

1. A short introduction giving an overview of your project and what assumptions you are making about the user.

Originally, we planned to make a polaroid camera, where we would take an image and print it out using a thermal printer. Due to the printer breaking at the last minute, we had to pivot ideas to a photo booth!

Our new project uses our arduino to take a picture with the camera component, along with some potentiometers and a button to toggle settings like saturation, brightness, and filters. Once these configurations are set, the user can press the other button to take the picture, and that picture will be served to a web server being hosted on the computer.

We are assuming that the user understands what the buttons and potentiometers are used for in our product. Also that the user has sufficient dexterity to press the buttons and turn knobs. If we were given more time, we might make an outer shell and larger buttons and knobs that would have some labels for a clearer user experience.

2. \*Revised and fleshed out requirements from part 1 of the progress milestone report.

Our requirements feedback from the milestone report was only positive, so we aimed to maintain the structure of our requirements.

R1: If the 'take-picture' button is pressed and the system is in the waiting\_for\_input state, the arduino shall send data to the web server, the take photo LED shall be set HIGH, the camera shall take a picture and send it to the web server, and the photo shall be displayed.

R2: If the 'filter' button is pressed, then the filter setting shall be iterated through a set of different photo styles. This value shall be used when rendering the next photo taken.

R3: If the 'saturation' potentiometer is turned, then the saturation setting shall be altered depending on the value of the potentiometer. Clockwise turning will increase the value, and vice-versa. Additionally, an LED shall be updated to display the value of the potentiometer in terms of its brightness. This value shall be used when rendering the photo.

R4: If the 'brightness' potentiometer is turned, then the brightness setting shall be altered depending on the value of the potentiometer. Clockwise turning will increase the value for brightness and Saturation, and vice-versa. Additionally, an LED shall be updated to display the value of the potentiometer in terms of its brightness. This value shall be used when rendering the photo.

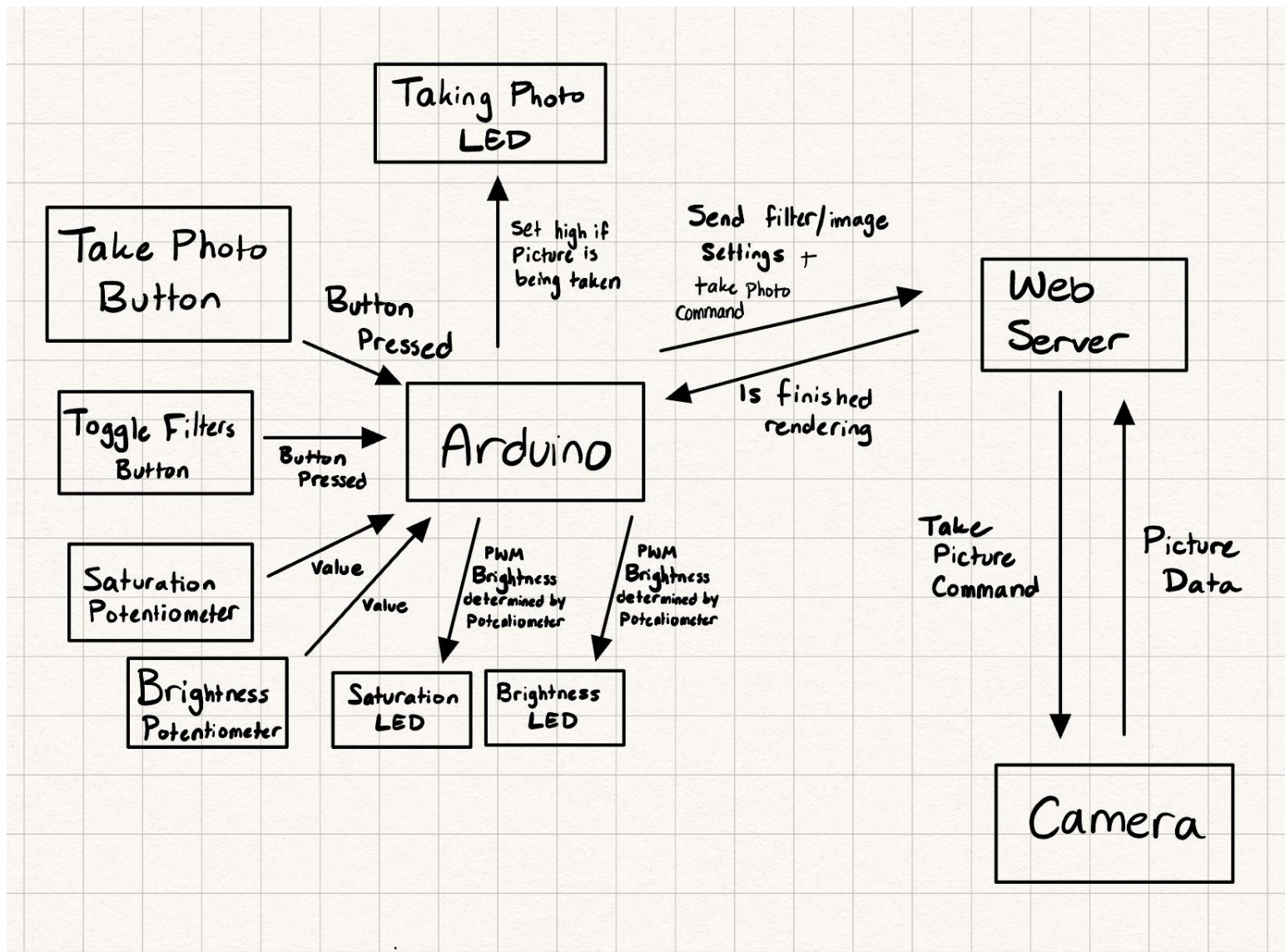
R5: Once the photo booth web server receives the photo, it shall render the image and display it. It shall then send a done rendering signal to the arduino, setting the take photo LED LOW.

