Преимущества C++ в нетипичных условиях

C++ в КР

1. Нехватка специалистов

C++ в КР

1. Нехватка специалистов
2. Отсутствие сообщества

C++ в КР

1. Нехватка специалистов
2. Отсутствие сообщества
3. Сложность масштабирования

C++ в КР

1. Нехватка специалистов
2. Отсутствие сообщества
3. Сложность масштабирования
4. Непопулярность

Реальные случаи эффективного использования C++

1. Оптимизация подсчета проходимости магазинов и пропускной способности перекрестков (python)

2. Оптимизация матчмейкинга в криптобирже (golang)

CASE#0 Проходимость магазинов и перекрестков

Зачем считать людей?

1. Контроль персонала

Зачем считать людей?

1. Контроль персонала
2. Аналитика

Зачем считать людей?

1. Контроль персонала
2. Аналитика
3. Эффективное расписание

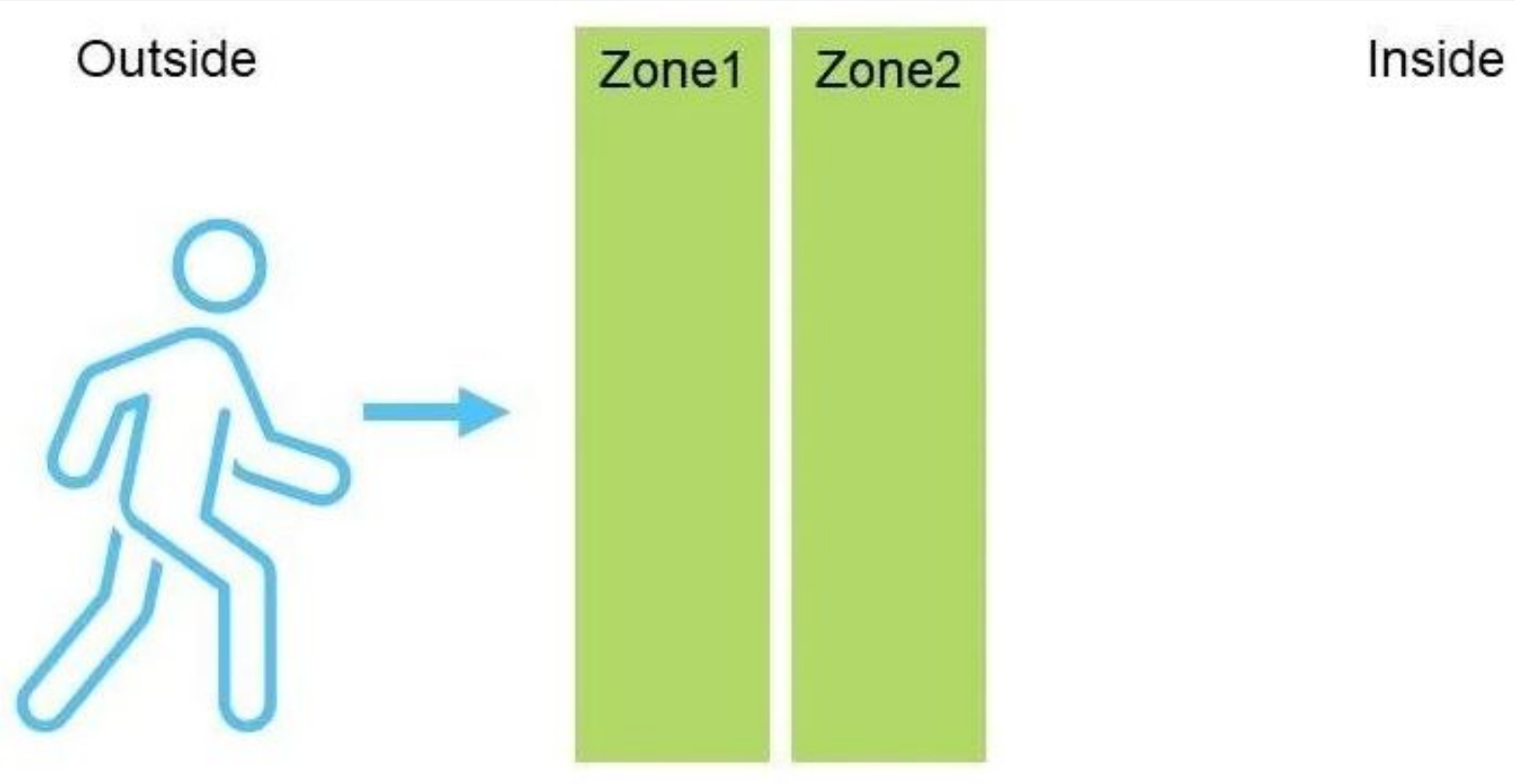
Зачем считать людей?

