An Automated Electrochemistry Platform for Studying pH-dependent Molecular Electrocatalysis

Michael A. Pence,[1,2] Gavin Hazen,[1,2] Joaquín Rodríguez-López[1,2*]

[1] Beckman Institute for Advanced Science and Technology, University of Illinois Urbana-Champaign, Urbana, Illinois 61801, United States
[2] Department of Chemistry, University of Illinois Urbana-Champaign, Urbana, Illinois 61801, United States

*Corresponding author: joaquinr@illinois.edu

Table of Contents

Supplemental Note S1. Solution handling hardware assembly

The solution handling robot was assembled from the hardware in **Table S1**.

Name	Price	Quantity
OPENBUILDS 24V Meanwell Power Supply Bundle	$78.74	1
Runze SV07-X-T16-K1.0-S	$1,160.00	1
Runze SY08- LS-0.9-1-5-1-Q	$585.00	1
Runze USB-30	$39.00	2
Fluorostore 1/8” OD PTFE tubing	$1.47/ft	30
Fluorostore 1/16” OD PTFE tubing	$0.45/ft	2
Idex Flangeless Fitting, Standard Knurl, Natural PEEK, 1/8" OD Tubing, 1/4-28 Flat-Bottom	$5.77	2
Idex Flangeless Fitting, Standard Knurl, Natural PEEK, 1/16" OD Tubing, 1/4-28 Flat-Bottom	$5.67	15
Pyrex media bottles (100 mL -1 L)	$9 - $15	14

Table 1. *Bill of materials for the solution handling hardware.*

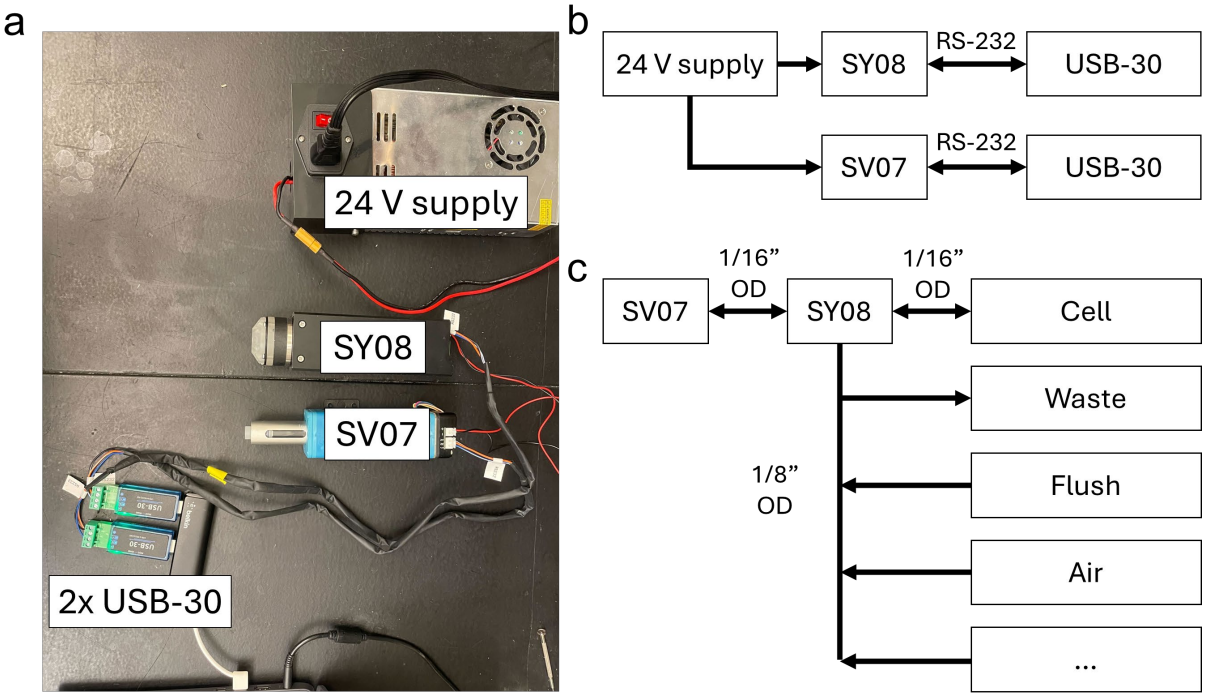

Figure S1. (a) *Electronically connected components of the solution handling hardware and* ***(b)*** *a schematic of the electronic connections.* ***(c)*** *Fluidic connections between components of the solution handling hardware and various solutions. Critical solutions (cell, waste, flush, and air) are shown, but other desired solutions can be connected as desired.*

The hardware components for the solution handling robot are depicted in **Figure S1a**, and assembled as shown in **Figure S1b**, as follows. First, plug in the power supply (24V) and then position the pump and valve accordingly. Connect the power wires to the pump and valve using

the red and black cables, making sure the polarity of the power terminals is correct. Next, plug in the USB/RS232 cables to both the pump and valve and connect the USB end of the USB/RS232 cable to the computer. Open the device manager to identify the COM ports to which the devices are connected as you plug them in. Turn on the power supply, and you should observe an illuminated LED on the pump and valve and hear a slight movement as the pump and valve initialize.