R6: If 30 seconds pass since the last input to the system, an interrupt signal shall be sent to the web server to display a cat image.

### 3. \*Revised architecture diagram from part 2 of the milestone report.

Our revised architecture diagram differs greatly from the one in the milestone report because of our change in project specifications. We now have a web server which we send commands to, which then gets the photos from the camera. This is all done through WiFi. The camera data is sent directly to the web server rather than the Arduino because our version of Arduino is unable to save images without the help of an SD card (which we did not have). Therefore, we found it easier to use http requests to get the images from the camera to the web server directly.

We didn't receive much feedback on our architecture diagram (only positive feedback), but we were sure to include each component used in our new architecture.



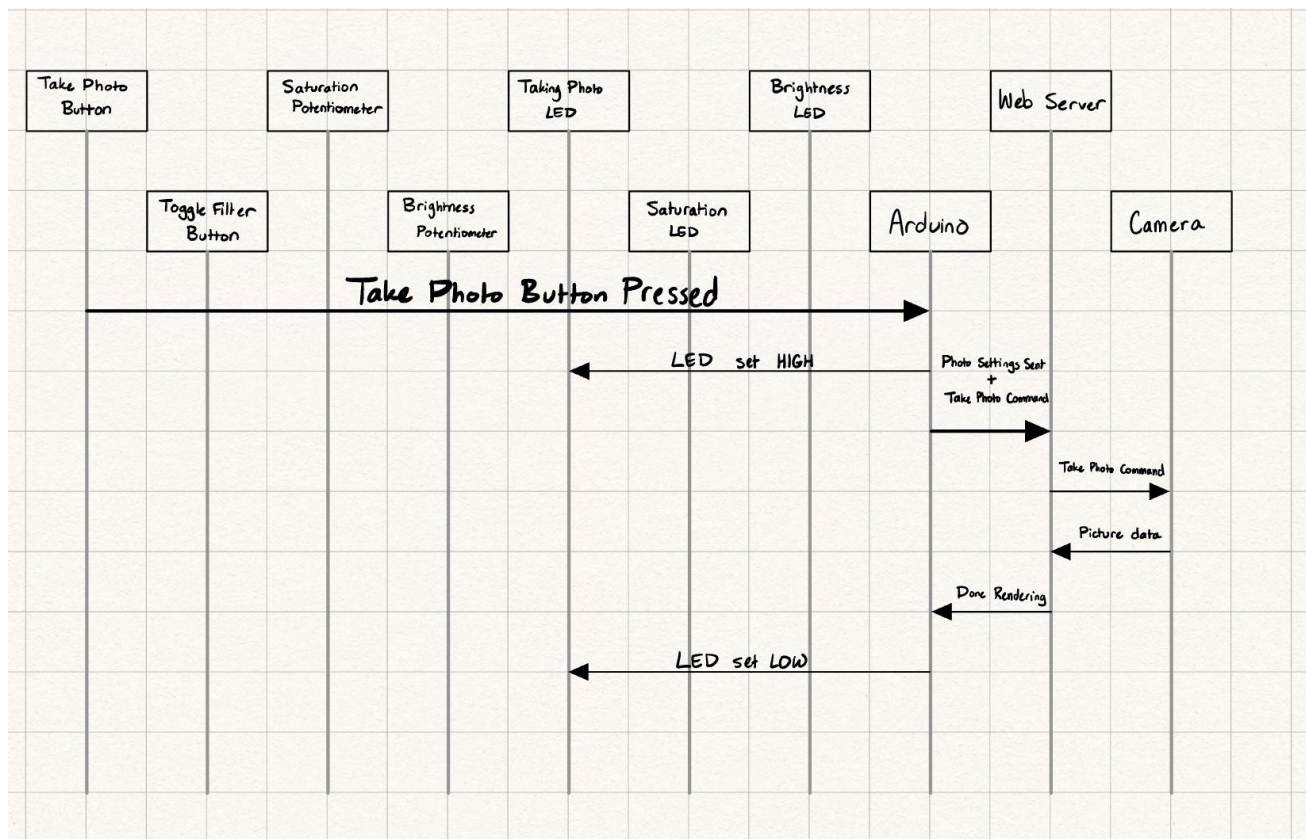
