РЖД лицей №8

Использование технического зрения для сортировки объектов роботом

Авторы: Д. А. Ручкина

2024 г.

Содержание

Введение

Техническое зрение – технология обработки изображений, получаемых с помощью цифровых видеокамер с целью обнаружения на них различных объектов (движущихся или неподвижных). Техническое зрение широко применяется в системах безопасности, для распознавания лиц, для контроля протекания технологических процессов и т.д.

Термин техническое зрение близок по значению к компьютерному зрению и, часто, может рассматриваться как его синоним. Однако компьютерное зрение чаще всего подразумевает обработку изображения с помощью мощных нейросетей, и является инструментом искусственного интеллекта. Эта технология в данный момент стремительно развивается, но в производственных условиях существенного применения пока что не получила. Это связано с тем, что современный искусственный интеллект все еще часто ошибается, в то время как на производстве важна точность, поэтому для производственных условий лучше подойдут надежные алгоритмы, обеспечивающие постоянство полученных данных.

Техническое зрение также получило распространение в системах безопасности. Здесь эта технология позволяет получать самую разную информацию о потенциально опасных объектах и ситуациях. Реагирование на изменения происходит максимально быстро, одновременно позволяя исключить оператора из процесса наблюдения и полностью исключить человеческий фактор.

Особую роль техническое зрение имеет в робототехнике. Здесь оно, прежде всего, позволяет определять координаты объекта манипулирования. Одной из основных сложностей в программировании промышленных роботов является задание координат объекта манипулирования. Обычно эти координаты приходится определять в ручном режиме, подводя схват манипулятора непосредственно к объекту.

Определив, таким образом, координаты далее программируют все необходимые перемещения. Очевидны недостатки этого подхода.

1. Подобное «жесткое» программирование основано на том, что в начале технологического процесса, перед каждой итерацией, объекты манипулирования будут находиться в одном и том же месте. Однако координаты объекта могут измениться под действием случайных факторов, например вибраций. Если координаты объекта будут отличаться от заданных в программе, то и эффективный захват будет невозможен.
2. Процесс программирования становится весьма трудоемким, особенно если объектов манипулирования много. Придется задавать координаты каждого из них.
3. При таком подходе возможно возникновение накопленной погрешности. Кинематические узлы промышленного робота имеют зазоры и люфты. Выполняя

раз за разом одни и те же действия погрешность перемещения узлов будет накапливаться, что может привести к смещению захвата относительно истинных координат.

Применение технического зрения позволит устранить все перечисленные недостатки. Естественно, что процесс программирования будет довольно сложным. Однако, подготовив соответствующую программу, в дальнейшем все действия значительно упростятся.

Одним из наиболее распространенных средств технического зрения является библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом OpenCV. В работе использовалась реализация этой библиотеки на языке программирования Python.

С точки зрения информатики техническое зрение находится близко к искусственному интеллекту и требует значительной вычислительной мощности. Также следует отметить, что очень часто эта технология во многом копирует процессы распознавания, свойственные человеку.

Например, для человека одним из отличительных признаков интересующего объекта является его цвет. Очень часто мы пытаемся разыскать объекты в поле зрения именно по их цвету и, во вторую очередь, по форме. В данной работе мы стремились использовать именно эту особенность.

1. БИБЛИОТЕКА КОМПЬЮТЕРНОГО ЗРЕНИЯ OpenCV

OpenCV (Open Computer Vision) – библиотека компьютерного зрения с открытым исходным кодом. Библиотека включает в себя несколько сотен алгоритмов компьютерного зрения. Имеет модульную структуру, то есть включает в себя несколько общих или статических библиотек.

В основном OpenCV разрабатывалась российской компанией Itseez (ранее известной как Аргус). Библиотека является кроссплатформенной и лицензируется как бесплатное программное обеспечение с открытым исходным кодом по лицензии Apache 2. Данная лицензия даёт пользователю право использовать программное обеспечение для любых целей и свободно изменять и распространять изменённые копии.

Рис. 1. Логотип OpenCV

OpenCV может использоваться в самых разных областях:

- распознавания лиц и жестов;
- взаимодействие человека и компьютера;
- робототехника;
- обнаружение объектов;
- сегментация и распознавание;
- стереоскопия стереовидение;
- дополненная реальность.

Также широкое распространение данная библиотека получила при использовании нейронных сетей.

В состав OpenCV входят следующие модули:

Core Functionality (основная функциональность) – ядро библиотеки, определяет структуры данных и функции, используется в других модулях.

Image Processing (обработка изображений) – обработка статичных изображений в форматах PNG, JPG и др.

Video Analysis (анализ видео) – используется для обработки потока данных с видеокамеры.

Camera Calibration and 3D Reconstruction (калибровка камеры и 3D-реконструкция) – позволяет создавать трехмерные модели объектов на основе нескольких изображений или видео.

2D Features Framework (фреймворк двумерных особенностей) – позволяет выполнять контурный анализ и осуществлять, таким образом, поиск объектов.

Object Detection (обнаружение объектов) – позволяет находить объекты, например лица, автомобили, птиц и другое.

High-level GUI (высокоуровневый графический интерфейс) – позволяет создавать графические интерфейсы.

Video I/O (ввод и вывод видео) – позволяет считывать и обрабатывать видеофайлы.

