In [1]:
```
%load_ext autoreload
%autoreload 2

import numpy as np
import matplotlib.pyplot as plt
import cv2

import duckietown_code_utils as dcu

%matplotlib inline
%pylab inline
```

```
DEBUG:commons:version: 6.2.4 *
DEBUG:typing:version: 6.2.3
DEBUG:geometry:PyGeometry-z6 version 2.1.4 path /usr/local/lib/python3.8/dist-packages
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
```

Image filtering

Now we want to use our image manipulation techniques to do some basic filtering of the images. We'll learn how in this activity, and at the end, you will update the file `preprocessing.py` so that we can send images with highlighted duckies to our Braitenberg agent.

In particular, let's say we want to avoid hitting duckies.

We then need to highlight our duckies in the image.

Let's start by loading a test image to work with as we did in the last activity.

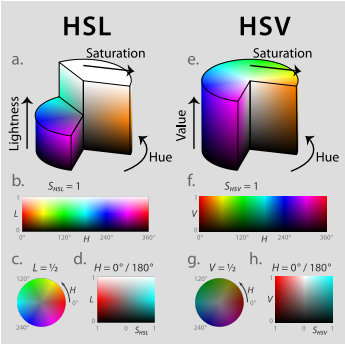
In [3]:
```
# experiment with different images!
fn = '../../assets/samples/big-duck/big-duck-08.jpg'
image = dcu.rgb_from_jpg_fn(fn)
plt.imshow(image);
```

Let's now convert it to HSV color space so that it is easier to filter by colors.

INFO:
HSL and **HSV** are the two most common cylindrical-coordinate representations of points in an RGB color model.

- HSL stands for hue, saturation, and **lightness**, and is often also called HLS.
- HSV stands for hue, saturation, and **value**, and is also often called HSB (B for brightness).
- A third model, common in computer vision applications, is HSI, for hue, saturation, and **intensity**.

In each cylinder, the angle around the central vertical axis corresponds to "hue", the distance from the axis corresponds to "saturation", and the distance along the axis corresponds to "lightness", "value" or "brightness". Note that while "hue" in HSL and HSV refers to the same attribute, their definitions of "saturation" differ dramatically.



In [4]:
```
hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
```

In OpenCV, the convention is the following:

- Hue is usually measured in degrees, 0 to 360, as shown in the scale belwo. However OpenCV uses a scale 0 to 179 - multiply by 2 to get the actual Hue in degree (so to get a hue of 180 you have to tell 90 to OpenCV.
- S and V are from 0 to 255.
  HSL_HSV

Now let's apply a simple image processing technique to highlight the region corresponding to a certain color.

To do this, we start by defining a region in HSV space with the following lower/upper bounds:

In [5]:
```
lower_hsv = np.array([171, 140, 0])
upper_hsv = np.array([179, 200, 255])
```

The two arrays have 3 components: H(ue), S(aturation), V(value).

The first array are the lower bounds; the second array are the upper bounds.

This means that the bounds we defined above will allow us to select the pixels that have:

