OBJECTIFS

Nous souhaitons développer un ADAS AEB (Automatic Emergency Braking) basé LIDAR. Une campagne de mesure a été menée avec un véhicule instrumenté avec un Velodyne VLP16.

Vous êtes en charge de développer un diagramme et des composants RTMaps pour réaliser cette fonction.

Architecture du programme

- **Composant SLIDER :**

  Nous utilisons un composant « slider_filtre » pour gérer la hauteur, la largeur et la profondeur du champ de détection. Notre composant intègre 6 sliders, donc contient 6 sorties.

  Le composant doit agir respectivement sur les 3 axes en positif et en négatif.

  Nous allons également ajouter un second composant « slider_seuil » qui nous permet de gérer le seuil en temps réel, faisant varier la valeur de 1 à 20.

- **Composant POINT_CLOUD_FILTER**

  Ce composant va nous permettre de filtrer l'image du décodeur pour rendre les données accessibles à d'autres composants. Dans notre cas, on va utiliser le nuage de points du décodeur et on appliquera un filtre pour limiter la zone de détection en fonction des paramètres des composants sliders.

  Dans un premier temps, on doit récupérer les données du LIDAR. Ensuite nous allons réaliser une fonction filtre qui nous permet de réduire ce nuage de point à une zone de détection. Nous allons donc devoir récupérer en entrée les données issues du slider puis être capable de les interpréter pour adapter la sensibilité du filtre.

  En sortie, nous voulons renvoyer la zone de détection filtrée.

- **Composant DETECTION**

  Ce composant va nous permettre de détecter le nombre de points dans la zone de détection et de renvoyer 1 si le nombre de point dépasse le seuil. Dans le cas de l'AEB, le nombre de points augment en fonction de la taille ou de la distance de l'obstacle détecté, cela veut dire que si l'obstacle est proche, la valeur renvoyée sera 1 et l'AEB peut s'enclencher.

  Pour répondre à cet objectif, nous allons créer une boucle qui calcule le nombre de points sur chaque image. Nous avons aussi besoin de récupérer la sortie du « slider_seuil » pour la comparer au nombre de points. En sortie, nous voulons une seule valeur scalaire.

Gestion des interfaces

- **Composant SLIDER :**

  Entrée : Action de l'utilisateur

  Sortie : paramètres désirés par l'utilisateur

- **Composant POINT_CLOUD_FILTER**

  Entrée : données du slider_filtre (float32), nuage de points du LIDAR (float64)

```
// Use the macros to declare the inputs
MAPS_BEGIN_INPUTS_DEFINITION(MAPSpoint_cloud_filter)
    MAPS_INPUT("points_in",MAPS::FilterFloat64,MAPS::FifoReader) // Pour récuperer les points du LIDAR
    MAPS_INPUT("slider_x_max_in", MAPS::FilterFloat32, MAPS::FifoReader) // Permet de récupérer les données
    MAPS_INPUT("slider_x_min_in", MAPS::FilterFloat32, MAPS::FifoReader) // d'un slider pour modifier
    MAPS_INPUT("slider_y_max_in", MAPS::FilterFloat32, MAPS::FifoReader) // les paramètres du filtre
    MAPS_INPUT("slider_y_min_in", MAPS::FilterFloat32, MAPS::FifoReader)
    MAPS_INPUT("slider_z_max_in", MAPS::FilterFloat32, MAPS::FifoReader)
    MAPS_INPUT("slider_z_min_in", MAPS::FilterFloat32, MAPS::FifoReader)
```

  Sortie : nuage de points filtré (float64) sous forme de tableau

```
// Use the macros to declare the outputs
MAPS_BEGIN_OUTPUTS_DEFINITION(MAPSpoint_cloud_filter)
    MAPS_OUTPUT("points_out",MAPS::Float64,NULL,NULL,1000000) // Renverra les points traités uniquement
    //MAPS_OUTPUT("Obstacle", MAPS::Integer32, NULL, NULL, 1)
MAPS_END_OUTPUTS_DEFINITION
```

  On prend un nombre suffisamment grand afin d'être sûr de renvoyer tous les points du tableau : 100000.

