KIVISort - СИСТЕМА АДАПТИВНОЙ СОРТИРОВКИ ОТХОДОВ

Кириленко В.Д.

ГБПОУ «Воробьевы горы», Центр Технического Образования, Москва, Россия

Аннотация

В данном проекте представлена автономная система, позволяющая классифицировать различный мусор, а так же управлять манипулятором-сортировщиком мусора. Сортировка выполняется на основе материала, формы, либо конкретного класса объекта. Разработка сосредоточилась на адаптивности и ускорении обучения, что позволяет системе приспосабливаться к изменениям в отходах.

ОГЛАВЛЕНИЕ

1. Введение

Загрязнение окружающей среды бытовыми отходами ведет к нарушению экологического баланса на всей планете. Однако, почти любой мусор пригоден для переработки и повторного использования. Одна из главных проблем цикла переработки мусора заключается в том, как его рассортировать на фракции, которые можно использовать для вторичной переработки. Роботизация этого процесса позволит как сократить затраты на весь цикл переработки отходов, так и уменьшить процент ошибки при сортировке.

Современный уровень техники и программного обеспечения позволяет производить полную автоматизацию сортировки мусора. Одним из наиболее элегантных решений можно выделить использование компьютерного зрения для поиска и алгоритмов машинного обучения для классификации мусора. Оно и будет описано в данном тексте.

2. Цели и задачи

Цель проекта состоит в разработке аппаратно-программного комплекса, позволяющего в автономном режиме сортировать различные объекты в зависимости от их класса. Процесс работы над проектом можно подразделить на следующие задачи:

1. Создание системы компьютерного зрения (далее — СКЗ), позволяющей с помощью закреплённой над рабочей зоной камеры расчитывать параметры ограничивающего прямоугольника
2. Изучение различных методов классификации объектов, основанных на визуальном представлении, и последующая реализация аналогичного алгоритма в проекте
3. Разработка виртуальной среды, позволяющей воссоздавать алгоритм работы готового устройства
4. Разработка и создание собственного прототипа манипулятора, способного сортировать объекты массой до 1.5кг и шириной до 110мм для проверки работоспособности всех систем в реальных условия
5. Сделать выводы о целесообразности продолжения проекта.

3. Описание системы, принципов работы и используемых технологий

3.1 Система компьютерного зрения

Рассмотрим получившуюся СКЗ. Задача была разбита на две подзадачи:

- Устранение дефектов и искажений изображения, получившихся вследствие возникновения перспективы
- Создание бинарной маски для отделения рассматриваемых объектов от заднего фона.

Для решения первой проблемы было принято решение разместить четыре ArUco[1] маркера по углам рабочей области. Подобные метки отличаются простотой их поиска на изображении и минимально зависимы от таких внешних условий, как освещение, угол наклона относительно камеры и т.п., что делает их отличным выбором для данного проекта. Основная идея состоит в поиске всех 4-х меток и, как следствие, 4-х углов рабочей зоны манипулятора, что позволяет вычислить матрицу преобразования и использовать функцию коррекции перспективы, в результате чего мы получаем изображение, максимально приближенное к ортогональному виду сверху. Такой результат позволит легко преобразовывать координаты объекта в локальную декартову систему отсчёта рабочего поля.

После того, как мы получили наиболее удобный вид изображения, рассмотрим вторую подзадачу. Она была решена путём проведения предварительной калибровки среднего значения цвета, переводом изображения в цветовое пространство HSV[2] и применением инвертированного алгоритма поиска цветов в диапазоне, что на выходе даёт бинарную маску, соответствующую отклонению цвета пикселей от калибровочного стандарта. Подобный алгоритм отлично подходит для поиска даже прозрачных объектов, которые ввиду преломления света всё равно будут найдены. Далее, выделяем отдельные объекты на получившейся маске, используя вспомогательные морфологический операции для повышения точности. Выделеные объекты удобно вырезать из оригинального изображения для дальнейшей обработки и классификации.

3.2 Распознание образов

3.2.1 Постановка задачи

Распознание образов является классической задачей в области машинного зрения и нейронных сетей и имеет множество путей решений в зависимости от

задачи. В данном случае, так как проблема поиска объектов решена без применения алгоритмов машинного обучения, задача распознавания сводится к определению метки объекта на некотором входном изображении, классификации.

Глобально, для решения поставленной задачи используются несколько методов. Наиболее подходящими из них являются следующие:

- Обучение свёрточной нейронной сети (CNN[3]) для классификации
- Обучение глубинной нейронной сети (DNN) для классификации
- Использование одного из алгоритмов классификации или кластеризации (напр., метод *k*-ближайших соседей).

