Homework 2: Linear and non linear solver comparison on a ground-plane detection

Environment We use the simple problem of ground-plane detection problem to compare the performance of an Eigen implemented linear regression solver and the Ceres non linear solver.

We use a V-REP robot equipped with a simulated kinect which samples a 3D point cloud. The robot try to detect a plane we model with the equation (1). The computation try to estimate the values of a, b, c.

$$z = ax + by + c \tag{1}$$

We control the robot movements using a joystick and a ROS package **learning_joy** also available in the git repository. The kinect outputs the point cloud coordinates in its own frame so we need to project it to the robot frame before any processing. To do so, we use the ROS TF library desinged to take care of all the changes of frame. Before doing the linear regression, we filter out some 3D points that eliminates the bogus points and the points that are too far from the sensor. The simulated kinect samples periodically a 3D point cloud. For each new point cloud, we update the linear regression problem and solve it again.
We run the computations in two world configurations : a plane world with some obstacles and a slope world (Figure 1).

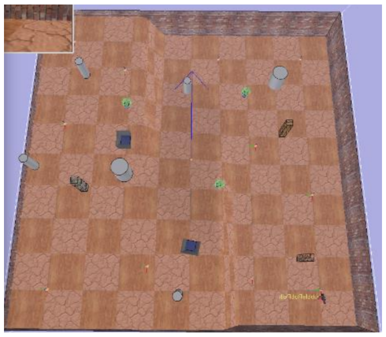


Figure 1: Slope world

Linear model (cf floor_plane_regression) Given the data points $(x_1, y_1, z_1), ..., (x_n, y_n, z_n)$, we estimate the most likely plane contained in this point cloud. To do so, we implement the following regression computation using the Eigen library. We search for parameters $\mathbf{w}^\top = (a, b, c)$ such that the the following error is minimal :

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(y(x_i, \mathbf{w}) - y_i)^2 = \| \begin{pmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} - \begin{pmatrix} z_1 \\ ... \\ z_n \end{pmatrix} \| = \|A \cdot \mathbf{w} - B\|$$

We compute the derivative of the error function with respect to $\mathbf{w}$, derive the value for which is null i.e.

$$\mathbf{w} = (A^\top A)^{-1} A^\top B$$

Given $\mathbf{w}^\top = (a, b, c)$, we display a marker to visualize the detected plan.

Non linear model (cf **floor_plane_ceres**) We use CERES, a non-linear optimiser developed by Google to solve, in particular, bundle adjustment problems. We compute an error functor given the parameters $\mathbf{w}^\top = (a, b, c)$ and a data point (x, y, z) in the class PlaneError.

$$residual = ax + by + c - z$$

We instantiate a cost function used by the ceres solver : we provider the error functor as input and it automatically differentiates it and gives a cost functio interface. We then run the solver.

Computation time We guide the robot to make a round of the worlds and measure the estimation time each time the sensor receives a data point cloud. This triggers the plane equation estimation. We average the samples over the number of estimation made in one round and get the values in Table 1.

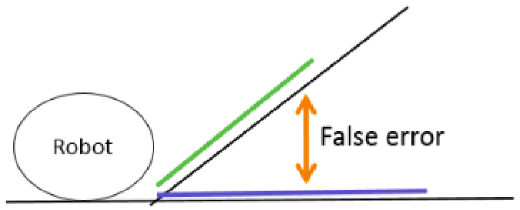
Table 1: Average estimation time of the plane equation from a point cloud (ms)

	Linear regression	Non linear regression
Plane scene	10	175
Slope scene	15	253

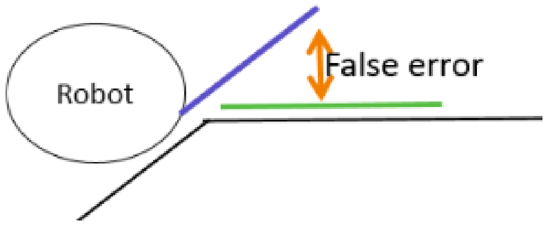
As expected, the non-linear estimation takes more time because of the computations which are more complex than the linear regression one.
We also observe that both techniques take more time in the slope world. This may be because of the point cloud sample : the robot sensor sample a 3D point cloud such which density decreases with the distance to the robot. That is, the nearer to the sensor, the more point samples there are. In a plane world the points are regurlarly distributed on the ground and the 3D point cloud density depends only on the sensor setting. However, in the slope world, the points are distributed on the groud and sometimes on the slope : there are more "near points" that in the plane settings so the point cloud size may increase. This leads to higher data matrix and then more time consuming matrix computation. It is all the more expressive with the Ceres solver which must compute the jacobian of a matrix which size is proportionnal to the point cloud size.

