CEES

NEES

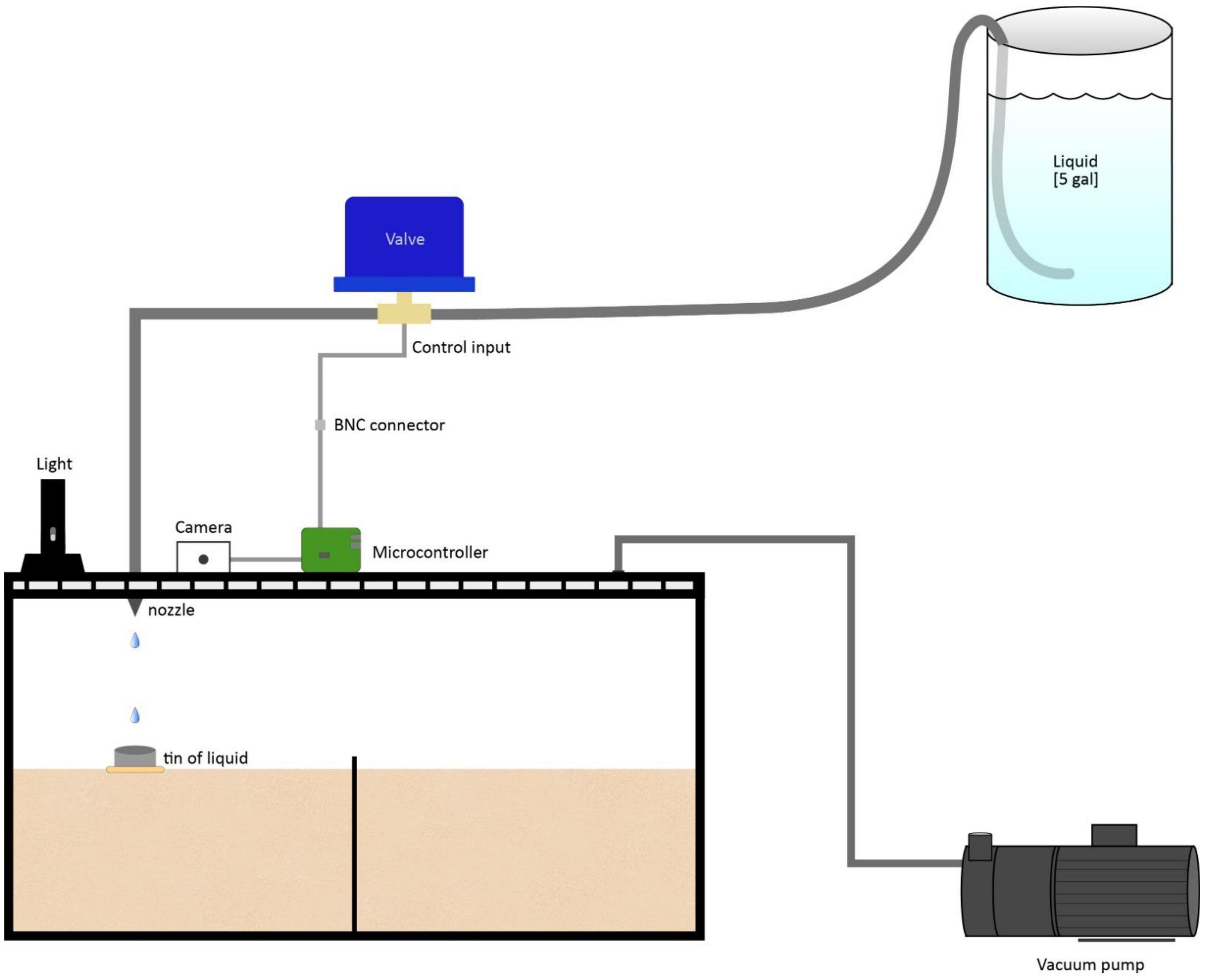
AUTOMATED SATURATION SYSTEM

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 wired to a digital to analog converter that emits a calculated DC 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