4. Three sequence diagrams for reasonable use case scenarios of your system (if there are more than 3, you should pick the 3 most likely use case scenarios to diagram, and include a note as to what scenarios you left out). Before \*each diagram, you should write a short (1-2 sentence) description of what the use case scenario is.

The only feedback we received on our sequence diagram from the milestone was that we forgot to add a description of the scenario. As seen below in the three scenarios, there are descriptions of each in addition to the diagrams themselves.

There are, of course, other scenarios not mentioned. For example: the scenario where the web server errors and the WDT resets the system; the scenario where the user does not take a picture in 30 seconds and the interrupt is triggered, sending a cat image to the web server; the scenario where the user takes a picture and updates the picture settings afterwards; etc.

Scenario 1:

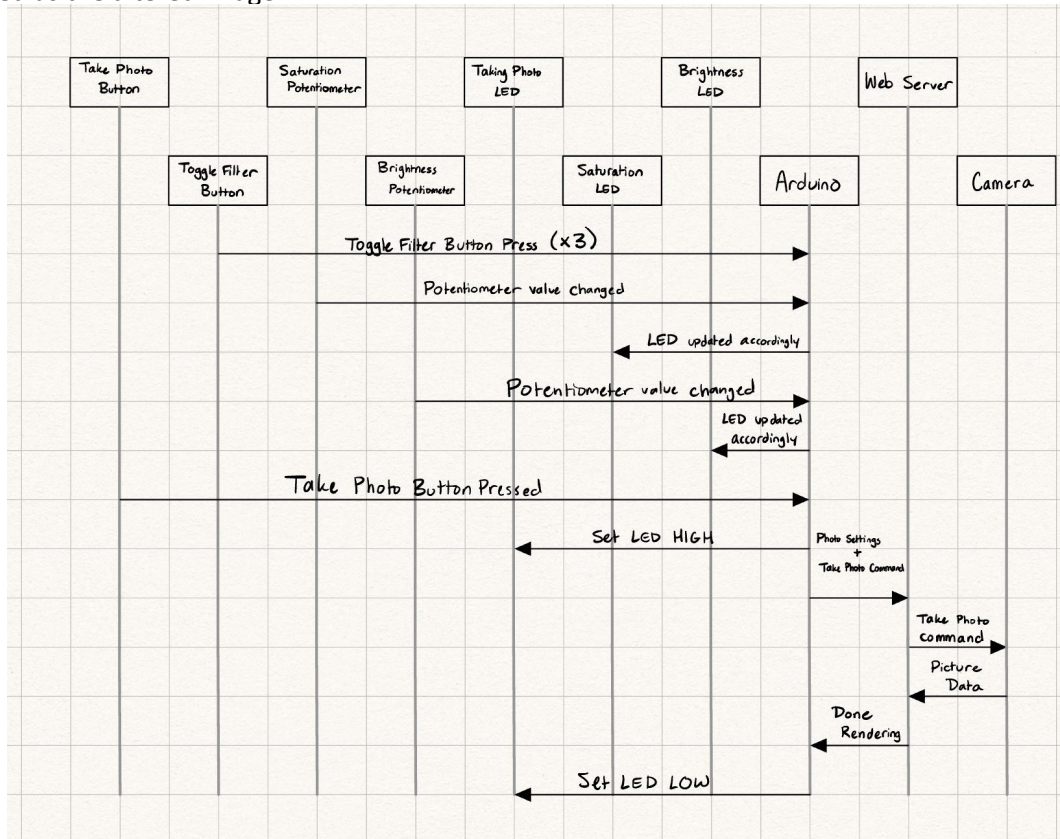
Beginner user. User hits the button to take a picture without changing any settings. Then the picture is sent to the arduino, and then to the web server to be displayed.





