УДК 004.932.75'1

СИСТЕМА ТЕСТИРОВАНИЯ КОНТРОЛЛЕРОВ, ОСНОВАННАЯ НА РАСПОЗНАВАНИИ ТЕКСТА НА ЭКРАНЕ

А. А. Докукин[0009-0001-9928-7851]

Федеральный исследовательский центр «Информатика и управление» Российской академии наук, г. Москва, Россия

adokukin@frccsc.ru

Аннотация

Описано решение задачи тестирования контроллеров на основе чтения информации с их экрана. Для этого разработана программно-аппаратная система, состоящая из камеры и программных модулей, реализующих необходимые алгоритмы и методы: модуля предобработки изображения; модуля определения типа меню; модуля обработки символов шрифта; модуль чтения текста, в том числе, написанного различными шрифтами; собственно модуля тестирования. Система реализована для контроллеров определенного типа с монохромным дисплеем 128 x 64 точек. Все методы реализованы на языке Python с использованием популярных библиотек. Система внедрена в эксплуатацию и на данный момент осуществляет автоматизацию нескольких наиболее трудоемких тестов. Поддерживается расширение их набора в виде плагинов.

Ключевые слова: *компьютерное зрение, распознавание текста, тестирование контроллеров.*

ВВЕДЕНИЕ

При разработке различных контроллеров, таких как контроллеры «умного дома» или контроллеры отопления, возникает задача тестирования встраиваемого в них программного обеспечения (ПО). Контроллерами принято называть универсальные устройства, реализованные на базе слабого процессора, с очень ограниченными другими ресурсами, и способные при этом непосредственно управлять исполнительными механизмами, такими как насосы, сервоприводы и т. п. При этом контроллеры реализуют достаточно сложную логику. Так, например, контроллеры отопления Smartweb [1] работают в составе распределенной

сети, каждый контроллер в которой реализует часть общей функциональности, при этом отправляя мониторинговые данные на облачный сервер.

При автоматизации тестирования контроллеров возникают различные проблемы. Процедуры отслеживания результатов обработки длинных сценариев невозможно разместить на контроллере вместе с его собственным ПО из-за ограниченности ресурсов: оперативной и постоянной памяти. Частичная эмуляция работы контроллера возможна на более мощных устройства, таких как персональный компьютер. Но в этом случае исключено взаимодействие с реальным экраном устройства, его файловой системой, исполнительными устройствами. Поэтому при проверке ПО реальных контроллеров значительные усилия ложатся на плечи людей-тестировщиков. От них требуются наблюдение и вмешательство в длительные сценарии, исполняемые непосредственно в сети контроллеров. При многократном исполнении тестовых сценариев внимание людей притупляется, поэтому возникает задача автоматизации рутинных операций такого типа более сложными методами, например, на основе машинного распознавания информации с экрана контроллера.

В настоящем исследовании рассмотрены реальные контроллеры серии Smartweb K [1] с дисплеем, его целью является поиск комбинации технических средств и алгоритмов для чтения информации с экрана контроллера и построение системы тестирования на основе полученных данных [2, 3].

Устройства обладают монохромным экраном размером 128 x 64 пикселя, изображение на котором формируется по определенным правилам. Тип меню, выбранного на контроллере, определяет состав отображаемой информации: положение строк текста и графических элементов. Задача естественным образом разбивается на две подзадачи: определение текущего типа меню и распознавание графических элементов в соответствии с шаблоном, в том числе чтение текста.

Такой подход обладает своими ограничениями, т. к. не позволяет обнаружить наличие символов за пределами полей шаблона, что легко дается человеку. Тем не менее он покрывает подавляющее число реальных сценариев тестирования.

ОПРЕДЕЛЕНИЕ ТИПА МЕНЮ

Контроллеры серии Smartweb K обладают достаточно простым монохромным дисплеем размером 128 на 64 пикселей, поэтому чтение информации с него представляется несложным. Однако эта простота не отменяет необходимости получать изображение экрана. В ходе настоящего исследования используемые для этого технические средства менялись, первый стенд был собран на основе бытовой веб-камеры из высшего ценового сегмента. Установка и получаемое изображение показаны на рис. 1. Как видно, несовершенство установки порождает дополнительную задачу – определение реального положения экрана на изображении камеры. Центровка и масштабирование изображения с помощью штатива и ПО веб-камеры не представляются возможным.

Рис. 1. Тестовый стенд на основе веб-камеры и получаемое изображение.