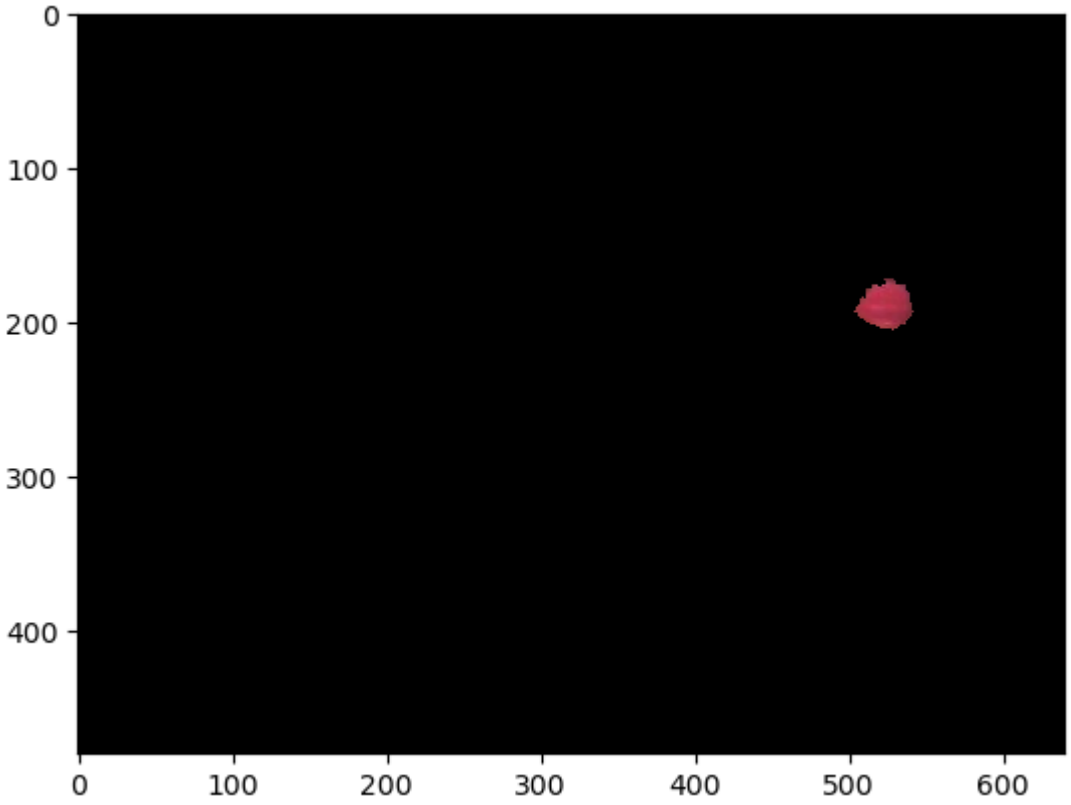
- Hue between OpenCV value 171 and 179, which are regular Hue values 342 to 358.
- Saturation between 140 and 200.
- Value between 0 and 255.

The *hue* is closest to what we intuitively call "color" and the one to use to do simple processing.

This is how to use our upper and lower bound to do the filtering of the image and show the result:

In [6]:
```
# Returns a matrix of 0 or 1 that satisfy the constraint (pixels within our
mask = cv2.inRange(hsv, lower_hsv, upper_hsv)

# Get a "masked image"
masked = cv2.bitwise_and(image, image, mask=mask)
plt.imshow(masked);
```



The bounds we just used highlight the beak. But we want the yellow of the duckie!

Activity

Your task now is to change those numbers above so that we highlight the duckies.

We've created a tool for you to make this easier and suggest you use the following steps:

1. Online color picker:
   Use this online color picker to click around on the sample images in the `/assets/samples` directory and get familiar with what color corresponds to which HSV values. (You can use any other tool, the link is just for reference). This will set you up to narrow in on the duckie yellow more quickly.
   Regardless of the color picker you use, always double-check whether it uses a different convention for HSV ranges.

If it does, make sure to convert the result to the OpenCV's convention.

2. Interactive HSV selector:
   Use the tool shown in this video to find the values interactively.
   It runs the code in HSV-bound.py to filter a test image based on the values you choose.

   To get this interactive method running, open a terminal on your computer, **navigate to the exercise folder** (`duckietown-lx/braitenberg/`) and type:

   ```
   dts code build
   ```

   to build your updated exercise. Then use

   ```
   dts code workbench --sim -L HSV
   ```

   to run the HSV tool.
   In the terminal output you will see lines like the following (you may have to scroll up in the terminal)