По результатам изучения имеющихся решений было выявлено следующее:

- Прямое обучение CNN не имеет смысла так, как они отлично подходят для случаев со множеством выраженных признаков(напр., классификация пород собак), но не работает для более сложных задач, как классификация материала
- Прямое применение глубоких нейронных (DNN) показывает значительно большую точность по сравнению со свёрточными, однако остаётся недостаточно точными в рамках проекта, а так же имеют слишком большое время обучения(см. прил 1), что критично так, как это время не должно превышать 10-и минут
- Прямое использование классификаторов/кластеризаторов имеет минимальное время обучения (в диапазоне от 0.02 до 1 мин.), однако имеет неприемлемо низкую точность (около 10%).

3.2.2 Разработанный метод

В процессе работы надо проектом был разработан метод, показывающий наилучшее время обучения и точность. Идея состоит в комбинации CNN ImageNet[4], позволяющей преобразовывать растровое RGB изображение в двумерный вектор признаков и одного из алгоритмов классификации/ кластеризации. Данный метод позволяет сократить время обучения до 1–2 мин. и увеличить точность распознавания до 93% при минимальных[1] требованиях к вычислительным мощностям. Подобная скорость обучения и точность классификации позволяют оперативно добавлять новые классы и/или расширять уже имеющиеся.

3.2.3 Выбор классификатора

Для выбора наиболее подходящего классификатора был проведён сравнительный анализ 5-и наиболее распространённых алгоритмов. В прил. 2.1

[1] CPU - Intel Core i5 2,4 GHz

приведены матрицы ошибок[2] для сравниваемых алгоритмов. Матрица ошибок[5] представляет собой матрицу M, которая рассчитывается по следующей формуле: $M = \{m_{ij}\}_{i,j=0}^{C}$, $m_{ij} = \sum_{k=0}^{N} \mathbb{I}[a(x_k) = j]\mathbb{I}[y_k = i]$ для некоторой выборки x_i ($i = 1,\ldots,N$, y_i - метка i-го объекта, $y_i \in \{1,2,\ldots,C\}$), каждый объект которой относится к одному из C классов и классификатор a, который эти классы предсказывает. Данная матрица показывает сколько объектов класса i были классифицированы как j. Так же в прил. 2.2 представлен график, позволяющий сравнить точность и время обучения изучаемых алгоритмов.

На основании полученных данный было принято решение о выборе алгоритма SVC[6] для классификации. SVC, или C-Support Vector Classification является разновидностью метода опорных векторов. Основная идея состоит в переводе исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Формально, алгоритм может быть описан следующем способом: точки имеют вид $\{(x_1, c_1), (x_2, c_2), \ldots, (x_n, c_n)\}$, где c_i является индексом класса, которому принадлежит точка x_i. Строим разделяющую гиперплоскость, которая имеет вид: $w \cdot x - b = 0$. Вектор w - перпендикуляр к разделяющей плоскости, $\frac{b}{\|w\|}$ - расстояние от гиперплоскости до начала координат. Далее, задача сводится к минимизации $\|w\|$ для формулы $c_i(w \cdot x_i - b) \geq 1,\ 1 \leq i \leq n$.

3.2.4 Реализация

Итоговая реализация *СКЗ* и классификатора выполнены на высокоуровневом языке Python с использованием следующих библиотек:

- OpenCV - для захвата видео с камеры, и всех обработок изображений в процессе работы
- Tensorflow 1.14 - для обработки ImageNet
- Scikit-learn - для обработки SVM
- OpenRV - библиотека собственной разработки для ускорения разработки.

Такой выбор обусловлен простотой использования и быстротой моделирования, что позволяет минимизировать сроки разработки. Далее, СКЗ и классификатор будут объединены под названием Система Зрительного Распознания Объектов (*СЗРО*).

[2] Тестирование проводилось на одном и том же оборудовании и обучающей выборке

3.3 Виртуальная модель

Рассмотрим систему моделирования (СМ). Она должна отвечать следующим требованиям:

- Возможность обмена данными между *СМ* и *СЗРО* через API
- Возможность масштабирования *СМ* до нескольких параллельно работающих устройств
- Поддержка различных кинематических систем манипулятора
- Возможность синхронизации *СМ* и реального манипулятора.

