UMTRI

Summer 2020

Improvement of Canvas, the User Interface and Data Structures of the Labeling Software

MDP-DEVIATE, Technology Subteam, Summer 2020: Ho Seok Song, Haomeng Zhang, Shengan Zhang, Emma Finkel
Researcher: Kathleen Klinich, Carol Flannagan, Jared Karlow UMTRI Group: CMISST

ABSTRACT

The previous UMTRI Video Labeling software had several problems with bounding box manipulation, inconvenient bounding box production, and lack of distinction among boxes. The UI and canvas team each set a goal in making a more user friendly interface for the bounding box production and for the information display to allow easier analysis and labeling using the software. We aimed to provide an easier method for users to create bounding boxes that represent the objects more accurately, and a more user friendly final view of all boxes on the canvas. We achieved these goals by changing the original click-and-drag method to a connect-the-dots method for the production, where the users specify points to include for the bounding box. For the view, unique colored shading and line stylings were added for more convenient distinction between them.

The UI team aims at providing a more accessible tree structure to the user, which presents the hierarchical relations for the current frame, instead of showing plain behavior list with information in all frames. We created the interfaces for different data structures based on the existing PYQT5 library and constructed the tree structure with add and delete functionality so that the total hierarchy would be very clear to the users.

INTRODUCTION

The first step was repairing several major bugs in the software such as the application crashing when the user attempted to remove an existing bounding box or the unique IDs for each objects all being redundantly assigned 0. The interface also had issues. Originally, for a person to create a bounding box for an object in a video, the user clicked “add bounding box” on the list of objects on the right side, then proceeded with a click and drag for a box which finalized when the user let go of the cursor. This bounding box production method doesn’t allow the user to pinpoint specific locations on the image where the user wants to define. Moreover, distinction between boxes could be problematic the boxes were completely transparent within the border and displayed their assigned color only when hovered by the cursor. We also wanted to ensure accessibility for individuals with color vision deficiency.

