Научная статья

Original article

УДК 004.93

РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМА РАБОТЫ СИСТЕМЫ КОМПЬЮТЕРНОГО ЗРЕНИЯ ДЛЯ ОПРЕДЕЛЕНИЯ ПОЛОЖЕНИЯ ЦЕНТРА ЗАДНЕЙ ОСИ ВЕДУЩЕГО АВТОМОБИЛЯ ОТНОСИТЕЛЬНО ЦЕНТРА ЗАДНЕЙ ОСИ ВЕДОМОГО АВТОМОБИЛЯ, ДВИЖУЩИХСЯ В КОЛОННЕ

THE DEVELOPMENT AND REALIZATION OF THE ALGORITHM OF THE COMPUTER VISION SYSTEM FOR DETERMINING THE POSITION OF THE CENTER OF THE REAR AXLE OF THE LEADING VEHICLE RELATIVE TO THE CENTER OF THE REAR AXLE OF THE CONVOYFOLLOWING VEHICLE

Малая Анастасия Геннадьевна, магистрант, МГТУ им. Н.Э. Баумана, (105005, Москва, 2-я Бауманская ул., д. 5, стр. 1) 8 (499) 263-639, malayaag@student.bmstu.ru

Калинин Алексей Владимирович, сотрудник кафедры "Специальная робототехника и мехатроника", МГТУ им. Н.Э. Баумана, (105005, Москва, 2-я Бауманская ул., д. 5, стр. 1) 8 (499) 263-639, kalinin_mvtu@mail.ru

Anastasiia G. Malaia, master student, Bauman Moscow State Technical University, BMSTU, (105005, ul. Baumanskaya 2-ya, 5/1, Moscow) 8 (499) 263-639, malayaag@student.bmstu.ru

Alexey V. Kalinin, employee of the "Special robotics and mechatronics" department, Bauman Moscow State Technical University, BMSTU, (105005, ul. Baumanskaya 2-ya, 5/1, Moscow) 8 (499) 263-639, kalinin_mvtu@mail.ru

Аннотация. В данной статье разработан алгоритм работы системы компьютерного зрения для определения положения ведущего автомобиля относительно ведомого автомобиля, движущихся в колонне. Входные данные системы компьютерного зрения представляют собой 3D-облако точек, поступающее с лидара. Вывод системы: расстояния между ведомым и ведущем автомобилями – l_d ; угол между направлениями кузовов ведомого грузовика и расстоянием до центра задней оси ведущего грузовика – α. Разработанный алгоритм реализован на языке программирования C++ с использованием ROS2.

Abstract. In this paper, we developed the algorithm of the computer vision system to determine the position of a leading car relative to the convoy- following car. The input data of the computer vision system is a 3D point cloud coming from a lidar. System output: distances between the driven and leading vehicles – l_d; the angle between the direction of the following truck and the distance to the center of the rear axle of the leading truck – α. The developed algorithm is implemented in the C++ programming language using ROS2.

Ключевые слова: беспилотный грузовой автомобиль, алгоритм работы системы компьютерного зрения, 3D облако точек, движение автомобиля в колонне, C++, ROS2.

Keywords: driverless truck, algorithm of computer vision system, 3D point cloud, convoy of driverless trucks, C++, ROS2.

Технология автономного движения грузового автомобиля, движущегося в колонне за направляющим автомобилем, управляемым человеком (рисунок 1) устраняет недостатки пилотируемых грузоперевозок, такие как: необходимость остановок на сон, душ и приемы пищи; усталость водителя; нападение на водителей грузовиков с целью ограбления.



Рисунок 1 – Метод выстраивания грузовой автомобилей колонной

Для управления ведомым грузовиком, движущимся в колонне за направляющим автомобилем, управляемым человеком, используется концепция алгоритма управления, основанная на Pure Pursuit Controller. [1]

В разработанном алгоритме центр задней оси ведомого автомобиля используется в качестве контрольной точки на транспортном средстве (обозначена зеленой точкой ведомого грузовика на рисунке 2).

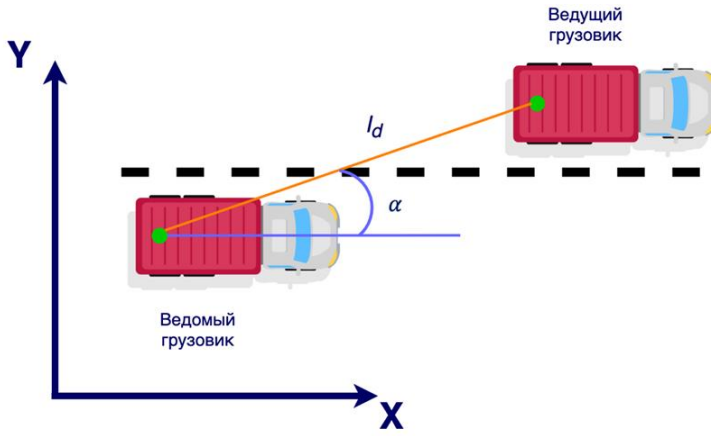


Рисунок 2 – Концепция алгоритма управления ведомым грузовиком

Расстояние между центром задней оси ведомого автомобиля и целевой точкой ведущего грузовика обозначается как l_d.

Цель разработанного алгоритма – поворот ведомого автомобиля на правильный угол и движение к целевой точке ведущего грузовика. Формула для вычисления угол поворота [1]:

$$\delta = \tan^{-1}\left(\frac{2\,L\,\sin\alpha}{l_d}\right)$$

Данное расстояние l_d, а также угол между направлением кузова ведомого грузовика и расстоянием до контрольной точки ведущего грузовика – α определяется с помощью датчика, установленного на ведомом грузовике.

