CEES

NEES

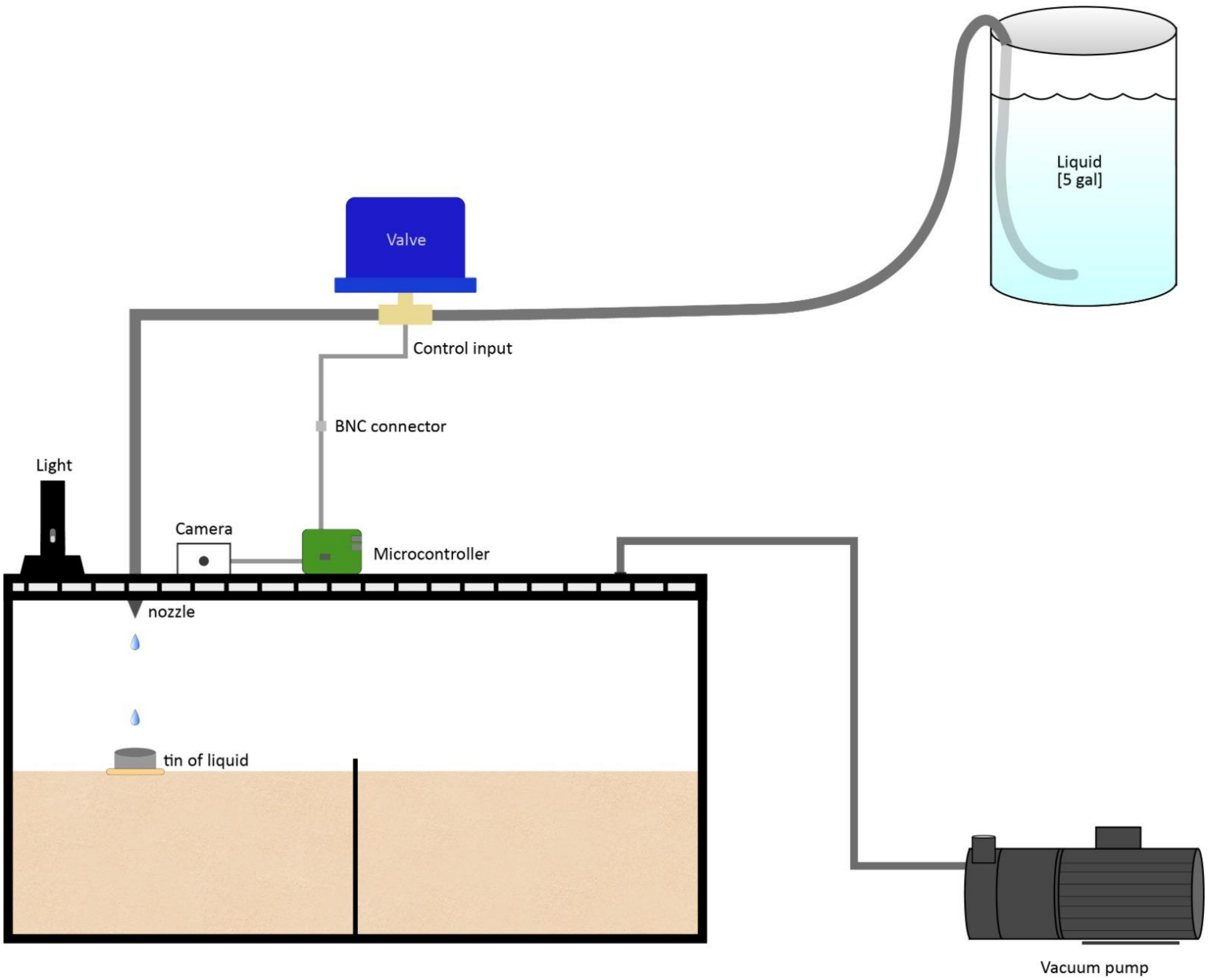
AUTOMATED SATURATION SYSTEM

VERSION 1.0

01/03/2020

Ivan Hammel

James Avtges


Liquid
[5 gal]
Valve
Control input
BNC connector
Light
Camera
Microcontroller
nozzle
tin of liquid
Vacuum pump

Automated saturation system

Introduction:

The Automated Saturation System is designed to mechanize the saturation portion of experiment preparation and free up valuable hours for the researchers. The system has two main components, software and hardware. The hardware consists of a lighting source to enhance the liquids reflectivity and a USB-interface camera plugged into a microcontroller. The microcontroller is in turn wired to a voltage amplifier that emits a calculated voltage through a BNC connector to the valve. The program processes the video feed to detect and count any moving pixels and use this value to determine whether a drop occurred. The output of every frame is then used in an algorithm to produce a corresponding voltage sent to the electronically controlled proportioning valve. In this way, after a brief initialization period, the Automated Saturation System will complete the model's saturation without further human involvement.

Usage

Initial Information

Upon initialization, the user will be prompted for several pieces of information. These are, in order:

- Title
- Experiment number
- User
- Desired drop rate (in seconds per drop)
- Viscosity of the liquid
- And additional notes.

This information as well as the date will be written to a text file in tab separated value form. Said file will then be saved with the following designation: *`title_experiment#.txt`*

Setting the Voltage

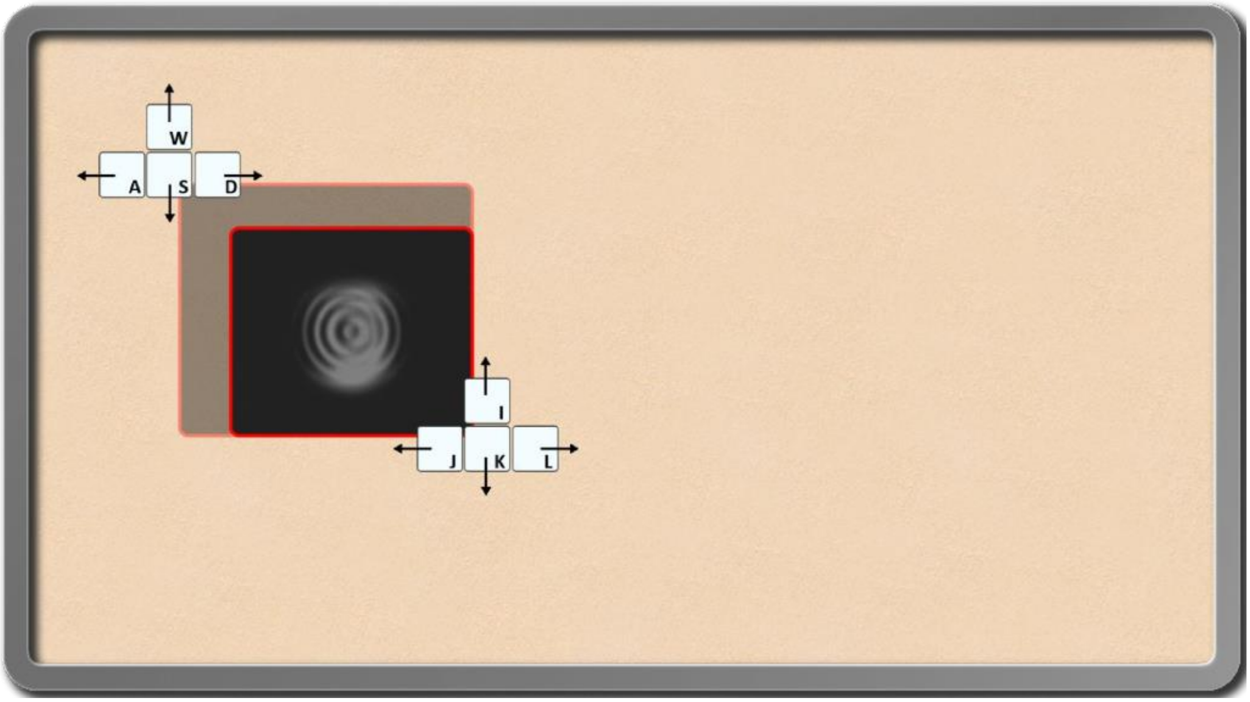
Once the initial information is set, a window with the image feed will pop-up. The program will then loop so that the user may set a voltage value correspondent to their desired drop rate. This will be done through the use of the `+` and `-` keys[1] to increase and decrease voltage respectively. Once the voltage is set the *`0`* key will be used to exit the loop.