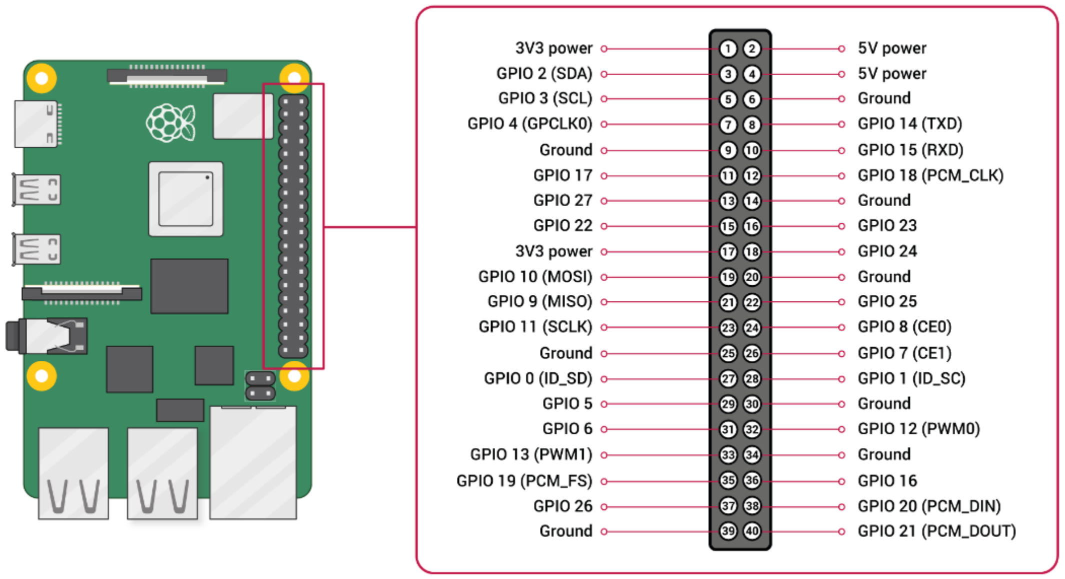
Prerequisites for Startup

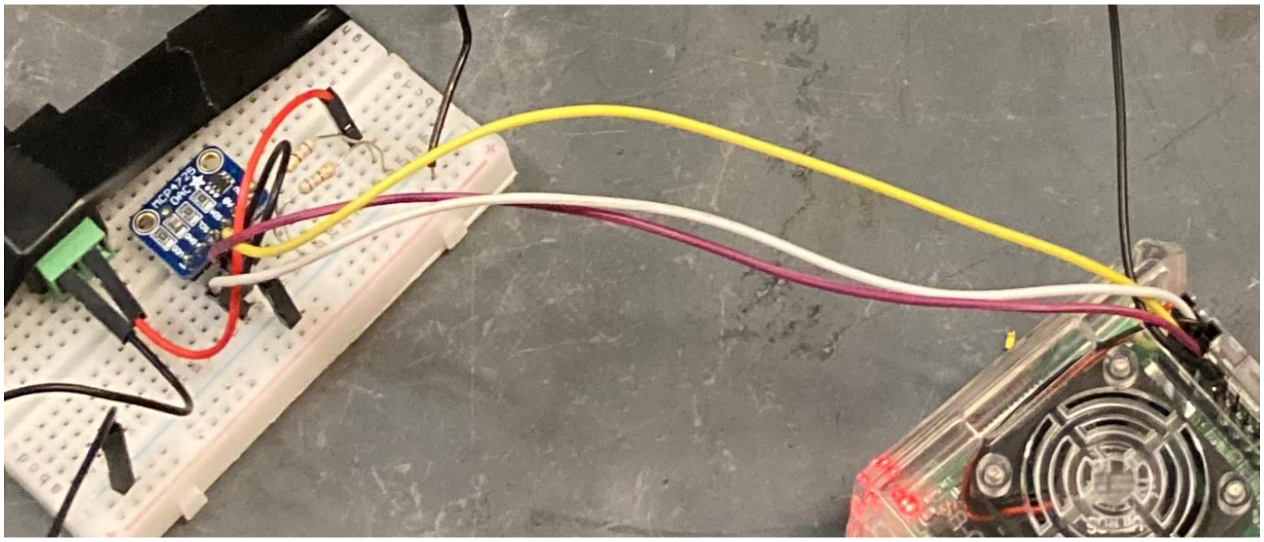
There are two primary ways to run the program:

1. Run it on the Raspberry Pi, with its own display, keyboard, and mouse, as a standalone computer
2. Run it via a remote access program on a Windows 10 device

If choosing Option 1, simply start up the Raspberry Pi and use the username and password, both "pi". Otherwise, refer to the following section for instructions using a Windows computer.