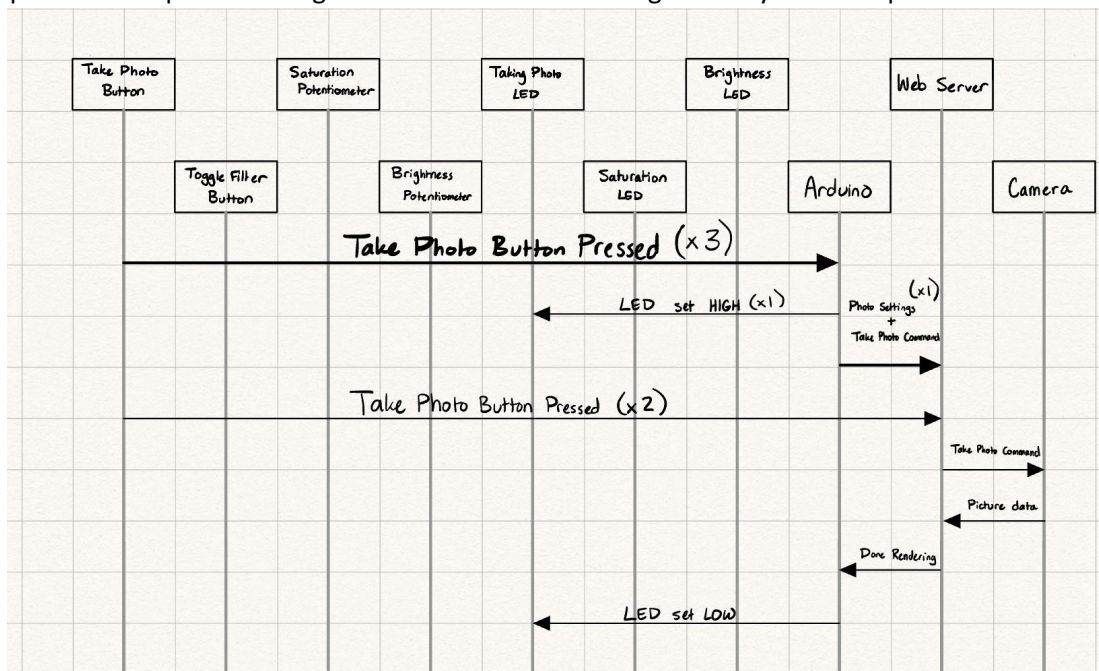
### Scenario 2:

Advanced user. User hits the filter button 3 times to toggle to the filter they prefer. Then the user changes the saturation and brightness potentiometers to alter certain settings before taking the photo. Then the user hits the button to take the picture, which is then sent to the arduino and web server to be displayed as the altered image.



