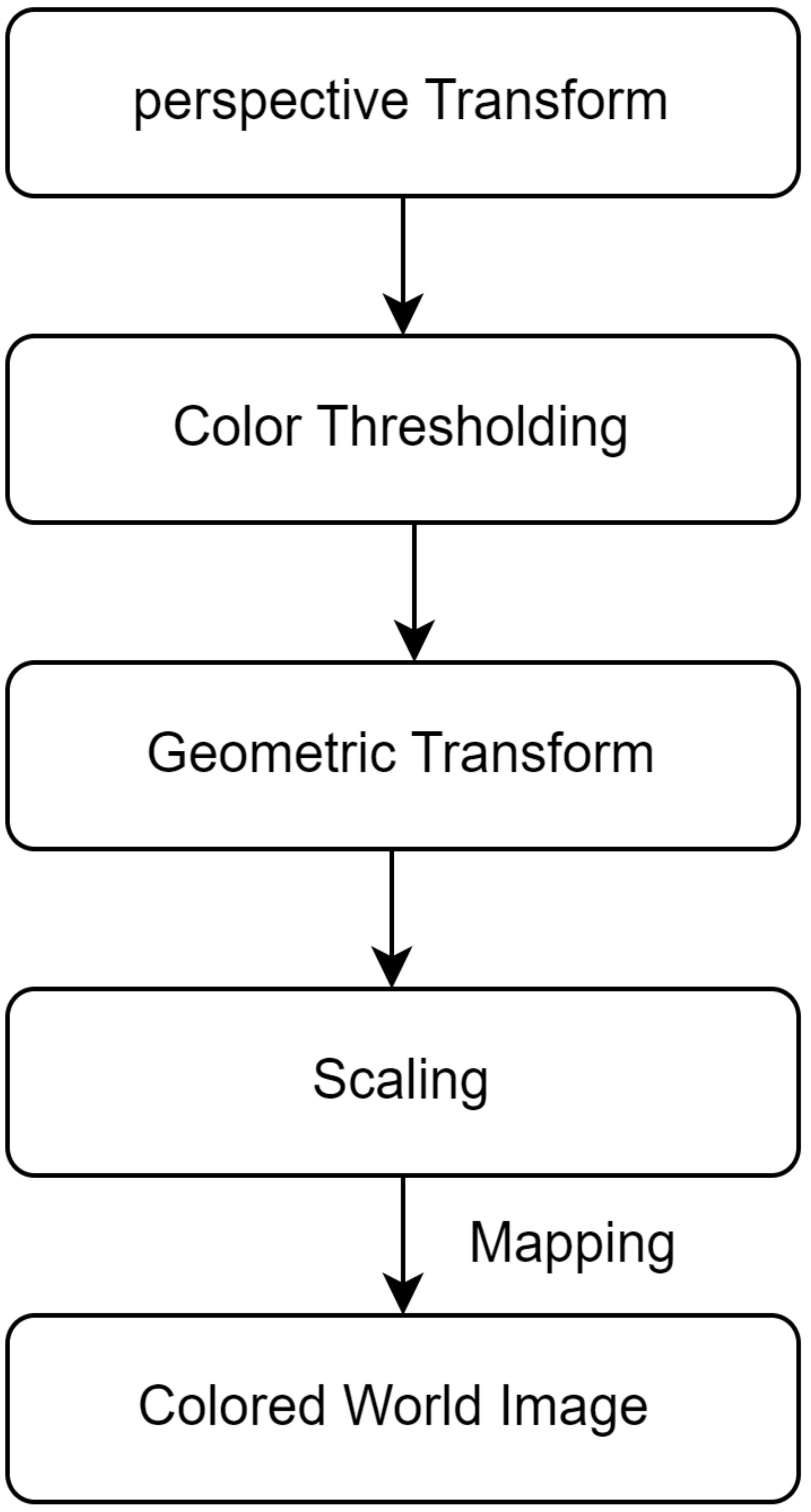
Github link:

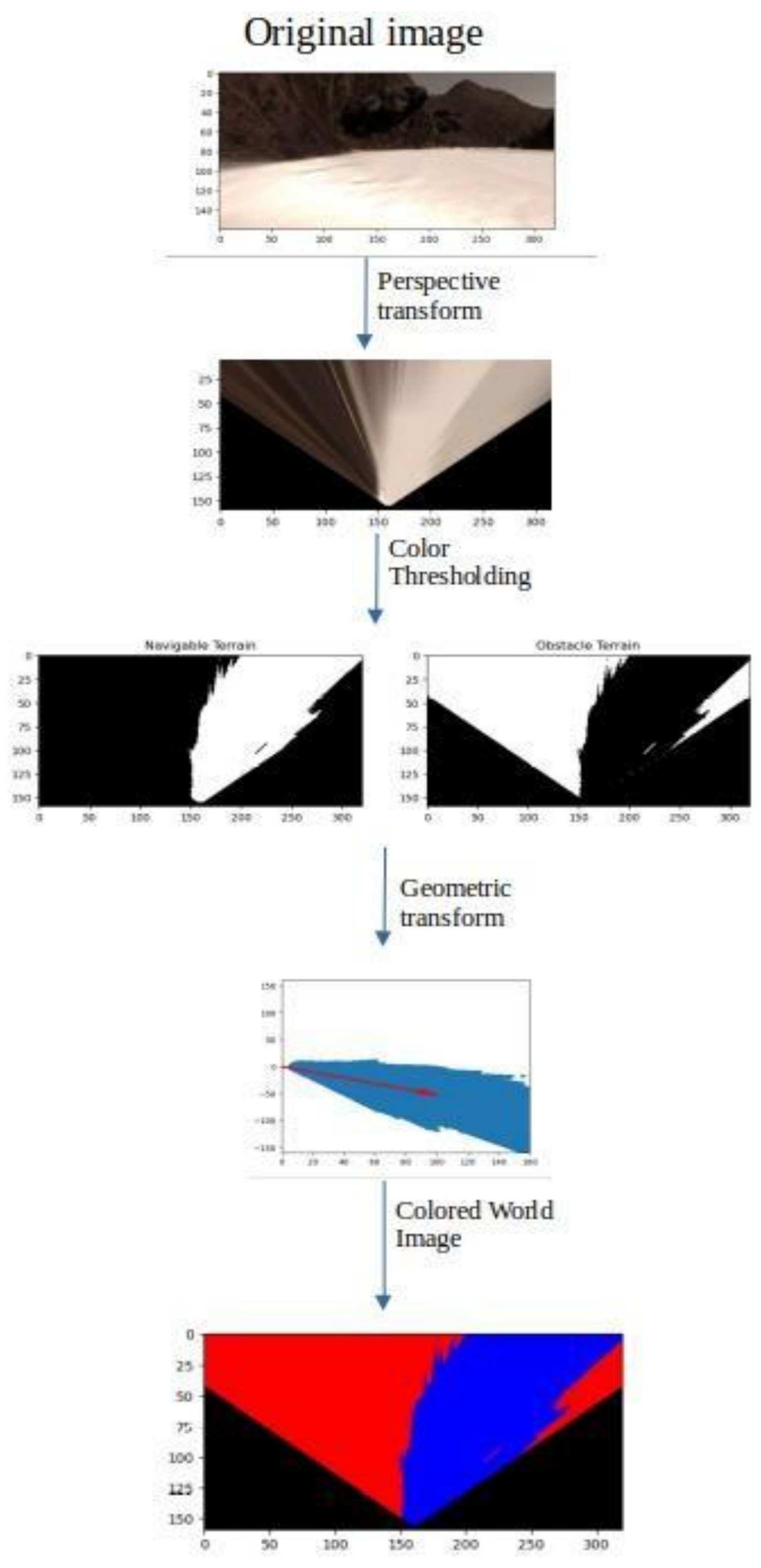
https://github.com/ahmedelsayed968/Rover_Search_And_Return

Video link:

https://drive.google.com/file/d/14Vdw9d5j1N71PI4x8JJOGn929szSP9EU/view?usp=sharing

Diagram of Methods' Sequences:

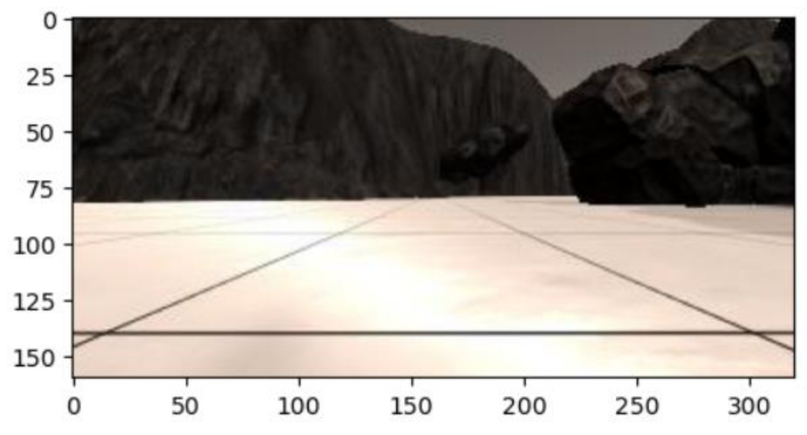


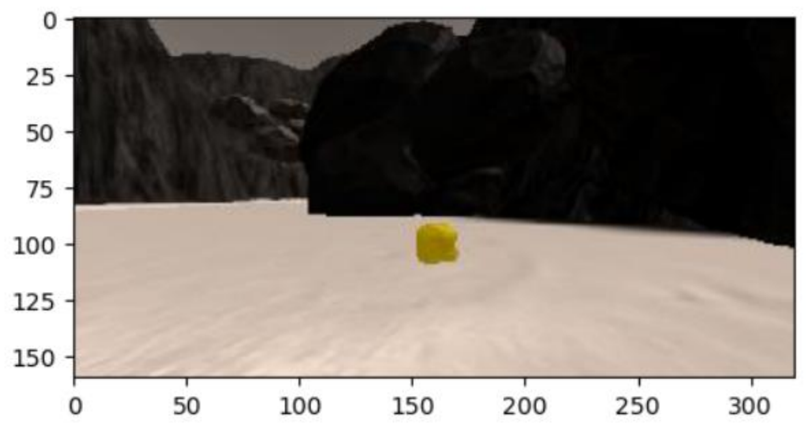


Methods' Codes:

1- Perspective Transform:

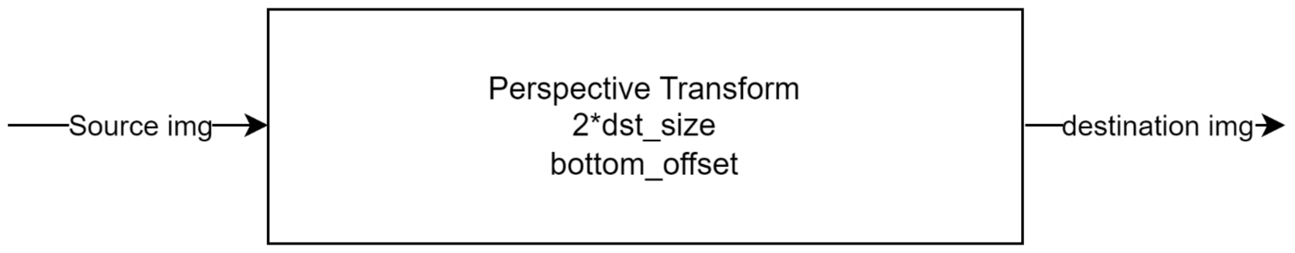
It is applied to convert an image viewed from the rover camera from a 3D view to a top-down 2D (warped) view. This is needed so that images viewed from rover camera can be eventually mapped to a ground truth of the world.





The grid image above is used to choose 4 source coordinates corresponding to the locations of the corners of the grid cell which is in front of the rover. Each grid cell represents a 1 square meter grid in the simulator environment. These four source points are subsequently mapped to four corresponding grid cell points in the output warped image where each 6x6 pixel square represents 1 square meter viewed from top-down.

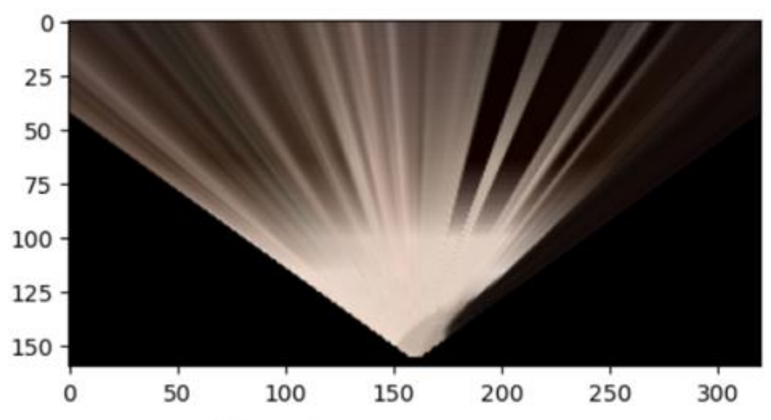
Eventually, the warped, 2D top-down image is thresholded to extract a particular region of interest (navigable terrain in above image), and then expressed in the rover's coordinate frame such that the region of interest is centered at the origin of this coordinate frame. Fixing a coordinate system with respect to the robot is central to our robot as it allows objects in the robot's environment to be described with respect to the robot's camera in this case.



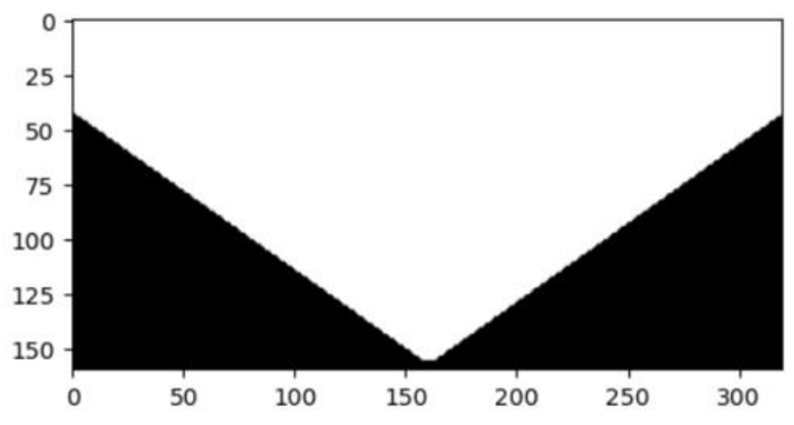
2*dst_size : size of 2D output image box of 6x6 pixels equaling 1 m^2

bottom_offset: bottom of camera image is some distance in front of rover

to warp the image to top-down view before thresholding we needed first to ignore the pixels out of the field of view (the two dark triangular areas on the bottom left and right). For that we have created a mask consisting of a warped white image