The following pictures show the wiring layout required for the system, as of May 2021. The digital-analog converter is controlled via I2C and receives power from the Raspberry Pi GPIO.

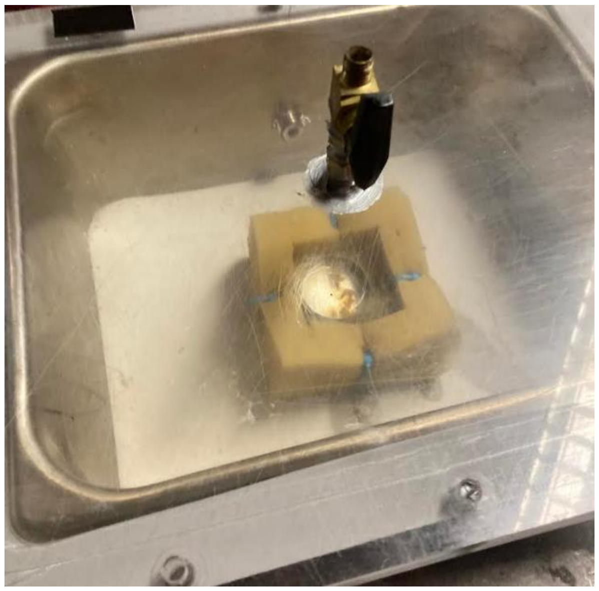




Wiring Table:

Wire Color	Connection 1 (Raspberry Pi)	Connection 2 (DAC)
White	5V Power (Pin 2)	VDD
Black	Ground (Pin 9)	GND
Yellow	SDA (Pin 3)	SDA
Purple	SCL (Pin 5)	SCL

When setting up the soil model, place a small tin (the size of a bottle cap or so) in the center of the soil and sponge so that the drops will fall into the tin. This will make the program's job of detecting drops easier by producing reflections from the overhead light and is highly recommended. Finally, connect the Raspberry Pi and the voltage-controlled valve with the BNC cable.



Prerequisites for Startup (Remote Access via Windows)

These prerequisites are only applicable when the Raspberry Pi is being run through a Windows device, in headless mode. If the Pi is plugged into its own display (mini-HDMI), keyboard, and mouse (both USB-A), these perquisites are not required for using the Raspberry Pi.

If operating the Pi through a Windows device (Windows 10 is assumed), firstly install PuTTY on your computer. Additionally, install VNC Viewer on the device.

