CENTRALE NANTES

TurtleBot localization with visual markers

- Mahmoud ALI & Kevin SERRANO
- Supervisor: Olivier Kermorgant

Introduction

Localization

TurtleBot

OmniCamera

Beacons (visual markers)

Environment

Localization

The robot localization problem is key in making truly autonomous robots.

Access to relative and absolute measurements.

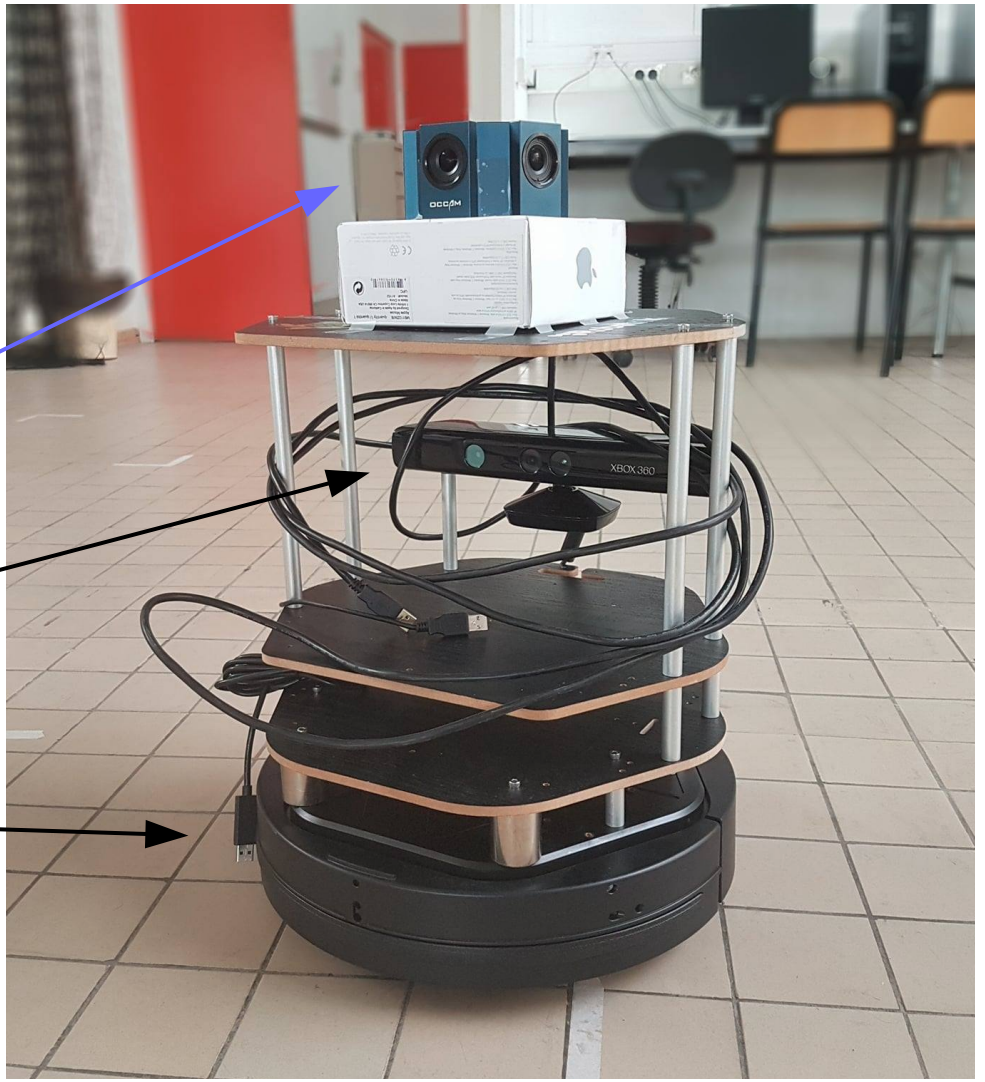
Combine then in an optimal way.

TurtleBot

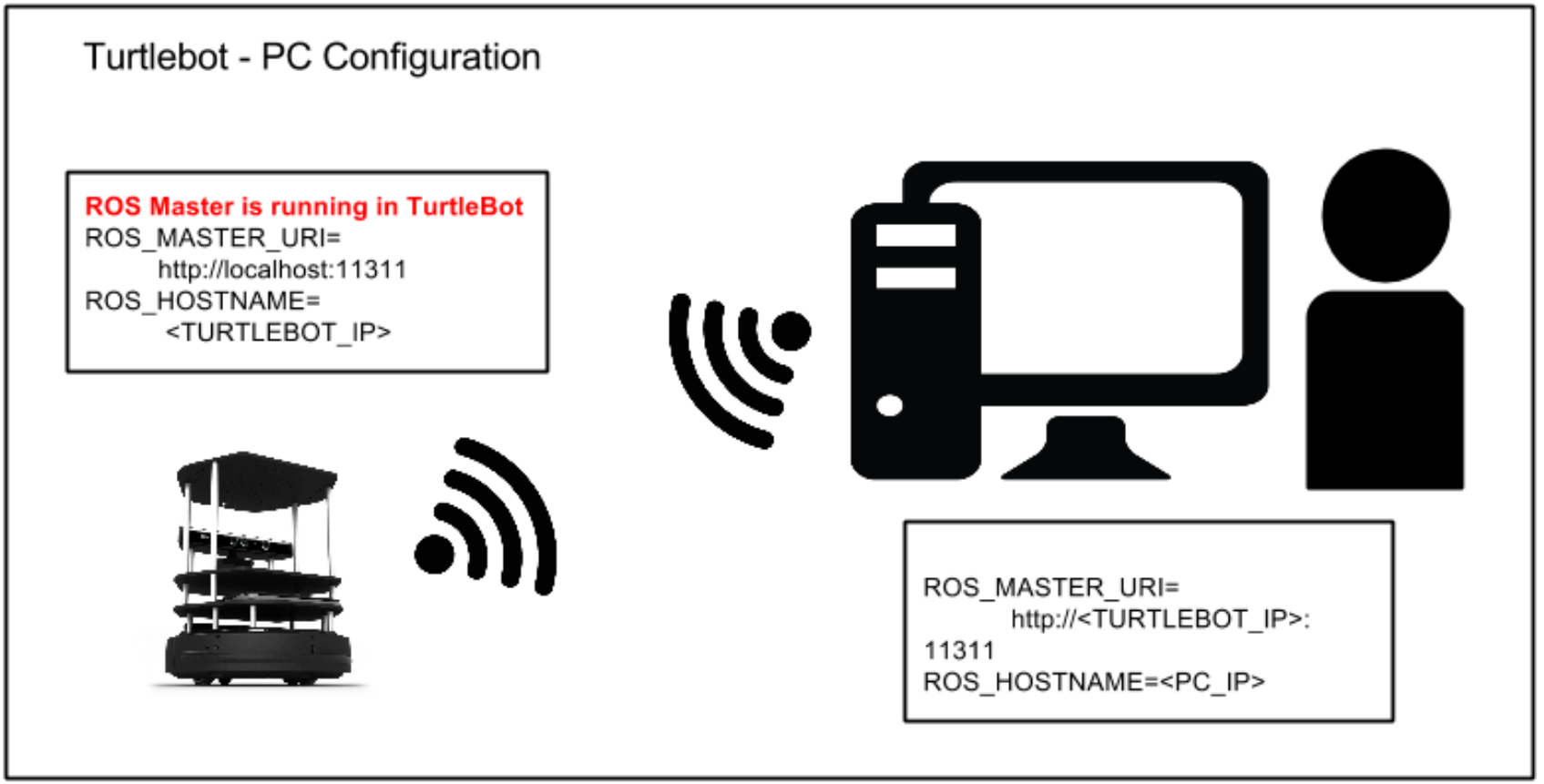
Occam 60

Kinect

Kobuki base



Network Configuration



http://wiki.ros.org/turtlebot/Tutorials/indigo/Turtlebot-Developer%20Habitats?action=AttachFile&do=get&target=natural_habitat.png

Omni 60

1.8 MP raw video

60 Hz synchronized capture

Real time 360° panorama

Compatible with OpenCV and ROS

Beacons

Beacons

Environment

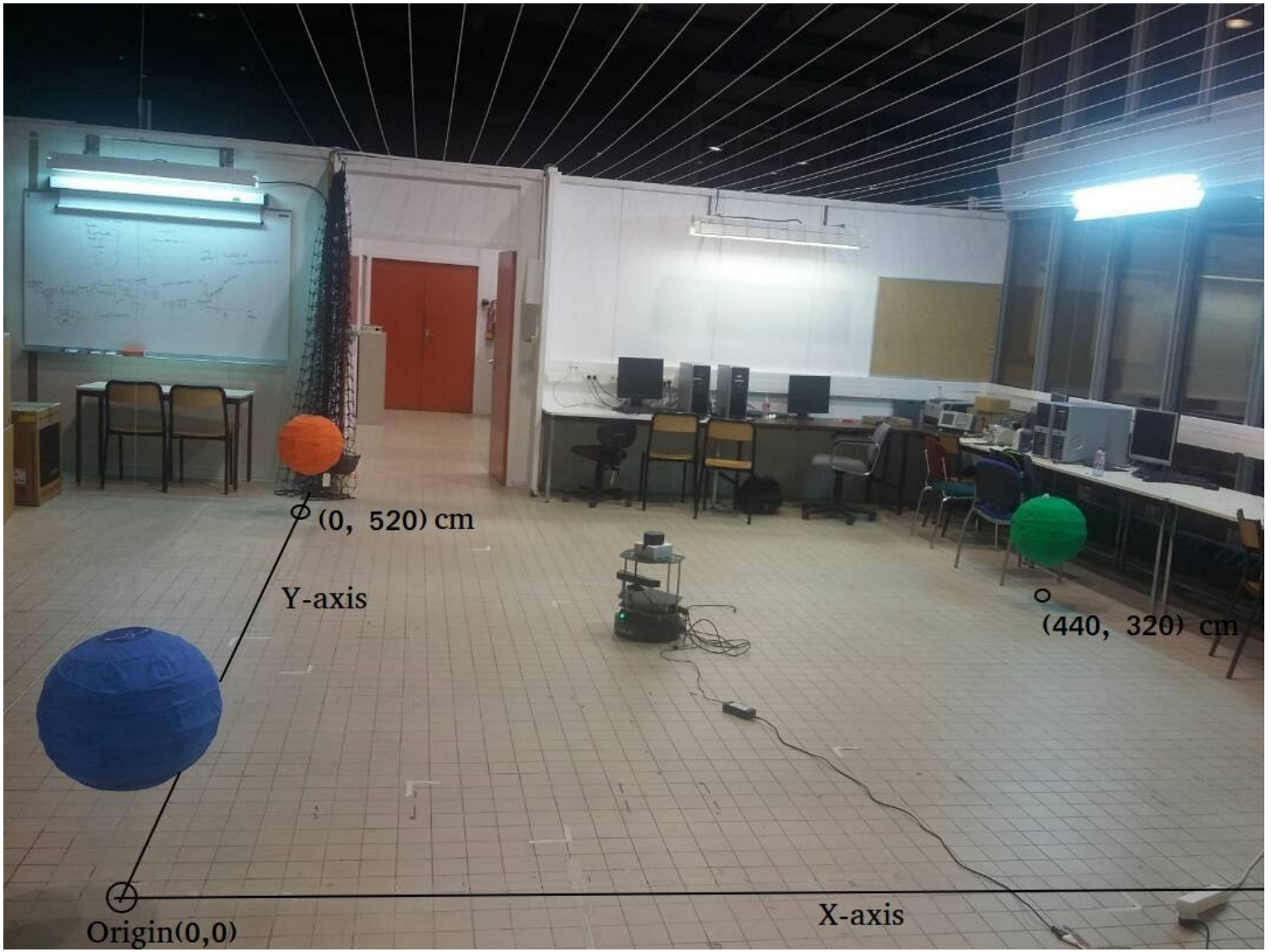


Image processing

Image rectifying

Image tiling

Color segmentation

Blob detection

Distance calculation

Time performance

ROS

Image rectifying

Camera calibration

$$s\, m' = K[R|t]M'$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Image rectifying

Equivalent transformation when $z \neq 0$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \qquad \begin{cases} x' = x/z \\ y' = y/z \\ u = x' f_x + c_x \\ v = y' f_y + c_y \end{cases}$$

Undistorted points (x', y')

Image rectifying

Adding lens distortion

$$\begin{cases} x' = x/z \\ y' = y/z \\ x'' = x' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + 2p_1 x' y' + p_2(r^2+2x'^2) + s_1 r^2 + s_2 r^4 \\ y'' = y' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + 2p_2 x' y' + p_1(r^2+2y'^2) + s_1 r^2 + s_2 r^4 \\ r = x'^2 + y'^2 \\ u = x'' f_x + c_x \\ v = y'' f_y + c_y \end{cases}$$

Normally $D_{coeffs} = [k_1, k_2, p_1, p_2, k_3]$

Distorted points $(x'', y'') \rightarrow (x', y')$

Image rectifying

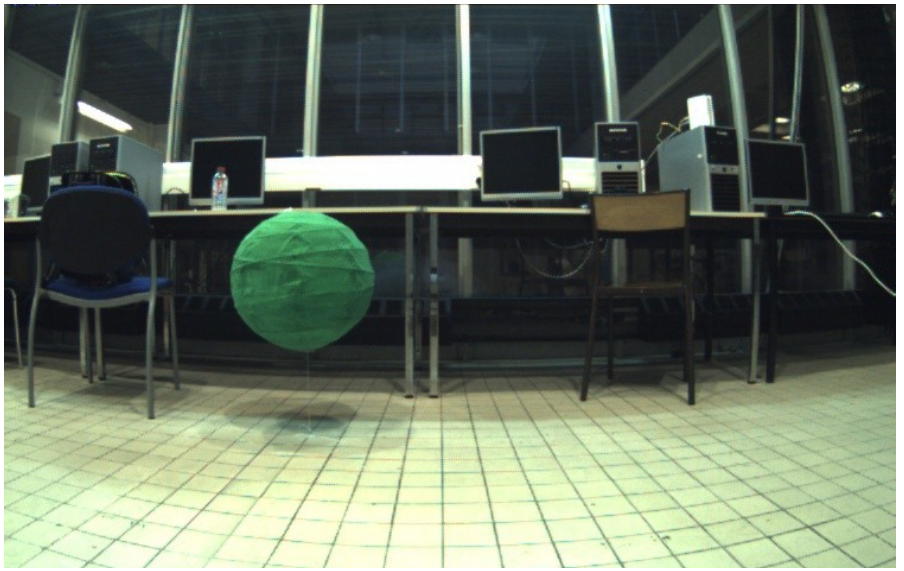


Image tiling

360° advantage (?)

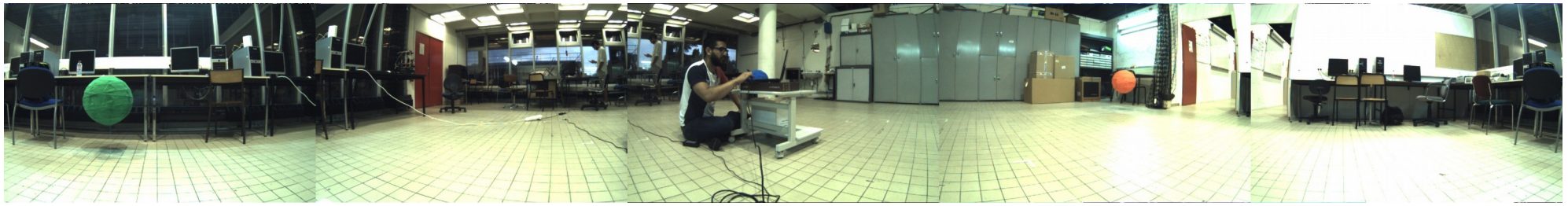


Image tiles

Rectified tiles

Color segmentation

Color spaces

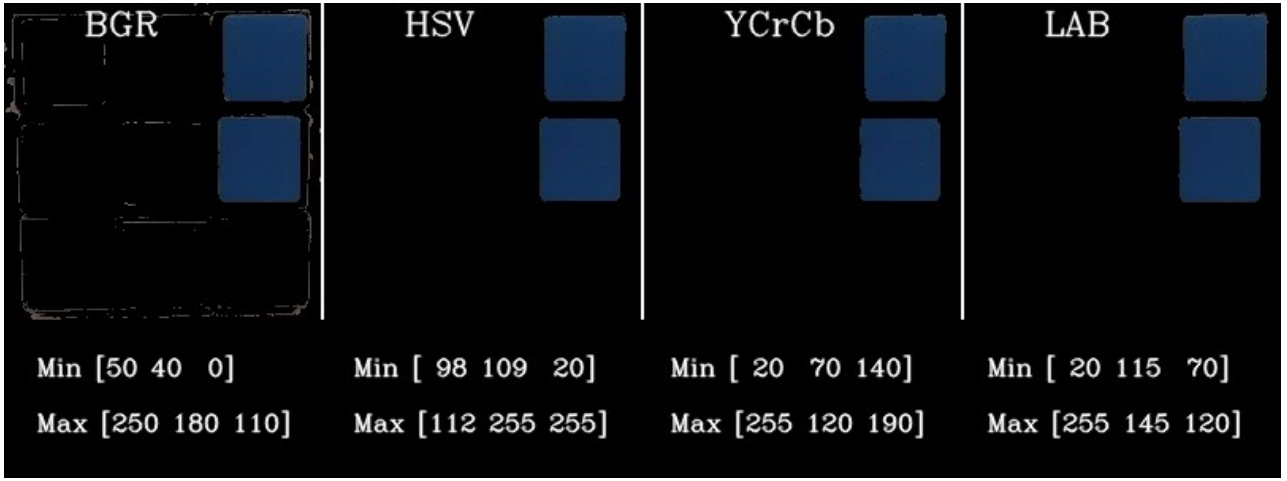
- RGB (additive color space)
- Lab (2 channels for color, 1 for lighting)
- YCrCb (2 channels for color, 1 for lighting)
- HSV (1 Channel for color)