Полная документация на OpenCV занимает около 1100 страниц текста, таким образом, получается очень большой набор инструментов, открывающий широкие возможности для работы с изображением.

Изначально OpenCV написана на языке программирования C++, но легко поддерживает привязку для Python, Java и MATLAB. Собственно наиболее распространенным средством разработки программ для OpenCV стал именно Python.

Очень часто OpenCV используется в сочетании с библиотекой NumPy.

NumPy (Numerical Python) – также библиотека с открытым исходным кодом для Python. Является очень мощным инструментом для обработки больших массивов данных и позволяет значительно ускорить эту обработку.

NumPy обеспечивает обработку многомерных массивов и поддержку высокоуровневых математических функций, предназначенных для работы с многомерными массивами.

По существу любое изображение является большим массивом данных. Любое видео можно рассматривать как последовательность кадров, то есть изображений, быстро сменяющих друг друга. То есть видео – это некий поток массивов.

Каждый элемент этих массивов – пиксель, содержит либо одно число (если изображение черно-белое) либо три числа (если изображение цветное). Большие размеры массивов, рассматриваемых в данной работе, обуславливают применение данной библиотеки.

Рис. 2. Логотип NumPy

Однако даже применение NumPy не обеспечивает требуемой скорости обработки. Чтобы «облегчить» массив чаще всего прибегают к бинаризации изображения.

Процесс бинаризации – это перевод цветного изображения или изображения в оттенках серого в двухцветное черно-белое. Основной параметр такого преобразования – порог. Порог – это значение, с которым сравнивается яркость каждого пикселя. По результатам пикселю присваивается значение 0 или 1.

Рис. 3. Пример бинаризованного изображения

Цель бинаризации – уменьшить размер массива и, таким образом, ускорить его обработку. Правильно выполнить бинаризацию – не простая задача. В случае неудачи изображение может быть искажено и это только ухудшит ситуацию. Бинризация позволяет дать максимально точный ответ на вопрос где находится интересующий нас объект.

Следующий шаг – устранение помех. Любое бинаризованное изображение содержит множество дефектов. Это может быть эффект «снега» или «рваные края». Помехи затрудняют обработку, замедляют ее. При обработке видео помехи постоянно появляются и исчезают, заставляя систему реагировать на них. Существует множество способов минимизировать помехи, но устранить их полностью увы невозможно.

После устранения помех следует определить координаты объекта. Эта задача также может быть решена с помощью OpenCV, однако следует иметь в виду, что полученные координаты будут выражены в пикселях, а на производстве чаще всего используются миллиметры. Кроме того системы координат камеры и производственной установки также не совпадают.

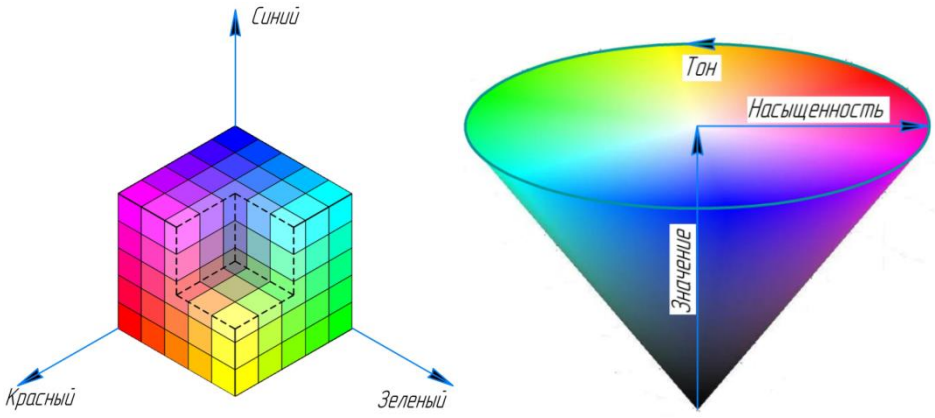
Все это приводит к необходимости корректировки координат и расчета коэффициента масштаба.

2. ПОИСК ОБЪЕКТОВ ПО ЦВЕТУ

OpenCV позволяет использовать несколько цветовых моделей (цветовых пространств) [1].

1. BGR (RGB) – используется по умолчанию и является стандартной цветовой моделью, основанна на смешении трех основных цветов: красного (red), зеленого (green) и голубого (blue).
2. HSV – цветовая модель, основанная на сочетании трех характеристик: оттенка (hue), насыщенности (saturation) и интенсивности (value) цвета. Часто отмечают, что эта модель наиболее близка человеческому глазу. Оценки цветовых оттенков как смеси трех цветов, которые дают люди, очень субъективны. Нам гораздо удобнее использовать такие понятия как «блеклый – насыщенный» или «светлый – темный». Поэтому часто эта модель является предпочтительной.
3. XYZ – модель, разработанная в лаборатории CIE с целью определения пределов чувствительности глаза человека к цвету.
4. GRAY – модель, преобразующая цветное изображение в черно-белое. В некоторых случаях эта модель используется для уменьшения «веса» обрабатываемого изображения.
5. YUV – цветовая модель, используемая на телевидении Y – яркость, а UV – два цветоразностных сигнала.

Наиболее часто в техническом зрении используют две цветовые модели: RGB и HSV (рис. 4) [1].