На рис. 2 представлены изображения некоторых из 49 различных типов меню, отображаемых контроллером. Как видно, они достаточно сильно различаются, но эти различия трудно описать, поэтому их классификация является идеальной задачей для нейронной сети. Единственным препятствием для ее применения является необходимость сбора обучающей выборки. Поскольку вручную переключать экраны контроллера и фотографировать их нет возможности, была написана программа генерации правдоподобных изображений различных меню со случайным заполнением. Это оказалось наиболее трудоемкой операцией данного этапа. Часть полученных изображений и представлена на рис. 2.

Рис. 2. Примеры различных типов меню контроллера из обучающей выборки.

Для классификации типов меню была подобрана трехслойная сверточная нейронная сеть [4] следующей конфигурации: (1 -> 32, 3 x 3), (32 -> 16, 3 x 3), (16 -> 49, 60 x 124). Здесь первая пара чисел – это число входных и выходных слоев, а вторая – размер ядра. Первые слои достаточно стандартные, особого внимания заслуживает последний. Число выходных слоев в нем равно числу классов, т. е. типов меню, а размер ядра приводит к тому, что результатом свертки изображения экрана становится одно число. Сеть была реализована с помощью библиотеки pytorch [5], очень быстро обучается (десятки секунд на ноутбуке без дискретной видеокарты) и еще быстрее работает. При обучении на 100 экземплярах изображений каждого класса точность распознавания тестовой выборки составляет 100%. Для теста были использованы по 10 изображений каждого класса.

Кроме того, построенная архитектура позволяет определять положение экрана контроллера на более широком поле. Для второго теста сгенерированные изображения располагались на более широком фоне (256 x 128) в случайной позиции. На выходе нейросети получается тензор размера 49 x 129 x 257, максимум значений которого соответствует классу и смещению экрана (использована нотация библиотек PIL и pytorch, в которых первая пространственная координата соответствует *x*, если речь идет про изображения, и *y*, если про тензоры). Тесты при том же размере выборок также показали 100%-ную точность (рис. 3).

Отметим также, что, эта архитектура сети, примененная к нефильтрованному изображению с камеры, дала отличный результат по стабильности опреде-

ления координат на разных типах экранов и скорости обработки потокового видео.

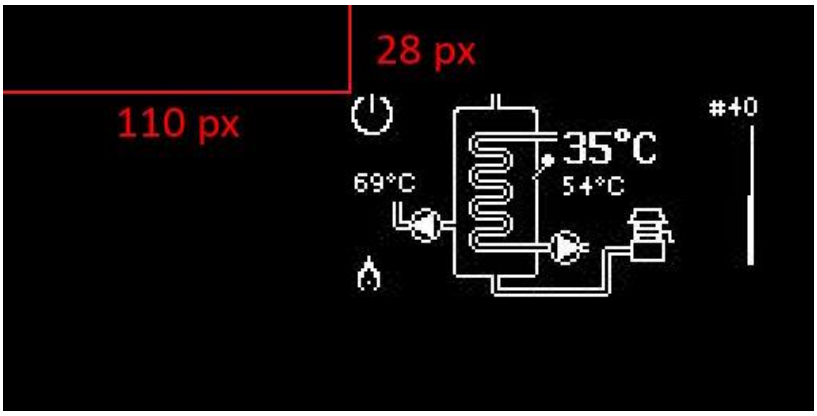


Рис. 3. Совместное определение класса меню и смещения экрана.

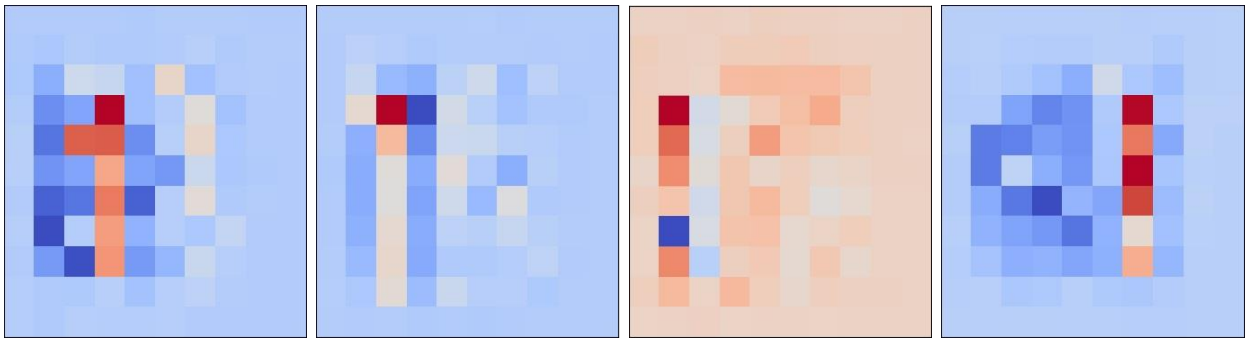
Наконец, эту же архитектуру можно использовать для определения масштаба изображения. Из-за того, что необходимо перебирать некоторый диапазон масштабов, эта процедура становится слишком трудоемкой для использования в реальном времени, но масштаб остается неизменным в течение тестовой сессии, поэтому такую калибровку можно производить один раз при запуске системы.