Color segmentation

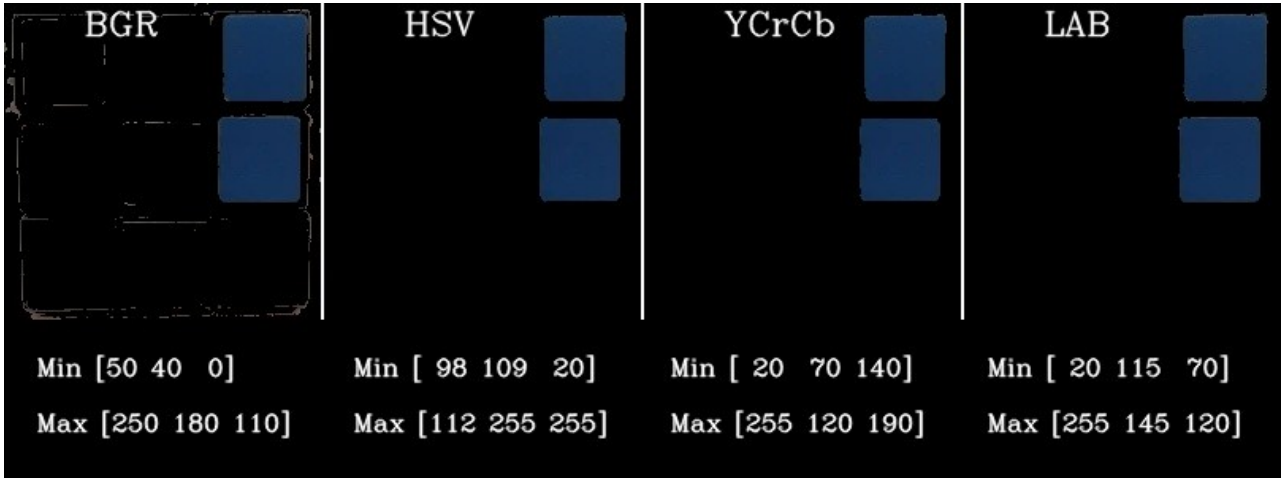
Outdoor

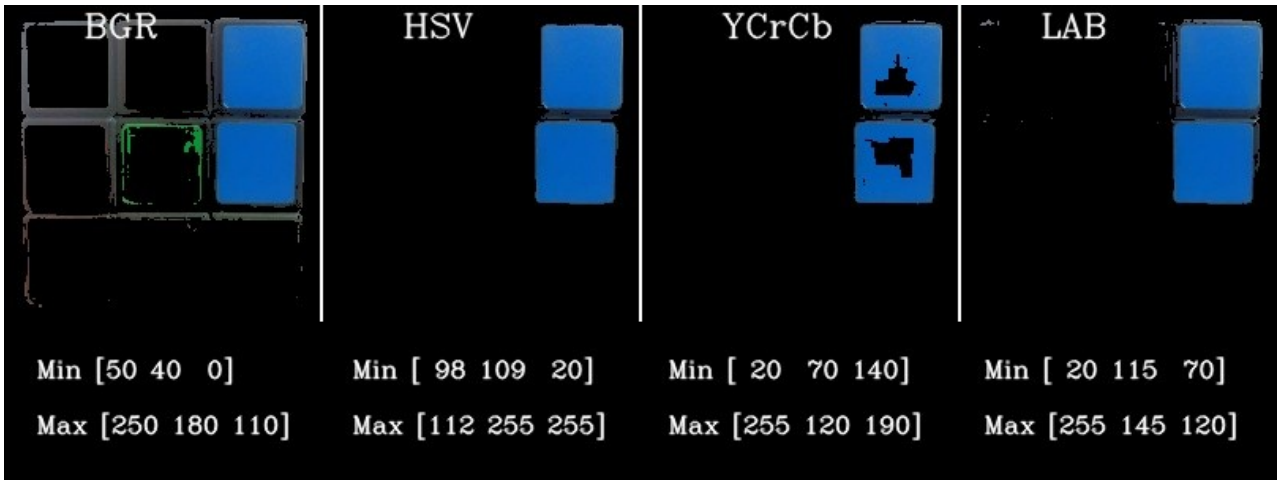
Indoor

Color segmentation



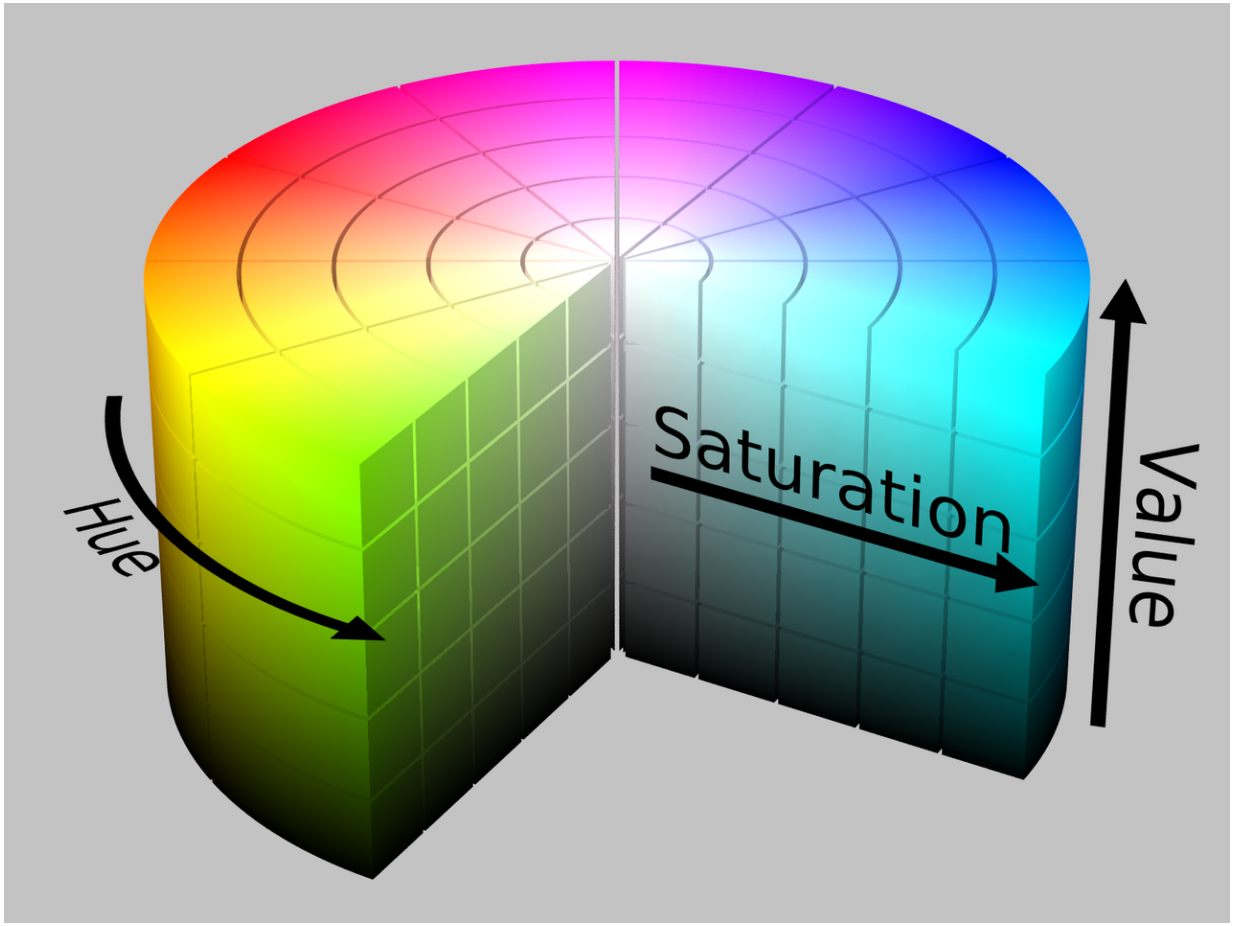
Color segmentation



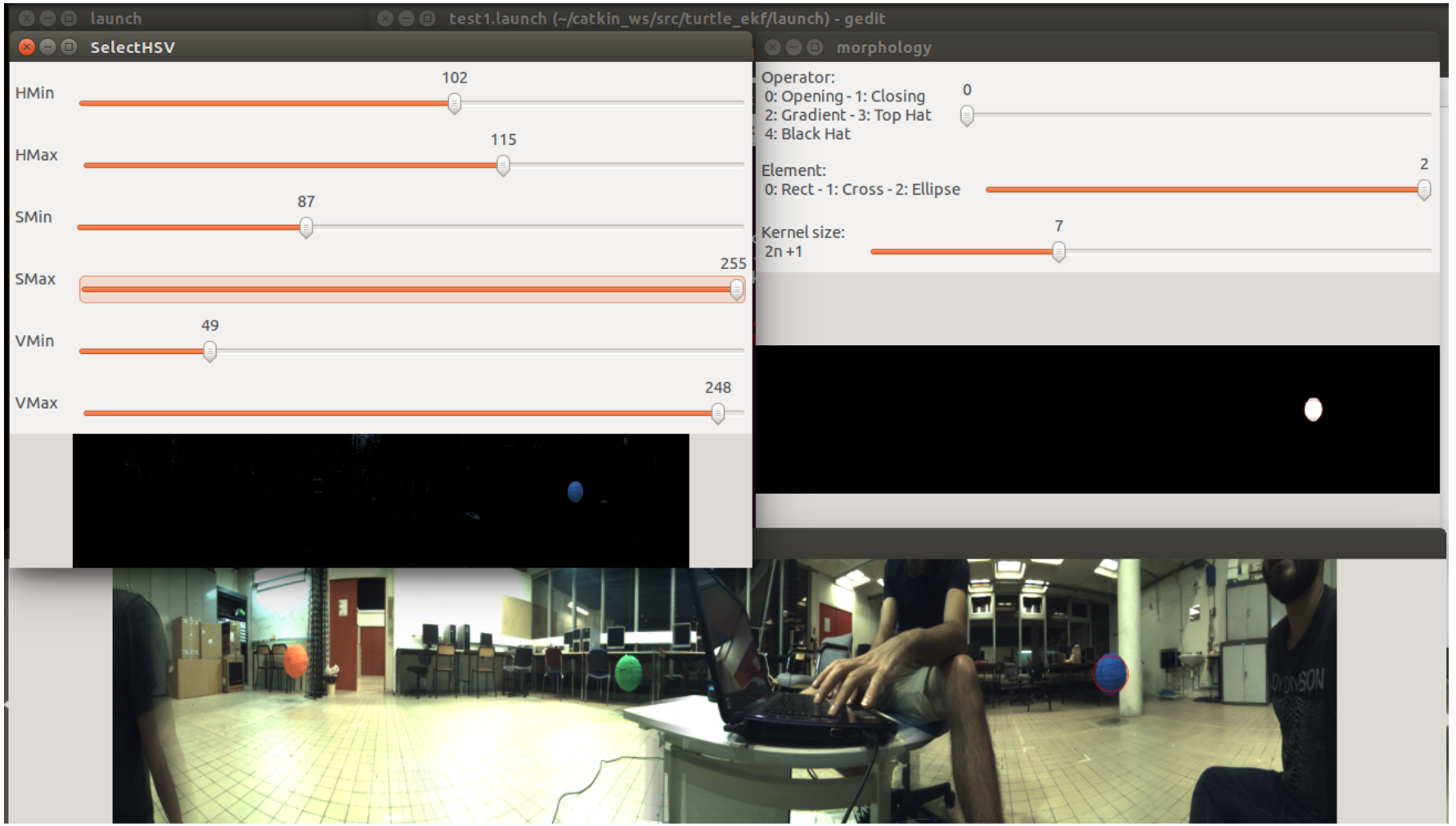


Color segmentation

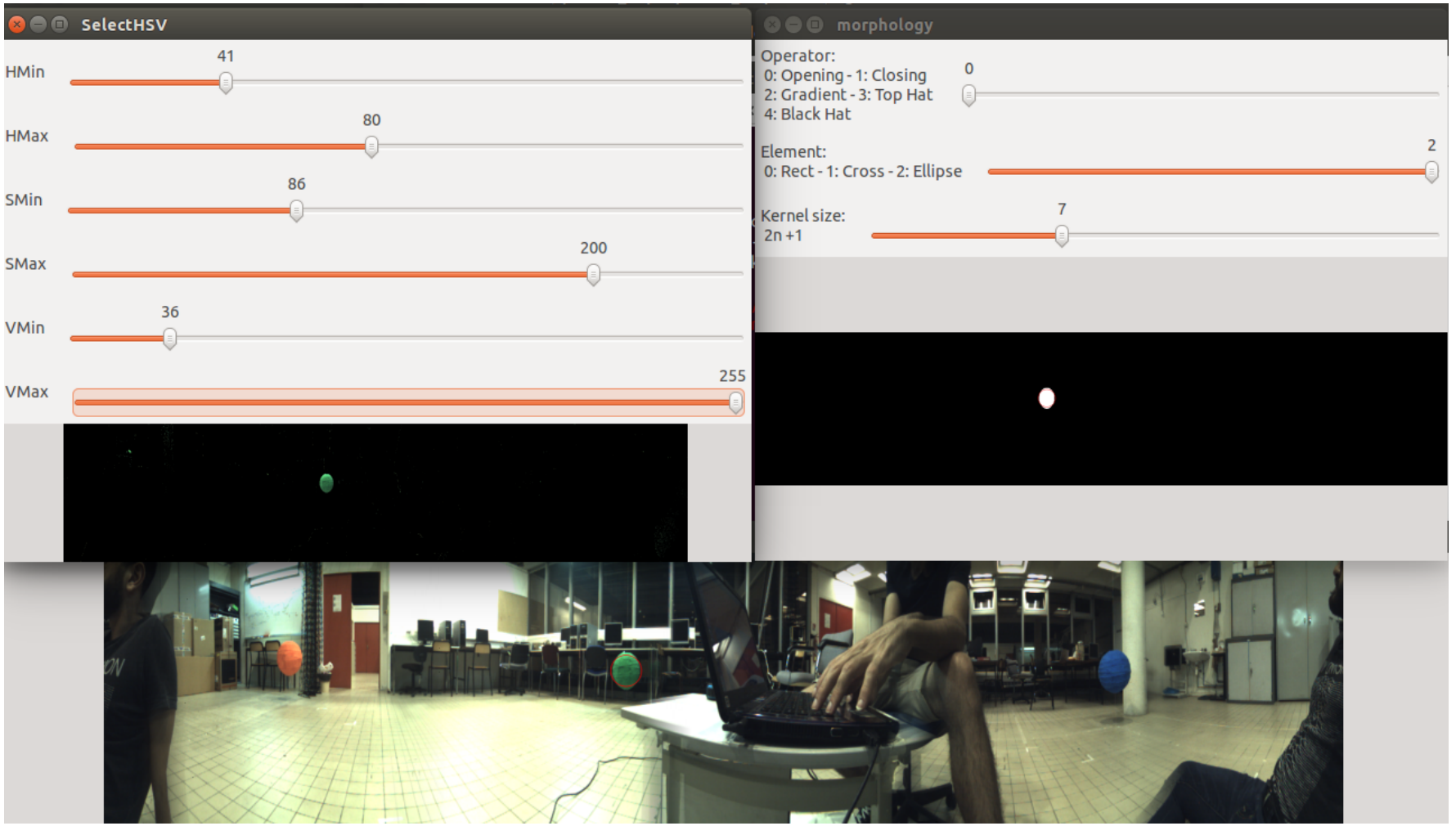
Winner ?



Color segmentation



Color segmentation



Color segmentation

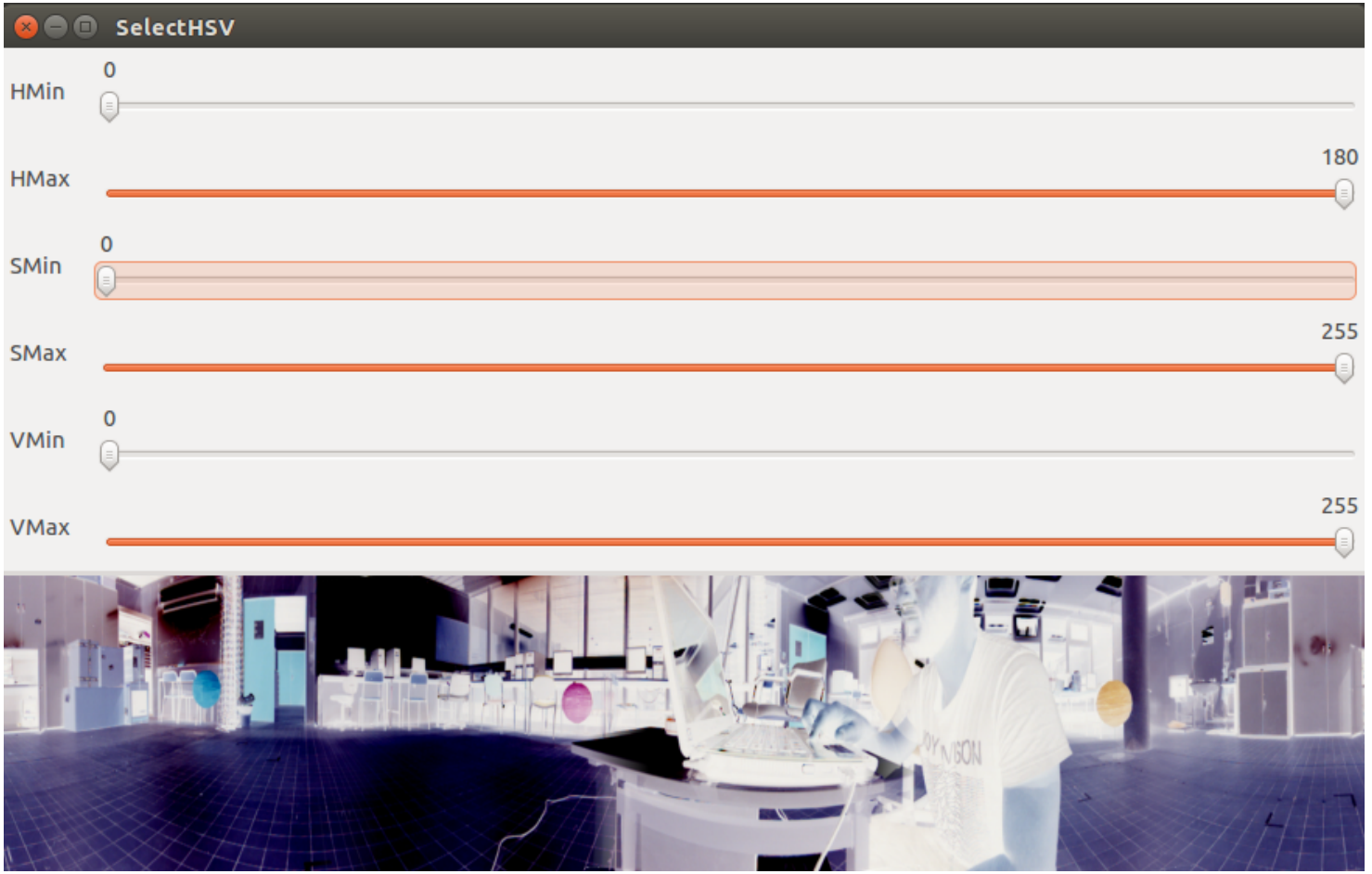
What about red/orange segmentation ?

https://upload.wikimedia.org/wikipedia/commons/thumb/a/ad/HueScale.svg/2000px-HueScale.svg.png

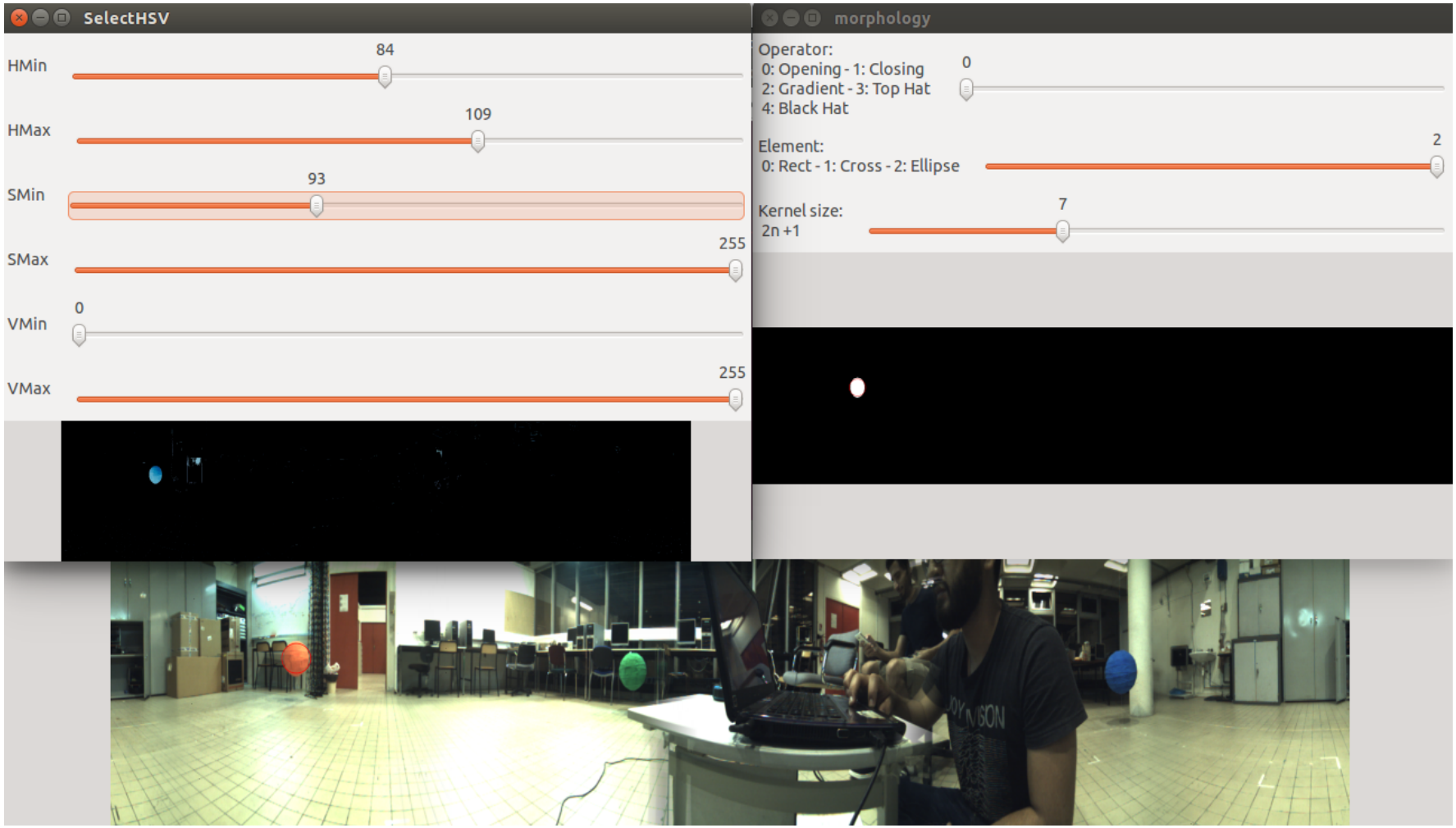
Color segmentation

Invert the RGB image!

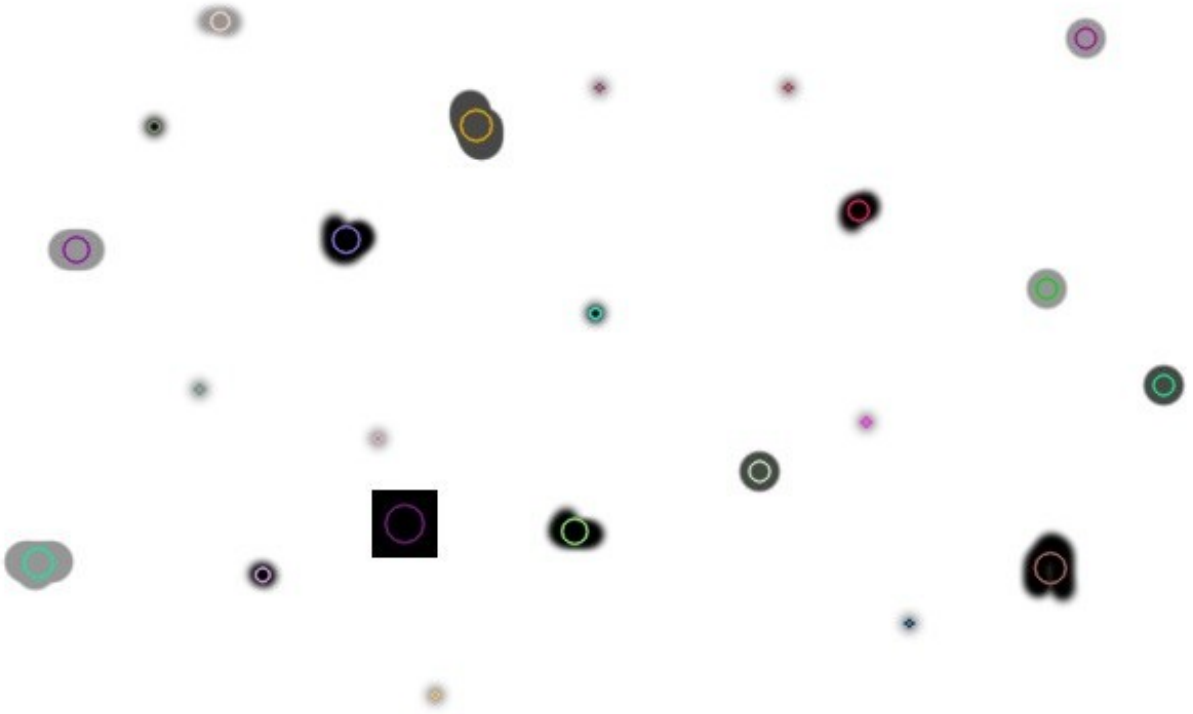
Color segmentation



Color segmentation



Blob detection



http://www.learnopencv.com/wp-content/uploads/2015/02/blob_detection.jpg

Blob detection

OpenCV algorithm: `SimpleBlobDetector`

1)Thresholding: creating several binary images

2)Grouping: connected white pixels

3)Merging: between binary images

4)Center and **radius** calculation

Blob detection

We can filter by

- Color (dark of white)
- Size
- Shape

Blob detection

We can filter by

- Color (dark of white)
- Size
- Shape
  - Circularity

Blob detection

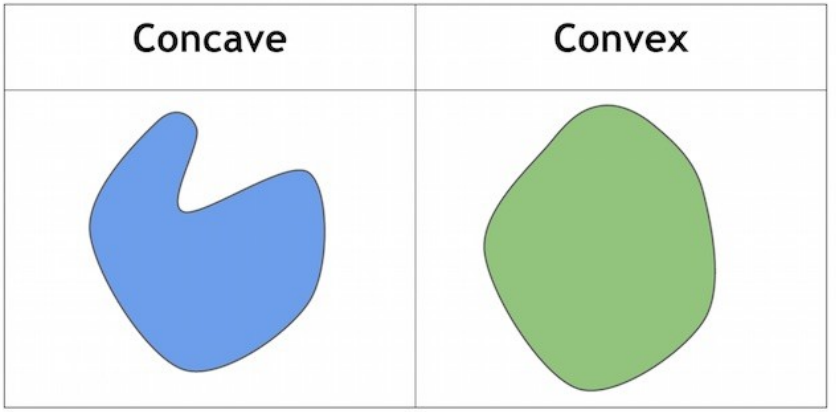
We can filter by

- Color (dark of white)
- Size
- Shape
  - Circularity
  - Convexity

Blob detection

We can filter by

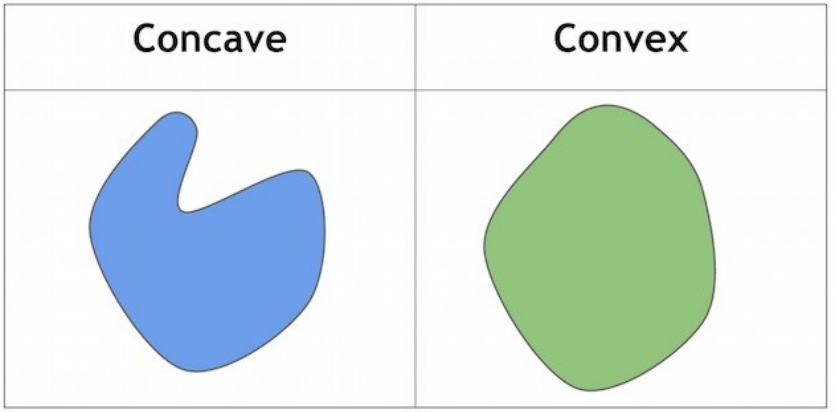
- Color (dark of white)
- Size
- Shape
  - Circularity
  - Convexity