а) б)

Рис. 4. Цветовые модели BGR а) и HSV б)

Ниже приведен код простой программы, предназначенной для поиска объектов различных цветов, использующий цветовую модель HSV.

```
# Импорт библиотек OpenCV и Numpy
import numpy as np
import cv2

# Получение изображения с камеры
capImg = cv2.VideoCapture(1)

def find_color(low, high, name):
    ''' Функция поиска объекта заданного цвета.
        low - список, содержащий нижние значения цветового
        интервала в формате HSV;
        high - список, содержащий верхние значения цветового
        интервала в формате HSV;
        name - название цвета на англ.'''
    # Преобразование границ цветового интервала в массив Numpy
    low_color = np.array(low, dtype = "uint8")
    high_color = np.array(high, dtype = "uint8")
    # Создание цветовой маски
    mask = cv2.inRange(frame_hsv, low_color, high_color)
    # Поиск контуров
    contours, hierarchy = cv2.findContours(mask,
                                cv2.RETR_LIST,
                                cv2.CHAIN_APPROX_SIMPLE)
    # Обработка найденных контуров
    for icontours in contours:
        rect = cv2.minAreaRect(icontours)
        area = int(rect[1][0]*rect[1][1])
        # Фильтрация контуров
        if area > 1400:
            box = cv2.boxPoints(rect)
            box = np.int0(box)
            # Отрисовка контуров
            cv2.drawContours(frame, [box], -1, (0, 255, 0), 1)
            # Определение координат центра
            center = (int(rect[0][0]), int(rect[0][1]))
            # Отметка в центре конутра
```

```
            cv2.circle(frame, center, 5, (0, 255, 0), 2)
            # Нанесение надписи
            cv2.putText(frame, name, (center[0]+25, center[1]),
                    cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)

# Обработка изображения с камеры
while True:
    ret, frame = capImg.read()
    frame_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Поиск объектов синего цвета
    find_color([105, 150, 50], [135, 255, 210], "Blue")
    # Поиск объектов зеленого цвета
    find_color([50, 50, 100], [90, 255, 210], "Green")
    # Поиск объектов желтого цвета
    find_color([25, 10, 160], [40, 180, 240], "Yellow")
    # Поиск объектов красного цвета 1
    find_color([0, 80, 150], [15, 255, 210], "Red")
    # Поиск объектов красного цвета 2
    find_color([165, 80, 150], [180, 255, 210], "Red")
    # Вывод найденных объектов
    cv2.imshow("Video signal", frame)
    # Условие перывания цикла
    key_press = cv2.waitKey(30)
    if key_press == ord("q"):
        break
# Очистка памяти
capImg.release()
# Закрытие всех окон
cv2.destroyAllWindows()
```

Результат работы программы представлен на рис. 5.

Рис. 5 Результат поиска объектов по цвету

Хорошо видно, что программа уверенно распознает объекты различного цвета.

3. ИСПОЛЬЗОВАНИЕ РАЗМЫТИЯ

Одной из проблем при использовании технического зрения является возникновение различных помех. Например «эффект снега», при котором на бинаризованном изображении появляются и исчезают отдельные пиксели или группы пикселей белого цвета. Другим примером являются «рваные края» обнаруженных объектов. «Эффект снега» поглощает ресурсы системы и снижает производительность, «рваные края» объектов затрудняют их обнаружение и определение центра найденного контура. Наиболее простым способом устранения этих помех является размытие изображения [2], [3].

Использование размытия в техническом зрении обеспечивает следующие преимущества:

1. устранение помех в виде эффекта снега, а также других артефактов (например ауры вокруг границ объектов);
2. устранение рваных краев изображений найденных объектов;
3. некоторое ускорение обработки изображений за счет уменьшения количества мелких деталей.

В OpenCV используются следующие способы размытия:

1. простое сглаживание (cv2.filter2D);
2. гауссово размытие (cv2.GaussianBlur);
3. медианное размытие (cv2.medianBlur);
4. двустороннее размытие (cv2.bilateralFilter).