1. Контроль персонала
2. Аналитика
3. Эффективное расписание
4. Безопасность

Hardware решения

1. Лазерный или ИК счетчик

ИК счетчик



Hardware решения

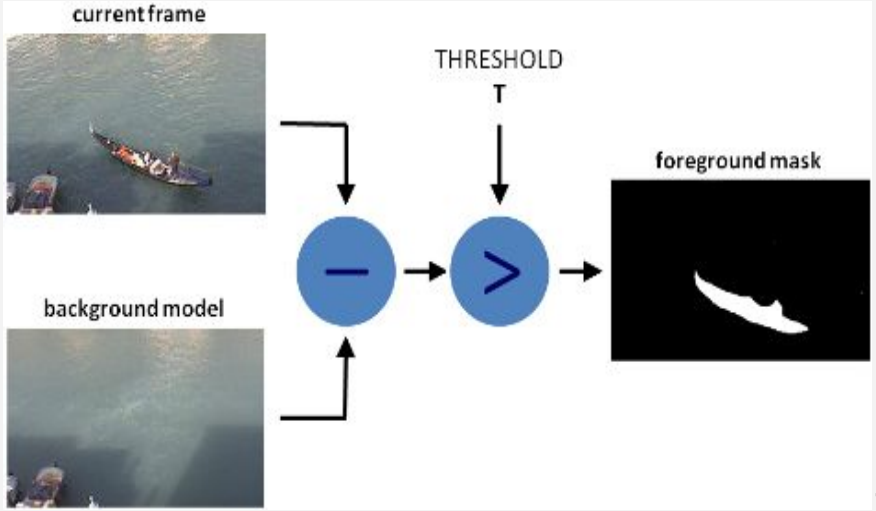
1. Лазерный или ИК счетчик
2. Ультразвуковой дальномер

Hardware решения

1. Лазерный или ИК счетчик
2. Ультразвуковой дальномер
3. Турникет

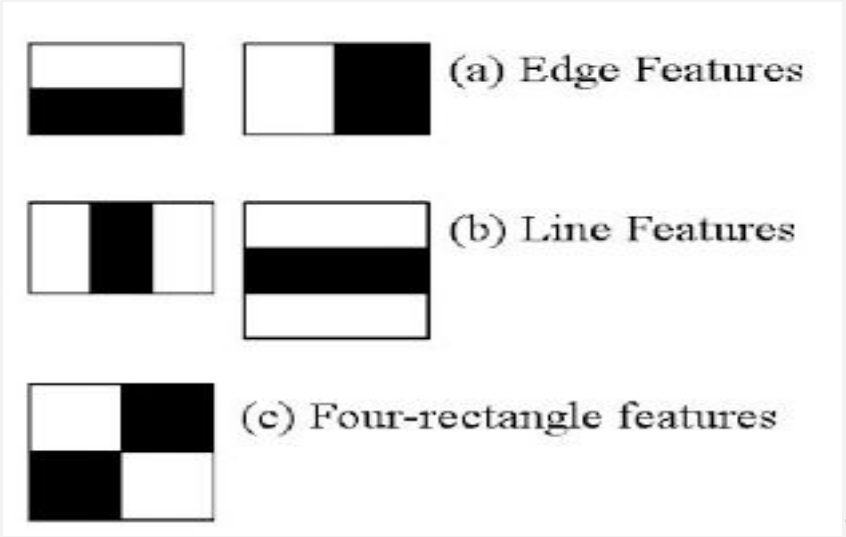
Software решения

1. Background subtraction



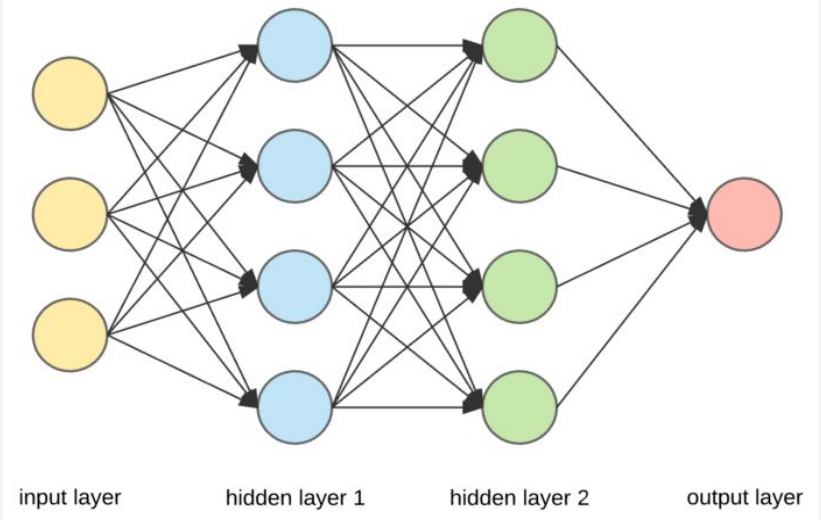
Software решения

1. Background subtraction
2. Каскады Хаара



Software решения

1. Background subtraction
2. Каскады Хаара
3. Нейронные сети



Первый результат с tensorflow + resnet

История Tensorflow

1. 2011 - стартует как проект DistBelief

История Tensorflow

1. 2011 - стартует как проект DistBelief
2. Рефакторинг и оптимизация

История Tensorflow

1. 2011 - стартует как проект DistBelief
2. Рефакторинг и оптимизация
3. 2015 - открыт для свободного доступа

