Detecting Dice

A Computer Vision Project Report

Sarah Brown
College of Engineering
University of Oklahoma
Norman, OK

***Abstract* — A computer vision project focused on filtering an image to the shape of dice in a tray and processing the numbers on the dice. Main application of project centers on the use of physical dice in online sessions for tabletop games. GUI interface allows players to show their dice rolls to other players during gameplay. Retains the positive aspects of physical dice in online environments.**

Keywords— computer vision, text recognition, filtering, dice

I. Introduction

A. Description of Problem

In many tabletop games, dice are used to predict outcomes and results of different scenarios. Standard sets of dice have seven different types of polyhedral dice (d4, d6, d8, d10, d12, d20, d%). As dice sets like these are common across many games, many people collect them in various colors and styles. However, as more events are occurring online due to safety risks of in-person gatherings, random number generators are often used instead. This can take some of the fun out of seeing player reactions to a really good roll of the dice.

B. Problem Solution

By introducing a dice tray setup that includes a camera that can observe the dice rolls, friends can share video streams and get real time reactions to dice rolls in online settings. This solution focuses on processing a video stream of dice rolling into a small tray to retrieve the dice values.

C. Differences Between Orignal Plan and Implemented Plan

The original project scope allowed for processing of all seven types of dice except for the d4 and would be motion activated. Due to difficultly processing some of the different die shapes, only the d6, d8, d10, and d12 obtain reliable results. In addition, the current setup relies on a GUI button being pressed to activate the processing as motion detection was not implemented. Finally, when dice are rolled into a tray, they can land in a random rotation, but this causes issues with text recognition as images are in the wrong orientation. Motion detection and automatic image rotation will be implemented in future adjustments to this project.

II. Overview of Process

A. Intial Approach

Initially, image processing focused on filtering based on HSV values and creating an image mask with on a range of HSV values. This approach worked well on dice after a calibration process and was used to identify dice type based on the number of edges found in the filtered picture. However, the filtering process had to be calibrated for each different die color and resulted in missing pixels in the number on the die face. Because of these issues, a different filtering processed was used following the progress report.

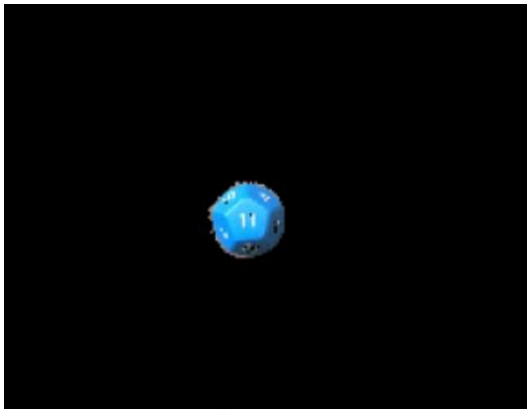


Figure 1. A d12 filtered with a HSV color range.

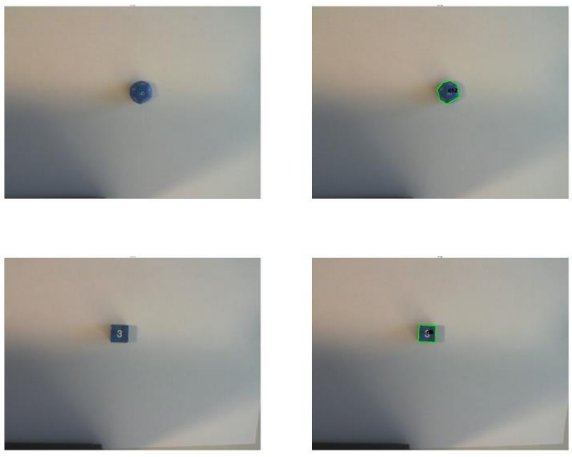


Figure 2. A d12 and a d6 correctly identified based on number of edges found in the image.