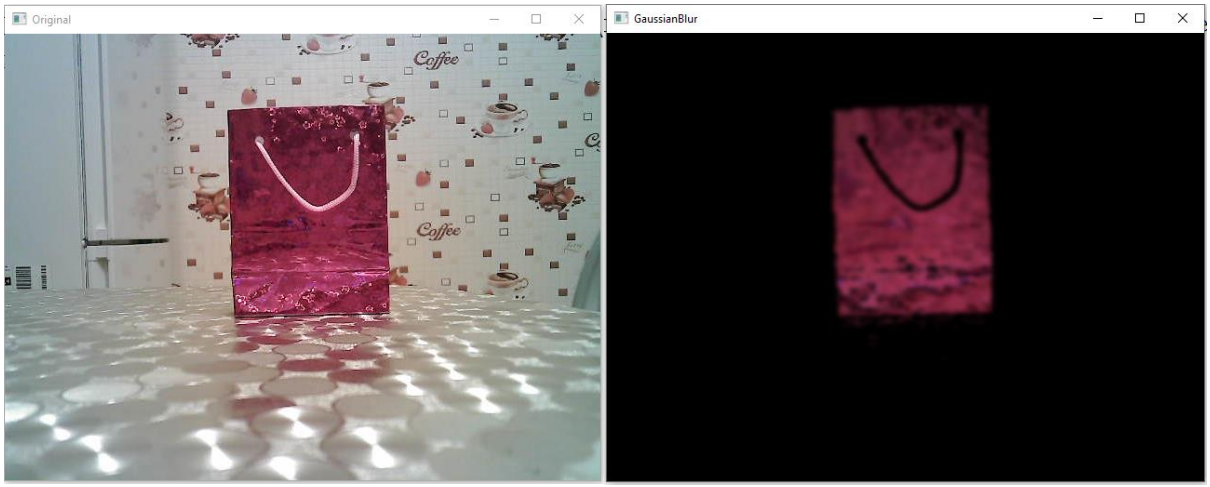
Ниже представлен код небольшой программы, позволяющей сравнить результаты выполнения всех четырех алгоритмов размытия.

```
# Импорт библиотек OpenCV и Numpy
import cv2
import numpy as np
# Получение изображения с камеры
cap = cv2.VideoCapture(1)

# Создание бесконечного цикла для чтения изображения
while True:
   ret, frame = cap.read()
   # Замена цветовой модели RGB на HSV
   hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

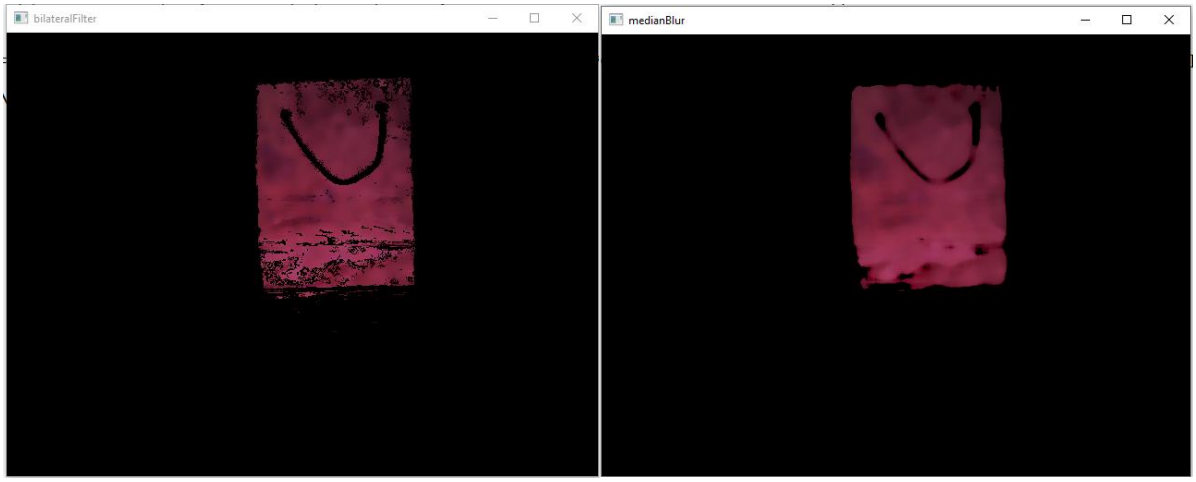
```
    # Задание границ цвета (анализируем объект красного цвета)
    lower_red = np.array([30,150,50])
    upper_red = np.array([255,255,180])
    # Создание цветовой маски
    mask = cv2.inRange(hsv, lower_red, upper_red)
    # Бинаризация
    res = cv2.bitwise_and(frame, frame, mask=mask)
    kernel = np.ones((15, 15), np.float32)/225
    # Применение к изображению простого сглаживания
    smoothed = cv2.filter2D(res, -1, kernel)
    # Применение к изображению медианного размытия
    median = cv2.medianBlur(res, 15)
    # Применение к изображению двустороннего размытия
    bilateral = cv2.bilateralFilter(res, 15, 75, 75)
    # Применение к изображению Гауссова размытия
    blur = cv2.GaussianBlur(res, (15, 15), 0)
    # Вывод полученных изображений
    cv2.imshow("bilateralFilter", bilateral)
    cv2.imshow("medianBlur", median)
    cv2.imshow("Original", frame)
    cv2.imshow("GaussianBlur", blur)
    cv2.imshow("Averaging", smoothed)
    # Условие прерывания цикла при нажатии клавиши Esc
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break
# Закрыть все окна и очистить память
cv2.destroyAllWindows()
cap.release()
```

Результаты работы программы представлены на рисунках 6, 7 и 8.



а) б)

Рис. 6. Не обработанное изображение а); изображение, полученное с помощью Гауссова размытия б)



а) б)

Рис. 7. Изображение, полученное с помощью двустороннего размытия а); изображение, полученное с помощью медианного размытия б)

Рис. 8. Изображение, полученное с помощью простого сглаживания

Для использования всех видов размытия применялись одни и те же начальные характеристики. Хорошо видно, что наилучшие результаты обеспечивает гауссово размытие. Этот способ позволяет максимально устранить помехи «снег» и «рваные края» (рис. 9 и 10).

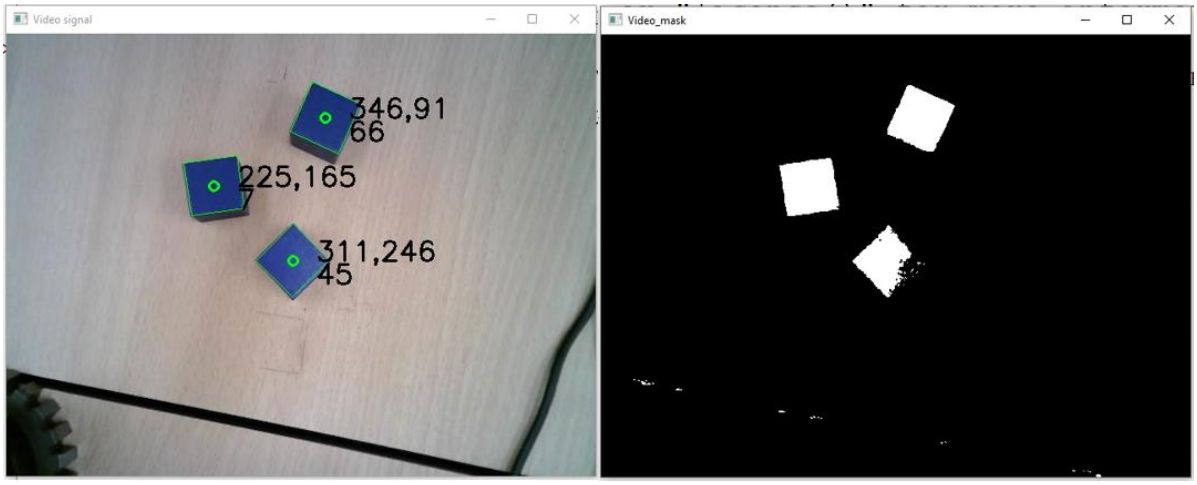


Рис. 9. Результат поиска объектов синего цвета без использования размытия

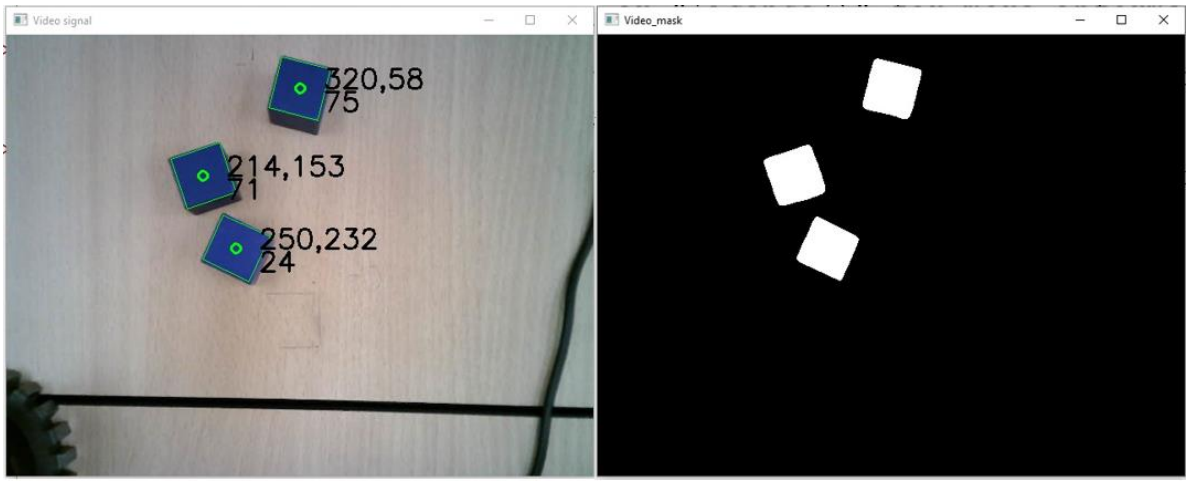


Рис. 10. Результат поиска объектов синего цвета с использованием размытия

Таким образом правильнее будет сделать выбор именно для размытия по методу Гаусса.

4. ТЕХНИЧЕСКАЯ РЕАЛИЗАЦИЯ

В своем проекте мы использовали робот платформы Dobot (рис. 11). Роботы Dobot достаточно просты в освоении и позволяют изучить самые различные области применения робототехники от перестановки объектов до лазерного гравирования. Роботы комплектуются широким набором рабочих элементов и различными устройствами, расширяющими его технологические возможности.