Fluidic connections are made as shown in **Figure S1c,** with additional media bottles of stock solutions hooked up as needed. Prepare each tube by attaching a flange and ferrule, ensuring the tubing extends slightly past the ferrule to minimize dead volume. Connect these tubes to the valve, threading them in until they are finger tight. **NOTE:** The tubing for solution lines can be 1/8" OD, but it is recommended that the fluidic connection between the pump and valve, as well as the solution line to the cell, are 1/16" OD tubing.

For non-volatile and non-hazardous solutions, inlets can be drilled into the lid of Pyrex media bottles so that fluidic tubing can be ran inside of the media bottle. Otherwise, it is suggested that well-sealed solvent reservoir caps be used, i.e VapLock Solvent Delivery Caps. Place the lines into their respective solutions, ensuring the following critical connections must be made:

- **Pump to valve:** Use 1/16" OD tubing instead of 1/8" OD and connect the pump to the center port of the selection valve. Use as short a connection as possible between these lines
- **Waste:** Connect to a container designated for waste disposal.
- **Cell:** Use 1/16" OD tubing instead of 1/8" OD and connect it to the electrochemical cell.
- **Flush:** Connect to the solvent intended for washing the cell.
- **Air:** Connect a line that leads to nothing, used to pull air into the lines and flush out the solution. Ensure this line is long enough to divert output solution away from the switching valve in case of a critical error.

Finally, assemble the electrochemical cell, as shown in **Figure 2**. Place the cell line into the electrochemical cell, ensuring it reaches the very bottom of the container. This completes the hardware assembly process.

Supplemental Note S2. Installing and using the elab API

The elab API is installed as follows:

1. Download and install the latest version of Python, ensuring that Python is added to PATH.
2. Download and install the latest version of Git.
3. Create a folder for installing the elab API.
4. Open terminal and change directory to the elab API location.
5. Type command “git clone https://github.com/jrlLAB/elabAPI.git” into terminal.
6. In the terminal, change the directory to the newly formed elabAPI folder where the setup.py file is located. Type command “pip install .”
7. In the terminal type command “pip install hardpotato” to install the Hard Potato library

The python libraries are now installed and can be used in a Python script by first importing the eLab library, as well as the Hard Potato library.

```
import elab
import hardpotato as hp
```

Next we will need to initialize the instruments to be used. For example, if we wanted to initialize the solution handling platform we would run commands to initialize the SV07 valve and SY08 pump at their respective COM ports.

```
valve = elab.SV07('COM1')
pump = elab.SY08('COM2')
```

We then use the elab.bundle() command to bundle multiple instruments into one object so that multi-instrument operations can be executed.

```
lab = elab.bundle([valve,pump])
```

We must establish pseudonyms for the switching valve so we can easily call commands to dispense desired solutions. **NOTE:** Many of the operations built into the API that use the switching valve rely on ports with specified pseudonyms of ‘cell', 'waste', 'air', and 'flush'. Pseudonyms are typically loaded from a .csv file that contains pseudonyms for the port #, formatted as below:

```
port, title
1, cell
2, waste
3, flush
4, air
5, tempo
6, buffer
…
```

The pseudonym file can be loaded in with either load_ports() method

```
lab.load_ports('ports.csv')
```

Or by passing a dictionary

```
lab.load_ports({'cell' : 1, 'waste' : 2, 'flush' : 3, 'air' : 4, 'tempo' :
5, 'buffer' : 6 })
```

We can run solution handling operations after the instruments are initialized, bundled, and setup with the port pseudonyms. If we are hooking up solution bottles to the instrument for the first time, we must initialize the lines, as there is dead volume that must be filled. Initializing lines will pass the solution through the selection valve directly into the waste port. The ports corresponding to the cell, waste, and air lines do not need to be initialized. For example, the solution lines defined in the above ports.csv file (or corresponding dictionary) are initialized as follows:

```
lab.init_line('flush')
lab.init_line('tempo')
lab.init_line('buffer')
```

To clean the cell by removing cell contents and flushing with solvent, we can run the following:

```
lab.clean_cell(10)
```

lab.clean_cell() takes a volume (in mL) as an input, and then cleans the experimental cell by removing that volume, and flushing with that volume in water. It requires a pump and switching valve to be connected to operate. It requires 'cell','waste','air', and 'flush' port pseudonyms.