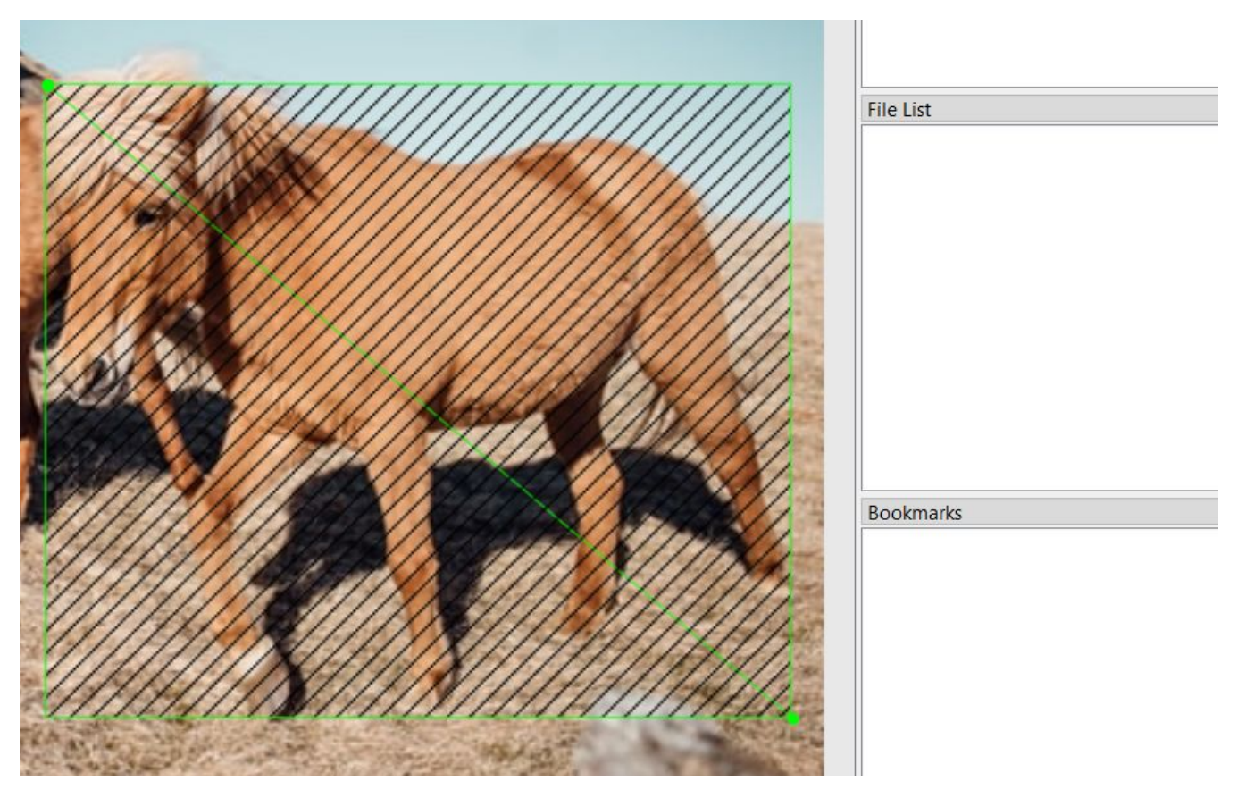


Figure 1. Previous bounding box production

