МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)**

Институт информационных технологий, математики и механики

Кафедра: Математического обеспечения и суперкомпьютерных технологий

Направление подготовки: «Программная инженерия»
Профиль подготовки: «Разработка программно-информационных систем»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема:

«Разработка алгоритма поиска графического элемента на экране по его шаблонному изображению»

Выполнил:
студент группы 381908-1
Ким Никита Олегович

_____
подпись

Научный руководитель:
доцент кафедры математического
обеспечения и суперкомпьютерных
технологий
Мееров Иосиф Борисович

_____
подпись

Нижний Новгород
2023

Содержание

Введение

Автоматизация тестирования графического интерфейса применяется во многих областях, где используются программные приложения:

- веб-разработка
- мобильная разработка
- игровая индустрия
- финансовая индустрия
- медицина

Это позволяет значительно сократить время и усилия команды тестировщиков. Например, при локализации, часто все, что нужно сделать, это убедиться, что локализованный пользовательский интерфейс ведет себя корректно по отношению к исходному пользовательскому интерфейсу.

Существует множество инструментов, позволяющих реализовать автоматизацию тестирования, однако, чаще всего они либо платные, либо используют не достаточно надежные и универсальные способы доступа к графическим элементам.

Одним из таких инструментов является pywinauto – кроссплатформенная библиотека на Python с открытым исходным кодом [1]. Её отличительным свойством является метод доступа к элементам интерфейса по их атрибутам, с использованием объектно-ориентированного подхода и встроенных возможностей операционных систем. При работе с pywinauto очень полезно дополнительно использовать так называемые инспекторы графических объектов, которые позволяют изучить приложение изнутри: как устроена иерархия элементов, какие свойства доступны.

Известными примерами таких инспекторов являются:

- Spy++ – входит в поставку Visual Studio, включая Express или Community Edition.
- Inspect.exe – входит в Windows SDK.

Также, имеется аналог собственной разработки – py_inspect [12]. Его удобство заключается в том, что атрибуты и имена графических объектов полностью соответствуют атрибутам и именам классов, используемых в pywinauto. Мною было проведено обновление данного инструмента в целях соответствия возможностям pywinauto по состоянию на 2022 год.

Среди других проектов с открытым исходным кодом наподобие pywinauto можно выделить следующие:

- PyAutoGui – популярная кроссплатформенная библиотека (имеет поиск на основе изображений, без текстовых манипуляций с элементами управления).
- Lackey – замена Sikuli на чистом Python (основанная на сопоставлении изображений с шаблонами).
- AXUI – одна из оболочек для MS UI Automation API.
- winGuiAuto – еще один модуль, использующий Win32 API.

(2022, November, 01 - from June, 04)

GitHub (number of stars)

1. RobotJS - 11 373 (+291)
2. pyautogui - 7 327 (+621)
3. AutoHotkey (C++) - 5 958 (+392)
4. Appium Desktop - 4 309 (+205)
5. pywinauto - **3 705** (+211)
6. WinAppDriver (C#) for Appium - 2 953 (+156)
7. sikuli - 1 658 (+8) (SikuliX1 - 2 034 (+204))
8. Python-UIAutomation-for-Windows - 1 549 (+167)
9. FlaUI - 1 442 (+148)
10. TestStack.White (C#) - 1 019 (+3)
11. autopy (Rust+Python) - 681 (+33) (autopy-legacy - 849)
12. lackey - 571 (+13) (pure Python replacement for Sikuli)
13. Winium.Desktop (C#) - 366 (+7)
14. pyatom - 334 (+10)
15. pyautoit - 225 (+13)
16. appium-for-mac - 184 (+4)
17. Guibot - 127 (+8)
18. fMBT - 122 (+3)
19. LDTP cobra - 104 (+0)
20. RAutomation (Ruby) - 98 (+1)

Рисунок 1. Рейтинг проектов по автоматизации приложений

С целью повысить конкурентоспособность (рис. 1), упростить реализацию простых сценариев автоматизации приложений, а также разрешить проблему невозможности получения доступа к графическим элементам методами библиотеки в некоторых случаях, было решено реализовать дополнительную функциональность, основанную на методе сравнения изображений.

В рамках работы поставлена задача реализации алгоритма, позволяющего по шаблонному изображению графического элемента найти его местоположение на экране и получить к нему доступ методами библиотеки. При этом недостаточно просто применить тот же подход, что и в pyautogui: прямой поиск шаблонного изображения со 100%-ной точностью. Переносимость скриптов между машинами с различными разрешениями экрана и шрифтами, а также типичные сценарии использования pywinauto диктуют дополнительные требования к алгоритму, а именно:

1. устойчивость алгоритма к изменениям размеров объекта относительно его шаблона с определенной точностью,
2. способность алгоритма находить n объектов, соответствующих шаблону, где n указывается пользователем.

В связи с этим потребовалось провести поиск подходящих под установленные критерии алгоритмов и выделить среди них тот, который будет показывать наилучшую точность и скорость. Основными источниками вспомогательной литературы были электронные ресурсы и популярные сайты.

Глава 1. Описание алгоритмов

Задача поиска объекта на изображении по шаблону известна уже довольно давно. За последние десятки лет было предложено множество алгоритмов для различных сценариев использования. Условно я разделил их на две большие группы:

1. алгоритмы, использующие попиксельное сравнение
2. алгоритмы, использующие особые свойства искомого объекта

В качестве представителя первой группы был выбран алгоритм Template Matching в реализации OpenCV. Во многом данный выбор был обусловлен простотой использования, популярностью и тем, что этот алгоритм используется в конкурентной библиотеке pyautogui.

Ко второй группе в том числе относятся нейронные сети, но в силу специфики моей задачи, а также учитывая наложенные ограничения и требования, было решено использовать другие алгоритмы. В качестве альтернативы рассматривается Feature Matching – алгоритм выделения ключевых точек объекта в реализации той же библиотеки OpenCV.

1.1 Базовый алгоритм Template Matching

Согласно описанию данного алгоритма в документации OpenCV [2]: Template Matching – это алгоритм поиска и определения местоположения шаблонного изображения на увеличенном изображении. Он просто перемещает изображение шаблона поверх входного изображения (как в 2D-свертке) и сравнивает шаблон и фрагмент входного изображения под изображением шаблона. Алгоритм возвращает изображение в оттенках серого, где значение каждого пикселя обозначает, насколько окрестности этого пикселя соответствуют шаблону.

В качестве формулы сравнения пикселей используется cv.TM_CCOEFF_NORMED [3]:

$$R(x,y) = \frac{\sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}},$$

где $R(x,\ y)$ – значение в пикселе с координатами $(x,\ y)$ результирующего изображения R, T – шаблонное изображение, I – входное изображение, $x' = 0 \dots w - 1, y' = 0 \dots h - 1$ – координаты пикселей по ширине и высоте соответственно.

Если входное изображение имеет размер W на H, где W – ширина, H – высота, а изображение шаблона имеет размер w на h, то результирующее изображение будет иметь размер $(W - w + 1, H - h + 1)$. Точка полученного изображения, имеющая максимальное значение, интерпретируется как наиболее вероятная позиция верхнего левого угла прямоугольника, а (w, h) – как ширина и высота прямоугольника. Этот прямоугольник – искомая область положения шаблона на изображении.

Реализация обработки входного изображения представлена функцией cv.matchTemplate(), а поиск точки с наибольшим значением выполняет cv.minMaxLoc().

Пример работы алгоритма:

Рисунок 2. Шаблон искомого объекта (увеличенный)