Blob detection

We can filter by

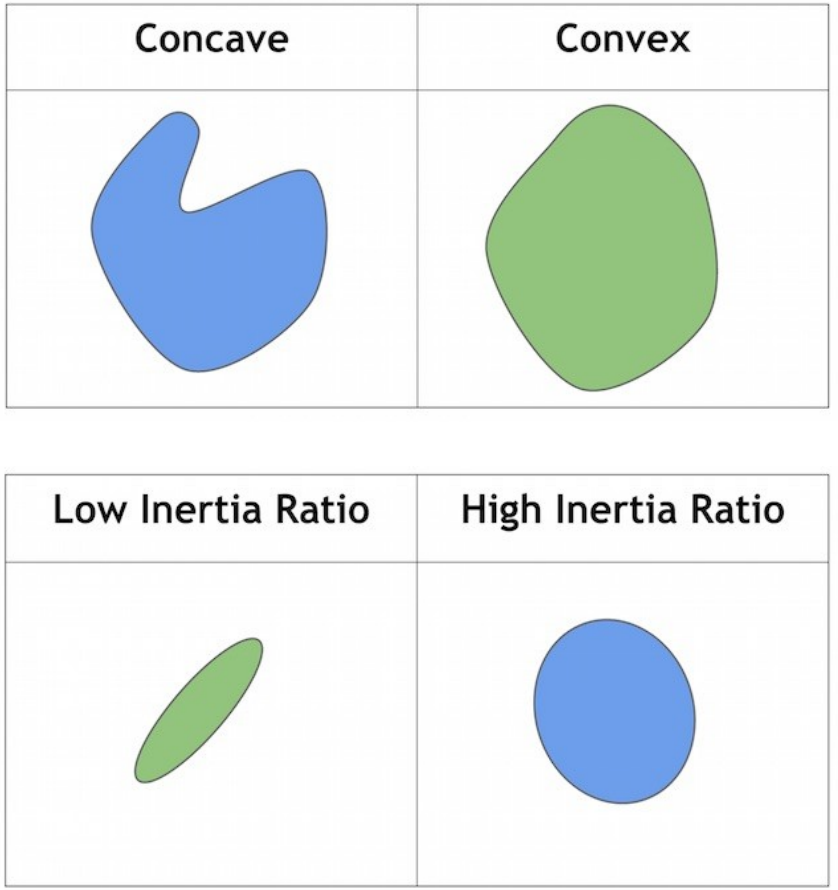
- Color (dark of white)
- Size
- Shape
  - Circularity
  - Convexity
  - Inertia ratio



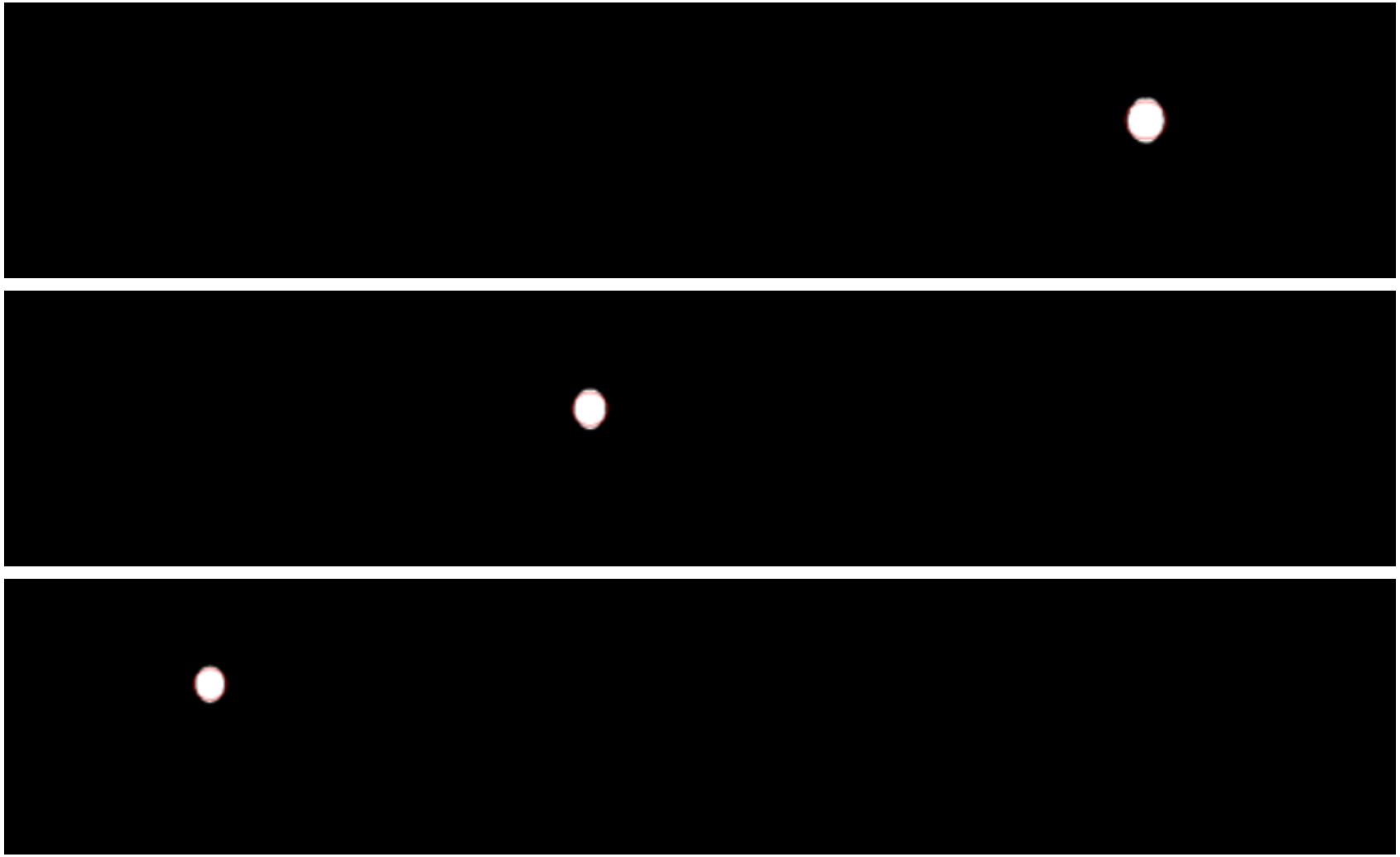
Blob detection

We can filter by

- Color (dark of white)
- Size
- Shape
  - Circularity
  - Convexity
  - Inertia ratio



Blob detection - input



Blob detection - output

Distance calculation

Triangle similarity

Objects of same shape but not same size

$$f = \frac{P \times D}{W}$$

$$\begin{cases} f \text{ is the focal length} \\ D \text{ is the distance from camera to object} \\ W \text{ is the real objects' width} \\ P \text{ is the apparent width in pixels} \end{cases}$$

Distance calculation

Triangle similarity

Objects of same shape but not same size

$$f = \frac{P \times D}{W}$$

$$\begin{cases} f \text{ is the focal length} \\ D \text{ is the distance from camera to object} \\ W \text{ is the real objects' width} \\ P \text{ is the apparent width in pixels} \end{cases}$$

Focal length previously obtained from calibration

Distance calculation

Triangle similarity

Objects of same shape but not same size

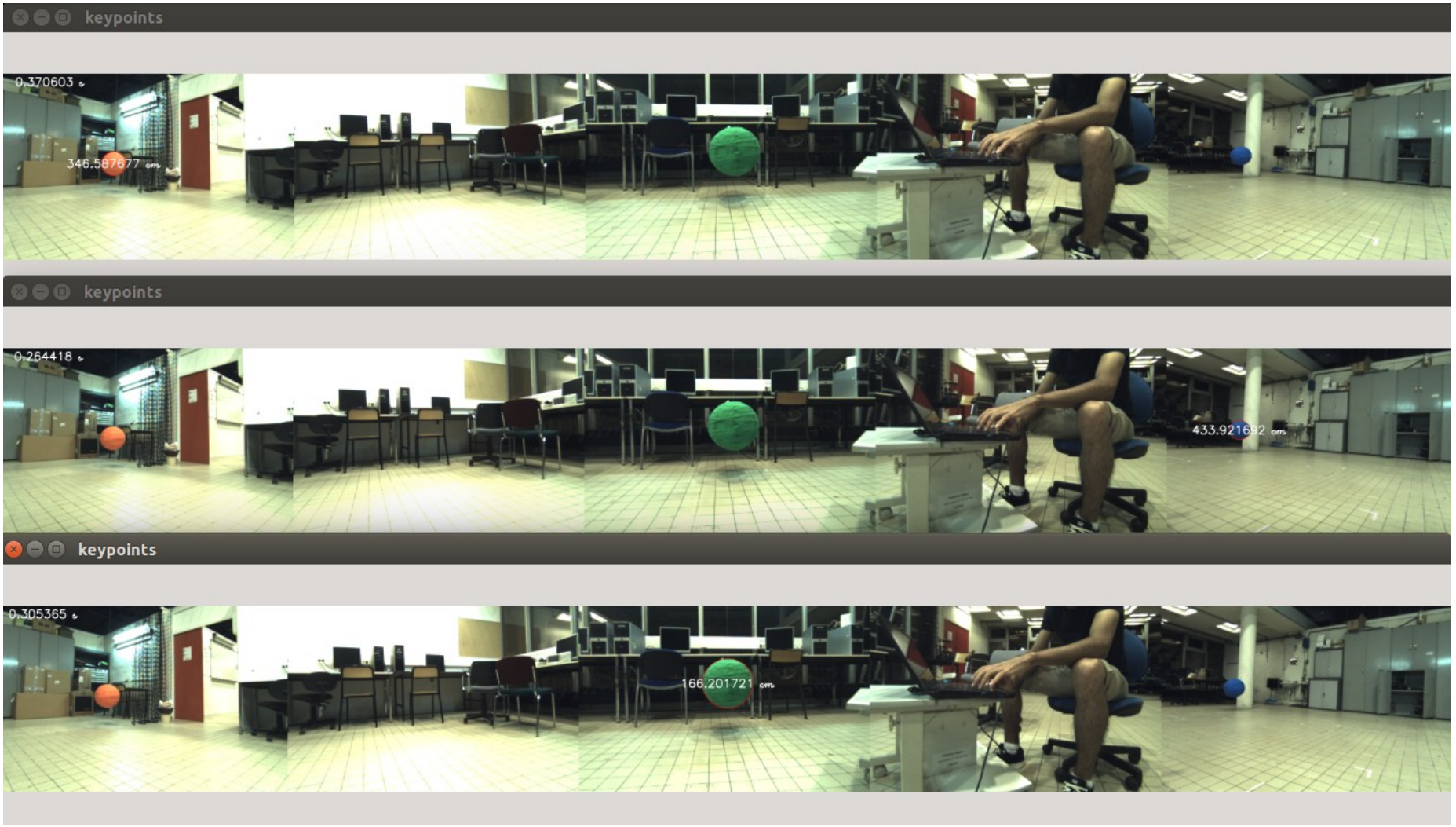
$$f = \frac{P \times D}{W}$$

$$\begin{cases} f \text{ is the focal length} \\ D \text{ is the distance from camera to object} \\ W \text{ is the real objects' width} \\ P \text{ is the apparent width in pixels} \end{cases}$$

Focal length previously obtained from calibration

$$D = \frac{f \times W}{P}$$

Distance calculation

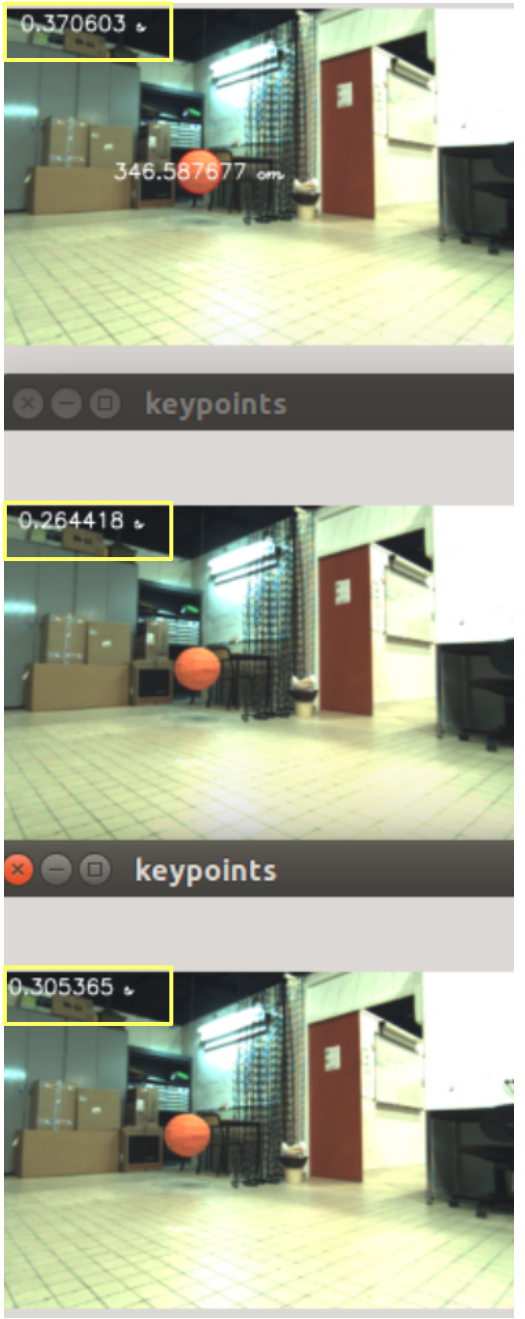


Time performance

We complete process a video frame at a rate of approximately 2ps.

Slow :(

Why?



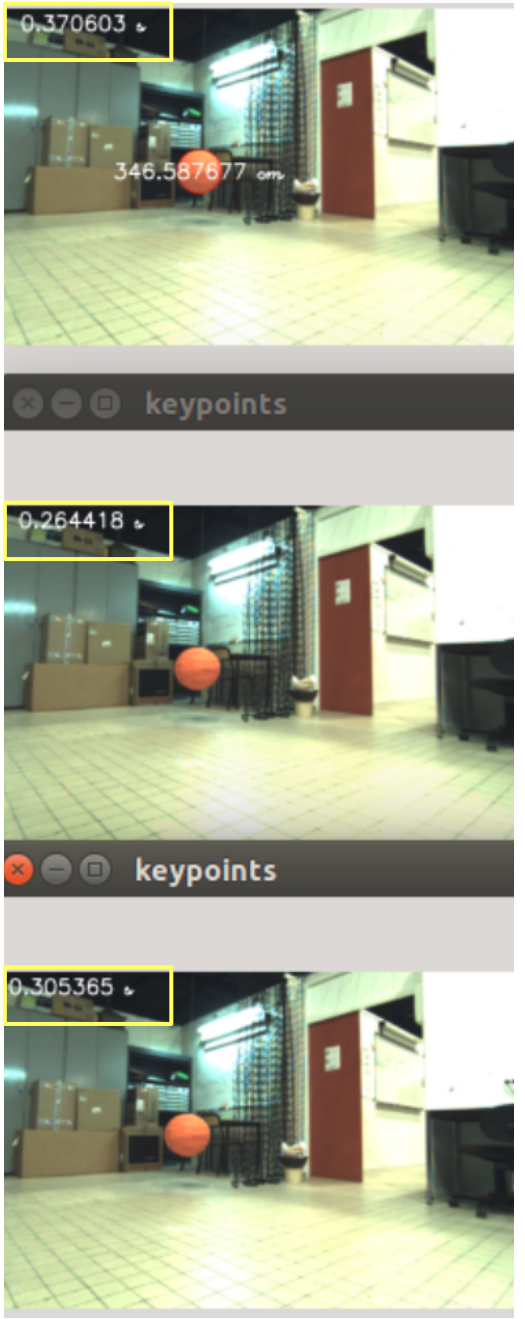
Time performance

We complete process a video frame at a rate of approximately 2ps.

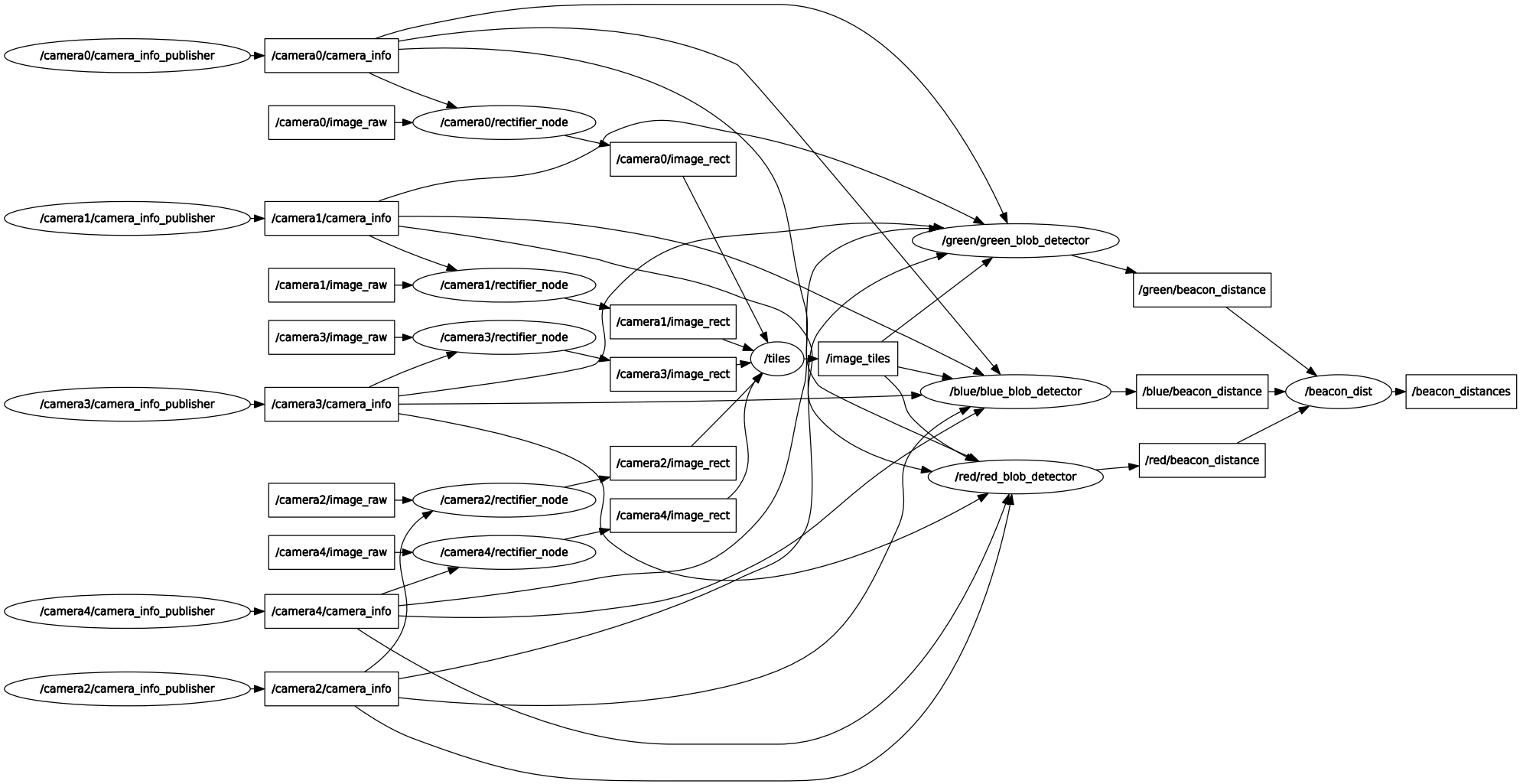
Slow :(

Why?