If we wanted to dispense a solution of our choosing we would hook it up to the switching valve and give it an appropriate pseudonym. The following code would dispense 3 mL of a redox active species, TEMPO, as described in the above ports.csv file (or corresponding dictionary).

```
lab.dispense('tempo',3)
```

One could imagine that dispensing more complex mixtures could make the code pretty messy, so instead we can use the mix_dispense() instead. Here we are mixing 1 mL of TEMPO with 4 mL of a buffer solution. First lets define our components as a list of lab.mix_component() functions:

```
mix = [lab.mix_component('tempo',1),lab.mix_component('buffer',4)]
```

We now dispense our mixture

```
lab.mix_dispense(mix)
```

After we dispense our mixture, we can use the Hard Potato library to setup up our potentiostat and run a CV experiment, as shown below. Further Hard Potato documentation can be found at https://github.com/jrlLAB/hardpotato.

```
hp.potentiostat.Setup(model='chi760e',path='chi760e.exe', folder='.')
hp.potentiostat.CV(Eini=0,Ev1=1,Ev2=0,Efin=0,sr=0.1,sens=1e-5).run()
```

The attached Data.zip file contains all experimental scripts and data, organized by the corresponding figures in the main text. The experimental scripts are annotated so that users can see the experimental workflow and use the script as templates for experiments of their own design. Additionally, a full reference for the API can be found at https://github.com/jrlLAB/elabAPI .

Supplemental Note S3. pH meter hardware assembly and usage

The pH meter was assembled from components in **Table S2.**

Name	Price	Quantity
Arduino UNO Rev3 SMD	$26.30	1
Gravity: Analog pH Sensor - Meter Kit	$29.50	1
Mettler Toledo Micro pH Electrode; S7	$467.00	1
InLab Cable S7-BNC 1.2m	$86.94	1
USB-A to USB-B 2.0 Cable	$2.00	1

Table S2. *Bill of materials for assembling the Arduino pH meter*

The pH meter was assembled as shown in **Figure S2a.** The pH probe is connected to the breakout board circuit which sends an analog signal to the ADC pins on the Arduino. The analog signal from the breakout board is sent to the A1 analog input on the Arduino, and the 5V and ground pins on the breakout board are connected to the respective pins on Arduino.

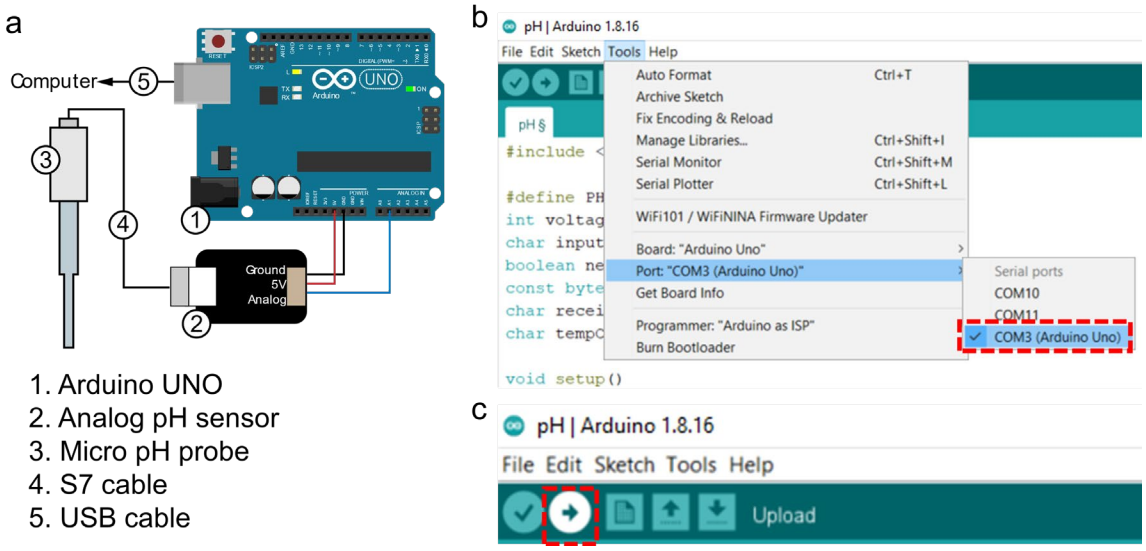

Figure S2. ***(a)*** *Schematic of the Arduino based pH meter and its assembly. The ground and 5V pins on the analog pH sensor are connected to the corresponding pin on the Arduino, and the analog data pin on the sensor board is connected to A1 on the Arduino.* ***(b)*** *Selecting the appropriate COM port in Arduino IDE.* ***(c)*** *Uploading the pH meter code from the Arduino IDE to the Arduino board.*

To communicate with the elab library the Arduino must first have the appropriate code uploaded:

1. Download and install the latest version of the Arduino IDE
2. Paste the code for the pH meter into the Arduino IDE. The code can be found at the following GitHub repository: https://github.com/jrlLAB/Arduino/blob/main/pH/pH.ino
3. Connect the Arduino UNO to the computer and select the appropriate COM port, as shown in **Figure S2b.**
4. Upload the Arduino code from the IDE to the Arduino board by hitting the upload button as shown in **Figure S2c.**

When the code has been uploaded to the Arduino board, the pH meter can be called using the eLab library. The general workflow for initializing the pH meter is like that in Supplemental Note S2. First, the eLab library must be imported

```
import elab
```

Then the valve, pump, and pH meter are initialized and bundled

```
valve = elab.SV07('COM1')
pump = elab.SY08('COM2')
pH = elab.pH_arduino('COM3')
lab = elab.bundle([valve,pump,pH])
```

