**Image Classification:** The image classification portion of the project was the most concerning part of the system that we had in mind. With none of the group members having experience in machine learning, deep learning, or image classification, it was a major setback to the team. In order to combat this, the computer engineer (Jimmy Vue) and electrical engineer (Tyler Martin) teamed up together to take on the task of successfully being able to detect and identify the lego pieces.

To start, we both had to do lots of research and brainstorming to decide our plan of attack. We researched and looked at many other lego sorting projects and/or projects that were doing similar tasks that we were also trying to achieve. While doing research, we noticed there was a common trend in these projects. Those trends were the use of Python, Tensorflow, OpenCV, Raspberry Pi, and a Raspberry Pi camera. Tensorflow and OpenCV were the main choices of libraries and functions for detecting and classifying images, with Tensorflow being the more common and the one that was deemed the easiest to use and learn. Our first idea was through the use of Python, we would incorporate Tensorflow and OpenCV libraries and functions in order to achieve what we needed. Tensorflow would be used to identify the lego pieces, while OpenCV was used to detect the lego pieces in the light chamber. After coming up with this plan, the next step was to then begin doing research on how to program in Python and how to incorporate Tensorflow and OpenCV in it. Upon doing research for Tensorflow, we learned about a lot of other concepts and materials we would also need to make. One of those concepts are models. We learned we had to create a model and teach that model with all the necessary information we wanted to give it. That model would hold all the images of the pieces and the various orientations of those pieces. The more times we trained that model, the more accurate results we would obtain.

During one of the weekly group meetings, while Tyler was doing some preliminary research into motor control, he came across an article talking about the recently published Google’s Teachable Machine. This program was made with Python using Tensorflow libraries, which is exactly what we had planned to do also. This program alone would help ease our workload substantially. Using Teachable Machine, we are able to make our own model by simply inserting a class and adding as many pictures as we want into that class. Teachable Machine is able to train models with an unlimited number of classes, which is a very important aspect for us since we are going to have near 80 classes due the number of unique pieces we have in the kit. After making the number of classes desired, you can begin teaching the model. You are able to adjust parameters such as the number of epochs, batch size, and learning rate.

The team came together one day and decided to conduct some tests using the Teachable Machine. We were concerned with how well the program would be able to accurately detect the axle pieces and the beam pieces due to them being of similar figure, lengths and color. We tested a lot and eventually came to the conclusion that the Teachable Machine could indeed tell each piece apart, but only if we had very good lighting, good orientation pictures, and good camera angle. It is good to note here that the team has not made the full model as of yet. The plan for making the full model was to incorporate a turntable and set up proper background and lighting for the turntable. The team would then place each piece on the turntable and have it rotate while a camera sits at an aerial view to capture the piece in various orientations until the resulting pictures are satisfied. Due to unfortunate circumstances brought on by COVID-19, the creation of the model has been put on hold and testing has been conducted with various household items and non-ideal pictures with the one kit we have been provided with.

In addition to allowing us to be able to make a model, Teachable Machine provides a code in order to help us test the model also. We then made a test model and downloaded the test code and immediately began testing. It took quite a bit of research on different Tensorflow functions to get the code working as we wanted it to. In the end, we were able to successfully load the model into the code, feed it a picture by setting the correct path to the picture, having the program give us its prediction and confidence, and correctly labeling each class with its respective prediction. Being satisfied with the results we obtained from the test code, we decided it was time to transfer everything over to the Raspberry Pi and begin testing/implementing everything on here.

