
# ABC Drone: Tactical Sports Footage

Jared Cooper and Connor Bartal
CS 4624 (Multimedia, Hypertext, and Information Access)
Spring 2021; May 14, 2021
Instructor: Edward A. Fox
Virginia Tech, Computer Science
Blacksburg, VA 24061

# Table of Contents

## Table of Tables

## Table of Figures

## Executive Summary

The ABC Sports Drone capstone team is an extension of the ABC Drone Project which is a group spearheaded by client Charles Kerr and in conjunction with the VT Club Ultimate team, Burn. The goal of the project as a whole is to provide high-quality footage and video editing abilities of amatuer sports to the masses. This capstone team is a subsection of the ABC Drone Project that has been tasked with creating software solutions and developing new techniques to help push this drone project to fruition. This report will cover the progress of the capstone team in developing new routines for the drone, and the pivots that have been introduced as the team has received new data.

The first goal that was tackled was identifying players on a field from an endzone-to-endzone view. This started with the analyzing of contours, along with their position and attributes to determine if a contour was a player. Artifacts from off the field of play proved to be greatly troublesome, so a field bounding solution was created to eliminate as many artifacts as possible that were not on the field of play. Fairly good accuracy was achieved with this method (~75%), but the goal was set at 85%+ accuracy for identification. After experimenting with motion-detection and object persistence, the best course of action seemed to be identification via a convolutional neural network. No datasets were available that matched the application of this network, so an original dataset needed to be created. Therefore, an application was developed that allowed for fairly quick extraction of data from sample videos. This data was fed to the neural network and constantly yields around 94% identification accuracy. Although the accuracy is high, the addition of the neural network reduces frame rates to approximately 1 FPS. This is largely due to the inability to incorporate a GPU into the program's computational resources.

Some market interviews with actual coaches revealed a larger interest in post-processing capabilities over live-identification, so the client decided to pivot the focus of the project. A system that allows for speed-editing of footage has been developed along with a companion application that will allow coaches to easily track stats and pre-edit film via a GUI. The speed editing program takes in the footage and allows the coach to use a video game controller to create quick cuts to eliminate down time, as well as pan, tilt, and zoom on the footage to ensure the action is always framed. The edits are recorded in an Edit Decision List (EDL) file which is then used, in conjunction with the video file, to create a fully-edited game film. With this method, a 90 minute game can be edited in 5 minutes or less. The program also allows for easy keeping of statistics and production of highlight clips. Players will have access to a directory that contains their statistics and all of their highlights from the game footage.

## Introduction

Living in the 21st century, chances are you played or were directly involved with sports at some point in your life. In fact, roughly half of the 74 million kids in the United States participate in organized sports (Swanson, 2017). Whether it's a high school rivalry, your son's little league baseball game, or even a professional football game, there is no doubt that playing and/or watching sports is a vital part of our world's culture. Thanks to the rapid development of technology, two of the most integral parts of sports today are live broadcasting and recorded game footage. The advancements we have made in recent years have not only allowed us to bring the thrill of the game to millions of viewers across the globe, but also provide players and coaches with high quality tactical footage that can be used for training purposes. From studying your opponent, analyzing weaknesses in your own game, or even learning new tactics, the ability to replay and watch game footage has become a key component in the professional sports world.

The basic idea for our project is to utilize drones to obtain high quality game footage for sports at all levels. The inspiration came from a possible interaction with the JinxCam, a hardware solution that allows the recording and live streaming of various sports games. The JinxCam was developed by Charles Kerr, the sponsor for this project, and utilized a camera on top of a 30 foot pole to capture game footage. This footage was recorded as well as live streamed to viewers over the internet. Charles was looking to make an improvement to his original design. He noted that one of the biggest weaknesses of the JinxCam was its limited height. The 30 foot pole was the biggest pole that he could safely carry around and set up, however this height limited the use of recorded game footage as it did not allow for a great view of the action taking place on the field. This is where the idea for drones came in. Drones bring height and maneuverability advantages to this problem. Not only are they easier to transport, but they also allow for greater heights and better recording angles to be achieved. After much discussion, the team decided that we would focus on using a tethered drone system for our final design. The use of a tether will not only allow us to circumvent the drone's limited battery life, but also record footage at a much higher resolution as the footage will be recorded directly to a computer at the opposite end of the tether. This approach will allow us to record hours of high quality video without ever moving the drone.

