Autonomous Duckieking: Ball Detection and Following

GROUP 6

Nitin Kumar Bhagat, Maurice Yan KAYIJAMAHE, Mohamed Amine Kina

JACOBS UNIVERSITY BREMEN

SPRING 2022

CO-548-A: RIS PROJECT

Alanwar Abdelhafez, Amr

1. Introduction:

1.1 Mission and Scope

The objective of this project was to implement a method that allows Duckiebots to reliably detect round/circular objects they may encounter when driving through a regular Duckietown.

What is in the scope:

- Navigation while following a circle or a round object.
- Signal provided while sending the data from the dashboard.
- Tracking the circle which is done using the hough transform.
- MinDist and MaxDist parameters for the HoughCircles function.
- Stop signal when the circular object is out of sight.

1.2 Problem Formulation

We seek to find a method that allows a Duckiebot to identify a specific object which is supposed to be circle in shape, in this instance a ball. In particular, we attempt to solve the following problem: (Given a duckiebot camera has a few objects in front of it can identify a ball and can be able to follow it)

1) Initial localization of the rest position,
2) Determine the position of the grid of the object by taking a frame of an image and a center of a circle,
3) Continuous following with publisher to wheel commands and pose updates from Subscriber class that takes images and detect circles, the same method of the initial localization is used frame by frame.

2. Solution Approach

We are using the Hough Transforms algorithm to detect the circle. We are using the implementation from cv2 to detect the number of circular objects in the camera setting. Considering the object detecting and vision part of the duckiebot, we are taking a ball as a round object(circle) that can be detected and make the duckieking follow the ball as long as it can see it.

2.1 Hough Transforms EXPLAINED!!!

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure.

The Hough transform takes a binary edge map as input and attempts to locate edges placed as straight lines. The idea of the Hough transform is that every edge point in the edge map is transformed to all possible lines that could pass through that point. However, Circle Hough Transform(CHT) has been implemented to detect a ball which has a circular view. We hence implement the HoughCIrcles() function of cv2 to detect the ball(circle) that we see on the dashboard screen.

Assumptions

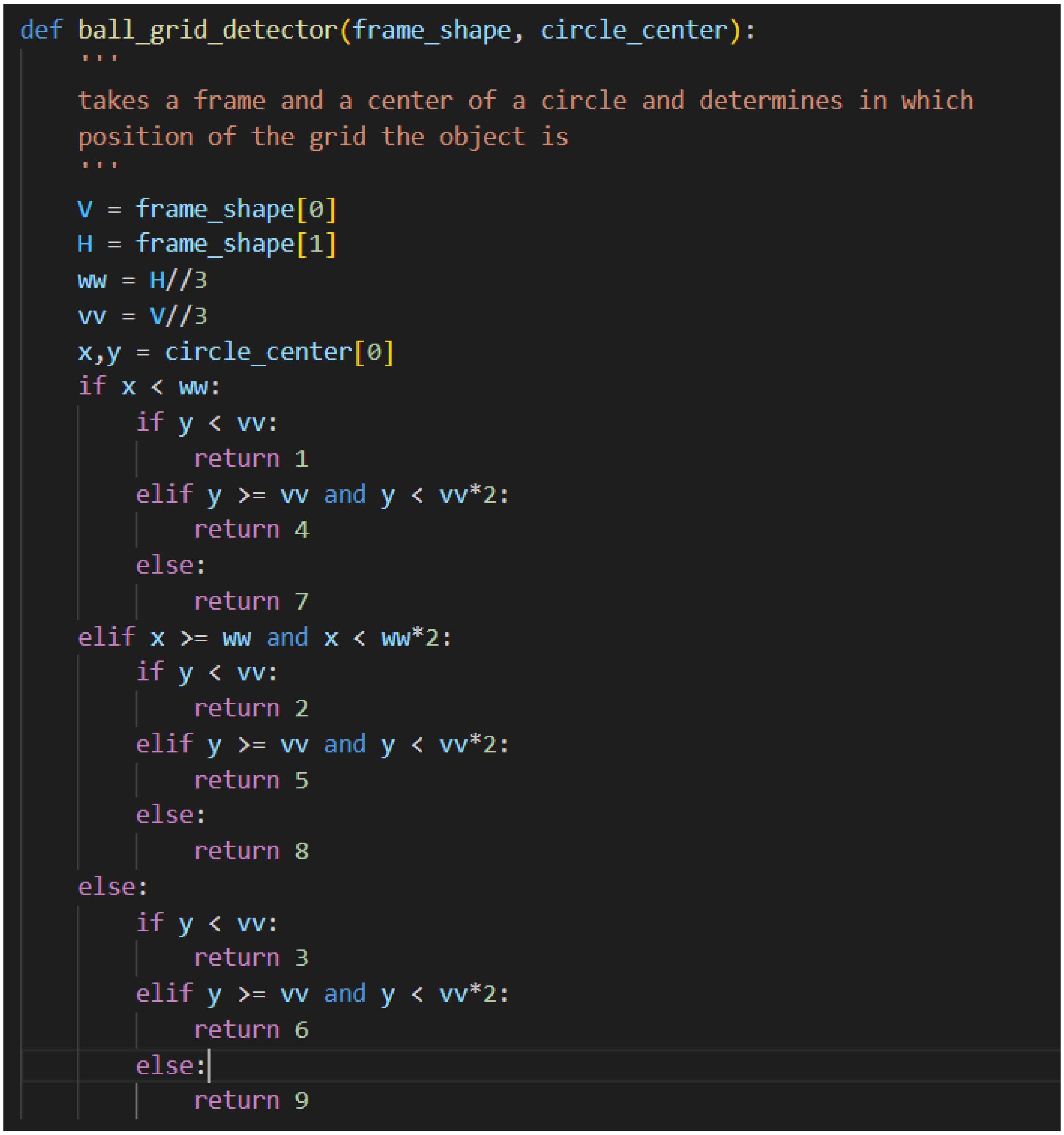
- Size, shape of grid are given and fixed.
- Color and size of field of view are given and fixed.
- A fairly minimal number of obstacles.
- Good light conditions (e.g. no illumination problems, ...)