  Propriétés :

```
// Use the macros to declare the properties
MAPS_BEGIN_PROPERTIES_DEFINITION(MAPSpoint_cloud_filter)
    MAPS_PROPERTY("x_min", -1.0, false, false) // Propriété du filtre initialisé à 1 ou -1
    MAPS_PROPERTY("x_max", 1.0, false, false)
    MAPS_PROPERTY("y_min", -1.0, false, false)
    MAPS_PROPERTY("y_max", 1.0, false, false)
    MAPS_PROPERTY("z_min", -1.0, false, false)
    MAPS_PROPERTY("z_max", 1.0, false, false)
MAPS_END_PROPERTIES_DEFINITION
```

On fixe ces valeurs à -1 et 1 pour pouvoir les multiplier par les données du slider. On rajoute également un chiffre après la virgule pour définir ces valeurs en tant que float au lieu de int.

- **Composant DETECTION :**

  Entrée : Nuage de points filtré issu du composant « point_cloud_filter »

```
// Use the macros to declare the inputs
MAPS_BEGIN_INPUTS_DEFINITION(MAPSDetection)
    // Pour récuperer les points traitées
    MAPS_INPUT("filtre_points_in", MAPS::FilterFloat64, MAPS::FifoReader)
    // Pour récuperer les données du slider qui permettra de modifier la sensibilitée de la détection
    MAPS_INPUT("zone_detection_in", MAPS::FilterFloat32, MAPS::FifoReader)
MAPS_END_INPUTS_DEFINITION
```

  Sortie : valeur scalaire (Integer32) qui prend la valeur 0 ou 1

```
// Use the macros to declare the outputs
MAPS_BEGIN_OUTPUTS_DEFINITION(MAPSDetection)
    // Mettra en sortie 0 ou 1 si détection.
    MAPS_OUTPUT("Obstacle",MAPS::Integer32,NULL,NULL,1)
MAPS_END_OUTPUTS_DEFINITION
```

Implémentation

- **Composant POINT_CLOUD_FILTER**

```
void MAPSpoint_cloud_filter::Core()
{
    // on initialise ce qu'on reçoie en entrée
    MAPSIOElt* ioEltIn = StartReading(Input("points_in"));
    if (ioEltIn == NULL)
        return;
    MAPSIOElt* slider_x_max_in = StartReading(Input("slider_x_max_in"));
    MAPSIOElt* slider_x_min_in = StartReading(Input("slider_x_min_in"));
    MAPSIOElt* slider_y_max_in = StartReading(Input("slider_y_max_in"));
    MAPSIOElt* slider_y_min_in = StartReading(Input("slider_y_min_in"));
    MAPSIOElt* slider_z_max_in = StartReading(Input("slider_z_max_in"));
    MAPSIOElt* slider_z_min_in = StartReading(Input("slider_z_min_in"));

    MAPSIOElt* ioEltOut = StartWriting(Output("points_out"));

    //On initialise les variables
    int j = 0;
    int i;
    double x_min = GetFloatProperty("x_min");
    double x_max = GetFloatProperty("x_max");
    double y_min = GetFloatProperty("y_min");
    double y_max = GetFloatProperty("y_max");
    double z_min = GetFloatProperty("z_min");
    double z_max = GetFloatProperty("z_max");
    double valueIn = ioEltIn->Float64();
    double slider_x_max = slider_x_max_in->Float32();
    double slider_x_min = slider_x_min_in->Float32();
    double slider_y_max = slider_y_max_in->Float32();
    double slider_y_min = slider_y_min_in->Float32();
    double slider_z_max = slider_z_max_in->Float32();
    double slider_z_min = slider_z_min_in->Float32();
```

```
for (i = 0; i < ioEltIn->VectorSize(); i = i + 3) { // on fait une boucle for pour prendre tous les points de coordonnées x, y et z
    double X = ioEltIn->Float64(i); // On initialise la prise de données
    double Y = ioEltIn->Float64(i + 1);
    double Z = ioEltIn->Float64(i + 2);
    if ((X < -1 || X > 1) && (Y < -0.5 || Y > 0.5) && (Z < -0.5 || Z > 0.5)) { // On filtre les points du véhicule
        // on filtre uniquement ce qu'on veut prendre (à savoir ce qu'il y a devant le véhicule mais on peut modifier grace à des sliders :
        if (X >= (x_min * slider_x_min) && X <= (x_max * slider_x_max) && Y >= (y_min * slider_y_min) && Y <= (y_max * slider_y_max) && Z >= (z_min * slider_z_min) && Z <= (z_max * slider_z_max)) {
            ioEltOut->Float64(j) = X; //On récupère les coordonnées souhaitées
            ioEltOut->Float64(j + 1) = Y;
            ioEltOut->Float64(j + 2) = Z;
            j = j + 3; // On compte le nombre de données qu'on aura dans le tableau de sortie pour le VectorSize
        }
    }
    //tests
}
ioEltOut->VectorSize() = j;
ioEltOut->Timestamp() = ioEltIn->Timestamp();
StopWriting(ioEltOut);
}
```

