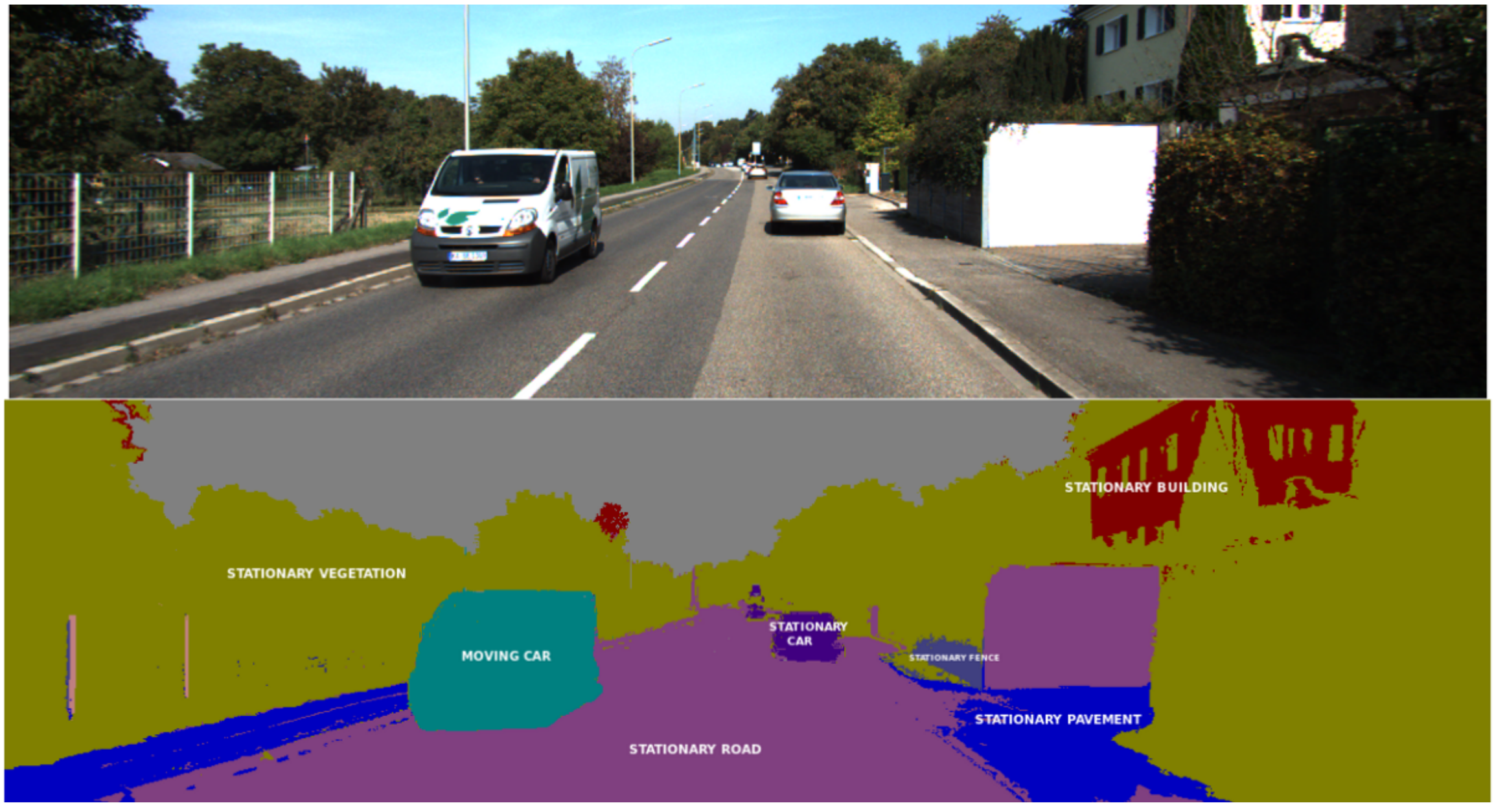
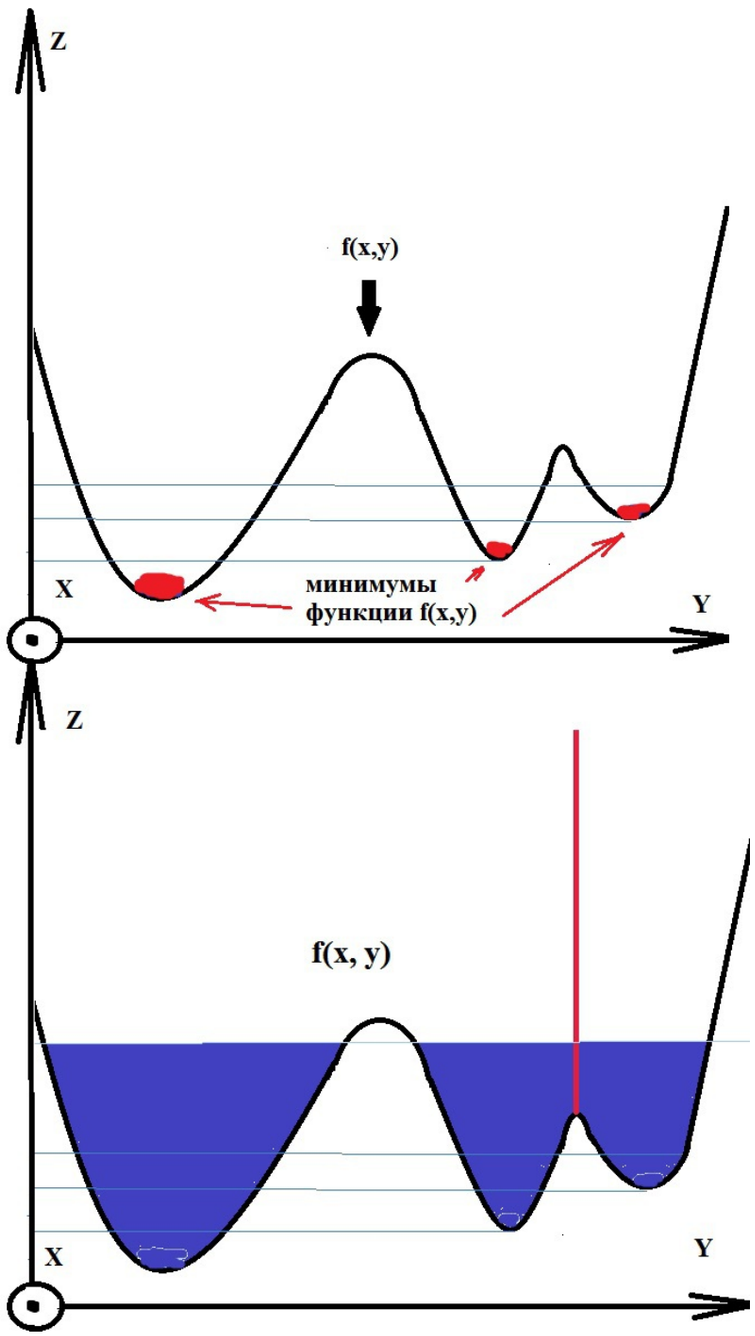
СЕГМЕНТАЦИЯ - АЛГОРИТМЫ

