Leap Motion, карта глубин и OpenCV

Обзор

1. Leap Motion
2. Как мы творили темную магию объемного зрения

Leap Motion

Особенности

- 2 камеры с линзами fish-eye
- 3 инфракрасных светодиода
- Интерфейс: USB 3.0 micro-B
- Дальность распознавания объектов $\approx$ 60 см
- Угол обзора $\approx 135^\circ$
- Есть стандартный SDK, который периодически обновляется
- Монохромные изображения с разрешением 620x240 на выходе

Leap SDK

SDK позволяет делать множество интересных вещей:

- Определять руки, ладони, отдельные пальцы
- Определять инструменты (объекты цилиндрической формы вытянутые и тонкие)
- Определять жесты, такие как свайпы, нажатия и пр.
- Возвращать сырые изображения с камер

SGBM

Чем Leap Motion плох?

Цитата из интервью с одним из разработчиков:
"... It's very different from other things like the Kinect, and in normal device operation we have very different priorities than most other technologies. Our priority is precision, small movements, very low latency, very low CPU usage - so in order to do that we will often be making sacrifices that make what the device is doing completely not applicable to what I think you're getting at, which is 3D scanning".

Чем Leap Motion плох?

- Камеры очень чувствительны к освещению
- Отражения света от светодиодов распознаются как объекты

Этапы

- Калибровка
- Построение карты глубин (карты разностей/disparity map)
- Построение облака точек
- Blob detection

Калибровка

В процессе производства камеры и линз всегда есть некоторые неточности, поэтому, всегда когда мы работаем со стереоскопией существует необходимость в калибровке: нахождении внешних и внутренних параметров стереопары.

Калибровка

Ниже представлена модель проективной камеры

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{11} & r_{11} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (1)$$

Внутренние параметры

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

(f_x, f_y) - фокусное расстояние
(c_x, c_y) - координаты принципальной точки(точка пересечения плоскости изображения с оптической осью, совпадающая с центром фотографии. В реальных камерах, как правило, бывает немного смещена из-за оптических искажений)

Внешние параметры

$$\begin{bmatrix} r_{11} & r_{11} & r_{11} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}$$

Матрица для перехода из ситемы координат мира в систему координат камеры

Другие переменные

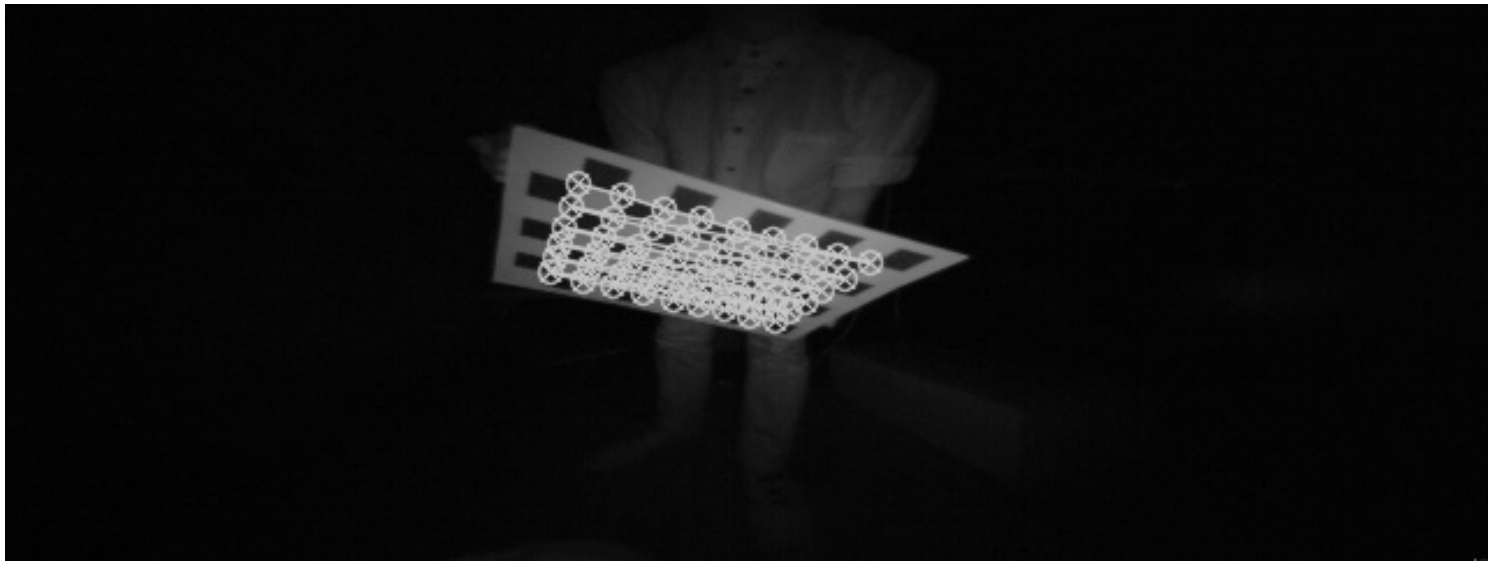
(X, Y, Z) - координаты исходной точки
(u, v) - координаты на изображении
s - переменная отвечающая за масштаб

