Abstract

In education, the effortless analysis of test results is essential to teacher effectiveness. Instructors across all levels have long utilized the traditional hardware-based OMR approach to fulfill this need. While this approach is proven and demands little effort from the instructor, it guarantees financial inconveniences. Costs begin immediately at the university level with the purchase and maintenance of the OMR scanner. Furthermore, the costs shift to the student having to purchase the answer sheet required for use with the school's specific hardware-based OMR scanner for each test. These financial challenges can be particularly burdensome for instructors and students with limited access to resources and immediate funding.

This paper introduces LiveTest, a highly customizable and configurable testing solution. LiveTest implements Pictron, which facilitates the creation of highly customizable answer sheets that can be graded using our dynamic OMRGrader module. Instructors can utilize any web browser to submit images of completed student answer sheets to a predefined test for grading. The test results and various analytics are available in real-time during the test, showing the "live" shifting of test data as the instructor adds more submissions. Additionally, the paper describes and analyzes the process of extracting the answer sheet from the image, standardizing the answer sheet, identifying the chosen answers, grading the test, and archiving the graded artifacts.

Introduction

Optical mark recognition (OMR) is the process of scanning a paper to determine the presence of a marking in a predetermined location. Instructors of all levels use OMR answer sheets to efficiently and accurately grade MCQ tests. A strong shift in education toward free-response questions has seen a shift away from MCQ, causing a lack of robust open-source MCQ technology and a shift toward training handwriting AI models [1]. However, more recently, due in part to strides made in computer vision, software-based OMR solutions are becoming more mainstream. Generally, these software solutions lack accessibility and cost too much to justify them for the average school [4].

Students have filled in tiny bubbles on answer sheets to perfection and handed them in after the test to their instructor for years. The instructor then takes the answers sheets to the scanner, pushes them through one by one, and uploads the results to a USB once finished. A particular professor had given a test and brought the completed answer sheet forms to the school's hardware-based OMR scanner but realized the scanner was broken. With the school having no immediate plans for fixing the scanner, he realized he would have to do it by hand. Checking and analyzing tests by hand can take up to ten times longer than software-based OMR [3]. Obviously, the professor did not want to grade these by hand and began work on using OpenCV to extract the results. The first attempts did not go well because of the structure of the sheet; the answer bubbles were not closed contours, the shading within each answer bubble was irregular, and too much time was spent trying to get it to work for that one test. Instead, the development of Pictron and OMRGrader was pursued.

LiveTest aims to provide intuitive, customizable, and completely free software-based OMR scanning. Several existing public repositories implement OMR scanning in various ways, but none provide the level of automation seen in paid apps. [5, 6, 11]

Related Works

Two existing, proven, software-based OMR solutions like LiveTest are publicly available. Omrquiz.com is a robust set of tools that are available to instructors free on the internet. One can create answer sheets freely with 3-5 choices and 1-200 questions to be graded later. This service is free if you just want the results of each answer sheet but involves nothing past this [6]. ZipGrade is the other publicly available software-based OMR application that allows for free usage with up to 100 monthly graded answer sheets. This is a tested, proven, enterprise-level software-based OMR. It has pre-built 20, 50, and 100 question-answer sheets that teachers can use, or they can build their custom answer sheets to be graded. The paid version allows for over 100 graded tests a month, and the paid version is a simple $7 a year [5]. This cost is highly affordable and could be a timesaver for the average instructor, but the costs should not fall on the instructor. Schools should evolve and adapt software-based OMR systems. However, not all of the systems are built for schools. Many are built for individual instructors, such as the two previously mentioned. When one looks at the OMR solutions that are multi-purpose and built for school-wide use, such as Remark Office, the annual cost for a single license for a single instructor is $1300, making this infeasible for most educators without the help of the school [11].

The best open-source library on GitHub for scanning diverse OMR documents is OMR Checker by Udayraj123. It implements a very robust process for general-purpose OMR [15]. You can configure the “OMR layout” with the app and then feed images to it individually, receiving the grades. Had I found it earlier, it would’ve been handy.

At the core of LiveTest, the system proposed and developed, are Pictron and OMRGrader, two custom-built Python modules that utilize well-known third-party libraries such as Pillow (a friendly fork of PIL), OpenCV, and NumPy. The rest of this paper will explain the two LiveTest components. Pictron was defined after finding little to nothing regarding producing customizable answer sheets in open source. OMRGrader was implemented due to the need to grade the new custom answer sheets.

Customization of answer sheets with Pictron begins with the configuration. Firstly, a user starts by defining the number of questions and choices. Then, they specify the bubble size, line spacing, answer spacing, font size, label spacing, and column width to their specific needs. Getting it to their needs is simply a matter of playing with the configuration. The bubble shape can also be configured as a circle, square, rectangle, or ellipsis. There is also the option to specify the course/test name and a different font for the answer sheet creation process. The answer sheet's logo can also be changed from the LiveTest logo to whatever they want. Even the alignment images in the four corners of the answer sheets can be changed.

```
pictron = Pictron(**info)
pictron.generate(course_name="Perf Course", test_name="TEST")
pictron.saveImage(outPath=perfTest, outName=f"{choice}-{count}")
```

```
info = {
"page_size": [
    8.5,
    11
],
"img_align_path": "checkerboard_144.jpg",
"logo_path": "images/LiveTestLogo_144x.png",
"bubble_shape": "circle",
"bubble_ratio": 1,
"font_path": "fonts/RobotoMono-Regular.ttf",
"font_bold": "fonts/RobotoMono-Bold.ttf",
"page_margins": [
    300,
    100,
    100,
    50
],
"font_alpha": 50,
"font_size": 5,
"bubble_size": 11.5,
"line_spacing": 22.5,
"column_width": 45,
"answer_spacing": 14,
"label_spacing": 15,
"num_ans_options": 7,
"num_questions": 200
}
```

Figure 1. Pictron initialization (top) configuration (left) resulting image (right)

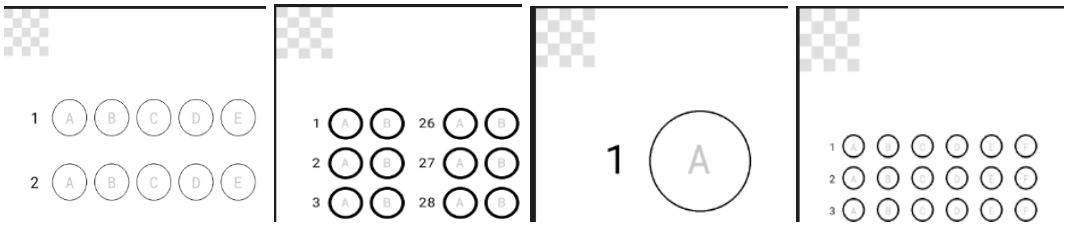


Figure 2. Examples of dynamic answer bubble placements.

The two relevant code snippets shown in figure three, taken from addAnswerBubbles, show the dynamic process. The addAnswerBubbles method starts by initializing x and y to the desired starting location, i, the number of bubbles placed, to zero, and n, the question number, to 1. The option set width is then calculated (line 414) to set the next column start_x. Now, we can start the loop that places all the answer bubbles. We must begin the loop (line 418) by checking to see if we're starting on a new question (line 420). If so, check if the y value is at the bottom and has no more room on the page (line 422) and start a new column (lines 423-425) if needed. Regardless, we will generate the question label (line 426) and then add it to start the new row (line 442). Once we have placed the question number, x is incremented by the label spacing (line 445). Then, the answer bubble is placed at location (x,y) with the appropriate answer label (lines 448-452). After placing the answer bubble, x is incremented by the sum of the bubble width and answer spacing (line 455). A check to see if our last answer choice was placed for this particular question is placed at the end (line 458) to reset x to the start x, increment y by the desired line spacing, and increment n by 1 (lines 459-461). Finally, at the end of the while loop, i is incremented (line 462) by one as we have placed a single answer bubble. This loop will repeat over and over until n reaches the set number of questions, which will signify that the answer sheet has been produced.

```
410         x = start_x
411         y = start_y
412         i = 0  # Overall count for number of answer choices placed
413         n = 1  # Question number
414         option_set_width = (self.bubble_width + self.answer_spacing) \
415             * self.num_ans_options + self.column_width
416
417         # begin outputting answer choices
418         while n <= self.num_questions:
419             # check to see if we are on to the next question
420             if i % self.num_ans_options == 0:
421                 # check if starting a new column is needed only when new answer is being created
422                 if y + self.bubble_height > self.img_height - self.page_margins[2]:
423                     start_x += option_set_width + self.label_spacing  # Shift to next column
424                     x = start_x
425                     y = start_y
426                 question_label = f"{n:>3}"

441                 # add question number to start off the new row
442                 self.addBubbleLabel(x - self.label_spacing, y, question_label)
443
444                 # dynamically allocate necessary spacing between question number and first answer choice.
445                 x += len(question_label) + ((self.font_size_adj // 2) * 3) + (self.bubble_width  // 2)
446
447             # add answer choice - determine if it will be filled or not
448             answer_label = chr((i % self.num_ans_options) + 65)  # A, B, C, etc.
449             self.addBubble(x, y, filled=(i % self.num_ans_options) == fill_answer,
450                            line_thickness=self.line_thickness)
451             self.addBubbleLabel(x, y, answer_label, (200, 200, 200)) \
452                 if (i % self.num_ans_options) != fill_answer else None
453
454             # increment x to place the next answer_choice
455             x += self.bubble_width + self.answer_spacing
456
457             # check to see if we placed all answer choices for this question.
458             if (i + 1) % self.num_ans_options == 0:
459                 x = start_x
460                 y += self.bubble_height + self.line_spacing
461                 n += 1
462
463             i += 1
```

Figure 3. Pictron.addAnswerBubbles ability to build answer sheets of varying sizes.

