Parking with GMapping, AMCL, and ARTags

Jordan Lane and Brad Ofrim

1. PURPOSE

Parking with GMapping, AMCL, and ARTags is a competition consisting of a race-track shaped obstacle course and a single Turtlebot. The obstacle course consists of a white line with multiple red lines to indicate either a stopping location or the location of a specific task that the Turtlebot must complete. The Turtlebot is scored based on the robot's performance at each task and the general traversal of the obstacle course.

This report aims to explain the competition itself, the logic behind the Turtlebot implementation, and how to run the ROS Package on the Turtlebot 2.

2. LAYOUT AND SCORING

The layout of the course consists of a white line, four stop lines, and three locations. The task at each location is optional, however, the Turtlebot will earn no points for any skipped locations. To complete the course, the Turtlebot must do a full loop of the course both starting and ending at the start line.

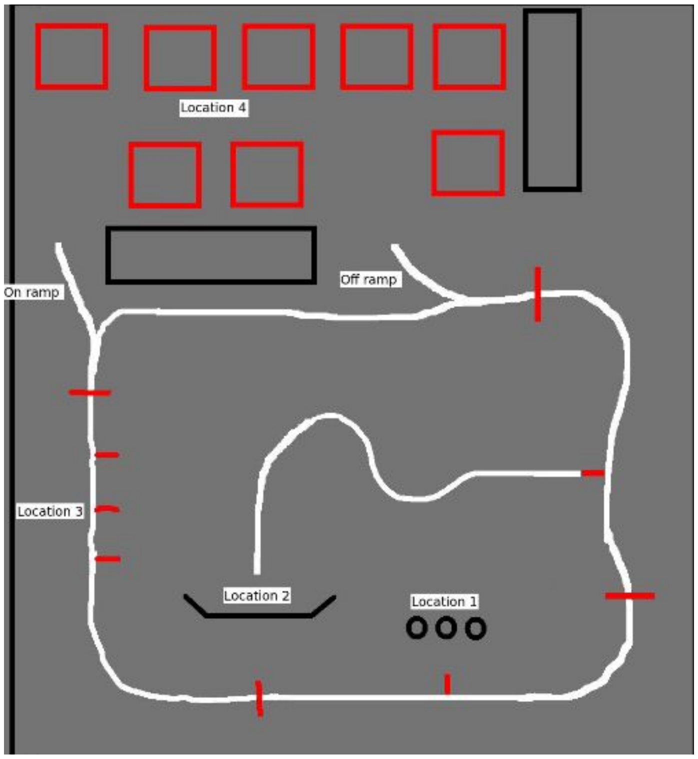

Figure 1: The Race Course

2.1 Location 1

Marked by a short red line, Location 1 is the first task on the course that the robot can complete. The first location is an object counting task, where the Turtlebot must count the number of objects 90° counterclockwise to the track. After counting, the Turtlebot must convey its results to the user with its onboard led lights and speakers.

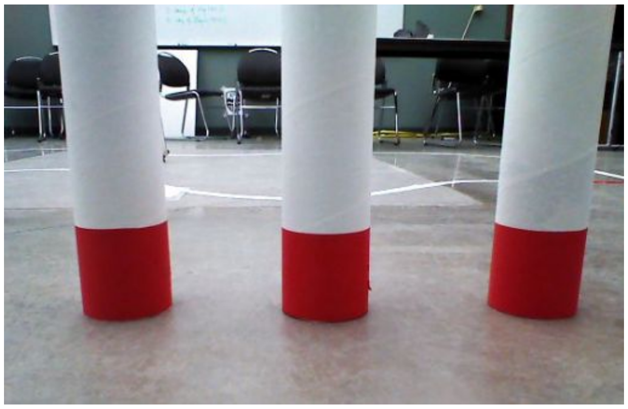

Figure 2: Objects for Location 1

2.2 Location 2

Location 2 is at the end of a small detour marked by another short red line. To get to Location 2, the Turtlebot must make a 90° turn at the small red-line, and follow a white line that branches off the original track.