B. *Final Approach*

The second approach focused on a series of filtering techniques applied to video frames from the webcam stream. This image was filtered to reduce noise and then used to find the edges with a Canny edge detection. The edges are then filtered based on contour length to identify the edges of the die. Using the die edges, the number of vertices is found and counted to identify the type of die. The die type is then used to specify the numbers that are expected to be seen on the current die to enable the use of a whitelist to increase text recognition accuracy. This approach still requires aspects of calibration as it uses contour length to help filter images, which is based on how far away the camera is from the dice.

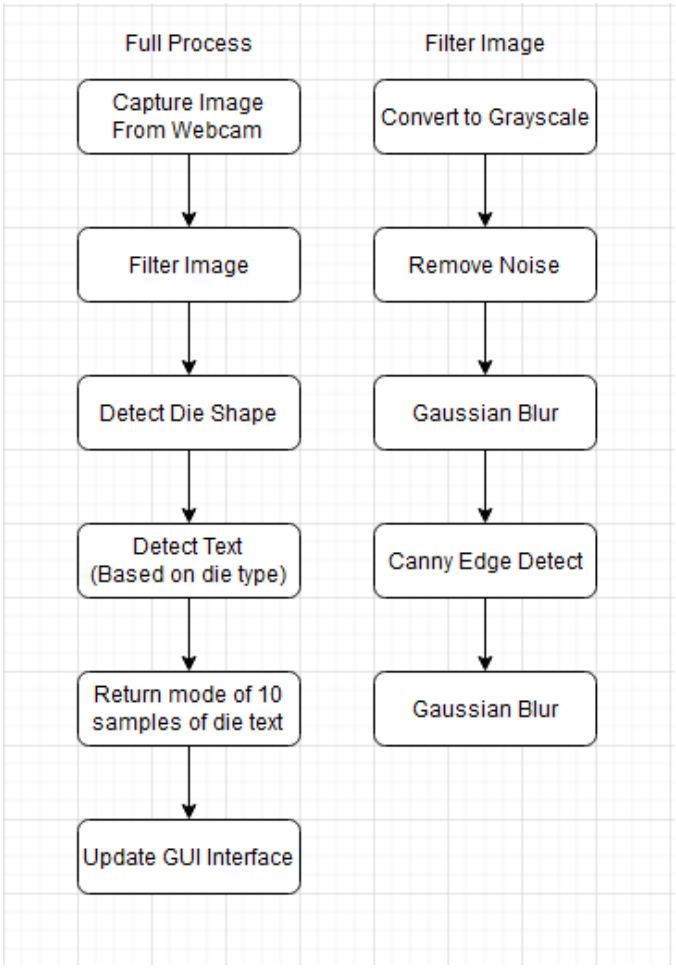


Figure 3. Outline of filtering and text recognition process.

III. TESTING SETUP

A testing setup was used to facilitate consistent testing results between different filtering configurations. This setup was constructed with a paper dice tray and a webcam elevated on a 3D printed stand. This setup created a constrained environment for the rolling of dice and guaranteed the die always landed in the cameras field of view. The testing setup was essential to comparing the results of different ways of filtering as it eliminated additional variables.

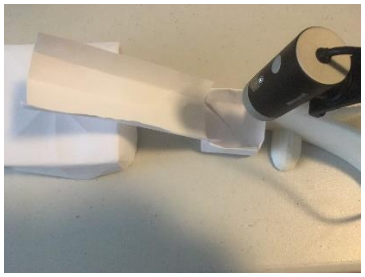


Figure 4. Testing setup with a paper dice tray and webcam.

IV. IMAGE PROCESSING

A. *General Image Filtering*

The image filtering goes through several steps. First, the image is converted to grayscale. This is followed by the use of a median blur to remove noise from the camera and other sources. After this, the image is blurred again with a Gaussian blur. After it is blurred, the edges are found with a Canny edge detect function. This image is saved for use with text recognition. The edges are then blurred again with another Gaussian blur to be used to find the edges of the die. This process was tested in different orders and with different strengths to determine the most effective process. Testing was done by showing the camera input, the filtering output, and the results after a mask to remove the edges of the die was applied. The edges of the die were removed to aid with the text recognition process. This also helped to isolate the individual die number on dice where more than one number at a time is visible.