Рис. 11. Робот платформы Dobot

Роботы имеют свою собственную среду программирования Dobot Magician, но также программирование может осуществляться с помощью Python, причем непосредственно как из Dobot Magician так и с помощью любой среды программирования, в которой можно работать с Python. Специально для программирования роботов Dobot была разработана специальная библиотека DobotDllType.

Непосредственное использование Python позволяет существенно расширить технологические возможности робота, например, применить техническое зрение [4].

Также в проекте использовалась веб-камера A4TECH модель PK-910H (рис. 12).

Рис. 12. Веб-камера A4TECH модели PK-910H на магнитной стойке

Характеристики камеры представлены в таблице 1.

Таблица 1. Характеристики камеры PK-910H

Тип матрицы	cmos
Интерфейс	usb2.0
Тип поставки	ret
Запись видео высокой четкости	1080p
Тип установки (крепления)	универсальный
Длина кабеля	1,5 м
Модель	PK-910H
Размеры, мм	50x55x60
Разрешение видео, пик.	1920x1080 пикселей
Разрешение матрицы, пик.	2 Мп
Скорость записи видео, до	30 кад/сек
Угол вращения web-камеры по вертикали	60°
Угол вращения web-камеры по горизонтали	360°
Угол обзора	70°
Максимальная частота кадров	30 Гц
Подключение	USB 2
Фокусировка	автоматическая
Масса, кг	0,2

Данная камера не обладает высокими характеристиками, однако отличается достаточной надежностью и компактностью.

В качестве основания для камеры применена магнитная стойка Туламаш 101144 (рис. 13).

Рис. 13. Магнитная стойка Туламаш 101144

Данная стойка оснащена достаточно мощным магнитом, позволяющим закрепить ее к любой магнитной поверхности. Сама стойка состоит из множества сочленений со специальным стальным тросом внутри. Это позволяет надежно закрепить камеру под любым углом.

5. СИСТЕМЫ КООРДИНАТ

Одна из проблем при использовании компьютерного зрения состоит в несовпадении систем координат и единиц измерения (рис. 14)

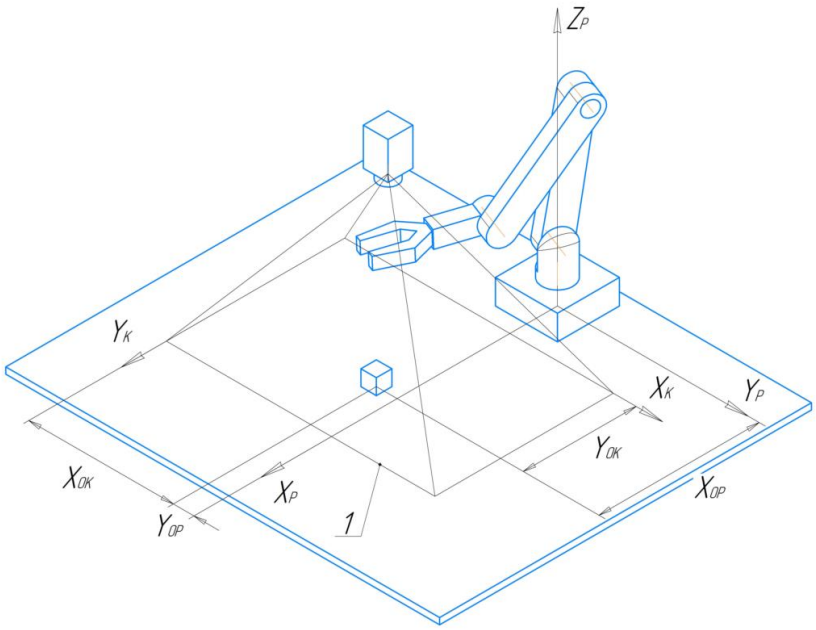


Рис. 14. Несовпадение систем координат робота и камеры

Начало системы координат робота (XpYpZp) находится в середине его основания. Начало системы координат камеры (XкYк) совпадает с левым верхним пикселем кадра. Таким образом, у обеих систем координат не совпадают ни начала, ни направления осей.

Введем следующие обозначения:

ΔX – расстояние между началом системы координат робота и началом системы координат камеры, измеренное вдоль оси X робота и выраженное в миллиметрах;

ΔY – аналогичное расстояние, но вдоль оси Y робота;

Scale – масштаб.

Эти особенности необходимо учесть при подготовке программы. Итоговая программа должна содержать выражения, присчитывающие координаты камеры в координаты робота.

Ниже представлена часть итоговой программы, в которой выполняются необходимые операции.