The first testing of the Raspberry Pi came from using a Raspberry Pi 3 that was given by our Project Lead. We tried testing on this version of the Pi and achieved no results. We were able to download Tensorflow, but we were getting a bunch of errors that we were not able to resolve. Upon ordering and finally receiving the Raspberry Pi 4, everything suddenly went smoothly. We were able to install Tensorflow 2 and run our test code as well as we did on our laptops. During this time, we discovered a demonstration video/article of the Object Detection API by Edje Electronics. This program allowed us to capture objects and classify them using a video feed from the Raspberry Pi camera. This was exactly the idea we had in mind of doing our own project. Best of all, the code for the Object Detection API was open source and module. We were able to simply utilize the code and insert our own model and fix it to our liking. We deemed that as our next step, to get the Object Detection code working on our Raspberry Pi. When trying to implement the Object Detection code into our Pi, we found out there were a few compatibility issues. This was a rather dated code, so the syntax used in the code was in Tensorflow 1 syntax (reminder that we are using Tensorflow 2 for our Pi as of that moment). We had two choices, either revert back to Tensorflow 1 or change the entire code to Tensorflow 2 syntax. Reverting back to Tensorflow 1 format was the more convenient and easier task to do since we ended up finding out that Tensorflow 1 allowed us to run both the Object Detection code as well as our test code with no issues. With the Object Detection code working, the results were rather mixed. With our own model loaded into the code and the video stream working, the camera was able to capture the objects, however the confidence level given by the program was not exactly what we had hoped for. To get the code to work, the video aspects of the Edje Electronics code were combined with the image operations and classification code we had been using to make predictions on single images. Essentially the code would pull an image from the video feed, normalize it and then make a prediction. In order to get this code to work, the camera resolution had to be reduced to 224x224, or the frame that was pulled from a higher resolution video needed to be resized to 224x224. We believe this to be the cause of the decreased accuracy and plan to focus our efforts on resolving this issue when we pick back up with the project in August. Another problem we encountered was the frames per second we were getting with the camera and detection. The average fps obtained was about 2fps. We recalled that while we were doing research on the Teachable Machine, one of the video demonstrations we viewed used a device called the Coral USB Accelerator. This device was also made by Google and its purpose is to speed up the process of image detection and classification. Looking at the various video demonstrations, the fps obtained from their results jumped from an average of 2fps all the way up to as high as 30fps. We concluded it may be necessary to implement the Coral USB Accelerator into our project too.

With the semester coming to a close, we have definitely done more than we set out to do at the beginning of the semester. This was all due to planning, researching, and getting an early start on setting everything up. Testing will still be happening throughout the summer and the team is hopeful that we will be able to create our full model with the Teachable Machine soon and finally be able to test on it.

At the start of semester 2, we revisited the test code for video predictions in an effort to improve the accuracy of the system. The code was essentially built from scratch again, changing a few key elements. There was a significant improvement to accuracy, but the fps was not improved. Our focus will be on the completion and integration of systems primarily, and if time allows, we will make efforts to improve the fps.

**Control Flow:** After deciding on a method to classify images, the next design problem to be solved was how to use the information provided by the image classification code to sort the pieces it identifies. In the initial image classification testing the computer engineer (Jimmy Vue) and electrical engineer (Tyler Martin) developed a method to assign labels to the numerical data returned by the image prediction. The image prediction outputs a prediction strength value for every class in the model each time a prediction is made. To turn this into usable data, the class with the highest probability is found using matrix operations. In a similar fashion, the index position of the highest probability is used to assign the proper class label from another matrix. This enables us to display exactly which part was identified if we desire and to use the classification labels in a series of case statements. Each case will have an assigned set of motor control assigned to it to ensure the part makes it to the correct container. The motor control code inside the case statements will be preset positions for the servo and stepper motors. Each case statement will also have a counter to track how many parts from each class pass through the scanner, telling us if we have missing or extra parts in a kit. For the overall flow of the system, IR sensors will be used to monitor the system at two points, the hopper output and at the scanning chamber.  The IR sensor at the hopper will monitor the parts flow and will be used to shut down the machine when all the parts are sorted. The IR sensor at the scanning area will be used to temporarily halt the main conveyor and the shakers to prevent multiple parts from falling onto the scanning conveyor.