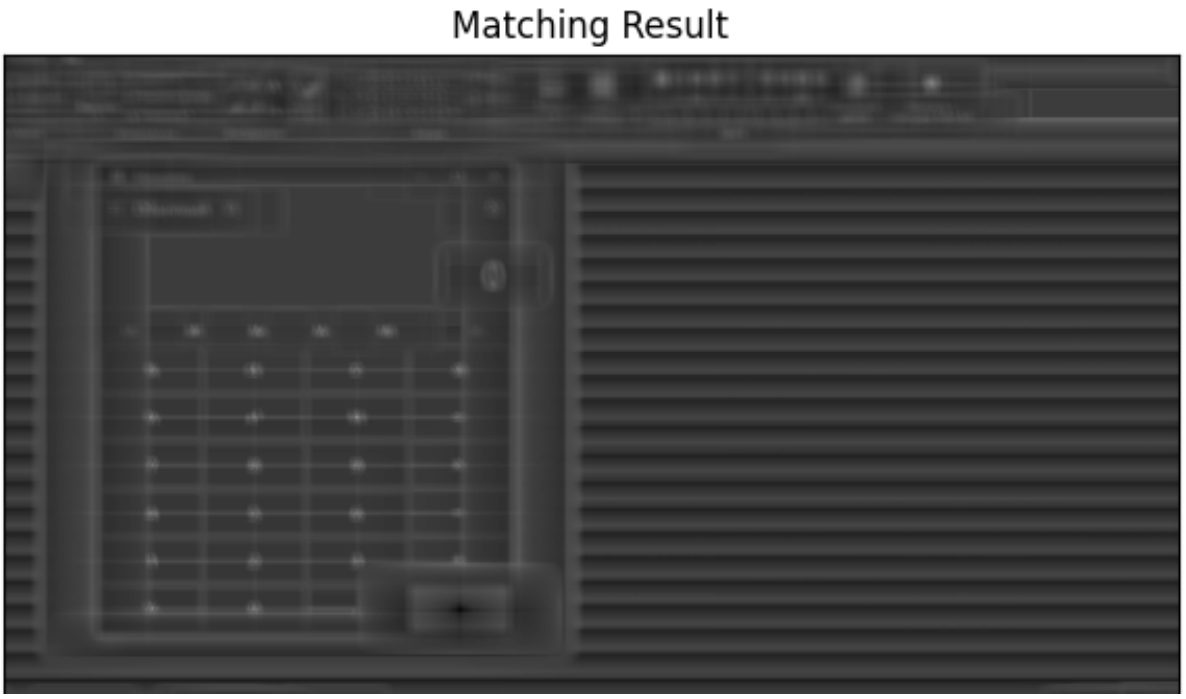

Рисунок 3. Результирующее изображение

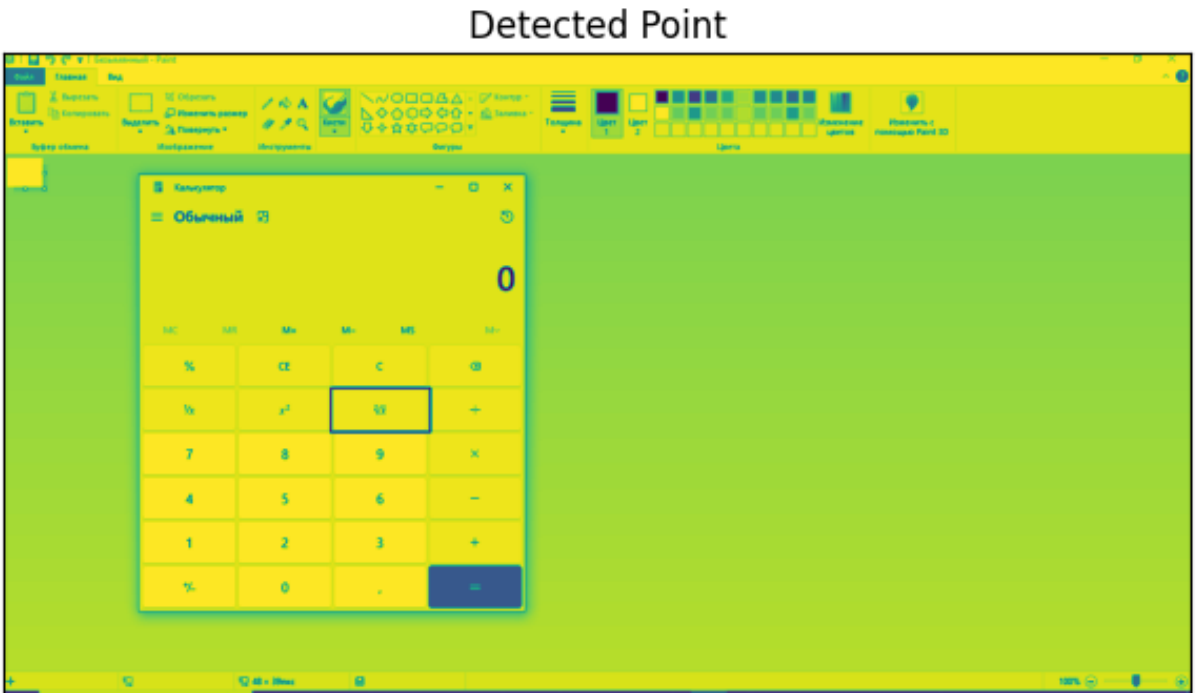

Рисунок 4. Пример визуального результата работы Template Matching

Среди преимуществ данного алгоритма можно отметить

- простота понимания и внедрения
- скорость
- приемлемая точность и возможность ее корректировки

Однако, данный алгоритм не устойчив к изменениям масштаба, что не позволяет использовать его базовую реализацию. Поэтому следует рассмотреть некоторые возможные модификации.

1.2 Алгоритм Modified Template Matching

Проблема алгоритма Template Matching заключается в том, что при изменении размеров искомого объекта, максимальное значение пикселя в результирующем изображении резко падает с 0.9 - 1.0 до 0.2 - 0.9, что не позволяет гарантировать, что объект вообще присутствует на изображении. Другими словами, пропадает устойчивость к отсутствию искомого объекта.

Идея модификации, которую я предлагаю, заключается в дополнительном анализе окрестности пикселя с максимальным значением, чтобы получить возможность распознавать случаи отсутствия и присутствия искомого объекта. Как видно из (рис. 5), в случае попадания в область с искомым объектом, окрестность полученной точки визуально похожа на яркий крест. Чем ярче пиксель результирующего изображения, тем большему значению он соответствует.

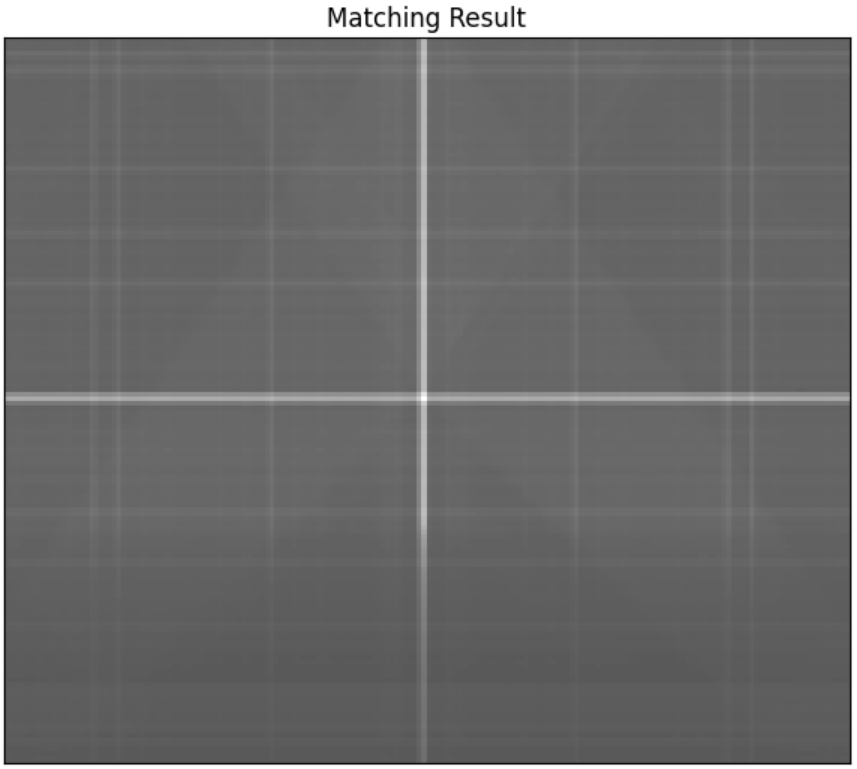

Рисунок 5. Приближенная окрестность результирующей точки

Математически это означает, что если сдвинуть шаблон по горизонтали или вертикали относительно полученной точки, пиксели области изображения под шаблоном будут все еще довольно сильно соответствовать пикселям шаблона. Особенно хорошо этот эффект проявляется, когда изображение искомого объекта содержит много одноцветных пикселей, что на практике встречается довольно часто в нашем случае. С другой стороны, при сдвиге по диагонали, точность соответствия заметно падает.

Мой алгоритм выделяет окрестность результирующей точки размером 3 на 3 (рис. 6) и сравнивает значения по диагонали со значениями по вертикали и горизонтали. Если разность между значением по горизонтали (вертикали) и средним из двух соседних значений по диагоналям превышает заданный порог, то счетчик увеличивается на 1. Если после всех четырех сравнений значение счетчика больше единицы, значит объект присутствует на изображении.

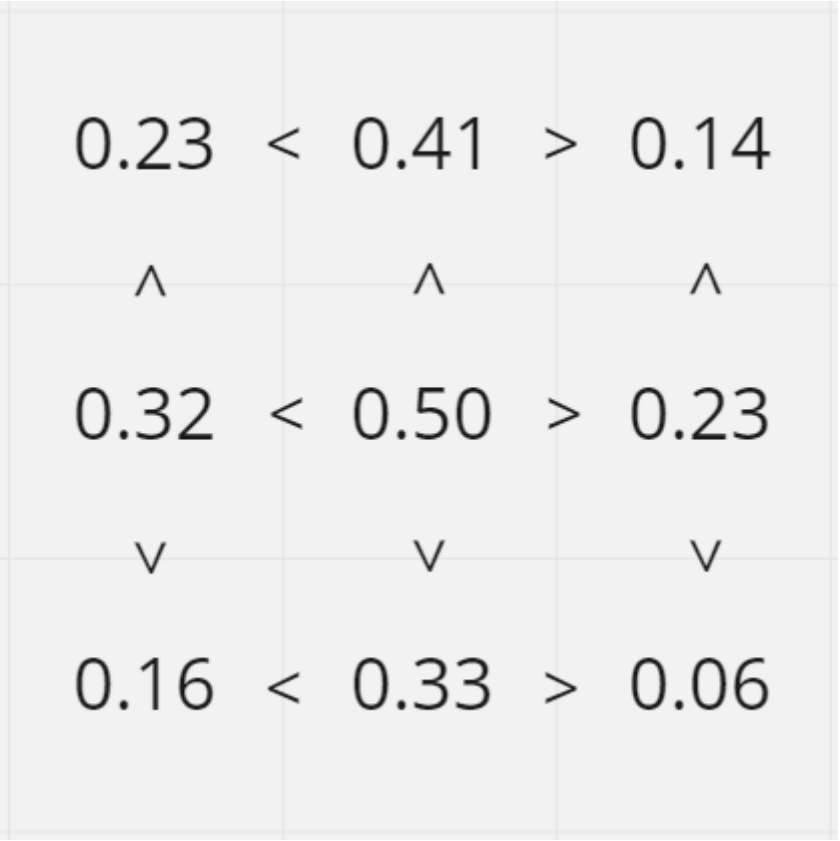

Рисунок 6. Округленные числовые значения в окрестности результирующей точки

1.3 Алгоритм Multi-Scale Template Matching

Данная модификация предложена Адрианом Роузброком (Adrian Rosebrock) в статье [4].

Устойчивость алгоритма Multi-Scale Template Matching к изменениям размеров объекта обеспечивается изменением размеров входного изображения, чтобы масштабы объекта на изображении и его шаблона совпадали как можно больше. Однако, значение в результирующей точке все так же будет в широких пределах 0.2 - 0.9, к тому же, автор рассматривает только случаи увеличения размеров объекта. Поэтому мне пришлось адаптировать данный подход под наши задачи.

Работа адаптированной версии алгоритма разделена на 2 этапа. Сначала запускаем Template Matching на последовательно уменьшающихся версиях входного изображения в масштабе от 100% до 20% с шагом в 4%, и в качестве лучшего результата выбираем тот, у которого точность максимальна, другими словами тот, после которого произошло падение в точности. Потом то же самое на увеличивающихся от 100% до 180% изображениях. Из полученных двух результатов выбираем лучший и проверяем, что величина перепада точности достаточно велика.

На приведенном примере (рис. 7) функция валидации будет выглядеть следующим образом: $0.718 - (0.676 + 0.620)/2 >= 0.01 * accuracy$, где 0.01 – экспериментально полученный коэффициент, $accuracy$ – желаемая точность алгоритма в диапазоне от 0.0 до 1.0, указываемая пользователем.