The ABC Drone Project comprises 10+ people, with each person fulfilling an important role in making the project successful. The scope of this capstone team is solely software, and it will be assumed that any prerequisites for the running of software have been fulfilled - for example, the final drone that will be used for collection of

footage is nearing completion, but footage used in development was collected by a slightly different drone. Much of the software developed during this semester is proof-of-concept and will have to be slightly modified once the final hardware is in place.

**Pivot to Post-Processing**

Originally, the focus of the ABC Drone Project was to allow for amateur sports teams, or sports that otherwise did not have access to live broadcasting capabilities, to live stream their games via the Open Broadcaster Software (OBS Studio, 2012), or some other live streaming software. The goal was to allow those who could not make it to the games to be able to watch their local teams or kids play from wherever they may be in the world. However, after conducting some market research and talking to various coaches and clients, we came to the conclusion that more value could be added to our product if we shifted our focus from live streaming to post-processing. Many of the coaches we talked to expressed more interest in the ability to speed-edit games than the ability to live stream them. Being able to quickly and easily generate high-quality tactical game video was attractive to many of the coaches we talked to. Statistics tracking, play tagging, and key play detection were among some of the features that we discovered would make our product much more valuable to our target audience. Because of this new found information, we decided to shift the focus of our project to a post-processing oriented solution. By making the product more enticing for coaches, in turn, the coaches are more likely to use the live-streaming capabilities if the feature is hassle-free.

When determining the best starting point for the project, it was initially thought that automatic detection and tracking of players would be the main focus, so many of the primary thoughts and programs were based around the OpenCV library (OpenCV, 2021). OpenCV is available in both C++ and Python, and after some further research, the team determined that Python would be the most flexible choice, without losing much performance since the Python library is built on the C++ implementation. The OpenCV library allows for easy manipulation of images, with a variety of additional functions to aid with computer vision applications. Had the team known about the pivot to a post-processing focus earlier in the semester, the library of choice may have been focused around something such as FFmpeg, which allows for greater post-processing control.

Although some of the progress made during the semester may be placed on the backburner for a while, the concepts and routines will be able to be easily modified to greatly improve more advanced features later in the life cycle of this project. This report will cover all of the processes that have been developed this semester as well as

highlight the progress made post-pivot along with the future plans. With further reading, this paper will serve as a comprehensive guide to individuals looking to understand the concepts behind controlling the drone as well as those looking to further iterate on the work and ideas that have been developed throughout the course of this semester.

**Project Stages**

After completing much of the preliminary research and work to get our project off the ground, we quickly discovered that many of our local machines did not possess the computational power we needed. This is largely due to the introduction of a Strided Network convolutional neural network (CNN) machine learning algorithm. Our initial software design allowed us to successfully track and identify players on a field around 70% of the time. We decided to introduce a machine learning algorithm to try and increase our accuracy. Using the limited data we had available, we were able to generate an algorithm that was able to identify a player on a field over 85% of the time, the accuracy we were hoping to achieve. However, this increase in accuracy came at the expense of run speed and frames per second. After incorporating our model into our tracking system, we lowered our frame rate to around one frame per second. This is unacceptable for live use and also aided in our decision to switch our project emphasis to a post-processing approach.

