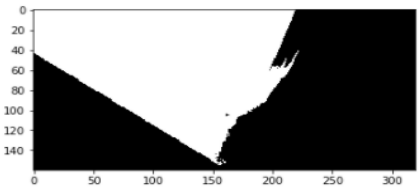
Search-and-Sample

Obstacle and rock sample detection was performed by color thresholding data from the camera. In this example code, RGB values > 160 identified ground pixels and set them to 0.

```
In [22]:
# Identify pixels above the threshold
# Threshold of RGB > 160 does a nice job of identifying ground pixels only
def color_thresh(img, rgb_thresh=(160, 160, 160)):
    # Create an array of zeros same xy size as img, but single channel
    color_select = np.zeros_like(img[:,:,0])
    # Require that each pixel be above all three threshold values in RGB
    # above_thresh will now contain a boolean array with "True"
    # where threshold was met
    above_thresh = (img[:,:,0] > rgb_thresh[0]) \
                & (img[:,:,1] > rgb_thresh[1]) \
                & (img[:,:,2] > rgb_thresh[2])
    # Index the array of zeros with the boolean array and set to 1
    color_select[above_thresh] = 1
    # Return the binary image
    return color_select

threshed = color_thresh(warped)
plt.imshow(threshed, cmap='gray')
#scipy.misc.imsave('../output/warped_threshed.jpg', threshed*255)
```

```
Out[22]: <matplotlib.image.AxesImage at 0xec785c0>
```



Process Image was used to take output images and create a world map.

Images and transformed to a top down view. Then the images are color threshed to determine ground pixels. The coordinate system is then converted to rover coordinates and then to world coordinates. Images are then added the world map.

```
In [7]:
# Define a function to pass stored images to
# reading rover position and yaw angle from csv file
# This function will be used by moviepy to create an output video
def process_image(img):
    # Example of how to use the Databucket() object defined above
    # to print the current x, y and yaw values
    # print(data.xpos[data.count], data.ypos[data.count], data.yaw[data.count])

        # First create a blank image (can be whatever shape you like)
    output_image = np.zeros((img.shape[0] + data.worldmap.shape[0], img.shape[1]*2, 3))
        # Next you can populate regions of the image with various output
        # Here I'm putting the original image in the upper left hand corner
    output_image[0:img.shape[0], 0:img.shape[1]] = img

        # Let's create more images to add to the mosaic, first a warped image
    warped = perspect_transform(img, source, destination)
        # Add the warped image in the upper right hand corner
    output_image[0:img.shape[0], img.shape[1]:] = warped

        # Overlay worldmap with ground truth map
    map_add = cv2.addWeighted(data.worldmap, 1, data.ground_truth, 0.5, 0)
        # Flip map overlay so y-axis points upward and add to output_image
    output_image[img.shape[0]:, 0:data.worldmap.shape[1]] = np.flipud(map_add)
        # Then putting some text over the image
    cv2.putText(output_image,"Populate this image with your analyses to make a video!", (20, 20),
                cv2.FONT_HERSHEY_COMPLEX, 0.4, (255, 255, 255), 1)
    if data.count < len(data.images) - 1:
        data.count += 1 # Keep track of the index in the Databucket()

    return output_image
```

Perception_step was edited in perception.py to create a world map from capture images during autonomous mode. The method is similar the process_image function in the testbook. The function also detects rocks and maps them on the world map.

Decision.py has been edited to hug the left wall when driving with:

Rover.steer = np.clip(np.mean(Rover.nav_angles * 180/np.pi) +12, -15,15)

```
        # If we're already in "stop" mode then make different decisions
        elif Rover.mode == 'stop':
            # If we're in stop mode but still moving keep braking
            if Rover.vel > 0.2:
                Rover.throttle = 0
                Rover.brake = Rover.brake_set
                Rover.steer = 0
            # If we're not moving (vel < 0.2) then do something else
            elif Rover.vel <= 0.2:
                # Now we're stopped and we have vision data to see if there's a path forward
                if len(Rover.nav_angles) < Rover.go_forward:
                    Rover.throttle = 0
                    # Release the brake to allow turning
                    Rover.brake = 0
                    # Turn range is +/- 15 degrees, when stopped the next line will induce 4-wheel turning
                    Rover.steer = -15 # Could be more clever here about which way to turn
                # If we're stopped but see sufficient navigable terrain in front then go!
                if len(Rover.nav_angles) >= Rover.go_forward:
                    # Set throttle back to stored value
                    Rover.throttle = Rover.throttle_set
                    # Release the brake
                    Rover.brake = 0
                    # Set steer to mean angle
                    Rover.steer = np.clip(np.mean(Rover.nav_angles * 180/np.pi) + 12 , -15, 15)
                    Rover.mode = 'forward'
    # Just to make the rover do something
    # even if no modifications have been made to the code
    else:
        Rover.throttle = Rover.throttle_set
        Rover.steer = 0
        Rover.brake = 0
```