ВАН СЕЮЙ



алгоритмов сегментации



1.Алгоритм сегментации по водоразделам (WaterShed)

Алгоритм работает с изображением как с функцией от двух переменных ***f=I(x, y)***, где ***x, y*** - координаты пикселя.

Значением функции может быть интенсивность или модуль градиента. Для наибольшего контраста можно взять градиент от изображения. Если по оси ***OZ*** откладывать абсолютное значение градиента, то в местах перепада интенсивности образуются хребты, а в однородных регионах – равнины. После нахождения минимумов функции ***f***, идет процесс заполнения “водой”, который начинается с глобального минимума. Как только уровень воды достигает значения очередного локального минимума, начинается его заполнение водой.

Когда два региона начинают сливаться, строится перегородка, чтобы предотвратить объединение областей. Вода продолжит подниматься до тех пор, пока регионы не будут отделяться только искусственно построенными перегородками.

5.2 Multiple Source Region Grow Algorithm

Initialize a priority queue Q of pairs
Choose a set of seed elements $\{s_i\}$
Create a cluster C_i from each seed s_i
Insert the pairs $< s_i, C_i >$ to Q
Loop until until Q is empty
    Get the next pair $< s_k, C_k >$ from Q
    If s_k is not clustered already and
    s_k can be clustered into C_k
        Cluster s_k into C_k
        For all un-clustered neighbors s_i of s_k
            insert $< s_i, C_k >$ to Q
Merge small clusters into neighboring ones

