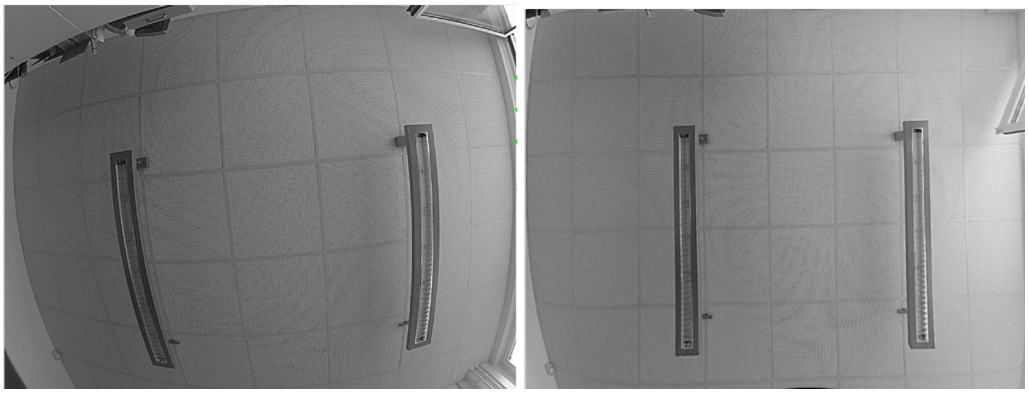
7. Assignment, Introduction to Robotics WS17/18 - Ver. 1.02

Prof. Daniel Göhring, Martin Dreher, Jakob Krause
Institut für Informatik, Freie Universität Berlin
Submission: online until Tuesday, 12 Dec 2017, 11:55 a.m.

Please summarize your results (images and descriptions) in a pdf-document and name it, e.g., "RO-07-<surnames of the students - group name>.pdf".
Submit your python code and video
Only one member of the group must submit the necessary files.
Do not copy solutions to other groups.
Every group must contain two people, unless granted differently.
Only submissions via KVV will be accepted.

This node should localize the car using at least three colored markers attached to the ceiling. The fisheye camera on top of the car provides an angle of view of 170 degrees. It can be used to detect colored light bulbs which are hanging from the ceiling.

1- Fish-eye camera calibration (2 points)
With the help of the camera_calibration package we can calibrate the fisheye camera. (http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration)
head_camera.yaml file contains intrinsic and extrinsic camera parameters. Copy the file to the Odroid in .ros/camera_info/ Folder.



Picture1: Fisheye Camera picture before and after Calibration.

Edit the Usb_cam launch file (*model_car/catkin_ws/src/manual_control/launch/fish_eye_camera.launch*) to set the size of image to 640x480 or use other packages:
https://github.com/AutoModelCar/model_car/tree/version-1/catkin_ws/src/fisheye_camera_matrix

https://github.com/AutoModelCar/model_car/tree/version-1/catkin_ws/src/fisheye_camera_matrix_msgs
Submit a distorted and an undistorted image by including it in your Pdf.

2- Find the color range for each balloon (3 points)

Create a function that takes an image from the callback as an argument.
It should associate an image coordinate to each balloon that was detected.

Find the color range of each balloon in order to detect them.
Minimizing time exposure can help to stay independent from surrounding light sources.
You can minimize the camera exposure time using v4l_ctrl as shown below:

```
$ v4l2-ctl --device=/dev/usb_cam --list-ctrls
$ v4l2-ctl --device=/dev/usb_cam --set-ctrl exposure_auto=1
$ v4l2-ctl --device=/dev/usb_cam --set-ctrl exposure_absolute=3
```

Old positions, only for the bagfile: The position for the colored light bulbs (baloon) are [Baloon_color(x(meter),y(meter))]:

- *Green:(2.33 , 1.15), Red (3.57, 3.0), Blue (3.57, 1.15), Purple (2.33, 3.0)*

For first tests, feel free to use this bagfile, which contains image data and odometry data:
http://ftp.imp.fu-berlin.de/pub/autonomos/data/modelcar/2017-08-09-visual_gps.bag.tar.gz

The current positions are - and corresponding to the frame in the image:

Green (2.29, 1.14), Red (3.55, 3.03), Blue (4.18, 1.77), Purple (2.29, 2.40),

Please submit the color intervals [r_min, r_max]x[g_min_g_max]x[b_min, b_max] for each color - or the method which you were using - in your Pdf.

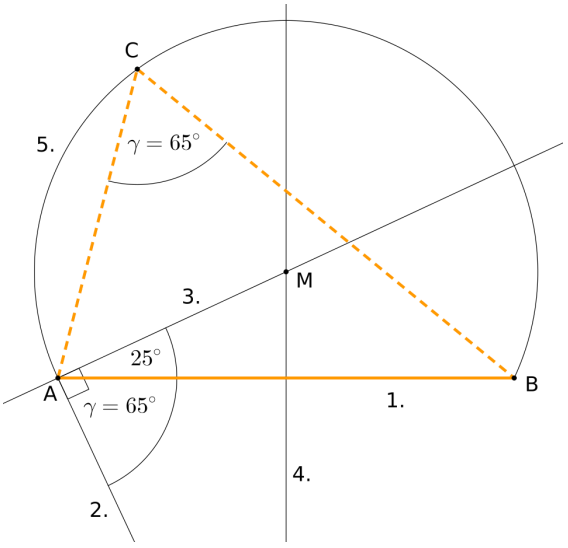
3- Visual GPS (5 points)

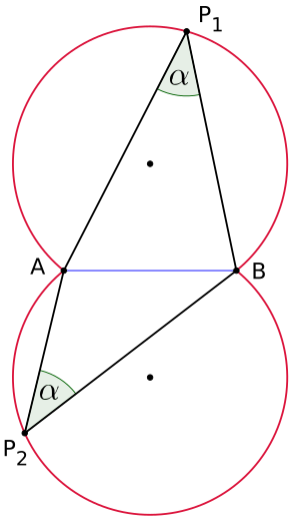
Create a function that takes a mapping of image coordinates to real world balloon coordinates as an argument. It should return a real world position (x and y) in the same coordinate system as the balloon coordinates.
Additionally it should return the observed yaw angle.