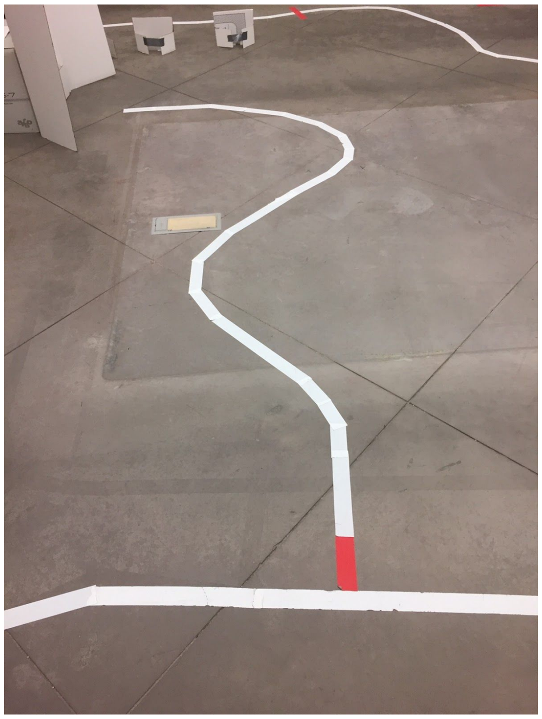

Figure 2: Detour to Location 2

After following the detour line, the Turtlebot arrives at Location 2; a white display board with two red shapes and one green shape. Similar to Location 1, the Turtlebot must count the number of shapes and output the result to the user. Also, the Turtlebot must remember the shape of the green object if it plans on performing the task at Location 3. After completing the second task, the Turtlebot must make a 180° turn and follow the white line to return to the original track.

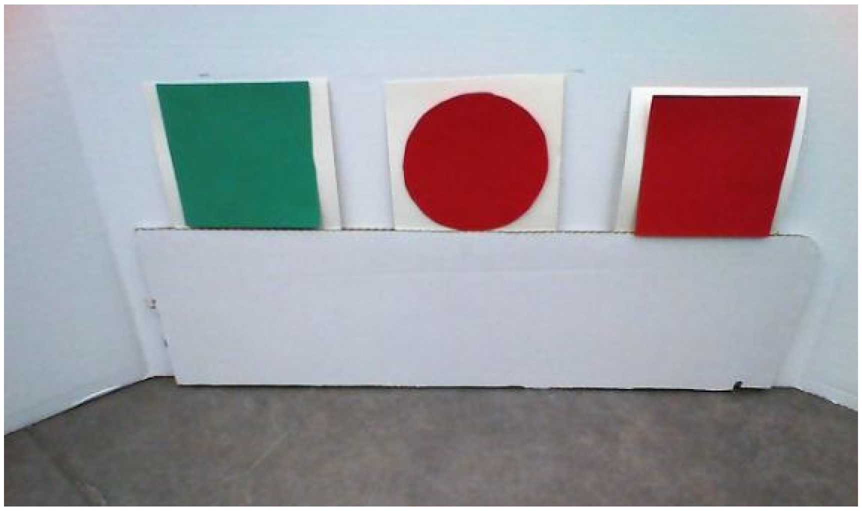

Figure 3: Location 2 Task

2.3 Location 4

Location 4 is located at the end of the white line off ramp. Unlike the other locations, Location 4 is an open area with no white line to guide the robot. The location consists of eight parking spots, which are red squares numbered one through eight.

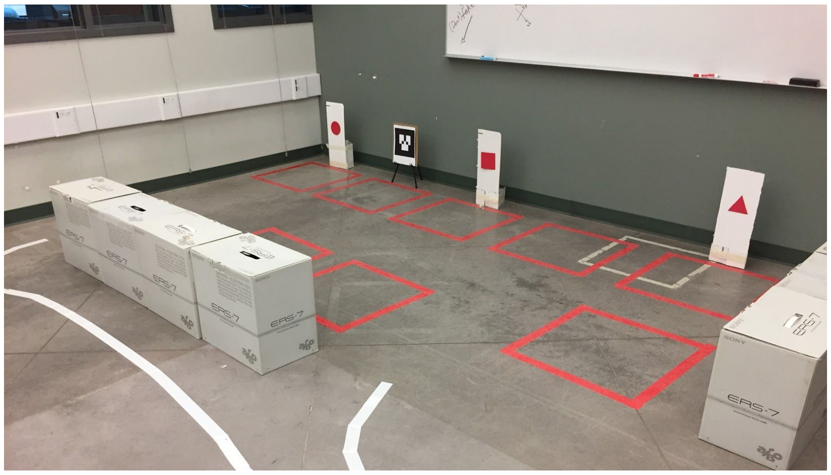

Figure 4: Location 4

Location 4 consists of three parking tasks. The first task is parking the robot in a parking spot with an AR Tag. After parking, the Turtlebot must indicate that it is in the AR parking spot by turning an LED green.