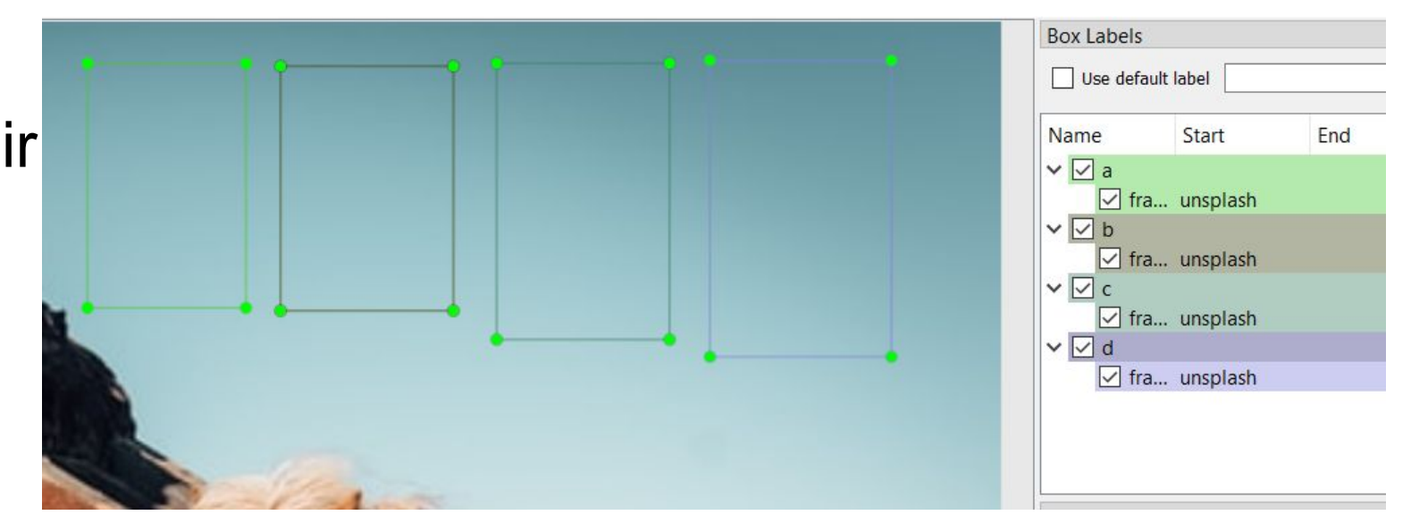
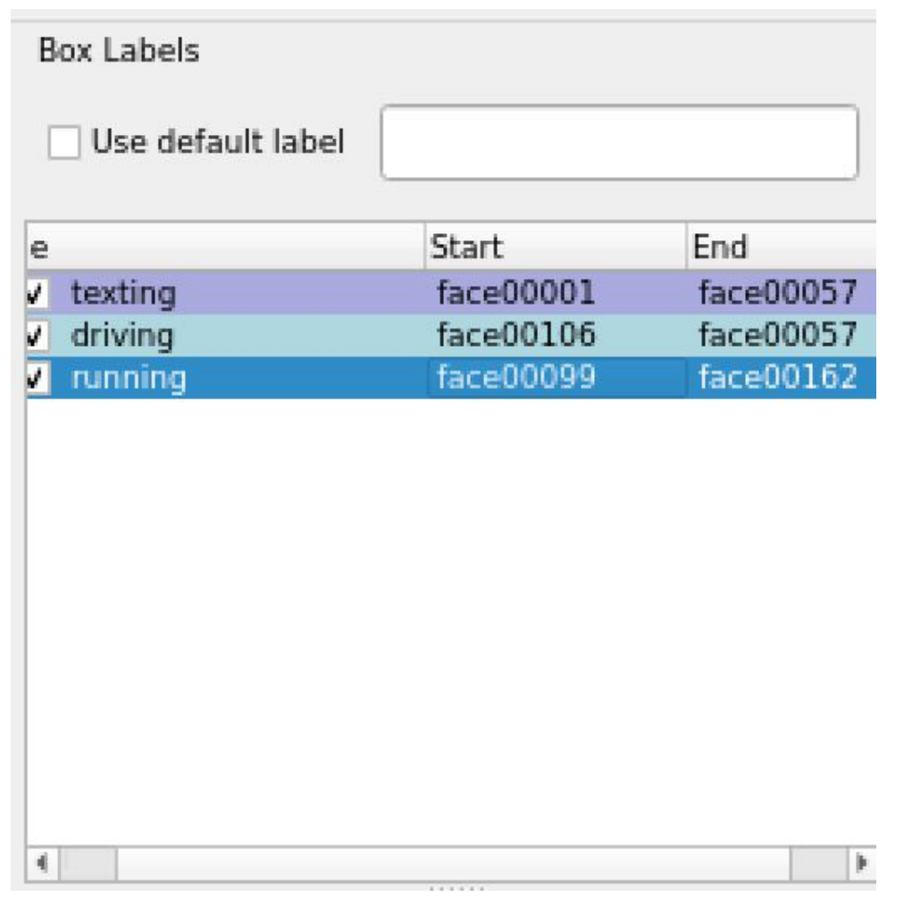