These dynamic configuration options allow educators to extensively customize their answer sheets, potentially creating new ones for every test or customizing them for students with disabilities. Figure four shows some more examples of answer sheets produced by Pictron.

Figure 4. More example of answer sheets produced by Pictron

OMRGrader is the component that implements the methods necessary for the dynamic grading process of the answer sheets produced by Pictron. First the steps of the process are described and then shown with detailed examples. To initialize the OMRGrader for usage shown in figure (figure 5 lines 541-546), begin by confirming the test's number of questions, choices, and whether or not the image is mechanically produced by Pictron or an actual submission photo. You can also specify whether to show the grading process by providing a step-by-step walk-through using OpenCV's show image method. The run method (lines 547-551) requires the actual image path or bytes object to be graded and the answer key as a list of key-value pairs. After a successful run, the grade of that submission, the graded results, and the choices dictionary are returned (figure 5).

```
541  grader = OMRGrader(
542      num_choices=4,
543      num_questions=100,
544      mechanical=False,
545      show_process=True
546  )
547  grade, graded, choices = grader.run(
548      file_path=f"submissionSheets/100-4/IMG_9348.png",
549      key = {
550          '1': 'A',
551          '2': 'B',
```

```
grade: 14.0
graded: {1: False,
rue, 12: True, 13:
2: False, 23: False
alse, 33: False, 34
, 43: False, 44: Fa
alse, 54: False, 55
64: False, 65: Fals
alse, 75: False, 76
85: False, 86: Fals
alse, 96: False, 97
choices: 100
```

Figure 5. OMRGrader initialization and run (left), run method return values (right)

The run method in OMRGrader is the helper function that runs the entire process of grading images of answer sheets produced by Pictron. It starts by isolating the document. To isolate the answer sheet from the image, we load the original image into a Python cv2 Matlike object and pre-process the image. The pre-process function shown in figure 6 prepares an image for contour detection by converting it to grayscale and applying Gaussian blur to smooth it out. It then uses OTSU's method to perform thresholding, inverting the result to highlight the edges. Canny edge detection is applied to identify the edges in the image. Finally, the detected edges are dilated to enhance their prominence and reduce background noise [7], making the edges of the

answer sheet more distinct for further processing. The dilation is specifically for dealing with potential splotchy brightness surrounding the answer sheet in the image.

```
71      gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
72      blur = cv2.GaussianBlur(gray, (5, 5), 0)
73      thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
74      edges = cv2.Canny(thresh, 50, 150)
75      kernel = np.ones((5, 5), np.uint8)
76      dilated = cv2.dilate(edges, kernel, iterations=2)
77      return dilated
```

Figure 6. Pre-Processing of submitted image

Once the image is pre-processed, the two lines of code in figure seven first use OpenCV's findContours function to detect contours in the pre-processed image, the cv2.RETR_EXTERNAL flag retrieves only the outermost contours, ignoring nested contours and cv2.CHAIN_APPROX_SIMPLE compresses contour points by only storing the start and end points of each vertically, horizontally, and diagonally compressed segment. The detected contours are then sorted by area in descending order using cv2.contourArea as the sorting key. Finally, only the top three largest contours are retained. The algorithm then loops over these large contours (figure 8 line 245), starting by calculating the perimeter and using it to perform contour approximation with a neutral epsilon value that works well for simple shapes. The first contour with four vertices found will have a four-point transformation performed to obtain a bird's eye view of the image [8, 12], finalizing the document isolation. If the document is not found, it will raise the DocumentExtractionFailedError, notifying the user that the image needs to be retaken.

```
contours, _ = cv2.findContours(image_proc, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = sorted(contours, key=cv2.contourArea, reverse=True)[:3]
```

Figure 7. findContours - CHAIN_APPROX_SIMPLE for finding contours in distorted images.

```
244         # loop over the contours
245         for i, c in enumerate(contours):
246             peri = cv2.arcLength(c, True)
247             approx = cv2.approxPolyDP(c, 0.025 * peri, True)
248 
249             if len(approx) == 4:
250                 print("Document found. Performing transformation.")
251                 transformed = four_point_transform(image, approx.reshape(4, 2))
252                 show_image("transformed", transformed) if self.show_process else None
253                 return transformed
254 
255         raise DocumentExtractionFailedError("Document could not be isolated")
```

Figure 8. Document extraction.

Now that the image is successfully transformed, the bubbles are grouped using getAnswerBubbles, shown in figure nine. It is important to deal with low-contrast photos that can come up as a result of a poorly taken image or even simply the camera quality on the phone/web camera before this. This is done by running an adaptive histogram equalization [9] on the transformed image using CLAHE or contrast-limited adaptive histogram equalization (figure 9 line 271). Unlike traditional histogram equalization, which works on the entire image, CLAHE divides the image into smaller regions called tiles and applies histogram equalization to each tile independently, preventing the amplification of noise and excessive contrast that can occur in varying degrees of brightness [16]. By setting a clip limit, CLAHE restricts the contrast amplification to prevent over-amplification, resulting in a balanced contrast enhancement across the image. After this, we apply a Gaussian blur to the grayscaled image with a 5x5 kernel to reduce noise. The blurred image is then thresholded using OTSU's method with the cv2.THRESH_BINARY_INV flag to convert it into a binary image, enhancing the edges by

making the foreground white and the background black [9]. The cv2.findContours function is then used to detect the contours in this binary image, focusing on external contours and simplifying the contour points (figure 9 line 276). If the contour is a circle, it is appended to question_contours. The contour is drawn on the original image in green if showing the process, and then the question contours are returned.

```
270         # prepare image to find contours
271         self.image = equalize_histogram(self.image) # levels out inconsistent brightness
272         gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
273         blurred = cv2.GaussianBlur(gray, (5, 5), 0)
274         self.thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
275
276         contours, _ = cv2.findContours(self.thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
277         question_contours = []
278
279         for contour in contours:
280             x, y, w, h = cv2.boundingRect(contour)
281             if self.is_circle(contour):
282                 question_contours.append(contour)
283                 cv2.drawContours(self.image, [contour], -1, (0,255,0), 2) if self.show_process else None
284         self.show_image("Bubbles identified image", self.image) if self.show_process else None
285
286         return question_contours
```

Figure 9. OMRGrader.getAnswerBubbles

Figure ten below showcases the OMRGrader run process up until now, in images arranged from left to right and top to bottom. At this point, which answer bubbles belong to which question must be identified. The answer bubble contours must be sorted into rows using a degree of proximity in their y coordinates. After the rows are established, they are dynamically sorted into questions using the sort_rows_to_questions method shown in figure eleven.