2.2 Implementation of Solution (CODE)

Our approach in implementing the idea of making the duckieking follow the round object is basically divided into 2 parts:

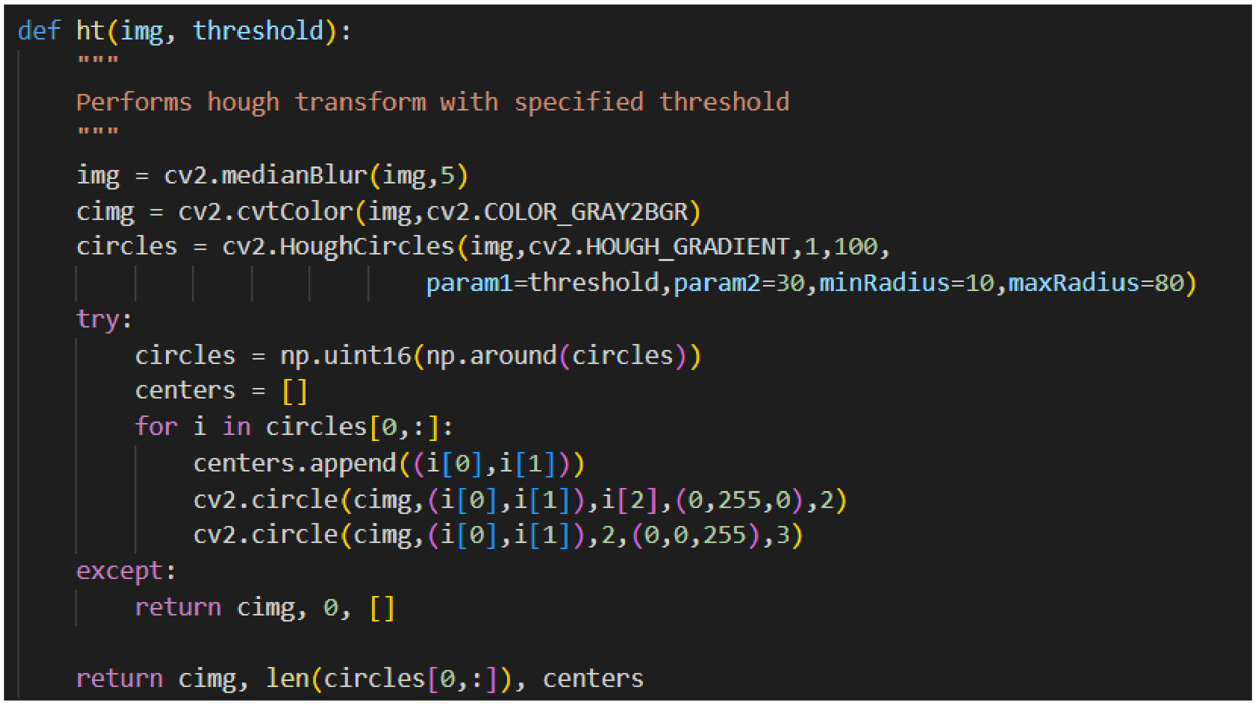
2.2.1 ‘Utils.py’ script contains the code for the algorithms used in the project. We used python in the implementation of the Hough Transformation and the execution and testing as well, attached are images of the hough’s concept and a snippet of the implemented code.