ОПРЕДЕЛЕНИЕ ОТДЕЛЬНЫХ СИМВОЛОВ

Задание типа меню однозначно определяет расположение на экране и содержимое полей. Это могут быть надписи, выполненные одним из трех шрифтов, либо иконки. Иконки, в свою очередь, могут быть статическими или динамическими, но последние представляют собой сменяющие друг друга статические. Таким образом, определение содержимого полей сводится к определению символов одного из четырех перечисленных алфавитов. Задача классификации таких символов, с одной стороны, упрощается по сравнению с задачей определения типа меню, т. к. классифицируемые изображения значительно уменьшаются. Вместо целых экранов размером 128 x 64 точек рассматриваются символы, размер которых имеет порядок 10 x 15. С другой стороны, задача усложняется, потому что значительно увеличивается число классов, а символы разных классов могут отличаться буквально на одну точку, например, «i», «l» и «!». Но самое главное: части символов могут совпадать с другими значками, и совместная класси-

фикация и определение их расположения становятся слишком затруднительными. Поэтому было решено использовать раздельно четыре нейронные сети: три шрифтовые и одну для иконок. Текстовые поля при этом ограничиваются по высоте размерами соответствующего алфавита. Ниже описаны исследования по обработке одного из шрифтов – 8-пиксельного.

Были рассмотрены две архитектуры: классическая сверточная нейронная сеть и нейросеть без обучения, разработанная специально для этой задачи. В качестве сверточной рассмотрена нейронная сеть, схожая с сетью, описанной в предыдущем разделе, с поправкой на размеры характерных изображений и число классов. Для ее обучения были использованы символы алфавита, по одному примеру для символа, и они же использовались для теста. Даже в такой простой постановке сеть не дала необходимого стопроцентного качества распознавания, причина чего становится понятной после исследования областей интереса на отдельных символах (рис. 4).



а) 1 б)] в) ! г) Ы

Рис. 4. Визуализация признаков классификации некоторых символов.

Полученные результаты показывают, что размеры выборок недостаточны для определения действительно характерных сочетаний в результате обучения сети. Для ее использования представляется необходимым разнообразить выборку с помощью различных аугментаций, таких как маскирование определенных областей. Но это также связано с трудностями по причине значительной похожести отдельных символов.

В то же время простота символов позволяет проводить непосредственное поточечное их сравнение. В целом использование обучения представляется из-

быточным для этой задачи. Однако удобство средств, предоставляемых библиотекой pytorch для работы со сверточными сетями, и скорость обработки изображений с их помощью являются важными практическими преимуществами. Это легло в основу схемы нейросетей без обучения, т. е. нейросетей, в которых веса вычисляются аналитически, минуя стадию обучения.

Для рассматриваемого шрифта была предложена однослойная схема (1 -> 143, 11 x 10), где 143 – число различных используемых символов, 8 x 11 – размер самой большой буквы, и 2 точки – пробелы вокруг. Работа сверточного слоя была модифицирована для работы по формуле $(2X - 1) \otimes T$, где X – изображение, T – маска буквы (1, если точка принадлежит букве, -1 – фон и отступы, 0 – за пределами ширины буквы), нормированная на ширину. Примеры таких масок для символов представлены на рис. 5 (можно сравнить их с результатами обучения на рис. 4).

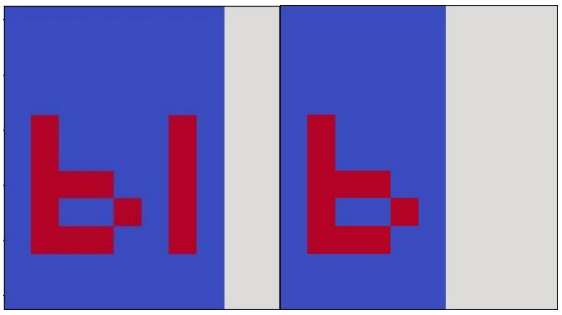


Рис. 5. Результаты вычисления ядер свертки для различных символов.