Код Для WaterShed:

```
Mat image = imread(“target.jpg", CV_LOAD_IMAGE_COLOR);
// выделим контуры
Mat imageGray, imageBin;
cvtColor(image, imageGray, CV_BGR2GRAY);
threshold(imageGray, imageBin, 100, 255, THRESH_BINARY);
std::vector<std::vector<Point> > contours;
std::vector<Vec4i> hierarchy;
findContours(imageBin, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);
Mat markers(image.size(), CV_32SC1);
markers = Scalar::all(0);
int compCount = 0;
for(int idx = 0; idx >= 0; idx = hierarchy[idx][0], compCount++)
{
   drawContours(markers, contours, idx, Scalar::all(compCount+1), -1, 8, hierarchy, INT_MAX);
}
std::vector<Vec3b> colorTab(compCount);
for(int i = 0; i < compCount; i++)
{
   colorTab[i] = Vec3b(rand()&255, rand()&255, rand()&255);
}
watershed(image, markers);
Mat wshed(markers.size(), CV_8UC3);
for(int i = 0; i < markers.rows; i++)
{
   for(int j = 0; j < markers.cols; j++)
   {
      int index = markers.at<int>(i, j);
      if(index == -1)
wshed.at<Vec3b>(i, j) = Vec3b(0, 0, 0);
      else if (index == 0)
wshed.at<Vec3b>(i, j) = Vec3b(255, 255, 255);
      else
wshed.at<Vec3b>(i, j) = colorTab[index - 1];
   }
}
imshow("watershed transform", wshed);
waitKey(0);
```

Алгоритм сегментации MeanShift

MeanShift группирует объекты с близкими признаками. Пиксели со схожими признаками объединяются в один сегмент, на выходе получаем изображение с однородными областями

Например, в качестве координат *в пространстве признаков* можно выбрать координаты пикселя *(x, y)* и компоненты *RGB* пикселя. Изобразив пиксели в пространстве признаков, можно заметить сгущения в определенных местах.

Чтобы легче было описывать сгущения точек, вводится ***фунция плотности***:

$$f(\vec{x}) = \frac{1}{Nh^d} \sum_{i=1}^{N} K(\frac{\vec{x}-\vec{x}_i}{h})$$

$\vec{x}_i$ – вектор признаков ***i***-ого пикселя, ***d*** — количество признаков, ***N*** — число пикселей, ***h*** — параметр, отвечающий за гладкость

— ядро. Максимумы функции F(x)- расположены в точках сгущения пикселей изображения в пространстве признаков. Пиксели, принадлежащие одному локальному максимуму, $K(\vec{x})$единяются в один сегмент. Получается, чтобы найти к какому из центров сгущения относится пиксель, надо шагать по градиенту

F(x)—для нахождения ближайшего локального максимума.

Для оценки градиента функции плотности можно использовать вектор среднего сдвига $M_h(\vec{x})$

В качестве ядра $K(\vec{x})$ в OpenCV используется ядро Епанечникова [4]:

$$K_E(\vec{x}) = \begin{cases} \frac{1}{2c_d}(d+2)(1-\vec{x}^T\vec{x}), \text{если } \vec{x}^T\vec{x} < 1 \\ 0, \quad \text{в остальных случаях} \end{cases}$$

c_d — это объем ***d***-мерной сферы с единичным радиусом.

$$grad(f(\vec{x})) = \frac{1}{Nh^d}\sum_{i=1}^{N} grad\left(K\left(\frac{\vec{x}-\vec{x}_i}{h}\right)\right) = \frac{1}{N(h^d c_d)}\frac{d+2}{h^2}\sum_{\vec{x}_i \in S_h(\vec{x})}(\vec{x}_i - \vec{x})$$

$$= \frac{n_{h,\vec{x}}}{N(h^d c_d)}\frac{d+2}{h^2}\left(\frac{1}{n_{h,\vec{x}}}\sum_{\vec{x}_i \in S_h(\vec{x})}(\vec{x}_i - \vec{x})\right)$$

$\vec{x}_i \in S_h(\vec{x})$ означает, что сумма идет не по всем пикселям, а только по тем, которые попали в сферу радиусом ***h*** с центром в точке, куда указывает вектор $\vec{x}$ в пространстве признаков [4]. Это вводится специально, чтобы уменьшить количество вычислений. $(h^d c_d)$ — объем ***d***-мерной сферы с радиусом h, Можно отдельно задавать радиус для пространственных координат и отдельно радиус в пространстве цветов. $n_{h,\vec{x}}$ — число пикселей, попавших в сферу. Величину $\frac{n_{h,\vec{x}}}{N(h^d c_d)}$ можно рассматривать как оценку значения $f(\vec{x})$ в области $S_h(\vec{x})$.