Since, the Hough Circle Transform works in a roughly analogous way to the Hough Line Transform, the application of the function is similar to that of Line detection. In this case, the algorithm determines the circle with three parameters where center position and radius can be determined.



```python
def ball_grid_detector(frame_shape, circle_center):
    '''
    takes a frame and a center of a circle and determines in which
    position of the grid the object is
    '''
    V = frame_shape[0]
    H = frame_shape[1]
    ww = H//3
    vv = V//3
    x,y = circle_center[0]
    if x < ww:
        if y < vv:
            return 1
        elif y >= vv and y < vv*2:
            return 4
        else:
            return 7
    elif x >= ww and x < ww*2:
        if y < vv:
            return 2
        elif y >= vv and y < vv*2:
            return 5
        else:
            return 8
    else:
        if y < vv:
            return 3
        elif y >= vv and y < vv*2:
            return 6
        else:
            return 9
```

We have the function to calculate the center of the circle with respect to the frame for determining the actual position of the circle for implementing the Hough Transform. x,y < ww and vv given the fact that it lies in between those distances.



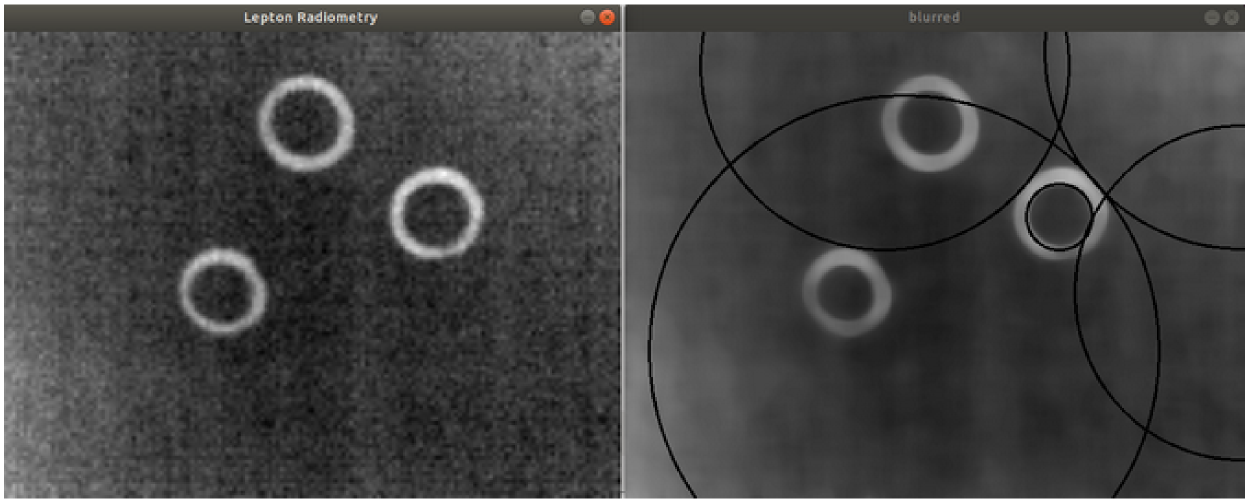
```python
def ht(img, threshold):
    """
    Performs hough transform with specified threshold
    """
    img = cv2.medianBlur(img,5)
    cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
    circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,100,
                                param1=threshold,param2=30,minRadius=10,maxRadius=80)
    try:
        circles = np.uint16(np.around(circles))
        centers = []
        for i in circles[0,:]:
            centers.append((i[0],i[1]))
            cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
            cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)
    except:
        return cimg, 0, []

    return cimg, len(circles[0,:]), centers
```