B. *Addition of Mask*

Using the blurred output from the Canny edge detection, the OpenCV function findContours was used to identify contours. These contours were then filtered by length to identify the edges of the die in the frame. By drawing the approximate of these contours on a blank image, it was possible to apply a mask to the frames to remove the edges of the die. This was implemented due to errors that were received during the text recognition step due to the shape of the die. In addition to masking based on the outline of the die, it is possible to find the middle of the contours and create different mask shapes. These masks are then applied to the center of the die. However, this had issues earlier because of the contours shifting due to camera noise. This noise was minimized with the addition of a second Gaussian blur in the filtering stage. However, masking with various shapes instead of the contours of the die has not been tested with text recognition at this point.

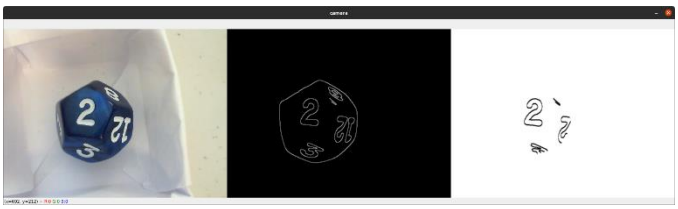


Figure 5. A d12 showing the different steps of processing. Left: Camera Input. Middle: Filtered Image. Right: Image with Mask.

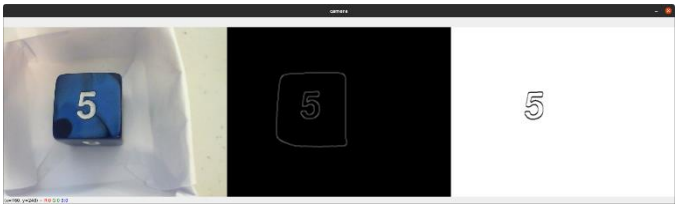
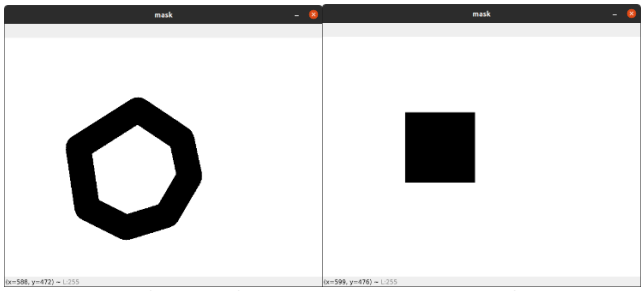


Figure 6. A d6 showing the different steps of processing. Left: Camera Input. Middle: Filtered Image. Right: Image with Mask.



Figures 7 & 8. Left, a mask for the d12 based on the edges of the die. Right, a mask for the d6 made by adding a square on the center of the contours found for the die's edges.

V. Text Recognition

The output of the frame being filtered and processed is then used to detect digits on the die's face. This is done with Google's open source OCR engine Tesseract. With the use of the Python version of this package, it is possible to identify text. There are various configuration modes for Tesseract and by implementing options for digit characters only, recognition results were improved immensely from initial tests. In addition, by creating different configuration variables per die type, it was possible to create whitelists to avoid misrecognizing digits. This specifically improved results for the d6 as removing the digit '8' helped the results for correctly identifying the '3' digit.

VI. GUI Interface

To allow users easy access, a graphic user interface was developed with the use of the package PySimpleGUI. This package was used to create a window to house the video stream and a button to trigger the image processing. After the roll button has been pressed, the program takes 10 sample images and outputs the mode of this data back out on the window. In addition, the window shows the label for the die based on the number of edges that have been identified.