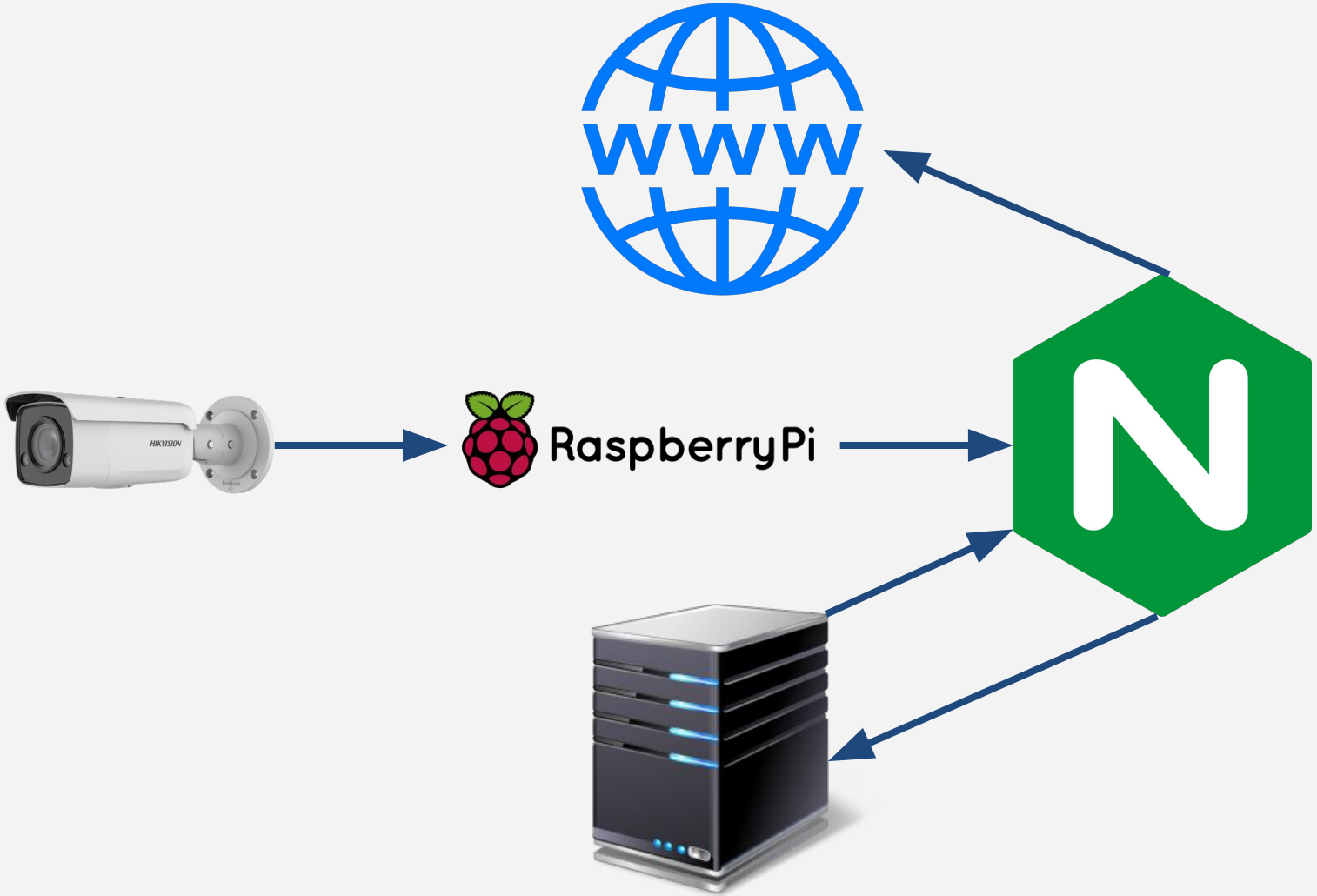
История Tensorflow

1. 2011 - стартует как проект DistBelief
2. Рефакторинг и оптимизация
3. 2015 - открыт для свободного доступа
4. 2016 - TPU