Pseudonyms are established using the same method as described above, but if automated pH meter calibration requires ports be connected to pH standards. **Note:** The pH standards' port names must be in format of "pHX", where X is the pH value of the standard. Upon loading the port pseudonyms and initializing requisite lines, automated calibration can be performed as shown for pH 4, 7, and 10 buffer.

```
lab.calibrate_pH([4,7,10])
```

This call will automatically dispense the pH standards, measure corresponding voltage values for each pH value, and automatically perform linear regression to generate a calibration curve. pH values can be measured on that calibration curve by using the following command:

```
lab.pH.measure()
```

Supplemental Note S4. Adding new instruments to the elab API

Adding an instrument to the elab API requires modification of the underlying python library. Here we will show the process of modifying the library with a new instrument. The first step is to clone the library into a new folder, and then navigate to the elabAPI\src\elab folder, which will contain all of the .py files for the library. We are going to write a class for our API to control the IKA C-Mag HS7 hotplate, so we create a new file called HS7.py.

In this python file we first will import the instrument class from main.py, which will allow us to inherit the default initialization method, which initializes serial communication with the instrument over a specified COM port and with a set baud rate (default is 9600). We also add some identifiers and variable, such as the max temperature (self.max_temp) that we would like the hotplate to go up to.

```
from .main import instrument

class HS7(instrument):

    def __init__(self, com_port, **kwargs):
        super().__init__(com_port, **kwargs)

        self.model = 'C-MAG HS7'
        self.type = 'hotplate'
        self.max_temp = 50
```

After setting up these initial parameters, we will add functions to the HS7 class that will translate high level python commands into ASCII commands that will be sent over serial to the instrument. First lets look at some of the commands that can be found in the IKA C-Mag HS7 datasheet. Note this list is not comprehensive, and we are adding just commands we are interested in.

Command	ASCII	Notes
Read actual external sensor value	IN_PV_1	
Read actual hotplate sensor value	IN_PV_2	
Adjust set temperature value	OUT_SP_1 x	X can be value from 0-500 *C
Adjust set stirring value	OUT_SP_4 x	X can be value from 0-1500 rpm
Start heating	START_1	
Stop heating	STOP_1	
Start stirring	START_4	
Stop stirring	STOP_4	

Table S3. *Example of ASCII commands for the IKA C-Mag HS7 datasheet.*

We can then write a function that will send the desired ASCII command over serial, encoded as UTF-8. We will show an example of a simpler command first, setting the speed of the stirring motor. The code will take an input variable that will define the speed of the motor in RPM, spin, and combine it into a string with the appropriate command from the data sheet, OUT_SP_4. For this instrument we must have a carriage return, so we add \r\n to the end of our command strings. We have an if/else statement to ensure that the speed is within the bounds specified in the data sheet. We then use the self.ser.write() method to encode the compiled ASCII command as UTF-8 and send it over serial to our instrument.

```
def set_spin(self, spin):
        if (1500 >= spin  > 0) == True:
            packet = f'OUT_SP_4 {spin}\r\n'
        else:
            packet = f'OUT_SP_1 {1500}\r\n'
            print(f'Value out of range! Set to {1500}')
        self.ser.write(packet.encode('utf-8'))
```

Some commands are used to query values, for example, IN_PV_1 will query the measured temperature response of an external thermocouple probe. Our function for this command must also include a way to read the resulting serial communication that is sent from the instrument back to the computer. The formatting of the received message can be quite different for every instrument, and this can take some time to understand the best way to receive message. If we send IN_PV_1 command to the hotplate, we receive the following as a response: `'21.4 1\r\n'`. The returned string gives us our value, and then has a space as a space before the carriage return. We can split the string at the space, as a delimiter, and extract and return the temperature value as a float.

```
def query_temp(self):
        packet = 'IN_PV_1\r\n'
        self.ser.write(packet.encode('utf-8'))
        return float(self.ser.readline().decode().split(' ')[0])
```

We can continue to write the remainder of the commands as needed. Different instruments will require different levels of involvement in reading and writing commands, and will require an effort from the user to find and interpret the commands in the data sheet. It is assumed that any instrument to be incorporated has a USB port for serial communication and a datasheet that provides a guide to communicating with the instrument over serial.

After we have setup all of the commands, we can add our new class to the library. We go to __init__.py file in elabAPI\src\elab folder, and edit that to include our new class using the following line.

```
from .HS7 import *
```

We also must adjust the __all__ list to include the HS7.py content in the library.

```
__all__ = ['main', 'HS7', 'pH_arduino', 'SV07', 'SY08',
           'E0RR80', 'AlicatMFC', 'Legato100', 'gen_serial', 'MUX8']
```

Now that our new class had been added, we navigate to the elabAPI folder, which contains the file setup.py. We open the terminal in this folder, and run the command ‘pip install .’ This will remove any previously installed versions of the elab API and install the new modified version.