В процессе изучения готовых физических движков для создания *СМ*, были изучены следующие SDK:

1. Gazebo,
2. V-rep,
3. Webots,
4. MRS,
5. Unity 3D,
6. Unreal Engine.

Наилучшим вариантом из перечисленных является Gazebo, однако от него пришлось отказаться ввиду высокой сложности воссоздания линейного перемещения системы. По аналогичным причинам, для проекта не подходят Webots, v-rep и MRS. Таким образом, для реализации *СМ* был выбран движок Unity 3D так, как он хорошо документирован, позволяет без особых затрат изменять кинематику и имеет встроенную поддержку последовательного соединения для синхронизации с реальным устройством, в отличие от Unreal Engine. С пример работы получившейся *СМ* можно ознакомиться по ссылке: {}.

3.4 Манипулятор

Прототип манипулятора выполнен на безе часто используемой в промышленных манипуляторах кинематики SCARA[7] *(Selective Compliance Articulated Robot Arm)*. Рабочая область может быть грубо описана, как прямоугольный параллелепипед с основанием 450мм × 900мм и высотой 250мм. Выбор данной кинематики обусловлен её простотой реализации и надёжностью.

Манипулятор приводится в действие тремя шаговыми моторами Nema 21, что позволяет с высокой точностью контролировать движение всех узлов. Для манипуляций с сортируемыми объектами используется комбинация из пневматического захвата с параллельными губками, установленного на серво-машинке и пневмоприсоски, сменяющих друг друга в зависимости от класса объекта (напр., для манипуляции с бутылкой используется грейферный захват, а для пакета - присоска). Серво-машинка используется для точного позиционирования, позволяющего эффективно брать такие объекты, как

бананы, бутылки и т.п. С подробным алгоритмом вычисления необходимого угла поворота можно ознакомиться в прил. 3.

Низкоуровневое управление манипулятором реализовано на плате Arduino Mega 2560, для коммуникации с управляющим компьютером реализован API для общения с *C3PO* и/или *CM*.

---

4. Заключение

В результате проделанной работы была разработана автоматическая система для сортировки, способная распознавать множество классов объектов и управлять роботом-сортировщиком, а так же виртуальная среда, позволяющая проводить тестирование работоспособности программных модулей, в том числе, *C3PO*. Использование адаптивного обучения позволяет без длительной остановки работы увеличить число классов распознаваемых объектов, что позволяет перенастроить всю систему в соответствие с конкретными требованиями.

Было принято решение продолжать развивать проект в следующих направлениях:

- Уменьшение процента ошибки при классификации объектов, путём сотрудничества с мусоросортировочным комплексом с целью сбора достаточной обучающей выборки
- Повышение точности локализации объектов
- Доработка манипулятора, путём увеличения точности перемещений и снижения ограничений на объекты (напр., возможность брать объекты шире 110мм и тяжелее 1.5кг).