Следующий этап - получить стрим со своей камеры

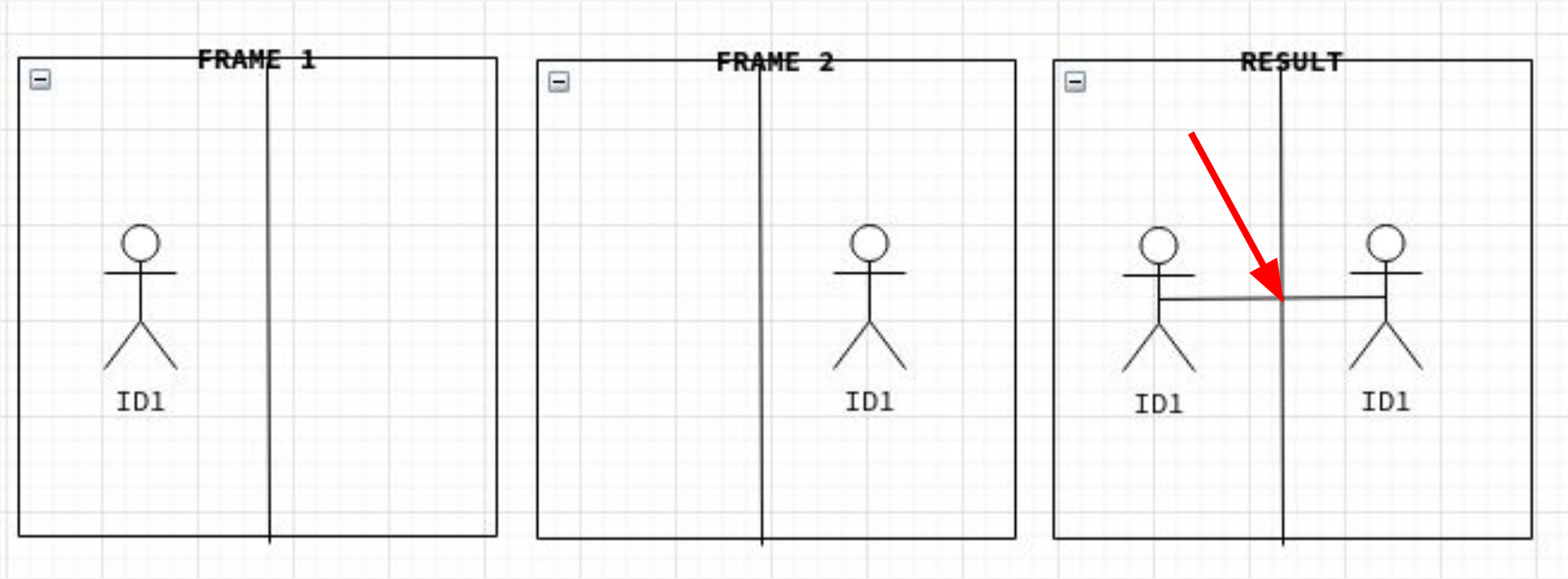
Необходимый минимум

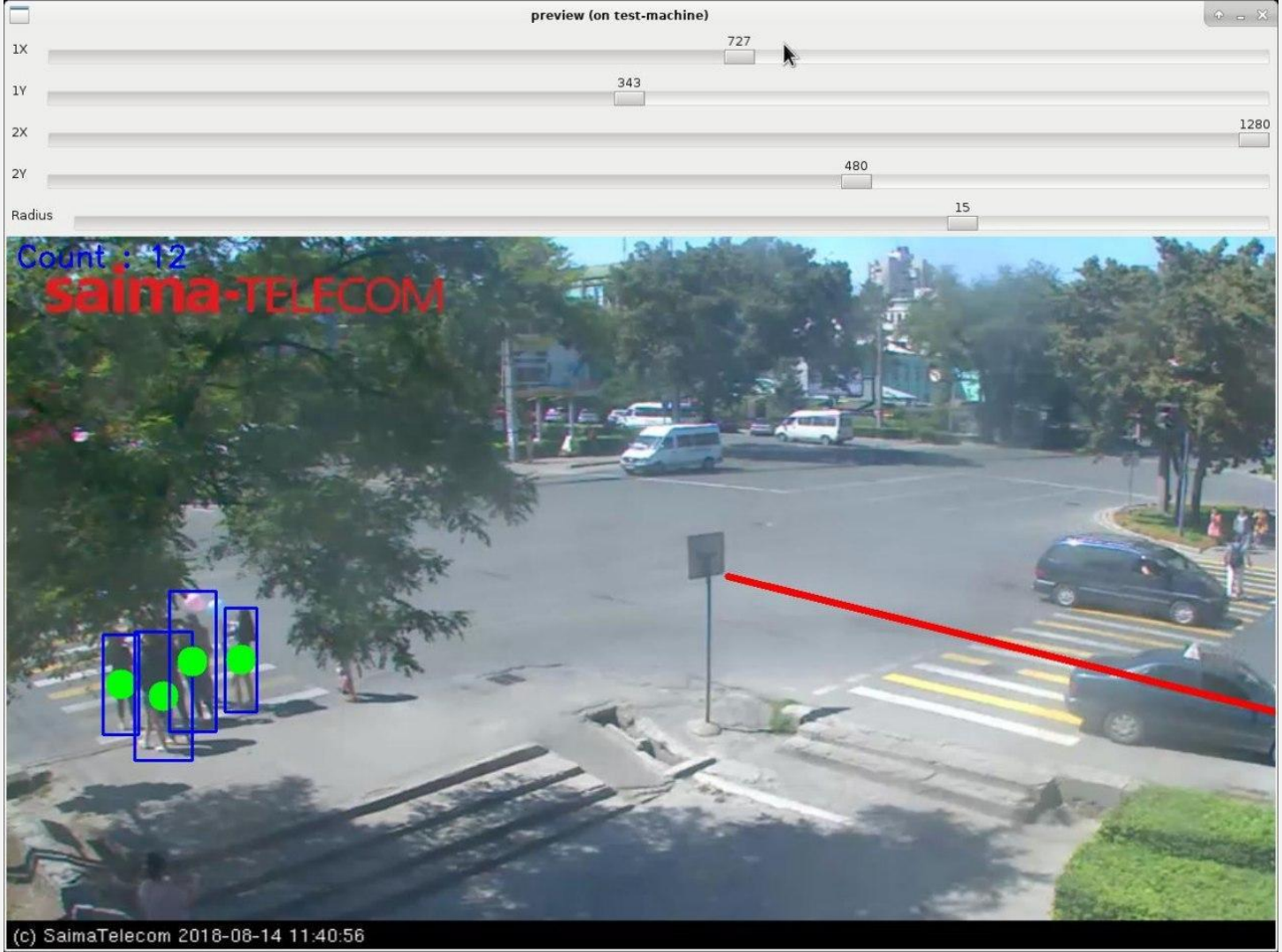
1. IP-камера
2. Raspberry PI + модем
3. Сервер с видеокартой Nvidia
4. Nginx + rtmp модуль.
5. Изолента (синяя)