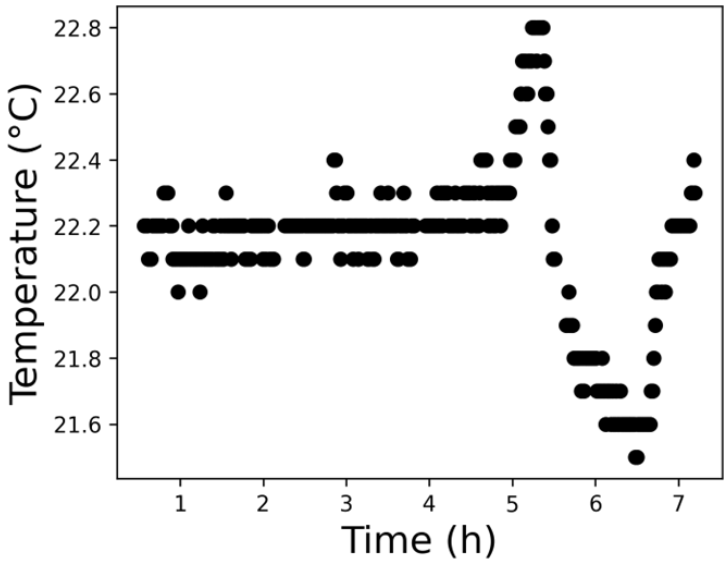

Figure S4. *Temperature changes over the course of titration of 0.1 M H_3PO_4 with 1 M NaOH. Temperature remains relatively constant through the majority of the experiment, with an average temperature of 22 ºC ± 0.2 ºC.*

Supplemental Note S5. Testing of different cleaning procedures

One important aspect of solution handling for automated experiments is implementing cleaning protocols that minimize sample carryover. We used cyclic voltammetry to test different cleaning protocols, measuring the voltammetric current response of TEMPO oxidation. The experimental script involved ten dispensing steps, alternating between 6 mM and 120 µM of TEMPO, with the cell being cleaned before each step. We tested three cleaning protocols: the first involved simply removing cell contents without any rinsing, the second included a water rinse following removal of cell contents, and the third consisted of the removal of cell contents, a water rinse, and priming of the cell with the solution to be dispensed. **Table S4** shows the mean and standard deviation of the measured currents for each protocol. From the results at low concentrations, we can see that water flushing reduces sample carryover, while priming ensures that residual water does not dilute the redox species and lead to a lower current than expected. The lowest standard deviation for the measured current at both concentrations was seen when using the water flush and priming.

Cleaning protocol	6 mM TEMPO current	120 µM current
Cell removal	84 ± 3 µA	4.6 ± 0.2 µA
Water flush	82 ± 1 µA	1.09 ± 0.03 µA
Water flush and priming	86.5 ± 0.6 µA	1.17 ± 0.02 µA

Table S4. *Comparison of the average (N=5) peak currents obtained using three different cleaning protocols.*

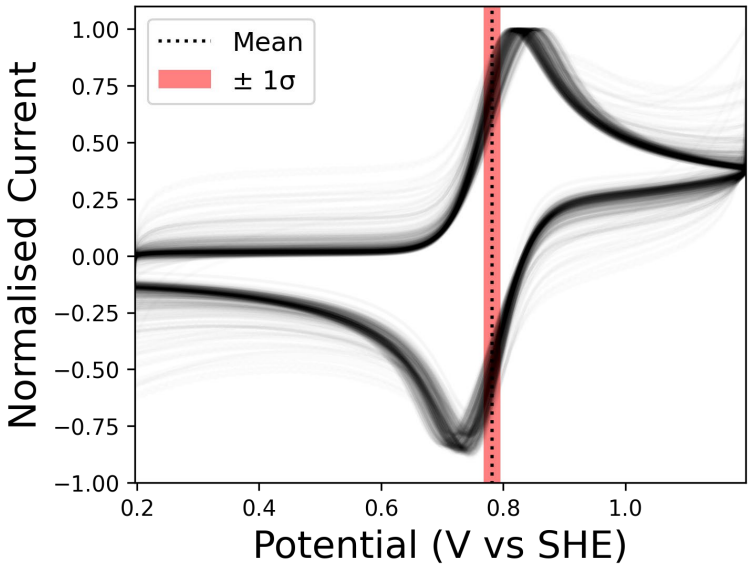

Figure S5. *All 400 voltammograms collected during measurement of TEMPO diffusion coefficient, with current normalized to the peak current. The potential was measured against an Ag/AgCl reference and is reported here against SHE. The mean* $E_{1/2}$ *value was found to be 0.78 V ± 0.01 V vs. SHE.*

Supplemental Note S6. Baseline subtraction

Baseline subtraction was performed by fitting the linear portion of the CV, and subtracting the peak current, from the corresponding current of the linear fit at the peak potential. The script for performing baseline analysis first smooths the current data, identifies points that likely belong to the baseline by analyzing the gradient, fits a linear regression to those points, and calculates an adjusted peak value based on this baseline. The three best and three worst fits of the diffusion coefficient data are shown in **Figure S6.** The baseline fitting was applied to all CVs used for obtaining TEMPO diffusion coefficient, and the resulting change in calculated diffusion coefficient distribution is shown in **Figure S7.**