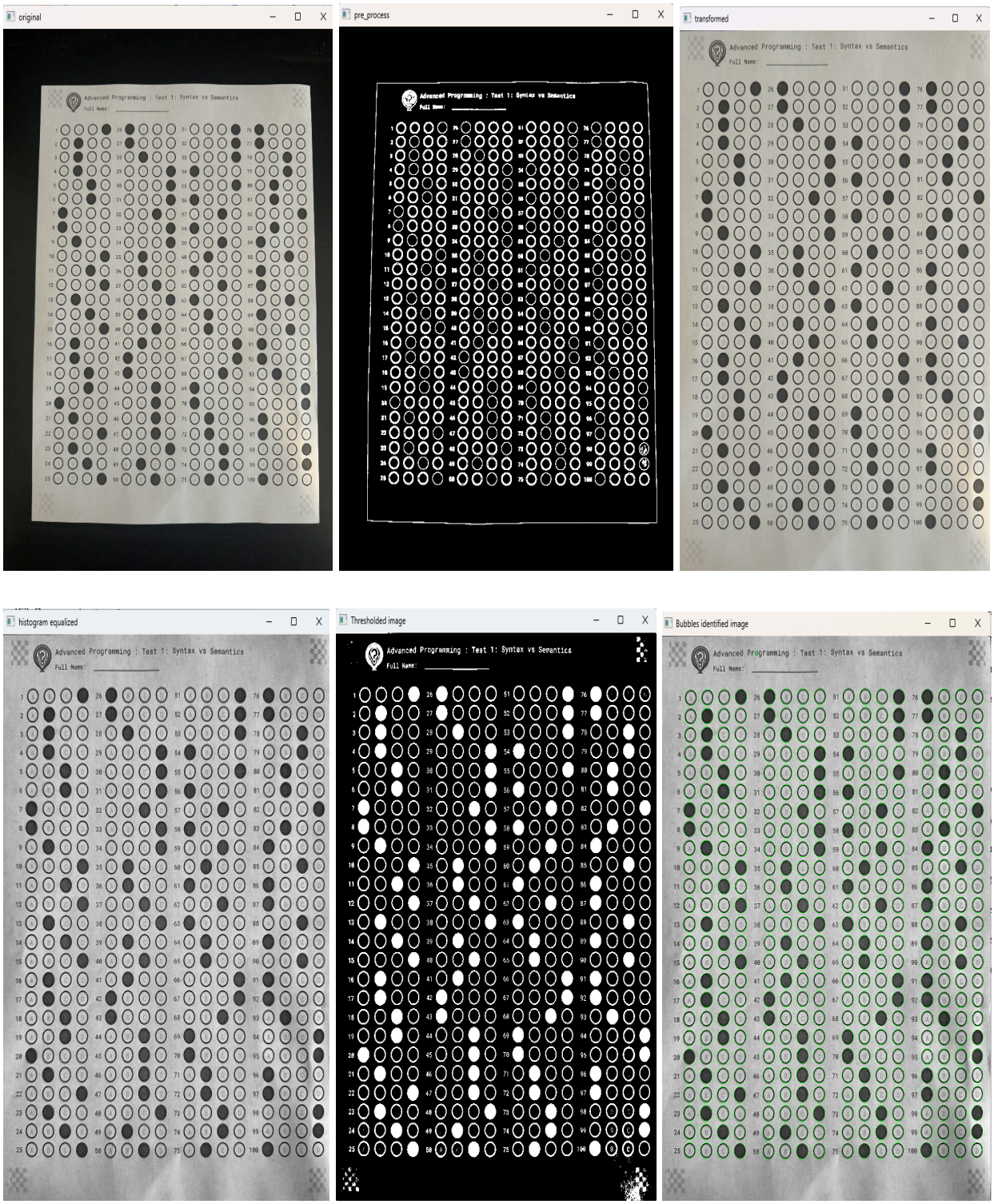


Figure 10. Images of Original, Pre-Processed, 4-point transformation, histogram equalization, re-thresholded, and answer bubbles highlighted.

```
342     def sort_rows_to_questions(self, rows):
343         questions = {}
344         col_num = 0
345         i = 0 # question number
346 
347         while i < self.num_questions:
348             questions[i + 1] = [
349                 rows[i % len(rows)][choice]
350                 for choice in range(col_num, col_num + self.num_choices)
351             ]
352             i += 1
353             if i % len(rows) == 0:
354                 col_num += self.num_choices
355 
356         return questions
```

Figure 11. OMRGrader.sort_rows_to_questions

After the contours have been grouped for each question, they are already in order from left to right and finally ready to be processed to determine each answer's choice using OMR. The questions are then run through identify_answer_choices shown in figure twelve. The method first initializes choices (figure 12 line 381). It then begins looping over each of the answer bubbles (lines 382-383). First, a mask is created to isolate each contour, and the non-zero pixels produced from the threshold within the masked area are counted (lines 385, 387, 388). The contour with the highest count for each question is identified as the chosen answer (line 391). The contour's total count, data, and choice letter are stored in the choices dictionary (line 392).

```
381         choices = {}
382         for question, contours in questions.items():
383             for j, c in enumerate(contours):
384                 mask = np.zeros_like(self.thresh, dtype="uint8")
385                 cv2.drawContours(mask, [c], -1, 255, -1)
386 
387                 mask = cv2.bitwise_and(self.thresh, self.thresh, mask=mask)
388                 total = cv2.countNonZero(mask)
389 
390                 # mark the choice with the highest total number of non zero
391                 if question not in choices or total > choices[question][0]:
392                     choices[question] = (total, c, chr(j + 65))
393 
394         return choices
```

Figure 12. OMRGrader.identify_question_choices

Finally, the grade_choices method is run with the originally passed answer key, highlighting correct answers in green and wrong answers in red. The final grade is placed in the top right corner and is shown in figure thirteen.

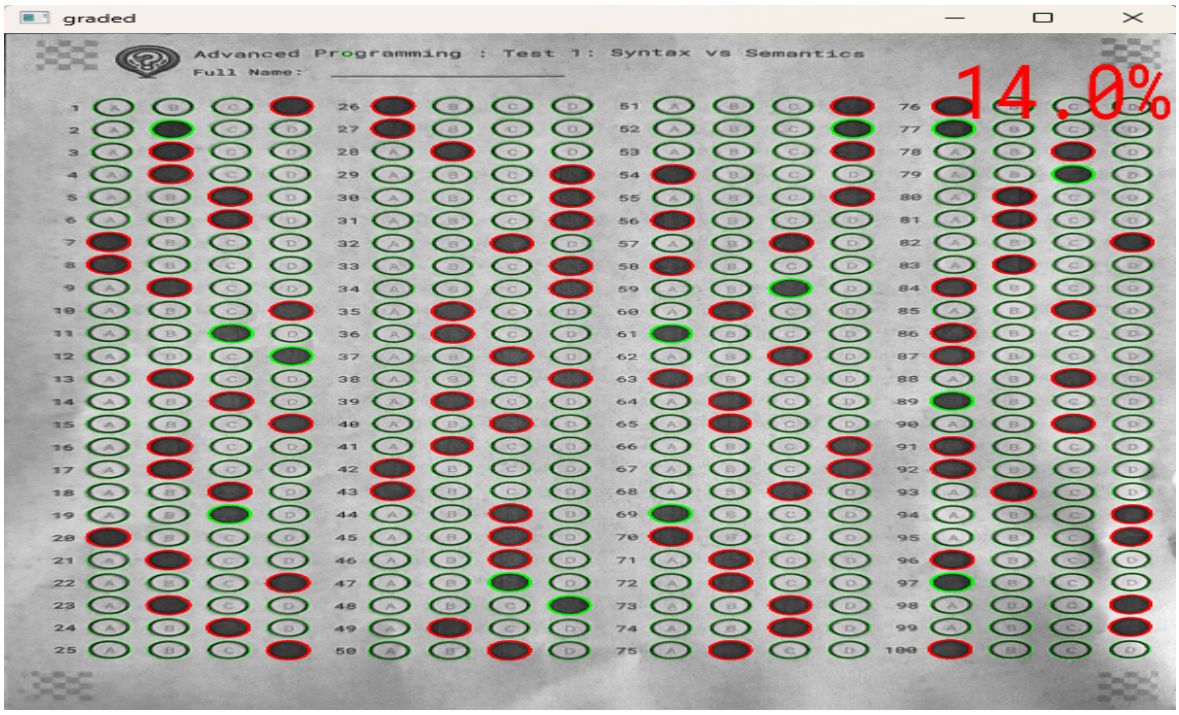


Figure 13. Final graded image of OMRGrader.run

Testing the Pictron and OMRGraders modules was a painstaking process made easier by the demo frontend app built using React, but it was well worth it. I used my iPhone 14 to conduct ten complete tests with multiple submissions each while recording the data in less than an hour. The total number of bubbles attempted for processing was over 3000. Of those, only 14 were graded incorrectly, putting the scanner's accuracy using an iPhone at over 99%, which has been achieved by other OMR grading systems [14]. Still, the submissions should be checked every time, as the scanner is still not 100% accurate. Below, figures fourteen through seventeen illustrate more detailed figures showing each step of the process to highlight the dynamic capabilities of OMRGrader and Pictron further.

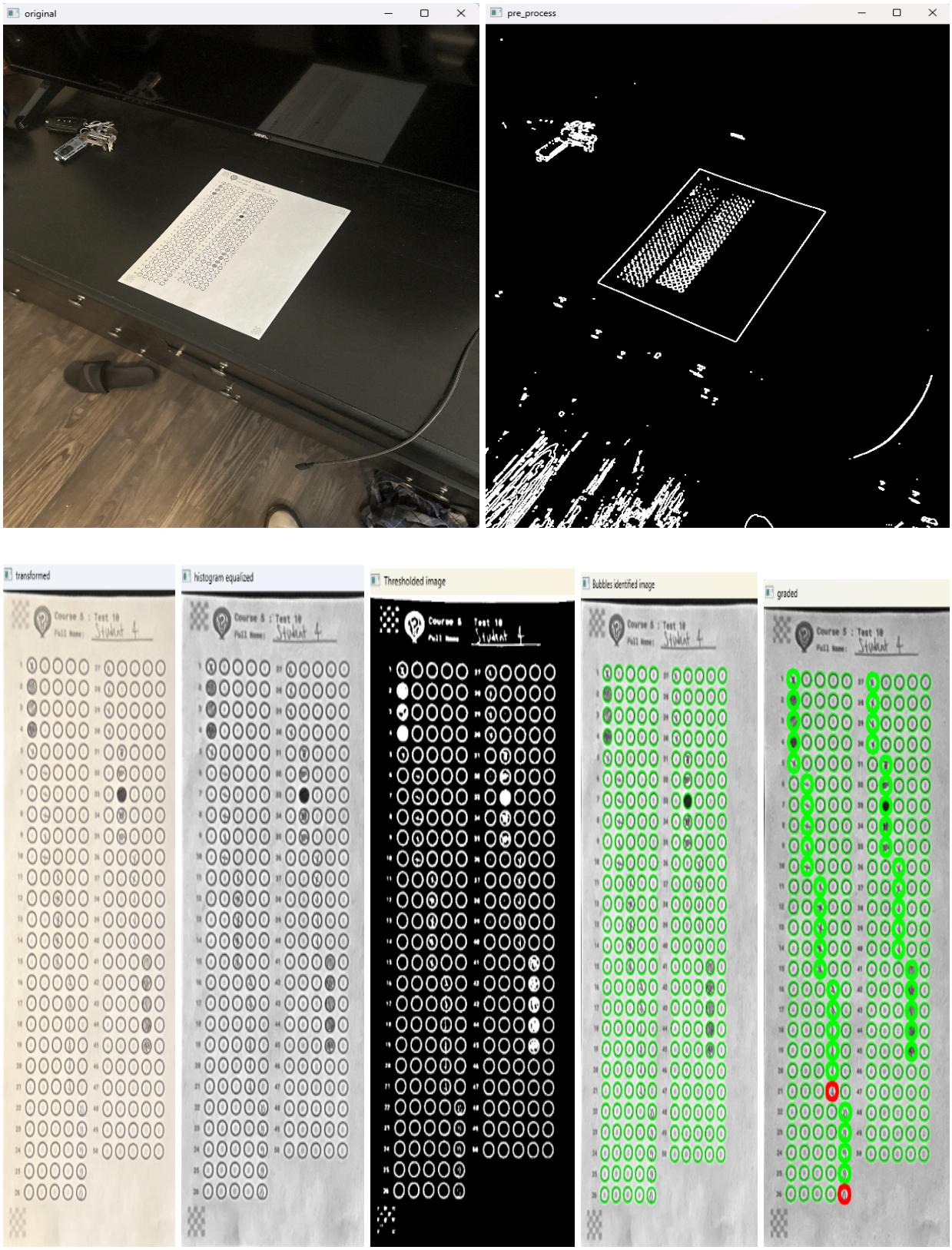


Figure 14. Showcases ability to take a varied brightness image and grade it.