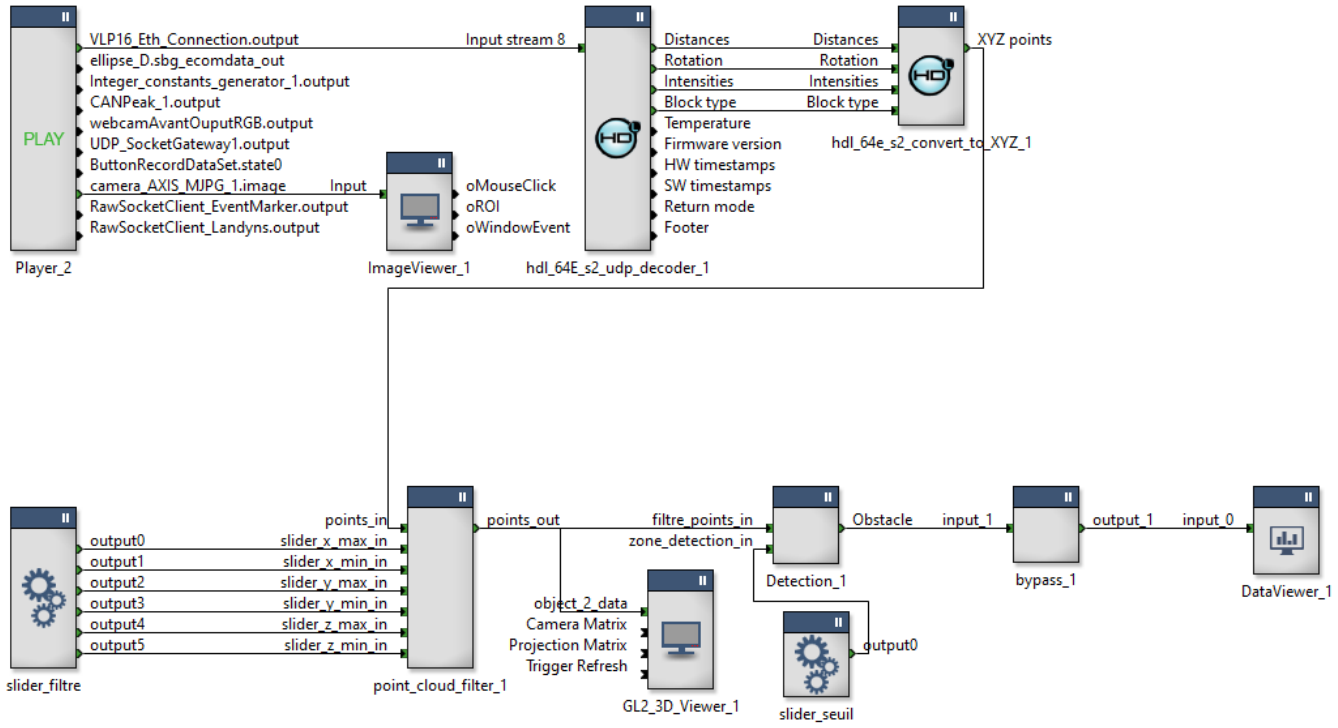
- **Composant DETECTION**

```
void MAPSDetection::Core()
{
    //on initialise nos entrées
    MAPSIOElt* ioEltIn = StartReading(Input("filtre_points_in"));
    if (ioEltIn == NULL)
        return;
    MAPSIOElt* zone_detection_in = StartReading(Input("zone_detection_in"));

    MAPSIOElt* ioEltOut = StartWriting(Output("Obstacle"));

    int obstacle = 0;
    int point_count = 0;
    double zone_detection = zone_detection_in->Float32();
    //On fait une boucle afin de connaitre le nombre de points qu'on visualise après notre filtre.
    for (int i = 0; i < ioEltIn->VectorSize(); i=i+3) {
        // On récupère les donnees x de chaque points
        double k = ioEltIn->Float64(i);
        if (k != 0) {
            // On compte le nombre de point détecté après le filtre
            point_count = point_count + 1;
        }
    }
    // si le nombre de points détectés est supérieur au seuil défini alors il y a un obstacle et on renvoie 1
    if (point_count > zone_detection) {
        obstacle = 1;
    }
    ioEltOut->Integer32() = obstacle;
    ioEltOut->VectorSize() = 1;
    ioEltOut->Timestamp() = ioEltIn->Timestamp();
    StopWriting(ioEltOut);
}
```