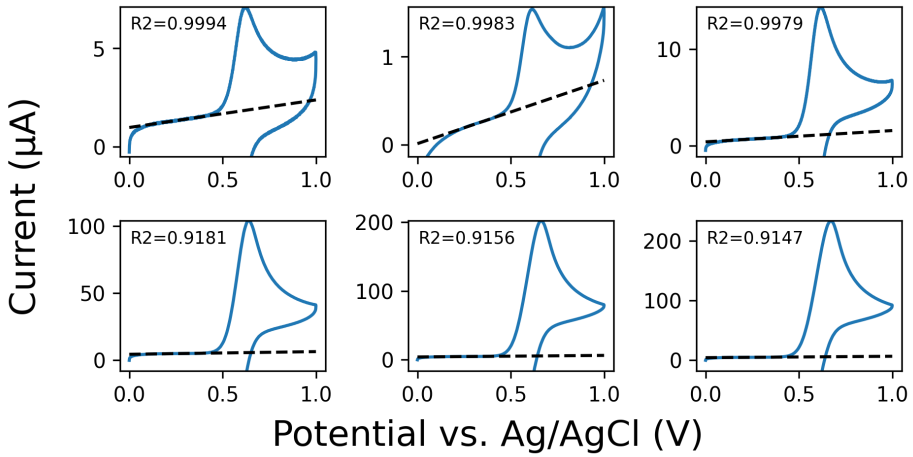

Figure S6. *Selected examples of the baseline fitting. The top row represents the three best fits, and the bottom row represents the three worst fits, determined by the R^2 value of the fit. The solid line is the experimental data and the dotted line is the linear fit.*

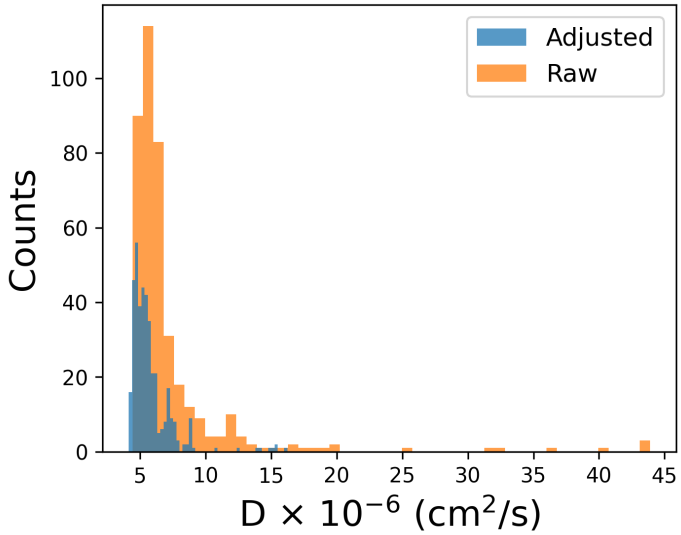

Figure S7. *Distributions of the diffusion coefficients calculated from the raw peak heights, compared to the distribution from diffusion coefficients calculated from peak heights adjusted by background subtractions.*

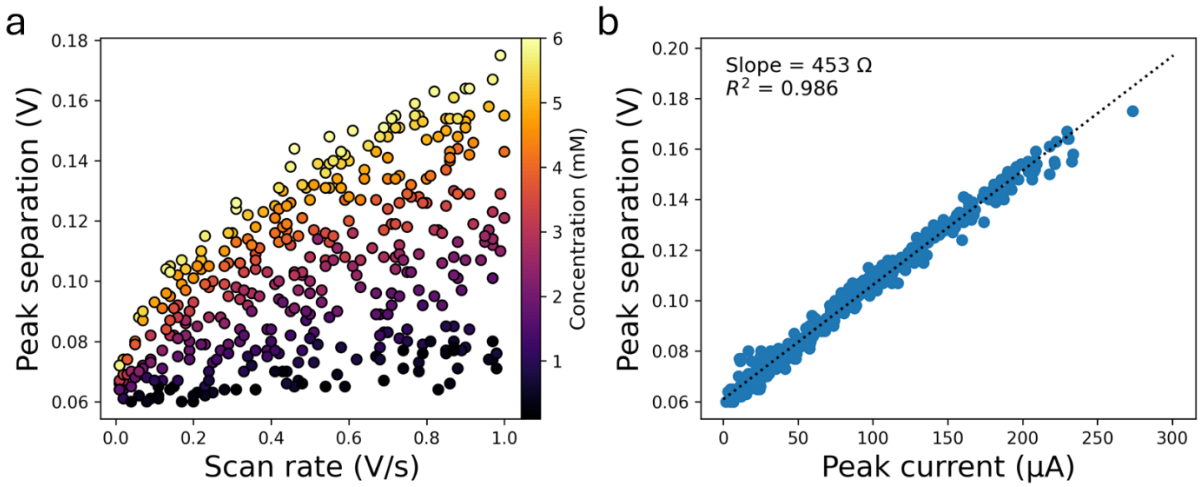

Figure S8. (a) *Plot of peak separation versus scan rate for all concentration of TEMPO.* ***(b)*** *Plot of peak separation versus the maximum peak current for all TEMPO values, as well as the resulting linear fit.*