Калибровка при помощи openCV

Один из стандартных способов для калибровки использование изображения шахматной доски.
В OpenCV есть несколько стандартных функций для калибровки с использованием шахматной доски и других шаблонов, нам понадобились:

- **FindChessboardCorners** - Позволяет найти на изображении углы шахматной доски
- **StereoCalibrate** - Из тех точек что мы получили с помощью **FindChessboardCorners** для левого и правого изображений, а так же из точек соответствущих точкам на реальной доске возвращает матрицы для работы с камерой.

Калибровка



Полученные данные

- Матрицы камер - внутренние параметры камер
- Коэффициенты искажений для камер
- Матрицы для перехода между системами координат камер
- Матрицы связывающие точки между изображениями с разных камер

Ректификация

Ректификация - процесс необходимый для большинства алгоритмов поиска карты глубин. Он заключается в выполнении преобразования при котором и правое, и левое изображение проецируются на плоскость, параллельную базовой линии (линии, соединяющей оптические центры камер). Тогда если (x, y) - точка левой проекции, а (x_0, y_0) - соответствующая ей точка правой проекции, то $y = y_0$

SGBM

Ректификация

Для ректификации используются функции **StereoRectify** и **InitUndistortRectifyMap**: После использования первой мы получаем матрицы для поворота систем координат камер и матрицы для перехода в новые системы координат. Также мы получаем матрицу которая позволит нам переходить от карты разностей к карте глубин. Вторая функция нужна для того чтобы работать с конкретными изображениями.

Алгоритмы нахождения карт разностей в OpenCV

Существует несколько встроенных в OpenCV алгоритмов для нахождения разностных карт:

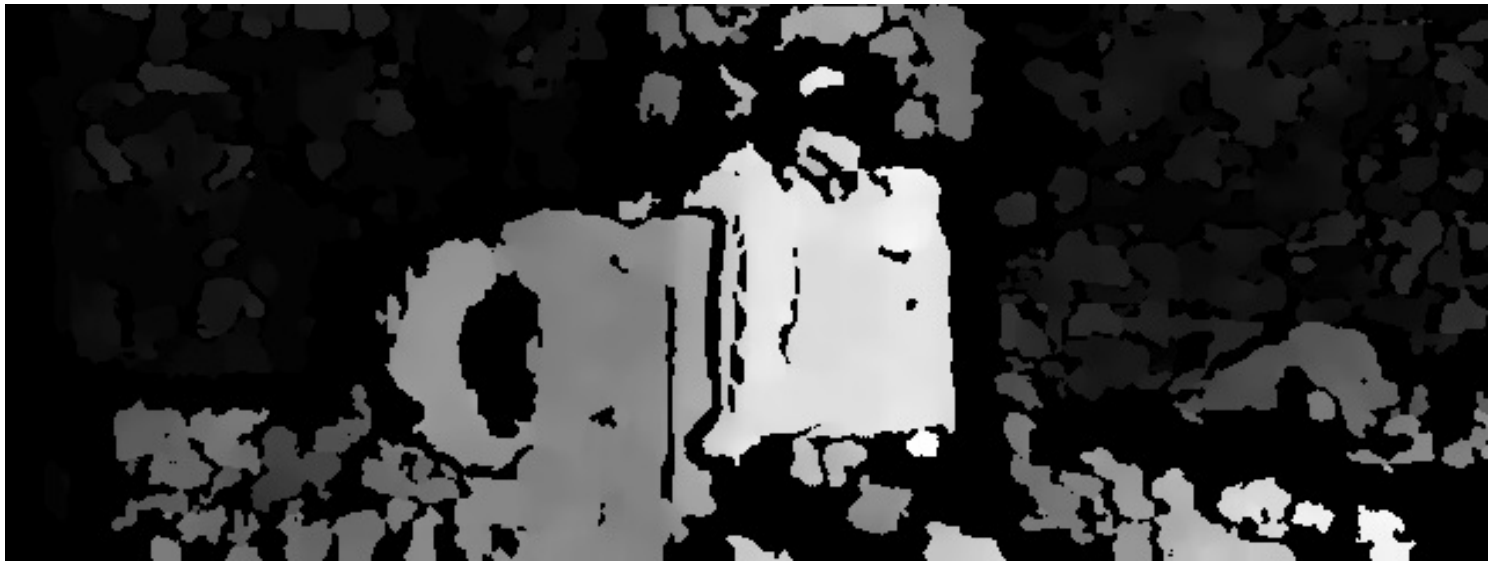
- StereoBM - быстро (доли секунд), много шумов.
- StereoSGBM - чуть медленней (все еще доли секунд), шумов меньше.
- StereoVar - очень медленно(порядка ста секунд), карты разностей близки к идеальным.

