### **−**Table of Contents

* [Introduction](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment6#introduction)
* [Object Detection](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment6#object_detection)
* [Kalman Filter](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment6#kalman_filter)

## Introduction

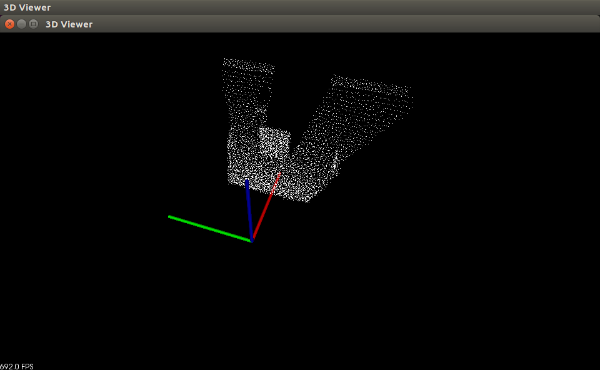
In this assignment we implement Kalman Filter to predict the moving ball with 0.1m radius using pointcloud data coming from kinect sensor.

Formun Üstü

Formun Altı

## Object Detection

To detect the object first I removed the plane data from the pointcloud (using planar segmentation method given in pdf)

Initial PointCloud Data:   


To implement the segmentation I changed pointCloud variable

//pcl::PointCloud<pcl::PointXYZ> \*pointCloud = new pcl::PointCloud<pcl::PointXYZ>();

pcl::PointCloud<pcl::PointXYZ>::Ptr pointCloud(new pcl::PointCloud<pcl::PointXYZ>());

Code for planar segmentation

//plane detection

pcl::ModelCoefficients::Ptr coefficients (new pcl::ModelCoefficients);

pcl::PointIndices::Ptr inliers (new pcl::PointIndices);

// Create the segmentation object

pcl::SACSegmentation<pcl::PointXYZ> seg;

// Optional

seg.setOptimizeCoefficients (true);

// Mandatory

seg.setModelType (pcl::SACMODEL\_PLANE);

seg.setMethodType (pcl::SAC\_RANSAC);

seg.setDistanceThreshold (0.1);

seg.setInputCloud (pointCloud);

seg.segment (\*inliers, \*coefficients);

if (inliers->indices.size () == 0)

{

PCL\_ERROR ("Could not estimate a planar model for the given dataset.");

}

std::cout << "Model coefficients: " << coefficients->values[0] << " "

<< coefficients->values[1] << " "

<< coefficients->values[2] << " "

<< coefficients->values[3] << std::endl;

std::cout << "Model inliers: " << inliers->indices.size () << std::endl;

Then I removed the planar indices:

//plane detection

//remove indices of plane

pcl::ExtractIndices<pcl::PointXYZ> extract;

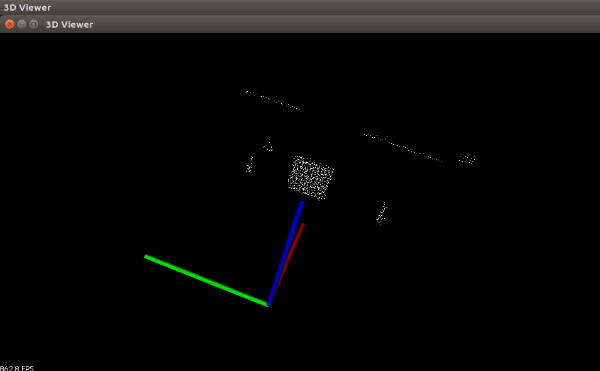
extract.setInputCloud(pointCloud);

extract.setIndices(inliers);

extract.setNegative(true);

extract.filter(\*pointCloud);

std::cout << "size after plane removal = " << pointCloud->points.size () << std::endl;

PointCloud After plane removal:   


Then I limit pointcloud x axis between two walls.