В отличие от схемы с обучением, данная схема дала 100%-ное качество при определении единственного символа. В неизвестной позиции в строке, т. е. в изображении символа, дополненного пустым фоном слева и справа, точность составила 99.3%, т. к. путаницу «I» и вертикальной черта «Ы» преодолеть не удалось. При определении же пяти случайных букв подряд точность составила около 97%. Ошибка с вклиниванием несуществующей буквы в последовательность дает сдвиг остальным и портит их. Например, в последовательности «ВмыЧ,» уверенно определилась буква «ь», что дало «мььЧ,» в качестве результата. Исправление этого эффекта рассмотрено ниже.

Тесты с использованием реальных изображений текста, полученных с камеры, выявили еще одну принципиальную сложность. Невозможность точного

позиционирования камеры по нескольким осям приводит к едва заметным перспективным искажениям, которые влияют на точность: изображения текста из некоторых областей экрана перестают распознаваться, поскольку характеристики символов не оставляют никакой толерантности к искажениям. В следующем разделе описаны алгоритмы предобработки изображения, позволяющие избегать этого эффекта.

ПРЕДОБРАБОТКА ИЗОБРАЖЕНИЯ

Классификация типов меню оказалась достаточно устойчивой к мелким перспективным искажениям экрана за счет сложности и размера изображения. Это не относится к распознаванию отдельных символов, поскольку несовпадение даже одной точки может привести к полному изменению результата классификации. Для устранения искажений был предложен простой алгоритм: матрица перспективного преобразования восстанавливается по четырем угловым точкам экрана, на ее основе рассчитывается расположение всех точек растра и затем выбирается ближайший к расчетному положению пиксель. Пример искажения, гипертрофированного для наглядности, и восстановленного растра приведен на рис. 6.

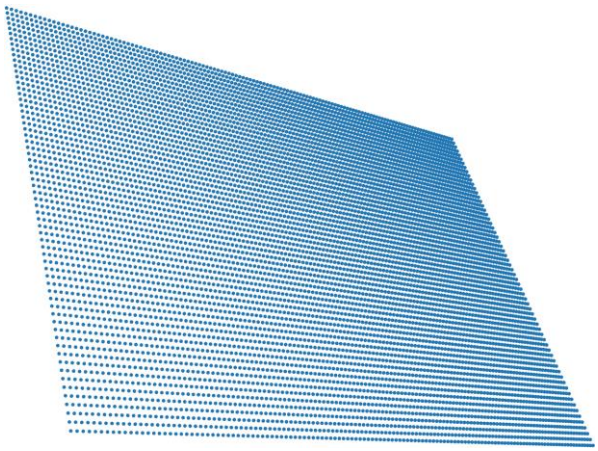


Рис. 6. Пример восстановленного растра.

Восстановление матрицы перспективного преобразования $A = \|a_{ij}\|_{3\times3}$ производится с помощью решения системы линейных уравнений:

$$a_{11}x_i + a_{12}y_i + a_{13} = r_i x_i',$$
$$a_{21}x_i + a_{22}y_i + a_{23} = r_i y_i',$$

$$a_{31}x_i + a_{32}y_i + 1 = r_i,$$

где (x_i, y_i) – координаты точки растра, (x_i', y_i') – координаты ее проекции на изображении. Для четырех точек, $i = \overline{1,4}$, система имеет единственное решение.

Еще одной проблемой является засветка соседних точек, особенно при увеличении выдержки для борьбы с мерцанием. Один из худших примеров такой засветки – это круги в символе процента (рис. 1), где черная точка находится в окружении 7 светящихся, и ее яркость значительно отличается от других точек фона. Для преобразования в бинарный вид картинки, получаемой с камеры, с устранением этой проблемы мы использовали метод Оцу [6], реализованный в библиотеке OpenCV для языка питон [7]. Заметим, что эта библиотека интенсивно используется в системе и заслуживает отдельного упоминания.

Более серьезную проблему представляло мерцание камеры, источником которого, по всей видимости, были даже не ее физические характеристики, а внутренние алгоритмы коррекции изображения, которые в сочетании с мерцанием экрана приводили к постоянной его деградации, вплоть до невозможности обработки через полчаса съемки. Эту проблему удалось решить с помощью замены камеры на промышленную камеру стандарта GigE Vision. На рис. 7 показаны внешний вид усовершенствованного стенда и полученное с камеры изображение меню, аналогичное рис. 1.