Warped Image



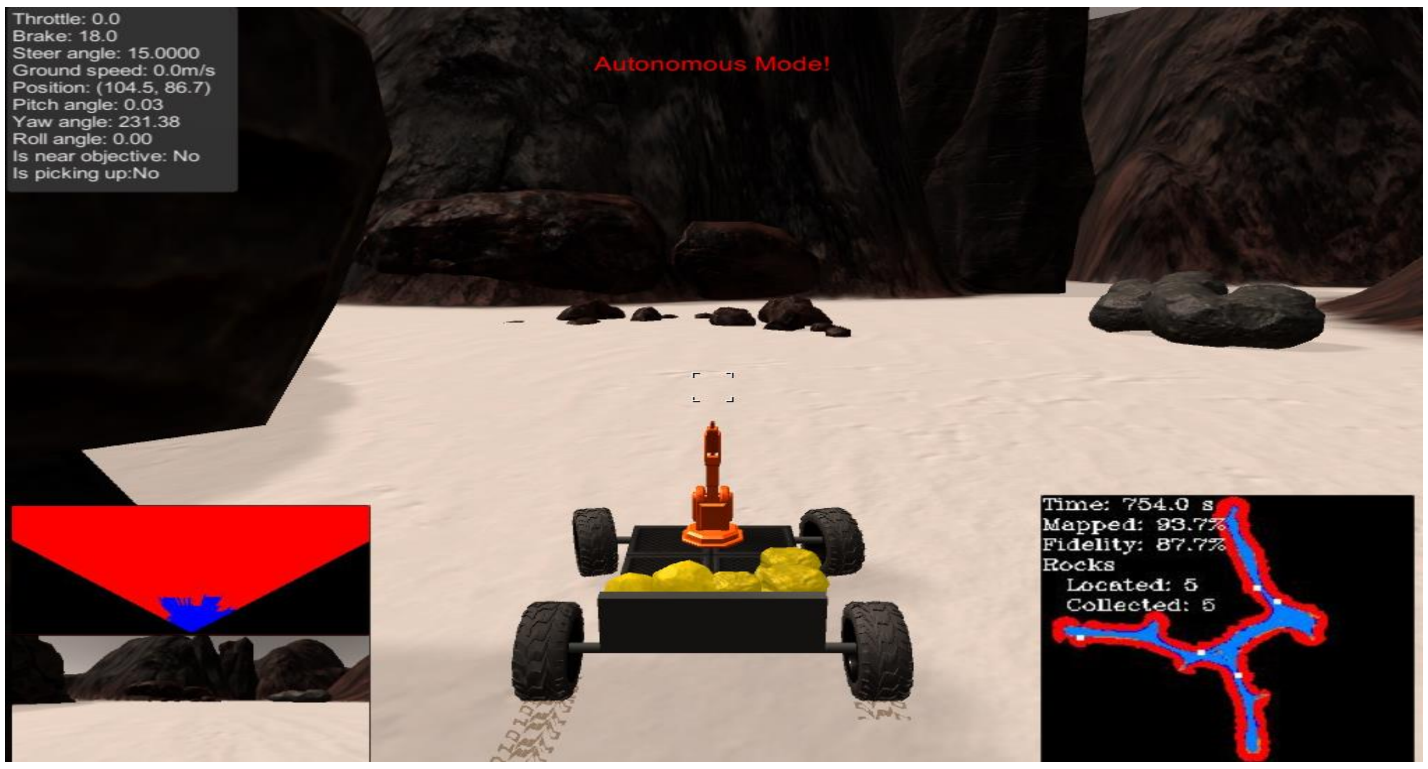
Mask for filtering out the pixels out of the field of view

2- Returning to initial position:

When the rover collects 5 rocks and mapping is more than or equal to 93%, it goes to end state before 5 m in both 2 axes at the deceleration equals 18 and then this state makes throttle equal zero then finally it stops at the initial position.

The deceleration is calculated by:

$$a = \frac{V_f{}^2 - V_o{}^2}{2d}$$



```
Throttle: 0.0
Brake: 18.0
Steer angle: 15.0000
Ground speed: 0.0m/s
Position: (104.5, 86.7)
Pitch angle: 0.03
Yaw angle: 231.38
Roll angle: 0.00
Is near objective: No
Is picking up:No
```

Autonomous Mode!

```
Time: 754.0 s
Mapped: 93.7%
Fidelity: 87.7%
Rocks
  Located: 5
  Collected: 5
```

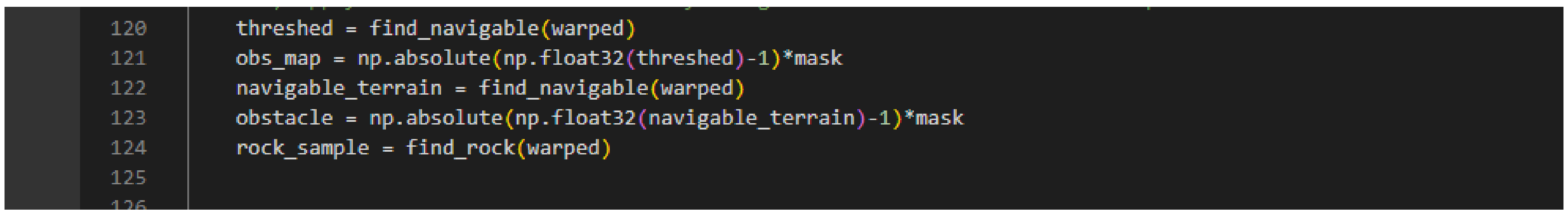
2- Color Thresholding:

→ `it's used to identify navigable terrain,obstacles and rock samples`

```python
# Identify pixels above the threshold
# Threshold of RGB > 160 does a nice job of identifying ground pixels only
def find_navigable(img, rgb_thresh=(150, 150, 150)):
    # Create an array of zeros same xy size as img, but single channel
    color_select = np.zeros_like(img[:, :, 0])
    above_thresh = (img[:, :, 0] > rgb_thresh[0]) \
        & (img[:, :, 1] > rgb_thresh[1]) \
        & (img[:, :, 2] > rgb_thresh[2])

    color_select[above_thresh] = 1
    return color_select


def find_rock(img, rgb_thresh=(110, 110, 50)):
    # Create an array of zeros same xy size as img, but single channel
    color_select = np.zeros_like(img[:, :, 0])
    thresh = (img[:, :, 0] > rgb_thresh[0]) \
        & (img[:, :, 1] > rgb_thresh[1]) \
        & (img[:, :, 2] < rgb_thresh[2])

    color_select[thresh] = 1
    return color_select
```

3- Apply color threshold to identify navigable terrain/obstacles/rock samples

obstacles are defined as the absolute of navigableterrain-1 multiplied by the mask that we defined previously that removes the pixels that are not in the view of the rover's camera; that we are not interested in.



```
    threshed = find_navigable(warped)
    obs_map = np.absolute(np.float32(threshed)-1)*mask
    navigable_terrain = find_navigable(warped)
    obstacle = np.absolute(np.float32(navigable_terrain)-1)*mask
    rock_sample = find_rock(warped)
```

4- Convert rover-centric pixel values to world coordinates

We defined a function to apply rotation and translation in both map world and obstacles' world using (pix_to_world) function where we used affine transformation which is a combination of translation, rotation and scaling.

the matrix expressed below is the rotation matrix

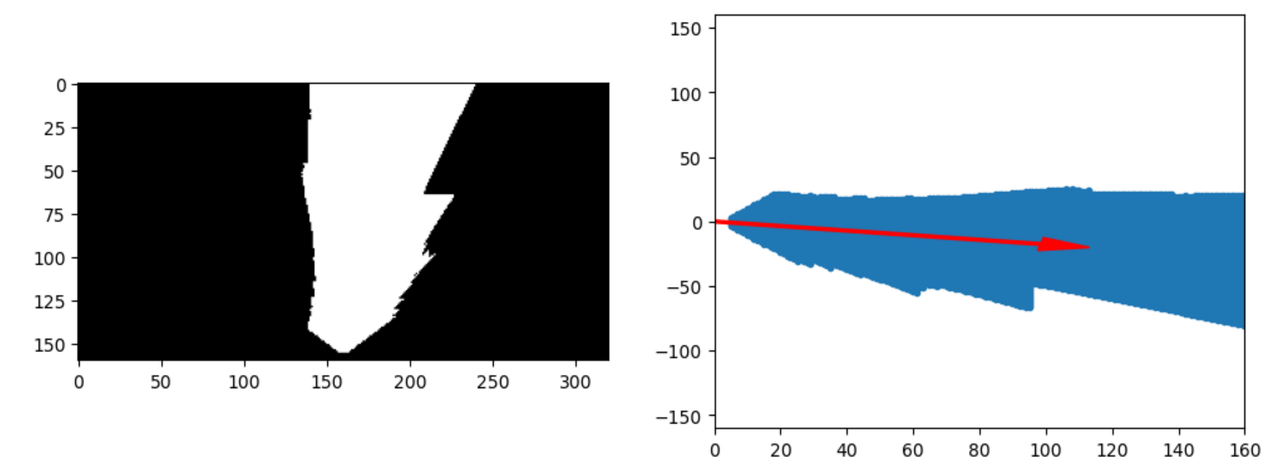
where x2 and y2 are the rotated positions

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

the translation matrix that we used with scaling:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