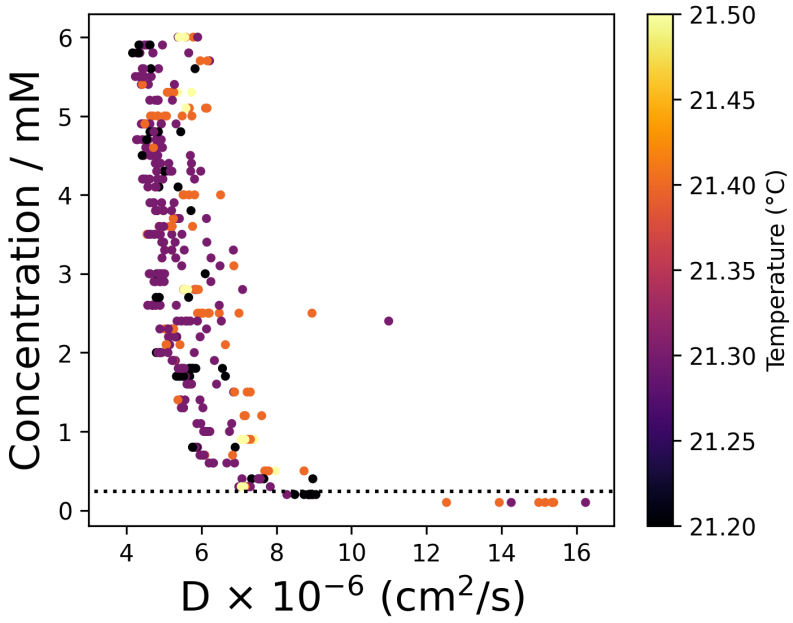

Figure S9. *Calculated diffusion coefficients of TEMPO plotted against the target concentration.*

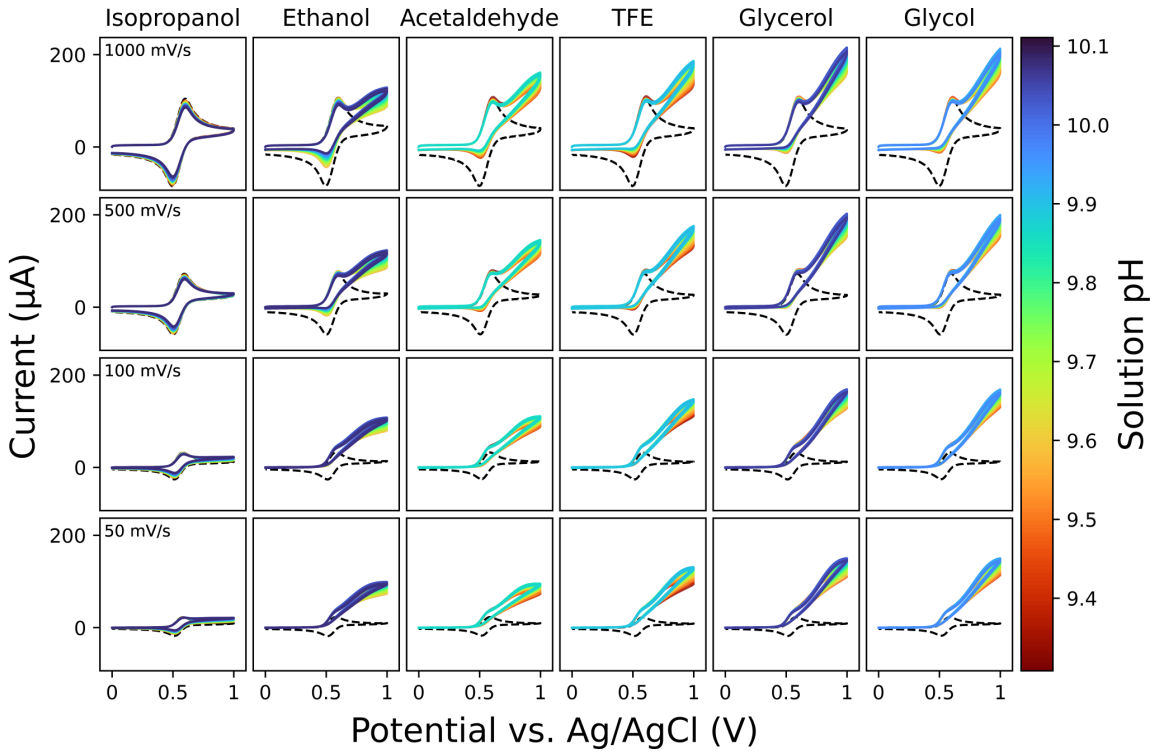

Figure S10. *Results of pH dependent voltammetric screening of TEMPO catalyzed alcohol oxidation, without a surface pretreatment. Each column represents a different substrate molecule, and each row is a different scan rate. Dashed lines are voltammograms in the absence of substrate.*