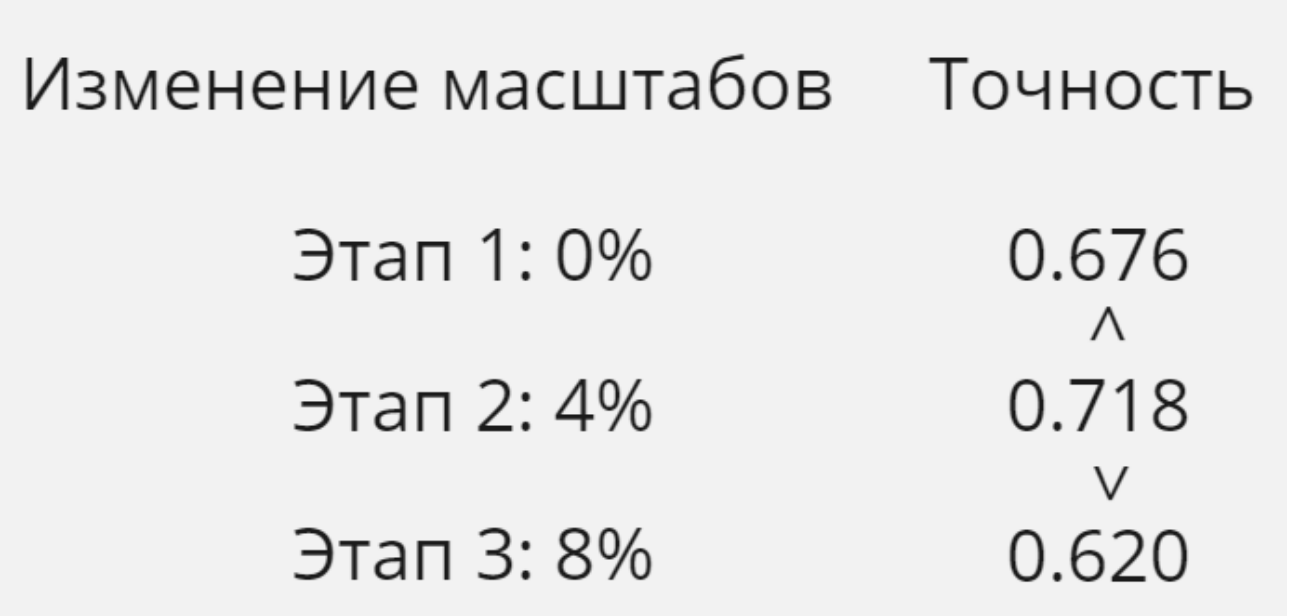

Рисунок 7. Пример промежуточных оценок точности совпадения шаблона с искомым объектом на изображении

Пример работы данного алгоритма:

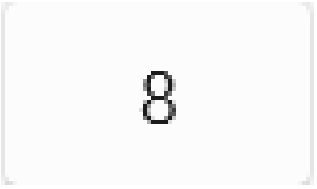

Рисунок 8. Шаблон искомого объекта (увеличенный)