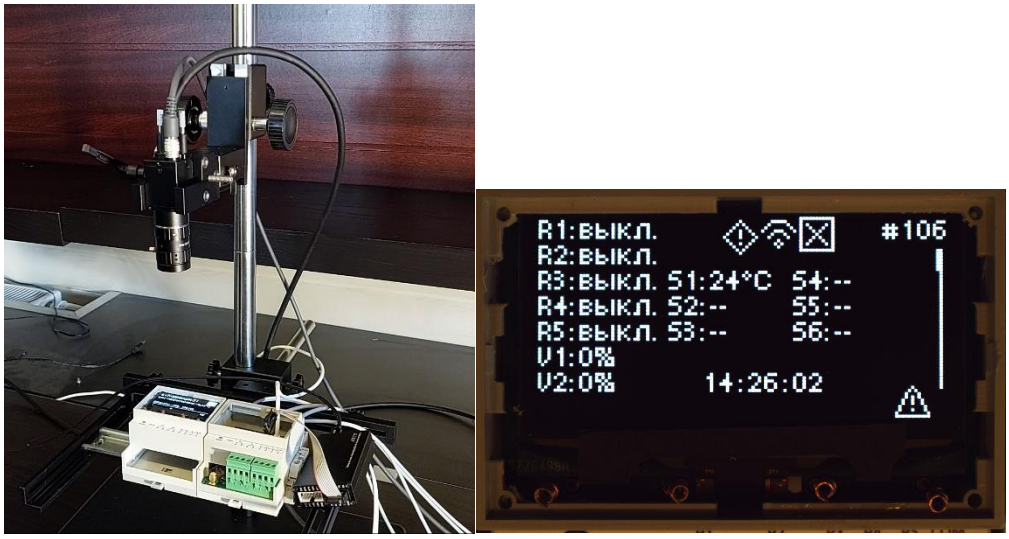


Рис. 7. Тестовый стенд на основе промышленной камеры и пример изображения.

Смена камеры решила проблемы с мерцанием – она работает независимо

от компьютера, и получаемое изображение не меняется месяцами. Кроме того, увеличилась четкость изображения и пропали проблемы с засветкой. Но новый штатив снял проблему позиционирования лишь частично, модуль устранения перспективных искажений был оставлен в системе. Устранения более сложных дисторсий, таких как бочкообразная, не потребовалось даже для веб-камеры. Бинаризация методом Оцу также была включена в последнюю версию системы.

РАСПОЗНАВАНИЕ СМЕСИ ШРИФТОВ

Методы, описанные выше, позволяют надежно определять изображенные на экране иконки и «читать» текст с экрана контроллера. Для этого используются некоторые простые эвристики, устраняющие неоднозначности шрифтов. Прежде всего, полное включение узкого кандидата в более широкий («Ь» в «Ы») трактуется в пользу более широкого. Похожие допущения используются при покрытии кандидата парой соседних. Особенно это актуально при подсчете числа пробелов между символами.

Однако такой подход работает только с полями, содержащими единственный шрифт. В реальности же на контроллере используются надписи из смеси шрифтов, например, числовой индекс меню и его название имеют разный размер (рис. 2, слева внизу). В этой ситуации граница между шрифтами заранее не известна. Более того, длинные названия могут прокручиваться горизонтально, и индекс с подписью могут поменяться местами или даже образовать две границы. Поэтому необходимо иметь возможность распознавать такие тексты в общем виде. Это порождает дополнительные сложности. Прежде всего отменяется сделанное ранее предположение, что все надписи содержатся в строках высотой с размер шрифта, что возрождает проблему схожести частей одних символов на другие. Похожая неприятность возникает из-за схожести символов разных шрифтов. Так, прописная «Р» маленького шрифта не отличается от строчной «р» большого. Единственное различие – это высота строки, на которой определяется символ. Ножка строчной «р» свешивается ниже базовой линии. Это различие использовалось при разборе текста. Предполагалось, что внутри полей находятся

строки символов и соседние элементы должны поддерживать выбор альтернативы. Описанная идея представлена на рис. 8. Красным цветом показана потенциальная строка, если в голубой рамке определяется двоеточие, синим цветом – точка. Для наглядности каждая строка показана только с одной стороны от рамки.

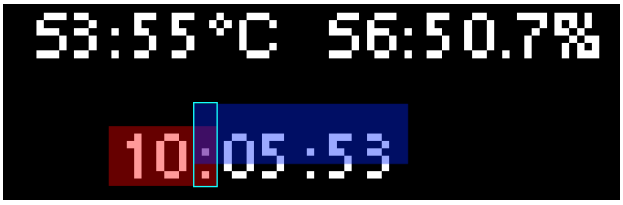


Рис. 8. Влияние соседних символов на выбор альтернативных гипотез.