Figure 2. Bounding boxes that are hard to distinguish

The original UI displayed three subwindows on the right hand side of the software interface. The top subwindow showed the name, start frame and the end frame of all coded behaviors, but many behaviors were not relevant to the current frame. The list of all coded behaviors would bury the information relevant to the current frame, thus decreasing the user's coding efficiency. A plain list did not show any interrelations among different behaviors. As for the functionality, the user needed to use the left hand side toolbox to add new behaviors, and it was also inconvenient to edit the starting and ending frames. These tasks decreased the user's labeling speed.So we wanted to revise this subwindow to provide a more user friendly interface that presents the most useful and relevant information.



Box Labels

☐ Use default label

e	Start	End
✓ texting	face00001	face00057
✓ driving	face00106	face00057
✓ running	face00099	face00162

Figure 3. Previous labeling software interface

OBJECTIVES

Canvas

1. Revise the colors of the bounding boxes and the lines that border them to make the boxes distinguishable from each other, along with accommodating individuals with color vision deficiency.
2. Change the data structure to make the bounding box production standalone for the UI team to use.
3. Fix reported bugs.

UI

1. Create some basic widgets, inherited from the QWidget in Pyqt5 Library for video_object, bounding_box, state_item classes, which can show the user more information about these classes instead of a plain behavior list.
2. Create a tree structure interface to show the hierarchical relationships among different objects and behaviors occurring in the current frame.
3. Implement Add/Delete functions for objects in the tree structure to facilitate user operations.