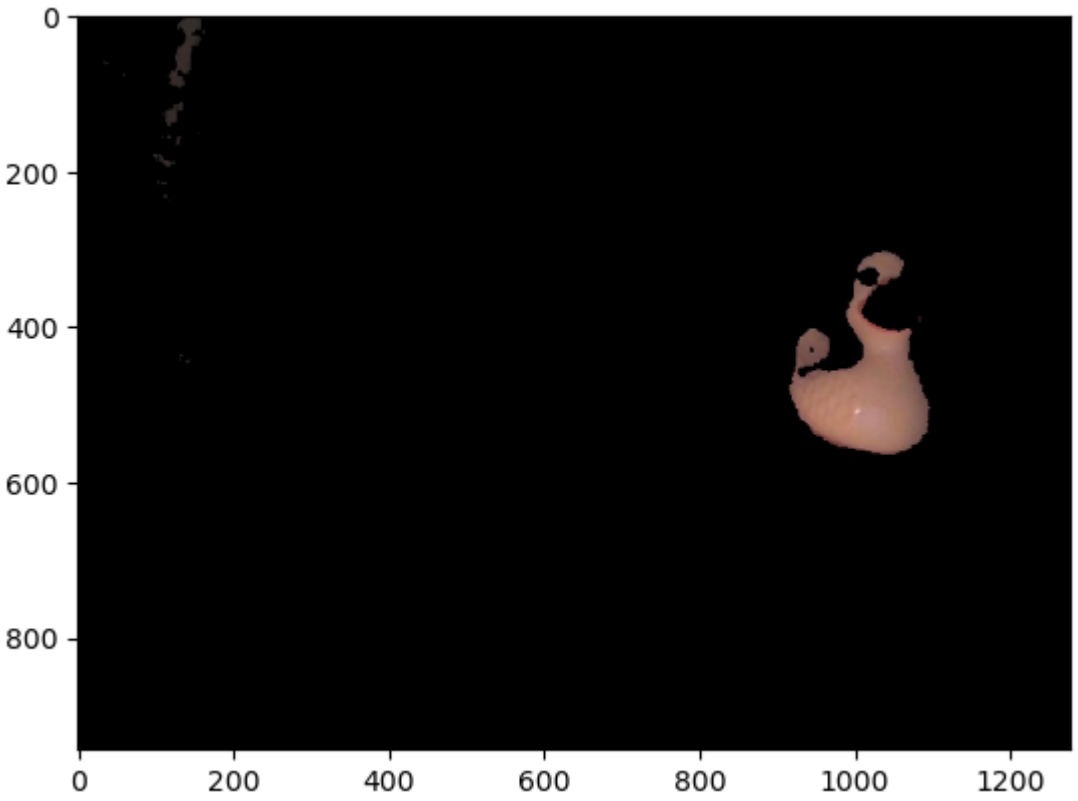
   ```
   VNC running at http://localhost:8087/
   ------------------------------------------------------------
   ```

   Following the link will bring up the VNC desktop, where you can click on the HSV activity icon and run the tool.

3. Commit lower & upper values:
   After you have a satisfactory result in the VNC tool, open the file preprocessing.py, and change the values of `lower_hsv` and `upper_hsv` to the ones that you found to highlight the duckie.
   This will be used by your Braitenberg agent in the next notebook.

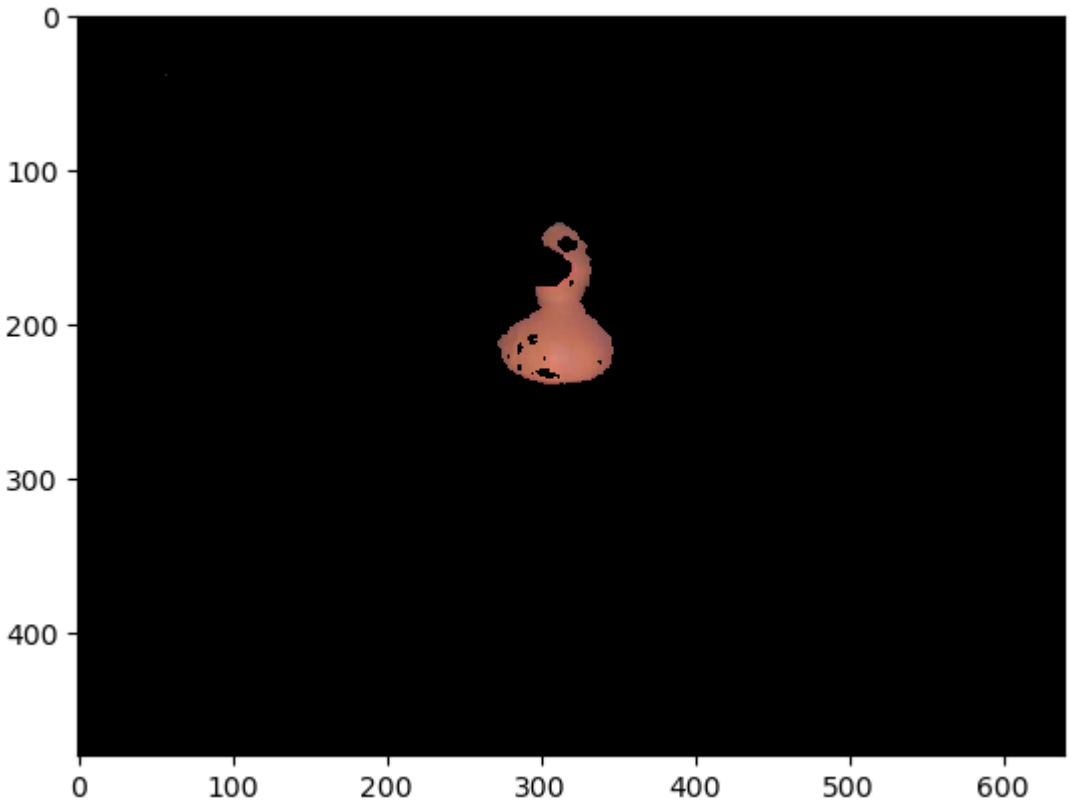
For reference, this is what your end result should look like.