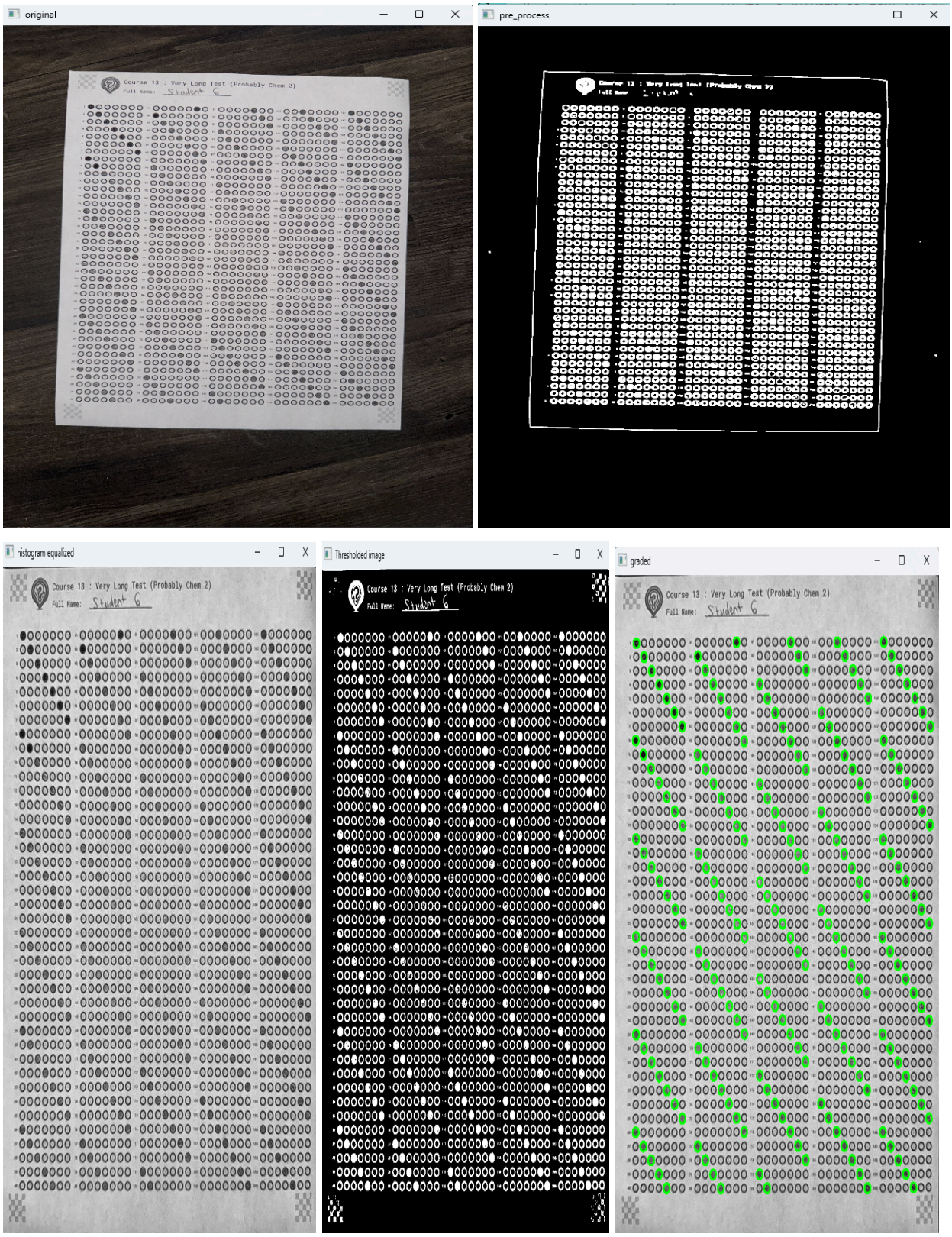


Figure 15. 1400 individual question bubbles graded with 100% accuracy.

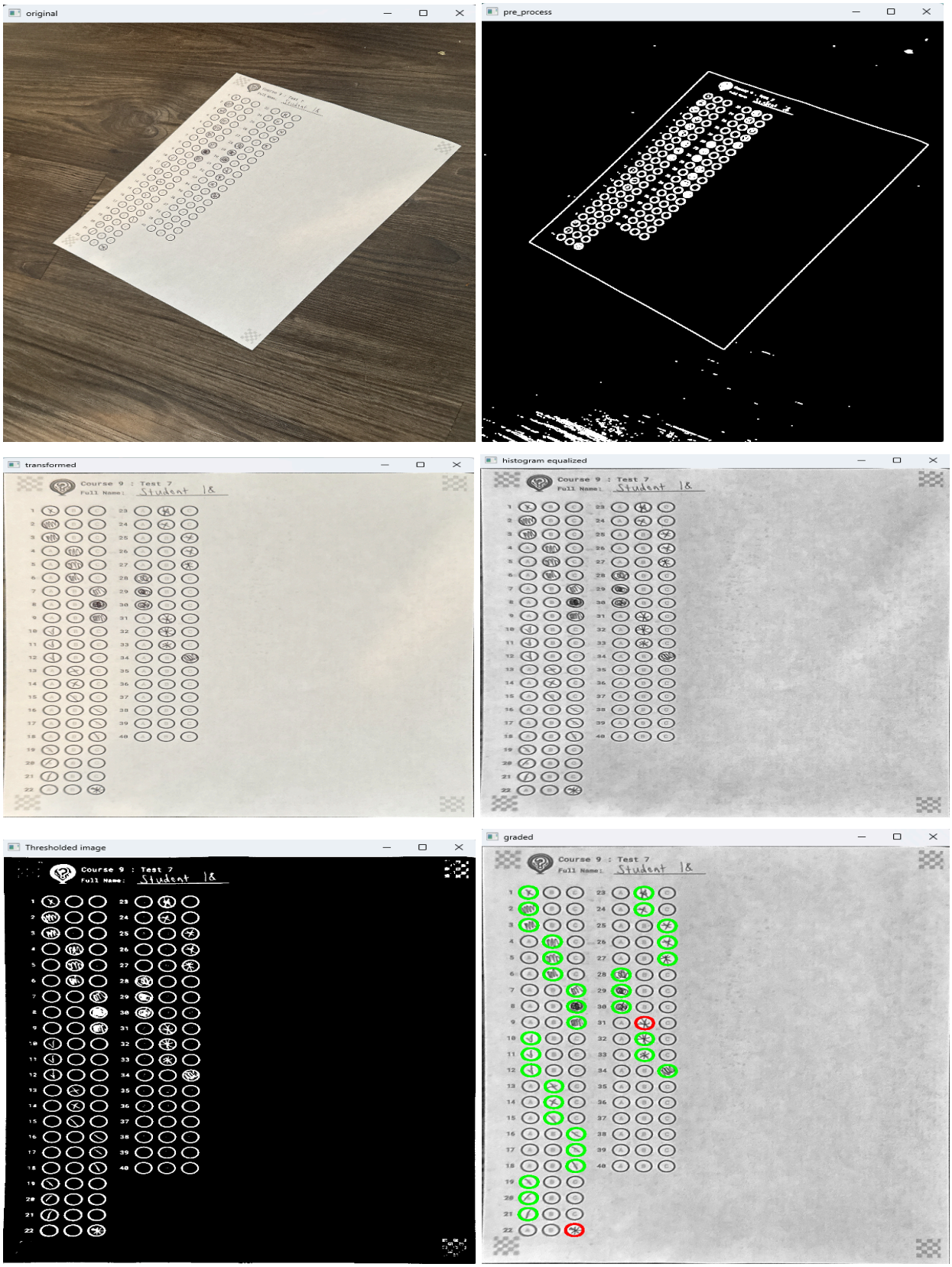


Figure 16: angled and tilted, splotchy lighting showcases the benefits of histogram equalization