https://www.realvnc.com/en/connect/download/viewer/

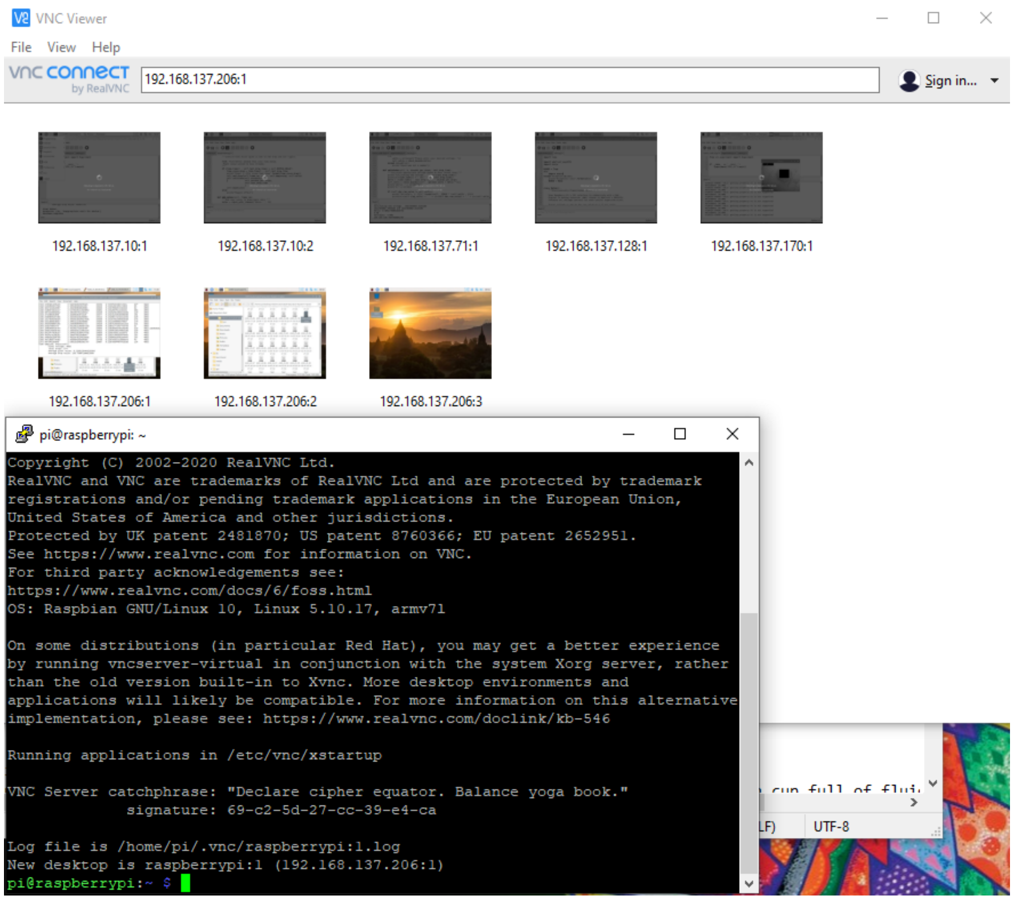
Then, plug an Ethernet cable into both the computer and the Pi. For connecting for the first time, make sure the computer can share network connections over Ethernet. Use this tutorial for instructions on how to configure the settings properly.

https://www.tomshardware.com/how-to/share-internet-connection-windows-ethernet-wi-fi

Then, open up PuTTY and use the host name "raspberrypi.local" and port 22, to connect via SSH. After clicking Open, a command prompt should open up asking for login information. For this Pi, both the username and the password are "pi".

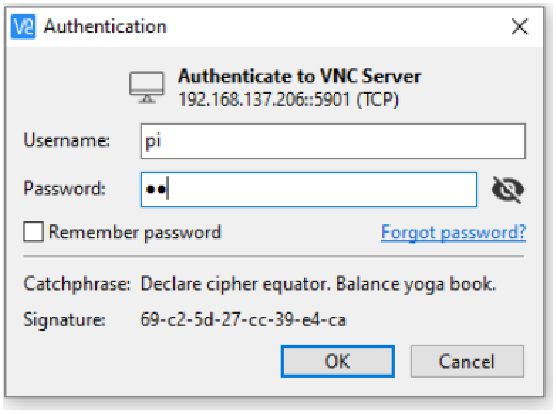
Occasionally, the prompt will not add an asterisk when a key is pressed as part of the password, though don't worry, the password is still recorded even though an asterisk may not show up on the screen.

After pressing enter there should be some confirmation that the Pi is connected. Now, type "vncserver" into the command prompt.



```
Copyright (C) 2002-2020 RealVNC Ltd.
RealVNC and VNC are trademarks of RealVNC Ltd and are protected by trademark
registrations and/or pending trademark applications in the European Union,
United States of America and other jurisdictions.
Protected by UK patent 2481870; US patent 8760366; EU patent 2652951.
See https://www.realvnc.com for information on VNC.
For third party acknowledgements see:
https://www.realvnc.com/docs/6/foss.html
OS: Raspbian GNU/Linux 10, Linux 5.10.17, armv7l

On some distributions (in particular Red Hat), you may get a better experience
by running vncserver-virtual in conjunction with the system Xorg server, rather
than the old version built-in to Xvnc. More desktop environments and
applications will likely be compatible. For more information on this alternative
implementation, please see: https://www.realvnc.com/doclink/kb-546

Running applications in /etc/vnc/xstartup

VNC Server catchphrase: "Declare cipher equator. Balance yoga book."
             signature: 69-c2-5d-27-cc-39-e4-ca

Log file is /home/pi/.vnc/raspberrypi:1.log
New desktop is raspberrypi:1 (192.168.137.206:1)
pi@raspberrypi:~ $
```