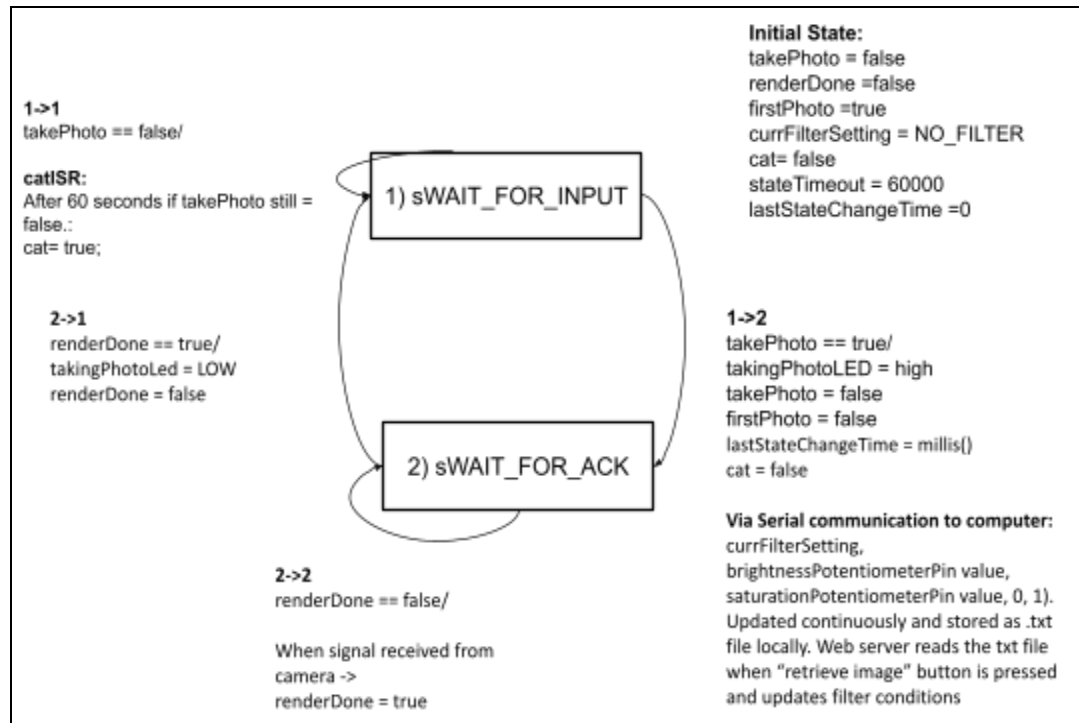
### Scenario 3:

Impatient user. User hits the button to take a picture, but keeps pressing the button again and again. The picture from the first button press is sent to the arduino and then web server to be displayed – all subsequent button presses are ignored until the arduino is again ready to take a picture.



5. \*Revised finite and/or extended state machine from part 4 of the milestone report. See part 13 below for directions on including your peer review feedback as an appendix.

Test	Transition/Function	Input	Output
1	1-1	takePhoto = False	nextState = sWAIT_FOR_INPUT
2	1-2	takePhoto = True	takingPhotoLed = HIGH takePhoto = false firstPhoto = false lastStateChangeTime = millis() nextState = sWAIT_FOR_ACK cat = false
3	2-1	renderDone= True;	takingPhotoLed = LOW renderDone = false nextState = sWAIT_FOR_INPUT
4	2-2	renderDone = false;	nextState = sWAIT_FOR_ACK When arduino receives signal that picture is taken. Set -> renderDone = true



#### Inputs:

**TakePictureButtonPress** - This determines if the button to take a photo has been pressed.

**ToggleFilterButtonPress** - This toggles the filter setting being used by the arduino.

**SaturationPotentiometerInput** - This value is determined by the potentiometer and will be used to determine saturation in the photo rendering.

**BrightnessPotentiometerInput** - This value is determined by the potentiometer and will be used to determine brightness in the photo rendering.