```
# Преобразование координат и запись их в список coordinates_list
'''Вводить координаты сюда'''
imshow("Video signal", frame)
coordinates_list.append(int(int(rect[0][1])*0.49+151.6))
```

```
coordinates_list.append(int(int(rect[0][0])*0.49-104))
```

В данном случае 0,49 – масштаб, который переводит координаты в пикселях в миллиметры. Числа 151,6 и 104 – разница в положении начал координат.

Эти данные придется установить до начала отработки программы и внести в итоговую программу вручную. Эти действия связаны с общей наладкой всей системы.

Ниже представлена программа для нахождения значений ΔX, ΔY, Scale.

```
capImg = cv2.VideoCapture(1)
# Список для записи координат и длин сторон
coordinates = []
while True:
   # Получение изображения с камеры
   ret, frame = capImg.read()
   # Изменение цветовой модели
   frame_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
   # Вывод изображения на экран
   cv2.imshow("Video signal", frame)
   # Задание границ цветового диапазона для поиска объектов синего цвета
   low_Blue = np.array([105, 150, 0], dtype = "uint8")
   high_Blue = np.array([135, 255, 210], dtype = "uint8")
   blue_mask = cv2.inRange(frame_hsv, low_Blue, high_Blue)
   # Определение контуров объектов
   contours, hierarchy = cv2.findContours(blue_mask,
                                          cv2.RETR_LIST,
                                          cv2.CHAIN_APPROX_SIMPLE)
   # Обработка контуров
   for icontours in contours:
      rect = cv2.minAreaRect(icontours)
      # Вычисление площади контура
      area = int(rect[1][0]*rect[1][1])
      # Фильтрация контуров
      if area > 1500:
         box = cv2.boxPoints(rect)
         box = np.int0(box)
         # Отрисовка найденного контура
```