A lot of text should show up (see the above screenshot), all that matters is the IP address that's displayed near the end of the output. *If the displayed IP address starts with **anything** other than "192"*, power off the Pi, unplug the Ethernet cable, essentially restart the startup process with powering up and connecting to PuTTY. The exact cause isn't known though any other IP address means that there was some issue with connecting to the internet via your computer.



Username: pi

Password: ••

Remember password

Catchphrase: Declare cipher equator. Balance yoga book.

Signature: 69-c2-5d-27-cc-39-e4-ca

If an IP starting with “192” does show up, open up VNC Viewer and copy the IP into the address bar of the program. Press Enter, put in the same login information again, and the viewer should be a virtual Raspberry Pi desktop. Now, you’re able to run the program!

Running the Program

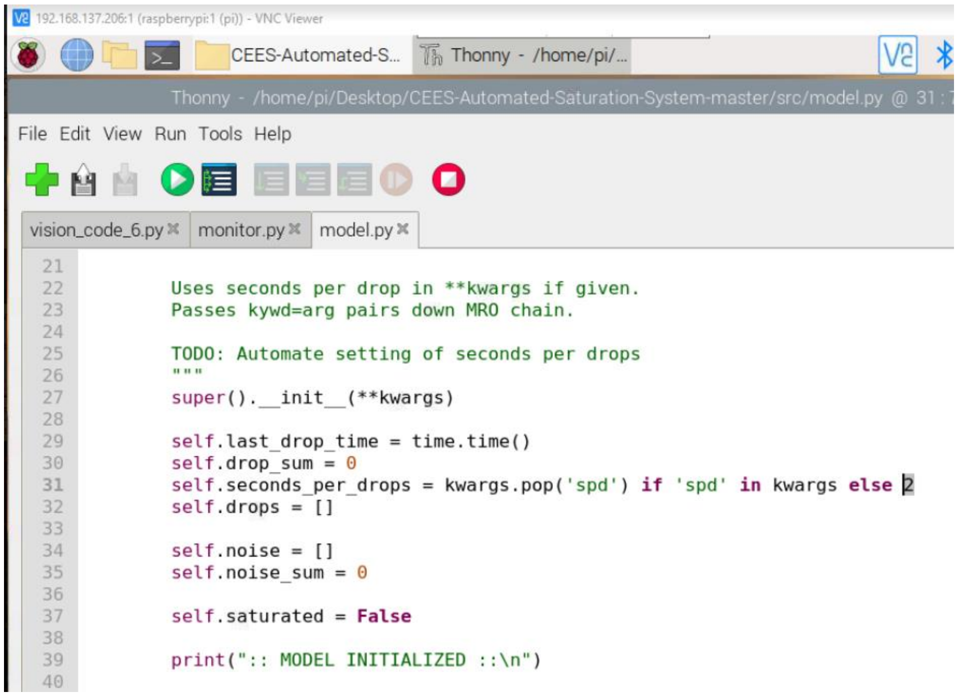
On the Desktop of the Pi, open the folder “CEES-Automated-Saturation-System”. It should be the only folder on the Desktop. Here, logs from previous runs of the program are stored and the code is stored in the folder “src”. To run the program, however, open and run the file *vision_code_6.py* in the top-level folder.

After the program is run, the user will be able to control the starting voltage (not required), as well as the region of interest for the camera. Once those two variables are initialized, press the *0* key to begin the autonomous portion of the program.

It is suggested to monitor the system for a few minutes after startup to make sure the system works properly, though after pressing 0 the system will be fully autonomous and will run until there is no viscous fluid left to saturate the soil.