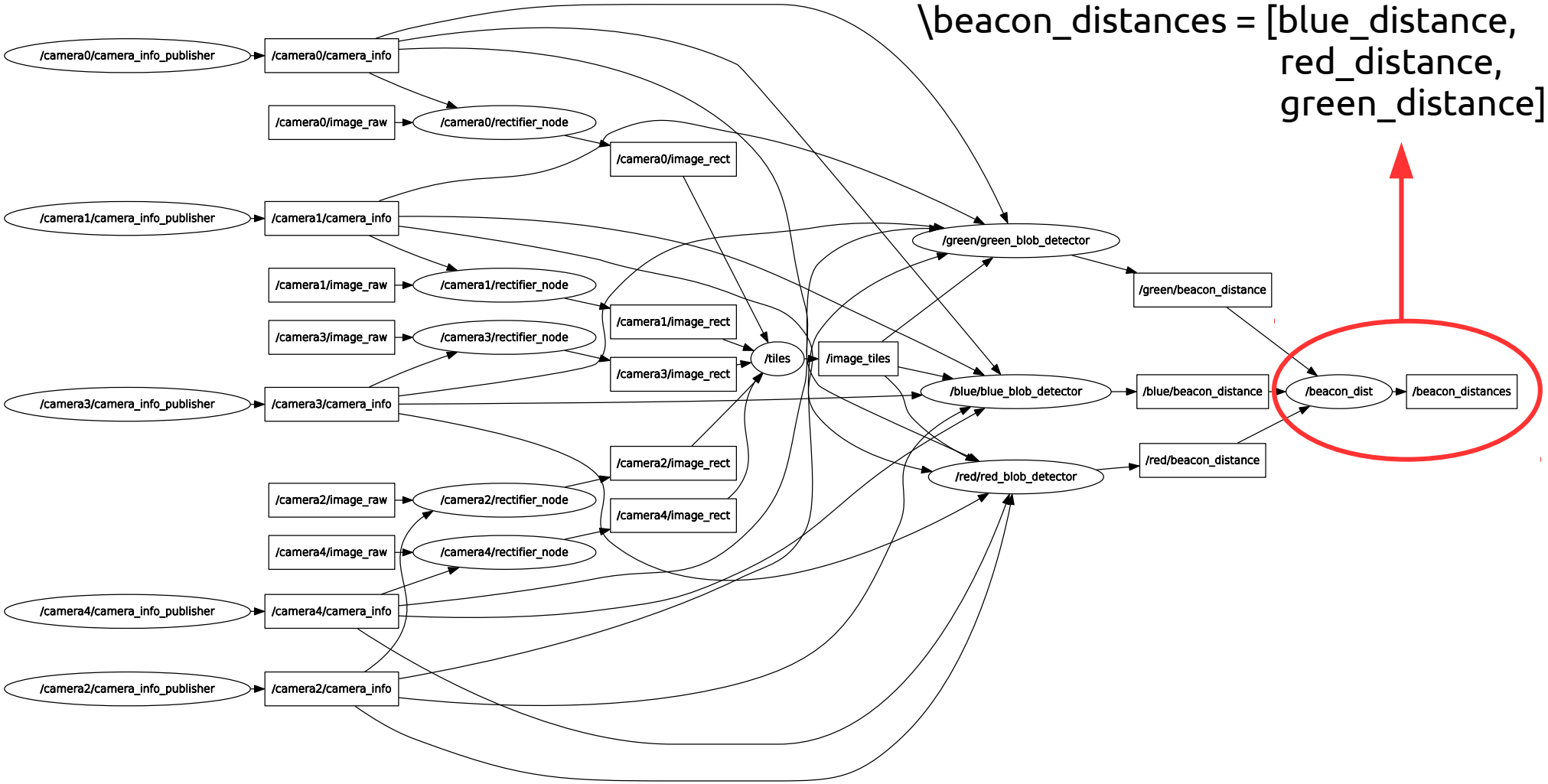
- 1.8 MP image
- Other nodes running on the same computer (ROS master)



ROS rqt_graph



ROS rqt_graph



Robot Localization

- **Absolute Localization**
- **Relative Localization**
- **Sequential use of relative and Absolute Localization**
- **Hybrid Localization**

Simple Example

- Let's consider that TurtleBot are forced to move along one direction -supposing +ve x-axis direction.

Simple Example

- **Available information:**

I. Wheels rotation – Using encoders.

II. Distance between the robot and the blue beacon (located at Origin) -using Camera, this distance represent the x- coordinate.

III. Kinematic model of the turtleBot- 2,0 robot.

IV. Geometrical parameters of the TurtleBot.

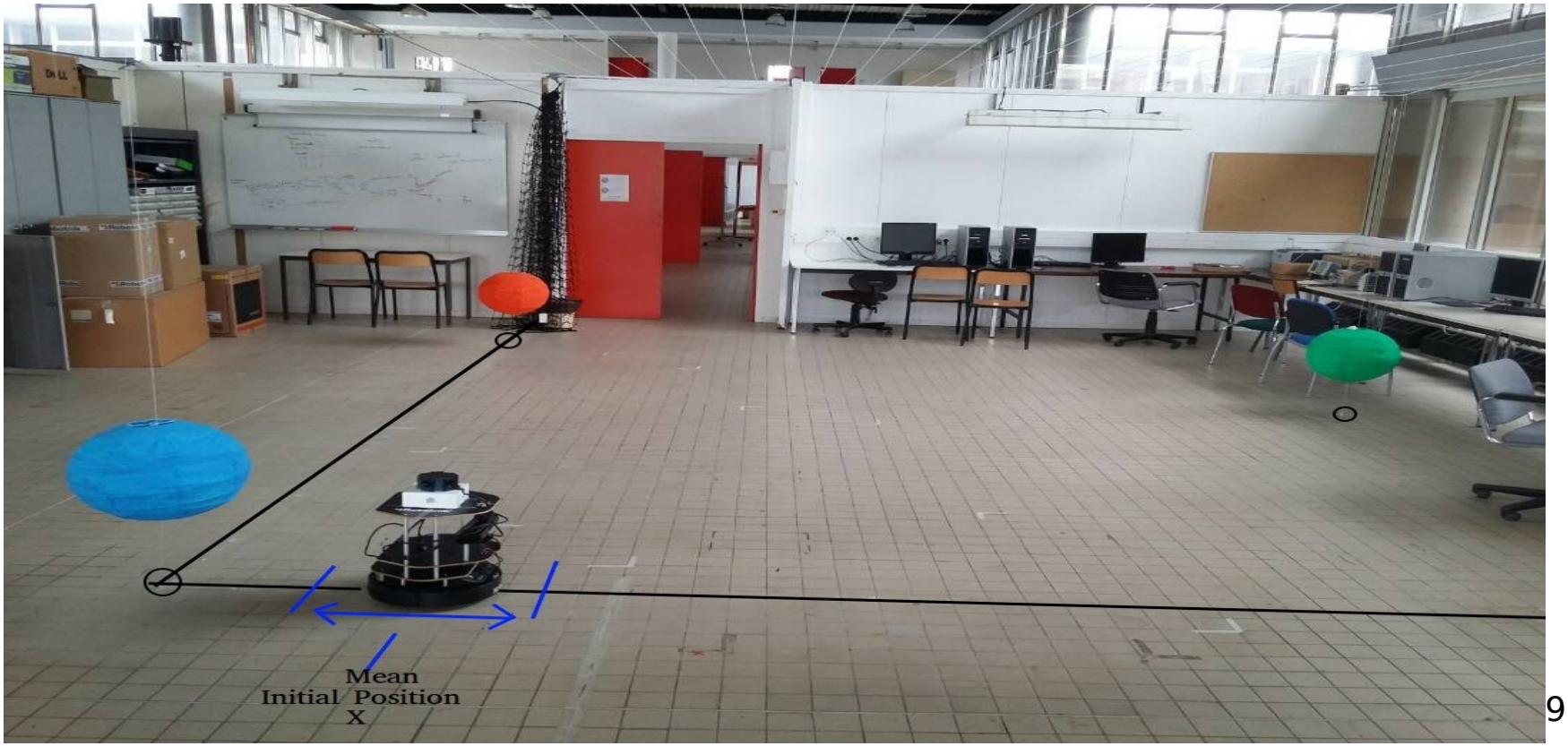
V. Initial position of turtleBot.

- **Objective is to:**

  determine position of TurtlBot (x-coordinate)

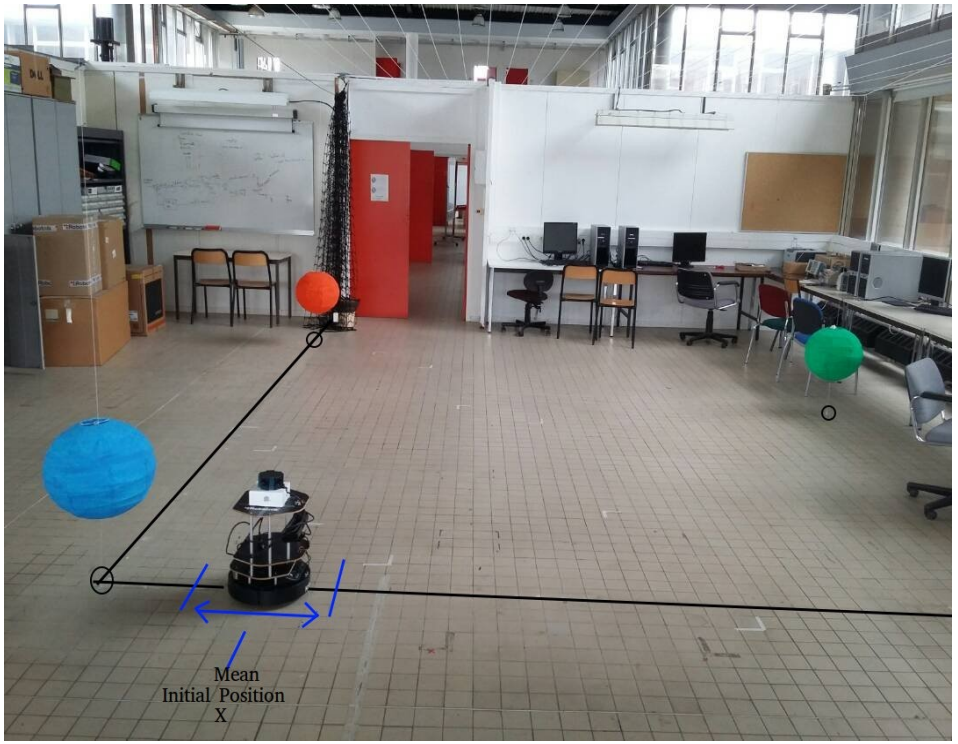
Simple Example

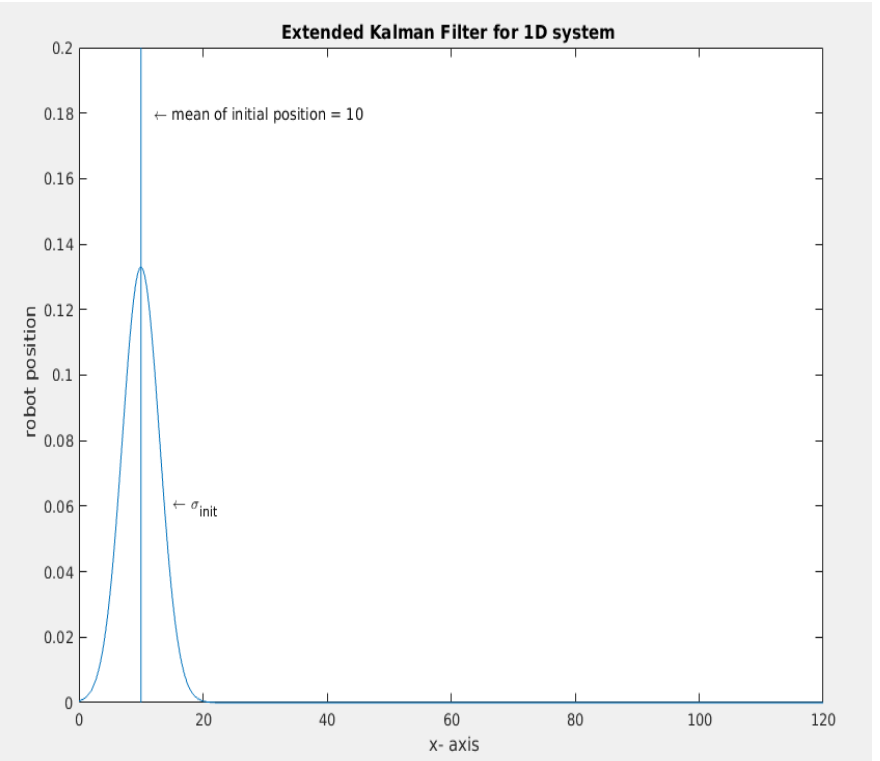
Attention: The available data has certain accuracy level.
Initial position of turtleBot.
Encoders resolution.
Distance measured by the Camera
Geometrical parameters of the TurtleBot.



Simple Example

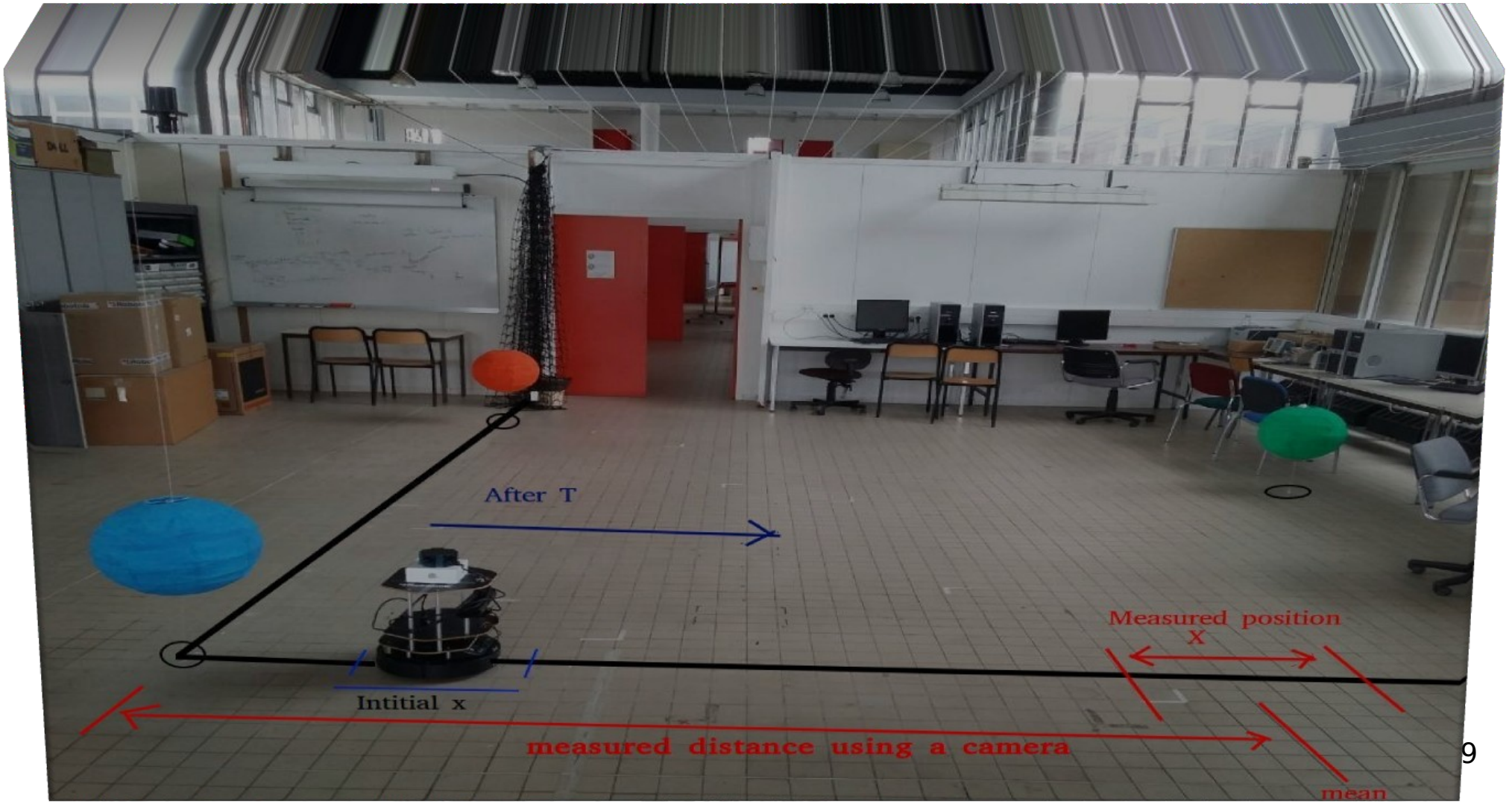
Attention: The available data has certain level of accuracy.
Initial position of turtleBot.
- can be represented using normal distribution.
- mean, variance.





Absolute Localization

- Using only the measurement to determine the position of the robot:



Relative Localization

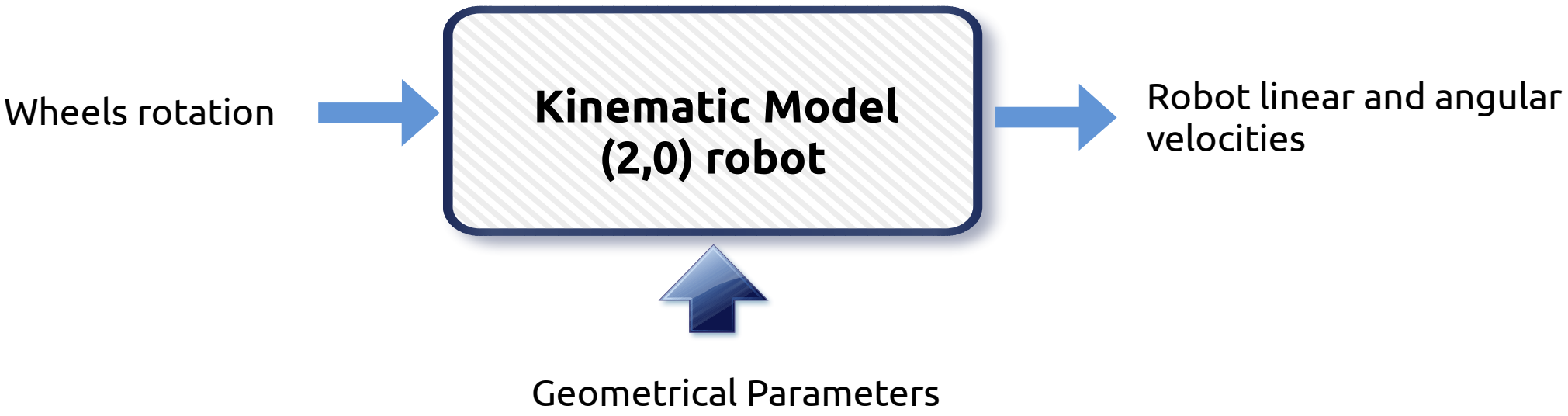
Odometry

- odometry is the use of data from motion sensors to estimate change in position over time.

- using the kinematic model of the robot - TurtleBot is (2,0) robot- we can express the translational and rotational speed of the robot as functions of the rotation speed of some of the robot’s wheels.

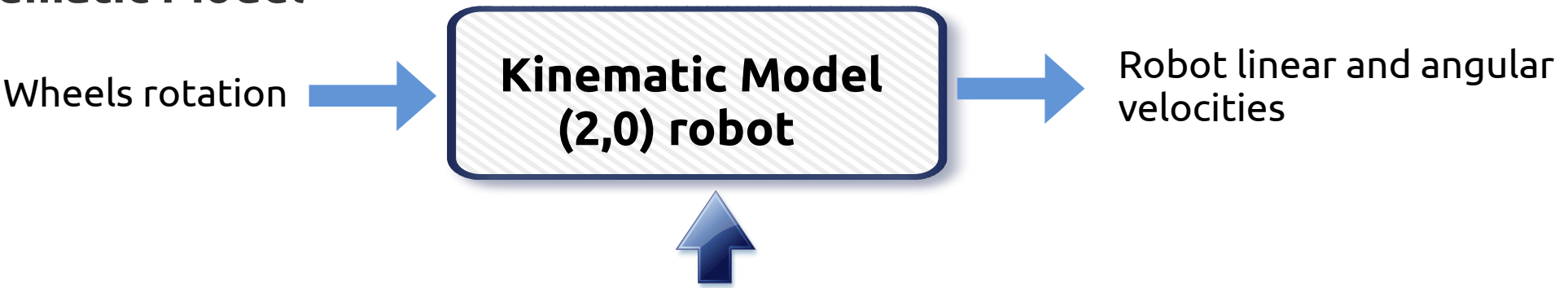
Relative Localization

Kinematic Model



Relative Localization

Kinematic Model

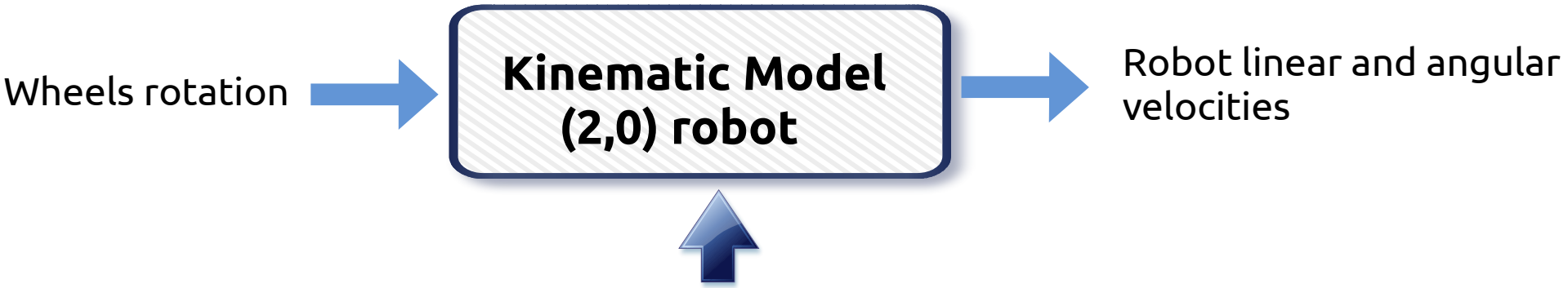


$$\begin{cases} v = \left(r_r \dot{q}_r + r_l \dot{q}_l\right)/2 \\ \omega = \left(r_r \dot{q}_r + r_l \dot{q}_l\right)/e \\ \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{\theta} = \omega \end{cases}$$

r_r	Right wheel radius.
r_l	Left wheel radius.
$\dot{q}_r$	Right wheel rotation velocity.
$\dot{q}_l$	Left wheel rotation velocity.
v	Linear velocity.
w	Angular velocity.