Для решения задачи была выбрана функция StereoSGBM.

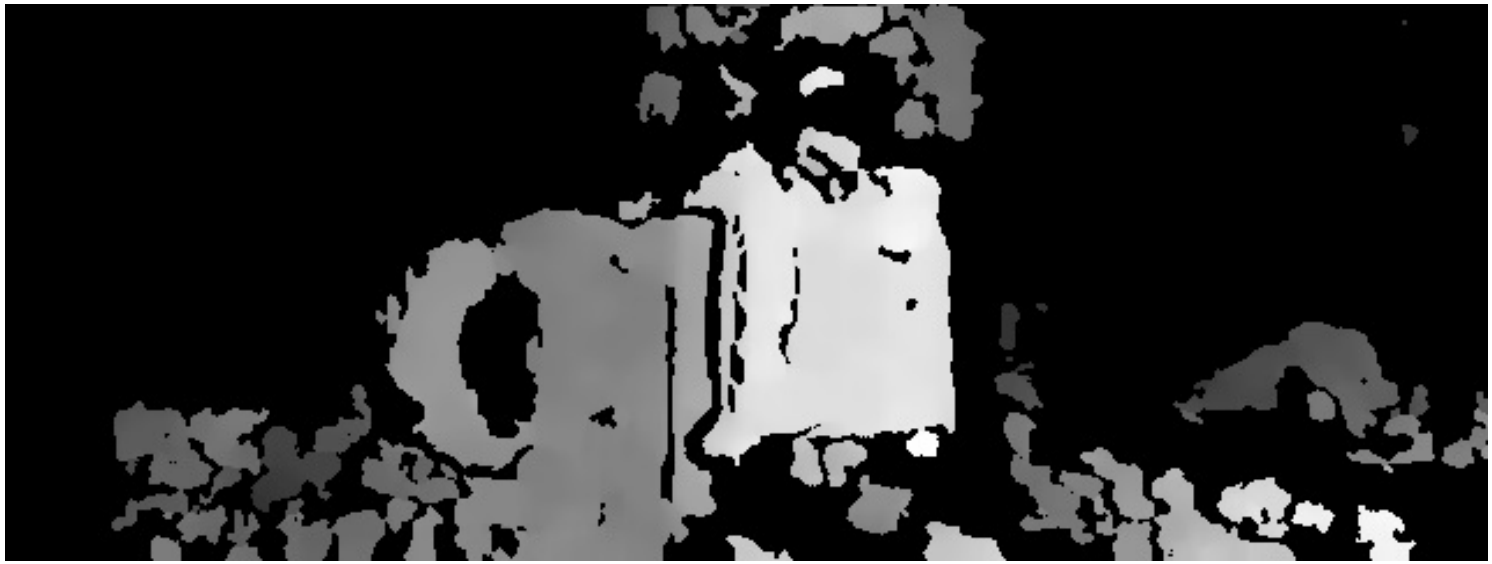
StereoSGBM

Получает на вход два изображения и некоторое количество параметров, возвращает карту разностей. Путем подбора волшебных чисел и написания утилиты, удобной для подбора коэффициентов, были получены результаты содержащие минимально возможное количество шумов.

SGBM



SGBM with LUT



Построение облака точек

Следующий этап - построить облако точек - это массив точек вида (x, y, z) - координат относительно камеры, в этом помогает функция **ReprojectImageTo3D** из библиотеки OpenCV.
Затем запоминаются z-компоненты при разных расстояниях от камеры и по полученным значениям интерполируем функцию зависимости расстояния до объекта от интенсивности цвета на карте глубин.

Blob detection

Blob detection - это нахождение регионов на изображении, которые отличаются по каким-то свойствам, вроде цвета или яркости, в сравнении с окружающими регионами.
Грубо говоря, blob (капля) - это регион изображения, в котором какие-то свойства постоянны или почти постоянны.
Все точки blob'а считаются в каком-то смысле одинаковыми.

Blob detection

Следующий этап - применить blob detection для карты глубин.
С этой задачей вновь помог справиться OpenCV.
Алгоритм, реализованный в OpenCV, таков:

- Конвертируем изображение "бинарное"изображение - только из черного и белого цветов по заданным извне границам
- Извлекаем компоненты связности
- Группируем центры компонент - близкие компоненты относятся к одному blob'y
- По полученным группам получаем результирующую позицию blob'a