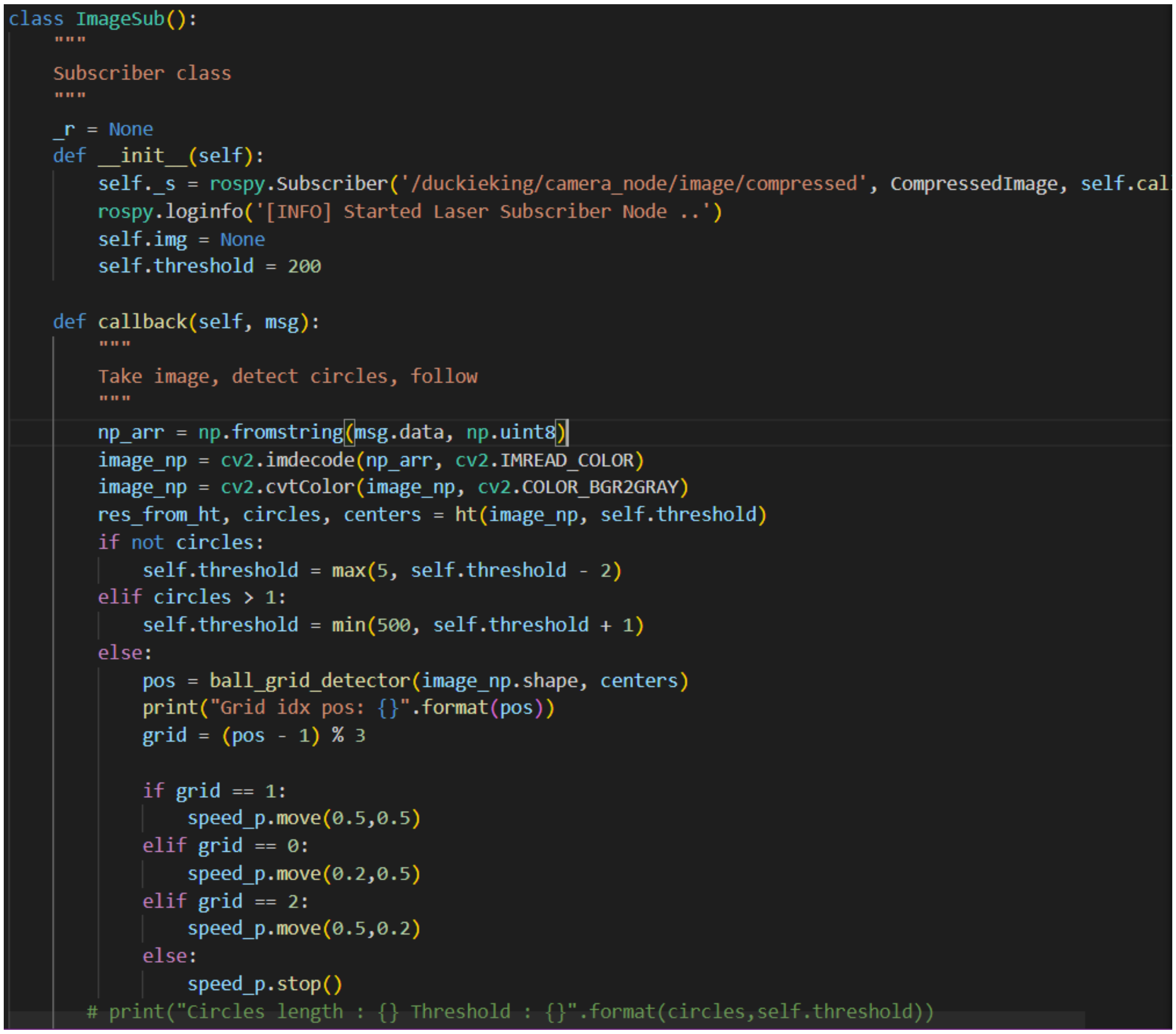
With the arguments:

Img: Applying median blur in order to reduce noise and avoid false circle detection
Cimg: converting it to grayscale
Circles: this function is the main application for detecting the circles - cv2.HOUGH_GRAIENT defines the detection method using cv library
Those parameters are necessary for the accuracy in detection among many circles, minRadius and maxRadius as the accumulator value of min and max distance.

