Real Time Analysis of a Photoresistor using GUI & Arduino

MatLab Project

Bryce Abernathy & Christian Fontejon

EGEE 215 - MatLab

Professor Saravana Raman

May 7, 2019

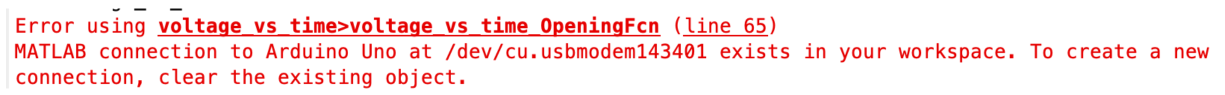
We first started this project by planning to read a potential difference across an LED and then to create a graph of voltage vs. time that would be running in real time using a GUI. We were going to be doing this through the use of arduino and matlab. However, as we began to tackle this project, we quickly ran into complications and gradually worked our way to what our final project now is. That is, using a photoresistor instead of an LED. There were several steps that we took to get to our final project and we will explain these steps later on in this report. But first, let us briefly explain the tools we will be using.

A Photoresistor, also referred to as a Light Dependent Resistor or photocell, is a resistor whose resistance decreases when the intensity of the light increases. It is commonly used as a sensor to detect changes in brightness. When light falls onto the photoresistor, electrons absorb energy, breaking the bonds with parent atoms, creating what is known as free electrons. Free electrons that are moving freely from one place to another carry an electric current. The amount of electric current flowing through the system depends on the number of free electrons generated. An increase in electric current means a decrease in resistance. This means that the resistance of the photoresistor will decrease when more light is exposed. For example, streetlights often use photoresistors to detect when it should turn on. When the surrounding light falls onto the photoresistor, it causes the streetlight to turn off. When there is no light, the photoresistor will cause the street light to turn on. A few advantages to photoresistors are that they are small in size, have a low cost, and that they are easy to carry from one place to another. The only downside to them is that the accuracy of photoresistors are very low, meaning that they do not always function as they are supposed to.

Arduino is an open-source platform used for building and designing electronics projects. Arduino consists of both a physical programmable circuit board and its IDE (Integrated Development Environment) that can run on any computer. With the software, you are able to write and upload code to the physical board. The Arduino IDE uses a simplified version of C++, making it easier for users to learn to program. Arduino hardware and software was designed for really anyone interested in creating interactive objects or environments. The Arduino hardware allows you to interact with buttons, LEDs, motors, speakers, GPS units, cameras, the internet, personal devices, etc.

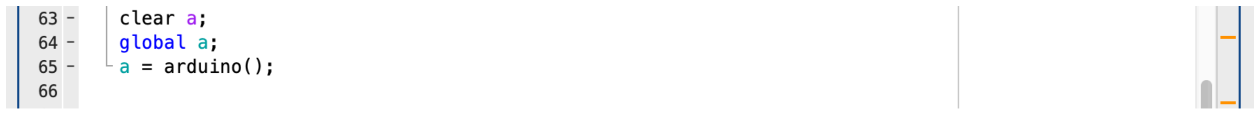
A GUI (Graphical User Interface) is an interface that uses icons as well as other visual indicators to interact with electronic devices rather than just text and a command line. A GUI uses windows, icons, and menus to carry out commands, such as opening, deleting, and moving files. GUI operating systems can be navigated using both a mouse and a keyboard, which can be used for keyboard shortcuts and arrow keys. GUI operating systems are very intuitive and easy to learn because commands do not have to be memorized like in a command line operating system. A few recognizable GUI operating systems may be Microsoft Windows, Apple's macOS, Chrome OS, and Linux's Ubuntu.

As we explained the tools we were using, we will now discuss the use of them in our first phase of our project. As mentioned, we initially started under the idea that we could read a voltage through an LED and then graph that voltage over time to create a nice graph on matlab. Before we discuss the results, let us discuss first how we tackled this initial idea. We first started by creating the GUI. We knew that we were going to need the graph, and a slider that we could use to control the voltage going through the LED. The GUI is very easy to build so let us not explain the dragging of boxes to create it but rather the brains behind it, the code. The GUI creates a format of code once one has saved it and lets the coder begin to customize within it. We first had to figure out where to place the code within all the different functions and thanks to the internet, we understood where to put our code. We then stated by creating a global variable a and then declaring a as the arduino. The reason we are clearing it is because matlab does not like having multiple objects being declared under the same variable. This is the error you would get if you do not clear a:



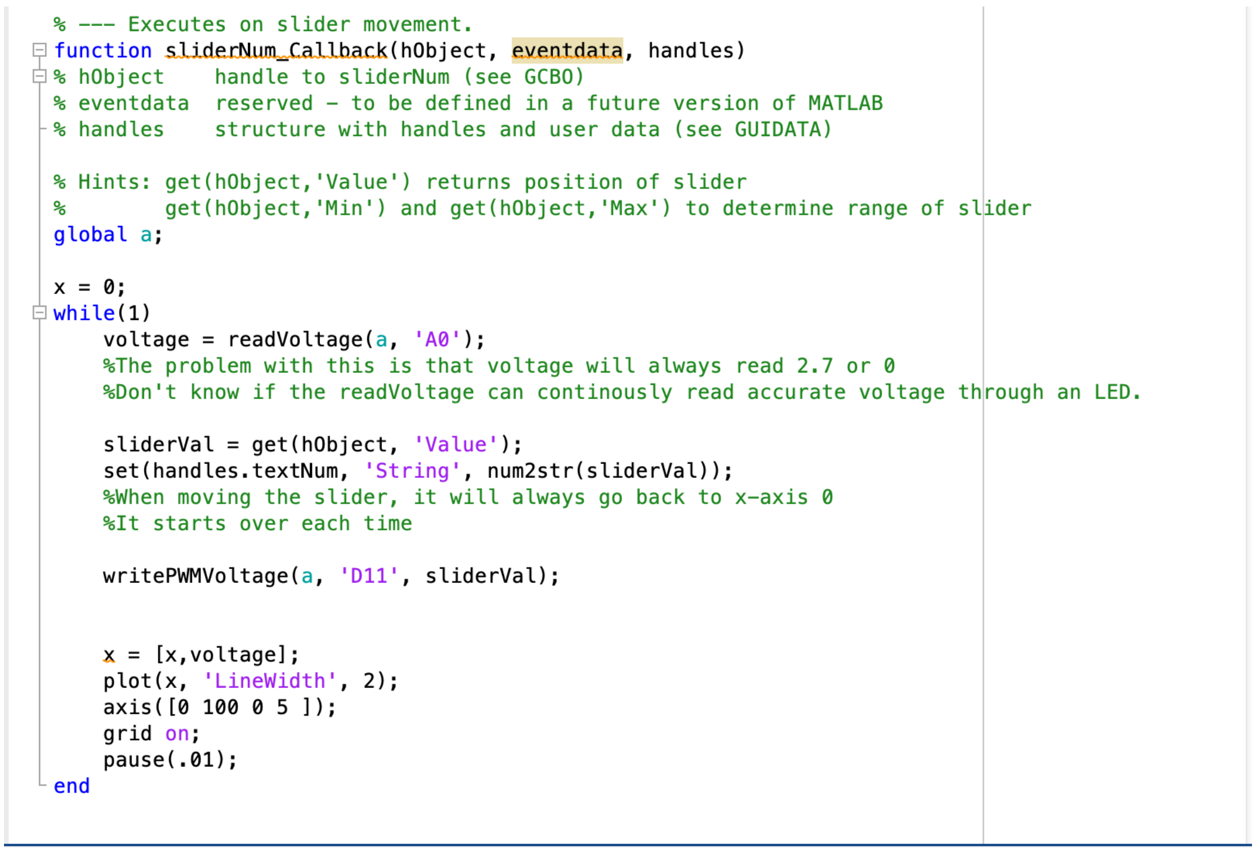
```
Error using voltage_vs_time>voltage_vs_time_OpeningFcn (line 65)
MATLAB connection to Arduino Uno at /dev/cu.usbmodem143401 exists in your workspace. To create a new
connection, clear the existing object.
```

This is the code for setting up arduino:



```
clear a;
global a;
a = arduino();
```

This is done to allow matlab to communicate with arduino. And I should mention that to even declare arduino in matlab, one must download the arduino hardware support package that matlab has on their website. Furthermore, once arduino is now declared and global, we can begin to use it in other functions. This leads us to the main code as it is the only other code we shall need to implement our desired results. Here is the code we developed:



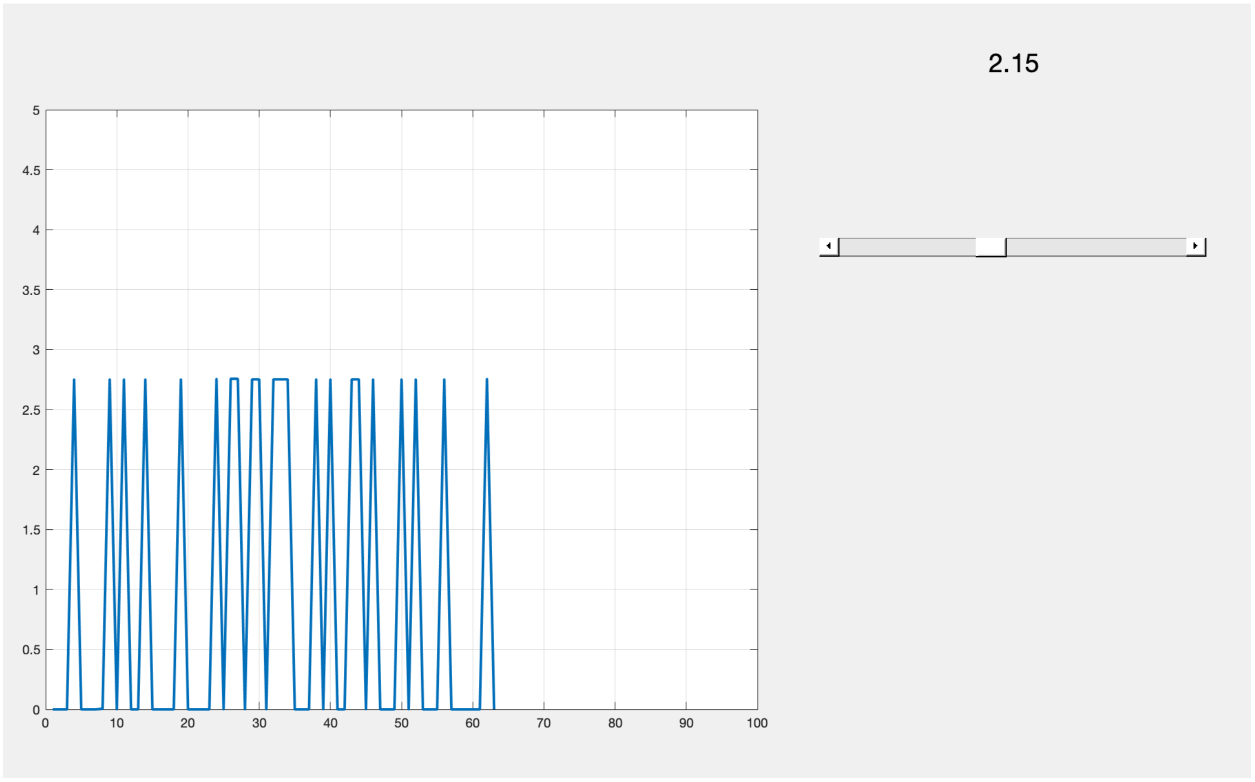
```
% --- Executes on slider movement.
function sliderNum_Callback(hObject, eventdata, handles)
% hObject    handle to sliderNum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global a;

x = 0;
while(1)
    voltage = readVoltage(a, 'A0');
    %The problem with this is that voltage will always read 2.7 or 0
    %Don't know if the readVoltage can continously read accurate voltage through an LED.

    sliderVal = get(hObject, 'Value');
    set(handles.textNum, 'String', num2str(sliderVal));
    %When moving the slider, it will always go back to x-axis 0
    %It starts over each time

    writePWMVoltage(a, 'D11', sliderVal);


    x = [x,voltage];
    plot(x, 'LineWidth', 2);
    axis([0 100 0 5 ]);
    grid on;
    pause(.01);
end
```

As you can see, we declared our global variable a and then moved to the meat of the code. We created an infinite while loop, that way we can read in real time, and then went to work inside of it. We started reading the voltage passing the LED through the readVoltage function and then stored it in voltage. The next part is how the slider works. When we move the slider, the value that it reads will then get stored in sliderVal. We then had to turn that into a number which is what the set() function is effectively doing. After we can get the slider number, we should send that number as voltage to our LED. This is done using matlab's writePWMVoltage() function and you can see that we are sending that voltage through Digital Pin 11 which is capable of PWM.