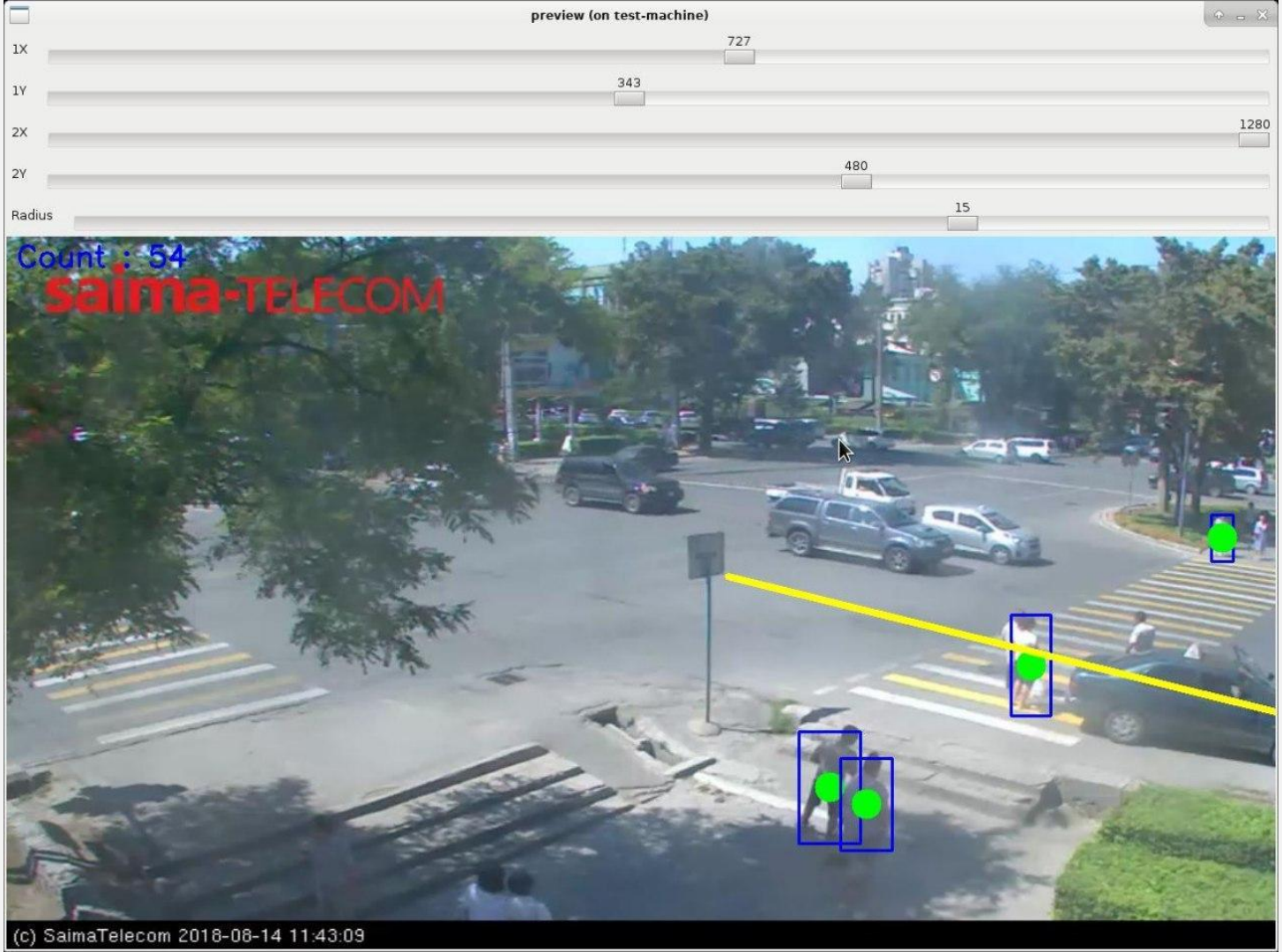

www
HIKVISION
RaspberryPi

Высокотехнологичный сетап для подсчета количества уходящих на перекур сотрудников.