Below can be seen an image where the right one has the detected result which is comparably more smooth i.e less noise and detects circles around the round objects:



2.2.2 ‘main.py’ script contains the code for the publisher and subscriber class implemented that enables the access of the wheel to follow the object(publish) as soon as it subscribes to the image view. We have the subscriber class that gets the input parameter of CompressedImage followed by a callback function for determining the grid of the object with the help of the function we take from ‘utils.py’ script. After it gets the position as a grid, it moves and stops as a certain condition is provided.

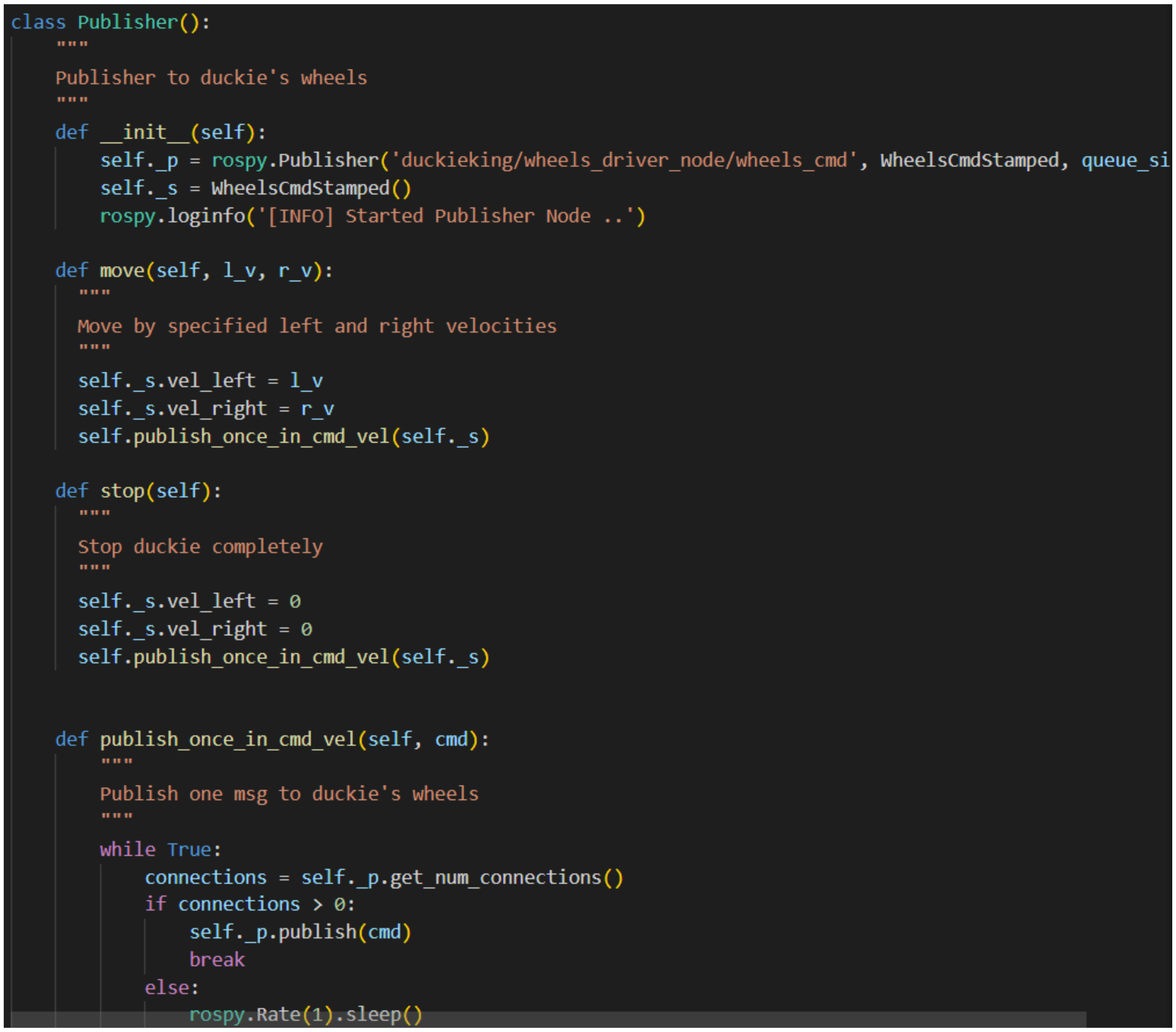


```
class ImageSub():
    """
    Subscriber class
    """
    _r = None
    def __init__(self):
        self._s = rospy.Subscriber('/duckieking/camera_node/image/compressed', CompressedImage, self.cal
        rospy.loginfo('[INFO] Started Laser Subscriber Node ..')
        self.img = None
        self.threshold = 200

    def callback(self, msg):
        """
        Take image, detect circles, follow
        """
        np_arr = np.fromstring(msg.data, np.uint8)
        image_np = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
        image_np = cv2.cvtColor(image_np, cv2.COLOR_BGR2GRAY)
        res_from_ht, circles, centers = ht(image_np, self.threshold)
        if not circles:
            self.threshold = max(5, self.threshold - 2)
        elif circles > 1:
            self.threshold = min(500, self.threshold + 1)
        else:
            pos = ball_grid_detector(image_np.shape, centers)
            print("Grid idx pos: {}".format(pos))
            grid = (pos - 1) % 3

            if grid == 1:
                speed_p.move(0.5,0.5)
            elif grid == 0:
                speed_p.move(0.2,0.5)
            elif grid == 2:
                speed_p.move(0.5,0.2)
            else:
                speed_p.stop()
      # print("Circles length : {} Threshold : {}".format(circles,self.threshold))
```