After we got the code for the hardware working properly, we moved to creating the graph on our GUI. We created a variable x that was equal to to itself for the x-axis and the voltage being the y-axis. We then plotted x to create the graph and set a line width for spatial purposes.

Then, we set our x and y axis lengths to fit the graph properly. After that, we made sure we had a grid and paused the delay as if we did not, the graph would run too fast to make sense of. And after all those steps, we ended our loop.

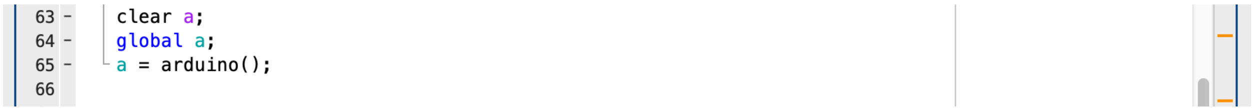
Here was the result:



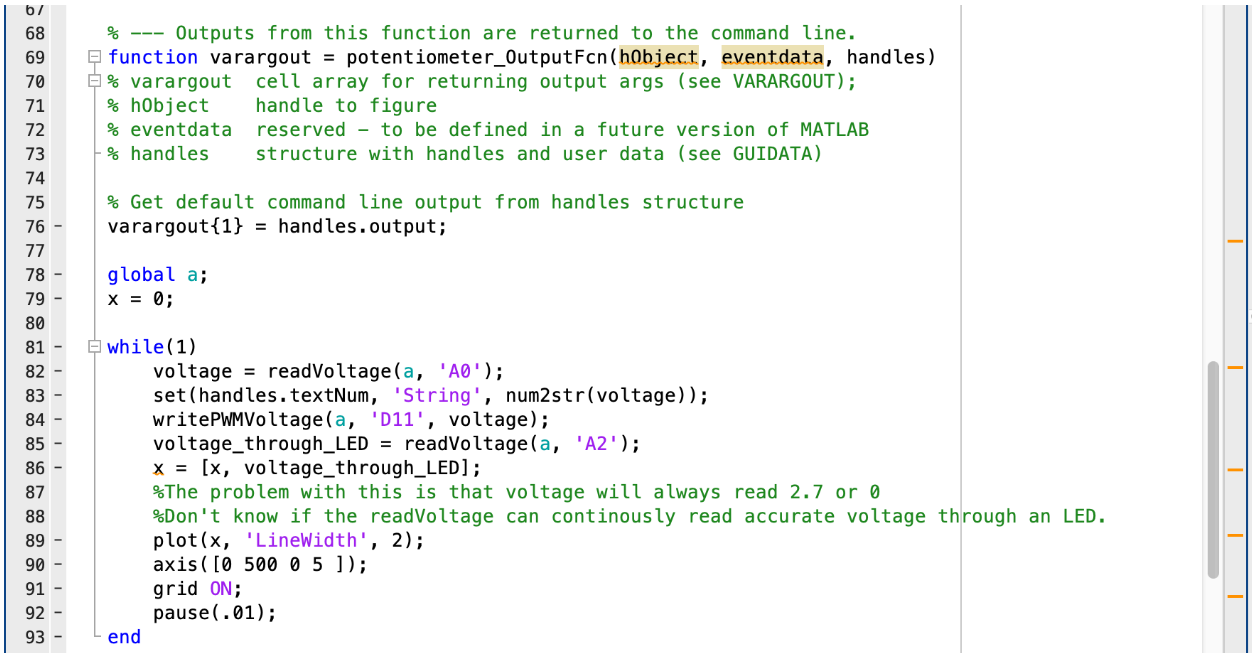
Once we ran our GUI, we began to notice some unwanted results; our graph would reset each time the slider moved and it would also not change on the y - axis. It stayed at exactly the same position. Something was surely wrong with our code and or what we were trying to accomplish was possibly not possible. It took us quite a while to finally agree that it was the latter that was the problem and not the code. We discovered, or should I say rediscovered, that the LED acts just like a boolean. It is either true or false. In our case, it is either high or low.

Since we are using PWM to send the voltage through the LED, it is impossible to read it using matlabs readVoltage function. We tried using both digital and analog pins to read the voltage and both had the same unworking results. This then led us to move on to our next idea, using a potentiometer instead of the built in GUI slider.