* It is recommended to wait before confirming the desired voltage with the *`0`* key. This is in order to ensure the voltage-drop rate correspondence as the system has a high degree of latency.

Setting the Frame

Once the correct voltage is set, the user will be prompted to adjust the section of the feed to be processed (designated by a red rectangle) to center around the drop, in order to reduce extraneous noise. This is done by using the *`W`*, *`A`*, *`S`*, *`D`* keys to control the upper right corner and the *`I`*, *`J`*, *`K`*, *`L`* keys to control the bottom right corner as depicted in the image below.

[1] All key presses must be done in the image feed in order to be registered



Terminating the Program

Once the model has been saturated to satisfaction or at any point beforehand the user may press *q* key to exit the program. Upon which the program will write the final voltage, total number of drops, average of changing pixels, and average of changing pixels when there is a drop. The user will then be prompted to add any additional notes which will also be written to the file. The file will then be closed and saved, the video feed shut off, and the voltage will slowly decrease until 0 to ensure the valve rests fully shut.

Additional Features

At any point in time several keys may be pressed to alter values and enhance performance. Apart from the *q* key, whose function is explained above, the *v*, *r*, *+*, and *-* keys are also interrupts. The *v* key allows the user to reset the desired voltage. The *r* key correspondingly allows the reselection of the frame to be processed. The *+* and *-* keys, on the other hand, increase or decrease both the current voltage and desired voltage by ≈0.5%.

Data Collection

After the initial information is written to the text file, the following information will be written every frame:

- Time since initialization
- Time since previous drop
- Frame number
- Average of moving pixels
- Moving pixels in the current frame
- Drop detection (T/F)
- Contiguous drop frame
- Current voltage

Appendix A: Materials

- 1x PV14 Electronically Controlled Proportioning Valve

- 1x Raspberry Pi 4

- 1x MCP4725 12-bit DAC

- 1x 2-input BNC connector

- 1x USB-interface Camera (various options)

- 1x Illumination source (Various options)