METHODS

Canvas

The click-and-drag method for bounding box production was changed to a connect-the-dots method, where the users click along the border of the target object. Then in the backend, the minimum x and y values, as well as maximum x and y values, are calculated from those points to produce the 4 vertices that make up a single rectangle bounding box. Although the boxes are created using multiple, freely placed dots, the boxes always show up as a rectangle by getting the minimum x, y and maximum x, y values from the dots. This makes the boxes consistent across several frames and makes them easier to track.

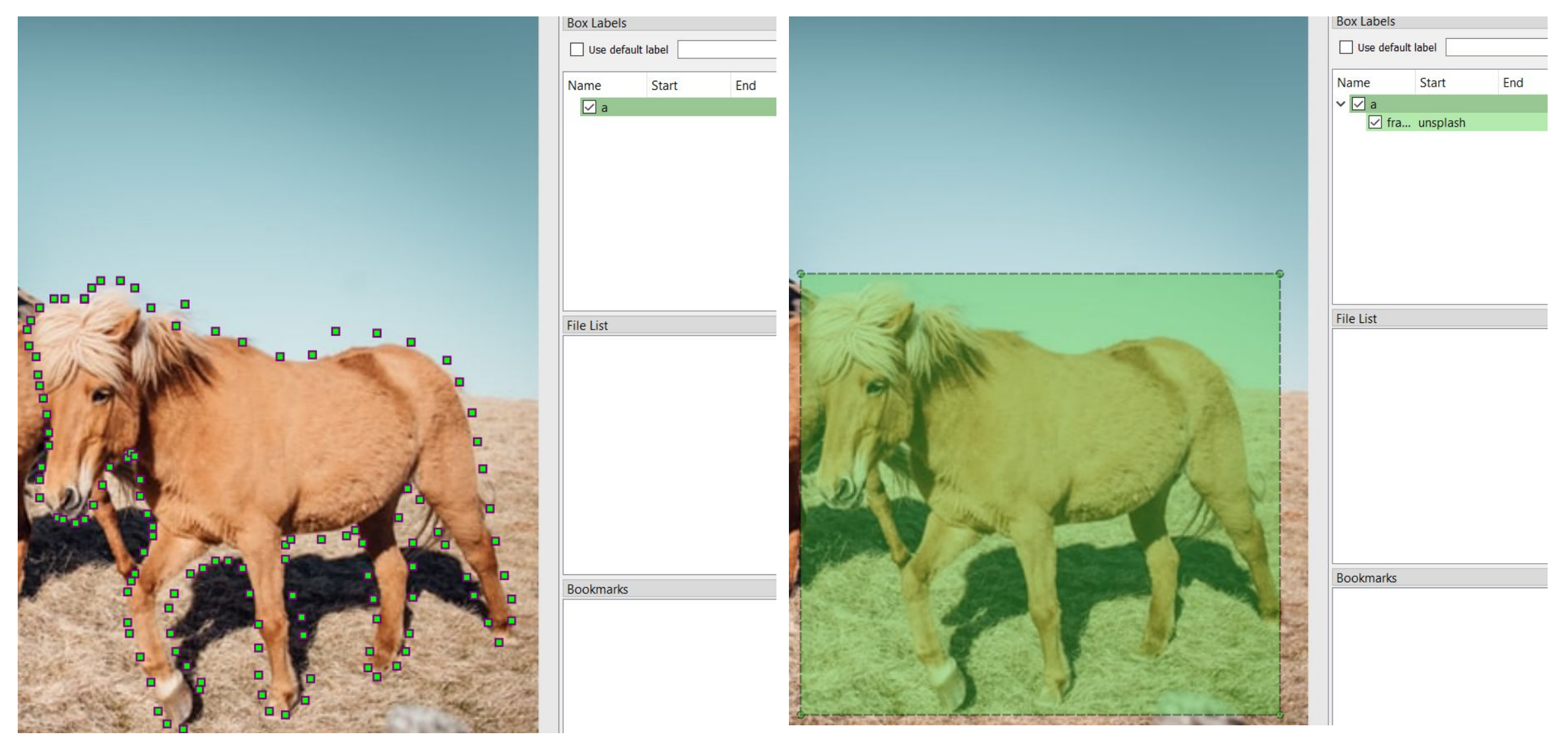


Figure 4. Improved connect-the-dots interface for bounding box production

The boxes also have default shadings that are unique to each, and line stylings were added so that the boxes are friendly for the individuals with color vision deficiency.

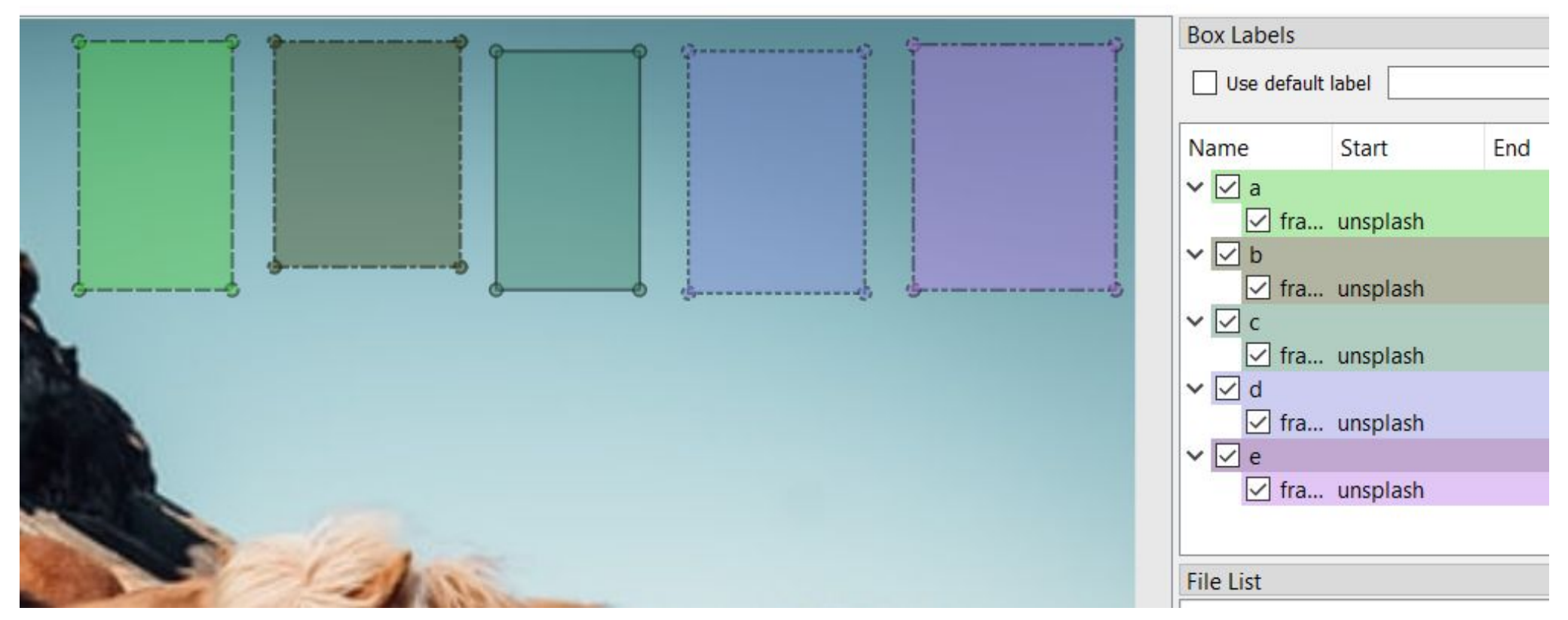
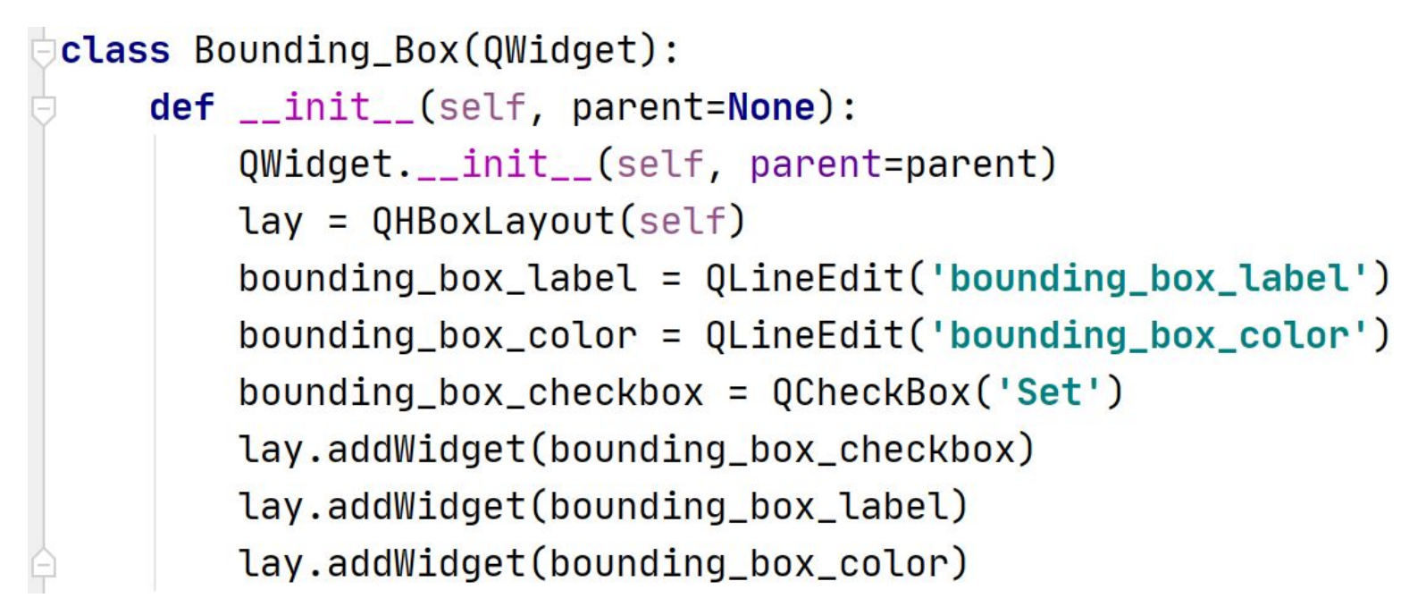