Finally, we have the integration or the data transfer to the wheel in order for it to react with the publisher to duckie's wheel. This basically is the node for the duckiebot to integrate the input to the backend to have wheels command running as an output. We have functions for the specific speed of the left and right wheel. So, given the velocity of the movement = 0, the wheel stops and so does the duckiebot.



```
class Publisher():
    """
    Publisher to duckie's wheels
    """
    def __init__(self):
        self._p = rospy.Publisher('duckieking/wheels_driver_node/wheels_cmd', WheelsCmdStamped, queue_si
        self._s = WheelsCmdStamped()
        rospy.loginfo('[INFO] Started Publisher Node ..')

    def move(self, l_v, r_v):
        """
        Move by specified left and right velocities
        """
        self._s.vel_left = l_v
        self._s.vel_right = r_v
        self.publish_once_in_cmd_vel(self._s)

    def stop(self):
        """
        Stop duckie completely
        """
        self._s.vel_left = 0
        self._s.vel_right = 0
        self.publish_once_in_cmd_vel(self._s)


    def publish_once_in_cmd_vel(self, cmd):
        """
        Publish one msg to duckie's wheels
        """
        while True:
            connections = self._p.get_num_connections()
            if connections > 0:
                self._p.publish(cmd)
                break
            else:
                rospy.Rate(1).sleep()
```

We have the ‘duckietown_msgs.msg’ package in the folder with main.py from where ROS2 message and service definitions are used in Duckietown.
REF: https://github.com/nlimpert/duckietown_msgs

3. Further Application of Solution

This autonomous duckieking that detects and follows the round object offers the ground idea and basis for the potential projects that can be built upon it or similar field projects.

This could be applied in real life in instances like football footage collection by using a trained rover on the pitchside that tracks the ball in a field with predetermined camera settings to record the football game and this would be a better option compared to manually manned cameras on pitchside.

It has potential in many field like tourism(for e.g. taking footage of hot air balloons for tourism industry), entertainment(e.g. Cricket or baseball matches where the match moves around a ball or round object), detecting number of people(head) in crowd, etc.