There are different ways to find the position of the car. Choose one method.

First Method (3 or more light bulbs), here you can use the distorted image, too:

1. Find the angle between center of camera(robot) and two lamps.
2. Find the center and radius of circle.
3. Calculate the intersection(s) of two circles.



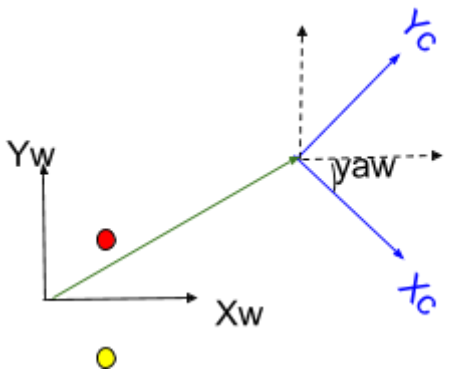


Source: Wikipedia, Fasskreisbogen

Second Method: (with 3 or more lamps)
(http://nghiaho.com/?page_id=671)

Finding the optimal rigid transformation matrix can be broken down into the following steps:

1. calibrate the camera, publish undistorted image
2. Find the centroids of both datasets
3. Bring both dataset to the origin then find the optimal rotation, (matrix R)
4. Find the translation t



Third Method: (with 3 or more lamps): Use a grid, cellsize 5 or 10 cm. For each pair of lamps which you perceived, compare the perceived angle between both lamps with the expected angle. Then, for each cell in your grid you will get a weight by calculating the angular difference as: (abs(angularDifference) in deg (if you use the occupancyGrid, make sure that the values stay in the interval of [0, 100]). Calculate the angular difference for each pair of lamps, i.e., add all the absolute angular differences together, assign each sum to the corresponding each grid cell. The grid cell with the lowest value is your final result. If you use an occupancy grid as a data structure, you can easily visualize the grid.

For any of the three methods:

Publish your estimated position as odom_gps (the message type: nav_msgs/Odometry).

Create an odometry message (http://docs.ros.org/api/nav_msgs/html/msg/Odometry.html) from the result of your

positioning function. For this task you need to encode the yaw angle into a quaternion.

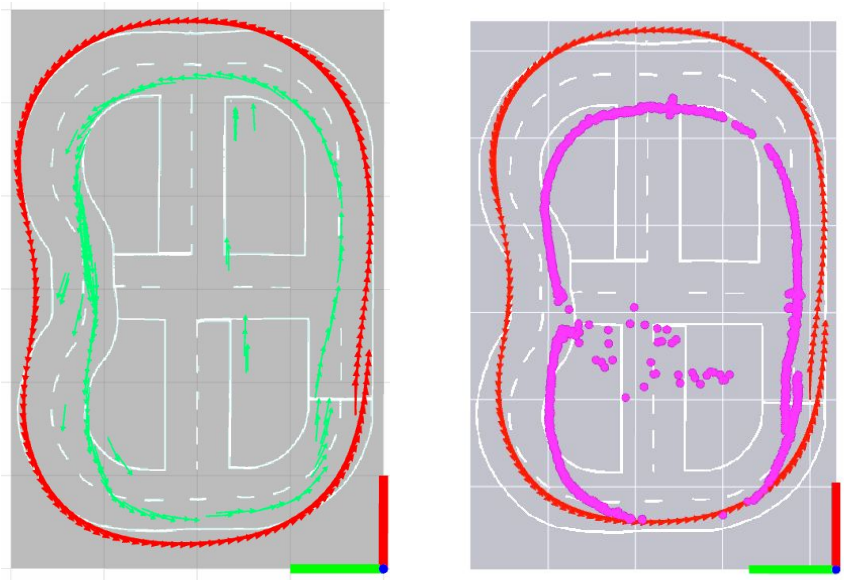
The covariance of the pose and the twist can be empty.

In your submission PDF include an example Image with detected balloon image coordinates and the resulting x, y position and yaw angle.

Move the car in a circle or another closed loop around the room, and visualize the position (odom) and your estimated position. Visualize both position sets using rviz.

Commit the source code to your catkin_ws_user git repository. Show the result of a test with the car in the lab (where the car is driving some kind of a circle) by attaching a video file (max 5 MB, ogv or mp4 format).

Attach the measured odometry and GPS position tracks into you Pdf, as done in the images below. Submit the video, which shows how your car moved, too. Put a link to your source code in your final pdf.



Plot examples(red:raw_odometry, green:visual_gps, purple: visual_gps)

Hint: To convert yaw angles to quaternions, you can use the following function:

```
def yaw_to_quaternion(yaw):
    """ convert a yaw angle (in radians) into a Quaternion message """
    return Quaternion(0, 0, math.sin(yaw / 2), math.cos(yaw / 2))
```