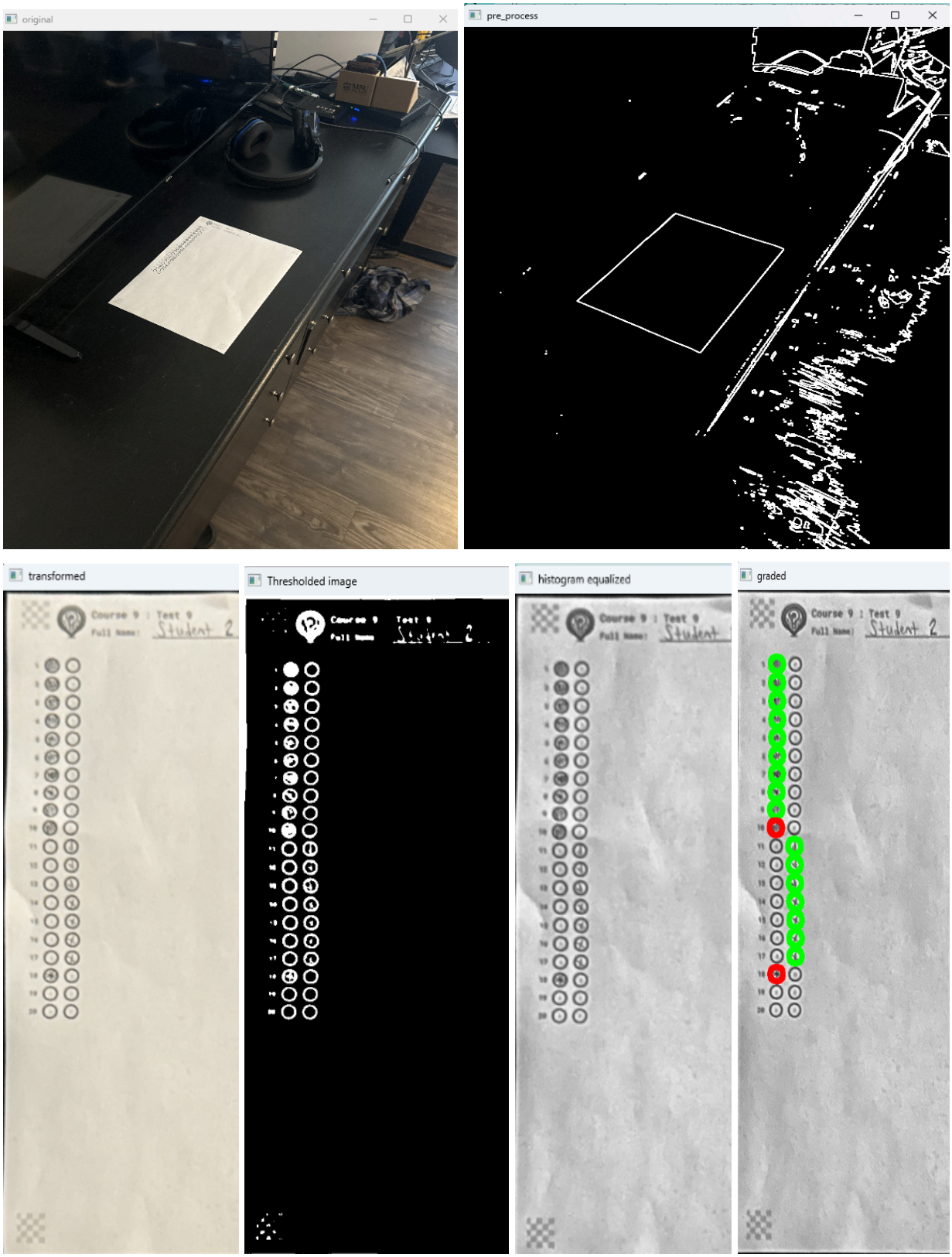


Figure 17: Far distance, angled, and slanted. Showcases ability to grade in diverse conditions.

To showcase LiveTest's potential, a generalized demo of the U.I. and the general flow will be given. LiveTest's database is built to run on a school-to-school basis. That means there is one set of courses, students, teachers, tests, enrollments, and submissions for every unique LiveTest instance. When teachers and students first log in, they are presented with the dashboard that lists all their courses, as shown in figure eighteen.

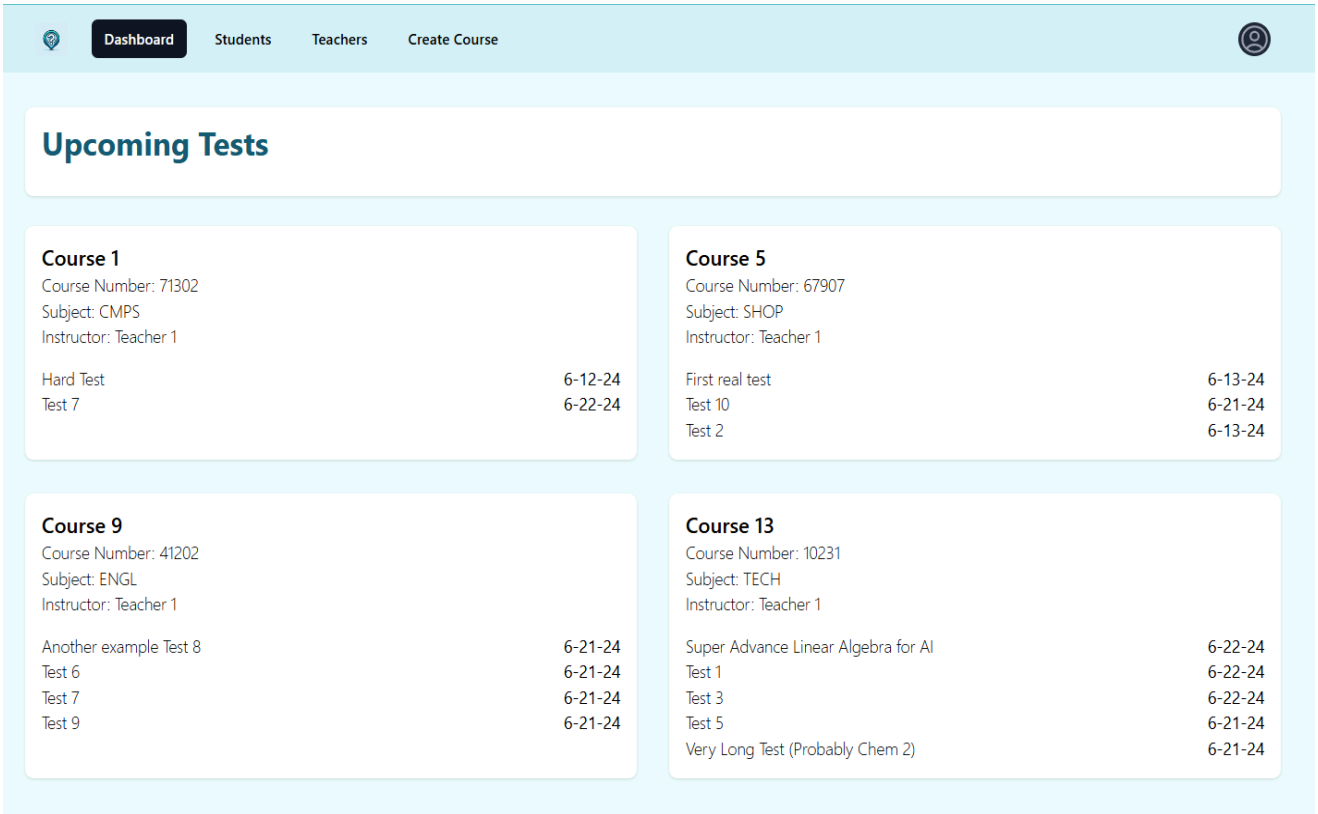
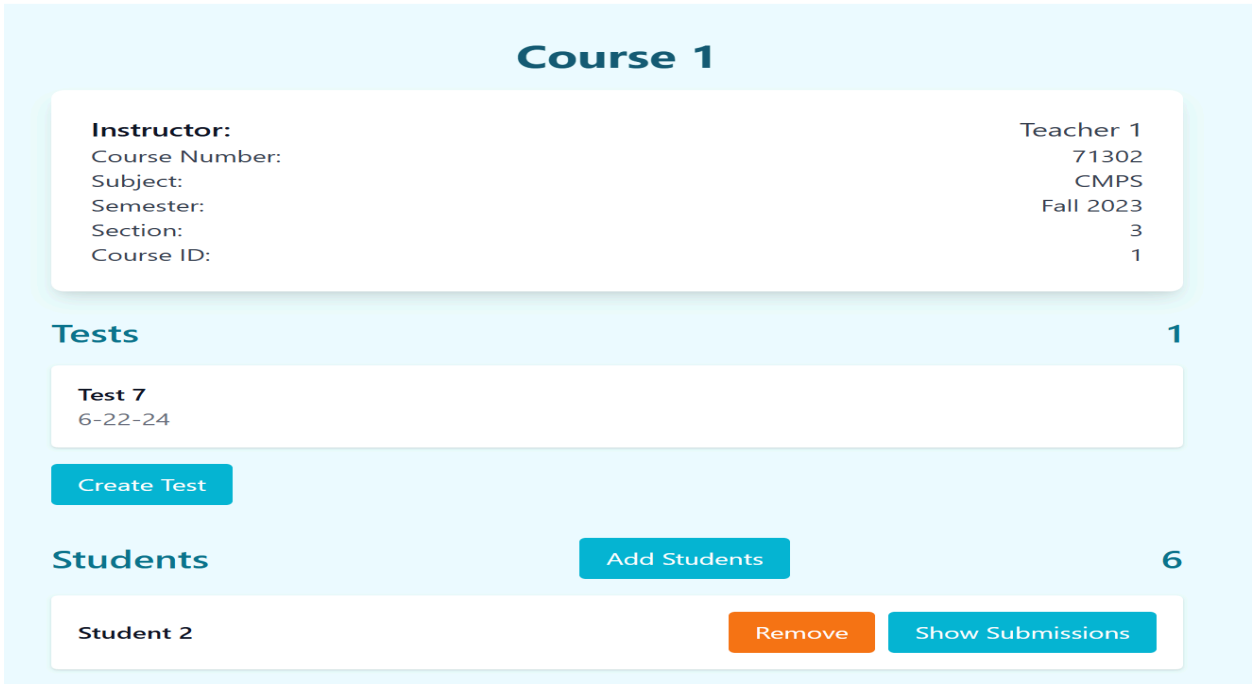


Figure 18. LiveTest dashboard.