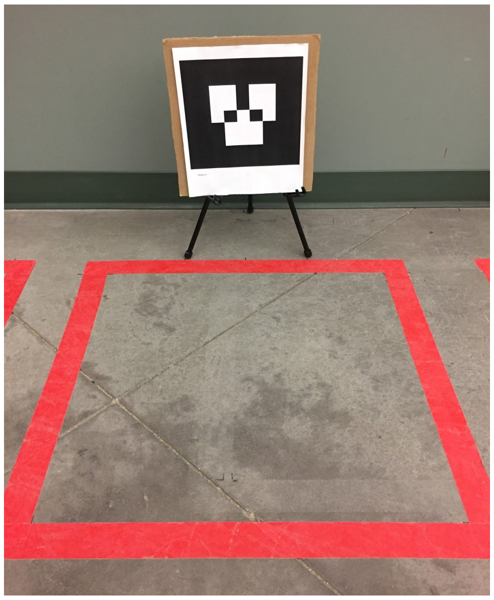

Figure 5: AR Tag Parking Spot

The second task of Location 4 consists of parking the robot in an empty parking spot that is specified at the beginning of the competition. After parking, the Turtlebot must indicate that it is in the empty parking spot by turning an LED red.

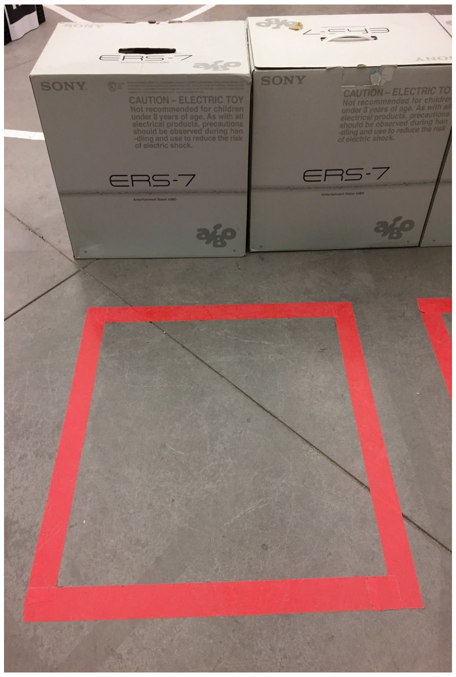

Figure 6: Empty Parking Spot

The third task of Location 4 is an object recognition task that requires the Turtlebot to park at a parking spot with the same shape identified at Location 2. After parking, the Turtlebot indicate that it is in the shape parking by turning an LED orange. Due to time restrictions, this task was omitted from our final traversal of the course.

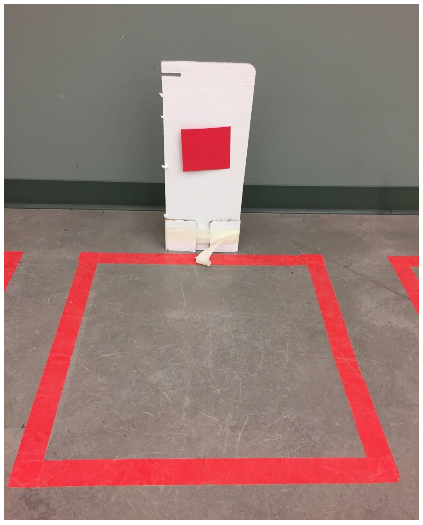

Figure 7: Shape Parking Spot

Points for Location 4 are given based on parking in the correct spot, and parking within the red lines. After completing the tasks for Location 4, the Turtlebot must navigate to the on-ramp and continue the course.