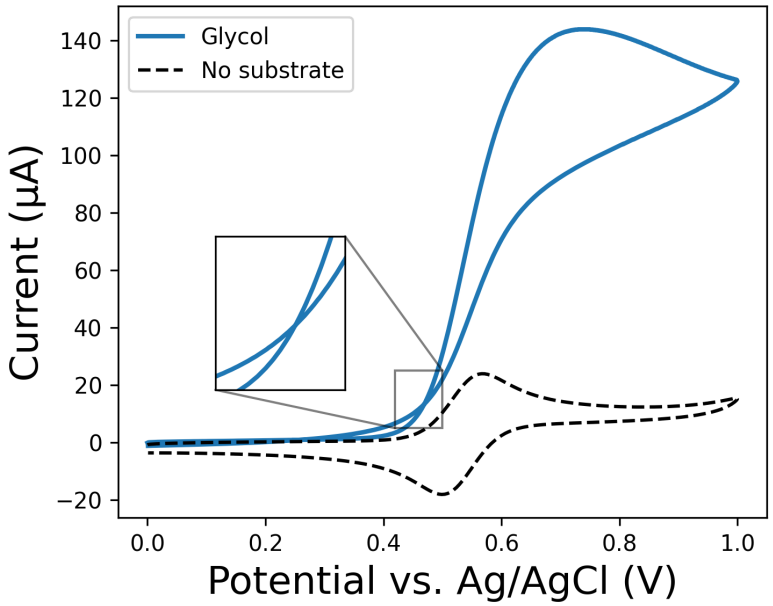

Figure S11. *Example of curve crossover seen in CVs of 2 mM TEMPO and 100 mM glycol in 0.1 M bicarbonate-carbonate buffer. Voltammograms were performed at 50 mV/s.*

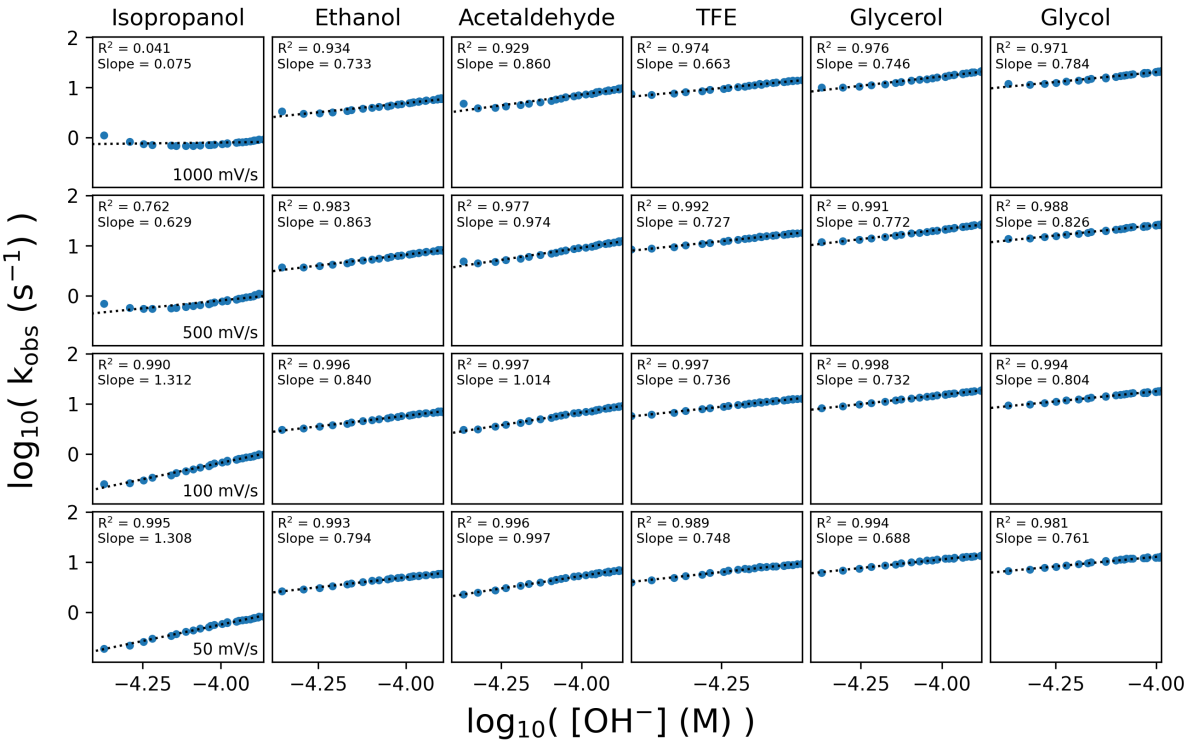

Figure S12. *Relationship between observed reaction kinetics and OH^- concentration for different substrates. The observed rate constant is plotted against the OH^- concentration on a log-log scale, along with the linear fit of the data. Each column represents a different substrate molecule, and each row is a different scan rate.*