Voici donc notre diagramme final avant le test :



Tests et validation

Lors du développement de notre diagramme, nous avons pu intégrer plusieurs composants qui nous permettent d'atteindre l'objectif. Pour que ces composants renvoient une valeur fiable au final (1 si obstacle détecté, 0 sinon), nous avons dû passer par plusieurs phases de test pour paramétrer nos différents composants.

Dans un premier temps, nous avons modifié la zone de détection pour éliminer les zones inutiles et pouvant provoquer des faux-positifs. Pour cela, nous avons tout d'abord éliminé la voiture en appliquant un filtre pour supprimer les points.

Nous avons commencé par des grandes valeurs que nous avons réduites au fur et à mesure de nos tests. Nous avons également modifié la valeur sur l'axe Z en fonction de la zone de détection, notamment lors de passage de pentes ou de dos-d'âne.

Ensuite, nous avons réduit le champ de vision en supprimant les points à droite et à gauche du véhicule, tout en ayant la possibilité d'adapter ces paramètres avec les sliders. Nous avons fait la même chose pour les axes X et Z, tout en gardant une hiérarchie avec le filtre du véhicule :

```
if ((X < -1 || X > 1) && (Y < -0.5 || Y > 0.5) && (Z < -0.5 || Z > 0.5)) { // On filtre les points du véhicule
    // on filtre uniquement ce qu'on veut prendre (à savoir ce qu'il y a devant le véhicule mais on peut modifier grace à des sliders :
    if (X >= (x_min * slider_x_min) && X <= (x_max * slider_x_max) && Y >= (y_min * slider_y_min) && Y <= (y_max * slider_y_max) && Z >= (z_min * slider_z_min) && Z <= (z_max * slider_z_max)) {
        ioEltOut->Float64(j) = X; //On récupère les coordonnées souhaitées
        ioEltOut->Float64(j + 1) = Y;
        ioEltOut->Float64(j + 2) = Z;
        j = j + 3; // On compte le nombre de données qu'on aura dans le tableau de sortie pour le VectorSize
    }
}
```

Les valeurs sont modifiables en fonction du véhicule ou du souhait de l'utilisateur, cependant nous recommandons les paramètres suivants :

