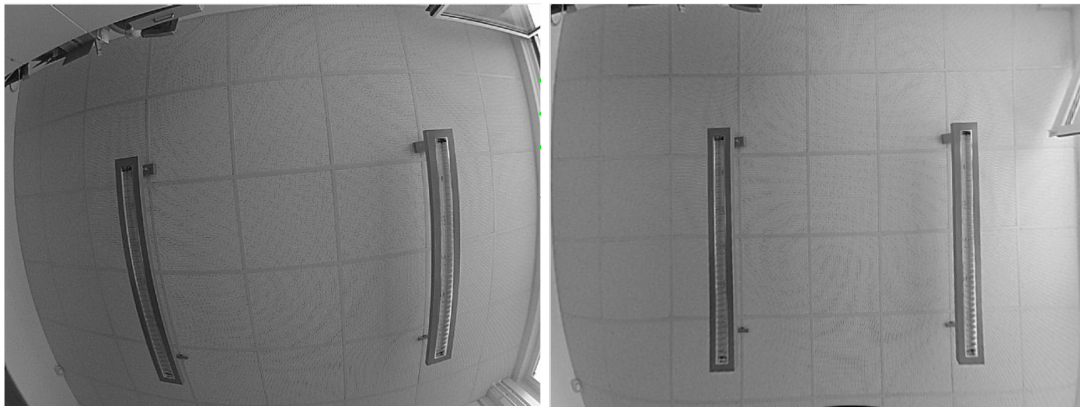
6. Assignment: Visual GPS

This node should localize the car using at least two 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, color balloon mounted from the ceiling.

1- Fish-eye camera calibration (1 point)
Using 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 befor and after Calibration.

Edit Usb_cam launch file 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

2- Find color rang of each balloon. (2 points)
In the second step, you should find the color range of each balloon in order to detect them.
Minimizing exposure maybe helps the color detection to be independent from environmental lights. Minimize camera exposure 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
```

Position of color Balloon mounted from the ceiling[Ballon_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)

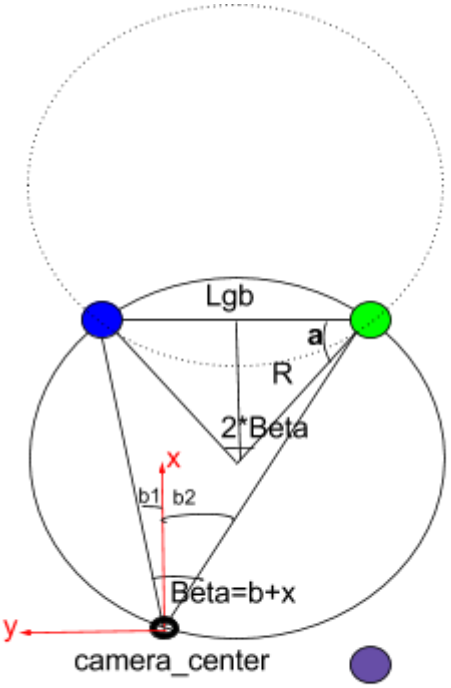
useful bagfile:
http://ftp.imp.fu-berlin.de/pub/autonomos/data/modelcar/2017-08-09-visual_gps.bag.tar.gz

3- Visual GPS(5 point)

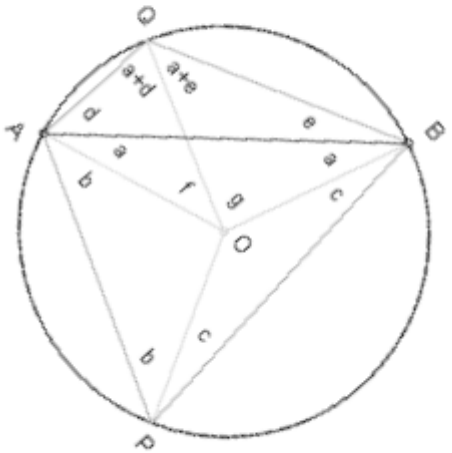
There are different ways to find the position of the car related to more than 2 known position.

First Method (more than 3 lamps):