Our code is primarily the same as it was before. We set up arduino, made it global, and then got to work in our main code.



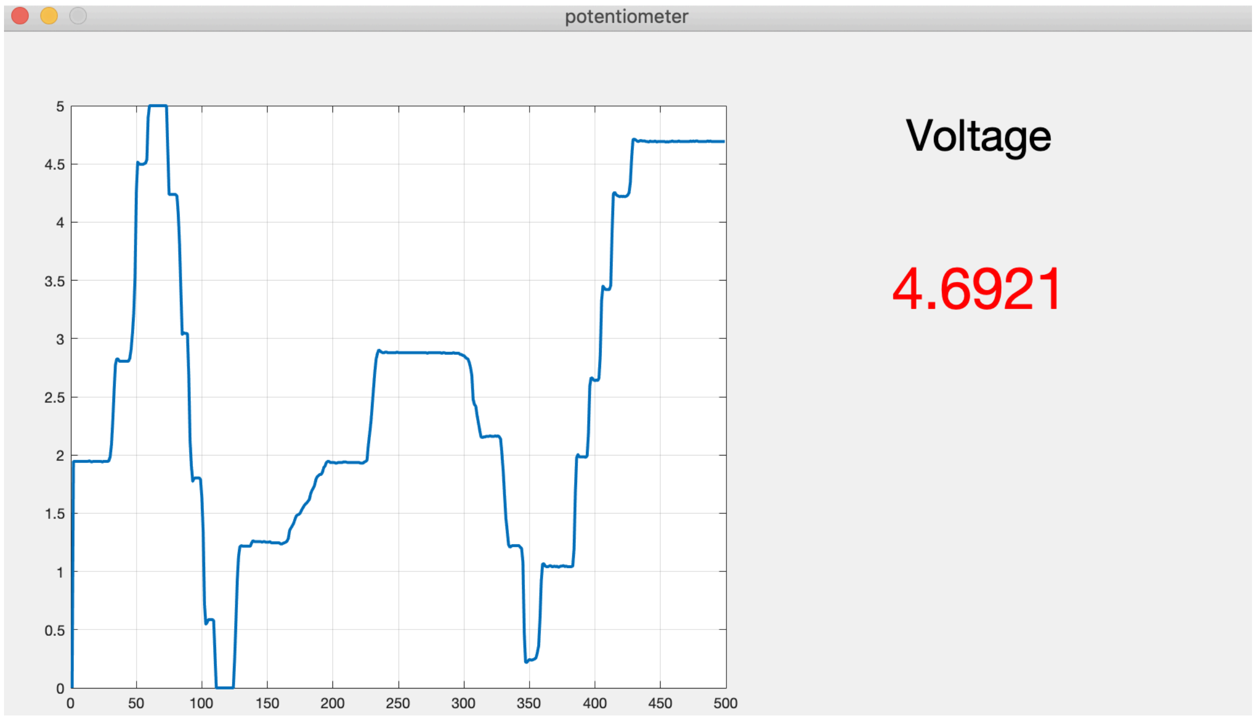
```
clear a;
global a;
a = arduino();

```



```

% --- Outputs from this function are returned to the command line.
function varargout = potentiometer_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

global a;
x = 0;

while(1)
    voltage = readVoltage(a, 'A0');
    set(handles.textNum, 'String', num2str(voltage));
    writePWMVoltage(a, 'D11', voltage);
    voltage_through_LED = readVoltage(a, 'A2');
    x = [x, voltage_through_LED];
    %The problem with this is that voltage will always read 2.7 or 0
    %Don't know if the readVoltage can continously read accurate voltage through an LED.
    plot(x, 'LineWidth', 2);
    axis([0 500 0 5 ]);
    grid ON;
    pause(.01);
end
```