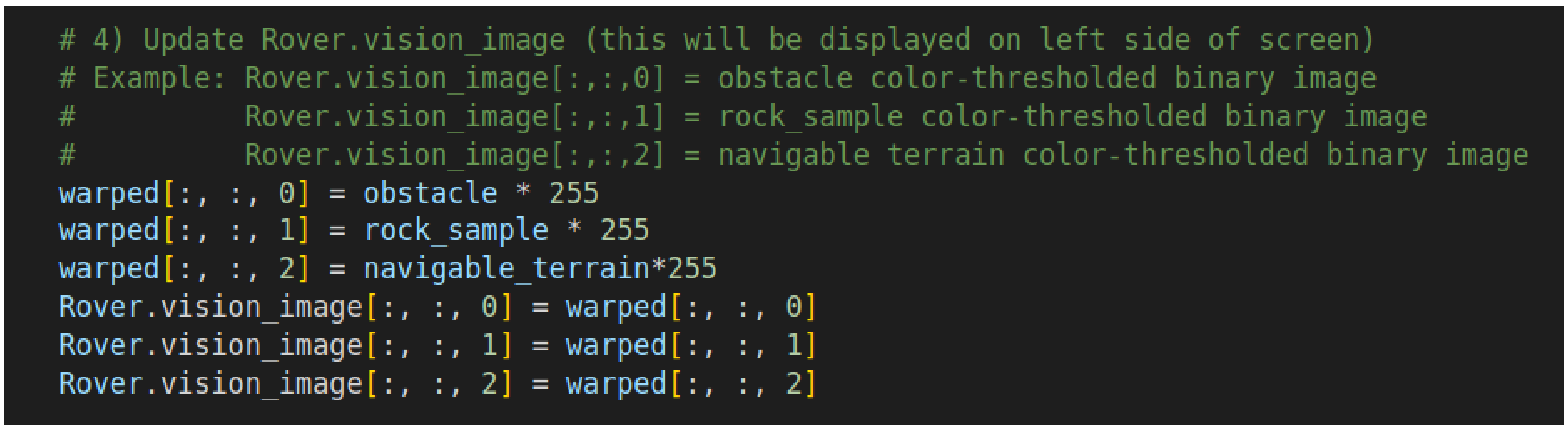
Then we defined a function to convert from image coordinates to rover coordinates where we Calculated pixel positions with reference to the rover position being at the centre bottom of the image.



5- Colored World Map:

→ it's the stage where all previous stages are summed up into coloring the world map

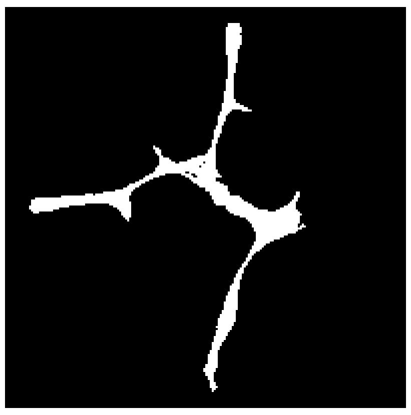
it uses the 3 color channels : Red for obstacles , Blue for terrain and all 3 for the rocks ie: appear bright

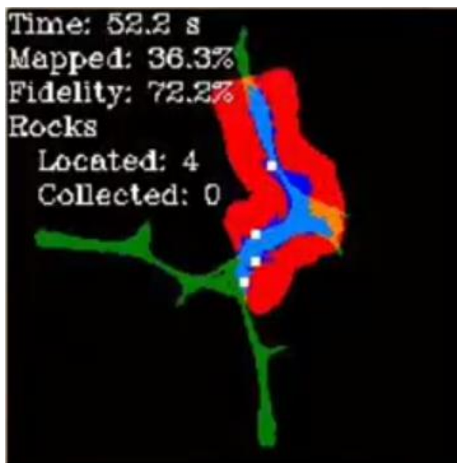

```
# 4) Update Rover.vision_image (this will be displayed on left side of screen)
# Example: Rover.vision_image[:,:,0] = obstacle color-thresholded binary image
#          Rover.vision_image[:,:,1] = rock_sample color-thresholded binary image
#          Rover.vision_image[:,:,2] = navigable terrain color-thresholded binary image
warped[:, :, 0] = obstacle * 255
warped[:, :, 1] = rock_sample * 255
warped[:, :, 2] = navigable_terrain*255
Rover.vision_image[:, :, 0] = warped[:, :, 0]
Rover.vision_image[:, :, 1] = warped[:, :, 1]
Rover.vision_image[:, :, 2] = warped[:, :, 2]
```

for the vision image :