Эту идею можно реализовать разными способами. В рамках проведенного исследования были осуществлены и проверены на практике два варианта ее реализации. Первый из них продолжает эксплуатировать концепцию нейросетей без обучения, использованную ранее для распознавания отдельных символов. Рассмотрим матрицу изображения $P_{H\times W}$, где H – его высота, W – ширина. «Шрифтовая» нейросеть F для шрифта высотой h и с числом символов n преобразует эту матрицу в тензор $T_{n\times H-h\times W} = F(P)$, в котором значение $T(k, y, x)$ соответствует оценке наличия символа с номером k в строке y и позиции x (положение определяется верхним левым пикселем). Аналогично можно работать со смесью шрифтов, для чего нужно объединить по первой координате тензоры $T_i = F_i(P)$ для нескольких шрифтов, дополнив пространственные измерения нулями, где необходимо. Для простоты выкладок опишем случай одного шрифта.

К полученному тензору T применим трехмерный [5] сверточный слой U с одним входным каналом, $n(F)$ выходными и ядром размера $n(F) \times 2w + 1$, где $n(F)$ – число символов в шрифте. Выходные каналы соответствуют символам шрифта и содержат поощрение этих символов от их соседей. В i-м канале j-я строка ядра содержит $2w - 1$ нулей и две единицы в позициях $w + 1 - w_j$ и $w + 1 + w_i$, поскольку поддержка левого символа зависит от его ширины, а правого – от ширины самого рассматриваемого символа. Тензор T далее заменим тензором $T + U\big(I(T)\big)$, где I зануляет все позиции T, содержащие не единицы. Дальнейшая обработка достаточно очевидна и основана на тех же эвристиках.

Второй вариант из упомянутых выше реализует ту же идею, но как некую вариацию метода ветвей и границ. Столбцы тензора T перебираются последовательно и величина поощрений рассчитывается только для максимальных элементов, т. е. элементов, чья оценка равна 1. Поддержка выражается в умножении оценки символа на оценку его левого соседа, после чего немаксимальные ветки отбрасываются.

Оба варианта были реализованы практически и показали одинаковое, стопроцентное качество распознавания тестовых примеров. Различие проявилось только в скорости обработки. Нейросетевой вариант оказался примерно на порядок медленнее из-за большего числа лишних вычислений. Так, распознавание текста из 26 символов на изображении размером 148 на 20 пикселей с помощью дополнительной свертки заняло 179 мс. Эффективность разогретой сети увеличивается – десять подобных изображений подряд обрабатываются 936 мс. При этом второй метод обрабатывает за 11 мс одно, а за 222 мс – десять изображений. Возможно, при использовании графических ускорителей соотношение изменится, но в текущих условиях для практического применения был выбран второй подход. Сочетание описанных компонентов позволяет проводить обработку изображения экрана на персональном компьютере без графического ускорения в реальном времени с частотой порядка 10 кадров в секунду.

ВСПОМОГАТЕЛЬНЫЕ СРЕДСТВА

Стоит отметить, что полученная система чтения содержимого экрана еще не равносильна системе тестирования контроллеров. Для воспроизведения сценариев пользовательского взаимодействия и проверки их результатов необходимо осуществлять осмысленную навигацию по меню. В распоряжении системы имеется несколько команд, соответствующих кнопкам контроллера: «вверх», «вниз», «внутрь» и «наружу». Но написание тестов на языке «15 раз нажать вверх, потом внутрь, …» попросту невозможно, даже если смириться с неудобством такого описания. Состав меню рассматриваемых контроллеров, а именно набор и порядок элементов, может изменяться динамически. Кроме того, активный пункт при входе в меню может выбираться непредсказуемо. Поэтому для навигации был

написан высокоуровневый язык, позволяющий управлять ей в терминах «перейди в меню калибровки датчика № 1» и «установи значение 10». Перемещение в меню при этом осуществляется системой автоматически на основе информации, отображаемой на экране.

Дополнительно реализован механизм подключения новых тестов к системе в виде плагинов. Детали реализации оставим за пределами настоящей статьи, поскольку к теме они относятся лишь косвенно.

ЗАКЛЮЧЕНИЕ

Представлена программно-аппаратная система для тестирования контроллеров на основе распознавания информации, отображаемой на их экране. Система содержит как общеизвестные, так и специально разработанные для нее алгоритмы и методы, выбор которых подтвержден результатами исследований.

В основе системы лежит обработка в реальном времени отображаемой на экране информации. После выделения экрана, устранения перспективных искажений и бинаризации изображения методом Оцу выполняется классификация типа меню, затем разбор соответствующих этому меню элементов изображения – отдельных символов и строк текста.

Созданная система введена в эксплуатацию и уже выполняет обработку наиболее трудоемких из ручных тестов. Разработан инструментарий их написания, дальнейшее масштабирование системы является чисто программистской задачей.