Relative Localization

Discrete Kinematic Model

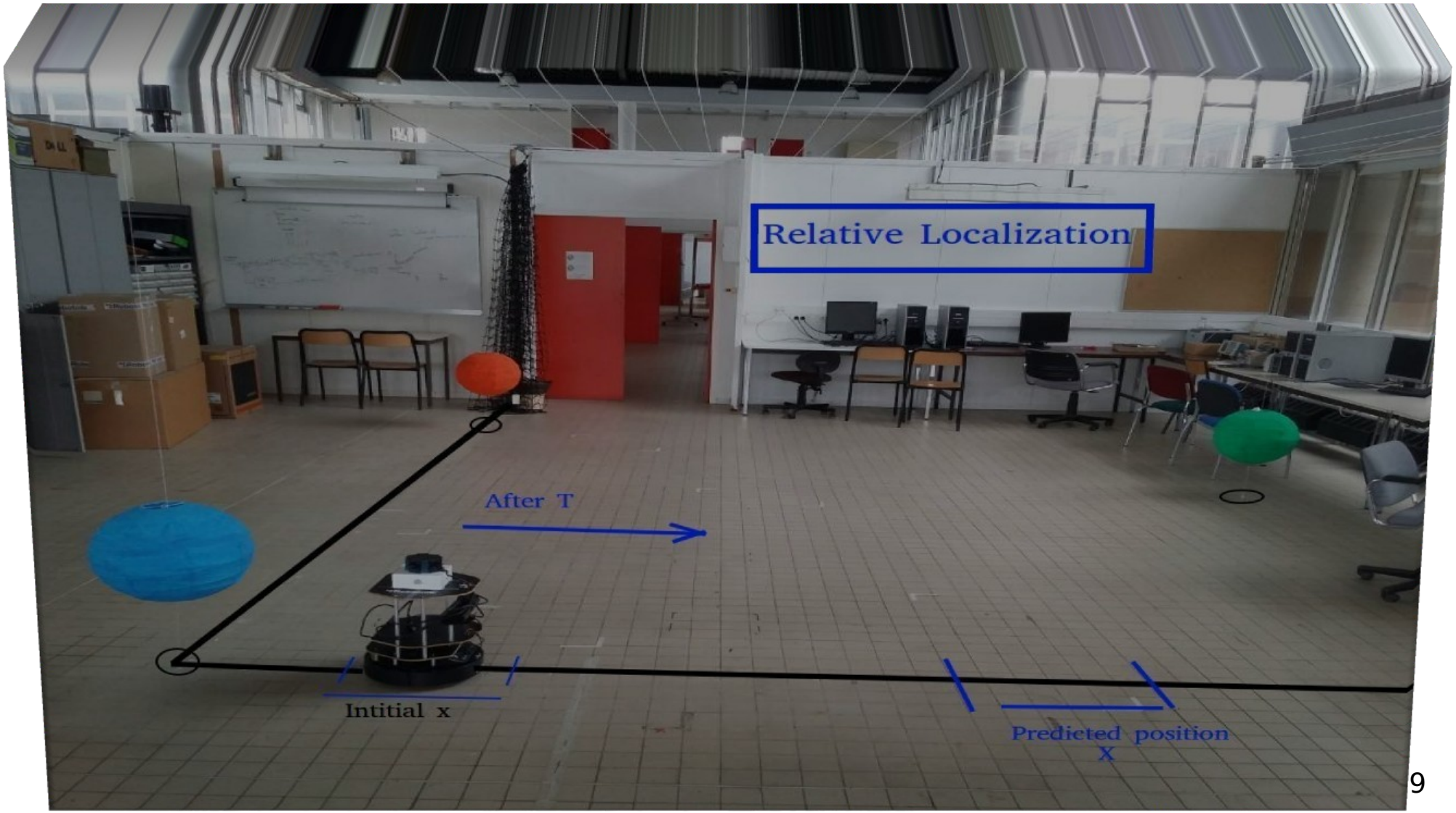


$$X_{k+1} = \begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \Delta D_k cos(\theta_k) \\ y_k + \Delta D_k sin(\theta_k) \\ \theta_k + \Delta\theta_k \end{bmatrix}$$

$$U_k = [\Delta D_k, \theta_k]$$

Relative Localization

Odometry



Relative Localization

Odometry uncertainty

Odometry precision depends on the acurracy of the geaometrical paramteres and on the accuracy of the wheels rotation velocity.

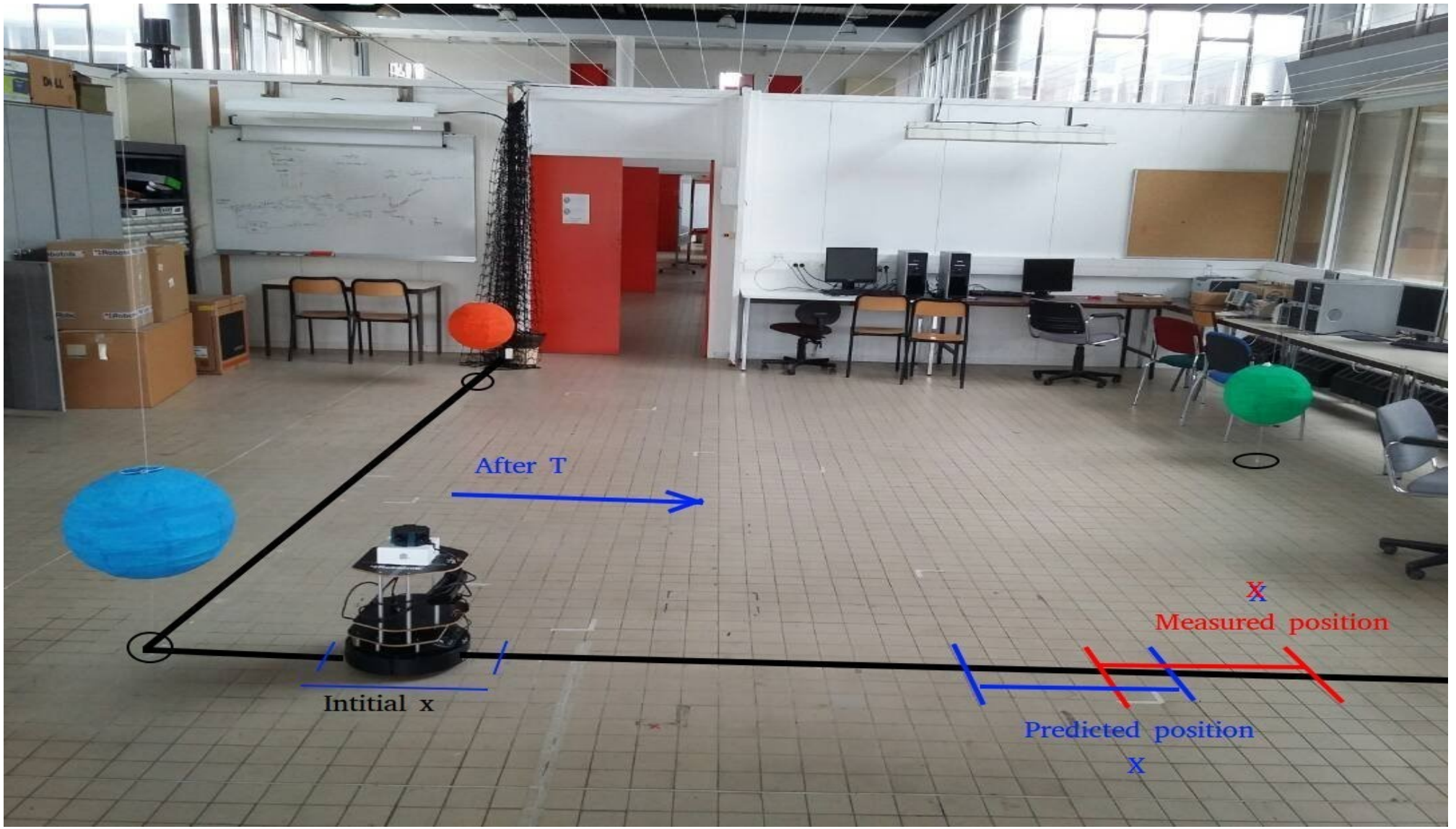
Odometry uncertainty increases with the distance the robot has moved and with time.

Sequential use of Relative and Absolute Localization

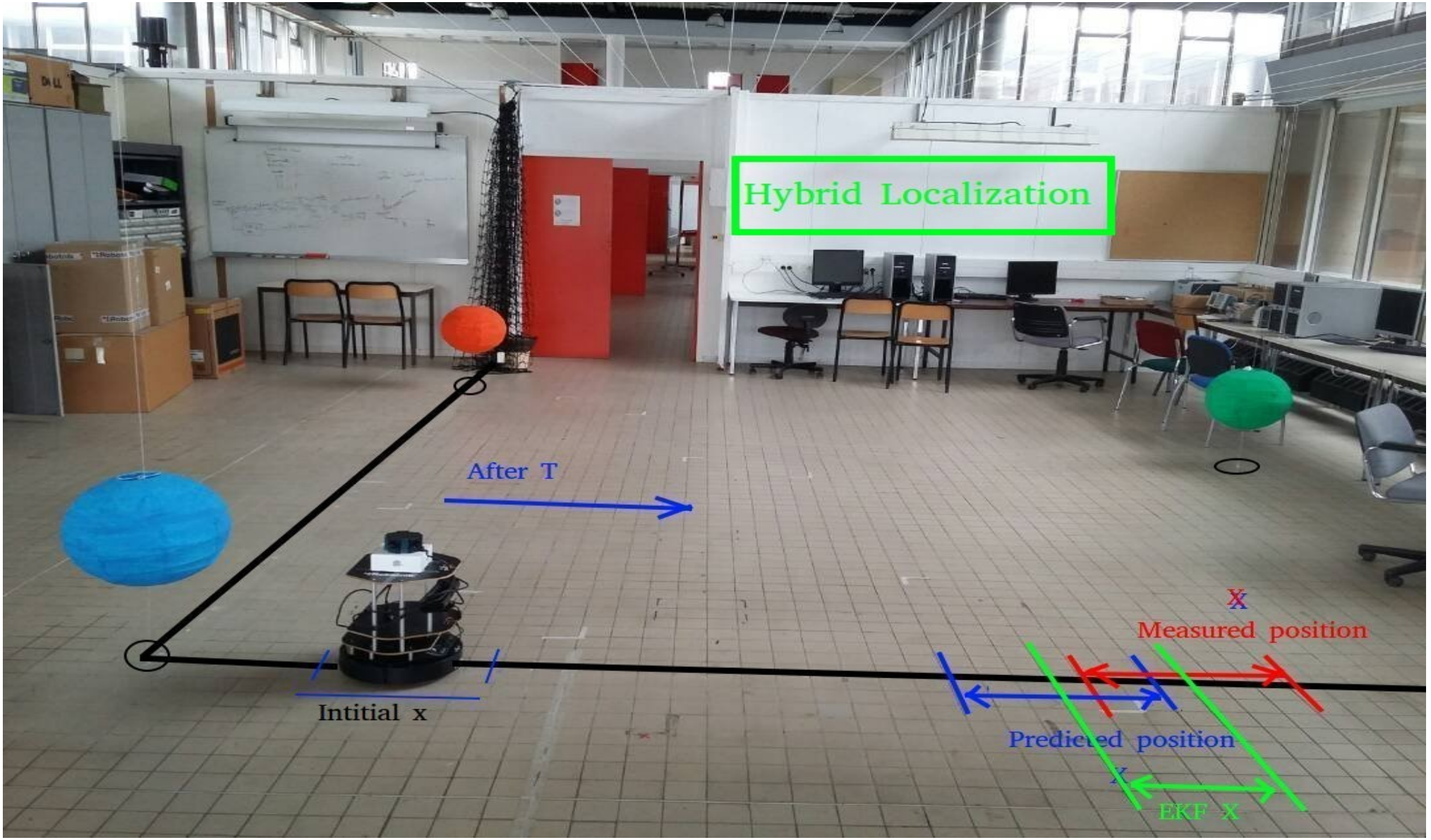
- The possobility of using relative localization for a while and periodically recalculate the position using the absolute localization – when the measurements are available.
- Each time we calculate the location using the absolute localization, we reset the high uncertainty due to the relative localization.
- It is convent to calculate the absolute localization, before the error of relative localization becomes too large.

Hybrid Localization

Combining both the relative and the measurement.



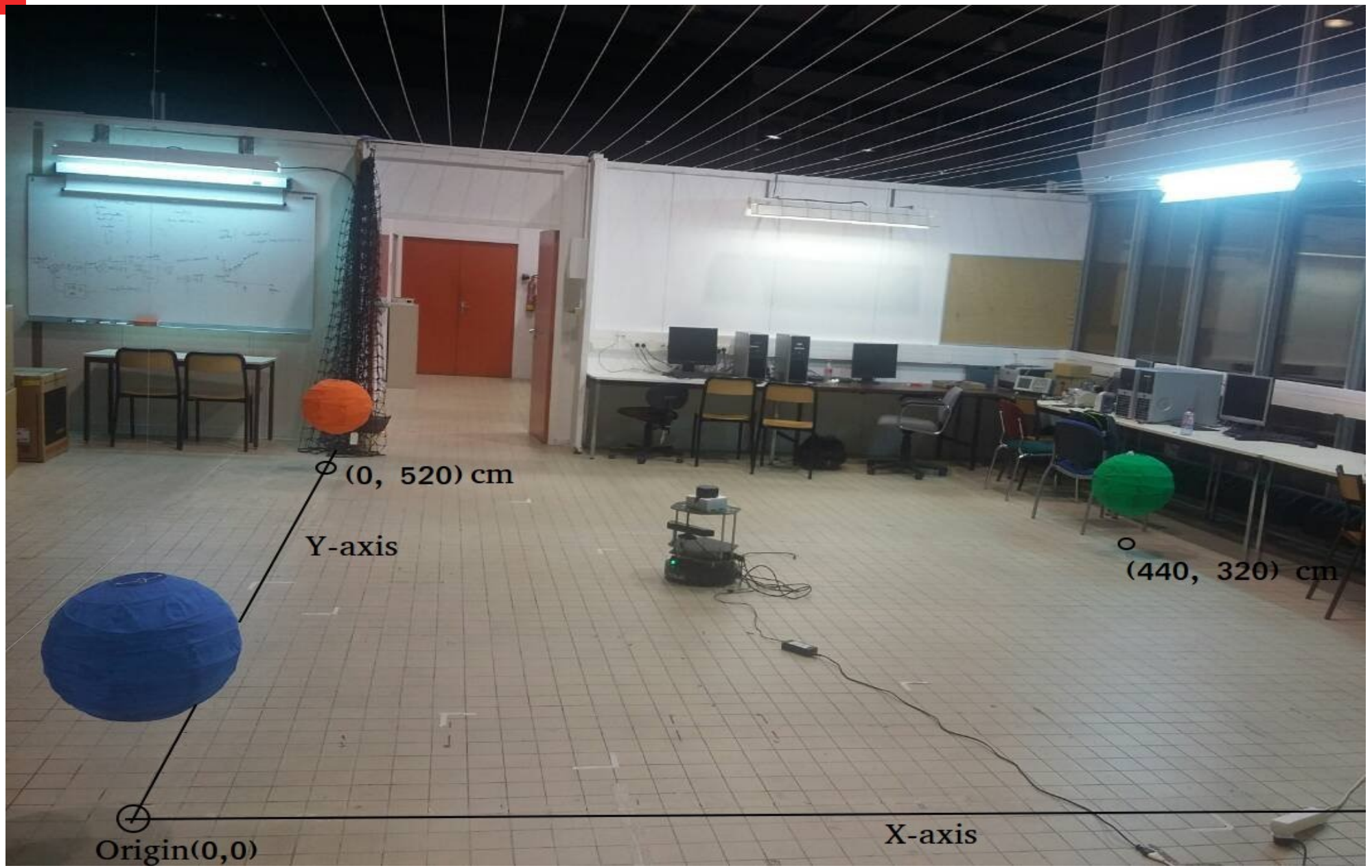
Hybrid Localization Extended Kalman Filter



Extended Kalman Filter

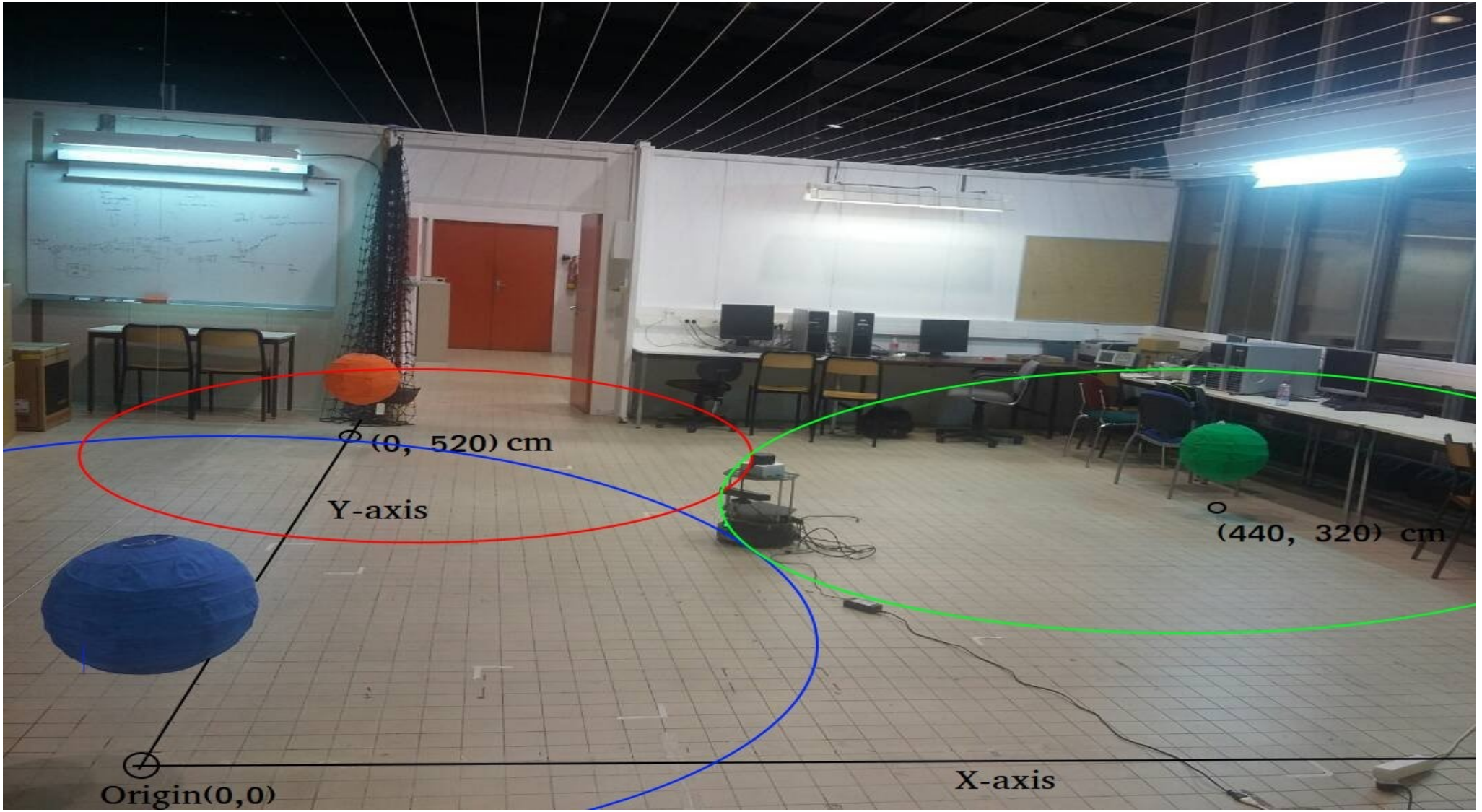
An algorithm that uses a series of measurements observed over time, containing some noise and inaccuracies, to modify the inaccurate prediction state of the system and produces an estimation of the state of system tends to be more accurate than those base on predicition/ evolution model.

Recall the Environement



TurtleBot Localization in 2D Plan

- Absolute Localization: using trilateration.

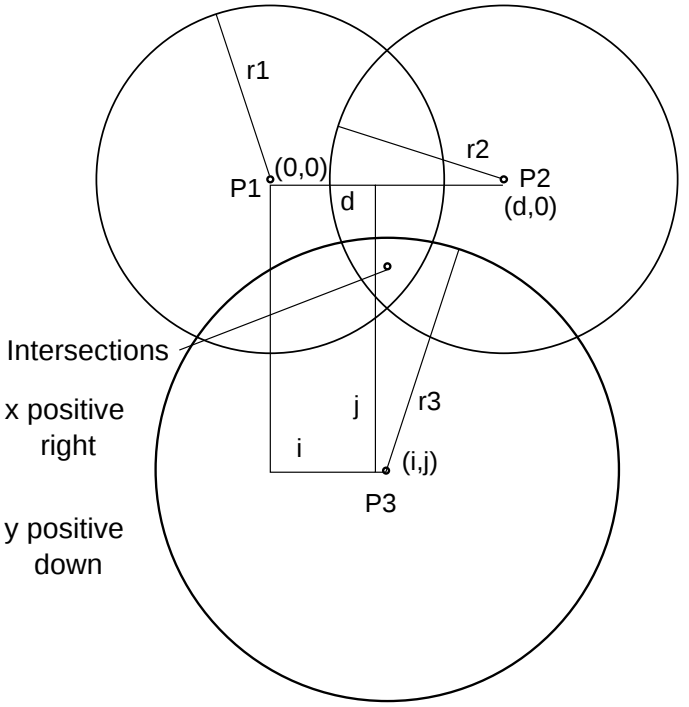


Trilateration

Use geometry to find intersection of three spheres

Special case conditions:

- Spheres must intersect
- All centers in $z = 0$
- Center P_1 at the origin
- Center P_2 on x-axis



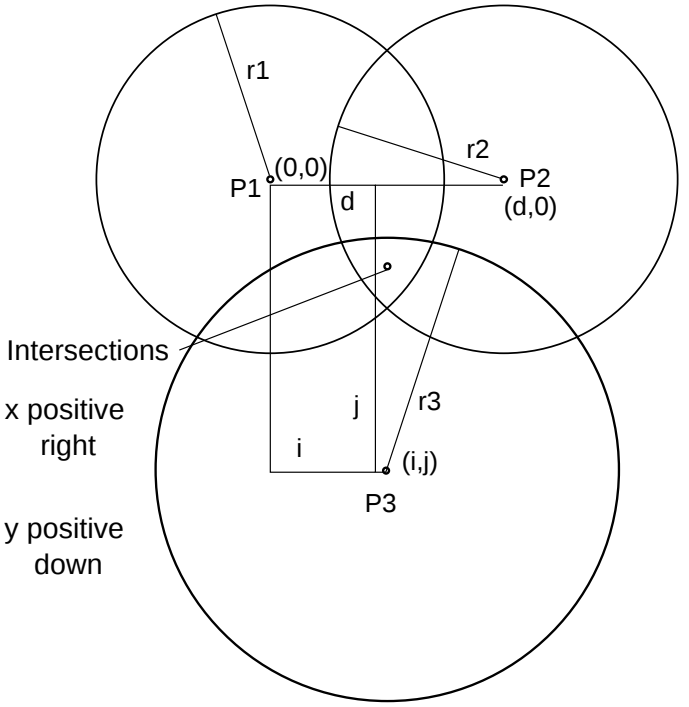
https://en.wikipedia.org/wiki/Trilateration#/media/File:3spheres.svg

Trilateration

The resulting equations for the intersection point are:

$$x = \frac{r_1^2 - r_2^2 + d^2}{2d}$$

$$y = \frac{r_1^2 - r_3^2 + i^2 + j^2}{2j} - \frac{i}{j}x$$



https://en.wikipedia.org/wiki/Trilateration#/media/File:3spheres.svg

Optimization technique

RobOptim is a numercial optimization library applied to robotics for C++.