$$M_h(\vec{x}) = \frac{1}{n_{h,\vec{x}}}\sum_{\vec{x}_i \in S_h(\vec{x})}(\vec{x}_i - \vec{x})$$

$$grad(f(\vec{x})) = f(\vec{x})\frac{d+2}{h^2}M_h(\vec{x})$$

Поэтому, чтобы шагать по градиенту, достаточно вычислить значение Mh(x) — вектора среднего сдвига.
Следует помнить, что при выборе другого ядра вектор среднего сдвига будет выглядеть иначе.
При выборе в качестве признаков координат пикселей и интенсивностей по цветам в один сегмент будут объединяться пиксели с близкими цветами и расположенные недалеко друг от друга. Соответственно, если выбрать другой вектор признаков, то объединение пикселей в сегменты уже будет идти по нему. Например, если убрать из признаков координаты, то небо и озеро будут считаться одним сегментом, так как пиксели этих объектов в пространстве признаков попали бы в один локальный максимум.
Если объект, который хотим выделить, состоит из областей, сильно различающихся по цвету, то **MeanShift** не сможет объединить эти регионы в один, и наш объект будет состоять из нескольких сегментов. Но зато хорошо справиться с однородным по цвету предметом на пестром фоне. Ещё **MeanShift** используют при реализации алгоритма слежения за движущимися объектами

Алгоритм сегментации GrabCut

Это интерактивный алгоритм выделения объекта, разрабатывался как более удобная альтернатива магнитному лассо (чтобы выделить объект, пользователю требовалось обвести его контур с помощью мыши). Для работы алгоритма достаточно заключить объект вместе с частью фона в прямоугольник (grab). Сегментирование объекта произойдет автоматически (cut).

Могут возникнуть сложности при сегментации, если внутри ограничивающего прямоугольника присутствуют цвета, которые встречаются в большом количестве не только в объекте, но и на фоне. В этом случае можно поставить дополнительные метки объекта (красная линия) и фона (синяя линия).

Рассмотрим идею алгоритма. За основу взят алгоритм интерактивной сегментации GraphCut, где пользователю надо поставить маркеры на фон и на объект. Изображение рассматривается как массив $z(z_1, \ldots, z_n, \ldots, z_N)$. **z** — значения интенсивности пикселей, ***N***-общее число пикселей. Для отделения объекта от фона алгоритм определяет значения элементов массива прозрачности $a(a_1, \ldots, a_n, \ldots, a_N)$, причем a_n может принимать два значения, если a_n **= 0**, значит пиксель принадлежит фону, если a_n **= *1***, то объекту. Внутренний параметр θ содержит гистограмму распределения интенсивности переднего плана и гистограмму фона:

$$\theta = \{h(z; a),\ a = 0,\ 1\}.$$

Задача сегментации — найти неизвестные a_n. Рассматривается функция энергии:

$$E(a, \theta, z) = U(a,\ \theta,\ z) + V\ (a,\ z)$$

Причем минимум энергии соответствует наилучшей сегментации.

$$U(a,\ \theta,\ z) = -\sum_n log\ h(z_n,\ a_n)$$

$$V\ (a,\ z) = \sum_{(m,n)\in C} \frac{1}{dis(m,n)}[a_n \neq a_m]exp(-\beta(z_m - z_n)^2)$$

V (a, z) — слагаемое отвечает за связь между пикселями. Сумма идет по всем парам пикселей, которые являются соседями, ***dis(m, n)*** — евклидов расстояние. [an != am] отвечает за участие пар пикселей в сумме, если $a_n = a_m$, то эта пара не будет учитываться.
$U(a, \theta, z)$ — отвечает за качество сегментации, т.е. разделение объекта от фона.

Найдя глобальный минимум функции энергии ***E***, получим массив прозрачности

$$\hat{\alpha} = argmin_{\ a}\ E(a, \theta)$$

энергии, изображение описывается как граф и ищется минимальный разрез графа. В отличие от GraphCut в алгоритме **GrabCut** пиксели рассматриваются в RGB пространстве, поэтому для описания цветовой статистики используют смесь гауссиан (Gaussian Mixture Model — GMM).

Мы рассмотрели только небольшую часть существующих алгоритмов. В результате сегментации на изображении выделяются области, в которые объединяются пиксели по выбранным признакам. Для заливки однородных по цвету объектов подойдет **FloodFill**. С задачей отделения конкретного объекта от фона хорошо справится **GrabCut**. Если использовать реализацию **MeanShift** из OpenCV, то пиксели, близкие по цвету и координатам, будут кластеризованы. **WaterShed** подой дет для изображений с простой текстурой.