Blob detection using the ZED camera

Objectives

In this laboratory exercise, you will use the ZED camera for blob detection. You will search for green blobs or red blobs in the camera image.

Deliverables

- ❏ Create blob_detector.py - software to detect round blobs in the camera image. Report the center and the size of each blob with a Rospy Publisher

Challenge Problems

- ❏ Create a moving average to filter the output of your detection
- ❏ Create a video of the image detection using *rqt_image_view*
- ❏ Test your detector on different colored blobs or different detection algorithms.

Preparing for Real Time Object Detection

Start with the instructor's solution of `echo.py` or your own you created this morning.

1. **Get test data:** Launch the teleoperation and the zed_ros_wrapper. Use rosbag to record all topics and drive the car around the different markers we have provided.

   ```
   rosbag record -O '/data/racecar/<INSERT_FILENAME_NAME_HERE>.bag'
   /tf /odom /scan /camera/rgb/image_rect_color
   /camera/rgb/camera_info
   ```

   If you're having trouble, you can use the instructor provided rosbag.

   ```
   # NOTE: This code must be run on your LOCAL VM.
   # This code will save the provided rosbag in /data/racecar on
   your local VM.
   sudo mkdir -p /data/racecar
   sudo chown racecar /data/racecar
   cd /data/racecar
   wget
   http://dl.dropboxusercontent.com/u/380036122/BWSI/ROS_BAGS/movin
   g_blob_test.bag
   ```

2. **Playback test data**:
   Use rosbag to playback the data you collected.Verify echo.py works with your test data. In the following steps, use the playback to verify your blob detection.

   ```
   # Here are the commands to playback your bag file.
   # This also opens a window for viewing the outputted image.
   rosbag play --loop /data/racecar/<INSERT_FILENAME_NAME_HERE>.bag
   # Now open up the video viewer
   rqt_image_view
   ```

   Once the video viewer is open, select the video topic you want to view using the dropdown.

   Note: If you get an error saying "*Failed to contact master*" when you try to playback the bag file, open up a new terminal and run "roscore" before trying the playback command again.

Create blob_detector.py

Start with the instructor's solution of `echo.py` or your own you created this morning.

In this assignment you are trying to objects that are bright splotches of color in a plain environment. The first thing you will do is segment the parts of the image that show your desired color (red or green). Then, you will find the largest segmented color to return as your blob.

Image Segmentation

In the lecture you learned about different color spaces. In particular, that it can be easier to find colors with the HSV (Hue-Saturation-Value) colorspace.

To transform an image into the HSV colorspace you can use OpenCV's inRange function.

You can find sample code of the inRange function here:
http://docs.opencv.org/trunk/df/d9d/tutorial_py_colorspaces.html

In this tutorial, it shows how to segment a blue circle out of an image. In our task we are trying to find red and green markers. To find the correct HSV values you can open your image in GIMP or any other online converter to get the HSV ranges. Additionally, OpenCV has a cvtColor function to convert BGR values to HSV.

Try running the image segmentation in realtime by augmenting echo.py. You can start working with different HSV ranges to improve your segmentation.

Finding Contours

Now that you're able to segment the desired color ranges of your image, you can try to segment them. OpenCV has many different methods for object detection that we encourage you to explore, however, in this exercise we are just going to determine the largest segmented area and return that as our blob.

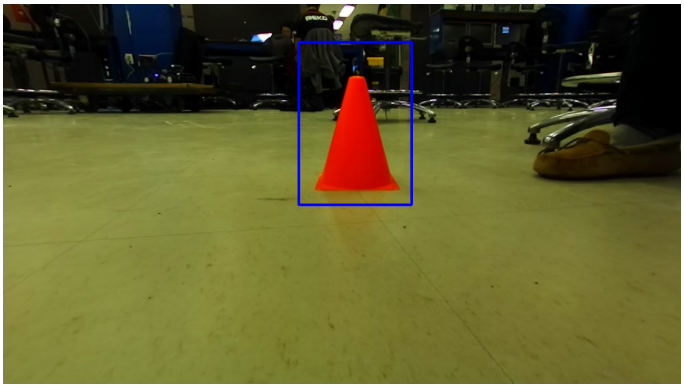
For our task, the instructors found using OpenCV's findContours method to be the most robust for our environment.
OpenCV has a tutorial here on the findContours method:
http://docs.opencv.org/trunk/d4/d73/tutorial_py_contours_begin.html

This method will return a list of contours. You will can find the largest contour by sorting contours with their area using the OpenCV function contourArea.

Real time blob detection:

Improve upon your blob detection by using real time data.

3. **Label detections:** draw a box or circle around each blob detection before publishing the image. eg:



4. **Publish blob locations and sizes**:
    a. **Create BlobDetections.msg**
        - Create this custom rosmsg to report the size, location, and color of multiple blobs:

```
BlobDetections.msg
        Header header
        std_msgs/ColorRGBA [] colors
        std_msgs/Float64 [] sizes
        geometry_msgs/Point [] locations
```

        - The lists `colors`, `sizes`, and `locations` are ordered. Update them with list of blob detections in decreasing order (largest first).

- Locations should be reported as relative pixel values. For example, if your image is 640 x 480 and you have a detection at (320,240), then the output is (.5,.5)
- Sizes can be raw pixel numbers. Return '4' if the blob radius is '4' in pixel units.

b. Publish detections on topic "`blob_detections`"

- Create a rospy Publisher that uses the custom message type `BlobDetections.msg`. Publish the detections in realtime.

5. Test the blob_detector.py

a. Launch teleoperation, zed_ros_wrapper, and your blob_detector
b. Use rosbag to record all topics
c. Drive around the room and near the colored markers we have provided.
d. Review the recording on your computer and verify the detection is working.

6. Enhance performance

a. Use rostopic hz to determine how fast blob_detector.py works. Compare it with echo.py and the original image topic. Try modifying your code to improve performance of your detector.