Figure 5. Colored shading and line stylings for the borders

METHODS

UI

Since the fundamental elements for the objects in the video are video, video_object, state_item, and bounding_box, we make classes of them inherited from QWidget in Pyqt5 library. Meanwhile, we add specific features required for each elements, such as combo box for state_item implemented by QComboBox(), check box for bounding_box by QCheckBox(), etc, as shown by the picture below



```
class Bounding_Box(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self, parent=parent)
        lay = QHBoxLayout(self)
        bounding_box_label = QLineEdit('bounding_box_label')
        bounding_box_color = QLineEdit('bounding_box_color')
        bounding_box_checkbox = QCheckBox('Set')
        lay.addWidget(bounding_box_checkbox)
        lay.addWidget(bounding_box_label)
        lay.addWidget(bounding_box_color)
```

Figure 6. Fundamental Class Implementation

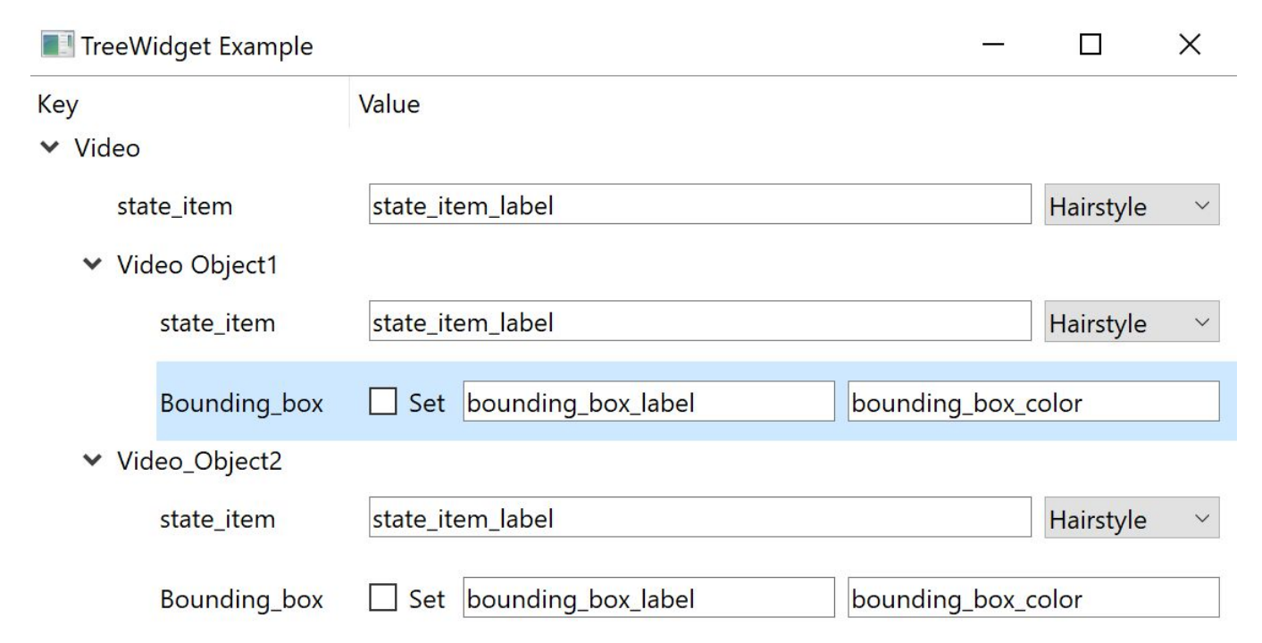


Figure 7. Tree Structure in User Interface

In our UI design, we want to use a tree structure to manifest a hierarchy relationship of objects in the video, for example, which hand objects belong to which person object. Therefore, we initialize our primary widget with QTreeWidget(), and other widgets can identify their parents when constructed as QTreeWidgetItem(tree) for root item, QTreeWidgetItem(root) for first-level items, QTreeWidgetItem(other_parents) etc. And the effect is shown in the figure above.
To make the UI operable by users, we implement primary Add/Delete buttons for the items. We used the connect (add()/delete()) function; once the user right clicks on one video_object and choose Add or Delete, our add() or delete() function will be triggered by this connect() function, and then the corresponding widgets can be added.