By clicking on any of the courses shown in figure eighteen, the course page shown in figure nineteen will render. Giving the ability to view further course details, create a new test, add/remove students from the course, show individual students' submissions, etc. A teacher can easily add students to a course using the course enrollment QR code, it will bring each student to a page where they either log in and are enrolled, or they register, and the course becomes their first. This helps simplify administration for teachers, freely organizing their courses.



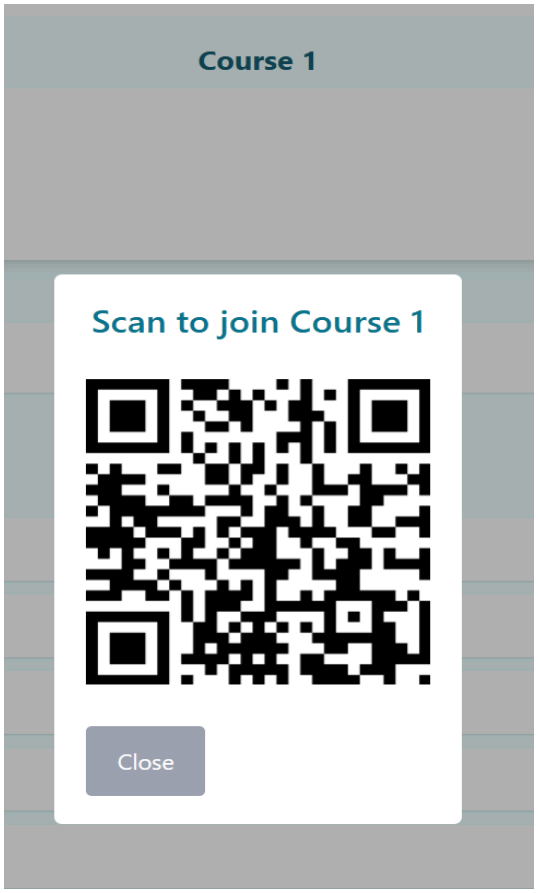


Figure 19. Course Page, Course enrollment QR code, Student's Submissions

Creating a test should be intuitive. With LiveTest, it is. The teacher simply gives the test a name, start/end time, number of questions/choices, and an answer key. Once all fields are filled out propely, the teacher clicks “Create Test”. LiveTest then automatically creates the test artifact for the specific course.

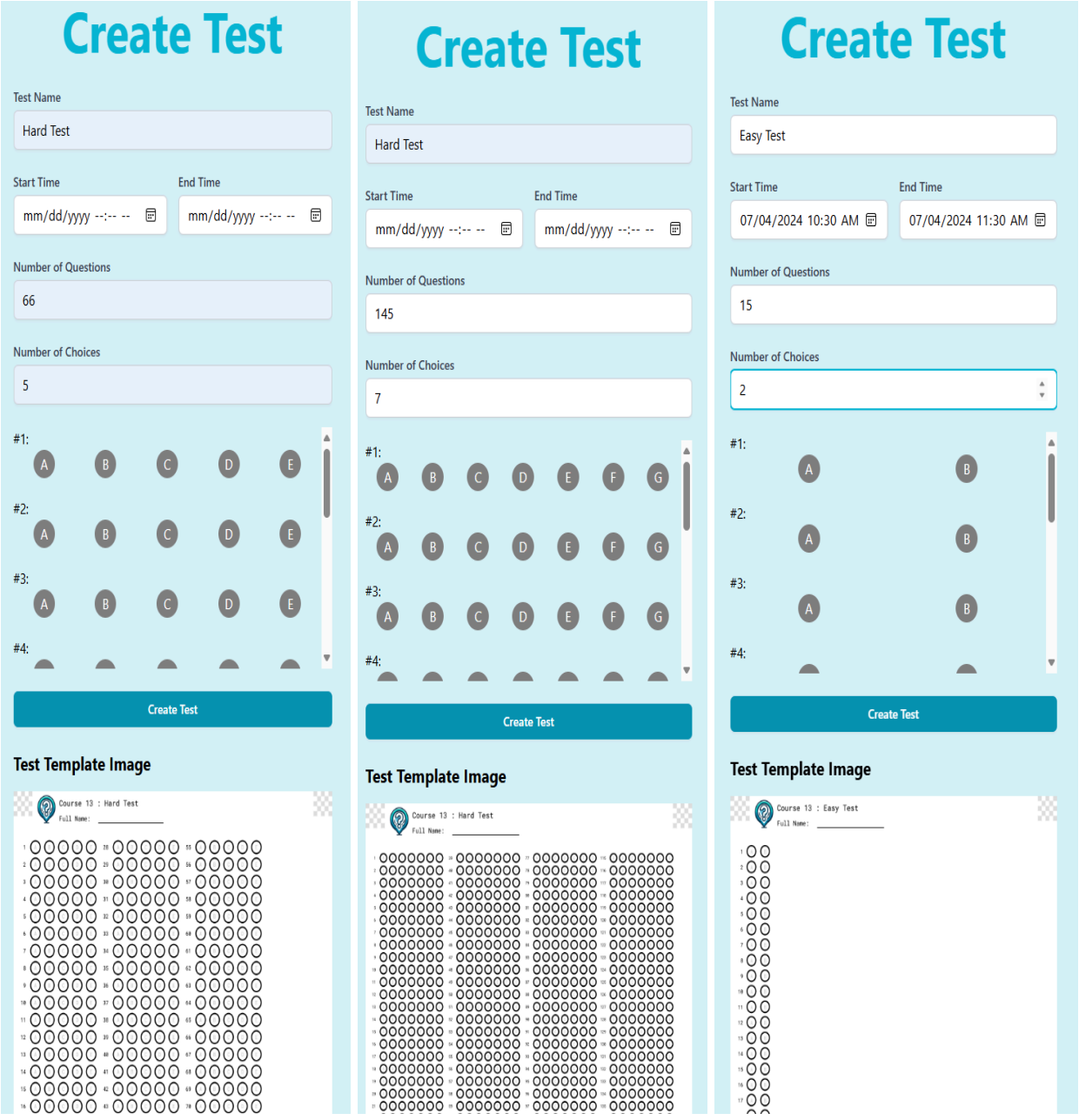


Figure 20. Create Test - U.I. Examples showcasing dynamic capabilities of Pictron

Once a test has been created, one can navigate to the individual “Test Page” shown below in figure twenty-one from the course page. Until the test begins, it is “IN WAIT” and can not accept submissions. You can edit the start/end time or name, delete the test, download blank answer sheets for the test, and also download the answer key that is used to grade them.

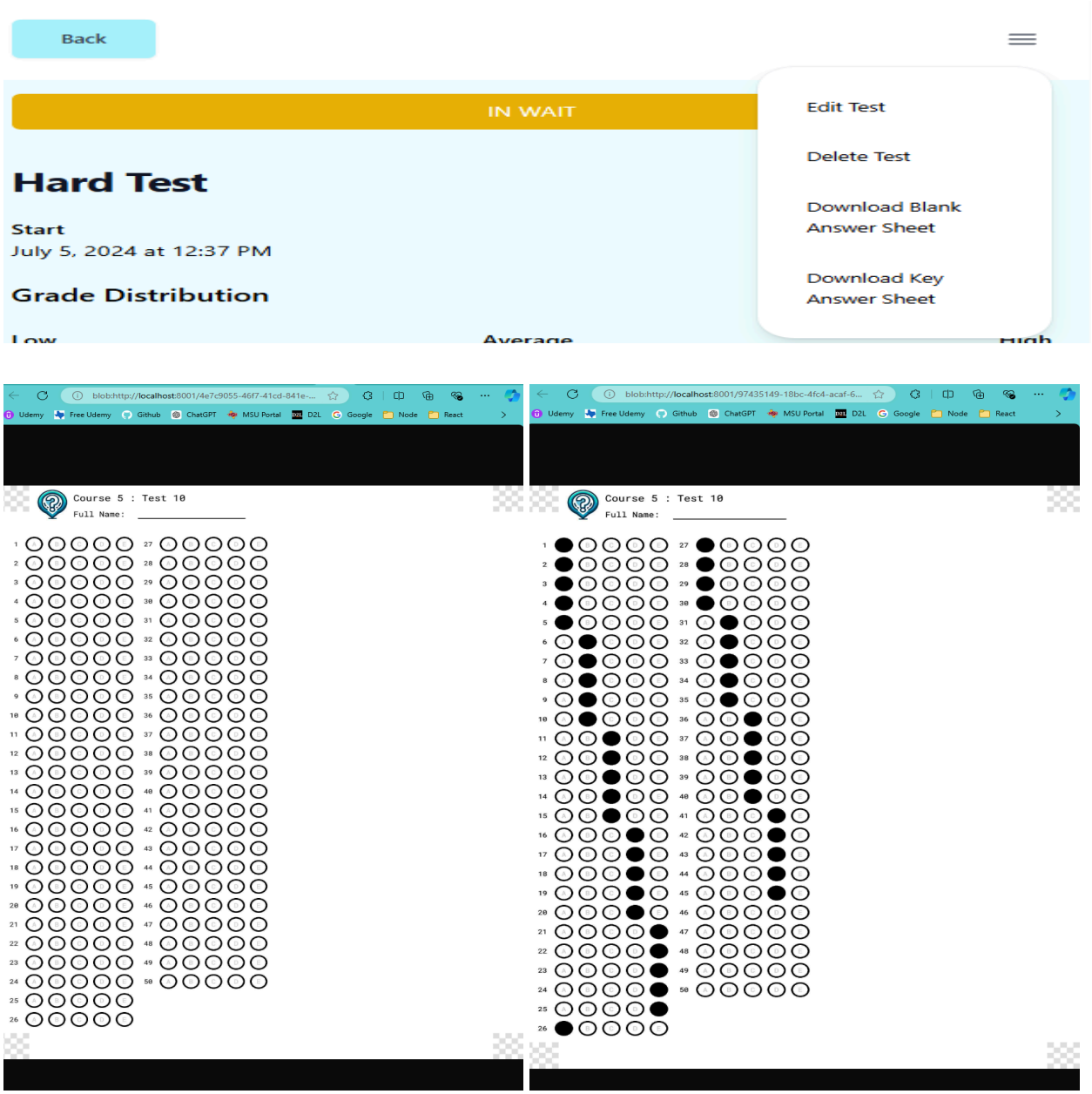


Figure 21. Test Page “IN WAIT” along with downloaded blank/key answer sheets

As shown in figure twenty-two, once the test starts, the bar will turn green, and the test becomes “LIVE,” signifying a “LiveTest.” When a test is “LIVE,” a teacher can submit a graded answer sheet by clicking the blue “Create a new submission” button.