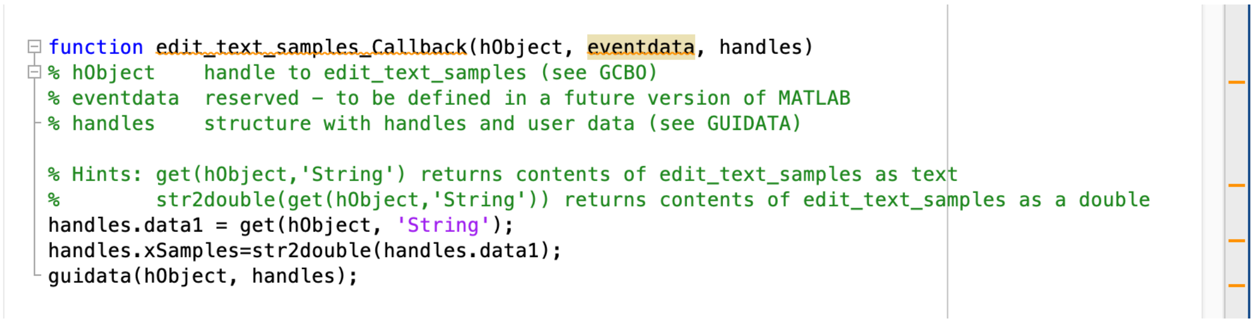
Our code was very similar to its predecessor but had some slight alterations. Inside of our infinite while loop, we first are reading the voltage at analog pin 0 which is connected to our potentiometer. We then had a text box in our GUI that was set up to display the voltage through our potentiometer. We then used the potentiometers voltage number to control the voltage going through the LED. After that, we then tried to read the voltage as before to see if that would make a difference in our graph. And lastly, we set up the graph.

The result:



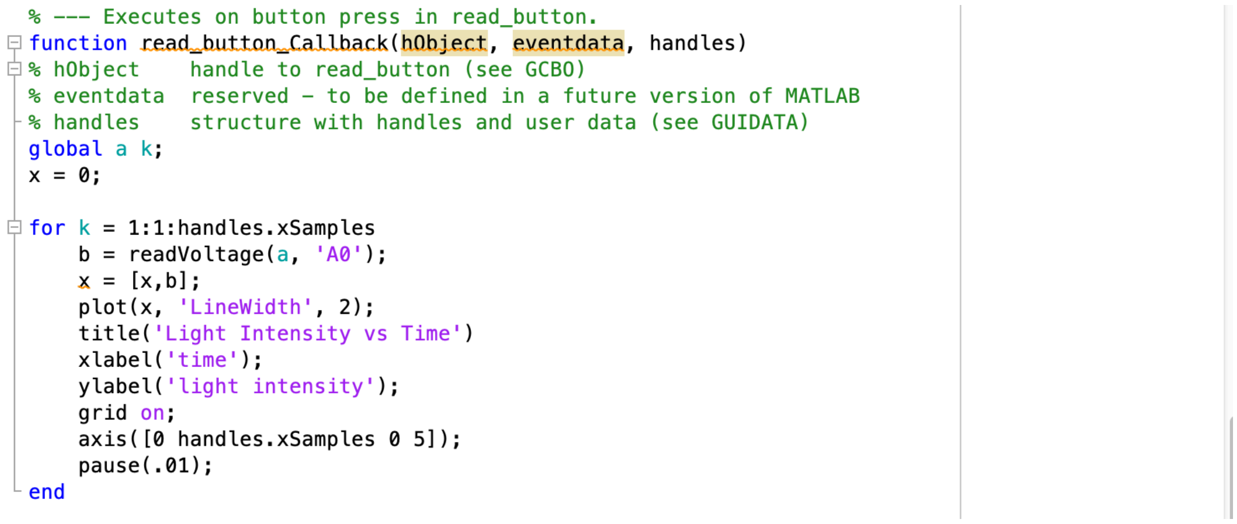
We observed when we ran the GUI that we still were not getting the results we wanted. We were not getting the voltage passing through the LED but the voltage from the potentiometer. We were able to get a nice graph due to fact that the read voltage function only works when reading from an analog pin. Consequently, we knew that the potentiometer was not getting the results we wanted and that reading a voltage through an LED is an impossible task. So we were forced to again reform our initial project and mold it into what it is now.

We knew we had to use the analog pin and that led us to grapple with different hardware tools we could potentially use. It was clear after researching that a photoresistor would be a perfect candidate for what we were trying to accomplish. Not only was it possible to use the read voltage function on matlab, but we could also get a riveting result that would work beautifully in real time and make for an applicable graph. We built our GUI with a graph and a enter box to fill in how long we wanted our graph to run. The code started off the same as before; we created a global variable a and then declared a as arduino. We first began by creating code to get the amount of samples the user wants to input. Basically, how long they want the graph to run.



```
function edit_text_samples_Callback(hObject, eventdata, handles)
% hObject    handle to edit_text_samples (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_text_samples as text
%        str2double(get(hObject,'String')) returns contents of edit_text_samples as a double
handles.data1 = get(hObject, 'String');
handles.xSamples=str2double(handles.data1);
guidata(hObject, handles);
```

The code basically allows the user to input a number of samples into the text box. We then will use that number of samples into our main code.