Changing Desired Drop Rate

The program works by determining the current drop rate and comparing it to a value set before starting the program. By default, this value is set to 2 seconds per drop, though to change it, open up the “src” folder, and then *model.py*. Then, change the value at the end of Line 31 to the desired value. Make sure to save the *model.py* file before running *vision_code_6.py*.


192.168.137.206:1 (raspberrypi:1 (pi)) - VNC Viewer
CEES-Automated-S...
Thonny - /home/pi/...
Thonny - /home/pi/Desktop/CEES-Automated-Saturation-System-master/src/model.py @ 31 :
File Edit View Run Tools Help
vision_code_6.py
monitor.py
model.py

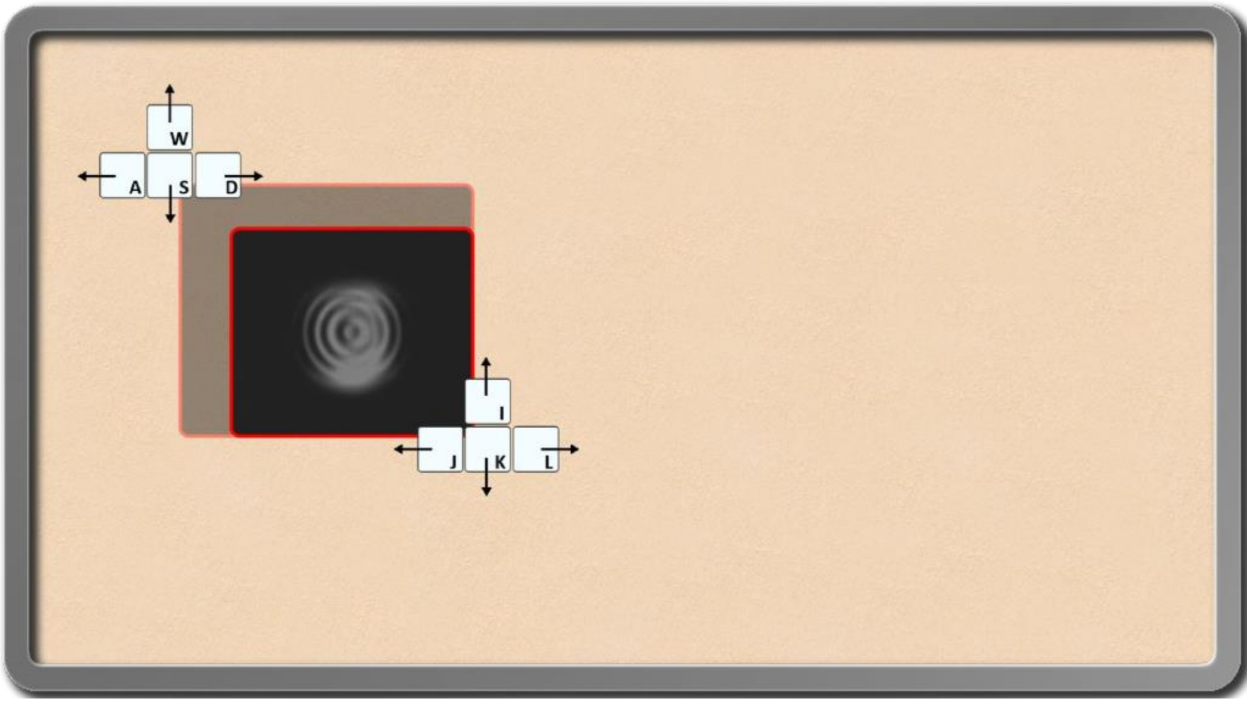
```
        Uses seconds per drop in **kwargs if given.
        Passes kywd=arg pairs down MRO chain.

        TODO: Automate setting of seconds per drops
        """
        super().__init__(**kwargs)

        self.last_drop_time = time.time()
        self.drop_sum = 0
        self.seconds_per_drops = kwargs.pop('spd') if 'spd' in kwargs else 2
        self.drops = []

        self.noise = []
        self.noise_sum = 0

        self.saturated = False

        print(":: MODEL INITIALIZED ::\n")
```

Setting the Voltage

Once the program has initialized, a window with the image feed will pop-up. Before starting the automated portion of the program, the user can use the *+* and *–* keys on the keyboard to set a starting voltage for the valve. The client will then print the voltage percentages, 0% to 100%, that the DAC is sending to the valve (0-5V). This step is **not** required for successful operation of the system, though if there are issues with the first few drops of the autonomous run this may be a method to improve performance. Before pressing the *0* key to end the startup phase and start the autonomous phase of the program, set the voltage so that the valve produces approximately the desired drop rate for the saturation.