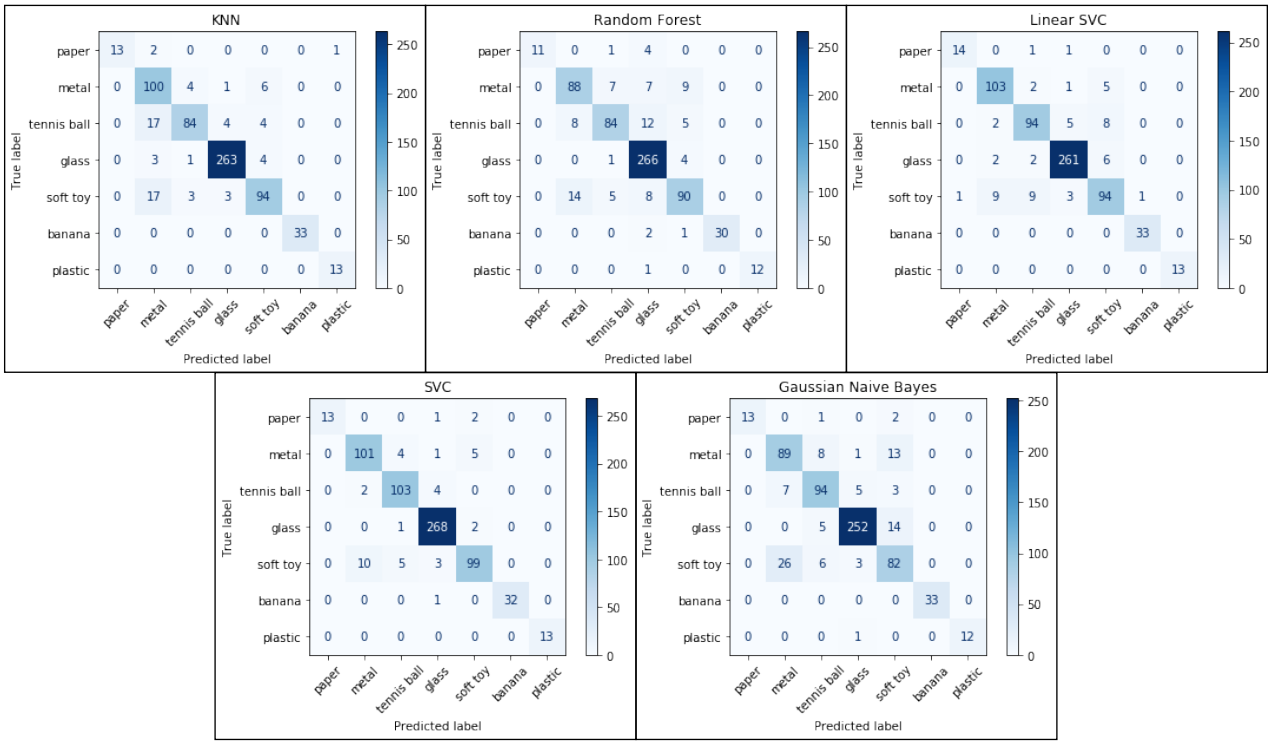
5. Литература

1. Detection of ArUco Markers. Электронный ресурс. Режим доступа: https://docs.opencv.org/trunk/d5/dae/tutorial_aruco_detection.html – Проверено 08.01.2020
2. Цветовая модель HSV. Электронный ресурс. Режим доступа: https://en.wikipedia.org/wiki/HSL_and_HSV – Проверено 08.01.2020.
3. Свёрточная нейронная сеть для решения задачи классификации. Электронный ресурс. Режим доступа: https://mipt.ru/upload/medialibrary/659/91_97.pdf – Проверено 08.01.2020.
4. ImageNet. Электронный ресурс. Режим доступа: http://image-net.org/index – Проверено 08.01.2020.
5. Матрица ошибок. Электронный ресурс. Режим доступа: https://en.wikipedia.org/wiki/Confusion_matrix – Проверено 08.01.2020.
6. R. Berwick, An Idiot's guide to Support vector machines (SVMs) – c.5-28.
7. SCRA Robot Kinematics. Электронный ресурс. Режим доступа: http://www.deltatau.com/Common/technotes/SCARA%20Robot%20Kinematics.pdf – Проверено 08.01.2020.

6. Приложение

Нейронная сеть (DNN)	Процессор	Видеокарта	Время тренировки	Точность
Inception v3	AMD Ryzen 3 3200G	MSI GeForce GTX 1050Ti	13ч 24мин	~50 %
Inception v4	AMD Ryzen 3 3200G	MSI GeForce GTX 1050Ti	21ч 3мин	~61 %
Inception v5	Intel Core i7 6700k	Gigabyte GeForce GTX 1070Ti	12ч 11мин	~65 %
MobileNet	AMD Ryzen 3 3200G	MSI GeForce GTX 1050Ti	20мин	~30%