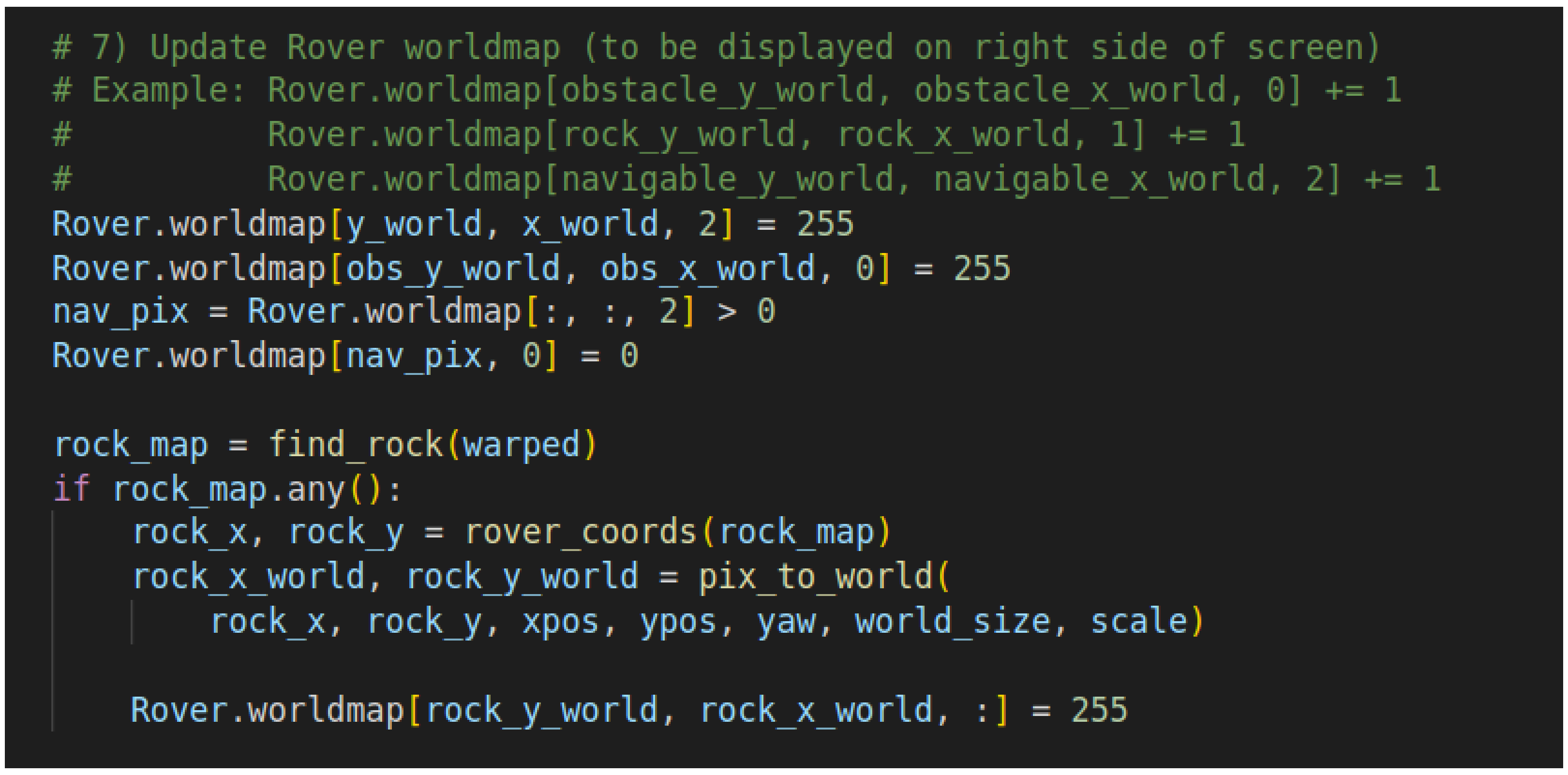
warped image channels are used to give the objects their color

to form the colored image instead of the black and white image



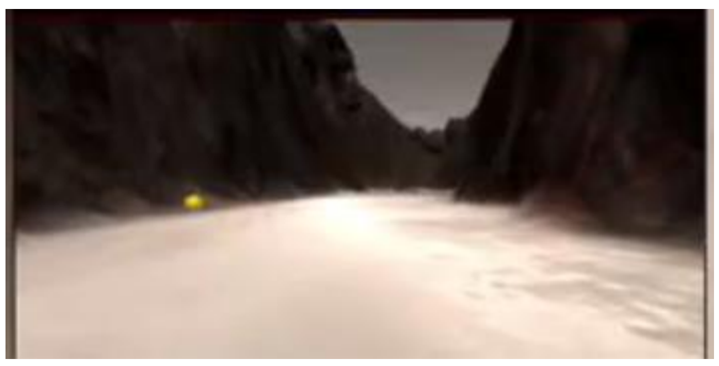
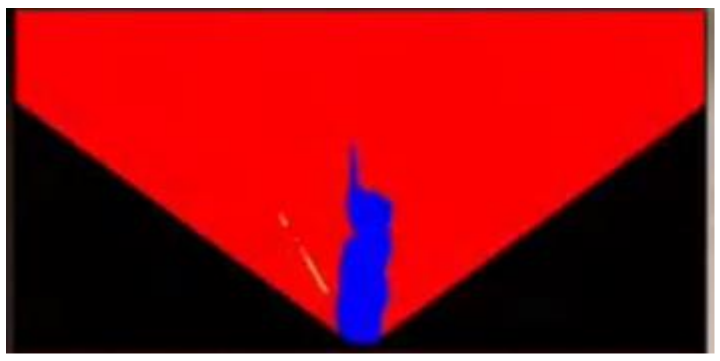


Similarly for the world image :


```
# 7) Update Rover worldmap (to be displayed on right side of screen)
# Example: Rover.worldmap[obstacle_y_world, obstacle_x_world, 0] += 1
#          Rover.worldmap[rock_y_world, rock_x_world, 1] += 1
#          Rover.worldmap[navigable_y_world, navigable_x_world, 2] += 1
Rover.worldmap[y_world, x_world, 2] = 255
Rover.worldmap[obs_y_world, obs_x_world, 0] = 255
nav_pix = Rover.worldmap[:, :, 2] > 0
Rover.worldmap[nav_pix, 0] = 0

rock_map = find_rock(warped)
if rock_map.any():
    rock_x, rock_y = rover_coords(rock_map)
    rock_x_world, rock_y_world = pix_to_world(
        rock_x, rock_y, xpos, ypos, yaw, world_size, scale)

    Rover.worldmap[rock_y_world, rock_x_world, :] = 255
```

coordinates received from the Geometric transform are used in the color thresholding functions to identify the object then address it to its right color

ie: Terrain is blue , obstacles are red and rocks are yellow.