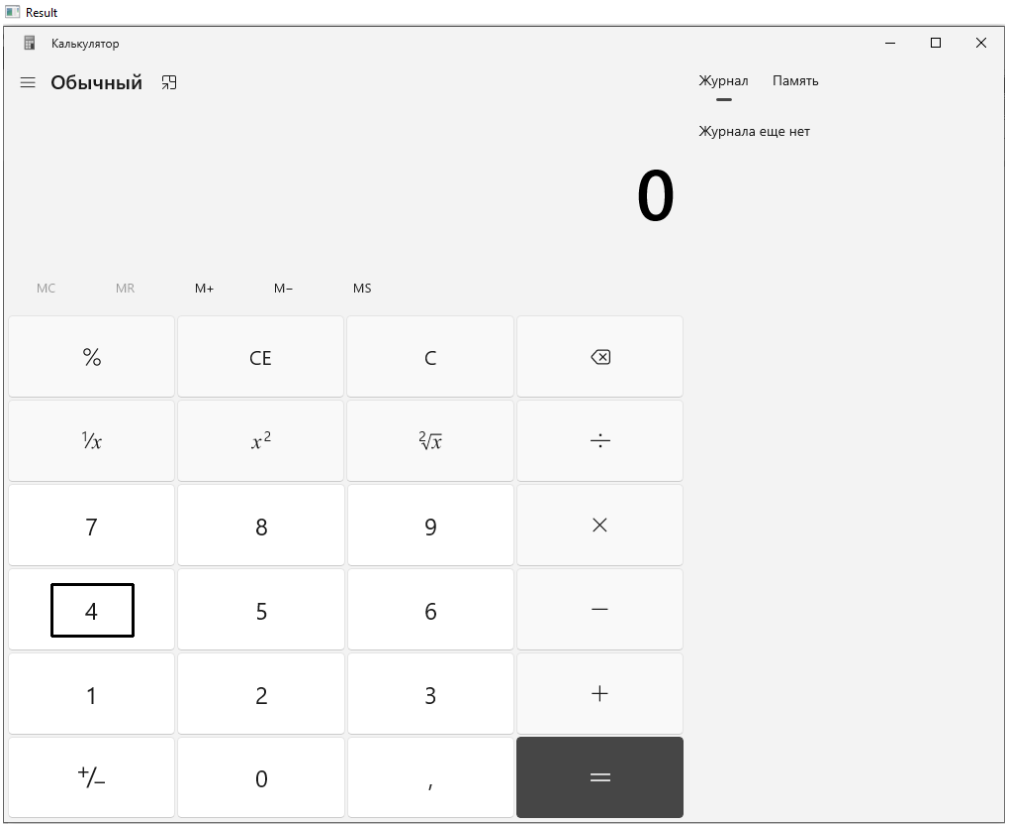

Рисунок 9. Промежуточный результат с уменьшением масштаба на 0%

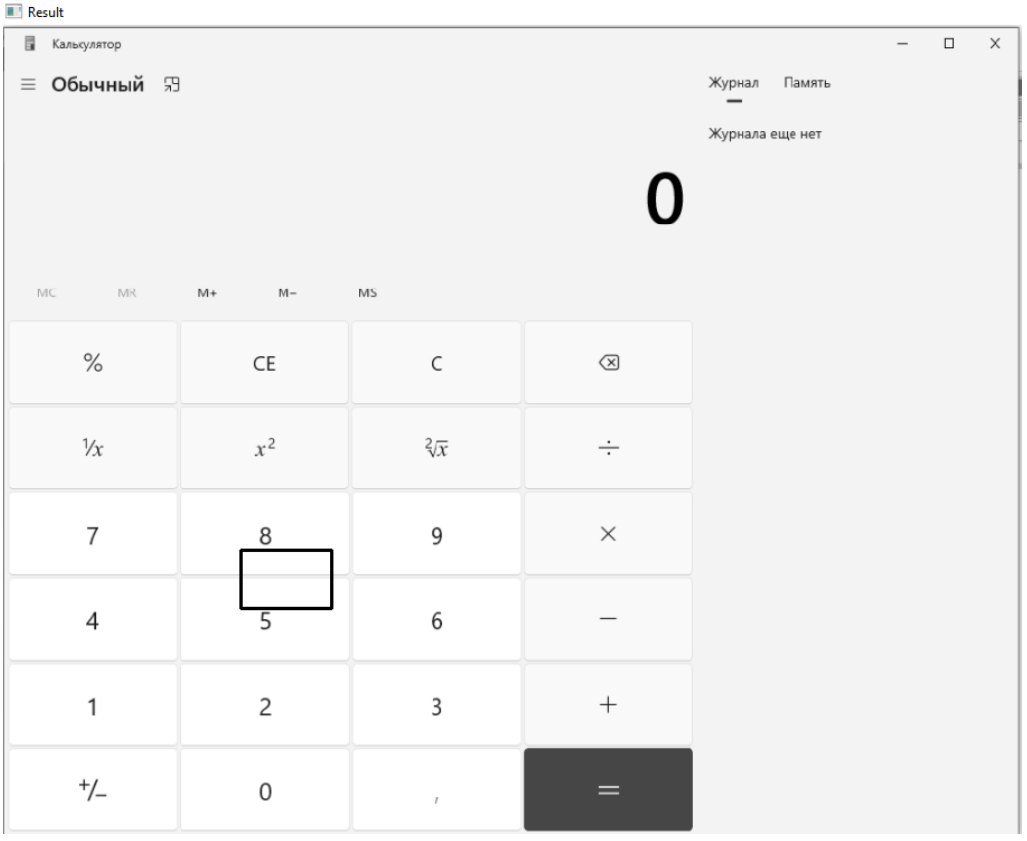

Рисунок 10. Промежуточный результат с уменьшением масштаба на 8%

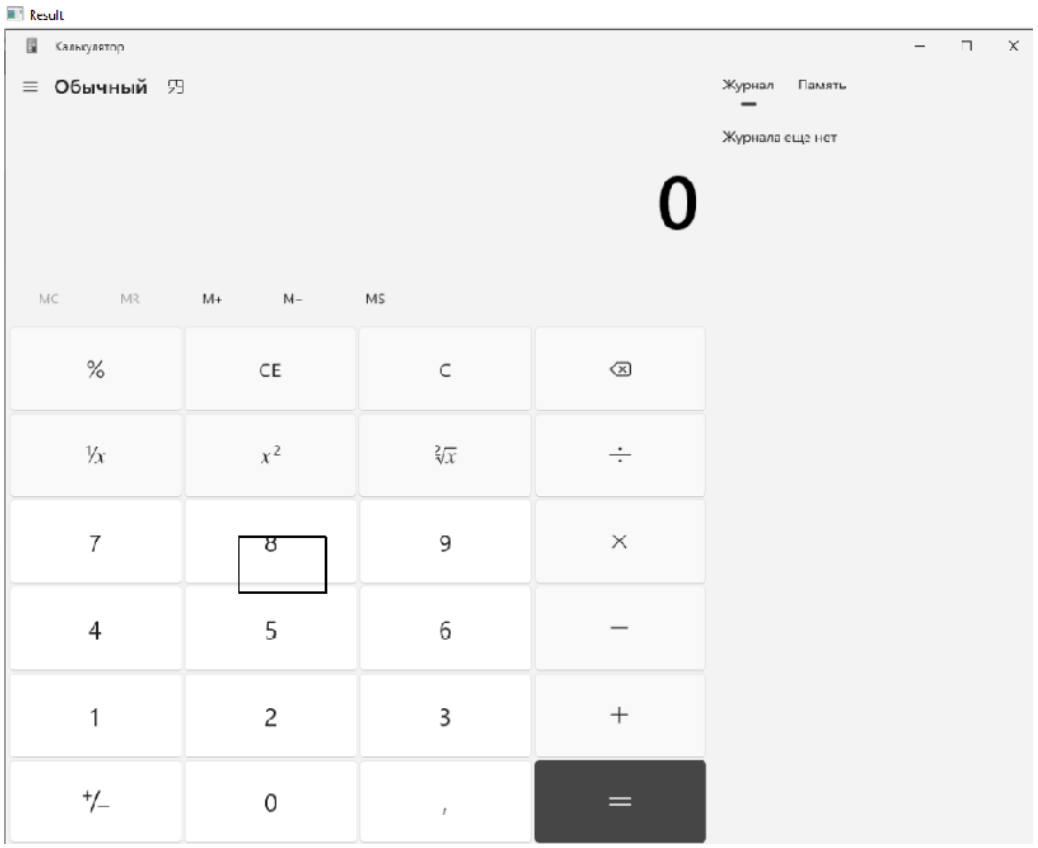

Рисунок 11. Промежуточный результат с уменьшением масштаба на 16%

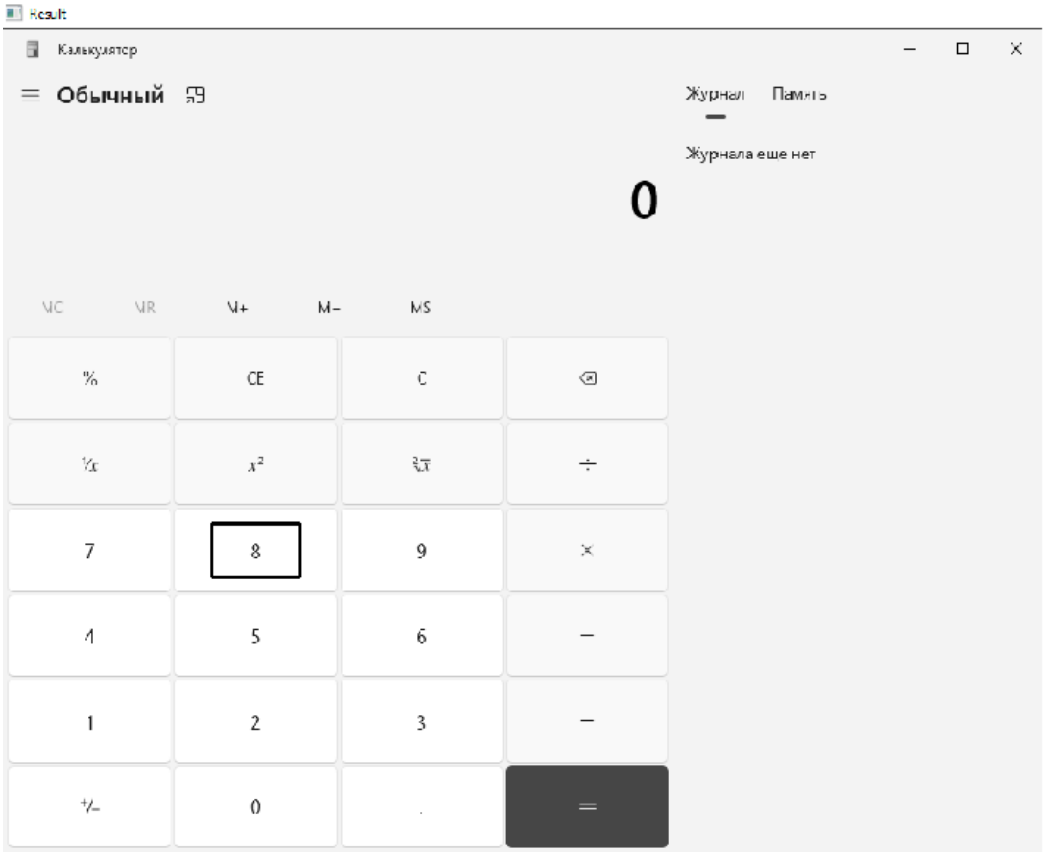

Рисунок 12. Финальный результат с уменьшением масштаба на 24%

1.4 Алгоритм Feature Matching

Согласно описанию данного алгоритма в документации OpenCV [5]: Feature Matching использует Brute-Force Matcher, который принимает дескриптор ключевой точки объекта в первом наборе и сопоставляется со всеми другими ключевыми точками во втором наборе с использованием некоторого вычисления расстояния. И возвращается самая ближайшая точка.

Для Brute-Force Matcher сначала мы должны создать объект BFMatcher, используя cv.BFMatcher(). Данный метод принимает два необязательных параметра.

Первый – это normType. Он определяет используемое измерение расстояния. По умолчанию это cv.NORM_L2, который хорошо подходит для дескрипторов SIFT, SURF, инвариантных к изменениям размеров объекта и поворотам.

Второй параметр – это логическая переменная crossCheck, которая по умолчанию имеет значение false. Если это значение истинно, средство сопоставления возвращает только те совпадения со значением (i, j), при которых i-й дескриптор в наборе A имеет j-й дескриптор в наборе B как наилучшее совпадение, и наоборот. То есть, две функции в обоих наборах должны соответствовать друг другу. Данный параметр обеспечивает стабильный результат и является хорошей альтернативой тесту на соответствие, предложенному Д.Лоу (D.Lowe) и кратко описанному в статье Feature Matching with FLANN [6].

Для определения ключевых точек объекта в нашем случае используется дескриптор SIFT, а для сравнения точек – метод BFMatcher.knnMatch(), который для каждой точки возвращает k наилучших совпадений, где k указано пользователем. Далее проводится вышеупомянутый тест на соответствие, обеспечивающий пользователю возможность управлять точностью алгоритма. Его формула в данной реализации выглядит следующим образом: $m.distance < (0.7 + (1 - accuracy) * 0.3) * n.distance$, где m, n – упорядоченные ближайшие найденные совпадения, $distance$ – расстояние между дескрипторами найденной точки и оригинальной, чем меньше, тем лучше, $accuracy$ – желаемая точность алгоритма в диапазоне от 0.0 до 1.0, указываемая пользователем. Если выражение истинно, то m считается как правильно найденное совпадение, иначе – ложное.

В результате мы получаем набор точек, наиболее соответствующих ключевым точкам шаблона, а в качестве центра искомого объекта на изображении берем центр масс этих точек, предварительно избавившись от крайних значений.

Пример работы алгоритма:

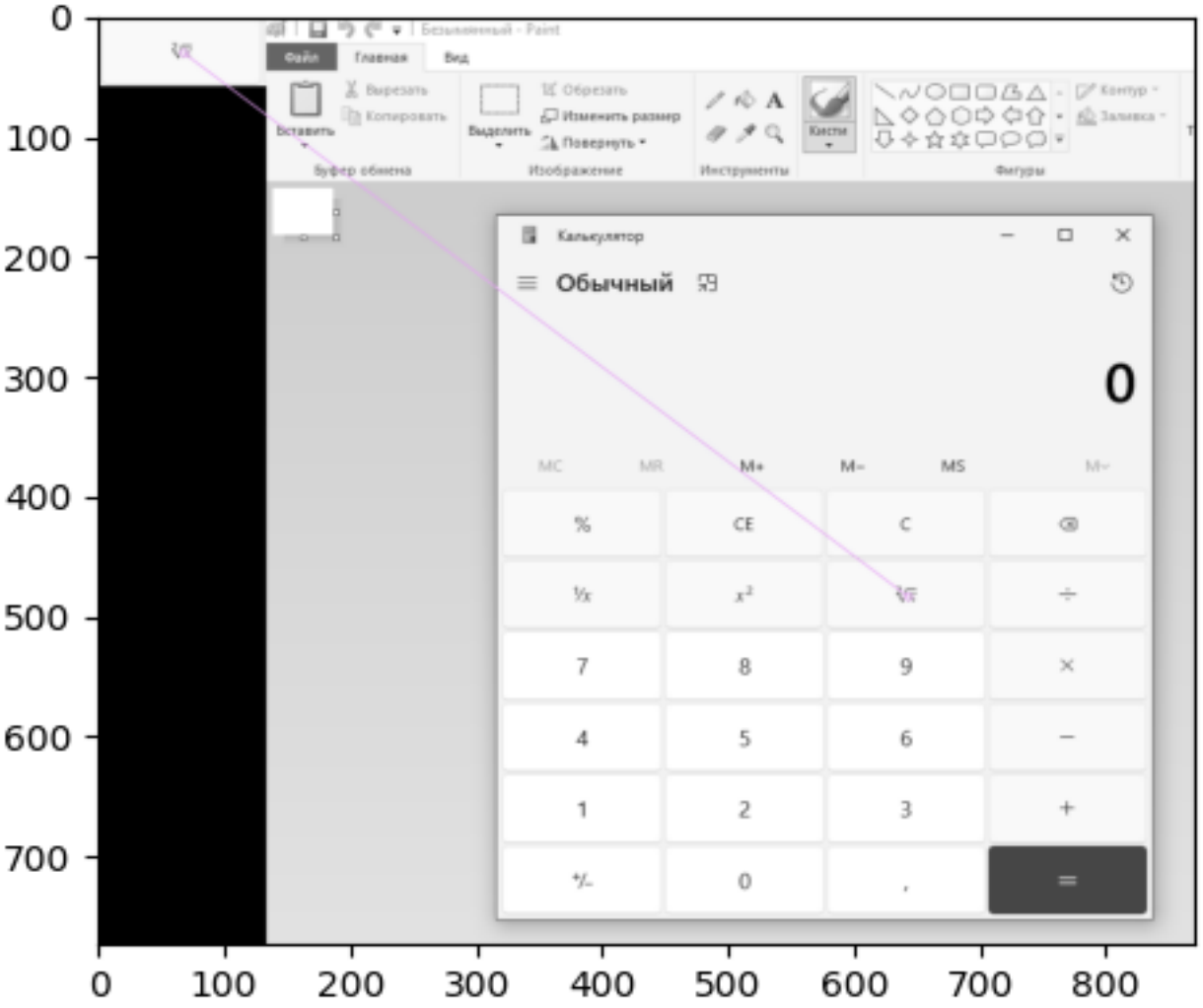

Рисунок 13. Пример соответствия ключевых точек

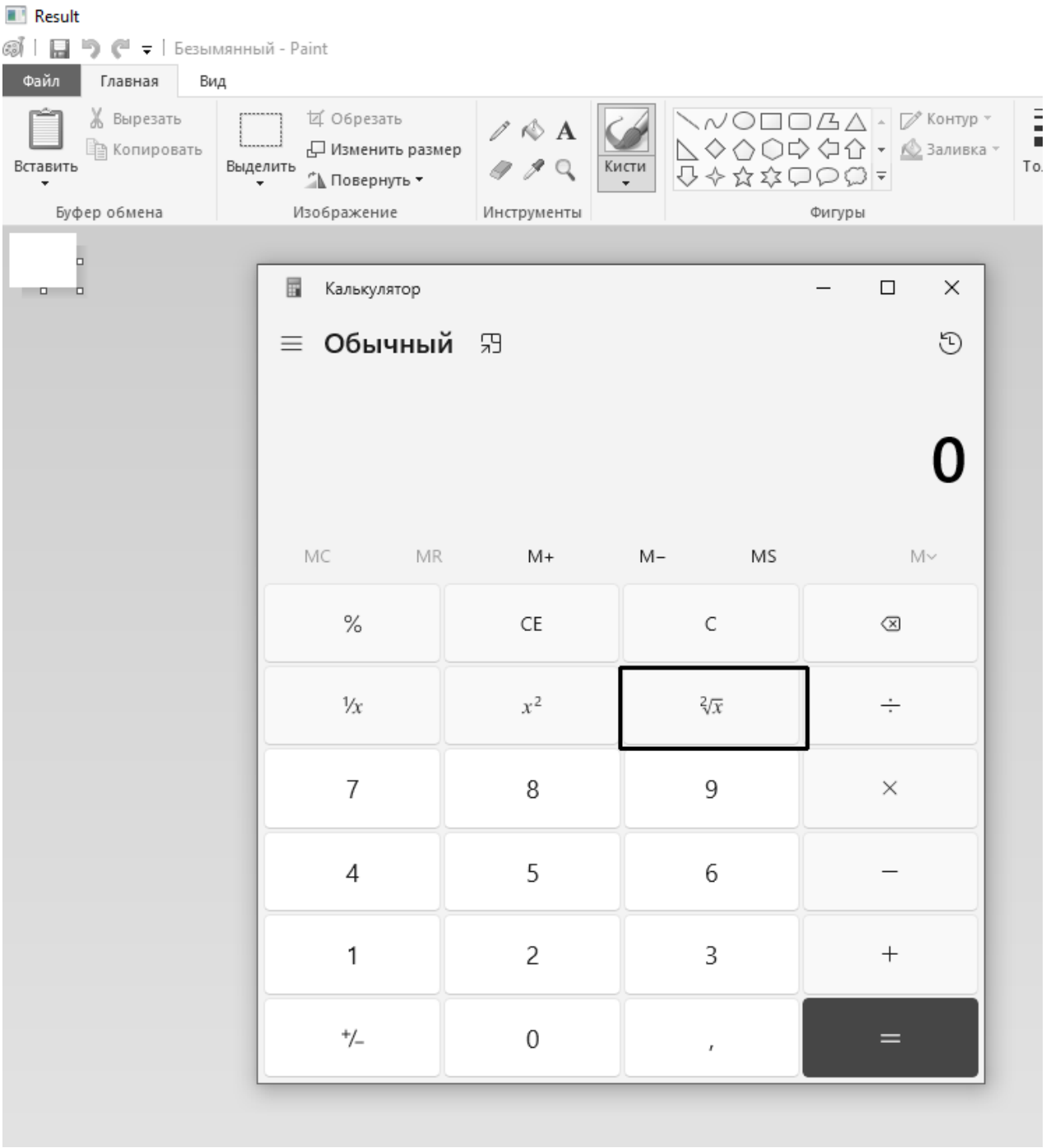

Рисунок 14. Пример результата работы алгоритма Feature Matching

Преимуществом данного алгоритма является устойчивость к аффинным преобразованиям и поворотам, однако, для лучшей точности необходимо наличие у шаблона множества ключевых точек, что не всегда возможно, как в приведенном примере (рис. 13).

Глава 2. Тестирование и сравнение алгоритмов

Итак, имеются 3 алгоритма: Modified Template Matching, Multi-Scale Template Matching и Feature Matching. Необходимо выяснить, какой из них лучше справляется с поставленной задачей.

В качестве метрик, используемых в процессе тестирования, выбраны:

- точность
- время работы алгоритма
- количество используемых сторонних библиотек

Также, введены дополнительные требования, выполнение которых обязательно:

1. возможность пользователя управлять точностью алгоритма
2. способность алгоритма находить n объектов, соответствующих шаблону, где n указывается пользователем
3. устойчивость алгоритма к отсутствию искомого объекта на изображении
4. время работы алгоритма при поиске 1 объекта не должно превышать 1 секунды

Следует отметить, что под точностью алгоритма понимается расстояние от найденной точки до центра объекта, нормированное на размеры объекта (рис. 15).

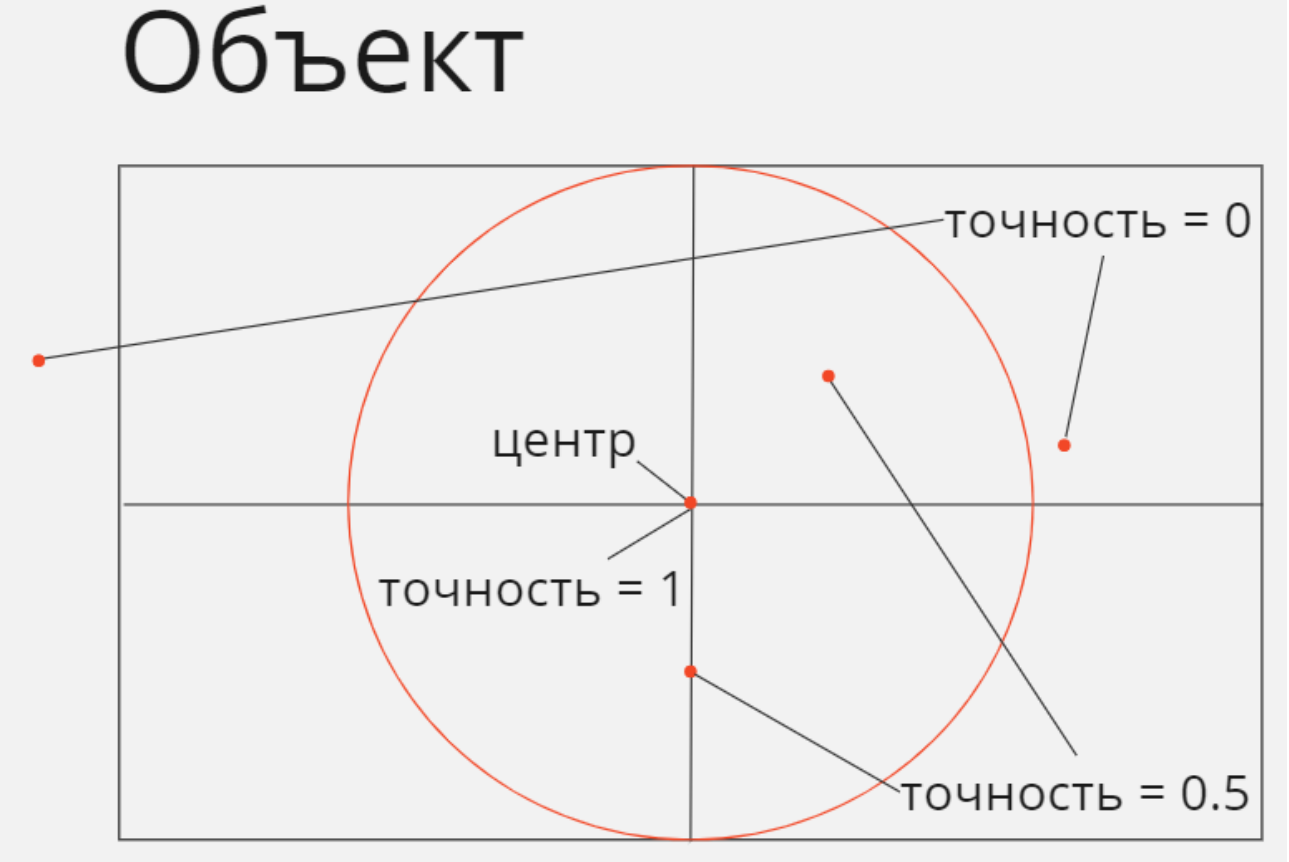

Рисунок 15. Механизм определения точности алгоритма в зависимости от положения результирующей точки

2.1 Обзорный тест

Цель данного теста заключается в сборе общей информации о том, как различные изменения размеров объекта влияют на точность каждого алгоритма, а также в первичной настройке коэффициентов валидации, используемых для определения отсутствия объекта на изображении.

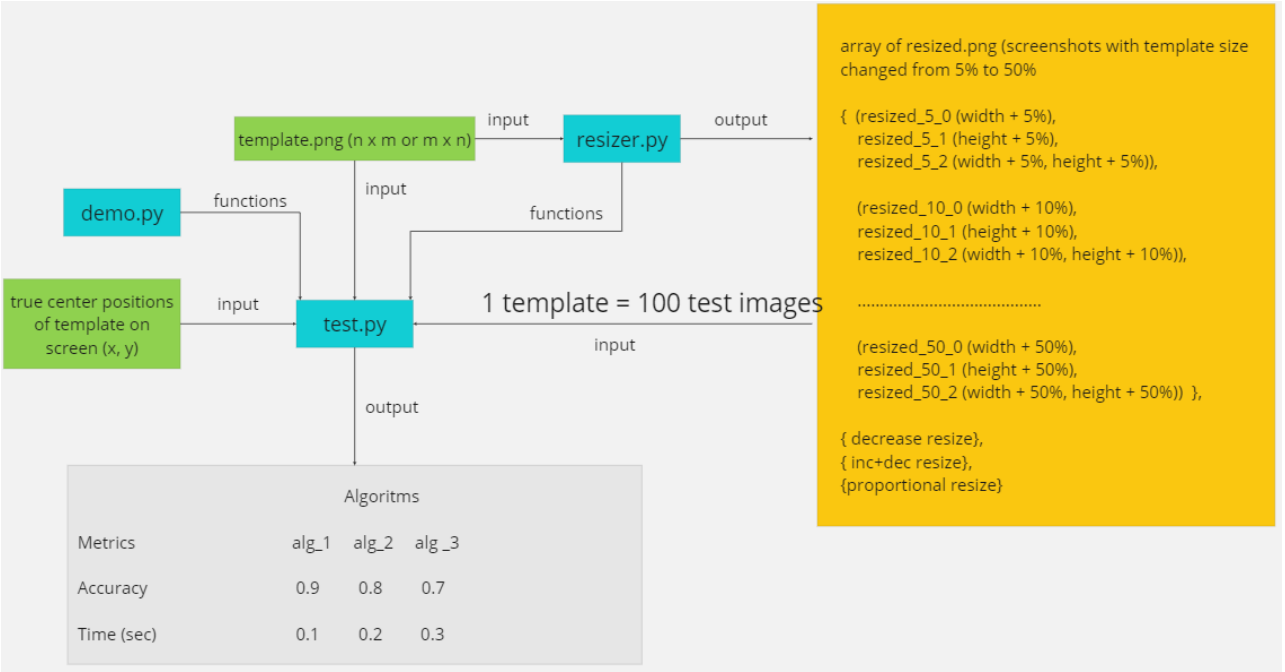

Рисунок 16. Схема процесса обзорного тестирования

Скрипт demo.py содержит реализации каждого алгоритма, скрипт resizer.py содержит функции предобработки данных для теста, а скрипт test.py содержит все описанные в данной главе тесты (рис. 16).