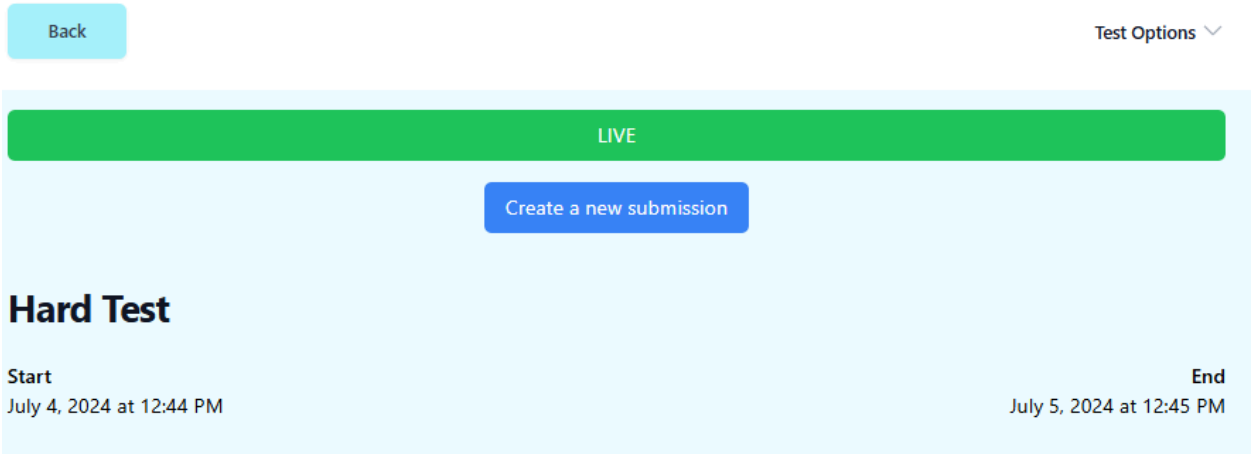


Figure 22. Test Page “LIVE”

The Submission Page shown in figure twenty-three starts by explaining the scanner's limitations with accepted and rejected examples. If instructors struggle with getting students to comply, they can simply show them the requirements. The instructor will select a student from the course the test was made for, take a picture (on IOS/Android), or select an image (PC/Mac) and attempt to grade it. For now, if the grading fails, you must resubmit. There are plans for a robust video feed accessible by web or phone cameras to automatically detect the document and repeatedly try to grade it until it works. However, for the initial release, the OMRGrader is showing good enough results to release with a primitive Submission Page that accepts single image submissions after the teacher selects the student. Once the image is successfully graded, the graded results will be returned, and the teacher can retake the photo or move on to the next student.

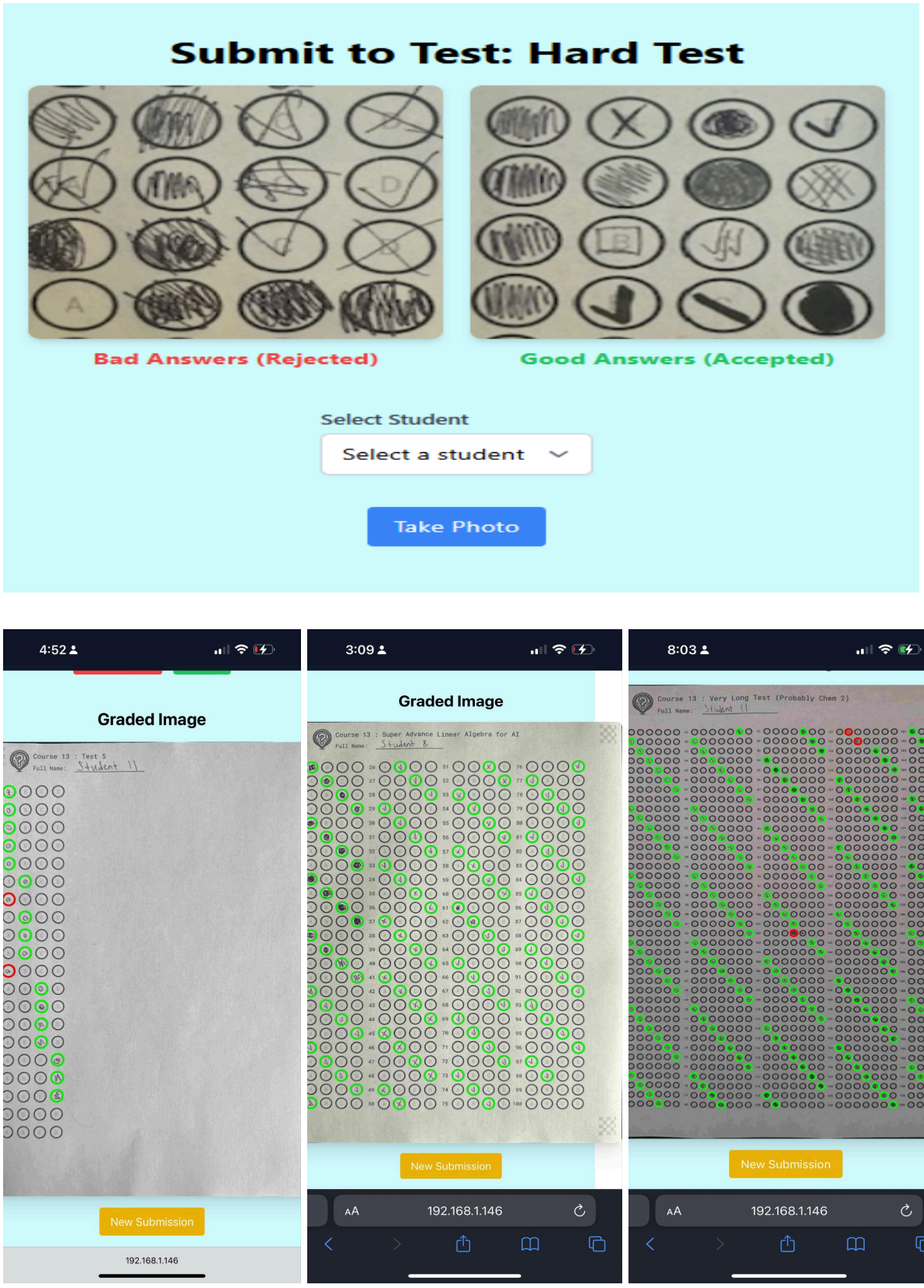


Figure 23. Submission Page

Once submissions have been made for the test, the analytics page will automatically populate the given charts for grade distribution and most missed questions shown in figure twenty four. The analytics shown is when the test is “FINALIZED”, but the test’s analytics are updated while it is “LIVE” as well.