Appendix B: Software

CEESClasses.py

```
import time
import datetime

class Experiment:
    # Constructor
    def __init__(self, title = 'Auto', exp = '000', user = 'Default', spd = 2, visc =
50, notes=''):
        # Independednt Data
        self.title = title
        self.experiment = exp
        self.date = datetime.datetime.now()
        self.user = user
        self.secPerDrops = spd
        self.viscosity = visc
        self.iNotes = notes
        self.filename = self.title + "_" + self.experiment + ".txt"
        self.began = time.time()

        # Drop Data
        self.optimalVoltage = 0
        self.dropData = []
        self.lastDrop = 0
        self.dropSum = 0
        self.volts = 0

        # Noise data
        self.noiseData = []
        self.noiseLength = 0   # number of noise frames
        self.noiseSum = 0     # sum of noise frames

    # Getters
    def vGet(self): return self.volts
    def dropAvg(self):
        return self.dropSum / len(self.dropData)

    # Setters
    def vSet(self, key):
        if key == 0:   self.volts = input("Please enter a voltage: ")
        elif key == 1: self.volts += 5
        elif key == 2: self.volts -= 5

        # Bound checking
        if self.volts > 4055: self.volts = 4055
        elif self.volts < 5:  self.volts = 5
```

```
        print("Voltage: {:.2f}%".format(100 * (self.volts / 4050)))

    def dSet(self):
        self.optimalVoltage = self.volts
        return True

    # Miscellaneous
    def addNoise(self, frameNumber, nonzero):
        self.noiseLength += 1
        self.noiseSum += nonzero
        self.noiseData.append((frameNumber, nonzero))
        return self.noiseSum / self.noiseLength

    def addDrop(self, frameNumber, nonzero):
        # data storage
        self.dropData.append([time.time() - self.began, time.time() - self.lastDrop,
frameNumber, self.noiseSum / self.noiseLength, nonzero, self.volts])
        self.dropSum += nonzero

        # voltage calculations
        if (self.lastDrop != 0):
            self.volts = self.optimalVoltage + ((time.time() - self.lastDrop) -
self.secPerDrops) * self.viscosity
            print("{:.2f}s since last drop".format(time.time() - self.lastDrop))
            print("Voltage: {:.2f}%".format(100 * (self.volts / 4050)))
        self.lastDrop = time.time()

        # Bound checking
        if self.volts > 4055: self.volts = 4055
        elif self.volts < 5:  self.volts = 5

        return self.volts

    def addNotes(self, frameNumber):
        self.fNotes = "Notes at frame" + frameNumber + input("Final Notes: ")

    def finalNotes(self):
        self.fNotes = "Final notes" + input("Final Notes: ")

    def terminate(self):
        # Open file in write mode
        dropFile = open(self.filename,"w")

        # Write initial information
        dropFile.write("Title: {}\n".format(self.title))
        dropFile.write("Experiment Number: {}\n".format(self.experiment))
        dropFile.write("Date: {}\n".format(self.date))
        dropFile.write("User/s: {}\n".format(self.user))
```

```
        dropFile.write("Viscosity: {}\n".format(self.viscosity))
        dropFile.write("DPS: {}\n".format(self.secPerDrops))
        dropFile.write("Notes: {}\n".format(self.iNotes))
        dropFile.write("TotalTime\tTimeSinceDrop\tFrame\tMovingPixelAvg\tMovingPix-
els\tVoltage")

        # Write drop data
        for drop in self.dropData:
            dropFile.write("{}\t{}\t{}\t{}\t{}\t{}\n".format(drop[0], drop[1],
drop[2], drop[3], drop[4], drop[5]))

        # Write final information
        dropFile.write("Final voltage: {}\n".format(self.volts))
        dropFile.write("Total drops: {}\n".format(len(self.dropData)))
        dropFile.write("Average pixel delta: {}\n".format(self.noiseSum /
self.noiseLength))
        dropFile.write("Average drop delta: {}\n".format(self.dropAvg()))
        dropFile.write(self.fNotes)

        # Close drop file
        dropFile.close()

        # Write separate file for noise data
        noiseFile = open(self.title + "_" + self.experiment + "_Noise", "w")

        # Write noise
        for noise in self.noiseData:
            noiseFile.write("{}\t{}\n".format(noise[0], noise[1]))

        noiseFile.close()
```

```
# Import libraries
from CEESClasses import Experiment
import adafruit_mcp4725
import numpy as np
import cv2 as cv
import datetime
import board
import busio
import time

# Define functions
def initialize():
    # Initialize I2C bus and MCP4725 board
    i2c = busio.I2C(board.SCL, board.SDA)
    dac = adafruit_mcp4725.MCP4725(i2c)

    # Initialize Experiment object
    title = "v4" #input("Title: ")
    exp = input("Experiment #: ")
    user = "I"   #input("User/s: ")
    spd = 2      #float(input("Desired drop Rate: "))
    visc = 60    #float(input("Liquid Viscosity: "))
    notes = "N/A"#input("Enter any additional notes: ")
    e = Experiment(title, exp, user, spd, visc, notes)
    return e

def imageProcessing(backSub, frame, coordinates):
    north, south, east, west = coordinates

    gray = cv.cvtColor(rect_img, cv.COLOR_BGR2GRAY)
    fgMask = backSub.apply(gray) #this is basically 80% of the program right here