Code walkthrough:

The decision code is based on Left wall hugging
where the steering angle is increased by an offset that is equal to 0.8 * the std of the navigation angle which will be a small value in straight lines and a big value in open areas
The rover will only walk and collect rocks on the left hand side which means that it needn't pass on the same road twice as it has never been on such a wall before and since rocks are always close to walls the probability of collecting all rocks is quite high

Modes:

The project is divided into 5 modes(stored in a stack): Forward , Stuck,Rock, Stop and End
1.Forward: the mode most often used, if navigable terrain looks good (ie there is enough terrain) and the velocity is less than maximum velocity (in our case 3m/s) it will speed up
If there isn't enough navigable terrain it will go into 'stop' mode , if it is stuck in a position for some time (in our case 4 sec) it will go into 'stuck' mode. if the rover mapped more than 92% of the map and collected 5 or more rocks it will go into 'end' mode
if everything seems normal with the rover walking at the max speed it will steer normally (between -15 and 15 degrees)

2.Stuck: if rover stuck and 1 sec passed set steering angle to mean angle(nav_angles + offset) in range -10 and +10 degrees if still stuck release the brake to allow turning since hugging left wall steering angle should be to the right so set steer angle to -10 degree
3.Rock: This mode describes what will happen when the rover detects a rock:

it steers towards the sample with steering angle equal to the mean angle

if rover is close to the sample then set mode to stop and set the brakes to pick it up

in case it gets stuck while in the rock mode (how to detect that it got stuck? Simply if the rover velocity<=0 and time ticks for more than 10 seconds then it is stuck) then go to the stuck mode

if the speed by which the rover is approaching the sample is too low (less than half of the rover's max speed) then speedup a little bit otherwise speed down a little bit to have time to pick it up.

4.Stop: if we're in stop mode but still moving keep braking , if the rover is no longer moving(v<0.2) then check if there is enough terrain and start speeding up again if there isn't enough navigable terrain then change the steering angle by -15 (Since hugging left wall steering should be to the right)

5.End: this mode is only used at the very end when the rover mission has finally been completed , it will have mapped at least 92% and collected 5 or more rocks . The mode does nothing more than break the motion of the rover .It only contains excessive brakes. this mode will make the rover stop at a distance very close to the starting position

Techniques Used to Increase Mapping Fidelity:

1. limit the measurements from the camera to 8 meters
2. Only update the map if the Rover Roll and Pitch values are around zero
3. Using The Memory Introduced to the Rover Attributes we could remember the already mapped area and not mapped it again

A. limit the measurements from the camera to 8 meters:

By limit the measurements from the camera to be mapped on the world map will help to increase the fidelity of mapping and will always mapping the front its vision by using this method:

```
def impose_range(xpix, ypix, range=80):
dist = np.sqrt(xpix**2 + ypix**2)
return xpix[dist < range], ypix[dist < range]
```

B. Only update the map if the Rover Roll and Pitch values are around zero:

Since the perspective transform method assume that the Rover is parallel to the ground, I've decided to only use the measurements where Roll and Pitch values were around zero. Anything above that would introduce significant distortion especially when the Rover bumps on some obstacle or breaks suddenly to catch a rock sample.

C. Using Memory to not map the same area twice:

Before Updating the world map, we should check that:
For example If the median of world_x_nav,world_y_nav pixels is already mapped before or the corresponding poler coordinates of them is already mapped before if that happed don’t map it again otherwise do so!.

1. Updating the obstacles in the map technique:

```
a.  if (dist_obstacle, angle_obstacle) not in Rover.memory_obstacles \
            or (x_obs_cart, y_obs_cart) not in Rover.memory_obstacles_cart:
        Rover.worldmap[obstacle_y_world, obstacle_x_world, 0] = 255
        Rover.memory_obstacles.add((dist_obstacle, angle_obstacle))
        Rover.memory_obstacles_cart.add((x_obs_cart, y_obs_cart))
```

2. Updating the rocks in the map technique:

```
a.  if (dist_rocks, angle_rocks) not in Rover.memory_rocks \
            or (x_rocks_cart, y_rocks_cart) not in Rover.memory_rocks_cart:
        Rover.worldmap[rock_y_world, rock_x_world, 1] = 255
        Rover.memory_rocks.add((dist_rocks, angle_rocks))
        Rover.memory_rocks_cart.add((x_rocks_cart, y_rocks_cart))
```

3. Updating navigable terrain in the map technique:

```
a.  if (dist_nav, angle_nav) not in Rover.memory_nav \
            or (x_nav_cart, y_nav_cart) not in Rover.memory_nav_cart:
        Rover.worldmap[navigable_y_world, navigable_x_world, 2] = 255
        Rover.memory_nav.add((dist_nav, angle_nav))
        Rover.memory_nav_cart.add((x_nav_cart, y_nav_cart))
```