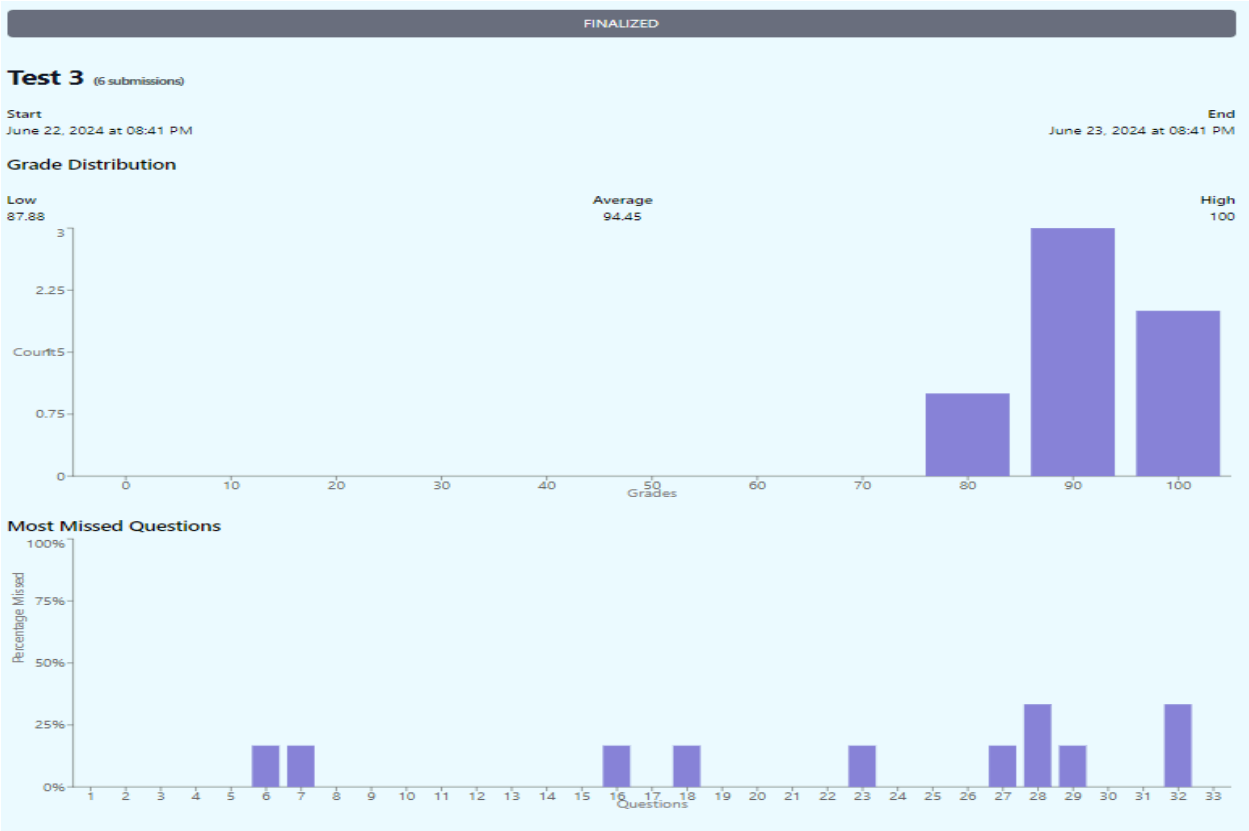


Figure 24. Test Analytics Page - Finalized

For grading purposes, a list of all the test’s submissions are available underneath the analytics page. In addition to this, a teacher can specifically view the submissions for individual students shown in figure twenty five. Although this demo may look useful, the frontend is lacking in comparison to the backend. A tech savvy individual could utlize LiveTest’s current capacity well enough to get their student’s tests graded but it is not yet to the level it needs to be for general usage or widespread adaptation.

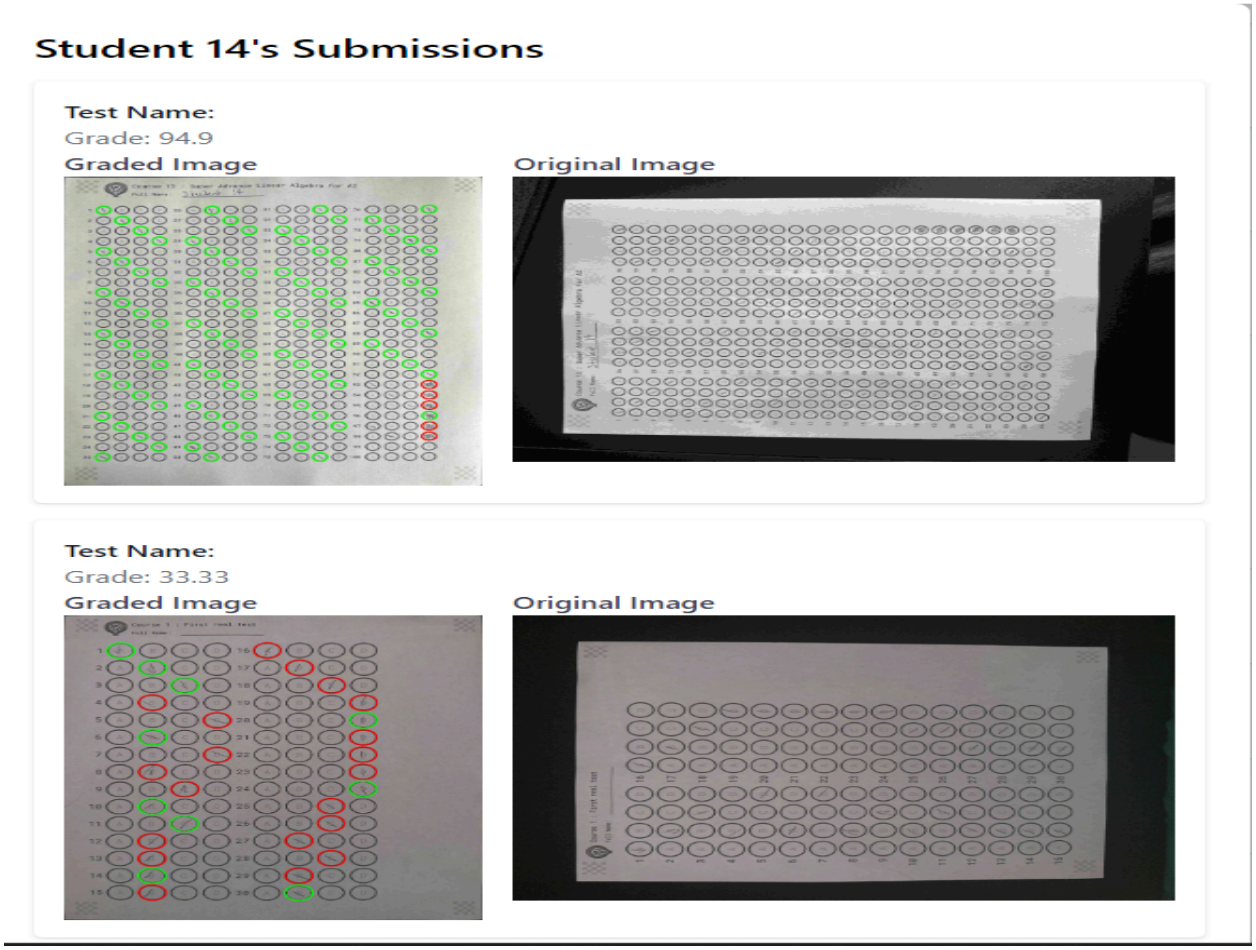


Figure 25. Student's Submissions Page

Conclusion

OMR has been a staple in educational testing, as it is efficient and accurate in grading multiple-choice exams. Despite technological advancements, OMR continues to be widely used and will likely remain so for the foreseeable future. While reliable, the traditional hardware-based OMR systems are often costly and inaccessible to many schools and educators. LiveTest was designed to tackle the financial and accessibility hurdles posed by traditional hardware-based OMR systems. Though a 99% accuracy rate was achieved in the trial, there is still much work to be done to provide a school-wide solution. With that in mind, the software has been open-sourced, concluding my research. Additionally, in the repo, you will find details on

installing the demo frontend/backend app using docker in just two commands to start creating and grading tests. Further information about the project can be found in the GitHub repository at https://github.com/gramcracker40/LiveTest. For now, testing is being performed at livetest.us where any teacher can sign up and give tests!

Acknowledgment

This research was inspired by Adrian Rosebrock's blog post on pyimagesearch.com, Bubble sheet multiple-choice scanner, and grader using OpenCV [13]. This read was very helpful and provided a lot of context for me while building OMRGrader.

References

1. Arjun Singh, Sergey Karayev, Kevin Gutowski, and Pieter Abbeel. 2017. Gradescope: A Fast, Flexible, and Fair System for Scalable Assessment of Handwritten Work. In Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale (L@S '17). Association for Computing Machinery, New York, NY, USA, 81–88. https://doi.org/10.1145/3051457.3051466
2. Calaguas, Noriel & Consunji, Paolo. (2022). Mobile Optical Mark Recognition Application As A Non-Inferior Alternative To Manual Marking For Lower And Middle Income Countries. 5. 10-19.
3. Cuerdo, R. (2021). Effectiveness of Automation in Evaluating Test Results Using EvalBee as an Alternative Optical Mark Recognition (OMR): A Quantitative-Evaluative Approach from a Philippine Public School. International Journal of Theory and Application in Elementary and Secondary School Education.
4. de Elias, E.M., Tasinaffo, P.M. & Hirata, R. Optical Mark Recognition: Advances, Difficulties, and Limitations. *SN COMPUT. SCI.* 2, 367 (2021). https://doi.org/10.1007/s42979-021-00760-z
5. LLC, Z. (n.d.). *IPhone and Android grading app for formative assessment and quizzes.* https://www.zipgrade.com/
6. *Omr Quiz*. OMR Quiz. (n.d.). https://www.omrquiz.com/
7. *OpenCV: eroding and dilating*. (n.d.). https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html
8. *OpenCV: Geometric Transformations of Images*. (n.d.). https://docs.opencv.org/3.4/da/d6e/tutorial_py_geometric_transformations.html#:~:text=

To%20find%20this%20transformation%20matrix%2C%20you%20need%204,Then%20apply%20cv.warpPerspective%20with%20this%203x3%20transformation%20matrix

9. *OpenCV: Histograms - 2: Histogram Equalization*. (n.d.). https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html
10. *OpenCV: Image thresholding*. (n.d.). https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html
11. *Remark Software Online Store*. Remark Software Store. (n.d.). https://buy.remarksoftware.com/
12. Rosebrock, A. (2021, July 4). *4 point OpenCV GetPerspective Transform example - PyImageSearch*. PyImageSearch. https://pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/
13. Rosebrock, A. *Bubble sheet multiple choice scanner and test grader using OMR, Python, and OpenCV*. https://pyimagesearch.com/2016/10/03/bubble-sheet-multiple-choice-scanner-and-test-grader-using-omr-python-and-opencv/
14. S. Agarwal, M. Varun and S. Prabakeran, "OMR Reader Info Scanner," 2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2023, pp. 205-209
15. Udayraj. (n.d.). *GitHub - Udayraj123/OMRChecker: Evaluate OMR sheets fast and accurately using a scanner or your phone.* GitHub. https://github.com/Udayraj123/OMRChecker

16. Vincent Stimper, Stefan Bauer, Ralph Ernstorfer, Bernhard Schölkopf, R. Patrick Xian. "Multidimensional Contrast Limited Adaptive Histogram Equalization." arXiv, 26 June 2019, https://arxiv.org/pdf/1906.11355.