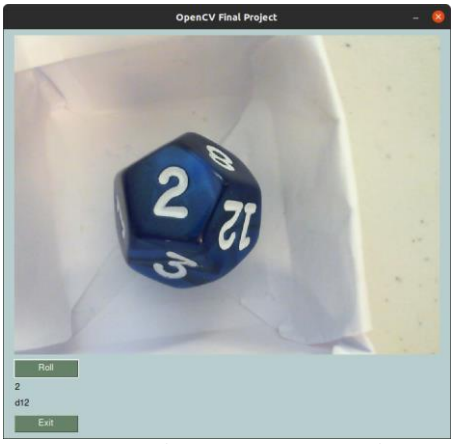


Figure 9. The GUI interface showing the result of rolling a d12.

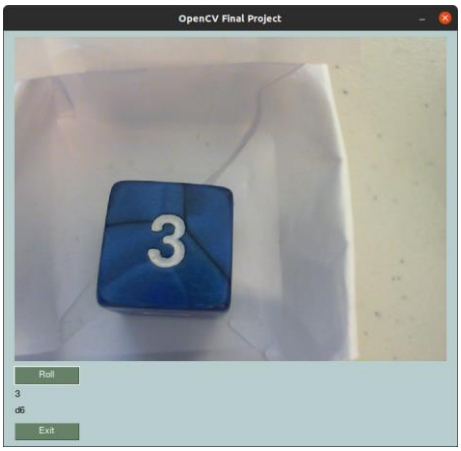


Figure 10. The GUI interface showing the result of rolling a d6.

VII. Conclusion

A. Challenges

The main challenges that were faced while working on this project centered on developing a filtering process that accommodated for various lighting conditions and did not require additional, direct lighting to be provided. This challenge was faced by shifting from the HSV filtering approach to the one described in previous sections. In addition, when first testing text recognition the edges of the die were not filtered out of the image. The edges resulted in errors and random characters, but the application of masks helped limit these errors. Another challenge related to text recognition was that the digits were often viewed as random characters. This was solved by adding a whitelist to the text recognition configuration which eliminated many errors.

B. Further Work

This project can be improved by adding elements to detect motion between video frames. Detecting motion would allow for digit recognition to begin automatically when dice are rolled into the dice tray. In addition, when dice are rolled into the tray instead of placed into the tray, they can land in many different configurations. Improvements to the filtering process are needed to accommodate for rotated dice to prevent errors arising in the digit recognition aspects of the project.