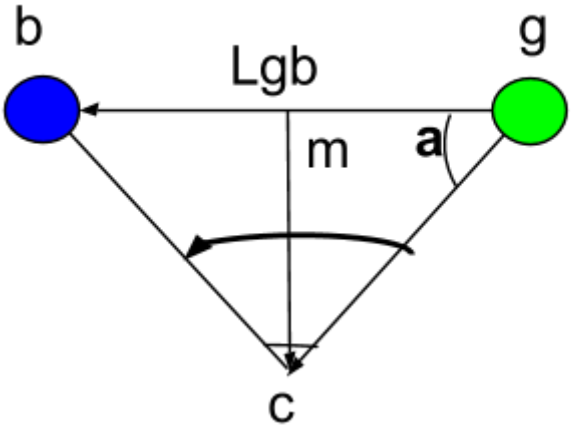
1. Find the angle between center of camera(robot) and two lamps.
2. Find the center and radius of circle.



a=pi/2-Beta



a=pi/2-(b+c)



$$\overrightarrow{gm} + \overrightarrow{mc} = \overrightarrow{gc}$$

$$\overrightarrow{gb} = u_1 * \overrightarrow{x} + u_2 * \overrightarrow{y}$$

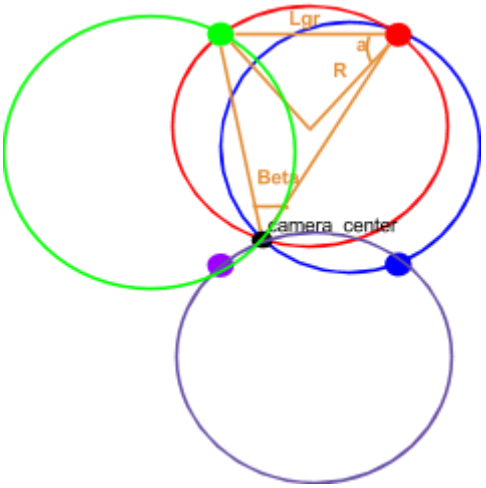
$$\overrightarrow{N_{gb}} = \frac{-u_2 \overrightarrow{x} + u_1 \overrightarrow{y}}{\sqrt{u_1^2 + u_2^2}}$$

$$\overrightarrow{mc} = \tan(a) \frac{\sqrt{u_1^2 + u_2^2}}{2} \overrightarrow{N_{gb}} = \tan(a) \frac{-u_2 \overrightarrow{x} + u_1 \overrightarrow{y}}{2}$$

$$\overrightarrow{gc} = \frac{\overrightarrow{gb}}{2} + tan(a) \frac{-u_2 \overrightarrow{x} + u_1 \overrightarrow{y}}{2}$$

$$c = g + \frac{\overrightarrow{gb} + tan(a)(-u_2 \overrightarrow{x} + u_1 \overrightarrow{y})}{2}$$

3. Do pervious steps for two other lamps.
4. Find the intersections between the circles.



$$(x_{car} - x_{oi})^2 + (y_{car} - y_{oi})^2 = R_i^2$$

$$\begin{cases} x_{car}^2 - 2x_{car}x_{oi} + x_{oi}^2 + y_{car}^2 - 2y_{car}y_{oi} + y_{oi}^2 = R_i^2 \\ -(x_{car}^2 - 2x_{car}x_{oj} + x_{oj}^2 + y_{car}^2 - 2y_{car}y_{oj} + y_{car}^2 = R_j^2) \end{cases}$$

$$x_{car}(x_{oi} - x_{oj}) + y_{car}(y_{oi} - y_{oj}) = (x_{oj}^2 - x_{oi}^2) + (y_{oj}^2 - y_{oi}^2 + R_i^2 - R_j^2)/2$$

$$A[x_{car}\ y_{car}]^T = B => [x_{car}\ y_{car}]^T = (A^T A)^{-1} A^T B$$

Second Method: (with more than 2 lamps)

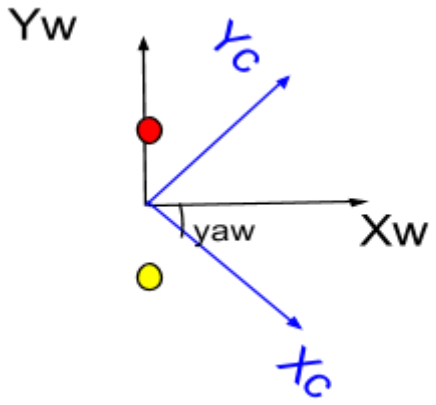
(http://nghiaho.com/?page_id=671)

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 dataset
3. Bring both dataset to the origin then find the optimal rotation, (matrix R)
4. Find the translation t

Useful formula:

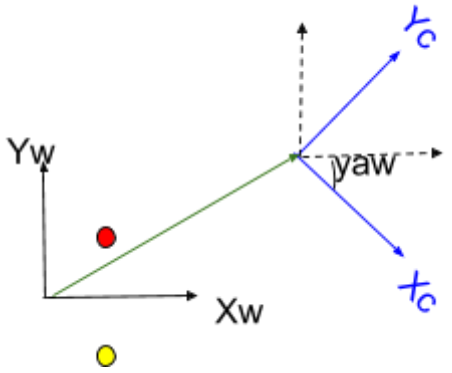
Pc_i=Pcamera_i- Pc_averag
Pw_i=Pworld_i-Pw_averag



X'w= Xc *cos(yaw)-Yc*sin(yaw)
Y'w=Xc*sin(yaw)+Yc*cos(yaw)

yaw=min[(X_w-X'_w)^2+(Y_w-Y'_w)^2]=>

closed form solution:
yaw=arctan(SUM(Xc_i*Yw_i-Yc_i*Xw_i)/SUM(Xc_i*Xw_i+Yc_i*Yw_i))



tx=Xw_average-[cos(yaw) -sin(yaw)]*Pc_average
ty=Yw_average-[sin(yaw) cos(yaw)]*Pc_average

At the end publish your estimated position as odom_gps (the message type: nav_msgs/Odometry). Move the car in a circle around the room, and visualize the real position and your estimated position using rviz.