#### Outputs:

**updateLEDs()** - this reads the voltage from both potentiometers then writes it to the corresponding LED for both saturation and brightness

**photoLedOn()** - this sets the takePhotoLed to high turning it on

**photoLedOff()** - this sets the takePhotoLed to low turning it off

**catISR()** - this triggers the interrupt service routine which involves sending the currFilterSetting, brightness value, saturation value, binary that states a picture should not be taken, binary that states a cat should be shown, via serial communication to listener which stores info at .txt file locally on computer. It also sets the cat bool to True to indicate that a cat is being shown.

\*\*\*

We were given the comment: "the boolean variables would do better being expanded into states in your FSM - the whole point of an FSM is to describe statefulness of the system, such as when a photo should be taken" in our initial feedback.

We are open to this change in the future, but believe it might be too late to completely rework the FSM to incorporate multiple more states. Because the system utilizes multiple more components like the web server and camera server, which also control parts of the system, it would be difficult to add more states solely for sTAKING\_PICTURE and others like this. Thus, we figured a WAITING\_FOR\_ACK state would be sufficient, since in the meantime of signaling to the web server to take the photo from the camera, the arduino is in a waiting state. This might make more sense when looking directly at the code.

\*\*\*

Variable:

**takePhoto** - this determines whether the photo should be taken

**bool renderDone** - this determines whether the webserver has received the image from the camera

**bool firstPhoto** - this determines whether the current photo being taken is the first photo

**saturationLedBrightness**- this determines the voltage that should be applied to the brightness LED

**brightnessLedBrightness** - this determines the voltage that should be applied to the saturation LED

**bool cat** - this determines the type of image to display on the web server

**filter currFilterSetting** - this determines the filter setting of the image and triggers presets on the webapp

#### 6. \*Revised traceability matrix from part 5 of the milestone report.

Unfortunately, we misunderstood the requirements for the traceability matrix in the milestone report. The feedback we received told us that we had not implemented it correctly. We have corrected this with our final traceability matrix below.

	R1	R2	R3	R4	R5	R6
<b>State</b>						
<b>1</b>	X	X	X	X		X
<b>2</b>	X				X	
<b>Transition</b>						
<b>1-1</b>		X	X	X		X
<b>1-2</b>	X					
<b>2-1</b>					X	
<b>2-2</b>	X					

7. \*An overview of your testing approach, including which modules you unit tested and which integration and system (software and user-facing/acceptance) tests you ran. You should explain how you determined how much testing was enough (i.e. what coverage or other criteria you used, how you measured that you achieved that criteria, and why you think it is sufficient for you to have met that criteria).

In our feedback from the milestone, we were told that our testing plan was detailed and sufficient, but we were not specific about how we evaluated how much testing was enough. Therefore, we evolved our testing plan to take the photo booth into account, and we attempted to explain our coverage justification in a bit more detail.

We also included some unit tests for helper functions we created outside of the FSM.

We utilized black box testing to drive our integration testing, whereas our unit was driven more by white box testing. Our unit testing uses our FSM artifact to guide the testing while testing each transition, but also tests for non-transitions and helper functions as well.

Test	Transition/Function	Input	Output
1	1-1	takePhoto = False	nextState = sWAIT_FOR_INPUT
2	1-2	takePhoto = True	takePhoto = false lastStateChangeTime = millis() nextState = sWAIT_FOR_ACK
3	2-1	renderDone= True;	renderDone = false nextState = sWAIT_FOR_INPUT
4	2-2	renderDone = false;	nextState = sWAIT_FOR_ACK renderDone = true (only sometimes)
5	ToggleFilter	currFilterSetting = GRAYSCALE_FILTER	currFilterSetting = SEPIA_FILTER
6	catISR	nil	cat = true



#### Integration Test:

1. Button press and TakingPhotoLED: Press the takePhoto button and the LED should turn on
2. Button press and web server render: Press camera button and after red LED turns off, the PictureDone should be set to true and the photo should appear in the web server
3. Potentiometer 1 turned to the right should lead to a darker picture and vice versa.
4. Potentiometer 2 turned to the right should lead to more saturated pictures and vice versa.
5. Both potentiometers turned to the right should lead to more saturated and darker pictures
6. Test each permutation
7. Pressing button 1 (1,2,3,4,5 times) followed by button 2 to take a picture will result in a picture showing up on the computer screen with the filter(Grayscale, Sepia, Blur, Invert, No Filter) respectively
8. After a picture is taken, and button 2 is pressed again the picture taken should have the same filter