RobOptim interfaces with various state-of-the-art solvers and lets you express your problem in C++ to ensure optimal performances. Similar *fmincon* in MATLAB

Caveat: It takes time to install because it needs quite a lot of dependencies and writing the CmakeFile.txt is not so intuitive.

Optimization technique

Another alternative would be to minimize a function given certain constraints.

Objective

$$\min \sum_{i=1}^{3} (x_i - x)^2 + (y_i - y)^2$$

Constraints

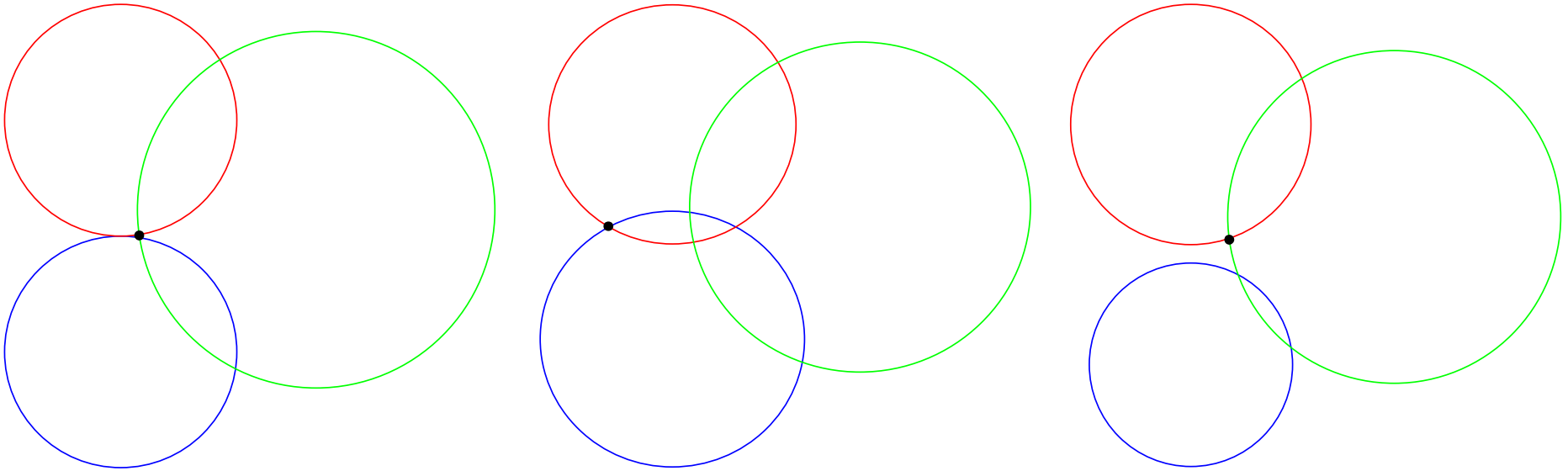
$$g_i = (x_i - x)^2 + (y_i - y)^2 \geq d_i^2$$

where

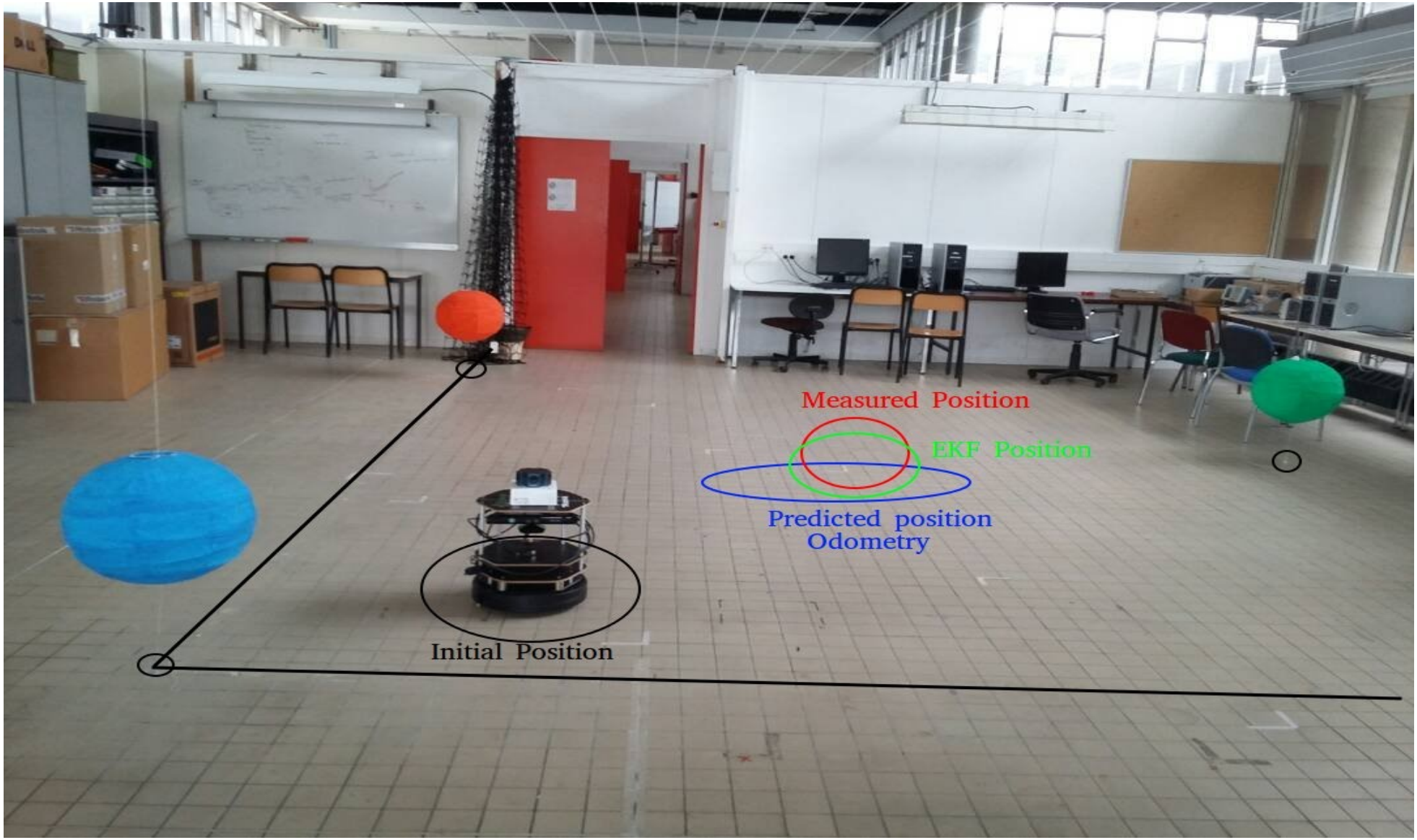
$$\begin{cases} (x, y) \text{ are the 2D coordinates of the robot} \\ (x_i, y_i) \text{ are the 2D coordinates of the i-th beacon} \\ d_i \text{ is the distance from the camera to the i-th beacon} \end{cases}$$

Optimization technique

Not so << optimal >>



TurtleBot Localization in 2D Plan EKF



TurtleBot Odometry

- TurtleBot publishes its pose and its twist through a topic called **"\odom"** using '**nav_msgs/Odometry**' message.

```
m@ali:~$ rosmsg show nav_msgs/Odometry
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

Odometry error and uncertainty

why we do not trust odometry:

- Odometry is sensitive to errors due to the integration of velocity measurements over time to give position estimates.
- ***Systematic errors:***

  Systematic errors are very serious because their effects accumulate over time, we can consider them the dominant errors on smooth indoor surfaces:

  • Wheel radius and track gauge errors.
  • Misalignment of wheels.
  • Non-point wheel-to-floor contact.
  • Finite resolution and sampling rate.

  Systematic errors deterministic, so they can be eliminated through proper calibration

Odometry error and uncertainty

why we do not trust odometry:

- ***Non systematic errors:***
- Non systematic errors are difficult to handle because they are unexpected, so they can not be compensated, we can consider them the dominate errors on rough, and irregular terrains.
  - Slipping due to Slippery floor.
  - Skidding due to Over-acceleration and fast turning.

Odometry topic interface to EKF

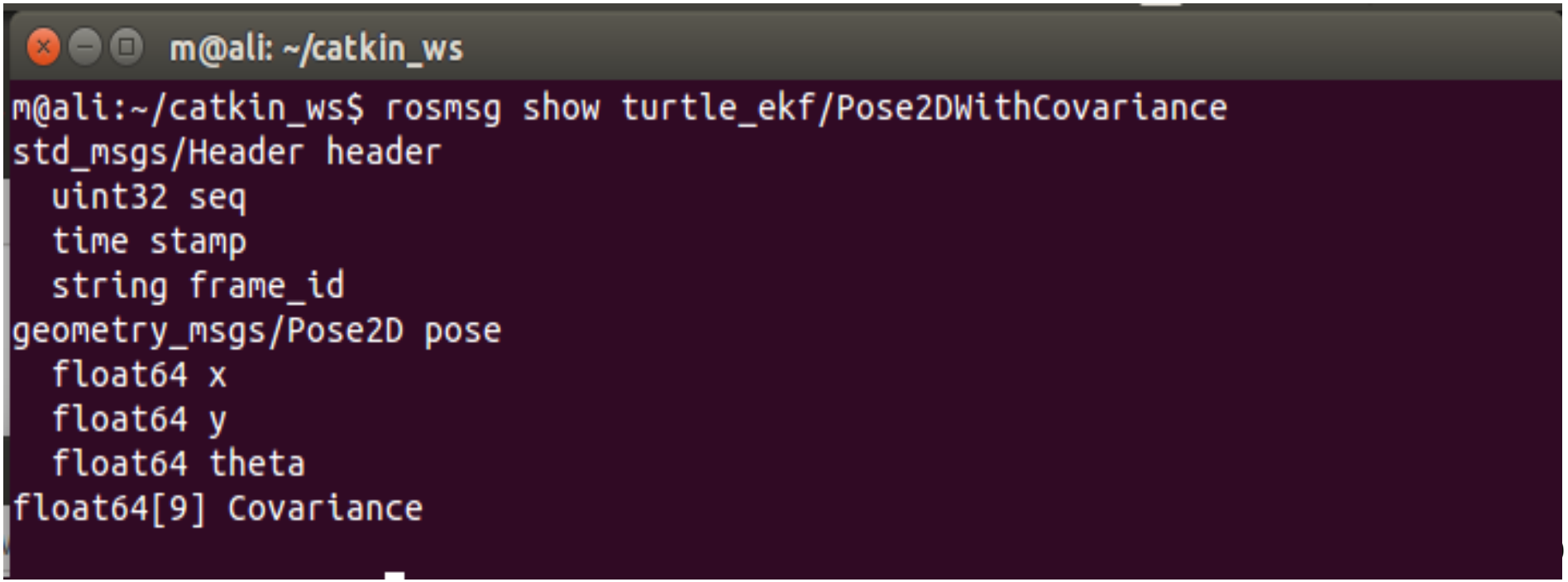
- ***nav msgs/Odometry*** message contains the 3D pose, orientation, tand wist of the robot plus their covariances.

- our objective is to localize the robot on a flat plan, so we just need 2D Pose to express the x,y coordinates and the orientation theta of the robot plus their covariance matrix.

Odometry topic interface to EKF

we have defined new message

- ***"turtle ekf/Pose2DWithCovariance"*** to be used associated with Extended Kalman filter node, it contains the relevant data we need from odometry ***"x, y, theta, and covariance matrix 3x3"***



Odometry topic interface to EKF

Quaternion to RPY angle Conversion:

```
/************************************ Steps: ********************************************
* subscribe to /odom topic and save the message in variable odom3D of type "nav_msgs/Odometry"
* define a message "odom2D" of type "turtle\_ekf/Pose2DWithCovariance" to be published
* define Quaternion of type tf transform using the received data
* form a rotation matrix m using the tf:quaternion
* define temporal variables roll, pitch, and yaw
* get the roll, pitch, and yaw angles from the rotation matrix m using m.getRPY method
* yaw is the rotation around z axis
* assign x,y,theta, and conariances values for the message that will be published
*****************************************************************************************/

   tf::Quaternion q(odom_3D.pose.pose.orientation.x,
                    odom_3D.pose.pose.orientation.y,
                    odom_3D.pose.pose.orientation.z,
                    odom_3D.pose.pose.orientation.w);

       tf::Matrix3x3 m(q);
       double roll, pitch, yaw;
       m.getRPY(roll, pitch, yaw);
       odom_2D.pose.theta=yaw;

       odom_2D.pose.x=odom_3D.pose.pose.position.x;
       odom_2D.pose.y=odom_3D.pose.pose.position.y;
       odom_2D.pose.theta=odom_3D.pose.pose.position.theta;

