This document details the work done to fulfill the requirements of a Spring 2020 Independent Study proposal filed to work with NoRILLA, a mixed reality platform which teaches students physics principles using interactive and constructive activities. In **bold** are learning objectives and deliverables, and below each item is the work that was done to fulfill its requirements.

Learn how to do Image and Video analysis by applying computer vision and machine learning algorithms in the Processing programming language

The Processing programming language has an established convention of `setup()` and `draw()` methods controlling a run loop. Calling functions in `setup()` defines the canvas and its attributes. `draw()` calls functions which modify the canvas. In the case of NoRILLA, the canvas is a depth image of the platform upon which the ramp sits. Much of the work for this project was done in the `ImageFilter` and `ImageRecognition` classes. Secondarily, `MainGameLogic` and `CompareGameLogic` were modified when debugging ramp segmentation code, which will be explained below.

Investigate issues in the NoRILLA ramp game blob detection code base (`ImageFilter`) (e.g. being sensitive to small angle changes in the camera)

At the beginning of the semester, the left-most part of the ramp was being tracked and aggravating issues related to the camera tracking the position of the ramp and objects placed on it. Eventually, we determined that tracking the left-most position of the ramp explicitly was unnecessary for segmentation. Removing this was the simplest and most correct way to deal with the issue.

Now, when the user moves onto a new scenario in the SmartRamps game, it is necessary to un-segment the ramps and re-segment them. Because the ramp can be moved or repositioned (intentionally or unintentionally), this is how we ensure that ramp segmentation happens correctly between scenarios.

Design an algorithm to properly and reliably segment the ramps independent of the position of the table when it shakes

Small parts of the lower ramp - just below and to the right of it, from the perspective of the camera - were further segmented to allow for more precise object tracking. This work was done in concert with another developer on the team.

In terms of pseudo-code, the way to capture and reliably segment the ramps was:

- Instantiate an empty array list to store the z-index of array pixels (`rampPixelsArray`).
- On initialization, if there are no objects on the ramp (detected by the quantity of pixels present on blobs 1 and 6, which pertain to the upper and lower ramps respectively), then the z-index of pixels for both ramps are stored in `rampPixelsArray`.

- Either by pressing the Play button or by pressing a key on the keyboard, the flag `canEraseRamps` is set to `true`.
- If `canEraseRamps` is `true`, then the pixel values that were stored in `rampPixelsArray` are segmented and erased.

Implement a blob detection algorithm to segment and consistently discern between the ramp and objects placed on it, potentially by adding more features (in addition to the MOI) to differentiate more reliably between the objects, independent of small angle changes in the camera

We calibrated the moment of inertia (MOI) values for objects on the ramp in multiple positions - left, center, and right. This process was repeated three times after we rewrote the platform segmentation code, the ramp segmentation code, and after the position of the RealSense camera was adjusted. These MOI values were combined with features such as item width and hollowness (in the instance of detecting a hollow disc vs. a disc with no hole in the middle).

We also rewrote and simplified the functionality to differentiate the various cars that can be placed on the ramp (normal car, metal weight on car, weight in front, weight in back). We did this by creating a line parallel to and just above the ramp and capturing the activated pixels along that line in the depth map. In combination with MOI, this successfully differentiated the cars from other objects placed on the ramp.

A particularly challenging issue arose when differentiating a non-hollow, medium-sized disc object from a similarly-sized gear with teeth. Our team worked with a CMU SCS student who wrote an algorithm to draw a convex hull in the depth map around the objects placed on the ramp and compare that to a hull that encircled the object itself. If the ratio of the hull to the convex hull is approximately 1, then there was negligible hollow space around the object, and it was likely to be a disc. If it was slightly less than 1, the object is likely to be a gear. This “solidity” value combined with MOI allowed us to successfully differentiate whether the user was placing a gear vs. disc on the ramp.

Get the code base to work reliably on RealSense D415 camera (transitioning from the Kinect camera) for NoRILLA EarthShake and SmartRamps games

Some bugs remain with the RealSense code base, but much work was done to remedy platform and ramp segmentation, and robust object detection. Future work will document changes made and resolve remaining bugs, ensuring that the NoRILLA EarthShake platform is as or more robust using a RealSense D415 camera than it was with the Microsoft Kinect.