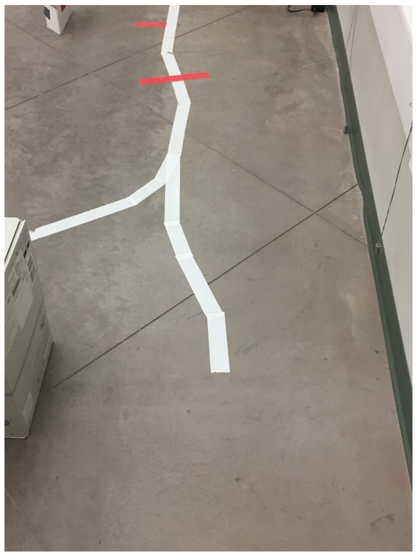

Figure 8: On Ramp

2.4 Location 3

The final location of the track is another object recognition task, that requires the Turtlebot to have previously completed the task at Location 2. Location 3 consists of three small red lines, with a red shape placed 90° counterclockwise to each red line. To complete the task, the Turtlebot must scan the object at each red-line, and indicate if it is the same shape as the green object at Location 2.

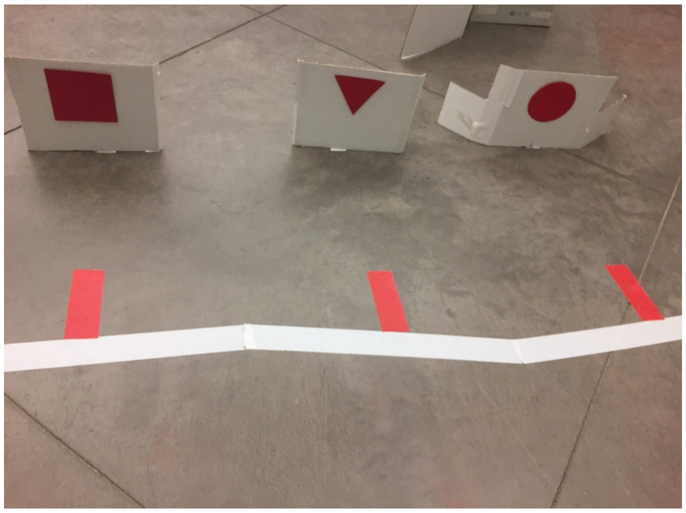

Figure 9: Location 3 Task

3. PRE-REQUISITES

This project is built using python 2.7, ros-kinetic, ros-kinetic kobuki, and turtlebot libraries for Ubuntu Xenial 16.04. If these are not installed, please refer to their installation pages on the official ROS wiki or the official python installation page.

- http://wiki.ros.org/kobuki/Tutorials/Installation/kinetic
- http://wiki.ros.org/kinetic/Installation
- https://www.python.org/downloads/

The source code for our project can be found at https://github.com/bofrim/CMPUT_412

Create or navigate to an existing catkin workspace and clone our repository. Our repository consists of multiple catkin packages so you will need to copy the files (or just the packages you want; in this case, “comp3”) from our repository into your catkin workspace.

4. EXECUTION

Open a console and source the catkin workspace that the package was placed. For example, if your workspace was called catkin_ws:

```
$ cd catkin_ws/
$ source ./devel/setup.bash
```

Next, build the project:

```
$ catkin_make
```

4.1 Turtlebot Racer

Before starting the Turtlebot, ensure that it is placed on the track behind the red start line.

```
$ cd catkin_ws/
$ roslaunch comp3 comp3.launch [video_device]
(in another terminal)
$ rosrun comp3 ultra_sm.py [parking_spot]
```

The comp3 launch file is launched first with the optional video_device parameter. The video_device parameter specifies the camera device that will be used for line navigation.

Next, we run the navigation statemachine *ultra_sm.py* and input the number of the empty parking spot that we want our rover to park in. The Turtlebot will now start to navigate the course.

5. CONCEPTS AND CODE

The Turtlebot racer consists of a state machine, an incoming data path, and an outgoing data path. The state machine consists of general driving nodes, a red-line decision node, and specific nodes for each location task. These nodes are assisted by image recognition nodes that are used to detect the white and red line centroids, as well as general nodes that control the leds, sounds, and motors of the Turtlebot.