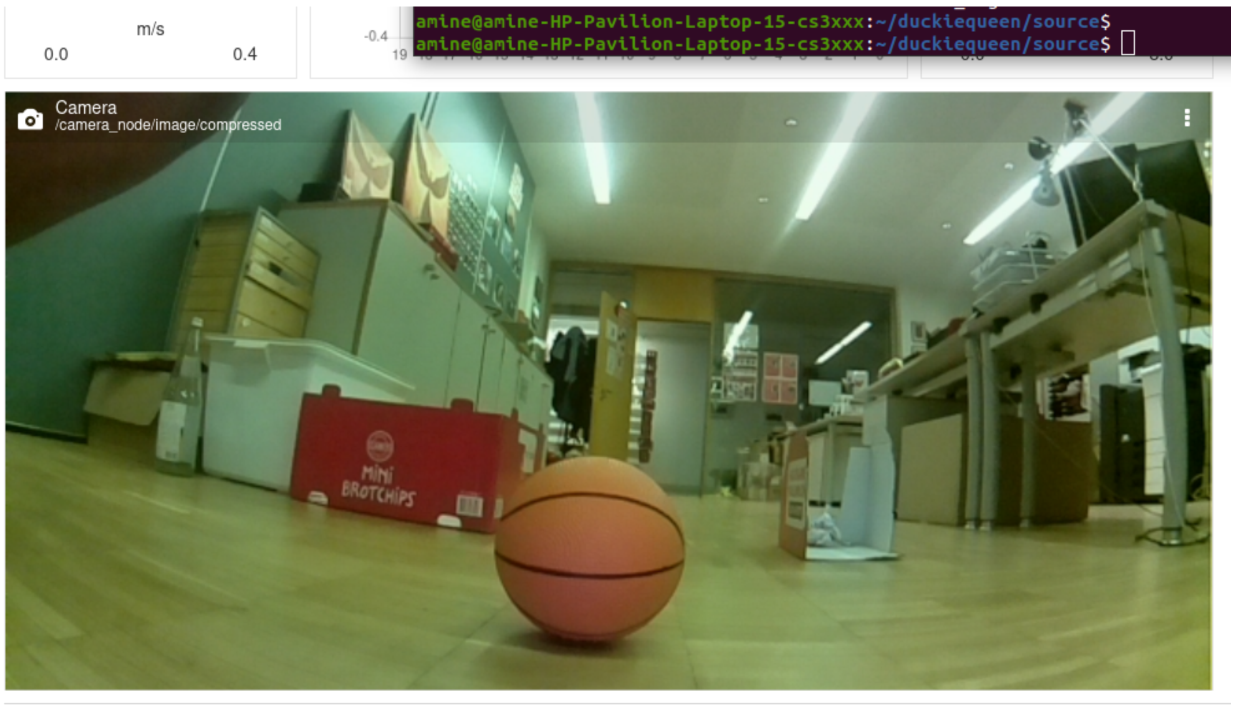
Some necessary improvements:

1. The camera with higher dynamic range and higher pixels.
2. The hough transform should be able to detect the circle at a very far distance. So, we need to acquire accuracy with its implementation at a more advanced level.
3. Should not require lots of storage and computation (since it's 3 dimensional).
4. Robustness to noise is a necessity.

4. The final result:

Video of the final result: TheHough transform detection
The link is attached as follows(also uploaded on github):
https://drive.google.com/file/d/1tby0ArOWmqUB0EClwud2TsRSJ6CEXYj5/view?ts=629429ac

Also the screenshot from the dashboard while the camera detects the round orange ball is attached as follows:



```
amine@amine-HP-Pavilion-Laptop-15-cs3xxx:~/duckiequeen/source$
amine@amine-HP-Pavilion-Laptop-15-cs3xxx:~/duckiequeen/source$
```

5. Conclusion

The implementation of the circle hough transform algorithm can hence be used to detect the circle in the noisy or imperfect images as a solution to various problems and also has potential developments in different industries. The solution, as simple as it may sound, has more compound implementation in real life and would take more than 3months to develop and implement in a stable python environment with the best efficiencies. However, this model can be further enhanced given more emphasis on the visibility and accuracy regarding the distant image capture. You can be run using the instruction in ‘readme’ easily and also you can refer to the video for the result.