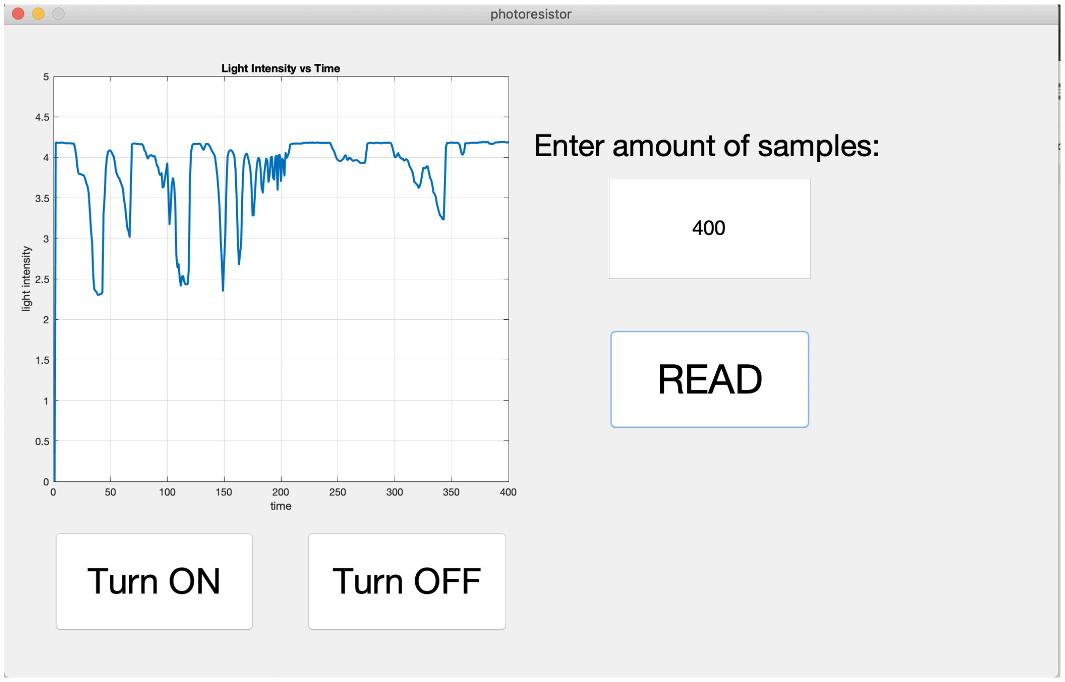


```
% --- Executes on button press in read_button.
function read_button_Callback(hObject, eventdata, handles)
% hObject    handle to read_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a k;
x = 0;

for k = 1:1:handles.xSamples
    b = readVoltage(a, 'A0');
    x = [x,b];
    plot(x, 'LineWidth', 2);
    title('Light Intensity vs Time')
    xlabel('time');
    ylabel('light intensity');
    grid on;
    axis([0 handles.xSamples 0 5]);
    pause(.01);
end
```