//limit x axis to detect only ball

bool balloccluded = true ;

int cloudsize= 0 ;

for (int i = 0; i < pointCloud->points.size (); i++) {

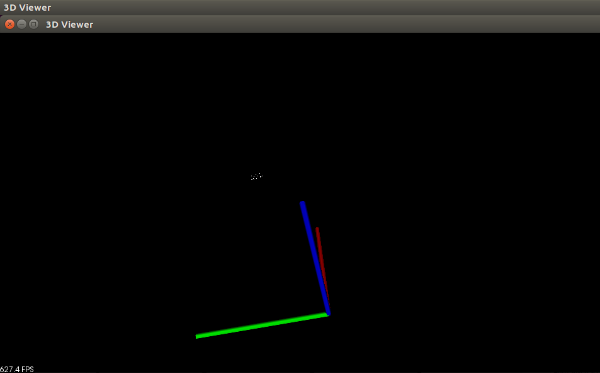
if( pointCloud->points[i].x < 3.8 and pointCloud->points[i].x > 1.6){

cloudsize++ ;

balloccluded = false ;

}

}

PointCloud of Ball:   


Then I avaraged the cloudpoints and transform it to world frame:

for (int i = 0; i < pointCloud->points.size (); i++) {

if( pointCloud->points[i].x < 3.8 and pointCloud->points[i].x > 1.6){

cloud->points[cloudsize].x = pointCloud->points[i].x ;

measured\_x = measured\_x + pointCloud->points[i].x;

cloud->points[cloudsize].y = pointCloud->points[i].y ;

measured\_y = measured\_y + pointCloud->points[i].y ;

cloud->points[cloudsize].z = pointCloud->points[i].z ;

cloudsize++ ;

// print point values for debugging purposes

std::cout << "p[" << i << "].x=" << pointCloud->points[i].x << " .y="

<< pointCloud->points[i].y << " .z="

<< pointCloud->points[i].z << std::endl;

}

}

if(cloudsize > 0){//turn measured data to world frame

measured\_x = (measured\_x / cloudsize)-1.4 ;

measured\_y = (measured\_y / cloudsize)+0.25 ; }

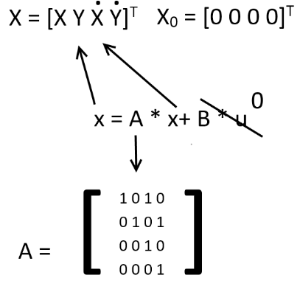
We now have x and y data of the ball location.

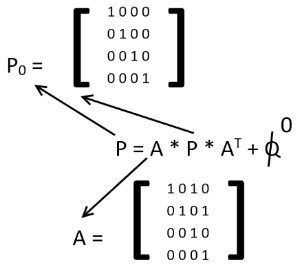
Formun Üstü

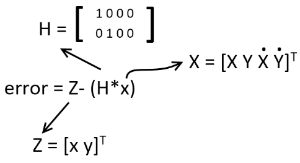
Formun Altı

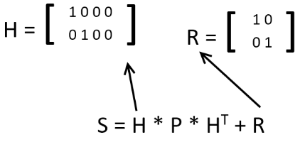
## Kalman Filter

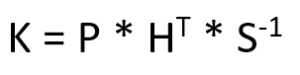
First Define kalman filter matrix.

State prediction:   


State prediction covariance:   


Measurement and prediction error:   


Measurement prediction covariance:   


Kalman Gain calculation:   


Code:   
In the code P is Ball\_cov\_diag   
K is Kalman\_gain   
X is new\_state   
error is measurement\_error   
A is F\_dyn

Initialization :

vector<float> new\_state ; //x y xdot ydot 4x1

vector< vector<float> > Ball\_cov\_diag ; //P 4x4

vector< vector<float> > F\_dyn ; //F 4x4

vector< vector<float> > F\_dyn\_t ; //F\_transpose 4x4

vector< vector<float> > temp ; //temp 4x4

vector< vector<float> > H ; //temp 2x4

vector< vector<float> > H\_t ; //H\_t 4x2

vector<float> R\_diag ; //R 1\_1 2\_2

vector< vector<float> > S ; //S 2x2

vector< vector<float> > Kalman\_gain ; //W 4x2

vector<float> measurement\_error ; //v 2x1

ros::Publisher estimatedPub\_x;

ros::Publisher measuredPub\_x;

ros::Publisher realPub\_x;

ros::Publisher estimatedPub\_y;

ros::Publisher measuredPub\_y;

ros::Publisher realPub\_y;

std\_msgs::Float32 data\_msg;

public:

StateEstimation() {//initalize matrix variables

state = 0 ;

new\_state.push\_back(0); //x

new\_state.push\_back(0);//y

//init 2 column matrix

S.push\_back(new\_state);//1st row

S.push\_back(new\_state);//2nd row

Kalman\_gain.push\_back(new\_state);//1st row

Kalman\_gain.push\_back(new\_state);//2nd row

Kalman\_gain.push\_back(new\_state);//3rd row

Kalman\_gain.push\_back(new\_state);//4th row

H\_t.push\_back(new\_state);H\_t[0][0] = 1;//1st row

H\_t.push\_back(new\_state);H\_t[1][1] = 1;//2nd row

H\_t.push\_back(new\_state);//3rd row

H\_t.push\_back(new\_state);//4th row

new\_state.push\_back(0);//xdot

new\_state.push\_back(0);//ydot

//init 4 column matrix

Ball\_cov\_diag.push\_back(new\_state);//1st row

Ball\_cov\_diag.push\_back(new\_state);//2nd row

Ball\_cov\_diag.push\_back(new\_state);//3rd row

Ball\_cov\_diag.push\_back(new\_state);//4th row

H.push\_back(new\_state);H[0][0] = 1;//1st row

H.push\_back(new\_state);H[1][1] = 1;//2nd row

temp.push\_back(new\_state);//1st row

temp.push\_back(new\_state);//2nd row

temp.push\_back(new\_state);//3rd row

temp.push\_back(new\_state);//4th row

F\_dyn.push\_back(new\_state);F\_dyn[0][0] = 1;F\_dyn[0][2] = 1;//1st row

F\_dyn.push\_back(new\_state);F\_dyn[1][1] = 1;F\_dyn[1][3] = 1;//2nd row

F\_dyn.push\_back(new\_state);F\_dyn[2][2] = 1;//3rd row

F\_dyn.push\_back(new\_state);F\_dyn[3][3] = 1;//4th row

F\_dyn\_t.push\_back(new\_state);F\_dyn\_t[0][0] = 1;//1st row

F\_dyn\_t.push\_back(new\_state);F\_dyn\_t[1][1] = 1;//2nd row

F\_dyn\_t.push\_back(new\_state);F\_dyn\_t[2][0] = 1;F\_dyn\_t[2][2] = 1;//3rd row

F\_dyn\_t.push\_back(new\_state);F\_dyn\_t[3][1] = 1;F\_dyn\_t[3][3] = 1;//4th row

Ball\_cov\_diag[0][0] = 1 ;Ball\_cov\_diag[1][1] = 1 ;Ball\_cov\_diag[2][2] = 1 ;Ball\_cov\_diag[3][3] = 1 ;

R\_diag.push\_back(0.1);//1\_1

R\_diag.push\_back(0.1);//2\_2

measurement\_error.push\_back(0.0);//v1

measurement\_error.push\_back(0.0);//v2

Prediction and Correction :

//time update (prediction)

new\_state[0] = new\_state[0] + new\_state[2] ;

new\_state[1] = new\_state[1] + new\_state[3] ;

new\_state[2] = new\_state[2] ;

new\_state[3] = new\_state[3] ;

std::cout << "Predicted x:" << new\_state[0] << "y:" << new\_state[1] << std::endl;

data\_msg.data = new\_state[0];

estimatedPub\_x.publish(data\_msg);

data\_msg.data = new\_state[1];

estimatedPub\_y.publish(data\_msg);

if(!balloccluded){//we can see the ball

Ball\_cov\_diag = matrix\_product(Ball\_cov\_diag, F\_dyn\_t , 4, 4, 4 ) ;

Ball\_cov\_diag = matrix\_product(F\_dyn, Ball\_cov\_diag , 4, 4, 4 ) ;

//compute kalman gain

Kalman\_gain = matrix\_product(Ball\_cov\_diag, H\_t , 4 , 2, 4 );//used as temp

S = matrix\_product(H, Kalman\_gain , 2 , 2, 4 );

S[0][0] = S[0][0] + R\_diag[0] ;

S[1][1] = S[1][1] + R\_diag[1] ;

Kalman\_gain = matrix\_product(Ball\_cov\_diag ,H\_t , 4 , 2 , 4) ;

//inverse S

float reverse\_cons = (S[0][0] \* S[1][1] - S[1][0] \* S[0][1]) ;

float temp\_S = S[0][0] ;