Accuracy We evaluate the accuracy of the ground extraction by computing an error function defined below. We use the following assumptions :

- The sample point cloud is in the robot body frame.
- The sensor range is such that, in the robot body frame, the sensed ground in front of the robot is always a plane which normal vector is $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}_{robotFrame}$.
  Warning : this means that in the slope world, we overestimate the error. In the following configuration (Figure 2), our assumption defines the blue plane as the plane to be detected but in the expermiment, it is the green one that the robot try to detect. The error computation will then value the gap between the green and the blue plane even though it is bigger than the real error. We proceed this way because we do not have an exact method to compute the slope angle. We assume that this additional error given that is occurs only at the beginning and the end of the slope. So this additional error is marginal



(a) At the beginning of the slope



(b) At the end of the slope

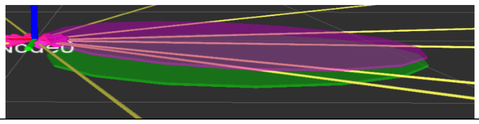
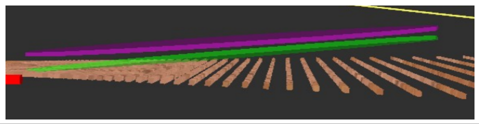
Figure 2: Additional error representation

We use the following error function E where $X_{est} = \begin{pmatrix} a_{est} \\ b_{est} \\ c_{est} \end{pmatrix}$ and $X_{exp} = \begin{pmatrix} a_{exp} \\ b_{exp} \\ c_{exp} \end{pmatrix}$ are the plane equation parameter. In the robot frame, the ground plane has the equation $z = 0$ so $X_{est} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ so:

$$E(N_{est}) = \|N_{est} - N_{exp}\|_2 = \|N_{exp}\|_2$$

We guide the robot to make a round of the worlds (Figure fig:planeWorld and fig:slopeWorld) and compute the estimation error each time the sensor receives a data point cloud which triggers the plane equation estimation. We average the samples over the number of estimation made in one round and get the values in Table 1.

Table 2: Average estimation error of the plane equation from a point cloud

	Linear regression	Non linear regression	Illustration (green: linear, purple: non-linear)
Plane scene	10^{-4}	$3.6 \cdot 10^{-3}$	
Slope scene	$1.42 \cdot 10^{-2}$	$4.9 \cdot 10^{-3}$	

Globally, both techniques detect the plane with reasonnable error.
In the plane world, the non-linear estimation error is a bit higher that the linear estimation one: this can be explained by the fact that a non-linear model is not the relevant one to choose to detect a plane out of a point cloud. Even though the error is still acceptable, the factor 10 between the two errors highlights how important the estiamtion model is.
In the slope world,

Robustness The **linear regression is not robust**: as soon as there i a bogus point, the regression will modify the model to adapt to the point even though it increases the error.

In the non-linear configuration, the CERES solver offers a 'robustify' option that allows us to define a loss function to penalize some data points in the estimation. The **loss function is the 0-1 function and may be relevant when the obstacle is in small part of the vision field of the robot.** It penalizes these points because they induces error in the ground plane estimation. This penalization may be too strict and make the estimation drift : once the obstacle takes a big enough part of the robot vision field, the estimator may think that it is the obstacle plane that it has to measure rather than the plane ground. This is due to the fact that the loss function is bigger when it estimates the ground plane rather than the obstacle while it still tries to estimate the ground plane. However, if **we implement a finer loss function** that would, for example, depend on the altitude of the seen points (only for this example because we want to estimate the ground plane). This loss function would **highly penalize points with** $z > 0$ since they are points on obstacle : this way the robot would not take them into account in its estimation.