In [7]:
```
result_img = '../../assets/samples/result.jpg'
result = dcu.rgb_from_jpg_fn(result_img)
plt.imshow(result);
```



In [58]:

```
# TODO
# Load images!
fn = '../../assets/samples/big-duck/big-duck-03.jpg'
image = dcu.rgb_from_jpg_fn(fn)
hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
lower_hsv = np.array([0, 13, 60])
upper_hsv = np.array([22, 140, 255])
# Returns a matrix of 0 or 1 that satisfy the constraint (pixels within our
mask = cv2.inRange(hsv, lower_hsv, upper_hsv)
# Get a "masked image"
masked = cv2.bitwise_and(image, image, mask=mask)
plt.imshow(masked);
```



In [ ]:

```
import sys

# import duckietown_code_utils as dcu
import numpy as np
import cv2

def nothing(x):
    pass

def main(fname: str = None):
    # Create a window
    cv2.namedWindow("image")
    # frame0 = dcu.image_cv_from_jpg_fn(fname or sys.argv[1])
    frame0 = cv2.imread(fname)
    lastL = np.array([0, 0, 0])
    lastU = np.array([255, 255, 255])

    # create trackbars for color change
    cv2.createTrackbar("lowH", "image", lastL[0], 179, nothing)
    cv2.createTrackbar("highH", "image", lastU[0], 179, nothing)

    cv2.createTrackbar("lowS", "image", lastL[1], 255, nothing)
    cv2.createTrackbar("highS", "image", lastU[1], 255, nothing)

    cv2.createTrackbar("lowV", "image", lastL[2], 255, nothing)
    cv2.createTrackbar("highV", "image", lastU[2], 255, nothing)
    while True:
        frame = frame0
        # get current positions of the trackbars
        ilowH = cv2.getTrackbarPos("lowH", "image")
```

```
        ihighH = cv2.getTrackbarPos("highH", "image")
        ilowS = cv2.getTrackbarPos("lowS", "image")
        ihighS = cv2.getTrackbarPos("highS", "image")
        ilowV = cv2.getTrackbarPos("lowV", "image")
        ihighV = cv2.getTrackbarPos("highV", "image")
        # convert color to hsv because it is easy to track colors in this co
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        lower_hsv = np.array([ilowH, ilowS, ilowV])
        higher_hsv = np.array([ihighH, ihighS, ihighV])
        if not np.allclose(lastL, lower_hsv) or not np.allclose(lastU, highe
            print(f"lower {lower_hsv} upper {higher_hsv}")
            lastL = lower_hsv
            lastU = higher_hsv

        # Apply the cv2.inrange method to create a mask
        mask = cv2.inRange(hsv, lower_hsv, higher_hsv)

        # Apply the mask on the image to extract the original color
        frame = cv2.bitwise_and(frame, frame, mask=mask)
        cv2.imshow("image", frame)

        # Press q to exit
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break
    # close windows
    cv2.destroyAllWindows()

if __name__ == "__main__":
    fn = 'assets/samples/big-duck/big-duck-09.jpg'
    main(fn)
```

Now go on to the third notebook.