В идеале должно работать так




preview (on test-machine)
1X
727
1Y
343
2X
1280
2Y
480
Radius
15
Count : 12
saima-TELECOM
(c) SaimaTelecom 2018-08-14 11:40:56


preview (on test-machine)
1X
727
1Y
343
2X
1280
2Y
480
Radius
15
Count : 54
saima-TELECOM
(c) SaimaTelecom 2018-08-14 11:43:09

Связка Python + Tensorflow дает
FPS = 8
Как результат - частые ложные срабатывания

Попытки увеличить FPS

1. Меньший размер кадра

Попытки увеличить FPS

1. Меньший размер кадра
2. Параллельная обработка кадров

Попытки увеличить FPS

1. Меньший размер кадра
2. Параллельная обработка кадров
3. Более мощный GPU/TPU

Попытки увеличить FPS

1. Меньший размер кадра
2. Параллельная обработка кадров
3. Более мощный GPU/TPU
4. Multiple Object Trackers

В результате всех попыток

FPS = 9

Darknet + Yolo

1. Написан на C
2. Использует cuda и cudnn
3. Заявленный FPS = 30

Интеграция и доработка Darknet

1. pydarknet

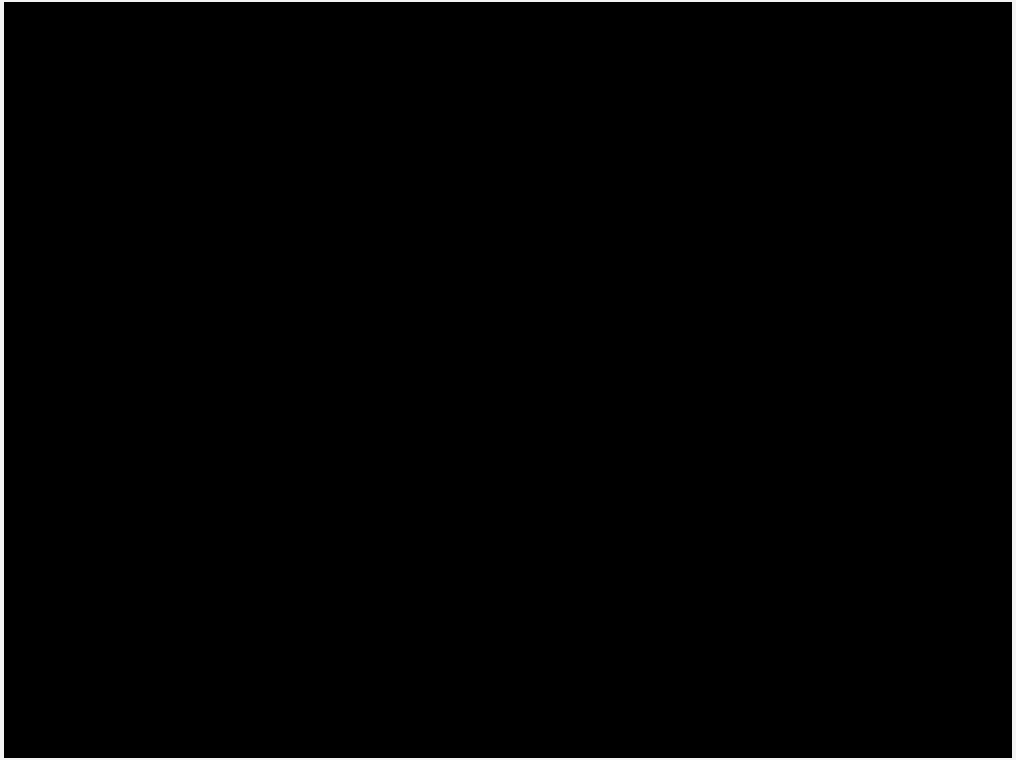
Интеграция и доработка Darknet

1. pydarknet
2. Поддержка OpenCV 4

Интеграция и доработка Darknet

1. pydarknet
2. Поддержка OpenCV 4
3. Устаревшие вызовы cudnn

После решения всех проблем с darknet FPS = 32



C++ удобней, потому что:

1. Легко интегрировать C библиотеку
2. Многопоточность
3. Контейнеры и алгоритмы
4. Профайлер

CASE#1: Модуль матчмейкинга для криптобиржи

Какие бывают типы заявок

1. *LIMIT* - когда выбираются предложения с ценой **ЛУЧШЕ** указанной лимитом

2. *MARKET* - когда выбираются предложения по лучшей **ДОСТУПНОЙ** цене

3. *STOP* - по достижению какого-то конкретного значения заявок на рынке такая заявка автоматически становится ***MARKET*** ордером, регистрируя минимальный убыток или прибыль трейдера (stop loss & take profit)