S[0][0] = S[1][1] / reverse\_cons ;

S[1][1] = temp\_S / reverse\_cons ;

S[0][1] = (-1 \* S[0][1]) / reverse\_cons ;

S[1][0] = (-1 \* S[1][0]) / reverse\_cons ; //S is not S\_inv

Kalman\_gain = matrix\_product(Kalman\_gain , S , 4 , 2 , 2);

//measurement update(corrected)

measurement\_error[0] = measured\_x - new\_state[0] ;

measurement\_error[1] = measured\_y - new\_state[1] ;

new\_state[0] += measurement\_error[0] \* Kalman\_gain[0][0] + measurement\_error[1] \* Kalman\_gain[0][1] ;

new\_state[1] += measurement\_error[0] \* Kalman\_gain[1][0] + measurement\_error[1] \* Kalman\_gain[1][1] ;

new\_state[2] += measurement\_error[0] \* Kalman\_gain[2][0] + measurement\_error[1] \* Kalman\_gain[2][1] ;

new\_state[3] += measurement\_error[0] \* Kalman\_gain[3][0] + measurement\_error[1] \* Kalman\_gain[3][1] ;

temp = matrix\_product(Kalman\_gain , H , 4 , 4 , 2);

//I - temp

temp[0][0] = 1 - temp[0][0] ;temp[0][1] =-temp[0][1];temp[0][2] =-temp[0][2];temp[0][3] =-temp[0][3];

temp[1][1] = 1 - temp[1][1] ;temp[1][0] =-temp[1][0];temp[1][2] =-temp[1][2];temp[1][3] =-temp[1][3];

temp[2][2] = 1 - temp[2][2] ;temp[2][0] =-temp[2][0];temp[2][1] =-temp[2][1];temp[2][3] =-temp[2][3];

temp[3][3] = 1 - temp[3][3] ;temp[3][0] =-temp[3][0];temp[3][1] =-temp[3][1];temp[3][2] =-temp[3][2];

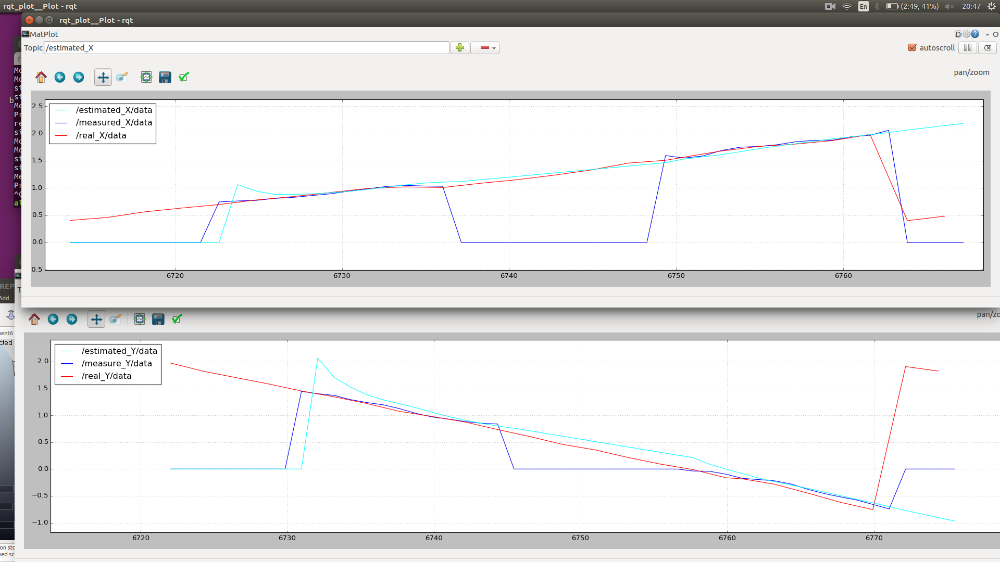
Ball\_cov\_diag = matrix\_product(temp,Ball\_cov\_diag,4,4,4);

std::cout << "Predict corrected x:" << new\_state[0] << "y:" << new\_state[1] << std::endl;

}else{//we cant see the ball

}

}

Errors:   


Video:

<https://youtu.be/mK0uUmk29Wg>