Таким образом, основными целями системы компьютерного зрения слежения являются:

1. определение расстояния между центром задней оси ведомого автомобиля и центром задней оси ведущего грузовика — l_d;
2. определение угла между направлением кузова ведомого грузовика и расстоянием до центра задней оси ведущего грузовика – α;

Подготовка входных данных для последующей обработки

Данные, поступающие с лидара, в виде 3D-облака точек представляют собой сырые данные, которые включают в себя точки всех объектов, находящийся в диапазоне видимости лидара. Работа с таким большим количеством точек замедляет процесс работы всей системы, таким образом, для последующей работы с необходимо уменьшить выборку и отделить полезные данные (положении ведомого автомобиля) от посторонних объектов.

Составим структурную схему алгоритма работы системы компьютерного зрения для определения положения центра задней оси ведущего автомобиля относительно задней оси ведомого автомобиля (рисунок 3).



Рисунок 3 – Алгоритм работы системы компьютерного зрения

Сужение области видимости

Для проведения опытов выбрана модель лидара Velodyne Ultra Puck VLP-32 [2], которая имеет 32 сканируемых луча, максимальный диапазон видимости 200 м и угол обзора 360° x 40°, угловое разрешение 0.1°, частота обновления 10 Гц, таким образом в секунду собирается $360/0.1 \times 10 \times 32 = 1.2 \times 10^6$точек. Так как основная задача ведомого автомобиля состоит в слежении за положением впереди идущего автомобиля, следовательно, необходимо выбрать интересующую область и удалить не интересующие точки, находящиеся за ее пределами.

Для реализации алгоритма на опытном прототипе ограничим интересующую область углом 50°, центрированную относительно центра ведомой машины, и диапазоном видимости 5.0 м. Таким образом, при ограничении угла до 50° количество точек сократиться до $50/0.1 \times 10 \times 32 = 0.16 \times 10^6$ точек, также количество точек сократиться в зависимости от объектов попавших или нет в диапазон видимости. Реализация метода ограничения угла области видимости на языке C++ представлена в листинге 1.

Листинг 1 – Реализация метода ограничения угла области видимости

```
pcl::PCLPointCloud2 PointcloudPipeline::pointcloud_angle_cropped
(pcl::shared_ptr<pcl::PCLPointCloud2> input_cloud){
   std::shared_ptr<pcl::PointCloud<pcl::PointXYZI>> input_cloud_XYZ_ptr =
std::make_shared<pcl::PointCloud<pcl::PointXYZI>>();
   std::shared_ptr<pcl::PCLPointCloud2> final_cloud =
std::make_shared<pcl::PCLPointCloud2>();
   pcl::fromPCLPointCloud2(*input_cloud, *input_cloud_XYZ_ptr);
   unsigned angle_definition = input_cloud_XYZ_ptr->height / 360;
   unsigned angle_part = angle_definition * (view_angle / 2);
   auto begin_iter = next(input_cloud_XYZ_ptr->begin(), 32*angle_part);
   auto end_iter = prev(input_cloud_XYZ_ptr->end(), 32*angle_part);
   input_cloud_XYZ_ptr->erase(begin_iter, end_iter);
   pcl::toPCLPointCloud2(*input_cloud_XYZ_ptr, *final_cloud);
   return *final_cloud;
}
```

Результат работы метода `pointcloud_angle_cropped`, осуществляющего алгоритм ограничения угла области видимости, представлен на рисунке 5:

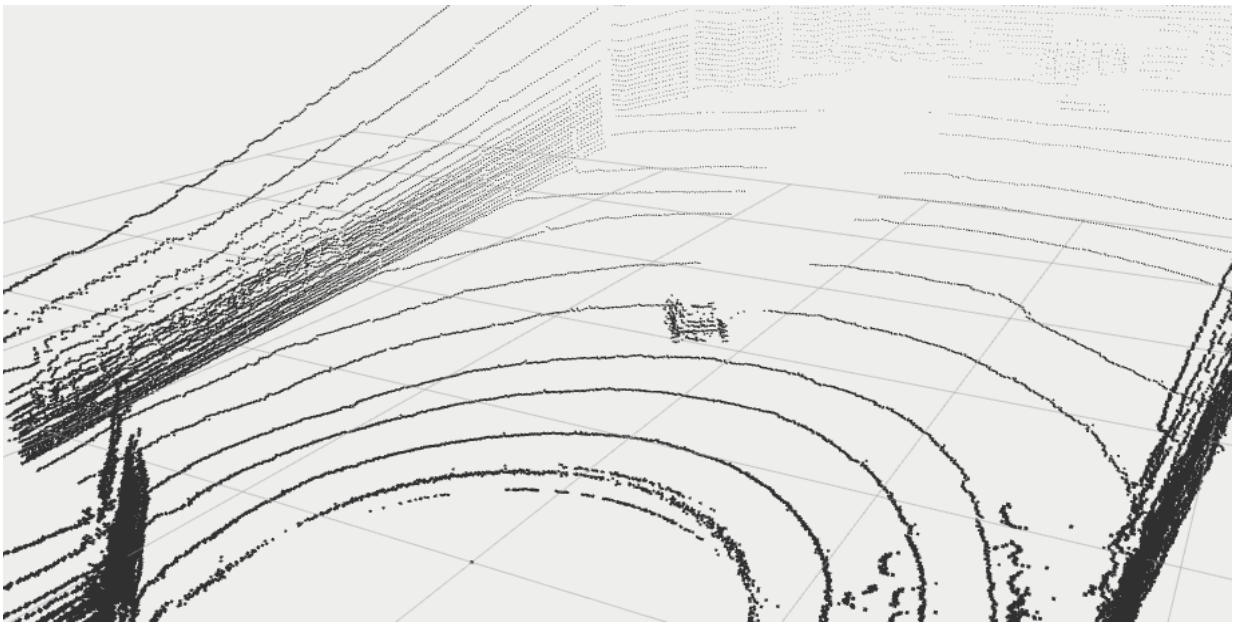


Рисунок 4 – Входное облако точек с лидара Velodyne Ultra Puck VLP-32

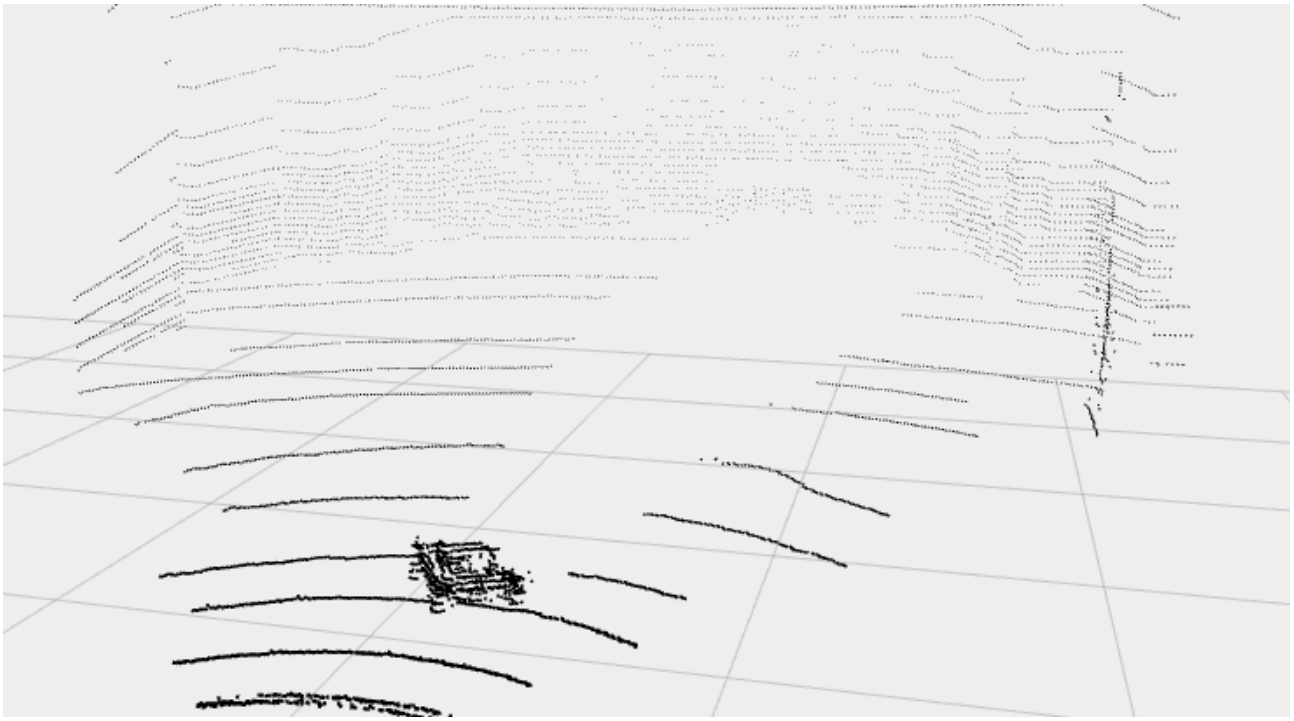


Рисунок 5 – Выходное облако точек метода pointcloud_angle_cropped

Реализация метода ограничения диапазона видимости на языке C++ представлена в листинге 2.

Листинг 2 – Реализация метода ограничения диапазона видимости

```
std::shared_ptr<pcl::PCLPointCloud2>
PointcloudPipeline::pointcloud_distance_cropped
(std::shared_ptr<pcl::PCLPointCloud2> cloud){
   std::shared_ptr<pcl::PCLPointCloud2> final_cloud =
std::make_shared<pcl::PCLPointCloud2>();
   pcl::PassThrough<pcl::PCLPointCloud2> passthrough;
   passthrough.setInputCloud(cloud);
   passthrough.setFilterFieldName("x");
   passthrough.setFilterLimits(1.0, view_distance);
passthrough.filter(*final_cloud);
   return final_cloud;
}
```

Результат работы метода `pointcloud_distance_cropped` , осуществляющего алгоритм ограничения диапазона видимости, представлен на рисунке 6:

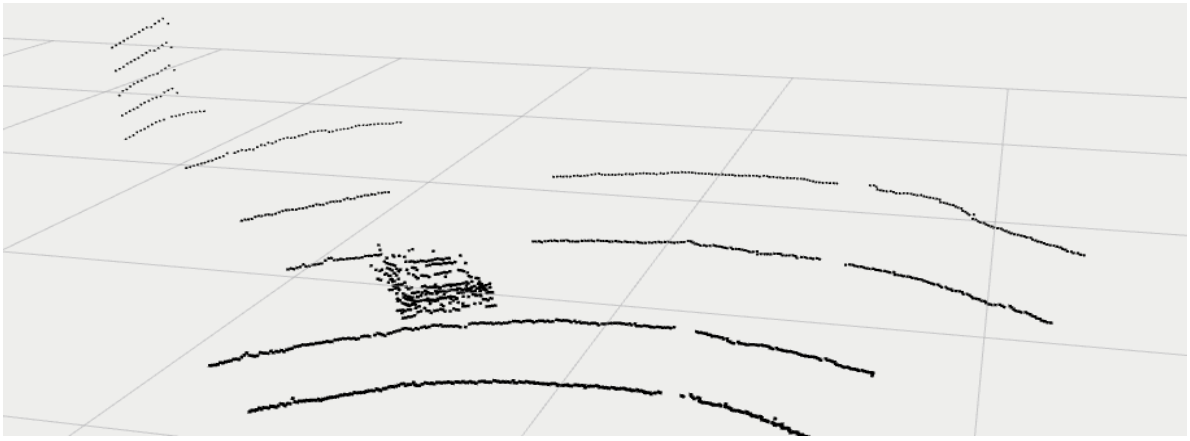
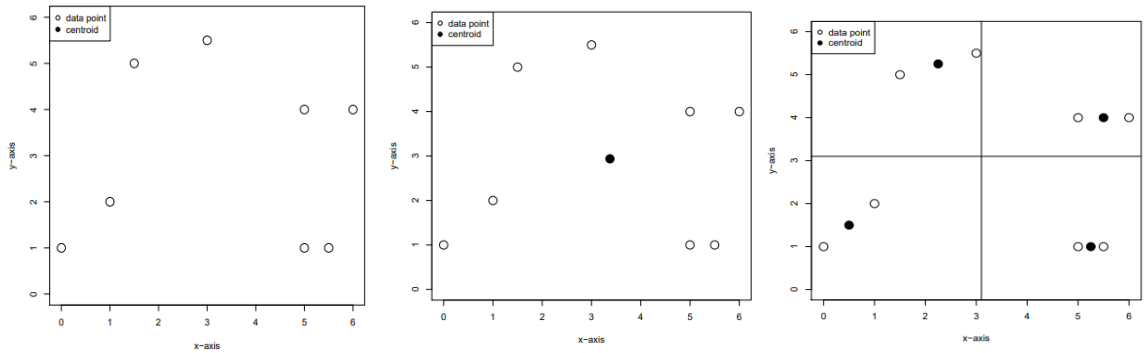


Рисунок 6 – Выходное облако точек метода pointcloud_distance_cropped

Вокселизация

Когда не требуется слишком высокое разрешение, которое может потребовать больше вычислительных ресурсов, необходимо проредить облако точек. В таком случае можно применить вокселизацию.

Основной принцип вокселизации заключается в создании трехмерной сетки на основе входных данных облака точек. Трехмерная сетка представляет собой набор трехмерный пикселей, которые в трехмерном пространстве называются вокселями. Для аппроксимации всех точек в сетке в вокселе вычисляется центр тяжести всех точек входного облака точек, попавших в воксель (рисунок 7). На данном рисунке черными точками обозначены центры тяжести каждого вокселя, на (б) и (в) показаны примеры вокселизации при различных размерах воксельной сетки.



а) (б) (в)

Рисунок 7 – Облако точек до (а) и после вокселизации (б), (в)

После обработки сетки получаются соответствующие входным данным облака вокселей.

Реализация метода вокселизации на языке C++ представлена в листинге 3.

Листинг 3 – Реализация метода вокселизации

```
void PointcloudPipeline::voxel_gridding(pcl::PCLPointCloud2::Ptr
input_cloud) {
   pcl::PCLPointCloud2::Ptr gridded_cloud (new pcl::PCLPointCloud2 ());
   pcl::VoxelGrid<pcl::PCLPointCloud2> vox;
   vox.setInputCloud (input_cloud);
   vox.setLeafSize (leaf_size, leaf_size, leaf_size);
   vox.filter (*gridded_cloud);
   *input_cloud = *gridded_cloud;
}
```

Результат работы метода `voxel_gridding`, осуществляющего алгоритм вокселизации, представлен на рисунке 8:

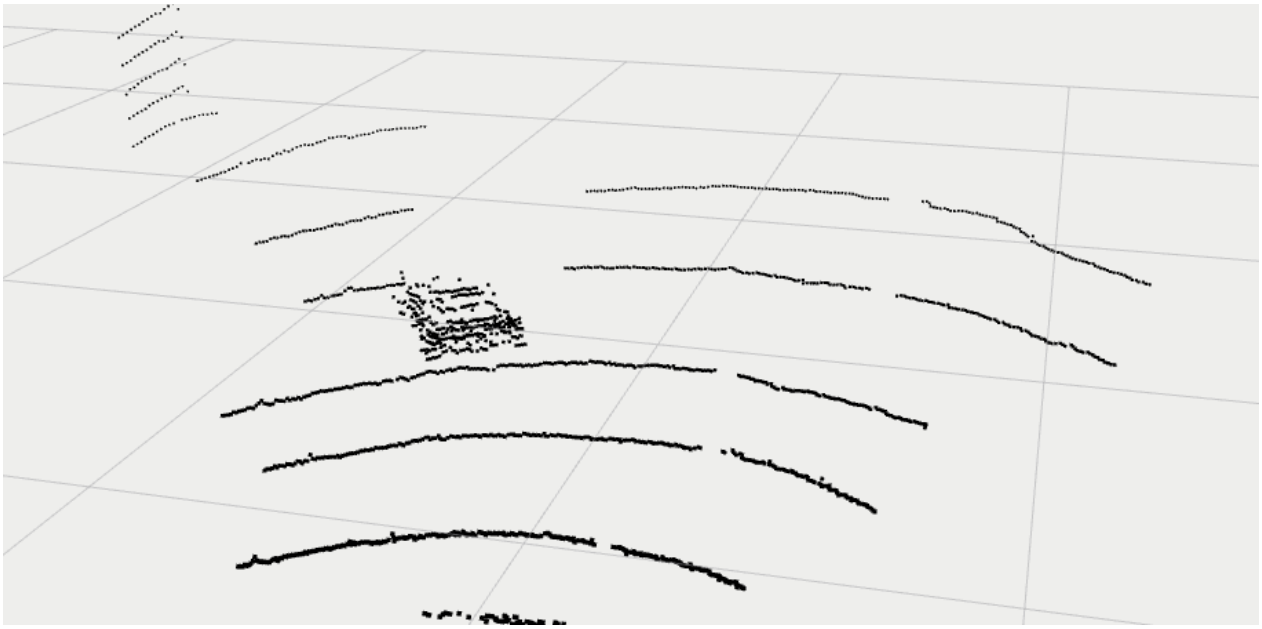


Рисунок 8 – Выходное облако точек метода voxel_gridding

Фильтрация плоскости земли

Для фильтрации плоскости земли необходим алгоритм, который будет определять ровную поверхность — плоскость. В качестве такого алгоритма был выбран алгоритм RANSAC, преимуществом которого является способность дать надежную оценку параметров модели, даже если в исходном наборе данных присутствует значительное количество выбросов.

Алгоритм RANSAC в основном включает в себя выполнение двух итеративно повторяющихся шагов в облаке точек: создание гипотезы и ее проверка.

Формы гипотез генерируются путем случайного выбора минимального подмножества из *k* точек и оценки соответствующих параметров модели формы. Минимальное подмножество содержит наименьшее количество точек, необходимых для однозначной оценки модели (для плоскости – 3). Затем оставшиеся точки в облаке точек тестируются с полученными фигурами-кандидатами, чтобы определить, сколько точек хорошо аппроксимируется моделью. После определенного количества итераций извлекается фигура с наибольшим процентом вкраплений, и алгоритм продолжает обрабатывать оставшиеся данные.

Реализация метода фильтрации плоскости земли на языке C++ представлена в листинге 4.

Листинг 4 – Реализация метода фильтрации плоскости земли

```
void
PointcloudPipeline::cloud_plane_filtration(pcl::PointCloud<pcl::PointXYZI>
::Ptr input_cloud, pcl::PointCloud<pcl::PointXYZI>::Ptr output_cloud,
double distance_threshold)
{
   pcl::PointIndices::Ptr inliers (new pcl::PointIndices);
   pcl::ModelCoefficients::Ptr coefficients (new pcl::ModelCoefficients);
   pcl::SACSegmentation<pcl::PointXYZI> seg;
   seg.setModelType (pcl::SACMODEL_PLANE);
   seg.setMethodType (pcl::SAC_RANSAC);
   seg.setDistanceThreshold(distance_threshold);
   seg.setInputCloud(input_cloud);
   seg.segment (*inliers, *coefficients);
   pcl::ExtractIndices<pcl::PointXYZI> extract;
   extract.setInputCloud(input_cloud);
   extract.setIndices(inliers);
   extract.setNegative(true);
   extract.filter(*output_cloud);
}
```

Результат работы метода `cloud_plane_filtration`, осуществляющего алгоритм фильтрации плоскости земли, представлен на рисунке 9:

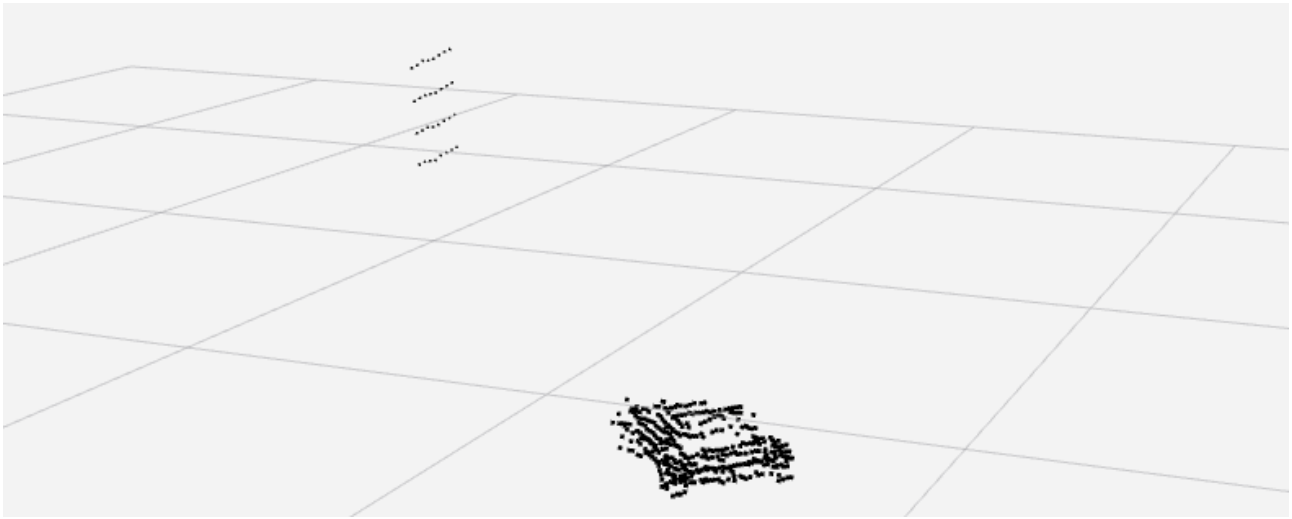


Рисунок 9 – Выходное облако точек метода cloud_plane_filtration

Кластеризация

После выполненных этапов: вокселизации, сужения области видимости и фильтрации плоскости земли; могут остаться точки, не отсеянные не по одному из признаков выбранных методов (например, яма на дороге). В таком случае необходимо отделить набор точек, принадлежащий ведомой машины от ненужных точек (выбросов), принадлежащих другим объектам окружающей среде. Для таких задач используют методы кластеризации – разбиение элементов некоторого множества на группы на основе их схожести. Задача кластеризации состоит в разбиении объектов из X на несколько подмножеств (кластеров), в которых объекты более схожи между собой, чем с объектами из других кластеров [3].