Алгоритм предобработки данных производит синтетические преобразования размеров шаблона от 5% до 50%, так как в реальных случаях изменения более чем на 50% практически не встречаются. Используются 4 типа изменения размеров: увеличение, уменьшение, их комбинация, а также пропорциональное увеличение и уменьшение с сохранением соотношения сторон изображения. Далее, каждый алгоритм запускается на полученной выборке из 100 изображений, измеряется его точность и время работы. В итоговую таблицу заносятся усредненные результаты по каждому процентному изменению размеров для каждого типа изменения.

Итоговая таблица выглядит следующим образом:

"%" increase	base_template_matching	modified_template_matching	multi-scale_template_matching	feature_matching
5	0	0,959558824	0,854187265	0,576822686
10	0	0,960648148	0,736019753	0,64527814
15	0	0,902777778	0,61913337	0,800995534
20	0	0,730112845	0,637161107	0,890406852
25	0	0,698962371	0,623129664	0,540242431
30	0	0,4375	0,499595119	0,842239765
35	0	0,395833333	0,499882637	0,76990753
40	0	0,490459473	0,490459473	0,643663975
45	0	0,432978036	0,492753623	0,638322935
50	0	0,428927184	0,486111111	0,49402224
"%" decrease	base_template_matching	modified_template_matching	multi-scale_template_matching	feature_matching
5	0	1	1	0,710296113
10	0	0,866071429	0,544730401	0,400185528
15	0	0,706362797	0,54130618	0,536842599
20	0	0,64721621	0,424885887	0,313928456
25	0	0,430555556	0,34060095	0,15130345
30	0	0,612124981	0,351401126	0,400764856
35	0	0,4375	0,333333333	0,155730644
40	0	0,541666667	0,333333333	0,312916183
45	0	0,625720323	0,277327052	0,24073581
50	0	0,645833333	0,363328755	0,066016474

Рисунок 17. Таблица результатов по метрике точности алгоритмов, часть 1

"%" incdec	base_template_matching	modified_template_matching	multi-scale_template_matching	feature_matching
5	0	0,966666667	0,879814957	0,467082337
10	0	0,880952381	0,672580242	0,377499239
15	0	0,720394845	0,376200466	0,604879379
20	0	0,572874494	0,199953836	0,244267675
25	0	0,273615372	0,217136181	0
30	0	0,401830818	0,142064372	0,208604969
35	0	0,339293913	0,406293054	0,298457597
40	0	0,367766032	0,209963825	0
45	0	0,421618451	0	0,202923703
50	0	0,441074435	0,375	0,222953999
"%" proportional_resize	base_template_matching	modified_template_matching	multi-scale_template_matching	feature_matching
5	0	0,970588235	0,893954374	0,507772317
10	0	0,865079365	0,449445322	0,73867567
15	0	0,682714262	0,381878998	0,586932787
20	0	0,541955121	0,206586851	0,675044639
25	0	0,298443556	0,248753109	0,656179955
30	0	0,043187472	0,093142679	0,868911752
35	0	0,158334173	0,156073956	0,461123391
40	0	0,173189209	0,173189209	0,404525355
45	0	0,4241201	0,676630435	0,307589758
50	0	0,363882278	0,392593866	0,30962708

Рисунок 18. Таблица результатов по метрике точности алгоритмов, часть 2

Как видно из таблицы результатов (рис. 17, рис. 18), исследуемые алгоритмы проявляют устойчивость к изменениям размера объекта, а их точность падает с ростом процента изменения. Время работы укладывается в отведенные рамки (рис. 19, рис. 20).

"%" increase	base_template_matching	modified_template_matching	multi-scale_template_matching	feature_matching
5	0,0028065	0,002082933	0,006387067	0,019769033
10	0,0010688	0,001836933	0,010941	0,014373933
15	0,000965867	0,0010229	0,009708733	0,0130608
20	0,0011429	0,002420067	0,006318233	0,013391133
25	0,001182167	0,001660233	0,005440233	0,014684833
30	0,001862667	0,001059167	0,0063719	0,013515367
35	0,001318533	0,0022161	0,005190733	0,013690467
40	0,0011362	0,001678433	0,0043893	0,013161367
45	0,001728867	0,001193167	0,0050478	0,012771267
50	0,001779967	0,002624633	0,006785833	0,012430167
"%" decrease	base_template_matching	modified_template_matching	multi-scale_template_matching	feature_matching
5	0,001144233	0,0013325	0,005200933	0,012933933
10	0,0009864	0,001019067	0,004444267	0,012342167
15	0,002331167	0,0019961	0,005189867	0,0120285
20	0,001024333	0,001661233	0,006263167	0,0126989
25	0,001127867	0,000992033	0,0076066	0,012099767
30	0,001279867	0,0020286	0,005719	0,013752367
35	0,002424733	0,001257533	0,006151733	0,012189067
40	0,0011637	0,0010045	0,007310433	0,0126078
45	0,000955767	0,001400967	0,005372033	0,0118245
50	0,001509233	0,0011862	0,007440633	0,011720567

Рисунок 19. Таблица результатов по метрике времени работы алгоритмов, часть 1

"%" incdec	base_template_matching	modified_template_matching	multi-scale_template_matching	feature_matching
5	0,0015008	0,00103855	0,0050319	0,01224815
10	0,0013268	0,0010076	0,00651585	0,01280935
15	0,00106125	0,000995	0,00675245	0,01258075
20	0,00097635	0,00151825	0,00719975	0,01197285
25	0,00096755	0,0012206	0,007452	0,0127057
30	0,00112795	0,0010581	0,00573205	0,01253705
35	0,0011629	0,001045	0,0057555	0,01187805
40	0,00120755	0,00102055	0,00491165	0,0118671
45	0,00129565	0,0010391	0,00578	0,0132572
50	0,00097355	0,001006	0,0108002	0,0121662
"%" proportional_resize	base_template_matching	modified_template_matching	multi-scale_template_matching	feature_matching
5	0,00097525	0,0010073	0,0049622	0,0118431
10	0,0010218	0,0011964	0,0081722	0,01177715
15	0,00103695	0,00106825	0,00682025	0,01246185
20	0,0009745	0,0010215	0,00841225	0,0129524
25	0,0011566	0,00105105	0,0078827	0,01219055
30	0,00101965	0,0010264	0,0075637	0,0123472
35	0,0009711	0,00101605	0,00750155	0,01205085
40	0,0009712	0,0010111	0,0110284	0,0129645
45	0,0009619	0,00099245	0,00547355	0,01265935
50	0,00096225	0,00115665	0,01283095	0,01236675

Рисунок 20. Таблица результатов по метрике времени работы алгоритмов, часть 2

Анализируя результаты данного теста, нельзя однозначно утверждать, какой алгоритм лучше подходит для решения поставленной задачи. Вследствие чего далее приводятся дополнительные тесты.

2.2 Итоговый тест на синтетических данных

Данный тест проверяет качество работы алгоритмов по 3 сценариям:

1. нахождение центра объекта на изображении по шаблону
2. нахождение центров n объектов на изображении по шаблону, где $n = 5$ в данном случае (рис. 21)
3. подтверждение отсутствия искомого объекта на изображении (рис. 22)

Рисунок 21. Формат входных данных для 2 сценария

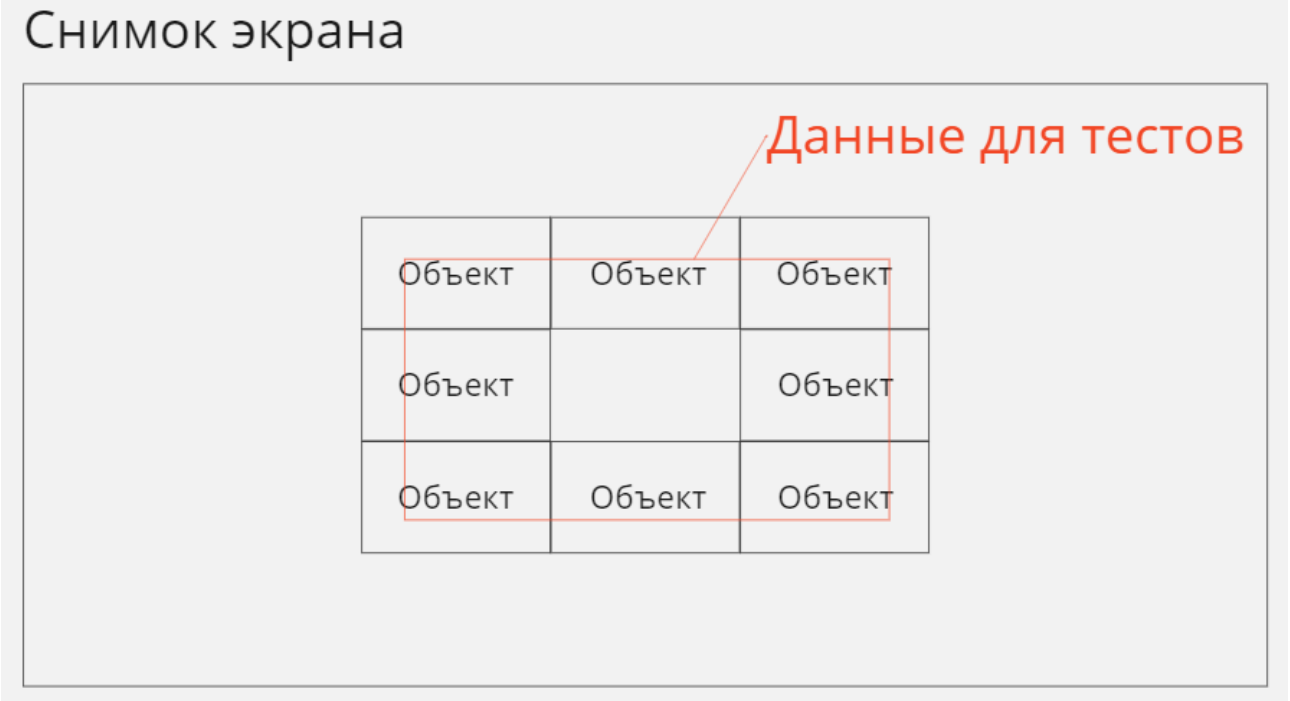

Рисунок 22. Формат входных данных для 3 сценария

К сожалению, настроить коэффициенты валидации таким образом, чтобы алгоритмы демонстрировали хорошую точность как при наличии объекта так и при его отсутствии, невозможно. На практике, сценарий 1 у пользователей встречается чаще, чем сценарий 3 (рис. 23). Поэтому был сделан выбор в пользу улучшения точности на первых двух сценариях, за счет третьего.