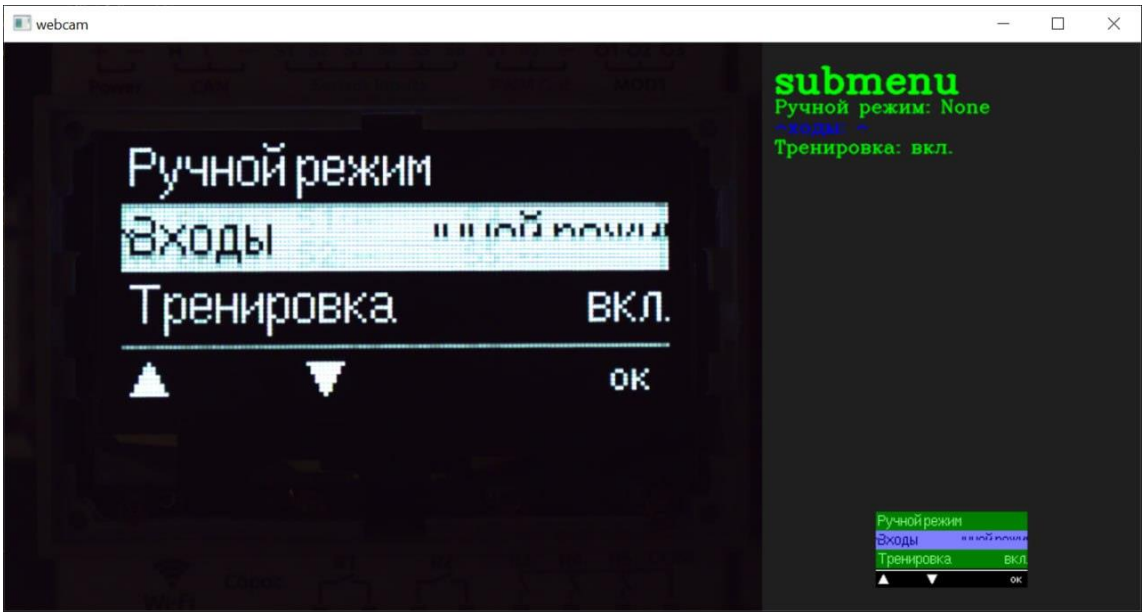


Рис. 9. Пример сбоя с появлением неожиданных символов
и результат его обработки.

Открытым принципиальным вопросом остается поиск неожиданных артефактов. Система справляется с проверкой содержимого заданных полей на соответствие известному шаблону. Появление символов в неожиданных местах или неизвестных системе символов остается незамеченным при таком подходе. Обработка нештатных ситуаций такого типа возможна лишь косвенно. Реальный пример показан на рис. 9.

На рисунке представлены реальная фотография экрана, полученная камерой в системе, и результат ее обработки. Как видно, система не смогла прочитать букву «В» в слове «Входы» и полностью проигнорировала мусорные символы справа. Ситуацию удалось обнаружить и исправить благодаря сбою навигации и привлечению оператора.

Вместе с тем для человека сбой достаточно очевиден, и даже в какой-то степени понятна его причина – эхо верхней строчки отобразилось ниже. При некотором уточнении эта картинка даже позволяет с большой надежностью исключить аппаратный сбой. Дальнейшие исследования предполагается направить на автоматизацию обработки нештатных ситуаций такого типа и, в первую очередь, на отслеживание появления графических элементов, которые можно трактовать как символы из неизвестного алфавита, или корректных символов за пределами полей шаблона.

СПИСОК ЛИТЕРАТУРЫ

1. SmartWEB-K // [Электронный ресурс]
URL: https://www.teplostart.ru/download/ booklets/SmartWEB-K.pdf (дата обращения: 06.10.2025)

2. *Докукин А.А.* О построении системы считывания информации с экрана контроллера // Интеллектуализация обработки информации (ИОИ-2024): Тез. докл. 15-й междунар. конф. (Гродно, 23–27 сент. 2024 г.) 2024. С. 92–93.

3. *Докукин А.А.* Развитие системы считывания информации с экрана контроллера // Математические методы распознавания образов (ММРО-2025): Тез. докл. 22-й Всеросс. конф. с междунар. участ. (Муром, 22–26 сент. 2025 г.) 2025.

C.139–140.