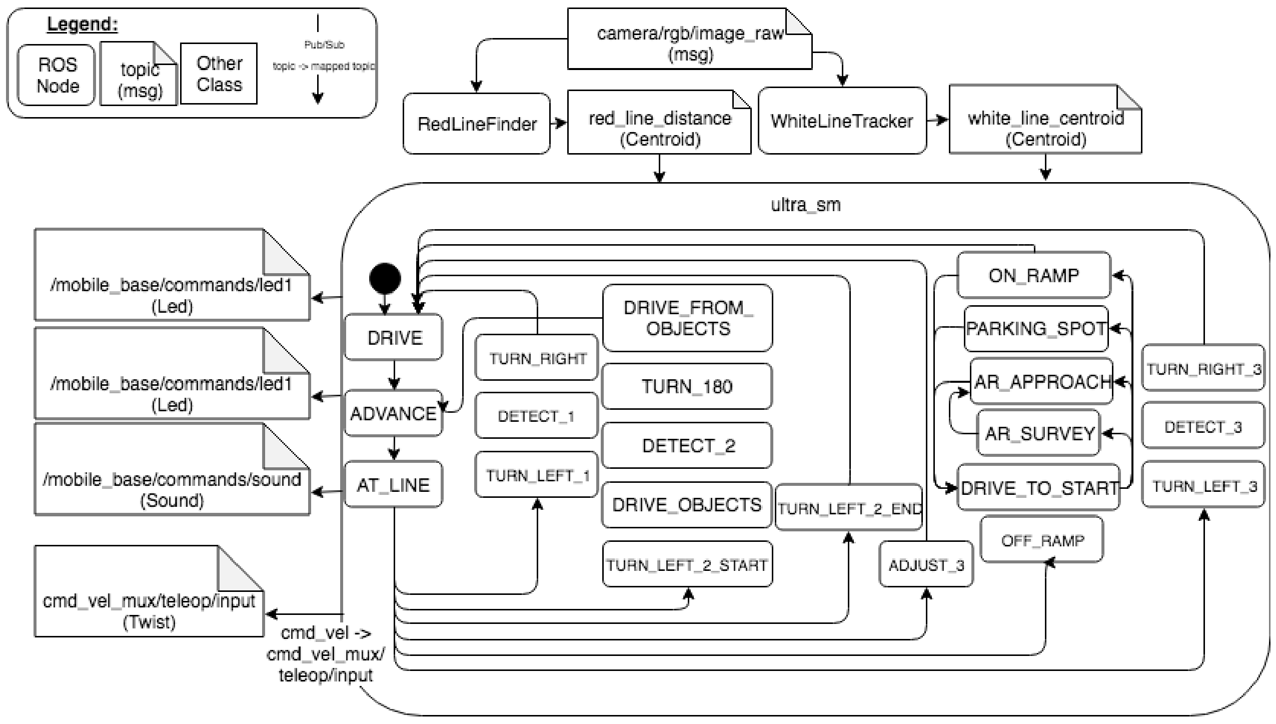

Figure 10: UltraSM Nodes

5.1 Course Navigation States

The general driving states consist of the Drive and Advance nodes. These states subscribe to topics published by the RedLineFinder and WhiteLineFinder nodes. This data is used to follow the white line and advance onto the various red lines of the course.

At each red line, the Turtlebot enters the key decision making state: the AtLine node. The AtLine state keeps track of how many red lines have been encountered, and decides if the Turtlebot needs to stop, or perform a specific task.

5.2 Location 1 States

Location 1 consists of three states: TurnLeft1, Detect1, and TurnRight1. The TurnLeft1 and TurnRight1 states are used to turn the robot towards the objects and back to the track.

Detect1 uses OpenCV to detect the number of red objects within the image and output the result using the Turtlebot's LEDs and speaker.

5.3 Location 2 States

Location 2 consists of six unique states: TurnLeft2Start, DriveObjects, Detect2, Turn180, DriveFromObjects, and TurnLeft2End.

The detour of Location 2 is unique from the rest of the track due to the white line ending with no red line. The detour line was also the most common location for the Turtlebot to lose the line due to glare or other various factors. Therefore Location2 has two unique drive functions: DriveObjects, and DriveFromObjects. These two drive states are similar to the general drive states, however, DriveObjects will follow the white line until it detects multiple shapes within the camera feed. DriveFromObjects is similar to the general drive state, however, it corrects for the vertical red line and corrects the Turtlebot if it loses the white line.

TurnLeft2Start, Turn180, and TurnLeft2End are generic turning nodes that are used to enter, leave, and navigate Location 2's detour.