4. CANCEL - это вспомогательный тип для отмены заявки с определенным ID


КНИГА ЗАЯВОК BTC/USD
КОЛИЧЕСТВО СУММА ВСЕГО ЦЕНА ЦЕНА ВСЕГО СУММА КОЛИЧЕСТВО
2 5.7 5.7 9,273 9,274 3.3 3.3 4
1 3.1 8.9 9,271 9,275 36.0 32.6 7
5 7.0 15.9 9,267 9,276 37.1 1.1 4
3 0.0 16.0 9,266 9,278 37.1 0.0 1
3 24.3 40.3 9,265 9,279 43.3 6.1 4
1 9.8 50.1 9,264 9,280 47.0 3.7 4
1 0.5 50.6 9,263 9,281 49.2 2.2 2
5 3.7 54.4 9,262 9,282 49.3 0.0 2
2 11.1 65.5 9,261 9,283 49.9 0.5 1
4 2.1 67.7 9,260 9,284 49.9 0.0 1
2 2.2 69.9 9,258 9,285 50.8 0.9 3
4 0.6 70.5 9,257 9,286 55.1 4.2 2
5 12.8 83.3 9,256 9,287 59.6 4.5 4
3 3.6 87.0 9,255 9,288 60.1 0.5 2
3 13.2 100.2 9,254 9,289 103.9 43.7 6
5 9.7 110.0 9,253 9,290 112.9 8.9 5
4 33.1 143.1 9,252 9,291 125.6 12.6 5
8 2.8 146.0 9,250 9,292 126.2 0.6 3
1 0.5 146.5 9,249 9,293 127.8 1.5 2
1 11.3 157.9 9,248 9,294 130.0 2.1 3
3 3.3 161.2 9,247 9,295 145.7 15.7 8
2 5.1 166.3 9,246 9,296 167.2 21.4 4
3 13.9 180.2 9,245 9,298 167.8 0.6 3
1 10.0 190.2 9,244 9,299 168.7 0.8 5
Лучшая цена Bid
Лучшая цена Ask
FULL BOOK
В РЕАЛЬНОМ ВРЕМЕНИ

Структура заявок и очередей

ID: 11245
TS: 163611147
Price: 3000.5
N: 4

ID: 11247
TS: 163611148
Price: 3000.4
N: 7

ID: 11249
TS: 163611149
Price: 3000.3
N: 12

ID: 11251
TS: 163611150
Price: 3000.2
N: 1

Очередь заявок на продажу активов

ID: 11246
TS: 163611151
Price: 3000.1
N: 10

ID: 11248
TS: 163611152
Price: 3000.0
N: 2

ID: 11250
TS: 163611153
Price: 2999.9
N: 14

ID: 11252
TS: 163611154
Price: 2999.8
N: 7

Очередь заявок на покупку активов

Пришла новая LIMIT заявка

Совпавшая LIMIT заявка

Пришла новая MARKET заявка

Совпавший MARKET ордер

Состояние очередей после обработки MARKET заявки

ID: 11245
TS: 163611147
Price: 3000.5
N: 4

ID: 11247
TS: 163611148
Price: 3000.4
N: 7

ID: 11249
TS: 163611149
Price: 3000.3
N: 3

Очередь заявок на продажу активов

ID: 11246
TS: 163611151
Price: 3000.1
N: 10

ID: 11248
TS: 163611152
Price: 3000.0
N: 2

ID: 11250
TS: 163611153
Price: 2999.9
N: 14

ID: 11252
TS: 163611154
Price: 2999.8
N: 7

Очередь заявок на покупку активов

Требования к контейнерам

1. Быстрая вставка в произвольное место
2. Быстрое удаление
3. Быстрый поиск
4. Элементы отсортированы

Что использовали (~80K запросов в секунду)

1. *SKIP LIST* - для реализации очереди
2. *go channels* - для синхронизации
3. *sync.Pool* - для повторного использования памяти

Что использовали (~80K запросов в секунду)

1. *gRPC* - для получения заявок от биржи
2. *NATS* - для отправки сделок
3. *Prometheus* - для мониторинга

Проблемы большой математики в golang

В Go для поддержки больших чисел есть пакет math/big

```
111  func TestBigMath(t *testing.T) {
112      a, ok := new(big.Int).SetString("99999999999999999999999999999999999999999999999999999999999999999999999999999", 10)
113      require.True(t, ok)
114      b, ok := new(big.Int).SetString("3334", 10)
115      require.True(t, ok)
116      exp, ok := new(big.Int).SetString("333399999999999999999999999999999999999999999999999999999999999999999999996666", 10)
117      require.True(t, ok)
118      c := new(big.Int).Mul(a, b)
119      assert.Equal(t, 0, c.Cmp(exp))
120  }
```