```
            cv2.drawContours(frame, [box], -1, (0, 255, 0), 1)
            # Определение координат центра и нанесение отметки центра на изобрежние
            center = (int(rect[0][0]), int(rect[0][1]))
            cv2.circle(frame, center, 5, (0, 255, 0), 1)
            # Вывод координат на изображение
            cv2.putText(frame, "%d,%d"%(rect[0][0], rect[0][1]), (center[0]+25, center[1]),
                        cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),1)
            # Занесение координат в список
            coordinates.append(int(rect[0][1]))
            coordinates.append(int(rect[0][0]))
            coordinates.append(int(rect[1][0]))
            # Вывод базового изображения и цветовой маски
            cv2.imshow("Video signal", frame)
            cv2.imshow("Video_mask", blue_mask)
        # Условие прерывания
        # Прерывание происходит очень быстро, как только заполнится список
        if len(coordinates) > 0:
            break

            print(int(rect[1][0]))
    # Вычисление данных для определения координат
    scale = 24/coordinates[2]
    delta_x = - coordinates[0]*scale + robot_x
    delta_y = coordinates[1]*scale - robot_y
    # Вывод необходимых значений на печать
    print("scale= ", scale)
    print("delta_x= ", delta_x)
    print("delta_y= ", delta_y)
    # Очистка памяти, закрытие всех окон
    capImg.release()
    cv2.destroyAllWindows()
```

Алгоритм позволяет вычислить необходимые характеристики ΔX, ΔY и Scale, и ввести их в дальнейшем в программу.

6. ИТОГОВАЯ ПРОГРАММА

Ниже представлена итоговая программа с необходимыми пояснениями.

```
# Подключение необходимых бибилотек и функций
from numpy import array, int0
from cv2 import VideoCapture, cvtColor, imshow, inRange, findContours, minAreaRect, boxPoints, drawContours, putText, norm, waitKey, RETR_LIST, COLOR_BGR2HSV, CHAIN_APPROX_SIMPLE, FONT_HERSHEY_SIMPLEX
import DobotDllType as dType

# Все движения робота программируются в функции move_robot
def move_robot(x, y, a, y_base=-100, x_base=250):
  '''x, y - координаты объекта, a - угол поворота объекта,
  n - количество объектов, y_base - базовая координата по y'''
  #Выбор рабочего инструмента (присоска)
  dType.SetEndEffectorParamsEx(api, 59.7, 0, 0, 1)
  dType.SetPTPJumpParamsEx(api,30,100,1)
  #Включить присоску
  dType.SetEndEffectorSuctionCupEx(api, 1, 1)
  # Пауза
  dType.dSleep(500)
  # Перемещение к кубику
  dType.SetPTPCmdEx(api, 0, x, y, (-43.5), -a, 1)
  # Пауза
  dType.dSleep(500)
  # Перенос кубика
  dType.SetPTPCmdEx(api, 0, (x_base), (y_base), (-43.5), 0, 1)
  # Пауза
  dType.dSleep(500)
  # Выключение присоски
  dType.SetEndEffectorSuctionCupEx(api, 0, 1)
  # Пауза
  dType.dSleep(500)
  # Подъем
  current_pose = dType.GetPose(api)
  dType.SetPTPCmdEx(api, 2, 250, (-100), 40, current_pose[3], 1)
```

```
# Функция поиска объектов
def find_object(h_low, s_low, v_low, h_high, s_high, v_high):
    '''h_low, s_low, v_low - нижние значения цветового интервала;
    h_high, s_high, v_high - верхние значения цветового интервала'''
    # Список для записи координат
    coordinates_list = []
    # Переменная для посчета количества объектов
    number_contours = 0
    # Определение цветового синих объектов интервала
    low_Color = array([h_low, s_low, v_low], dtype = "uint8")
    high_Color = array([h_high, s_high, v_high], dtype = "uint8")
    # Создание цветовой маски
    mask = inRange(frame_hsv, low_Color, high_Color)
    # Поиск контуров объектов
    contours, hierarchy = findContours(mask, RETR_LIST,CHAIN_APPROX_SIMPLE)
    # Заключение каждого найденного контура в прямоугольник
    # минимального размера
    for icontours in contours:
        rect = minAreaRect(icontours)
        # Вычисление площади контура
        area = int(rect[1][0]*rect[1][1])
        # Фильтрация конутров
        if area > 1500:
            # Посчет количества контуров
            number_contours += 1
            #Вычисление координат вершин и отрисовка контура
            box = boxPoints(rect)
            box = int0(box)
            drawContours(frame, [box], -1, (255, 0, 0), 3)
            center = (int(rect[0][0]), int(rect[0][1]))
            # Нанесение на изображение текстовой информации
            # о координатах объекта
            putText(frame, "%d,%d"%(rect[0][0], rect[0][1]), (center[0]+10, center[1]),
                    FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
            # Определение угла поворота объекта
            angle = int(rect[2])
```

```
            # Нанесение на изображение текстовой информации
            # об угле поворота
            putText(frame, "%d"%int(angle),
                    (center[0]+25, center[1]+25),
                    FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
            # Преобразование координат и запись их в список coordinates_list
            '''Вводить координаты сюда'''
            imshow("Video signal", frame)
            coordinates_list.append(int(int(rect[0][1])*0.49+151.6))
            coordinates_list.append(int(int(rect[0][0])*0.49-104))
    # Условие, если объектов не найдено
    if number_contours == 0:
        return "Объектов не найдено"
    else:
        return number_contours, coordinates_list, angle

# Подключение робота
CON_STR = {
    dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
    dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
    dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}

api = dType.load()

state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:",CON_STR[state])