Структура данных K-D Tree

Для алгоритма поиска близких точек к заданной точке эффективно и удобно использовать свойства K-мерного дерева (K-D Tree), так как оно предпочтительнее из-за его характеристик скорости и производительности для обработки больших объемов данных, а также характерного разделения данных.

K-D Tree — это структура данных, которая делит k-мерное пространство, где каждый узел является многомерной координатой. Каждый узел дерева

представляет собой гиперплоскость, которая перпендикулярна координатной оси текущего измерения деления и делит пространство на две части в этом измерении. Одна часть находится в левом поддереве, а другая - в правом поддереве. Если размерность деления текущего узла равна n, координаты всех точек в левом поддереве в измерении p меньше текущего значения, а координаты всех точек в правом поддереве в измерении n больше или равны текущему значению.

Таким образом, алгоритм построения K-D Tree включает в себя два важных шага:

1. Деление происходит на медианное значение выбранного измерения, маленький - левый узел, а большой - правый.
2. При выборе измерений используется метод максимальной дисперсии, то есть каждый раз, когда выбирается измерение для деления, выбирается измерение с наибольшей дисперсией.

Так как облако точек находится в 3-х мерном пространстве, то может быть построено 3-х мерное дерево (3-D Tree), которое представлено на рисунке 10.

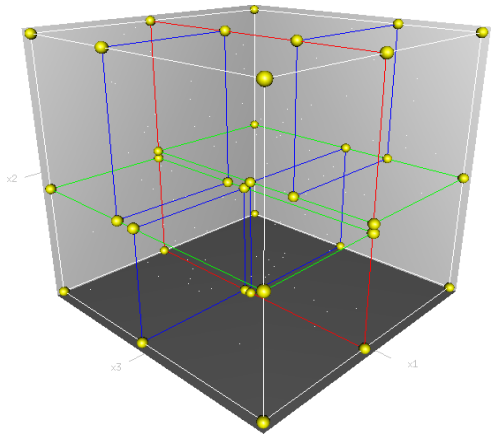


Рисунок 10 – Пример 3-D Tree

Извлечение евклидова кластера

После структурирования облака точек в виде дерева, перейдем к кластеризации. Для кластеризации выберем метод, основанный на пространственной декомпозиции, которые находят подразделения и границы, позволяющие группировать данные вместе на основе заданной меры

«близости». Воспользуемся наиболее естественную функцию расстояния, возникающую в геометрии, — евклидово расстояние, которое представляет собой наименьшее расстояние между двумя точкам евклидового расстояния:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{k=1}^{n}(p_k - q_k)^2}$$

где: $p=(p_1,\ldots,p_n)$, $q = (q_1,\ldots,q_n)$— точки с координатами в n-мерном пространстве.

Предполагая, что мы используем структуру K-D Tree для поиска ближайших соседей каждой точки алгоритмические шаги будут следующими [4]:

1. настроить пустой список кластеров C и очередь точек, которые необходимо проверить Q;
2. затем для каждой точки выполните следующие действия $p_i \in P$ (входной набор данных облака точек, представленный в виде K-D Tree):
   - добавить p_i в текущую очередь Q;
   - для каждой точки $p_i \in Q$ выполните:
     - поиск множества P_i^k точек-соседей p_i на сфере с радиусом $r < d_{th}$(максимальный установленный порог расстояния);
     - для каждого соседа $p_i^k \in P_i^k$ проверить , не была ли уже обработана точка, и если нет, добавить ее в Q;
   - когда список всех точек Q был обработан, добавить Q в список кластеров C и сбросить Q в пустой список;
3. алгоритм завершается, когда все точки $p_i \in P$обработаны и теперь являются частью списка кластеров точек C.

Реализация метода кластеризации на языке C++ представлена в листинге 5.

Листинг 5 – Реализация метода кластеризации

```
std::shared_ptr<pcl::PointCloud<pcl::PointXYZI>>
PointcloudPipeline::clustering(pcl::PointCloud<pcl::PointXYZI>::Ptr
input_cloud) {
   pcl::search::KdTree<pcl::PointXYZI>::Ptr tree (new
pcl::search::KdTree<pcl::PointXYZI>);
   std::shared_ptr<pcl::PointCloud<pcl::PointXYZI>> final_cloud =
std::make_shared<pcl::PointCloud<pcl::PointXYZI>>();
```

```
    std::vector<pcl::PointIndices> cluster_indices;
    pcl::EuclideanClusterExtraction<pcl::PointXYZI> ec;
    tree->setInputCloud (input_cloud);
    ec.setClusterTolerance (clusterTolerance);
    ec.setSearchMethod (tree);
    ec.setInputCloud (input_cloud);
    ec.extract (cluster_indices);
    std::vector<pcl::PointIndices>::const_iterator i =
cluster_indices.begin ();
```

Результат работы метода `clustering`, осуществляющего алгоритм кластеризации, представлен на рисунке 11:

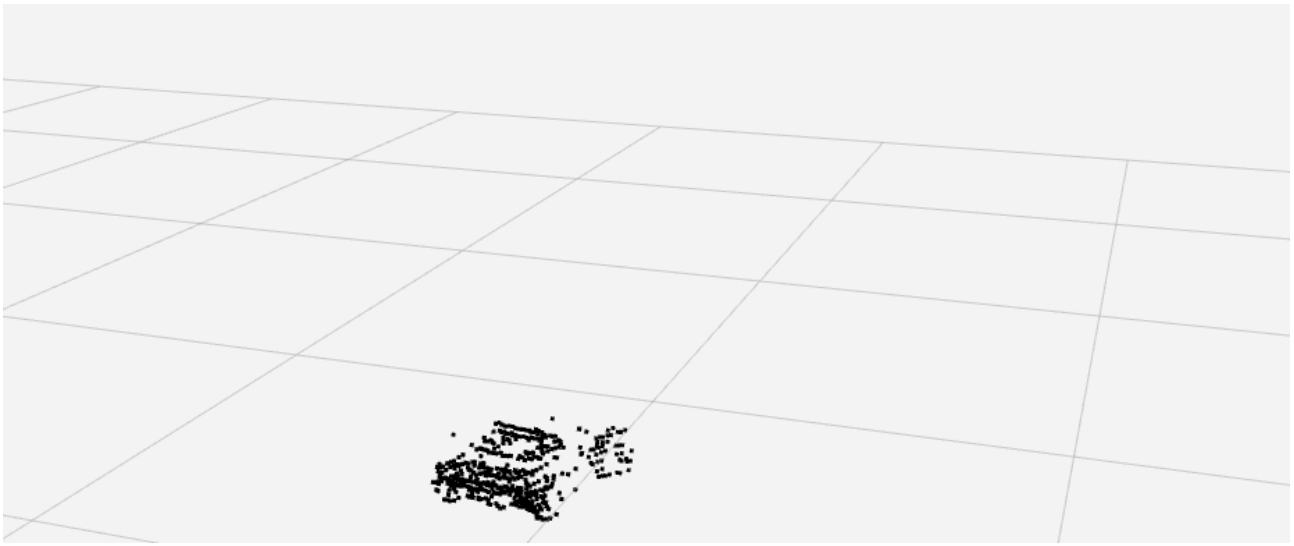


Рисунок 11 – Выходное облако точек метода clustering

Определения положения задней оси ведущего автомобиля относительно задней оси ведомого автомобиля

Так как, основными целями системы компьютерного зрения слежения является определение расстояния между центром задней оси ведомого автомобиля и центром задней оси ведущего грузовика — l_d и определение угла отклонения от направления кузова ведомого грузовика и центром задней оси ведущего грузовика – α , то на выходе система компьютерного зрения должна выдавать два значения: l_d и α.

Для определения центра задней оси ведущего автомобиля необходимо ограничить облако точек объемной фигурой с помощью технологий ограничивающих объемов (bounding volume), которые позволяют описывать сложные объекты простыми геометрические фигурами[5]. Для задачи определения центра ведущего автомобиля была выбрана технология OBB

(Object-oriented bounding boxes) Объектно-ориентированного ограничивающего параллелепипеда, которая ограничивает объем вне зависимости от глобальных координатных осей и показывает истинное направление объекта. Для реализации технологии OBB был создан метод `void PointcloudPipeline::bb_car(std::shared_ptr<pcl::PointCloud<pcl::PointXYZI> > input_cloud)` (листинг 6).

Листинг 6 – Реализация метода технологии OBB

```
void
PointcloudPipeline::bb_car(std::shared_ptr<pcl::PointCloud<pcl::PointXYZI>
> input_cloud) {
   pcl::MomentOfInertiaEstimation <pcl::PointXYZI> feature_extractor;
   pcl::PointXYZI min_point_OBB;
   pcl::PointXYZI max_point_OBB;
   pcl::PointXYZI position_OBB;
   feature_extractor.setInputCloud (input_cloud);
   feature_extractor.compute ();
   feature_extractor.getOBB (min_point_OBB, max_point_OBB, position_OBB,
rotational_matrix_OBB);
   car_width = max_point_OBB.y - min_point_OBB.y;
   car_height = max_point_OBB.z - min_point_OBB.z;
   car_length = max_point_OBB.x - min_point_OBB.x;
   x_center = position_OBB.x;
   y_center = position_OBB.y;
   z_center = position_OBB.z;
  }
```

Результат работы метода `bb_car`, осуществляющего алгоритм определения границ автомобиля (OBB), представлен на рисунке 12:



Рисунок 12 – Объектно-ориентированный ограничивающий параллелепипед метода bb_car

Для определения координат определения положения центра задней оси ведомого автомобиля относительно центра задней оси ведомого автомобиля, необходимо учитывать положение лидара относительно центра задней оси. В разрабатываемом прототипе лидар будет располагаться на продольной оси

автомобиля на расстоянии X_{lidar} от центра задней оси автомобиля (рисунок 13).

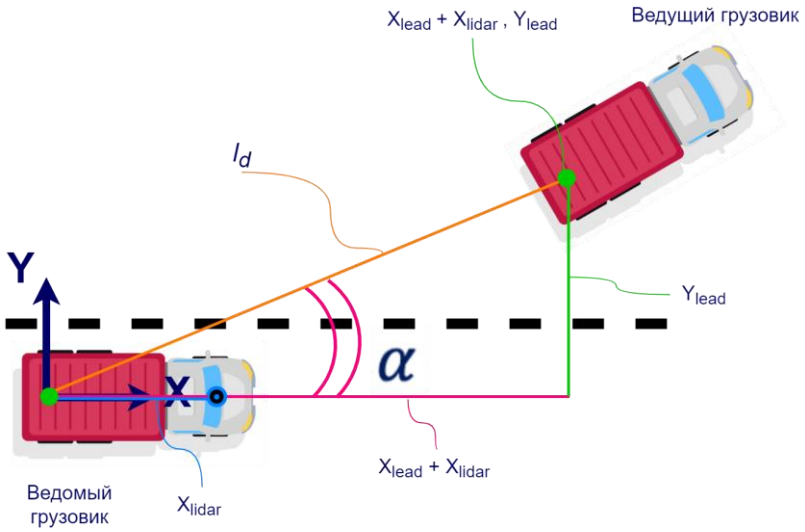


Рисунок 13 – Нахождение расстояния l_d и угла α с помощью лидара

Так образом выходные величины α и l_d будут находится по формулам:

$$\alpha = tan^{-1}\left(\frac{Y_{lead}}{X_{lead}+X_{lidar}}\right) \qquad l_d = \sqrt{(Y_{lead})^2 + (X_{lead} + X_{lidar})^2}$$

Реализация метода кластеризации на языке C++ представлена в листинге 7.