In order to increase the computation power of our tracking system, we are looking to incorporate Amazon Web Services (AWS) into our software solution. AWS has the ability to provide us with the necessary computational resources we need to make our solution a viable option for coaches. At the time of writing this report, April 29, 2021, our goal to incorporate AWS has been pushed off to a later time. This is largely due to the expense of this migration and therefore all of the code written is still aimed to be run locally. More information on our plan to incorporate AWS can be found in the Future Plans section of this report. Although this is now being done locally, the goal for the AWS integration consists of two main components: (1) Edit Decision List (EDL) file generation, and (2) post processing video generation and output. The EDL file is a simple list of all the necessary pan, tilt, and zoom (PTZ) actions needed in order to frame the video shot on the action present on the field. Since the systems used for development do not possess the necessary hardware to incorporate the automatic tracking into the EDL file creation process, the ABC Drone team decided to implement a manual tracking solution we are calling Plan B. From a high level perspective, Plan B sits in the EDL File creation stage of our overall solution. It utilizes a video game controller (such as Xbox or Playstation) to allow a user to manually enter the pan, tilt, and zoom actions. These actions, similar to the automatic PTZ actions, are recorded in

the EDL file that will then be used to frame the gameplay on the field. Figure 1 gives an example of a framed shot that centers all the players in the middle of the camera shot.

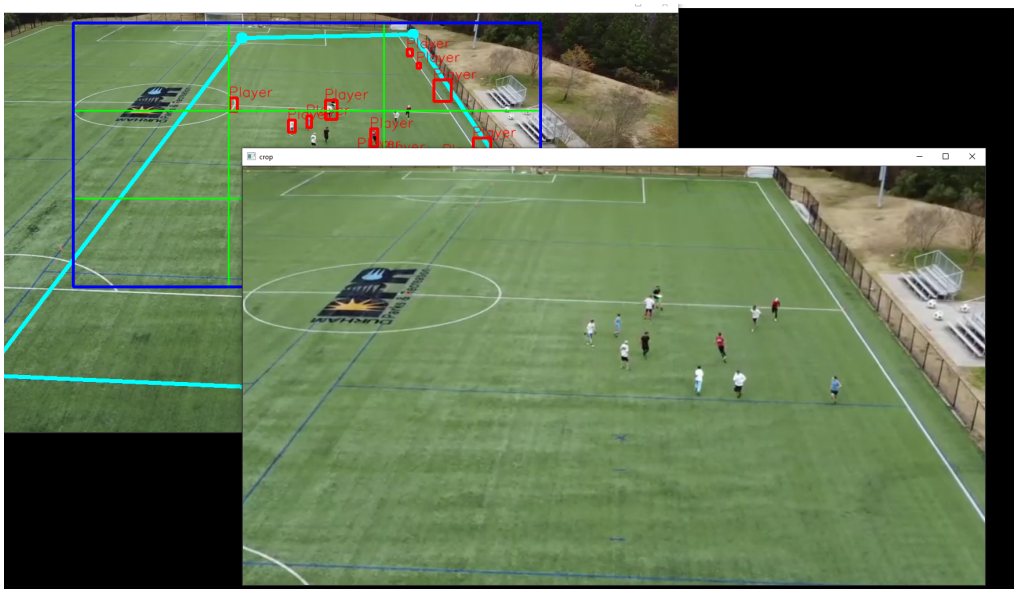


Figure 1. Example of PTZ that better frames the action on the field.

As you can see from the images in Figure 1, our PTZ system allows for a much better shot of the action taking place on the field. If you draw your attention to the window behind the frame in focus, you will see what the raw footage looks like, i.e., very far away and not necessarily focussed on any of the action taking place on the field. This is the motivation behind our PTZ capabilities. We want to generate footage that will allow coaches to focus solely on the action on the field, not worrying about the rest of the field that is not occupied during game play. The blue box that you see over a portion of the background image is where we will draw our EDL file from. That is in essence what we want the final video to look like. The EDL file will contain all the necessary PTZ information for each frame that achieves the desired result. This file will then be used in the final stage of our solution that will produce the final output video.

The final component of our solution will include a simple output program. The goal of this part is to output the final footage that has been framed and zoomed accordingly. This program will take the cut video file and the EDL file as inputs, and output the cropped video, the blue box from Figure 1, to the user. As it currently stands this stage of the solution is currently executed locally, however long term goals place this component on AWS to decrease the run time of this stage.