Setting the Frame and Starting the Program

After running the program, 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.



Try to center the ROI as close to the drop as possible and move the camera and light source to minimize ripples.

When set to a reasonable location, press *0* to start the automated portion of the program. The video feed within the ROI will then go black, and all changing pixels will be shown as white. Make sure that the feed only shows white pixels when a drop lands, with as few exceptions as possible. The system has methods in place to ignore ripples, though the better the physical location, the better the system will perform.

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 program will then be closed, the video feed shut off, and the voltage will slowly decrease until 0 to ensure the valve rests fully shut. In the folder "CEES-Automated-Saturation-System-master", a log file will be created that shows data for each drop recorded by the system during the run.

Additionally, the program will automatically terminate if the clogging procedure reaches completion. In the event of a clog (no drops detected for 20+ seconds), the program will:

- Every increment of time (10+ seconds) without a drop, the valve opens a specified percentage.
- If the valve makes it to fully open without a drop, and there are no further drops for 1 minute, the program will automatically terminate.

Common Errors and Other Notes

This section is a list of common errors, notes, and other things to keep in mind when running the program.

1. If the program doesn't initialize due to an I2C error, it's likely due to a ground wire becoming loose, so check all of the connections on the breadboard.
2. If the drop rate is sufficiently fast, the program has difficulty differentiating between what is a drop and what is a ripple in the accumulating fluid. The program has experimentally been tested at rates as fast as 2-3 drops per second (0.33-0.5 seconds/drop), though any rate faster than 1 second per drop is not recommended due to the potential damage to soil models.
3. If running using a Windows machine and PuTTY/VNC Viewer, the program will continue running even if the computer is locked or goes into sleep. However, make sure the computer does not shut down as this may stop the program.
4. As of May 2021, the program has not been tested for consecutive periods of more than 3 hours. It is uncertain if any unforeseen effects appear after running the system for longer periods of time. Make sure to check on the system every few hours until the system is sufficiently proven and reset the system if needed.
5. When adjusting the ROI, the popup window will not be as big as the actual camera feed. All four sides of the red rectangle may not be visible, but after resizing the window or shrinking the ROI, it will appear.
6. The Region of Interest can be changed after the system is in autonomous mode, and this may be useful to center the ROI on the drops. However, with each keypress to change the ROI the system will consider every pixel in the ROI to have changed, recognize this mistakenly as a drop, and close the valve. This will have no effect on performance as the valve will close rather than open, though drop rate will temporarily drop after such a change.

7. The system takes a short amount of time after starting to lower the threshold for a drop. After starting, wait a few minutes to make sure the system properly detects all drops before walking away. If necessary, change the position of the camera and the light source.

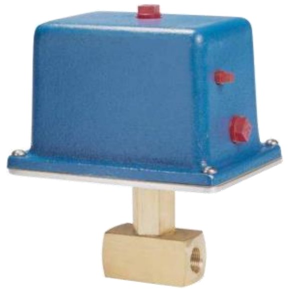
Data Collection

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

Author's Note: This is not updated as of June 16, 2021. Refer to the GitHub repository for the most updated 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()
```

Vision_Code.py

```
# 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
$V_{GPIO\text{-}Out}$	Pin Output Voltage		-	3.3	V
$I_{GPIO\text{-}Out}$	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
$V_{GPIO\text{-}Out}$	Pin Output Voltage	1.6	-	V
$V_{GPIO\text{-}in}$	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 and simplify setup
- Frame for camera and lighting system to improve positioning consistency
- Set remote access protocol

Software

- Improve user interface
    - Needs GUI
- Auto-set frame based on moving pixels
- Add a key to change seconds/drop value
- 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.