В C++ чего только нет :) Gmp, boost.multiprecision, NTL, CLN

```
1  #include <gmpxx.h>
2
3  TEST(config, bigmath_example) {
4      mpz_class a("99999999999999999999999999999999999999999999999999999999999999999999999999999");
5      mpz_class b("3334");
6      mpz_class exp("333399999999999999999999999999999999999999999999999999999999999999999999999996666");
7      mpz_class c = a * b;
8      ASSERT_TRUE(exp == c);
9  }
```

Golang math/big

```
111  func TestBigMath(t *testing.T) {
112      a, ok := new(big.Int).SetString("99999999999999999999999999999999999999999999999999999999999999999999999999999", 10)
113      require.True(t, ok)
114      b, ok := new(big.Int).SetString("3334", 10)
115      require.True(t, ok)
116      exp, ok := new(big.Int).SetString("333399999999999999999999999999999999999999999999999999999999999999999999996666", 10)
117      require.True(t, ok)
118      c := new(big.Int).Mul(a, b)
119      assert.Equal(t, 0, c.Cmp(exp))
120  }
```

C++ gmp

```
1  #include <gmpxx.h>
2
3  TEST(config, bigmath_example) {
4      mpz_class a("99999999999999999999999999999999999999999999999999999999999999999999999999999");
5      mpz_class b("3334");
6      mpz_class exp("333399999999999999999999999999999999999999999999999999999999999999999999996666");
7      mpz_class c = a * b;
8      ASSERT_TRUE(exp == c);
9  }
```

Более сложные формулы с math/big перестают быть читаемыми

```
takerFeeCalculator: func(mm *Matchmaker, orderedVol, matchedVol, price *big.Int) *big.Int {
    if matchedVol.Cmp(orderedVol) == 0 {
        tmpr := new(big.Rat).SetFrac(orderedVol, bigInt1)
        VqftNum := new(big.Rat).Mul(mm.takerFeeCoeff, tmpr)
        VqftDenom := new(big.Rat).Add(mm.takerFeeCoeff, bigRat1)
        VqftRat := VqftNum.Quo(VqftNum, VqftDenom)
        Vqft := roundBigRat2Int(VqftRat)
        return Vqft
    }

    VqtRat := new(big.Rat).SetFrac(matchedVol, bigInt1)
    VqtRat = VqtRat.Mul(VqtRat, mm.takerFeeCoeff)
    Vqft := roundBigRat2Int(VqtRat)
    return Vqft
},
```

$$\frac{val \cdot coeff}{1 + coeff}$$

Более сложные формулы с gmp немного веселее

```
12  mpz_class VolumesFeesCalculator::sell_quote_taker_fee(const VolumesFeesCalculator *obj,
13                                                        const mpz_class &ordered_vol,
14                                                        const mpz_class &matched_vol,
15                                                        const mpz_class &price) {
16    UNUSED(price);
17    if (matched_vol == ordered_vol) {
18      mpq_class Vqft = obj->m_taker_fcoeff * ordered_vol;
19      Vqft.canonicalize();
20      Vqft /= obj->m_one_plus_ft;
21      return mpq_2_mpz(Vqft);
22    }
23    mpq_class Vqft = matched_vol * obj->m_taker_fcoeff;
24    return mpq_2_mpz(Vqft);
25  }
```

$$\frac{val \cdot coeff}{1 + coeff}$$

Что использовал в версии на C++ (~140K запросов в секунду против ~80K запросов)

1. Для заявок - первая версия использовала просто *std::set*, затем *boost::multi_index_container*
2. *GMP* - для больших чисел
3. *аллокаторы* - сначала small object allocator (как в loki), затем tcmalloc
4. *memory alignment* - структуру заголовка заявки переделал так, чтоб помещался в кэш линию
5. Трюк с холодным/горячим кэшем и подсказки по branch prediction в коде.

Что использовал в версии на C++ (~140K запросов в секунду против ~80K запросов)

1. *gRPC* - для получения заявок от биржи
2. *NATS* - для отправки готовых сделок в модуль регистрации сделок
3. *Prometheus* - для мониторинга

Преимущества C++:

1. GMP ощутимо быстрее
2. STL и Boost
3. Субъективно более красивый код
4. Производительность

Links

1. https://github.com/pjreddie/darknet - neural network framework written in C and CUDA
2. https://github.com/Lezh1k/darknet - same but can work with opencv >= 4.
3. https://pjreddie.com/darknet/yolo/ - real-time object detection system
4. https://www.youtube.com/channel/UCHG0_GSE0q02ywTfSoq2s3g - Japan Potato's YouTube Channel. Demonstration video is taken from this channel
5. https://www.investor.gov/introduction-investing/general-resources/news-alerts/alerts-bulletins/investor-bulletins-14 - common information about trading orders
6. https://gmplib.org/ - library for arbitrary precision arithmetic
7. https://www.boost.org/doc/libs/1_46_1/libs/multi_index/doc/index.html - boost.multiindex collection

Вопросы :)