Commit the source code to your catkin_ws_user git repository. Show the result in a live demo with the car in the lab or attach a video file and put a link to your source code in your final pdf.

3- Kalman filter (Combine odometry and visual gps)(2 points)

Use Kalman filter and combine the odometry data and GPS data to find better position. You can use the simple model of the car in the prediction step.

dX= Vx cos(yaw)

dY=Vx sin(yaw)

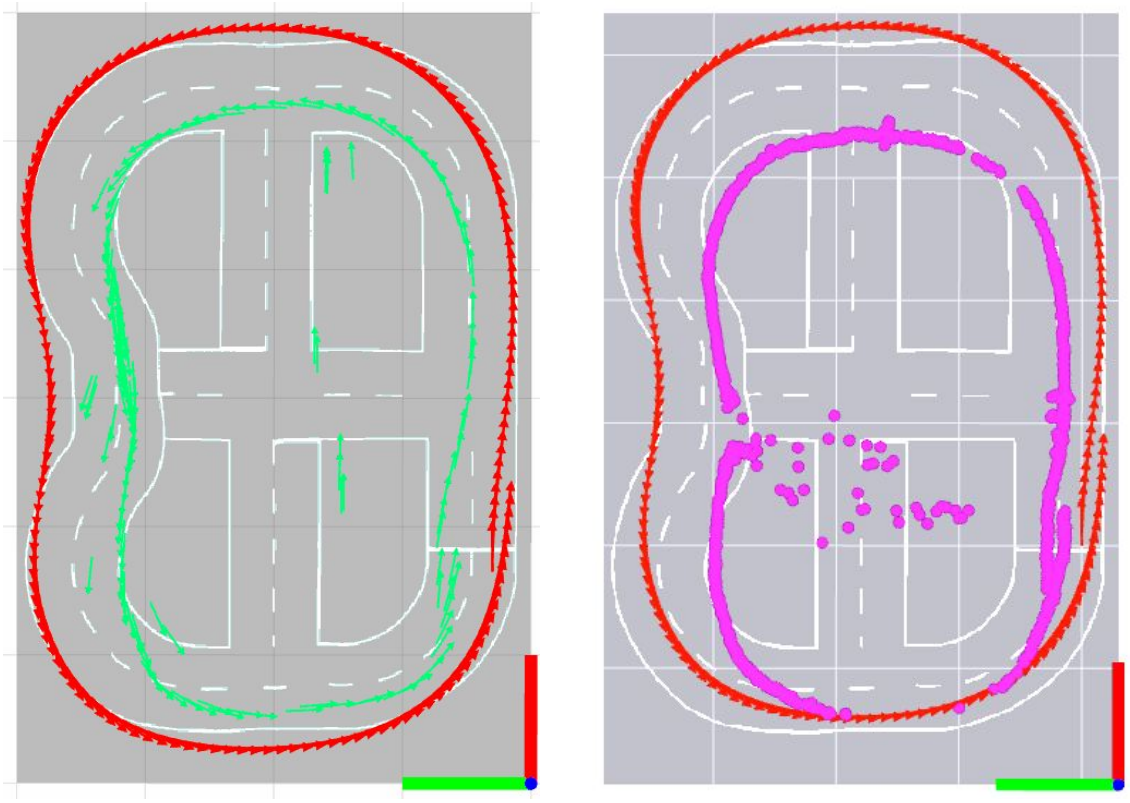
At the end publish your estimated position as odom_gps (the message type: nav_msgs/Odometry). Move the car in a circle around the room, and visualize the real position and your estimated position using rviz.

Commit the source code to your catkin_ws_user git repository. Show the result in a live demo with the car in the lab or attach a video file and put a link to your source code in your final pdf.

Kalman Filter Formula:

Prediction:
kalman_h: position of the car!
kalman_V is the process noise
kalman_h=[x,y,yaw]=[cos(yaw)*v+tx;sin(yaw)*v+ty,yaw]
kalman_H=[dx,dy,dyaw]=[1,0,sin(yaw)*v;0,1,-cos(yaw)*v;0,0,1]
kalman_P+=**kalman_V**

Update:
y: measurements and calculation and find the position of the car.
kalman_P: If the initial position and velocity are not known perfectly, the covariance matrix should be initialized with suitable variances on its diagonal
kalman_W: measurement noise W is normally distributed, with mean 0 and standard deviation σ_z.
Error = y-kalman_h;
kalman_S = (kalman_H*kalman_P)*kalman_H.transpose() + kalman_W;
kalman_K=(kalman_P*kalman_H.transpose())*kalman_S.inverse();
kalman_x += kalman_K * Error;
kalman_P -= kalman_K*kalman_H*kalman_P;



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