4. *Le Cun Y., Boser B., Denker J. S., Henderson D., Howard R. E., Hubbard W., Jackel L. D.* Handwritten Digit Recognition with a Back-Propagation Network // NIPS'89: Proc. 2nd Int. Conf. Neural Information Processing Systems. 1989. P. 396–404.
URL: https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf

5. *Ansel J., Yang E., He H. et al*. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation // 29th ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '24). 2024. Vol. 2. https://doi.org/10.1145/3620665.3640366

6. *Otsu N.* A Threshold Selection Method from Gray-Level Histograms // IEEE Transact. on Systems, Man, and Cybernetics 9.1. 1979. P. 62–66. https://doi.org/10.1109/ TSMC.1979.4310076

7. *Bradski G.* The OpenCV Library // Dr. Dobb's J. Software Tools. 2000. Vol. 120. P. 122–125.

A SYSTEM FOR TESTING CONTROLLERS BASED ON ON-SCREEN TEXT RECOGNITION

A. A. Dokukin[0009-0001-9928-7851]

Federal Research Center "Informatics and Control" of the Russian Academy of Sciences, Moscow, Russia

adokukin@frccsc.ru

Abstract

A solution for the problem of testing controllers based on reading information from their screens is described. A hardware and software system has been developed for this purpose, consisting of a camera and software modules implementing the necessary algorithms and methods: an image preprocessing module; a menu type detection module; a font character processing module; a text reading module, including one written in various fonts; and the testing module itself. The system has been developed

for a specific type of controller with a monochrome 128x64 pixel display. All methods are implemented in Python using popular libraries. The system has been launched into test operation and currently automates several of the most labor-intensive tests. The test set can be expanded using plugins.

Keywords: *computer vision, text recognition, controller testing.*

REFERENCES

1. SmartWEB-K (In Russian) // [Electronic resource] https://www.teplostart.ru/download/booklets/SmartWEB-K.pdf (access date: 06.10.2025)

2. *Dokukin A.A. O postroenii sistemy schityvaniya informacii s ekrana kontrollera* (in Russian) // Intellectual Data Processing (IDP-2024): Book of Abstracts 15th Int. Conf. (Grodno, September 23–27, 2024) 2024. P. 92–93.

3. *Dokukin A.A. Razvitie sistemy schityvaniya informacii s ekrana kontrollera* (In Russian) // Mathematical Methods for Pattern Recognition (MMPR-2025): Book of Abstracts 22nd Russian Conf. with Int. Participation (Murom, September 22–26, 2025) 2025. P. 139–140.

4. *Le Cun Y., Boser B., Denker J. S., Henderson D., Howard R. E., Hubbard W., Jackel L. D.* Handwritten Digit Recognition with a Back-Propagation Network // NIPS'89: Proc. 2nd Int. Conf. Neural Information Processing Systems. 1989. P. 396–404. https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf

5. *Ansel J., Yang E., He H. et al*. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation // 29th ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '24). 2024. Vol. 2. https://doi.org/10.1145/3620665.3640366

6. *Otsu N.* A Threshold Selection Method from Gray-Level Histograms // IEEE Transact. on Systems, Man, and Cybernetics 9.1. 1979. P. 62–66. https://doi.org/10.1109/ TSMC.1979.4310076

7. *Bradski G.* The OpenCV Library // Dr. Dobb's J. Software Tools. 2000. Vol. 120. P. 122–125.

СВЕДЕНИЯ ОБ АВТОРЕ

ДОКУКИН Александр Александрович – 1980 года рождения, старший научный сотрудник ФИЦ ИУ РАН, кандидат физико-математических наук. Закончил с отличием факультет ВМК Московского государственного университета им. М.В. Ломоносова в 2002 году. В 2005-м закончил аспирантуру того же факультета. В 2008-м защитил диссертацию на соискание степени кандидата физ.-мат. наук по теме «Синтез полиномов над экстремальными алгоритмами вычисления оценок». С 2000-го года по настоящее время работает в ВЦ РАН (затем ФИЦ ИУ РАН). Область научных интересов – распознавание образов, анализ данных. Опубликовал лично и в соавторстве 139 научных работ.

Aleksandr Aleksandrovich DOKUKIN graduated with honors from the Faculty of Computational Mathematics and Cybernetics at Lomonosov Moscow State University in 2002. In 2005, he completed his postgraduate studies at the same faculty. In 2008, he defended his dissertation for the degree of Candidate of Physical and Mathematical Sciences on the topic "Synthesis of Polynomials over Extremal Estimation Algorithms." From 2000 to the present, he has worked at the Computing Center of the Russian Academy of Sciences (later the Federal Research Center “Computer Science and Control” of the Russian Academy of Sciences). His research interests include pattern recognition and data analysis. He has published and co-authored 139 scientific papers.

email: adokukin@frccsc.ru

ORCID: 0009-0001-9928-7851

Материал поступил в редакцию 3 ноября 2025 года