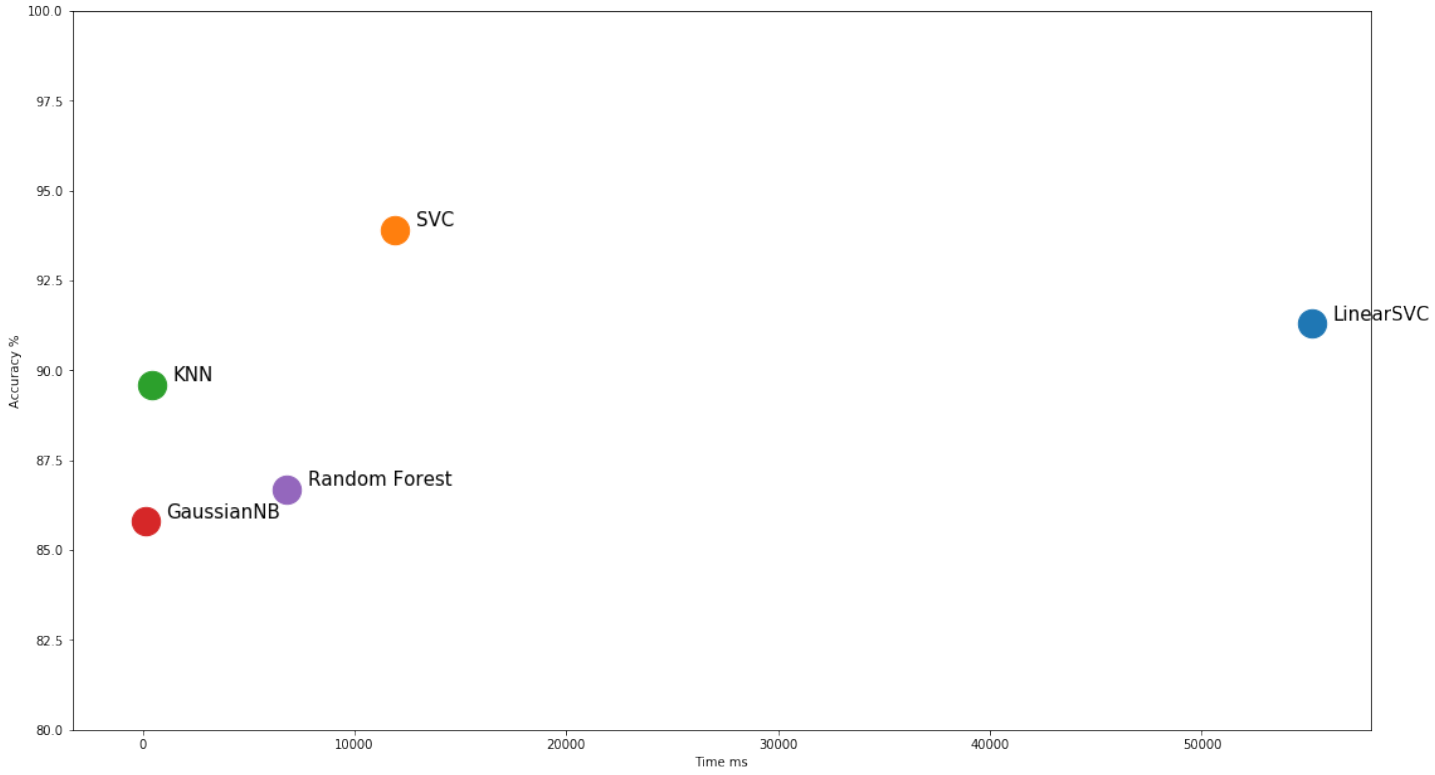
Прил. 1 — время обучения DNN (8 классов по 100 примеров)



Прил. 2.1 — Матрицы ошибок для различных классификаторов

Алгоритм	Время обучения, мс	Точность, %
SVC	11900	93.9
LinearSVC	55200	91.3
KNearestNeighbors	418	89.6
Random Forest	6800	86.7
Gaussian Naive Bayes	118	85.8

Прил. 2.2.1 — сравнения времени обучения и точности алгоритмов классификации

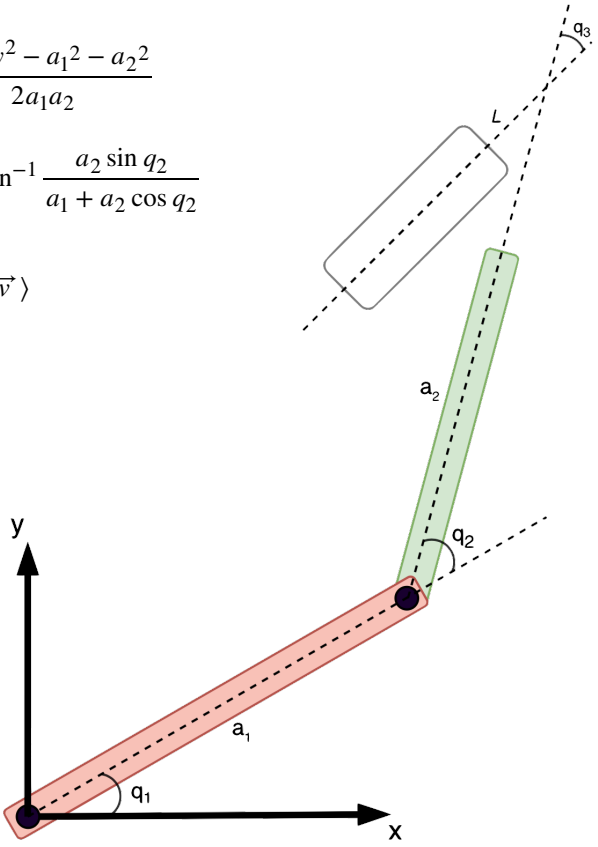


Прил. 2.2 — Распределение классификаторов по времени и точности

$$q_2 = \cos^{-1} \frac{x^2 + y^2 - {a_1}^2 - {a_2}^2}{2a_1a_2}$$

$$q_1 = \tan^{-1} \frac{y}{x} - \tan^{-1} \frac{a_2 \sin q_2}{a_1 + a_2 \cos q_2}$$

$$q_3 = cos^{-1} \langle \overrightarrow{a_2}, \overrightarrow{v} \rangle$$



Прил. 3 — расчёт кинематики манипулятора