Detect2 is the main image detection state used to detect the various shapes at Location 2. When the Turtlebot arrives at the whiteboard, it processes images to count the total number of shapes and attempt to identify the green shape. In order to detect the green shape with reasonable confidence, the system processes images until it repeatedly detects the same shape for a few consecutive images and records this shape for later. If it does not settle on a decision after a maximum allowed number of images were processed, it chooses the shape that it thought it saw the most and records this shape for later.

5.4 Location 4 States

Location 4 states include "Off Ramp", "Drive To Start", "AR Survey", "AR Approach", "Parking Spot", and "OnRamp".

Drive To Start is the initial state, and also acts as a middle state between tasks. The Drive To Start has a waypoint in the middle of Location 4 which acts as a starting point for all tasks at Location 4.

The AR Survey state is the start of the AR task. In the AR Survey state, the robot attempts to locate an AR tag. To do this, it slowly turns in place until it finds a valid AR tag within the camera's field of view. Once a tag is detected it transitions to the AR Approach state.

In the AR Approach state, the robot uses the "alvar" ROS package to find the pose of the AR tag relative to the robot. Once an AR Tag has been discovered, the Turtlebot begins driving straight towards the AR Tag until it reaches a set distance from the tag. The Turtlebot then calculates the pose of the AR Tag and creates a "move base" goal directly in front of AR Tag. Once the goal has been reached, the robot returns to the Drive To Start state and prepares for the next task.

Next, the Turtlebot starts the Empty Parking Spot task by transitioning into the Parking Spot state. In the Parking Spot state, the robot simply looks up a pose from a preset list of waypoints and sets a "move base" goal to navigate to this waypoint. Once this waypoint is reached, the robot transitions back to the Drive To Start State.

Finally, the robot moves to the On Ramp state. In this state, the robot simply navigates to a waypoint located at the beginning of the white line. From here, the robot transitions to the standard drive state.

5.5 Location 3 States

Location 3 states include the TurnLeft3, Detect3, and TurnRight3. The TurnLeft3 and TurnRight3 states are used to turn towards each red shape and back to the white line of the course.

Detect3 uses OpenCV to detect the shape and compare it to the shape saved during the Location 2 task. The robot displays the results by using green LEDs for a successful match, or red LEDs for a failed match.

5.6 Image Processing

The navigation and tasks performed by the turtlebot rely on image processing to operate. Here are a few high-level pseudo code samples showcasing some of the more important aspects of the image processing:

5.6.1 Color masking

```
Given: An HSV image, maximum and minimum HSV thresholds, optional noise
removal size, optional noise filter fill size

    Mask the input image, removing pixels below the minimum threshold
    Mask the input image, removing pixels above the maximum threshold
    Combine the maximum and minimum masks
    Apply an openCV morphology open filter to the mask with the removal
size
    Apply an openCV morphology close filter to the mask with the fill size
    Return: the thresholded and filtered image
```

5.6.2 Finding the centroid of the closest red line

```
Given: A masked image

    Find all contours in the image
    Pick the contour with a mass above a certain threshold whose center is
the lowest on the screen
    Return: the centroid of that contour
```

5.6.3 Detecting shapes

```
Given: A masked image

    Get contours of all shapes in the mask with masses above a threshold
    Create an empty array to hold discovered shapes
    Approximate each contour as a polygon
    Count the Number of sides the shape has
    If it has 3 sides
        It is a triangle, append a triangle to the array of shapes
    If it has 4 sides
        It is a square, append a square to the array of shapes
    If it has 5 sides
        It is a pentagon, append a pentagon to the array of shapes
    If it has more than 6 sides
        It was probably a circle, append circle to the array of shapes
    If it has any other number of sides, ignore it.
    Return: The array of discovered shapes
```

5.6.4 Studying a scene to confidently detect an image

```
Given: A function for getting a masked image

    Initialize a map of shape -> counts
    Store triangle, square, and circle in the map
    For at most a predetermined maximum number of iterations
        Get an image using the given function
        Detect the shape in that mask (see 5.5.3)
        Add one to the count of the detected shape in the map
        If we have iterated more than a predetermined minimum of iterations
            Compute: count/number of iterations for all shapes in the map
            If the computed value is larger than a confidence factor
                Return the detected shape
    Since no shape was detected yet, return unknown
```

5.6.5 Counting Objects

```
Given: Mask of an image

    Find all the contours in the mask, return the number of contours that
are bigger than a certain size
```