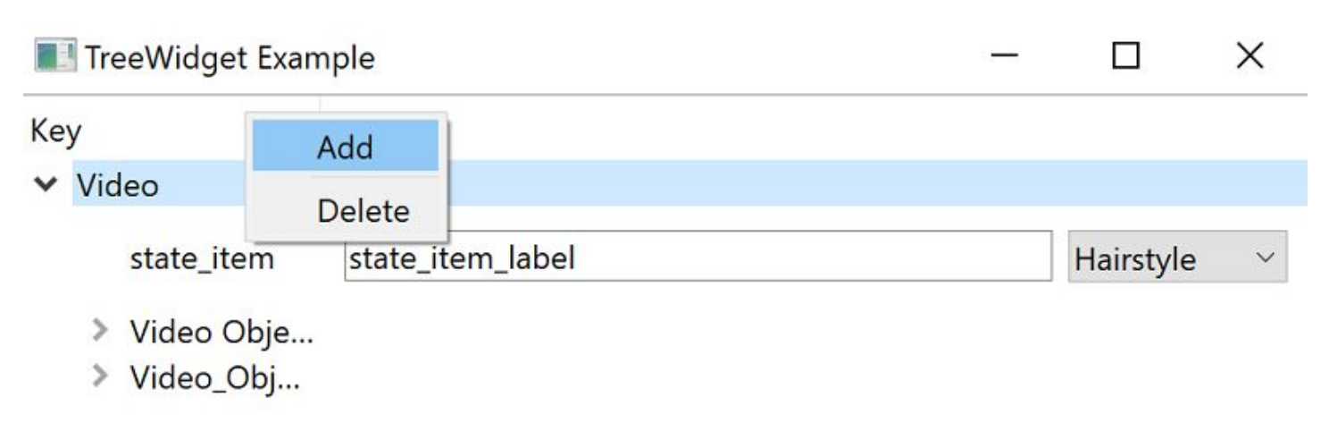


Figure 8. Add Functionality Example

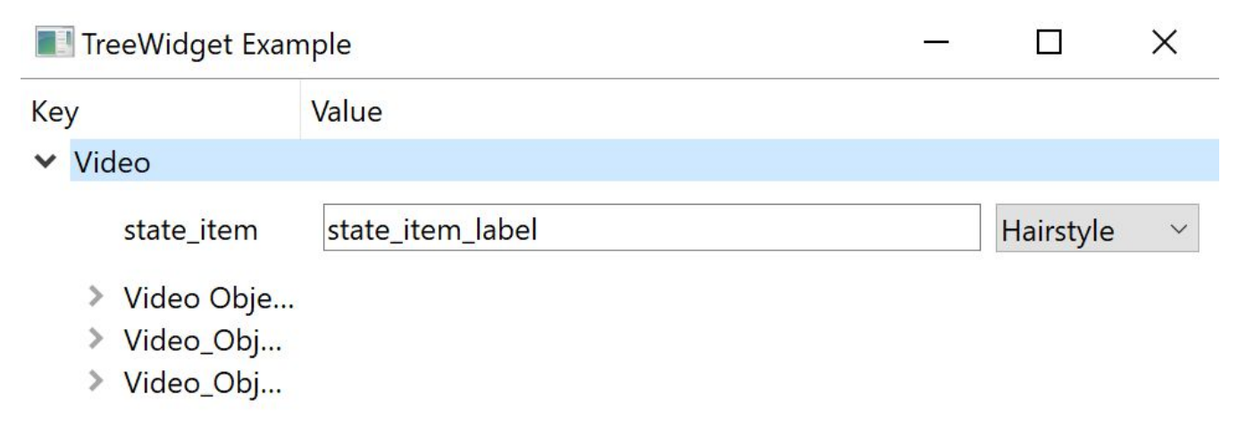


Figure 9. The tree structure after Add

CONCLUSIONS

The main objective of this research is to improve the overall interface of the UMTRI video labeling software. The creation of the bounding boxes is now more user-friendly by allowing users to zero in on specific locations in the image, along with ensuring each box has a rectangular shape for consistency across frames. We predict that these user friendly improvements will greatly increase the efficiencies of the software tools, while the improvement in accessibility will broaden the software’s target audience. Currently, we are still working on separating the bounding box class into its own Python file so that the bounding box production interface can be combined with the tree structure interface for the final result.

As regard to the UI implementation, we have completed our objectives and finished testing in a separate Python file. The updated interface contains more information about the current frame, instead of previous plain behavior list.The irrelevant information is hidden so that the interface archives a better clarity. The tree structure presents the hierarchical relationship among different objects and behaviors, which makes the information more understandable. The updated add and delete functionality is easier to operate, which potentially increases the user's coding efficiency. Our current work is merging the code into the previous software code and do some final testing.