    fgMask_RGB = cv.cvtColor(fgMask, cv.COLOR_GRAY2RGB)

    cv.rectangle(frame, (10, 2), (100,20), (0,0,0), -1)
    cv.putText(frame, str(capture.get(cv.CAP_PROP_POS_FRAMES)), (15, 15),
               cv.FONT_HERSHEY_SIMPLEX, 0.5 , (255,255,255))

    #Replacing the sketched image on Region of Interest
    frame[north: south, west : east] = fgMask_RGB
    nonzero = cv.countNonZero(fgMask)
    return frame, nonzero

def fSet(coordinates, key):
    validKeys = [ord('w'), ord('W'), ord('a'), ord('A'), ord('s'), ord('S'),
                 ord('d'), ord('D'), ord('i'), ord('I'), ord('j'), ord('J'),
                 ord('k'), ord('K'), ord('l'), ord('L'),]
```

```
    north, south, east, west = coordinates

    # Top-Right Corner
    if key == ord('w') or key == ord('W'):   north -= 5
    elif key == ord('a') or key == ord('A'): west -= 5
    elif key == ord('s') or key == ord('S'): north += 5
    elif key == ord('d') or key == ord('D'): west += 5

    # Bottom-Left Corner
    elif key == ord('i') or key == ord('I'): south -= 5
    elif key == ord('j') or key == ord('J'): east -= 5
    elif key == ord('k') or key == ord('K'): south += 5
    elif key == ord('l') or key == ord('L'): east += 5

    # Bound checking
    if (north < 0): north = 0
    elif (north > height): north = height
    if (south < 0): south = 0
    elif (south > height): south = height
    if (east< 0): east = 0
    elif (east > width): east = width
    if (west < 0): west = 0
    elif (west > width): west = width

    if key in validKeys: print("Coordinates: ({}, {}), ({}, {})".format(north, west,
south, east))
    return [north, south, east, west]

def clogProtocol(currentVolts, fgMask):
    unclogVolts = currentVolts
    latencyDelay = time.time()
    while (nonzero < pix_avg * 3):
        dac.raw_value = unclogVolts
        nonzero = cv.countNonZero(fgMask)
        if (latencyDelay-time.time() > 10): unclogVolts += 81
    while(unclogVolts > currentVolts):
        unclogVolts -= 81;
        time.sleep(0.1)

def summary():
    print("Final voltage: {:.2f}".format(voltage))
    print("Total drops: {}".format(num_drops))
    print("Average pixel delta: {:.2f}".format(pix_avg))
    print("Average drop delta: {:.2f}".format(dp_avg))

def terminationProcedure(volts):
    capture.release()
    cv.destroyAllWindows()
```

```
    volts =int(volts/10)*10
    while (volts >= 10):
        volts -= 5
        #dac.raw_value = volts
        time.sleep(0.1)

if __name__ == '__main__':
    # Initialize settings
    e = initialize()
    frames= 0
    liquid = True
    defaults = False
    calibration = False
    backSub = cv.createBackgroundSubtractorMOG2(history = 40, varThreshold = 60, de-
tectShadows = False)

    # Open video feed
    capture = cv.VideoCapture(0) # 0 - laptop webcam; 1 - USB webcam; "cv.sam-
ples.findFileOrKeep(args.input))" - file
    if not capture.isOpened:
        print('Unable to open: ' + args.input)
        exit(0)

    # get image dimensions
    height = int(capture.get(4))
    width = int(capture.get(3))
    coords = [0, height, width, 0]

    while (liquid):
        # Takes frame input from camera
        ret, frame = capture.read()
        if frame is None:
            break

        # Rectangle marker
        r = cv.rectangle(frame, (coords[3], coords[0]), (coords[2], coords[1]), (100,
50, 200), 3)
        rect_img = frame[coords[0]:coords[1], coords[3]: coords[2]]

        # Main processing of program
        if (defaults):
            frames += 1
            frame, delta = imageProcessing(backSub, frame, coords)
            if(calibration):
                if(delta > pixAvg * 5): e.addDrop(frames, delta)
                else: pixAvg = e.addNoise(delta)
            else:
```

```
                time.sleep(0.01)
                if (frames < 250):
                    pixAvg = e.addNoise(delta)
                    print(delta)
                else:
                    print("CALIBRATION COMPLETE")
                    calibration = True

        # Show the image in a resizeable frame
        cv.namedWindow('Frame',cv.WINDOW_NORMAL)
        cv.imshow('Frame', frame)
        k = cv.waitKey(60) & 0xFF

        # Checks for keypresses
        if ((k == ord('q')) or (k == ord('Q'))): liquid = False
        elif (k == ord('0')): defaults = e.dSet()
        # confirm default voltage and rectangle size and location
        elif ((k == ord('r')) or (k == ord('R'))): defaults = False
        # don't save data while default voltage and rectangle values are reset
        elif ((k == ord('c')) or (k == ord('C'))): calibration = False
        # calibration sequence restarts (100 frames to average noise level)
        elif ((k == ord('n')) or (k == ord('N'))): e.addNotes()
        # insert additional notes, find out how to parallelize
        elif ((k == ord('e')) or (k == ord('E'))): e.vSet(0)
        elif (k == ord('+')): e.vSet(1)
        elif (k == ord('-')): e.vSet(2)
        # set voltage
        elif (k != 0xFF): coords = fSet(coords, k)
        # set frame
        dac.raw_value = e.vGet()

    terminationProcedure(e.vGet())
    e.finalNotes() # Make parallel with termination procedure
    e.terminate()  # ^^
```