X_max : 7

X_min : 1 (ce qui donne -1 grâce à sa propriété, idem pour Y_min et Z_min)

Y_max : 1

Y_min : 1

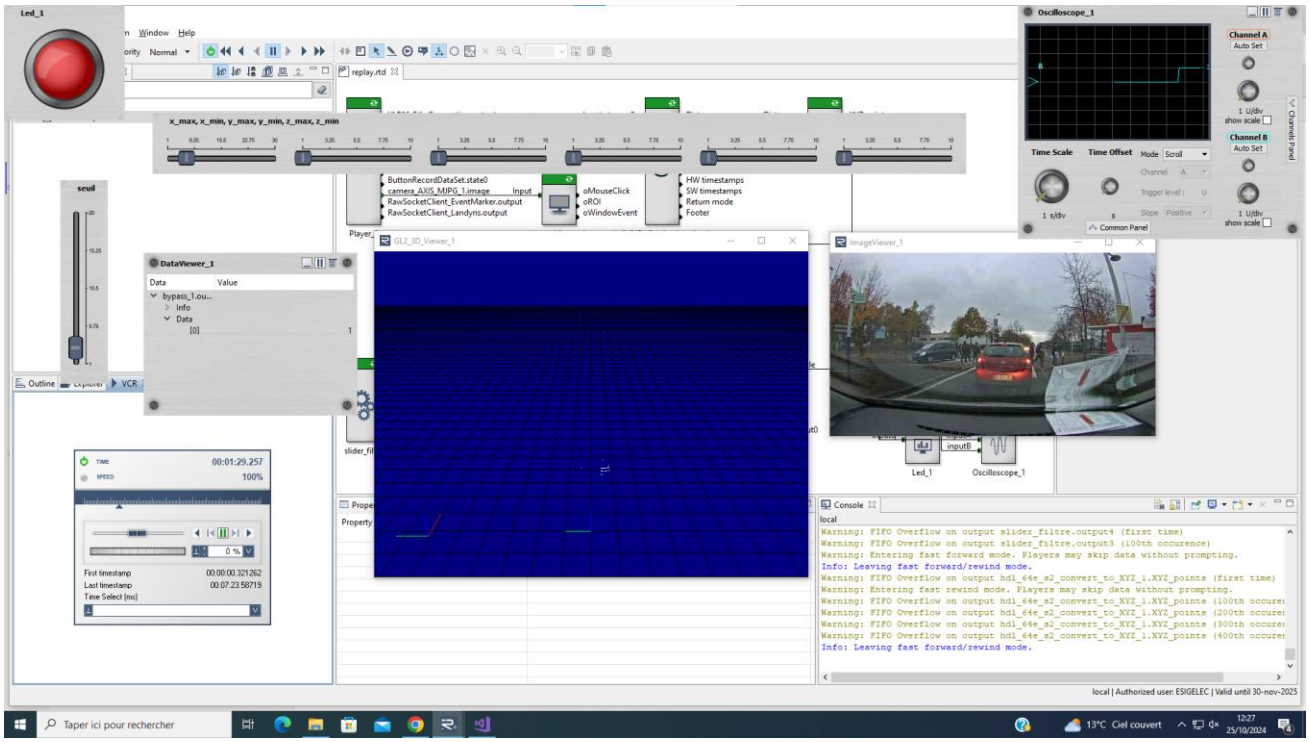
Z_max : 1

Z_min : 1

Seuil : 3

Nous n'avons cependant pas eu le temps de gérer l'analyse de latence.

Voici un exemple de détection :



Conclusion

Pour conclure, notre digramme est fonctionnel et nous permet d'obtenir une bonne détection lorsqu'un obstacle passe devant le véhicule. Nous avons peu de faux-positifs car la zone de détection est faible, ils interviennent principalement dans de virages serrés avec un obstacle dans le coin.

Ce problème est réglable en modifiant la valeur du seuil.

D'autres paramètres peuvent être pris en compte pour l'amélioration du système : données véhicule, intégration avec d'autres radars par exemple.