———An additional feature we included in our solution was player stats and highlight clip generation. During the live recording of a game, users will be able to use our GUI

application featured in Figure 2. Through this GUI they will be able to record stoppages in play along with key plays throughout the game. Upon completion of the filming, the application will create a cut video, eliminating all segments in the video that contain no actual gameplay. This will reduce the size of the video file as well as speed up the overall post-processing operations. Upon completion of this cut video generation, our users can also choose to run our player stat generation program. This script creates a folder for each player on the roster under a parent Players folder. Inside each player folder there will be a text file containing all of their stats from a given game as well as the individual highlight clips associated with each stat.

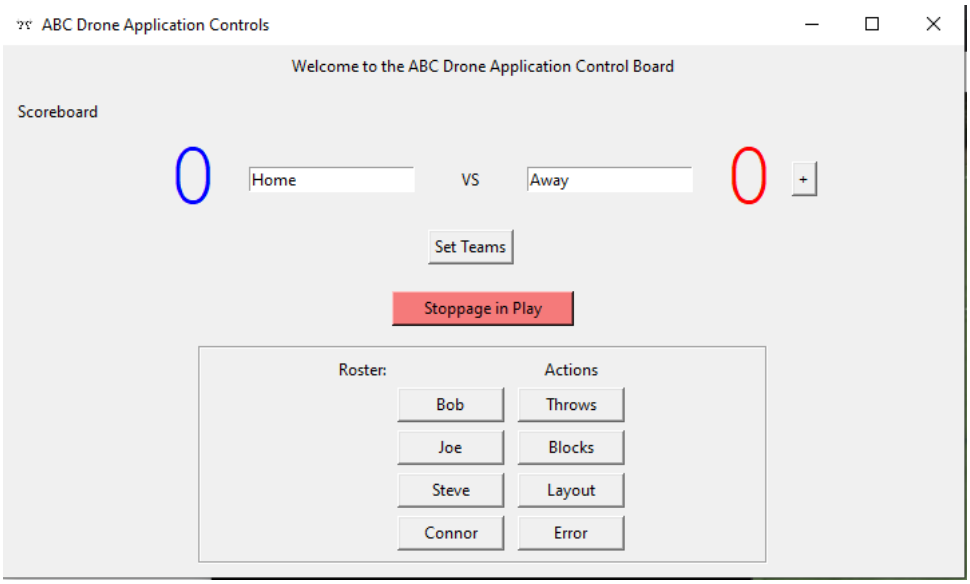


Figure 2. Proof of Concept GUI for Cut Editing and Statistics Logging.

**System Overview**

Given these major components, the general overview of our software solution is as follows. A user will use our GUI application to live record and track the gameplay on the field through a tethered drone. This process will produce both a cut video and a key play log file. The cut video will consist only of active gameplay as any stoppage in play will have been cut out. The key play log file will contain information regarding important plays throughout the game, i.e., scores, errors, fouls, tackles, etc. … From here, coaches can choose to export player stats and highlights as mentioned earlier, or continue on to the EDL creation phase of the process. There are two main ways the EDL files are created: through the manual tracking in Plan B, or through the automatic tracking using our machine learning neural network. Either of these two options will generate the EDL file that identifies how the video should be cut up and framed in order to best frame the gameplay in the output video. After this is finished, both the cut video and the newly created EDL file will be sent over to our output video generation program.

This program will take the video and use the EDL file to produce an output video that is cropped and framed. After this is complete, the output video will be made available to the user for viewing and distribution. Figure 3 better illustrates the flow of this entire process.