Appendix C: Specifications

Raspberry Pi 4 B (RPi):

Symbol	Parameter	Minimum	Average	Maximum	Unit
V_{PSU}	Power Supply Voltage	-0.5	4.9	6.0	V
I_{PSU}	Power Supply Current	0.600	-	3.0	A
I_{USB}	USB Peripheral Draw	0	-	1.2	A
$V_{GPIO-Out}$	Pin Output Voltage		-	3.3	V
$I_{GPIO-Out}$	Pin Output Current	-	-	16[1]	mA
T_{SoC}	System on Chip Temperature	-45	40	85	°C
T_{LAN}	USB-Ethernet Controller	0	40	70	°C

[1] Total current from all GPIO pins must not exceed 51 mA

Table 1: Operating Specification – Rpi

Symbol	Parameter	Minimum ***High***	Maximum ***Low***	Unit
$V_{GPIO-Out}$	Pin Output Voltage	1.6	-	V
$V_{GPIO-in}$	Pin Input Voltage	1.3	0.6	V

Table 2: Logic High-Low Value – Rpi

MCP4725 12-bit DAC:

Symbol	Parameter	Minimum	Average	Maximum	Unit
V_{PSU}	Power Supply Voltage	2.7	-	5.5	V
I_{PSU}	Power Supply Current	-	210	400	µA
V_{Out}	Output Voltage	2.7	-	5.5	V
I_{Out}	Output Current	4	-	20	mA
T_{DAC}	Operating Temperature	-40	-	125	°C

Table 3: Operating Specifications – MCP4725

PV14 Electronically Controlled Proportioning Valve:

Symbol	Parameter	Minimum	Average	Maximum	Unit
V_{PSU}	Power Supply Voltage	12	-	24	V
I_{PSU}	Power Supply Current	2.5	5	5	A
$V_{Control}$	Valve Control Voltage	-	-	5	V
$I_{Control}$	Valve Control Current	4	-	20	mA
T_{PV14}	Operating Temperature	0	-	49	°C

Table 4: Operating Specifications – PV14

Appendix D: Planned Improvements

Hardware

- Pi-Camera to optimize GPU usage and enhance image quality
- Pi-driven LED lighting system to enhance drop detection
- Frame for camera and lighting system to ensure constancy
- Implement initialization upon power-on
- Set remote access protocol

Software

- Decrease required user interaction
    - Adjust voltage level, program auto-records drop rate
    - Autostart and End, no info required
- Improve user interface
    - Needs GUI
- Auto-set frame based on moving pixels
- Develop machine learning capabilities
    - To set and improve drop definition (in moving pixels)
    - To improve algorithm based on viscosity and seconds per drop
    - To improve backSub settings
- Remote access into Pi for checkups (VNC)

Experimental

- Set upper voltage bounds to avoid stream (per viscosity)
- Test maximum permissible flow rate in order to enhance saturation speed

Miscellaneous

- Convert drop rate to m^3/s for replication of system
- Save unprocessed saturation video feed to test system with same data, but different parameters
- Derive equation for pressure and flow rate

Appendix E: Key Table

Key	Function
0	`Exits calibration sequence`
+	`Increases voltage`
-	`Decreases voltage`
q	`Terminates (quits) the program`
r	`Resets frame adjustment sequence`
v	`Resets voltage calibration sequence`
w	`Moves top-left corner of frame up`
a	`Moves top-left corner of frame left`
s	`Moves top-left corner of frame down`
d	`Moves top-left corner of frame right`
i	`Moves bottom-right corner of frame up`
j	`Moves bottom-right corner of frame left`
k	`Moves bottom-right corner of frame down`
l	`Moves bottom-right corner of frame right`

NOTE:

The Automated Saturation System is under revision.

Feedback on bugs and desired improvements is welcome.