Рисунок 23. Матрица потерь статистической игры «Пользователь – Алгоритм»

Для каждого сценария в случайном порядке выбирается половина из всех возможных типов процентного изменения размеров объекта и применяется к заранее отобранным 10 шаблонам. Итоговые значения метрик берутся как среднее по всем результатам. Также, стоит отметить, что в 3 сценарии точность (accuracy) измеряется по-другому: если алгоритм находит объект на изображении, то его точность считается равной нулю, если же алгоритм не находит объект, точность считается равной единице.

Так как алгоритмы изначально не были приспособлены ко 2 сценарию, пришлось написать их отдельные версии. Для первых двух алгоритмов, являющихся модификациями Template Matching, после обнаружения очередного объекта, соответствующая ему область вырезается из изображения, чтобы не участвовать в дальнейших вычислениях. Для Feature Matching проводится дополнительная кластеризация множества найденных точек совпадений методами библиотеки scikit-learn [7], где число кластеров соответствует числу искомых объектов. Результатом работы будут центры кластеров.

Синтетические тестовые данные			
Точность	Modified Template Matching	Multi-Scale Template Matching	Feature Matching
Нахождение центра объекта	0,609458423	0,532352832	0,373208526
Нахождение центров 5 объектов	0,595458304	0,529959255	0,453800886
Подтверждение отсутствия объекта	0,6984	0,257	0,6782
Время	Modified Template Matching	Multi-Scale Template Matching	Feature Matching
Нахождение центра объекта	0,00277723	0,020727167	0,02114883
Нахождение центров 5 объектов	0,110872614	0,680886441	0,146272179
Подтверждение отсутствия объекта	0,002695276	0,015875516	0,021888964
	Modified Template Matching	Multi-scale Template Matching	Feature Matching
Используемые библиотеки	opencv, numpy	opencv, numpy, imutils	opencv, numpy, math, scikit-learn

Рисунок 24. Результаты итогового теста на синтетических данных

Анализируя полученные результаты (рис. 24) можно предварительно сказать, что лучше всего справляется с поставленной задачей алгоритм Modified Template Matching. Multi-Scale Template Matching не сильно отстает в точности, за исключением очень малого значения в 3 сценарии. Feature Matching показал худшие результаты на первых двух сценариях и приемлемую точность на 3 сценарии. Во многом это произошло из-за того, что данные в нашей задаче редко обладают достаточным количеством ключевых точек. В силу той же проблемы, провести качественную кластеризацию затруднительно.

Чтобы прийти к заключительным выводам, необходимо провести тестирование на реальных случаях изменения размеров объекта.

2.3 Итоговый тест на реальных данных

В данном тесте будет проверяться только 1 сценарий, так как результаты на 2 сценарии коррелируют с 1, а соотношения результатов 3 сценария не зависят от причины изменения размеров объекта.

В качестве тестовых данных используются 5 элементов графического интерфейса, чьи размеры были изменены посредством использования различных разрешений экрана на различных компьютерах с операционной системой windows 10.

Реальные тестовые данные			
Нахождение центра объекта	Modified Template Matching	Multi-Scale Template Matching	Feature Matching
Точность	0,715814227	0,761402977	0,505188136
Время	0,002590295	0,008120715	0,036852102

Рисунок 25. Результаты итогового теста на реальных данных

На реальных случаях изменения размеров объекта (рис. 25) алгоритмы показывают более высокую точность, чем на синтетических. Этот эффект возникает из-за сравнительно малых изменений масштабов на практике, а также искусственно увеличенной сложности тестов на синтетических данных. Feature Matching на данном этапе исключается из рассмотрения ввиду худших показателей. При этом, показатель точности алгоритма Multi-Scale Template Matching лучше, чем у Modified Template Matching. Данный факт не позволяет объявить Multi-Scale Template Matching более эффективным из-за его худших результатов на 3 сценарии синтетических данных (рис. 24). Однако возникает предположение, что комбинация вышеупомянутых алгоритмов может обладать большей средней точностью по всем сценариям относительно Modified Template Matching. Реализацию этой комбинации назовем Advanced Template Matching.

2.4 Тестирование алгоритма Advanced Template Matching

Алгоритм Advanced Template Matching содержит адаптивное изменение масштабов изображения от Multi-Scale Template Matching и функцию валидации от Modified Template Matching. Реализация изменению не подвергалась.

Синтетические тестовые данные			
Точность	Modified Template Matching	Multi-Scale Template Matching	Advanced Template Matching
Нахождение центра объекта	0,529551273	0,500194846	0,506961477
Нахождение центров 5 объектов	0,450839419	0,487056614	0,606218323
Подтверждение отсутствия объекта	0,743	0,3476	0,7438
Время	Modified Template Matching	Multi-Scale Template Matching	Advanced Template Matching
Нахождение центра объекта	0,002485896	0,019801279	0,021344192
Нахождение центров 5 объектов	0,11036972	0,703255067	0,798938782
Подтверждение отсутствия объекта	0,001668989	0,011851912	0,013061589
	Modified Template Matching	Multi-scale Template Matching	Advanced Template Matching
Используемые библиотеки	opencv, numpy	opencv, numpy, imutils	opencv, numpy, imutils

Рисунок 26. Результаты тестирования Advanced Template Matching на синтетических данных

Реальные тестовые данные			
Нахождение центра объекта	Modified Template Matching	Multi-Scale Template Matching	Advanced Template Matching
Точность	0,715814227	0,761402977	0,717287366
Время	0,00458003	0,019035462	0,012682215

Рисунок 27. Результаты тестирования Advanced Template Matching на реальных данных

На реальных данных (рис. 27) произошло повышение точности, в то время как на синтетических данных (рис. 26) наблюдается заметный рост результативности во 2 сценарии, по сравнению с незначительными изменениями показателей в 1 и 3 сценариях. Значения показателей времени работы алгоритма ожидаемы и удовлетворительны. Исходя из полученных результатов по всем тестам, алгоритм Advanced Template Matching объявляется наиболее подходящим для решения поставленной задачи.

Итоговые коэффициенты валидации выглядят следующим образом: $similarity = accuracy * 0.02$, где $similarity$ используется как пороговое значение для упомянутых в пункте 1.2 сравнениях (рис. 6), а $accuracy$ задается пользователем как желаемая точность алгоритма в диапазоне от 0.0 до 1.0.

Заключение

В данной работе проведено исследование и сравнение алгоритмов поиска графического элемента на изображении по его шаблону. В качестве основных подходов к решению поставленной проблемы были выделены 2 способа сопоставления изображений: попиксельное сравнение и сравнение ключевых точек.

Представителем 1 подхода был выбран популярный алгоритм Template Matching. Так как его базовый вариант реализации в библиотеке OpenCV не способен соблюдать установленные требования, была предложена модификация Modified Template Matching и использована известная модификация Multi-Scale Template Matching.

Представителем 2 подхода был выбран алгоритм Feature Matching. Для его корректной работы на пользовательских сценариях с несколькими одинаковыми элементами на экране, к исходному коду была добавлена кластеризация методом k-means.

После адаптации алгоритмов к условиям поставленной задачи реализован набор тестов, проверяющих их результативность по таким показателям, как точность попадания в область искомого объекта и время исполнения. Полученные результаты первого тестирования на синтетически сгенерированных и реальных данных позволили исключить Feature Matching из рассмотрения. В то же время, определить наиболее эффективный алгоритм из оставшихся оказалось затруднительно, вследствие отсутствия полной корреляции результатов между тестом на синтетических данных и на реальных.

Чтобы разрешить возникшую проблему, предложен и реализован алгоритм Advanced Template Matching, объединяющий в себе преимущества Modified Template Matching и Multi-Scale Template Matching. В результате повторного тестирования, предложенный алгоритм показал удовлетворительные результаты. Несмотря на ухудшение показателей в отдельных сценариях, общее повышение точности по сравнению с конкурентами позволяет классифицировать Advanced Template Matching как наиболее подходящий для решения поставленной задачи.

В результате разработан алгоритм, способный находить объект на изображении по шаблону, а также проявляющий устойчивость к изменениям размеров искомого объекта относительно шаблона. Внедрение Advanced Template Matching в библиотеку pywinauto повысит ее конкурентоспособность и позволит пользователям решить следующие актуальные проблемы:

- необходимость порой продолжительного изучения документации и использования дополнительных инструментов для реализации простых сценариев,
- невозможность получить доступ к графическим элементам методами библиотеки pywinauto в некоторых особенных случаях,
- нестабильность процесса автоматического тестирования приложений при переносе с одной ЭВМ на другую.

Результирующий алгоритм демонстрирует приемлемую точность и скорость, а также соответствует всем требованиям. Дальнейшая работа будет направлена на сбор пользовательских отзывов с целью более точной настройки при необходимости и поддержку новой функциональности библиотеки pywinauto.

Список литературы

1. Contents - pywinauto 0.6.8 documentation // Read the Docs URL: https://pywinauto.readthedocs.io/en/latest/contents.html (дата обращения 28.05.2023).
2. OpenCV: Template Matching // OpenCV documentation index URL: https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html (дата обращения 28.05.2023).
3. OpenCV: Object Detection // OpenCV documentation index URL: https://docs.opencv.org/4.x/df/dfb/group__imgproc__object.html (дата обращения 28.05.2023).
4. Adrian Rosebrock. Multi-scale Template Matching using Python and OpenCV // PyImageSearch URL: https://pyimagesearch.com/2015/01/26/multi-scale-template-matching-using-python-opencv (дата обращения 28.05.2023).
5. OpenCV: Feature Matching // OpenCV documentation index URL: https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html (дата обращения 28.05.2023).
6. OpenCV: Feature Matching with FLANN // OpenCV documentation index URL: https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html (дата обращения 28.05.2023).
7. sklearn.cluster.KMeans // scikit-learn 1.2.2 documentation URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (дата обращения 28.05.2023).
8. Python 3 для начинающих и чайников - уроки программирования URL: https://pythonworld.ru (дата обращения 28.05.2023).
9. Stack Overflow - Where Developers Learn, Share, & Build Careers URL: https://stackoverflow.com (дата обращения 28.05.2023).
10. Все статьи подряд / Хабр URL: https://habr.com/ru/all (дата обращения 28.05.2023).
11. pywinauto URL: https://github.com/pywinauto/pywinauto.
12. py_inspect URL: https://github.com/pywinauto/py_inspect.
13. Image Matching for UI objects URL: https://github.com/KimNikita/image_matching.