       odom_2D.Covariance[0]= odom_3D.pose.covariance[0];
       odom_2D.Covariance[4]= odom_3D.pose.covariance[7];
       odom_2D.Covariance[8]= odom_3D.pose.covariance[35];
```

What we measure

The measurement is the distances to the three beacons (blue – red – green).

The observation equation is:

$$g(X_K) = \begin{bmatrix} dist_{blue} \\ dist_{red} \\ dist_{green} \end{bmatrix} = \begin{bmatrix} sqrt(x - x_b)^2 + (y - y_b)^2 \\ sqrt(x - x_r)^2 + (y - y_r)^2 \\ sqrt(x - x_g)^2 + (y - y_g)^2 \end{bmatrix}$$

This equation $g(X_K)$ will be used to calculate the expected output from the predicted state to compare it with the measurement.

What we measure

Measuremnet error sources:

- Beacon:

size, it is supposed to be 45 cm.
shape, it is supposed to be perfect sphere.
Location, beacon centre is sipposed to be at (x_bi, y_bi).

- image processing algorithm:

Image sampling in x, and y direction 'pixels'
Focal length of each camera
Position of the camera with respect to the robot base.
Light, it is supposed to be indoor localization, but the project room walls made from glass

- Environement:

- Grid that is used for localization ' mosaic'.
- objects with the same color of one of the beacon and has a circular shape.

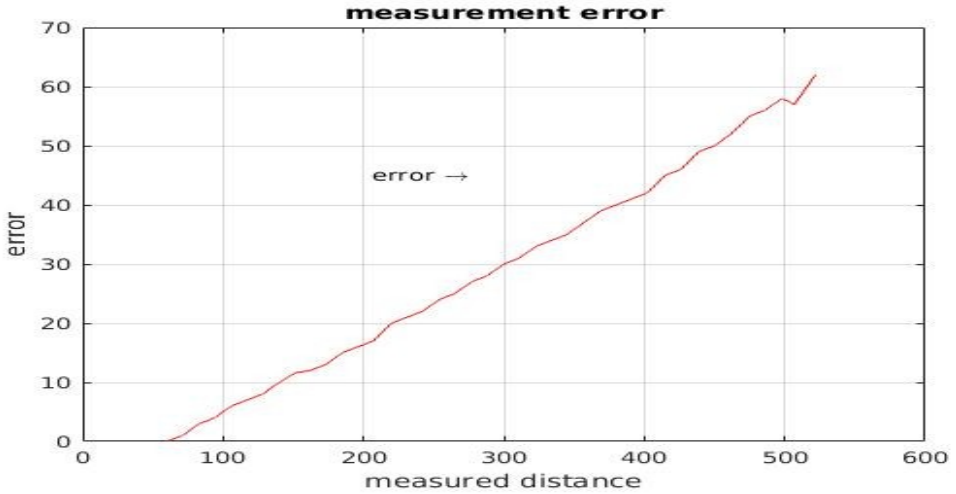
What we measure

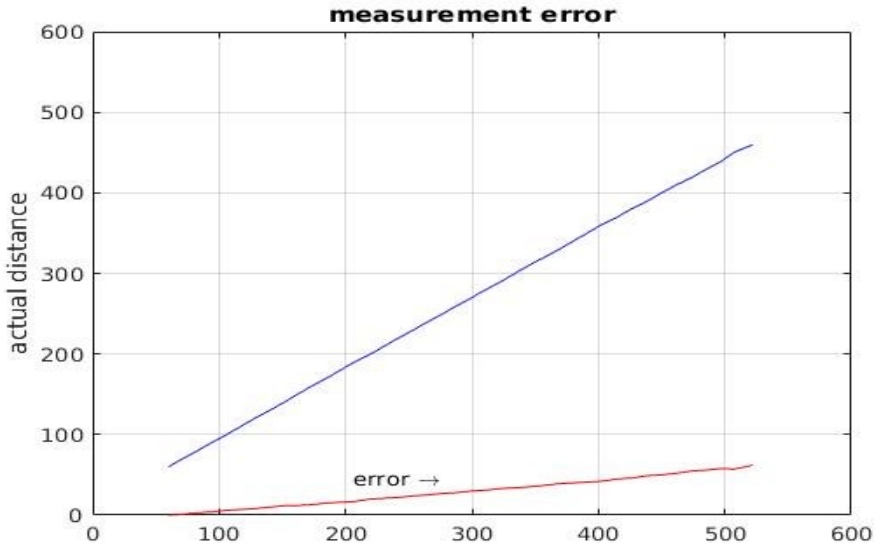
Unfortunately …

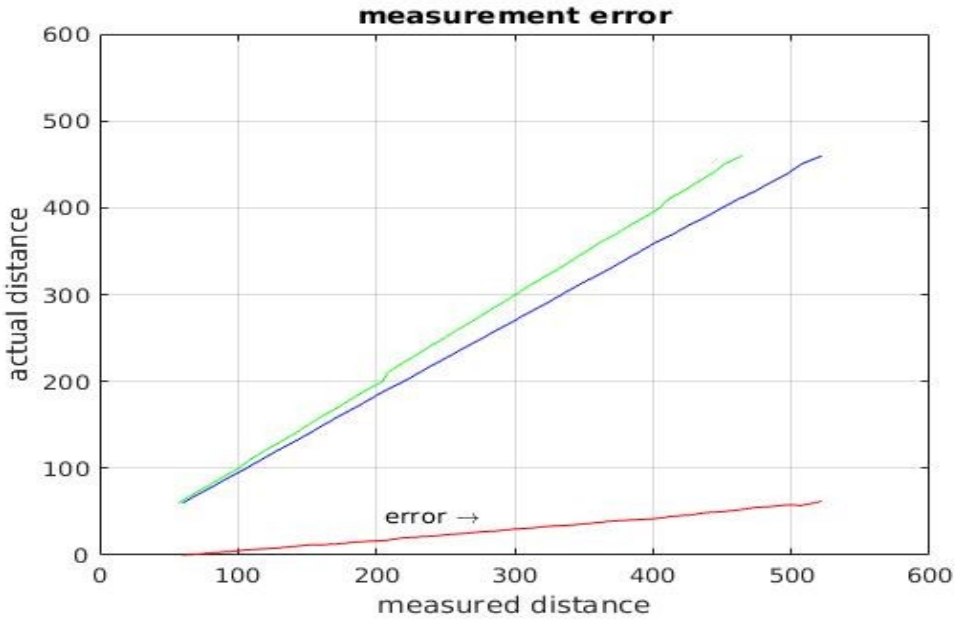
Our Environment is perfect for the odometry not for our measurement !

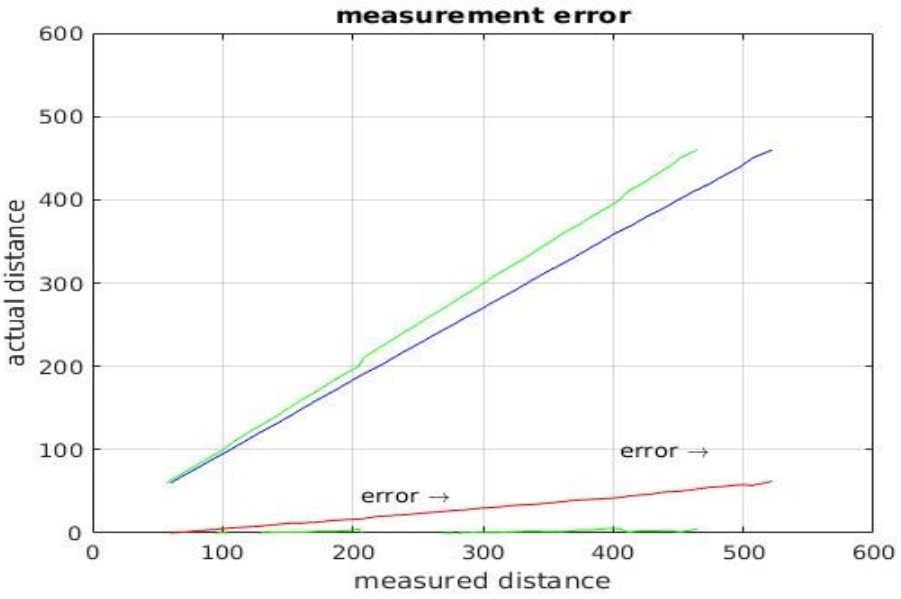
Measurement error analysis:

we recorded some measurements to calculate the measurement error and to measure how the measurement is accurate.



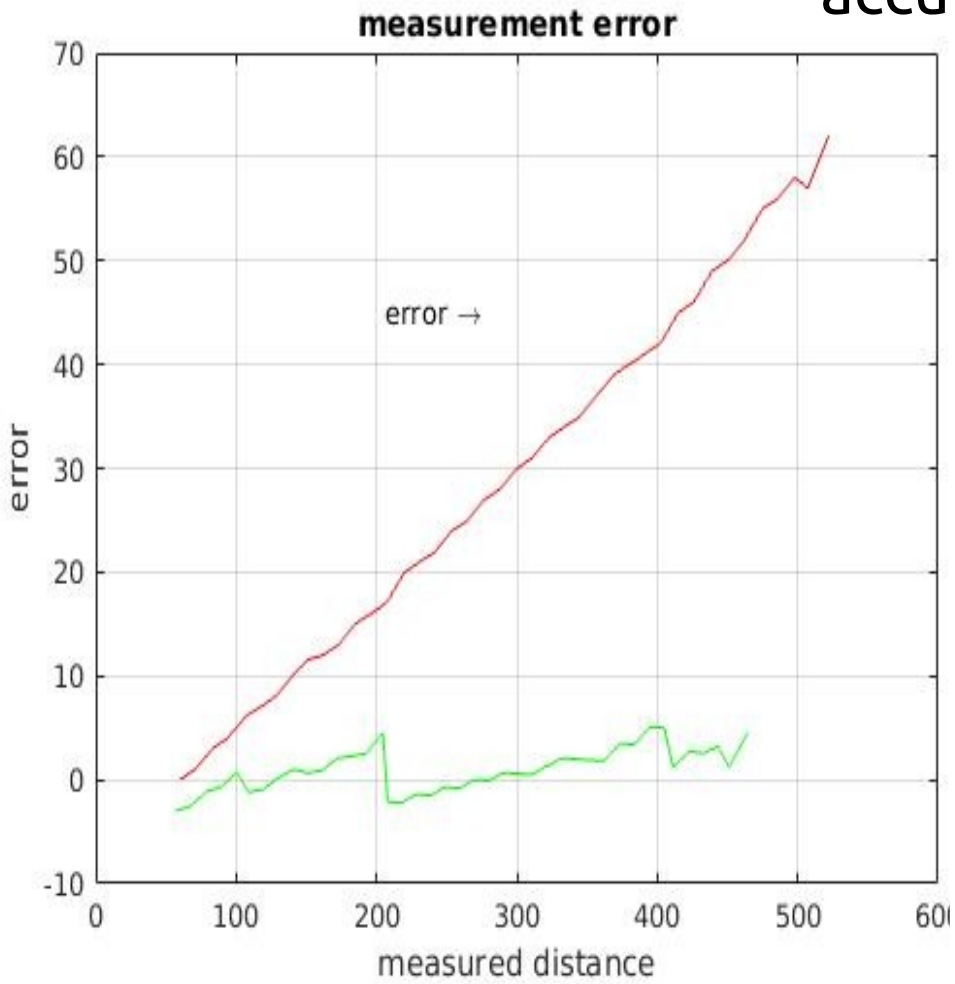


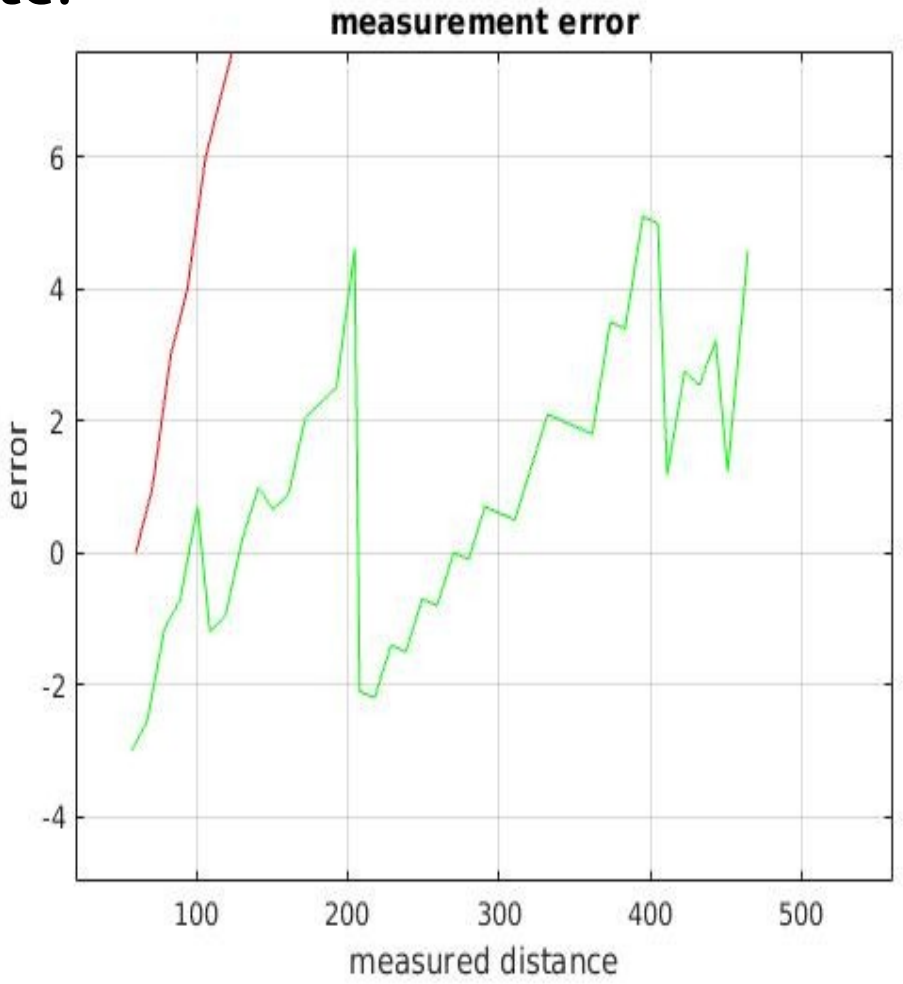




Measurement error analysis:

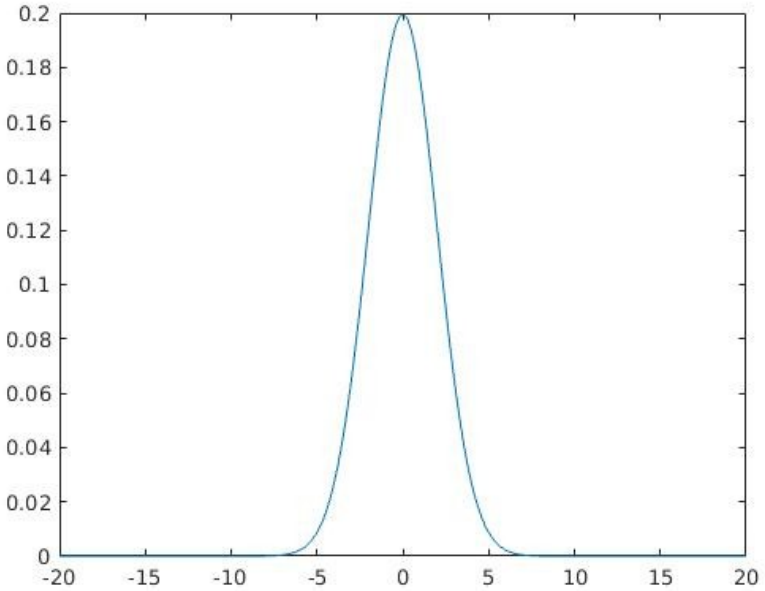
We recorded some measurements to calculate the measurement error and to measure how the measurement is accurate.





Measurement error analysis

- After error compensation:
  For calculating the variance related to the measurement.



Results

Total numbers (N):	41
Mean (average) value:	0.973853658537
Sample variance (s^2):	**4.40244287805**

EKF node

Node to implement the EKF algorithm:

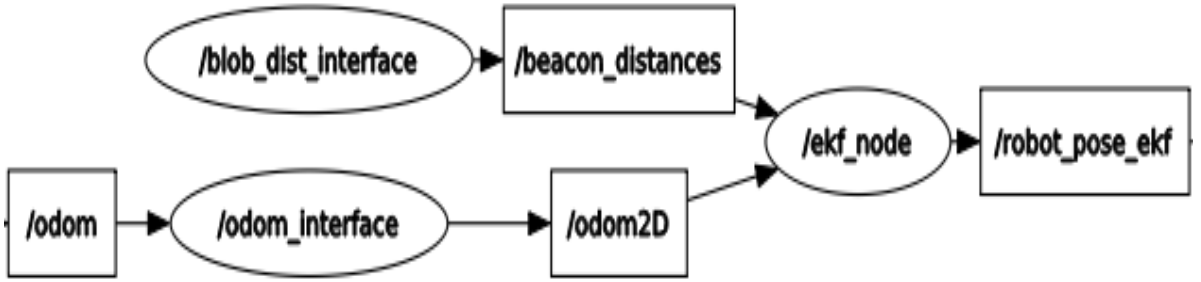
- **It subscribes to:**

/odom2D: this topic contains the odom2D pose (x,y,theta) of the turtlrbot and their covariance.

/beacon distances : this topic contains three distance from the robot to each beacon in order "blue, orange/red, green".

- **It publishes to:**

/robot pose ekf: it publishes the 2D pose (x,y,theta) of the turtlrbot and their corresponding
covariance after modifying the predicted pose using the measurement.



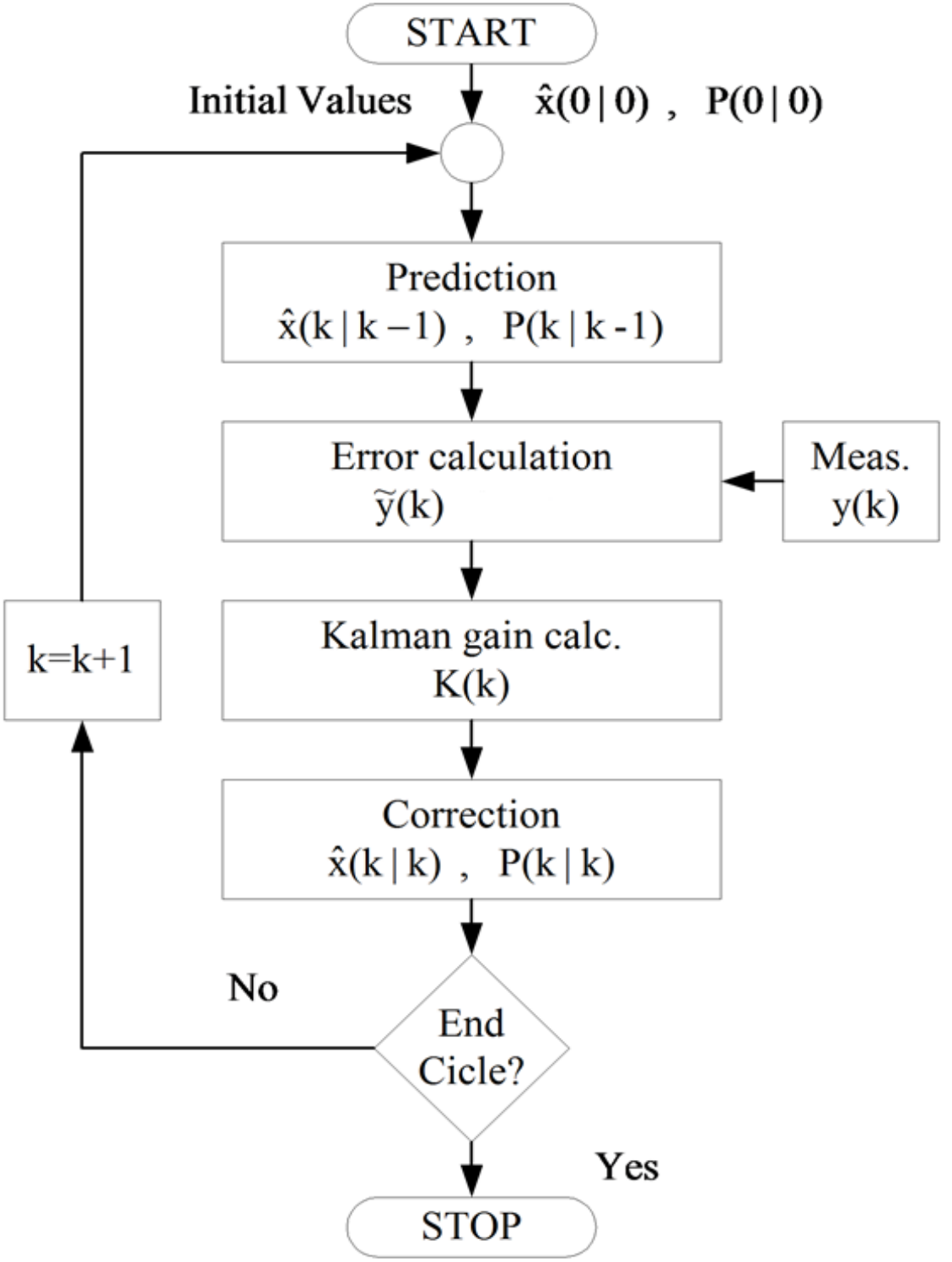
EKF Algorithm

Prediction Model:

- calculate of the predicted state.
- update the covariance matrix.

Correction Step:

- error calculation between the measurement and the expected measurement.
- Calculation of Kalman Gain K.
- Modify the predicted state.
- update the covariance matrix.



EKF Equations

Evolution Model – measurement/observation equation:

$$X_{k+1} = f\left(X_k, U_k^*\right) + \alpha_k$$

$$U_k^* = U_k + \beta_k$$

$$Y_k = g_{mag}\left(X_k\right) + \gamma_k$$

Prediction Model:

$$\hat{X}_{k+1|k} = f\left(\hat{X}_{k|k}, {U^*}_k\right) + \alpha_k$$

$$\hat{P}_{k+1|k} = A_k\, P_{k|k}\, {A_k}^T + B_k\, Q_\beta\, {B_k}^T + Q_\alpha$$

Where:

$$\begin{cases} A_k = \frac{\partial f}{\partial X}\left(\hat{X}_{k|k}, {U^*}_k\right) \\ B_k = \frac{\partial f}{\partial U}\left(\hat{X}_{k|k}, {U^*}_k\right) \end{cases}$$

EKF Algorithm

Correction Step:

$$\hat{X}_{k+1|k+1} = f\left(\hat{X}_{k+1|k}, {U^*}_k\right) + K_k\left(Y_k - \hat{Y}_k\right)$$

$$\hat{P}_{k+1|k+1} = \left(1 - K_k\, C_k\right) P_{k+1|k}$$

Where

$$\begin{cases} \hat{Y}_k = g_{mag}\left(\hat{X}_{k+1|k}, {U^*}_k\right) \\ C_k = \frac{\partial g}{\partial X}\left(\hat{X}_{k|k}, {U^*}_k\right) \\ K_k = P_{k+1|k}\, {C_k}^T \left(C_k\, P_{k+1|k}\, {C_k}^T + Q_\gamma\right)^{-1} \end{cases}$$

EKF Initilization Step:

- suppose we know the initial pose of the Turtlebot in the world at the starting time Xinit.
- Intialize all the matrices will be used in EKF equations.
- we used Eigen3 libarary to able to make all the computations in a matrix form.

EKF Initilization Step code:

```
// Initilization
MatrixXd Q_alpha =  MatrixXd::Zero(3,3); // evolution model error
MatrixXd Q_gamma =  MatrixXd::Zero(3,3); // measurement error
MatrixXd P =  MatrixXd::Zero(3,3);       // state covariance matrix
MatrixXd C =  MatrixXd::Zero(3,3);       // observation jacobian Matrix
MatrixXd D =  MatrixXd::Zero(3,3);       // residual covariance
MatrixXd K =  MatrixXd::Zero(3,3);       // kalman gain

VectorXd X(3);                           // state vector (x, y, theta)
VectorXd Y(3);                           // measurement vector ( d1, d2, d3)
VectorXd Y_hat(3);                       // expected output

geometry_msgs::Point B1, B2, B3;         // beacon coordinates
B1.x=0;  B1.y=0;  B2.x=0;  B2.y=5.20; B3.x=4.40; B3.y=3.20;
```

EKF Predicition Step:

- The second step is to make a prediction about where the Turtlebot currently is based on odometry.
- The prediction is easy to obtain via the /odom topic published by the turtlebot.
- Update the covarance matrix.

EKF Predicition Step code

```
// Predicted_state Xk+1/k "extracted from odom2D"
 X(0)= odom2D.pose.x;
 X(1)= odom2D.pose.y;
 X(2)= odom2D.pose.theta;

// state_trans_uncertainty_noise: Q_alpha "extract covariance matrix from odom2D"
 Q_alpha(0,0)= odom2D.Covariance[0];
 Q_alpha(1,1)= odom2D.Covariance[4];
 Q_alpha(2,2)= odom2D.Covariance[8];

// update covariance matrix
 P=P+Q_alpha;
```

EKF Correction Step:

IF THE MEASUREMENT ARE RELEVANT

- calculate the expected measurement from the predicted state Y_hat.
- calculate the residual by subtracting the expected measurement from the actual one (Y - Y_hat)
- calculate the uncertainty inherent in the residual .
- calculte the kalman gain .
- update the estimated state using the kalman filter .
- update the covariance matrix by calculating the uncertainty of our final estimate state.

EKF Correction Step code

```
if (Y(0)> 0.3 && Y(0)< 6 && Y(1)> 0.3 && Y(1)< 6 && Y(2)> 0.3 && Y(2)< 6 ){
    // Actual measurement  Y
      Y(0)= dist.data[0]/100;    // divide over 100 to convert from cm to m
      Y(1)= dist.data[1]/100;
      Y(2)= dist.data[2]/100;

    // Expected measurement Y_hat
      Y_hat(0)= sqrt( (B1.x-X(0))*(B1.x-X(0)) + (B1.y-X(1))*(B1.y-X(1)) );
      Y_hat(1)= sqrt( (B2.x-X(0))*(B2.x-X(0)) + (B2.y-X(1))*(B2.y-X(1)) );

     Y_hat(2)= sqrt( (B3.x-X(0))*(B3.x-X(0)) + (B3.y-X(1))*(B3.y-X(1)) );
```

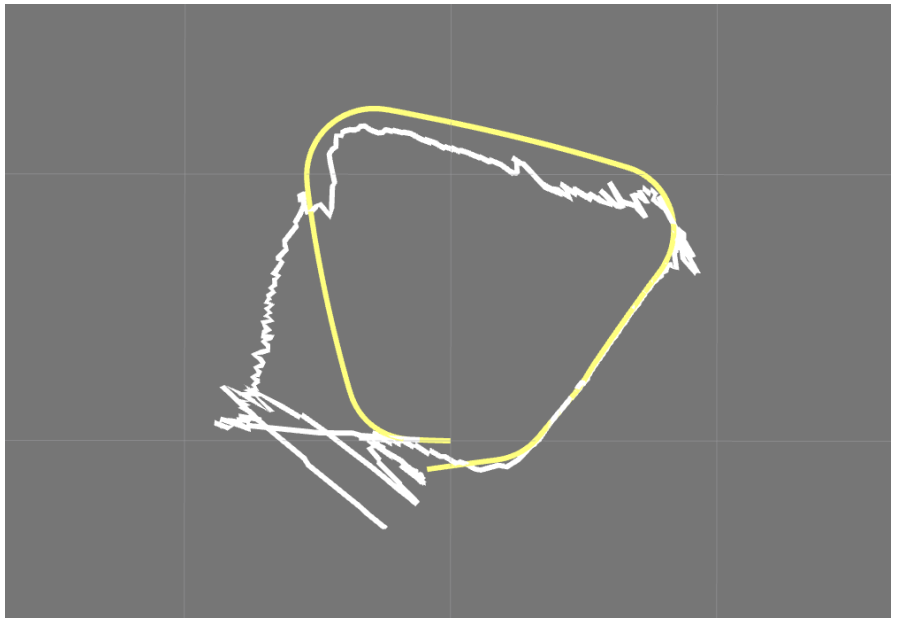
EKF Correction Step code

```
// Measurement jacobian C_matrix
   C(0,0)= 2*(X(0) - B1.x) / Y_hat(0);
   C(0,1)= 2*(X(1) - B1.y) / Y_hat(0);
   C(1,0)= 2*(X(0) - B1.x) / Y_hat(1);
   C(1,1)= 2*(X(1) - B1.y) / Y_hat(1);
   C(2,0)= 2*(X(0) - B1.x) / Y_hat(2);
   C(2,1)= 2*(X(1) - B1.y) / Y_hat(2);
// measurement covariance Q_gamma

   Q_gamma(0,0)= (.0000842*Y(0))+0.0158;
   Q_gamma(1,1)= (.0000842*Y(1))+0.0158;
   Q_gamma(2,2)= (.0000842*Y(2))+0.0158;

 // Residual covariance D is Q_(Y-Y_hat)
   D= ( C*P*C.transpose() + Q_gamma );

 // Correction propotional gain  K
   K= P*C.transpose() * D.reverse();


  // Estimation Step: estimated_state Xk+1/k+1 = Xk+1/k + Kk (Yk - Yk_hat)
   X = X + K*(Y-Y_hat);