Code Appendix

The code for this project is shown below. This can also been seen on the github project here: https://github.com/SarahBrown/ECE5973-CV/blob/main/Final%20Project/final_project_2021.py

```
import PySimpleGUI as sg
from cv2 import cv2
import numpy as np
import imutils
import pytesseract
import statistics

configd4 = '--oem 3 --psm 6 outputbase digits -c tessedit_char_whitelist=1234'
configd6 = '--oem 3 --psm 6 outputbase digits -c tessedit_char_whitelist=123456'
configd8 = '--oem 3 --psm 6 outputbase digits -c tessedit_char_whitelist=12345678'
configMisc = '--oem 3 --psm 6 outputbase digits -c tessedit_char_whitelist=0123456789'

CAMERA = 2

def filter(img):
    imgReturn = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgReturn = cv2.medianBlur(imgReturn,3)
    imgReturn = cv2.GaussianBlur(imgReturn,(3,3),5)
    imgCanny = cv2.Canny(imgReturn, 0, 255,(7,7))
    imgReturn = cv2.GaussianBlur(imgCanny,(5,5),5)

    return(imgReturn, imgCanny)

def detectShape(process_img):
    shape = 'unknown'; sides = None
    mask = np.zeros(process_img.shape[:2], dtype="uint8")

    cnts = cv2.findContours(process_img.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)

    for cnt in cnts:
        if cv2.arcLength(cnt,False) > 500:
            accuracy = 0.02*cv2.arcLength(cnt,True)
            approx = cv2.approxPolyDP(cnt, accuracy, True)
            sides = len(approx)

            cv2.drawContours(mask,[approx],-1,(255,255,255),15)

    if (sides == 3):
        shape='d4'
    elif(sides==4):
        shape='d6'
    elif(sides==7):
        shape='d12'
    else:
        shape='circle'

    return shape, mask

def detectTextd4(img, mask):
    maskNot = cv2.bitwise_not(mask)
    masked = cv2.bitwise_and(img, img, mask=maskNot)
    masked = cv2.dilate(masked, (3,3), iterations=2)
    masked = cv2.GaussianBlur(masked,(3,3),5)
    masked = 255 - masked

    masked_text = pytesseract.image_to_string(masked, config=configd4)
    return(masked_text)

def detectTextd6(img, mask):
    maskNot = cv2.bitwise_not(mask)
    masked = cv2.bitwise_and(img, img, mask=maskNot)
    masked = cv2.dilate(masked, (3,3), iterations=2)
    masked = cv2.GaussianBlur(masked,(3,3),5)
    masked = 255 - masked

    masked_text = pytesseract.image_to_string(masked, config=configd6)
    return(masked_text)

def detectTextd8(img, mask):
    maskNot = cv2.bitwise_not(mask)
    masked = cv2.bitwise_and(img, img, mask=maskNot)
    masked = cv2.dilate(masked, (3,3), iterations=2)
    masked = cv2.GaussianBlur(masked,(3,3),5)
    masked = 255 - masked

    masked_text = pytesseract.image_to_string(masked, config=configd8)
    return(masked_text)

def detectTextMisc(img, mask):
    maskNot = cv2.bitwise_not(mask)
    masked = cv2.bitwise_and(img, img, mask=maskNot)
    masked = cv2.dilate(masked, (3,3), iterations=2)
    masked = cv2.GaussianBlur(masked,(3,3),5)
    masked = 255 - masked

    masked_text = pytesseract.image_to_string(masked, config=configMisc)
    return(masked_text)
```

```
def createWindow():
    sg.theme("LightGreen")
    # Define the window layout
    layout = [
        [sg.Image(filename="", key="-IMAGE-")],
        [sg.Button("Roll", size=(10, 1), key="-ROLL-")],
        [[sg.Text("", size=(60, 1), justification="left", key="-c-")],
        [sg.Text("", size=(60, 1), justification="left", key="-shape-")]],
        [sg.Button("Exit", size=(10, 1))],
    ]

    # Create the window and show it
    window = sg.Window("OpenCV Final Project", layout, location=(800, 400))
    return window

def main():
    window = createWindow()

    cap = cv2.VideoCapture(CAMERA)

    while True:
        event, values = window.read(timeout=20)

        if event == "Exit" or event == sg.WIN_CLOSED:
            break

        ret, frame = cap.read()

        if event == "-ROLL-":
            dice_value = None; die_shape = ''; dice_values = []; shapes = []

            for i in range(10):
                process_img, process_canny = filter(frame)
                die_shape, mask = detectShape(process_img)
                print(die_shape)
                if(die_shape == 'd4'):
                    dice_value = detectTextd4(process_canny, mask)
                elif (die_shape == 'd6'):
                    dice_value = detectTextd6(process_canny, mask)
                elif(die_shape == 'd8'):
                    dice_value = detectTextd8(process_canny, mask)
                else:
                    dice_value = detectTextMisc(process_canny, mask)

                shapes.append(die_shape)
                print(dice_value)
                if (dice_value != '\x0c'):
                    dice_values.append(int(dice_value[:2]))
                ret, frame = cap.read()

            print(dice_values)
            print(shapes)
            if (len(dice_values) > 0):
                dice_value = statistics.mode(dice_values)
            print(dice_value)

            window["-c-"].update((dice_value))
            window["-shape-"].update((shapes[0]))


        imgbytes = cv2.imencode(".png", frame)[1].tobytes()

        window["-IMAGE-"].update(data=imgbytes)


    window.close()


main()
```