#### System:

##### User facing:

1. Test waiting for >60 seconds before pressing the filter button -> Cat screensaver should be printed
2. Test holding down button for longer than 10 seconds: Camera should take >1 picture since the cycle is reset after a picture is taken and the takePhoto will be true again.
3. Test clicking the button 2 times in quick succession: camera should take only one picture since the moment the ButtonPress changes it should start the chain of transitions and there is no store of the number of times the button is pressed.

##### Acceptance test:

1. Test if picture and filter button are accessible
2. Test if button can be pressed with minimal effort to trigger camera shot
3. Test if the potentiometers can be turned without unplugging other elements of circuit

We can check if testing is enough if we define the boundary circumstance for reasonable camera function. It should at the most basic level be able to

1. take multiple pictures without the need for an external reset.
2. Picture is visible to users when "retrieve image" button on webserver
3. Pressing the filter button on the arduino will elicit a change in the image on the webserver without having to "retrieve image again"
4. Changing the brightness of the image and saturation of the image will affect the appearance of the image in the intended manner when twisted at the same time, one after the other and when the sequence is reversed- all without having to "retrieve image"

Testing is complete when we can verify that it fulfills the functionality required of it in the reasonable constraints defined above. We also want to confirm that there are no abnormalities in the activity and check liveness/safety requirements to ensure that the product is safe and will not malfunction. We saw the impact of poor testing in the slides we made earlier in the semester that investigated different products' failures and their impact on everyday life.

8. At least 2 safety and 3 liveness requirements, written in propositional or linear temporal logic, for your FSM. Include a description in prose of what the requirement represents. Each requirement should be a single logic statement, made up of propositional and/or linear temporal logic operators, that references only the inputs, outputs, variables, and states defined in part 5 of this report. We will discuss safety and liveness requirements on the week of November 27th. You can also refer to chapter 13 of Lee/Seshia or chapters 3 and 9 of Alur's textbook (Brown login required).

Safety requirements:

1. System shall not take photo if it is already in process of rendering previous photo
  - $G(\text{CURR\_STATE} \neq \text{sWAIT\_FOR\_ACK} \rightarrow \neg \text{takePhoto})$
2. Rendering process shall complete before attempting to take another photo
  - $G(\text{sWAIT\_FOR\_ACK} \rightarrow (\neg \text{takePhoto} \cup \text{renderDone}))$

Liveness requirement:

1. A Photo shall be taken within a certain time after the take photo button is pressed.
    - $G(\text{takePhoto} \rightarrow F(\text{CURR\_STATE} = \text{sWAIT\_FOR\_ACK}))$
  2. Rendering process will eventually finish after a photo is taken
    - $G(\text{takePhoto} \rightarrow F(\text{renderDone}))$
  3. Acknowledgement of rendering completion shall be eventually received after the photo is taken
    - $G(\text{takePhoto} \rightarrow F(\text{sWAIT\_FOR\_ACK} \text{ AND } F(\text{renderDone})))$
- 
9. A discussion of what environment process(es) you would have to model so that you could compose your FSM with the models of the environment to create a closed system for analysis. Composing state machines was covered on November 15/17. For each environmental process, you should explain and justify whether it makes sense to model it as either a discrete or hybrid system and either a deterministic or non-deterministic system (for example, a button push could be modeled as a discrete, non-deterministic signal because it is an event that could happen at an arbitrary time, but a component's position could be modeled as a hybrid, deterministic signal based on some acceleration command). You are not being asked to actually do this modeling or any sort of reachability analysis, just to reason about this sort of modeling at a high level.