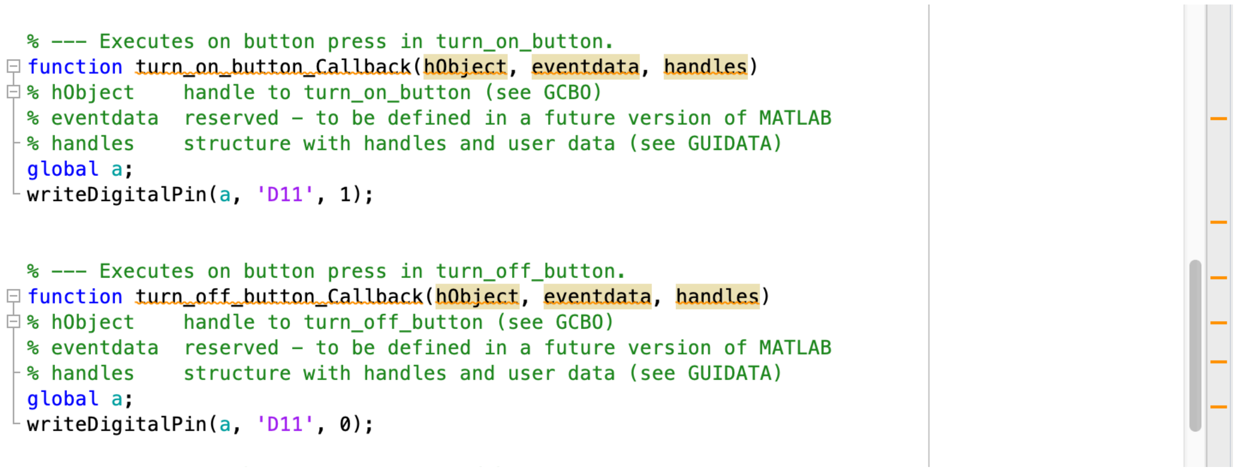
This time, instead of using an infinite while loop, we implemented a for loop which run from one to whatever the user entered as their number. This ultimately gives the user more control of the graph. We then read the voltage of the photoresistor and graphed it. The rest of the code is the same as before except for the axis in which the x parameter is set by the user in the number of samples that they want.

Implementing the code was as easy as ever due to the significant amount of time we had already spent on creating a real time functioning graph from our previous project attempts. When we ran our code, we experienced some self gratifying pleasurable results; it was working exactly as we wanted and expected it to.

The result:



It was a relief to see it in action and quite enjoyable. It was working as a result from the readings of voltage coming through an analog pin. We then could read precise numbers and graph it effectively in our GUI. But we did not stop there. We added in an LED to see its effects play out on the photoresistor. Afterall, our initial project revolved around an LED and it only made sense to have it apart of our final project as well. To implement the LED we modified the GUI with two buttons: turn on and turn off. Those two respectively would turn the LED on and off. As we saved the GUI, the code in matlab added in two new functions. We then simply added to those function by simply declaring a (which is global and also the arduino) and then by using a matlab arduino function to write voltage to the correct pin. This can be seen here:



```
% --- Executes on button press in turn_on_button.
function turn_on_button_Callback(hObject, eventdata, handles)
% hObject    handle to turn_on_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;
writeDigitalPin(a, 'D11', 1);


% --- Executes on button press in turn_off_button.
function turn_off_button_Callback(hObject, eventdata, handles)
% hObject    handle to turn_off_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;
writeDigitalPin(a, 'D11', 0);
```

There are many applications for photoresistors within our technology driven society. A great example of photoresistors in our everyday lives are the ones used in streetlights. Photoresistors are used to control when the light should turn on and when the light should turn off. When the surrounding light falls onto the photoresistor, it causes the streetlight to turn off. And there is absence of light, the photoresistor causes the street light to turn on. A few more applications of photoresistors may include various devices such as alarm systems, night-lights, camera light meters, clock radios, audio compressors, etc. They are additionally utilized in dynamic compressors together with a little radiant or neon light, or light-discharging diode to control gain reduction. A typical utilization of this application can be found in numerous guitar amps that join a locally available tremolo effect, as the wavering light fluctuations control the dimension of signal feeding through the amplifier circuit. The photoresistor is mostly utilized as a light sensor. They are often used when it is required to detect the presence and absence of light or measure the intensity of the light - the application in which we test and use for our very own project.

In summary, we first started this project under the impression that we can graph the voltage through an LED. This of course was a fallacious thought as the LED uses a digital pin which in turn, can only read either high or low and we saw that in effect in the video. We moved to a potentiometer which gave us a nice graph due to the use of the analog pin and the illusion that we got our project to work. That was because we were only reading the voltage through the potentiometer and not through the LED. Lastly, we discovered the photoresistor and how we could use it to display the light intensity using a Voltage vs Time graph.

In reality the initial idea we had was possible but only when using the proper devices. Since we already had all the code from our previous attempts, implementing the photoresistor was as easy as one plus one. Overall, our failures led to a better understanding of what we were actually trying to accomplish and resulted in a pretty neat project.

Works Cited

"MATLAB." *MathWorks*, www.mathworks.com/products/matlab.html.

"Photoresistor." *Physics and Radio Electronics,* ttps://www.physics-and-radio-electronics.com/electronic-devices-and-circuits/passive-components/resistors/photoresistor.html