# Включение камеры
capImg = VideoCapture(1)

# Поиск объектов
while True:
    # Запись изображения с камеры в переменную frame
    ret, frame = capImg.read()
    # Выбор цветовой модели
    frame_hsv = cvtColor(frame, COLOR_BGR2HSV)
    # Вывод изображения с камеры
```

```
    imshow("Video signal", frame)
    # Поиск объектов
    '''Добавлять объекты сюда'''
    object_Blue = find_object(105, 150, 0, 135, 255, 210)
    object_Green = find_object(30, 50, 70, 90, 255, 210)
    # Условие прерывания цикла в случае окончания обработки всех контуров
    key_press = waitKey(30)
    if key_press == ord('q'):
        break

# Перемещений роботом синих объектов
if object_Blue == "Объектов не найдено":
    print("Синих объектов не найдено")
else:
    for i in range(object_Blue[0]):
        move_robot(object_Blue[1][0+2*i], object_Blue[1][1+2*i],
                object_Blue[2],-100, 230-29*i)

# Перемещений роботом зеленых объектов
if object_Green == "Объектов не найдено":
    print("Зеленых объектов не найдено")
else:
    for i in range(object_Green[0]):
        move_robot(object_Green[1][0+2*i], object_Green[1][1+2*i],
                object_Green[2], -150, 230-29*i)
```

На рисунке 15 представлен результат работы программы.

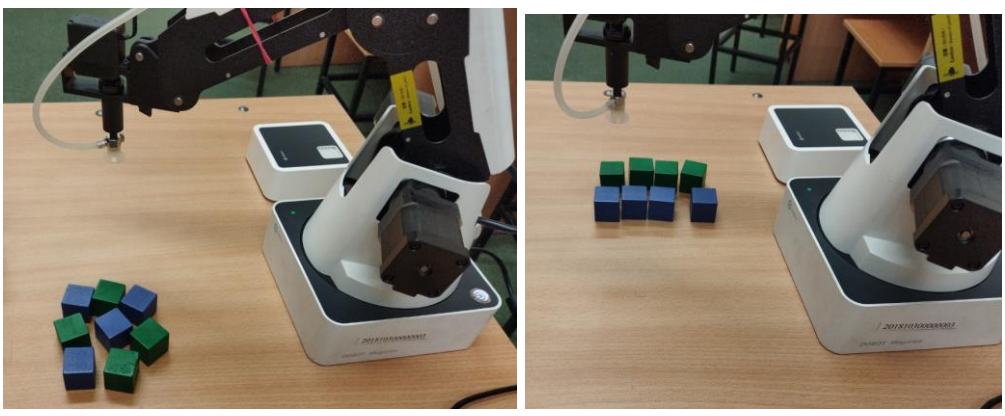


Рис. 15 Сортировка объектов роботом

Некоторые погрешности в расстановке объектов связаны с невысокой точностью перемещений робота. Роботы Dobot хорошо подходят для учебных целей, но выполнять весьма точные перемещения увы не могут.

Выводы

Программирование робота с использованием технического зрения открывает множество перспектив.

1. Программирование становиться более гибким. Если раньше координаты каждого тела приходилось задавать отдельно, то теперь робот сам «находит» тела и манипулирует ими.
2. Наилучшей цветовой моделью является модель HSV. Она наиболее близка к человеческому глазу и выполнять программирование с ней гораздо легче.
3. Применение гауссова размытия позволяет нейтрализовать помехи и повысить точность обнаружения координат.
4. Применение технического зрения требует некоторых навыков программирования, однако осилить и понять основные принципы способен человек даже без специальной подготовки.

Полученные результаты будут интересны для учебных заведений, готовящих специалистов в области программирования и управления, а также для предприятий, использующих робототехнические комплексы в производственном процессе. Это позволит освободить рабочих от выполнения рутинных операций, повысить производительность и точность операций.

Вместе с тем подобная работа требует навыков программирования, тщательной и точной настройки оборудования и умения решать возникающие проблемы. Этими проблемами могут быть, например, недостаточная освещенность, возникновение электронных помех в цепях управления роботом или камерой, погрешности в механических узлах робота.

Также следует отметить, что технология технического зрения соответствует принципам и концепциям «Индустрии 4.0», четвертной промышленной революции. Основным принципом которой, является автоматизация, цифровизация и широкое внедрение искусственного интеллекта.

Проделанная работа – лишь начало. В дальнейшем планируется реализовать иные алгоритмы распознавания, например не только по цвету, но и по форме. Кроме того, возможно с использованием нейронных сетей.

Библиографический список

1. Шакирьянов, Э. Д. Компьютерное зрение на Python. Первые шаги : учебное пособие / Э. Д. Шакирьянов. – Москва : Лаборатория знаний, 2021. – 163 с.
2. Селянкин, В. В. Компьютерное зрение. Анализ и обработка изображений / В. В. Селянкин. – 3-е изд., стер. – Санкт-Петербург : Лань, 2023. – 152 с.
3. Шапиро, Л. Компьютерное зрение : учебное пособие / Л. Шапиро, Д. Стокман ; под редакцией С. М. Соколова ; перевод с английского А. А. Богуславского. – 4-е изд. – Москва : Лаборатория знаний, 2020. – 763 с.
4. Ян, Э. С. Программирование компьютерного зрения на языке Python / Э. С. Ян ; перевод с английского А. А. Слинкин. – Москва : ДМК Пресс, 2016. – 312 с.