Environmental Processes:

1. Button pushes (present in takePhoto button and toggleFilter button)
  - Discrete, Non-deterministic: It is a predefined set of outputs ( true/false, grayscale/mirror...) and can be triggered at anytime
2. Potentiometer Outputs ( Present in our brightness and saturation)
  - Hybrid, Deterministic : It is a continuous range of values when the potentiometer is turned left and right and the values are constantly being outputted.
3. Time based ISR ( catISR)

- Discrete, Deterministic: It is a predefined service routine with a predetermined output of a cat jpeg being launched. We also know exactly when it will be triggered (every 60s if a photo has not been taken) hence it is deterministic.
- 4. WatchDog timer:
  - Discrete, Deterministic : It has a predefined action of resetting the program. We know if it will be triggered if the loop hangs for more than 8 seconds hence it is deterministic.
    - We model time with the millis() call and keeping track of the last call to the updateFSM() function
- 5. Camera Acknowledgement:
  - Discrete, Non-deterministic: there are only 2 modes of renderDone and we don't know exactly how long the render will take to complete and when it will update the value.
    - We do not need to model the image because the image is taken directly from the camera in the jpg format. Then this image is directly sent to the web server and is displayed on the screen. Because this format is already determined by the camera itself, there is no need to model the image itself.

10. A description of the files you turned in for the code deliverable. Each file should have a high-level (1-3 sentence summary) description. For each of the assignment requirements (PWM, interrupts, serial, etc), you should indicate which file(s) and line(s) of code fulfill that requirement.

#### **Arduino Code**

arduino\_code.h - Arduino header file which contains enums and declares functions.

arduino\_code.ino - Contains the camera statemachine code

- WDT: setup() and loop()
- PWM: updateLeds()
- Serial, sends text to web app: updateFSM(), catISR()

tests.ino - This contains unit tests for the arduino code.

#### **Camera Code**

CameraCode.ino - This is the code run on the camera that hosts a server and takes a photo when receiving a request.

- Wifi, sets up its own wifi network for the web app to connect to: setup() function

**\*\*other files are boilerplate from libraries we used \*\***

#### **Web Server Code**

App.tsx - The primary center of the web app, this file continuously polls the dump of state data provided by serial\_listen.py and uses this information to modify and augment the provided image, as well as display other conditions (such as the cat screen ISR). The provided image in question is also provided to the web app via serial\_listen.py, wherein App.tsx continuously checks for the image in the build folder, rendering it in the center of the screen.

- Wifi, connects to Camera wifi

Serial\_listen.py - This code is run on the computer the arduino is plugged into. It receives serial communication from the arduino, and saves the information in files so that it can be accessed by the web app.

- Serial, communicates with Arduino listening for text and also sending acknowledgements back
- \*\* other files are boilerplate from other libraries we used \*\*

11. A procedure on how to run your unit tests (for example, if you have mock functions that are used by setting a macro, similar to lab 6, make sure to note that).

We have a test file called tests.ino which contains all of our unit tests. These tests are run by the function runAllTests(), which is currently in the loop function in arduino\_code.ino, but commented out. To run the tests, please comment out all code in loop() and uncomment runAllTests().

12. A reflection on whether your goals were met and what challenges you encountered to meet these goals. You should only include challenges met or newly addressed since the milestone.


Overall, we are happy with the final product and believe we have met our goals. We experienced quite a few challenges along the way, including ordering a camera with a library incompatible with our arduino version, ordering a printer that relied on bluetooth and was not compatible with our arduino, and shorting the working printer we had for our earlier polaroid idea.


These challenges forced us to pivot our idea at the last minute to a photo booth, rather than a polaroid, but we believe that we learned lots while through these challenges, and did well to make a completely new product with very little time.

Our new project included the working parts of the camera that we were able to salvage from the polaroid part of the project, and some new ideas of how we could repurpose it, finally settling on a web server to host the images rather than printing them out. This pivot was not easy to do with such a time crunch, but we surpassed our initial goals with the photo booth idea, and were even able to implement filters and features into the photo booth that would not have been possible with the printer.

13. As an appendix, include the review spreadsheets you received after the milestone demo. Each defect should be marked as “fixed” or “will not fix” with a justification.

**IMPORTANT NOTE:** We do not have edit access to any of the spreadsheets linked below, but we have gone through each sheet and have fixed each comment that was left.

 SciLi Group Peer Reviews - Polaroid

 Farran & Kevin FSM Peer Review, Polaroid

 Polaroid FSM Peer Review