  // Estimated_covariance_est: Pk+1/k+1 = ( I - Kk*Ck ) * Pk+1/k
   P= ( I - K*C ) * P;
```

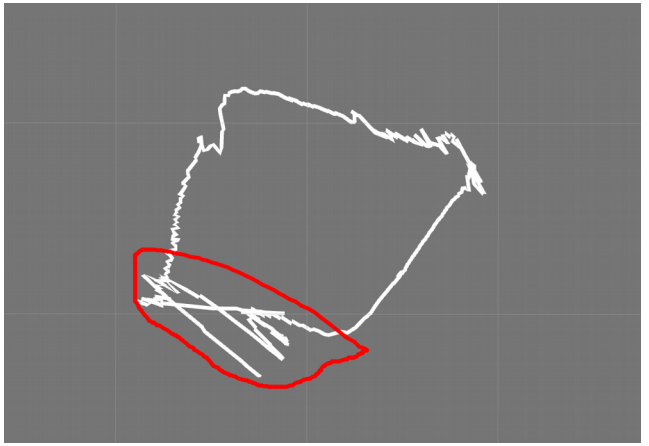
EKF Results

one track has been recorded for the robot

Viedo Link



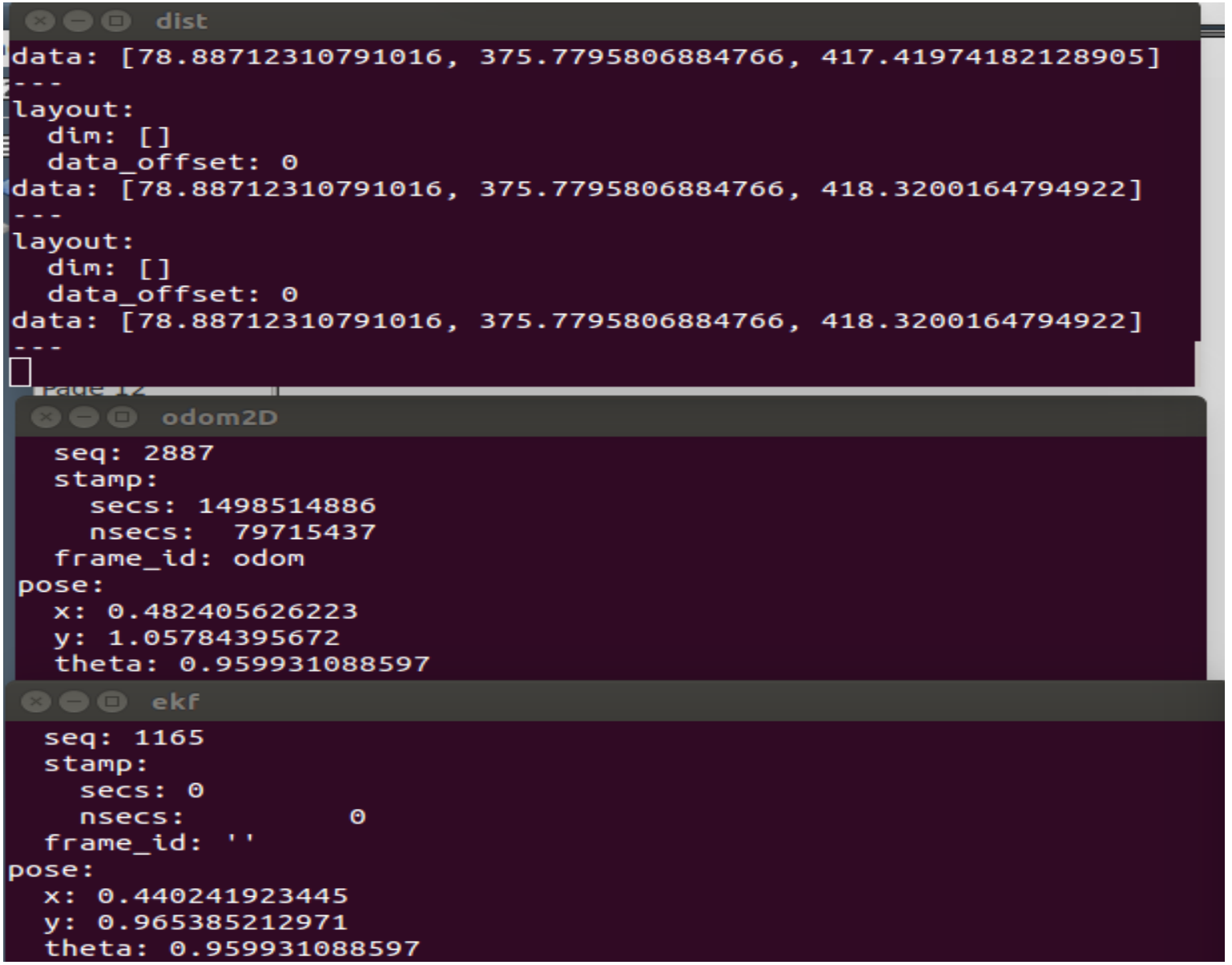
EKF vs Odometry



EKF path

Odom path

EKF Results



1st terminal: beacons distances, 2nd terminal: odometry, 3rd terminal: EKF

Visualization with Rviz

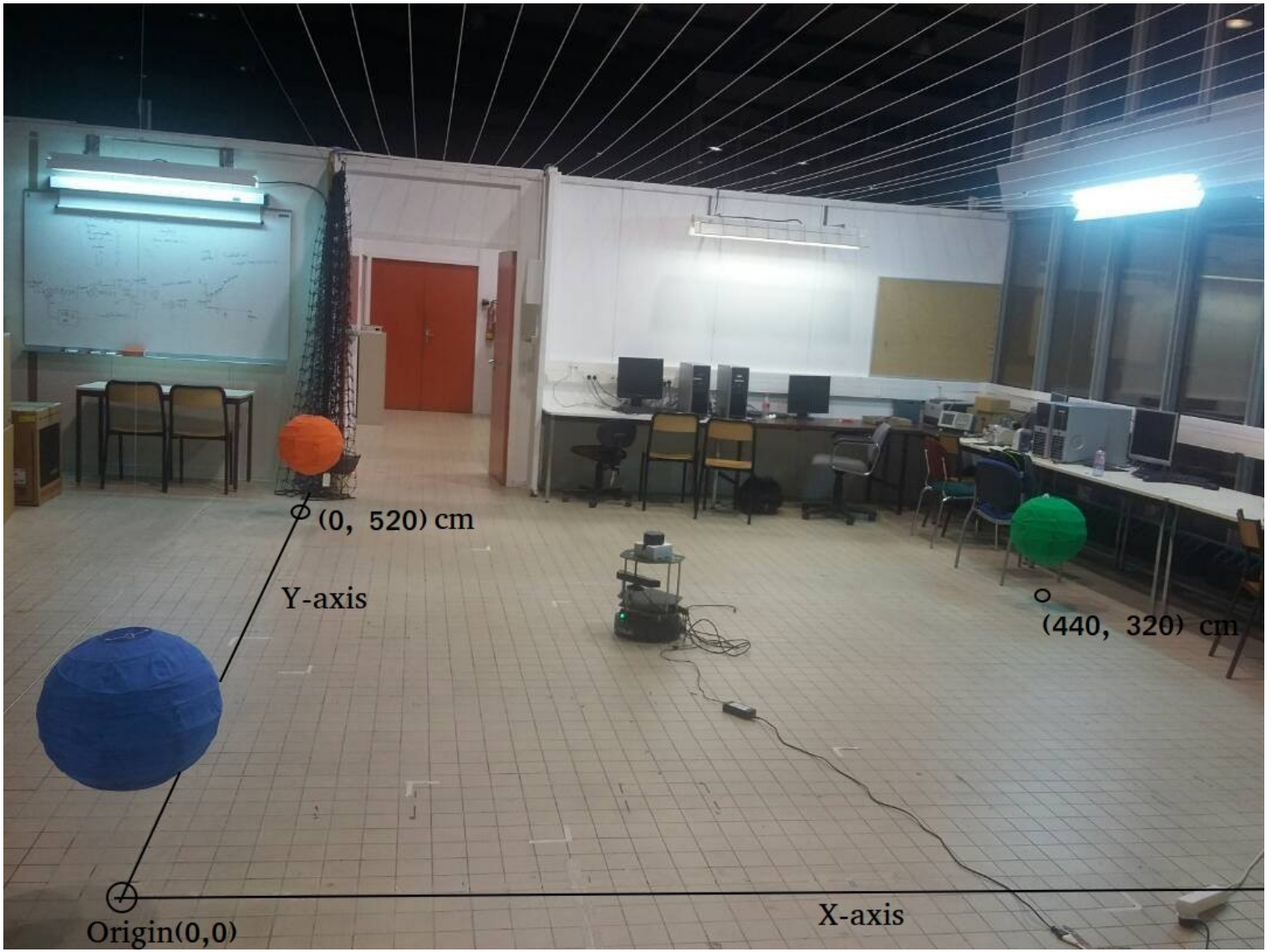
Available 3D model in `turtlebot_rviz_launchers` ROS package

Beacons represented as spheres with their corresponding location

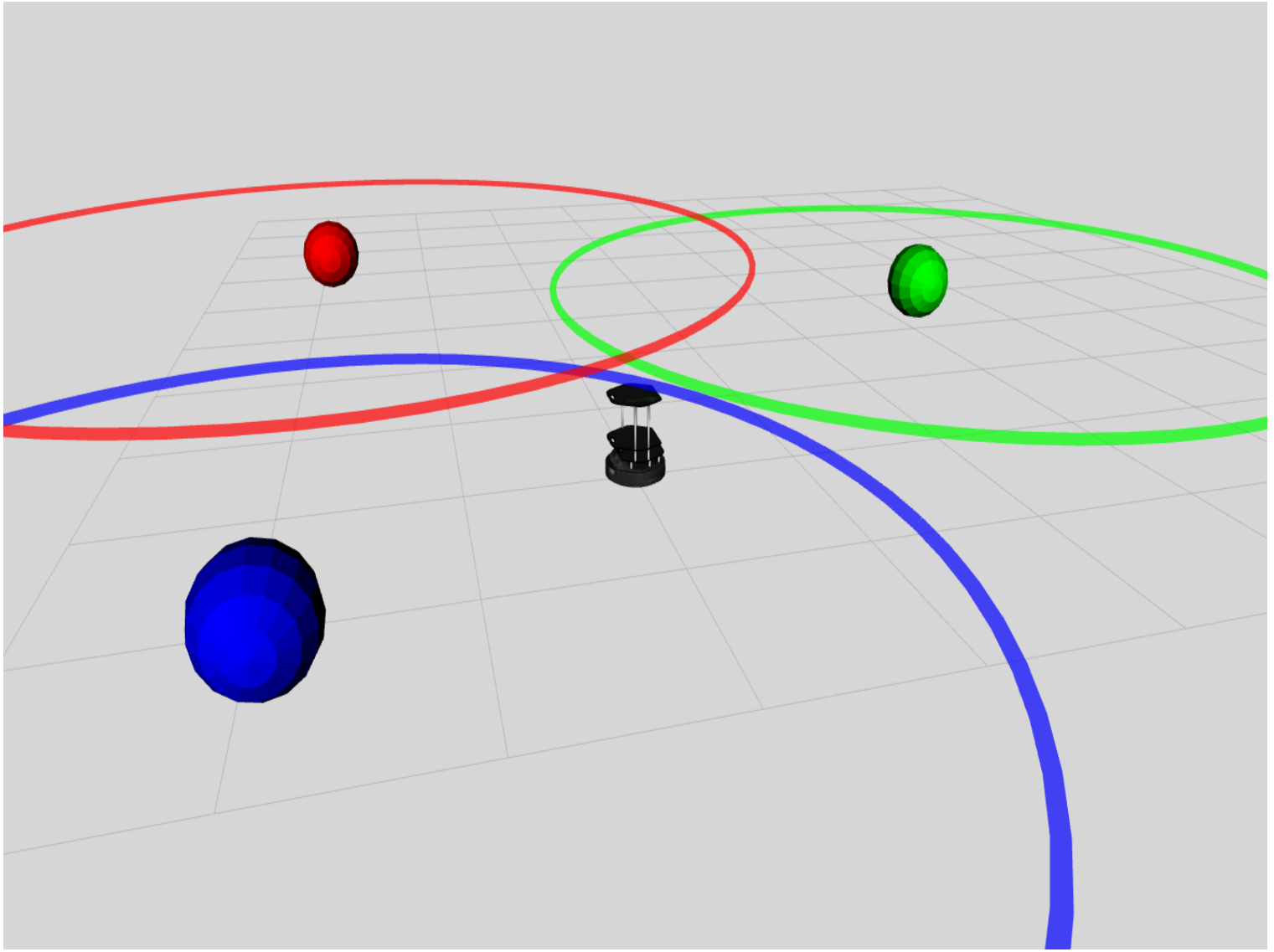
Visualization of distances to beacon as a circle, where the radius represents its distance

Odometry and EKF path visualization

Visualization with Rviz



Visualization with Rviz



Visualization with Rviz

Available 3D model in `turtlebot_rviz_launchers` ROS package

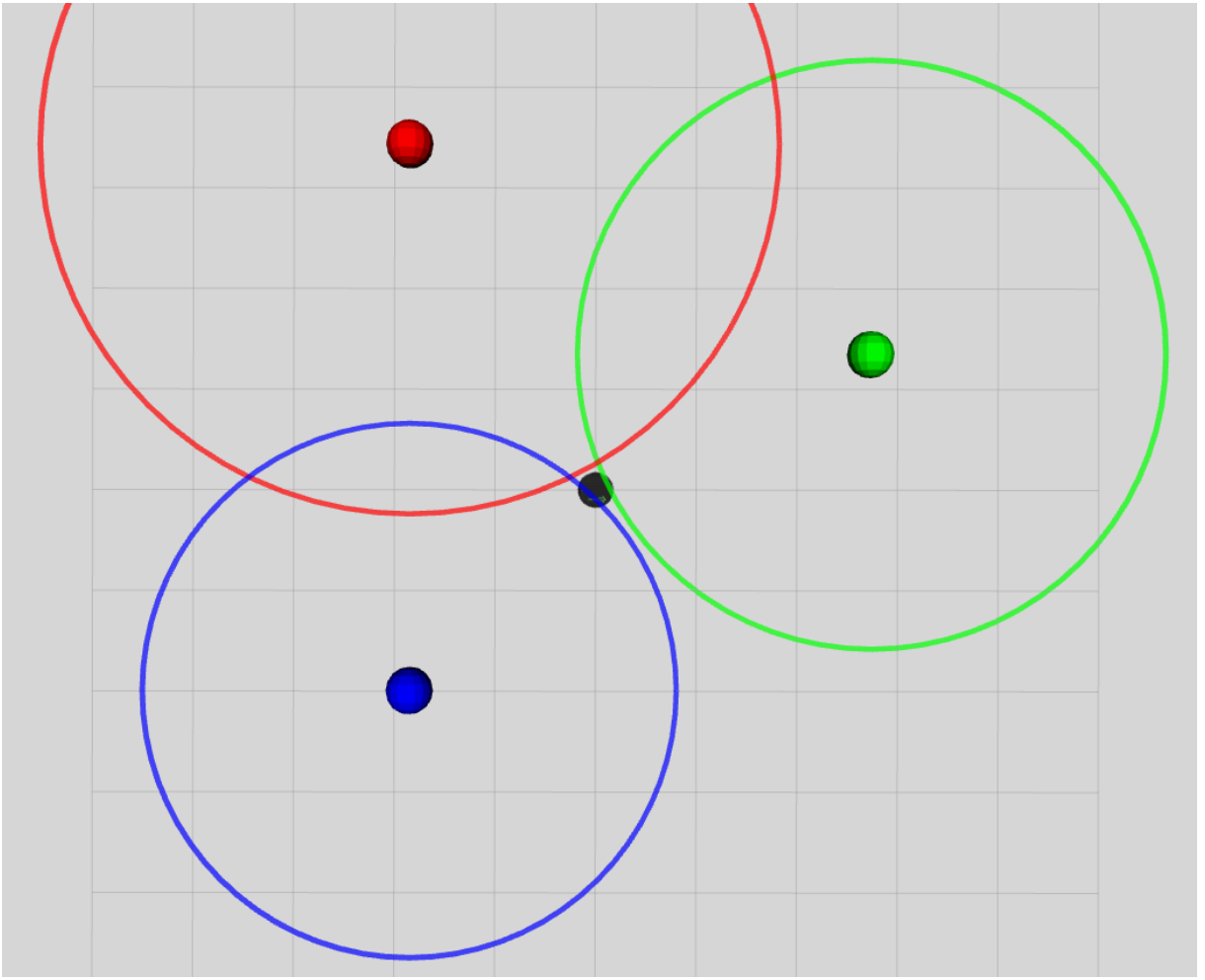
Beacons represented as spheres with their corresponding location

Visualization of distances to beacon as a circle, where the radius represents its distance

Odometry and EKF path visualization

Visualization with Rviz

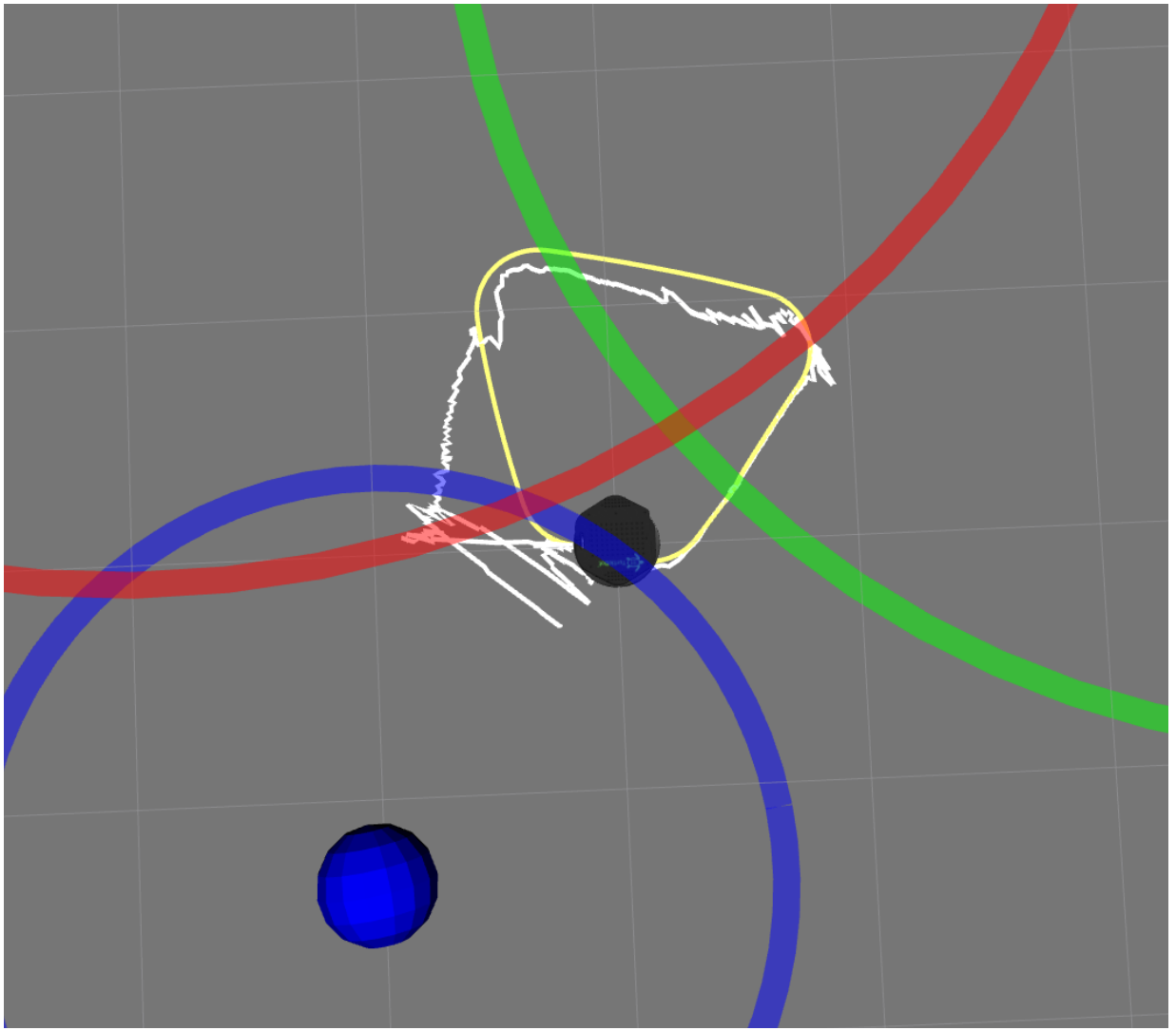
Trilateration results



Visualization with Rviz

Odom (yellow)

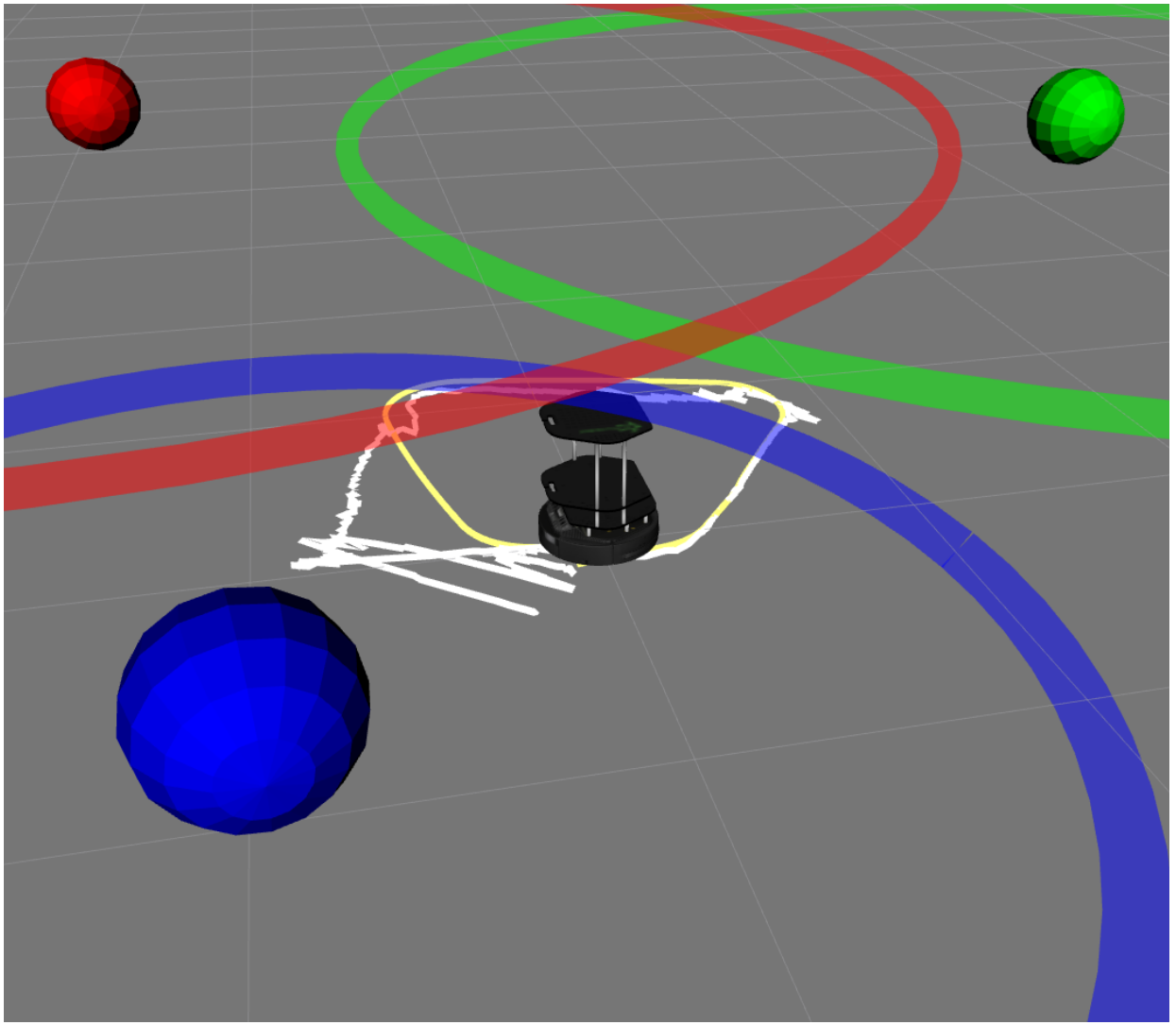
EKF (white)



Visualization with Rviz

Odom (yellow)

EKF (white)



Conclusions

Conclusions

Robust color segmentation using HSV color space.

Conclusions

Robust color segmentation using HSV color space.

Software architechture design plays an important role (modularity and organization).

Conclusions

Robust color segmentation using HSV color space.

Software architechture design plays an important role (modularity and organization).

EKF localization is hard !

Conclusions

Robust color segmentation using HSV color space.

Software architechture design plays an important role (modularity and organization).

EKF localization is hard !

Why ? (in our case)

Conclusions

Robust color segmentation using HSV color space.

Software architechture design plays an important role (modularity and organization).

EKF localization is hard !

Why ? (in our case)

- Slow image processing time.

Conclusions

Robust color segmentation using HSV color space.

Software architechture design plays an important role (modularity and organization).

EKF localization is hard !

Why ? (in our case)

- Slow image processing time.
- High uncertaintity on distance measurements obtained from camera.

Future work

Improving image processing

Advanced EKF

Azimuth angles

Different localization methods