Приложение 1. Исходный код рассматриваемых алгоритмов

Листинг № 1

```
def modified_template_matching(template, accuracy=0.95, second_try=True):
    import cv2 as cv
    import numpy as np
    from PIL import ImageGrab

    screenshot = ImageGrab.grab(None)
    screenshot.save('screen.png')
    screenshot = cv.imread('screen.png', cv.IMREAD_GRAYSCALE)

    if isinstance(template, str):
        template = cv.imread(template, cv.IMREAD_GRAYSCALE)
        if template is None:
            print('Cannot read image, check cv2.imread() documentation')
            return None
    else:
        print('Invalid format of image')
        return None

    res = cv.matchTemplate(screenshot, template, cv.TM_CCOEFF_NORMED)

    _, max_value, _, max_location = cv.minMaxLoc(res)
    h, w = template.shape
    box = (-1, -1, -1, -1)

    if max_value >= accuracy:
        box = (max_location[0], max_location[1],
               max_location[0] + w, max_location[1] + h)
    elif second_try and max_location[1]-1>=0 and max_location[0]-1>=0 and
max_location[1]+1<res.shape[0] and max_location[0]+1<res.shape[1]:
        similarity = accuracy * 0.02

        val00 = res[max_location[1]-1][max_location[0]-1]
        val10 = res[max_location[1]][max_location[0]-1]
        val20 = res[max_location[1]+1][max_location[0]-1]

        val01 = res[max_location[1]-1][max_location[0]]
        val21 = res[max_location[1]+1][max_location[0]]

        val02 = res[max_location[1]-1][max_location[0]+1]
        val12 = res[max_location[1]][max_location[0]+1]
        val22 = res[max_location[1]+1][max_location[0]+1]
```

```
        valid_score = 0
        if val10 - (val00+val20)/2 >= similarity:
            valid_score += 1
        if val01 - (val00+val02)/2 >= similarity:
            valid_score += 1
        if val21 - (val20+val22)/2 >= similarity:
            valid_score += 1
        if val12 - (val02+val22)/2 >= similarity:
            valid_score += 1

        if valid_score >= 2:
            box = (max_location[0], max_location[1],
                   max_location[0] + w, max_location[1] + h)
    return box


def multi_scale_template_matching(template, accuracy=0.95):
    import numpy as np
    import imutils
    import cv2 as cv
    from PIL import ImageGrab

    screenshot = ImageGrab.grab(None)
    screenshot.save('screen.png')
    screenshot = cv.imread('screen.png', cv.IMREAD_GRAYSCALE)

    if isinstance(template, str):
        template = cv.imread(template, cv.IMREAD_GRAYSCALE)
        if template is None:
            print('Cannot read image, check cv2.imread() documentation')
            return None
    else:
        print('Invalid format of image')
        return None

    h, w = template.shape
    found_dec = (0, 0, 0)
    prev_max= [0, 0]
    next_min= [0, 0]
    for scale in np.linspace(0.2, 1.0, 20)[::-1]:
        resized = imutils.resize(
            screenshot, width=int(screenshot.shape[1] * scale))
        r = screenshot.shape[1] / float(resized.shape[1])

        if resized.shape[0] < h or resized.shape[1] < w:
            break
```

```
        result = cv.matchTemplate(resized, template, cv.TM_CCOEFF_NORMED)
        (_, maxVal, _, maxLoc) = cv.minMaxLoc(result)

        if maxVal > found_dec[0]:
            prev_max[0] = found_dec[0]
            found_dec = (maxVal, maxLoc, r)
        else:
            next_min[0]=maxVal
            break

    found_inc = (0, 0, 0)
    for scale in np.linspace(0.04, 0.8, 19):
        resized = imutils.resize(
            screenshot, width=int(screenshot.shape[1] * (1+scale)))
        r = screenshot.shape[1] / float(resized.shape[1])

        result = cv.matchTemplate(resized, template, cv.TM_CCOEFF_NORMED)
        (_, maxVal, _, maxLoc) = cv.minMaxLoc(result)

        if maxVal > found_inc[0]:
            prev_max[1]=found_inc[0]
            found_inc = (maxVal, maxLoc, r)
        else:
            next_min[1]=maxVal
            break

    i = 0
    if found_dec[0] > found_inc[0]:
        (maxVal, maxLoc, r) = found_dec
    else:
        i=1
        (maxVal, maxLoc, r) = found_inc

    if next_min[i]==0:
        next_min[i]=prev_max[i]
    if maxVal - (prev_max[i]+next_min[i])/2 < 0.01 * accuracy:
        return (-1, -1, -1, -1)
    (startX, startY) = (int(maxLoc[0] * r), int(maxLoc[1] * r))
    (endX, endY) = (int(maxLoc[0] * r + w), int(maxLoc[1]  * r +h))

    return (startX, startY, endX, endY)


def feature_matching(template, accuracy=0.95):
    import numpy as np
    import cv2 as cv
```

```
from PIL import ImageGrab
import math

screenshot = ImageGrab.grab(None)
screenshot.save('screen.png')
screenshot = cv.imread('screen.png', cv.IMREAD_GRAYSCALE)

if isinstance(template, str):
    template = cv.imread(template, cv.IMREAD_GRAYSCALE)
    if template is None:
        print('Cannot read image, check cv2.imread() documentation')
        return None
else:
    print('Invalid format of image')
    return None

w, h = template.shape[::-1]
box = (-1, -1, -1, -1)

sift = cv.SIFT_create()

kp1, des1 = sift.detectAndCompute(template, None)
kp2, des2 = sift.detectAndCompute(screenshot, None)

if des1 is None or des2 is None:
    return box

best_matches = []

bf = cv.BFMatcher(cv.NORM_L2, crossCheck=False)
matches = bf.knnMatch(des1, des2, k=2)

for pair in matches:
    if len(pair) < 2:
        continue
    m, n = pair
    if m.distance < (0.7 + (1-accuracy)*0.3)*n.distance:
        best_matches.append(m)

if len(best_matches)==0:
    return box

list_x = []
list_y = []

for mat in best_matches:
```

```
        screenshot_idx = mat.trainIdx

        (x, y) = kp2[screenshot_idx].pt

        list_x.append(x)
        list_y.append(y)

    if len(list_x) < 1 or len(list_y) < 1:
        return box
    elif len(list_x) > 9 and len(list_y) > 9:
        max_location = np.mean(sorted(list_x)[math.ceil(0.1 * len(list_x)):
math.ceil(-0.1 * len(list_x))]), np.mean(sorted(list_y)[math.ceil(0.1 *
len(list_y)): math.ceil(-0.1 * len(list_y))])
    else:
        max_location = np.mean(list_x), np.mean(list_y)

    box = (int(max_location[0] - w//2), int(max_location[1] - h//2),
           int(max_location[0] + w//2), int(max_location[1] + h//2))

    return box


def advanced_template_matching(template, accuracy=0.95, second_try=True):
    import cv2 as cv
    import numpy as np
    import imutils

    if isinstance(template, str):
        template = cv.imread(template, cv.IMREAD_GRAYSCALE)
        if template is None:
            print('Cannot read image, check cv2.imread() documentation')
            return None
    else:
        print('Invalid format of image')
        return None

    from PIL import ImageGrab

    screenshot = ImageGrab.grab(None)
    screenshot.save('screen.png')
    screenshot = cv.imread('screen.png', cv.IMREAD_GRAYSCALE)

    h, w = template.shape
    box = (-1, -1, -1, -1)

    found_dec = (0, 0, 0, 0)
    for scale in np.linspace(0.2, 1.0, 20)[::-1]:
```

```
        resized = imutils.resize(screenshot, width=int(screenshot.shape[1] *
scale))
        r = screenshot.shape[1] / float(resized.shape[1])

        if resized.shape[0] < h or resized.shape[1] < w:
            break

        result = cv.matchTemplate(resized, template, cv.TM_CCOEFF_NORMED)
        (_, maxVal, _, maxLoc) = cv.minMaxLoc(result)

        if maxVal > found_dec[0]:
            found_dec = (maxVal, maxLoc, r, result)
        else:
            break

    found_inc = (0, 0, 0, 0)
    for scale in np.linspace(0.04, 0.8, 19):
        resized = imutils.resize(screenshot, width=int(screenshot.shape[1] *
(1+scale)))
        r = screenshot.shape[1] / float(resized.shape[1])

        result = cv.matchTemplate(resized, template, cv.TM_CCOEFF_NORMED)
        (_, maxVal, _, maxLoc) = cv.minMaxLoc(result)

        if maxVal > found_inc[0]:
            found_inc = (maxVal, maxLoc, r, result)
        else:
            break

    if found_dec[0] > found_inc[0]:
        (maxVal, maxLoc, r, res) = found_dec
    else:
        (maxVal, maxLoc, r, res) = found_inc

    if maxLoc == 0:
        return box

    if maxVal >= accuracy:
       box = (int(maxLoc[0] * r), int(maxLoc[1] * r), int(maxLoc[0] * r) + w//2,
int(maxLoc[1] * r) + h//2)
    elif second_try and 0 < maxLoc[1] < res.shape[0]-1 and 0 < maxLoc[0] <
res.shape[1]-1:
        similarity = accuracy * 0.02
        val00 = res[maxLoc[1]-1][maxLoc[0]-1]
        val10 = res[maxLoc[1]][maxLoc[0]-1]
        val20 = res[maxLoc[1]+1][maxLoc[0]-1]
```

```
        val01 = res[maxLoc[1]-1][maxLoc[0]]
        val21 = res[maxLoc[1]+1][maxLoc[0]]
        val02 = res[maxLoc[1]-1][maxLoc[0]+1]
        val12 = res[maxLoc[1]][maxLoc[0]+1]
        val22 = res[maxLoc[1]+1][maxLoc[0]+1]

        valid_score = 0
        if val10 - (val00+val20)/2 >= similarity:
            valid_score += 1
        if val01 - (val00+val02)/2 >= similarity:
            valid_score += 1
        if val21 - (val20+val22)/2 >= similarity:
            valid_score += 1
        if val12 - (val02+val22)/2 >= similarity:
            valid_score += 1
        if valid_score >= 2:
           box = (int(maxLoc[0] * r), int(maxLoc[1] * r), int(maxLoc[0] * r) +
w//2, int(maxLoc[1] * r) + h//2)

    return box
```