Листинг 7 – Реализация метода определения положения задней оси ведущего автомобиля относительно задней оси ведомого автомобиля

```
void PointcloudPipeline::generate_ld_alpha(){
   double ld;
   double alpha;
   double x_temp = rotational_matrix_OBB(0,0)*(-
center_to_rear_axle_distance);
   double y_temp = rotational_matrix_OBB(1,0)*(-
center_to_rear_axle_distance);
   double x_lidar = x_temp + x_center;
   double y_lidar = y_temp + y_center;
   double x_dist_from_lead_to_own = x_lidar + lidar_to_rear_axle_distance;
   double y_dist_from_lead_to_own = y_lidar;
   ld = std::sqrt((x_dist_from_lead_to_own * x_dist_from_lead_to_own) +
(y_dist_from_lead_to_own * y_dist_from_lead_to_own));
   alpha = std::atan2 (y_dist_from_lead_to_own, x_dist_from_lead_to_own);
   std_msgs::msg::Float64 alpha_msg;
   std_msgs::msg::Float64 ld_msg;
   alpha_msg.data = alpha;
   ld_msg.data = ld;
   ld_publisher_->publish(ld_msg);
   alpha_publisher_->publish(alpha_msg);
}
```

Вывод работы метода `generate_ld_alpha`, осуществляющего алгоритм определения положения центра задней оси ведущего автомобиля относительно центра задней оси ведомого автомобиля, представлен на рисунке 14 (вывод расстояния l_d) и на рисунке 15 (вывод угла α).

```
anastasiia@anastasiia-X510UNR:~/brtadehome/CLWDT$ ros2 topic echo /ld
data: 2.9885975096744053
---
data: 3.018565957266211
---
data: 3.0319579248508037
---
data: 3.0906088973346963
---
data: 3.072562408424185
---
data: 3.325171057268376
---
data: 3.103201761140996
---
data: 3.110855517558081
---
data: 3.0961805078958995
---
data: 3.1098141574051357
---
```

Рисунок 14 — Вывод полученного расстояния l_d

```
^Canastasiia@anastasiia-X510UNR:~/brtadehome/CLWDT$ ros2 topic echo /alpha
data: -0.004890039847794071
---
data: 0.10625378420437476
---
data: -0.006638004628261193
---
data: 0.08844449119143204
---
data: 0.07677711215356017
---
data: 0.06867431248387232
---
data: 0.06291856976648201
---
data: 0.06562009949486071
---
data: 0.04999118970885866
---
data: 0.04354137061803641
---
```

Рисунок 15 — Вывод полученного угла α

В результате выполнения данной работы был разработан и реализован алгоритм определения положения центра задней оси ведущего автомобиля относительно центра задней оси ведомого автомобиля для системы компьютерного зрения.

Литература

References

1. Malaia A.G. Kalinin A.V., «Development of control algorithm concept for convoy-led driverless wheel truck» volume 4 No. 6 (2021): Scientific and educational journal of teachers and students "StudNet" – URL: https://stud.net.ru/razrabotka-koncepcii-algoritma-upravleniya-bespilotnogo-kolyosnogo-tyagacha-dvizhushhegosya-v-kolonne-za-napravlyayushhim/
2. Datasheet Velodyne Ultra Puck VLP-32 – URL: https://www.mapix.com/wp-content/uploads/2018/07/63-9378_Rev-D_ULTRA-Puck_VLP-32C_Datasheet_Web.pdf
3. Colcov S. N. “Cluster analysis” — 34 стр. Internet Studies Lab, Москва, ВШЭ — URL:
4. https://linis.hse.ru/data/2015/06/17/1083971020/%D0%BA%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B9%20%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_10.pdf
5. Radu Bogdan Rusu “Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments” — Institut für Informatik der Technischen Universität München] — URL: http://mediatum.ub.tum.de/doc/800632/941254.pdf
6. Introduction to discrete-oriented polyhedra for collision detection – URL: https://sudonull.com/post/98755-Introduction-to-discrete-oriented-polyhedra-for-collision-detection

© Малая А.Г., Калинин А.В., 2021 Научно-образовательный журнал для студентов и преподавателей «StudNet» №2/2022.

Для цитирования: Малая А.Г., Калинин А.В. РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМА РАБОТЫ СИСТЕМЫ КОМПЬЮТЕРНОГО ЗРЕНИЯ ДЛЯ ОПРЕДЕЛЕНИЯ ПОЛОЖЕНИЯ ЦЕНТРА ЗАДНЕЙ ОСИ ВЕДУЩЕГО АВТОМОБИЛЯ ОТНОСИТЕЛЬНО ЦЕНТРА ЗАДНЕЙ ОСИ ВЕДОМОГО АВТОМОБИЛЯ, ДВИЖУЩИХСЯ В КОЛОННЕ// Научно-образовательный журнал для студентов и преподавателей «StudNet» №2/2022.