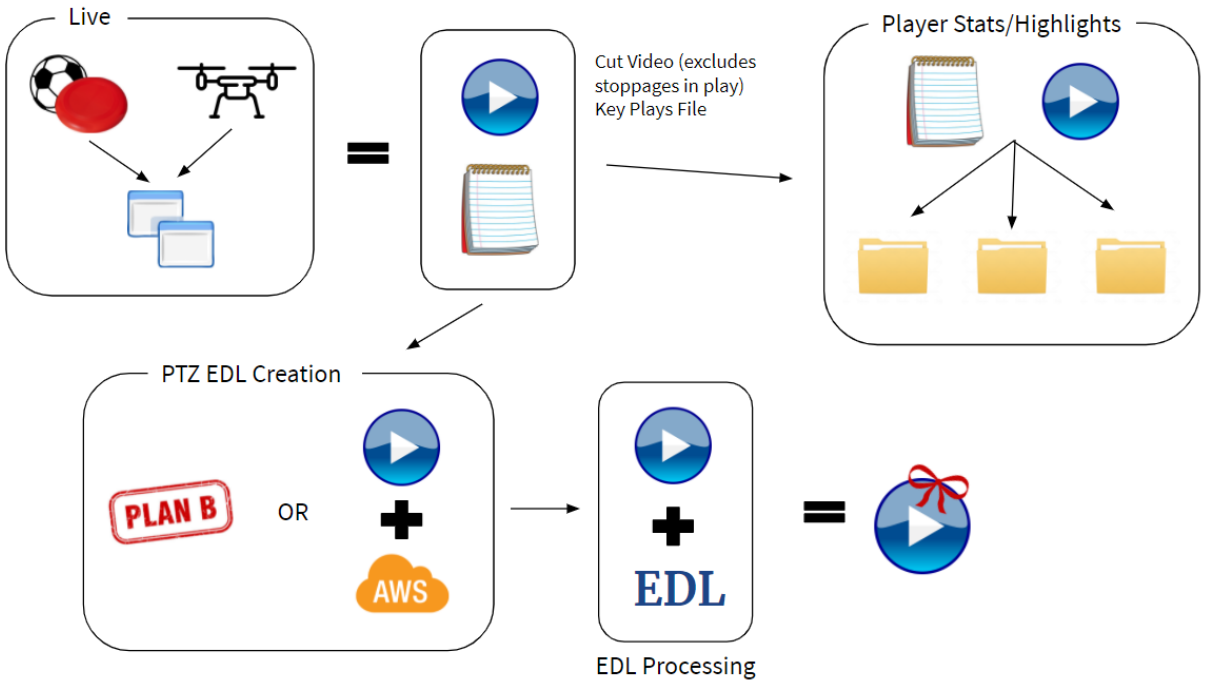


Figure 3. High Level Overview of the Software Solution Workflow.

A complete and comprehensive overview is that this project will allow coaches to utilize a tethered drone to capture high quality game video in an extremely easy fashion. Upon completion of the filming, the coaches will be able to take the game video and upload it to our software solution. Due to the automated nature of our program, after uploading the video our users will simply be able to wait until our program has finished its processing. Upon completion of the processing, the video will then be made available to the coaches for analysis or distribution. The goal is to allow coaches and players to go from game time to edited tactical videos in a few hours.

## Requirements

This project as a whole has both hardware and software requirements. For the sake of modularity and simplicity, we will be focusing on the requirements for the software side of this project. From here on out, we will be assuming that ways of collecting the necessary video to use with our solution are taken care of. We will point

out that the recommended videos for this project are filmed looking from endzone to endzone. This gives the best tactical view of the action on the field and will give coaches the necessary angles needed for post-play analysis.

Figure 4. Sample Drone Camera Capture Position and Angle.

All of the code for this project was written in Python. Due to our heavy use of the OpenCV library for computer vision, we used Python 3.9 to increase compatibility with all of the libraries we used. Steps on how to install Python 3.9 can be found in the User Manual section of this paper. We also made use of various different libraries throughout the development of the code. Table 1 lists the Python libraries as well as the version numbers used.

| **Python Library** | **Version Number** |
|---|---|
| numpy | 1.20.2 |
| imutils | 0.5.4 |
| opencv_python | 4.5.1.48 |
| Kerras | 2.4.3 |
| matplotlib | 3.4.1 |
| Pillow | 8.2.0 |

| | |
|---|---|
| youtube_dl | 2021.4.7 |
| tensorflow | 2.4.1 |
| scikit_learn | 0.24.1 |
| pygame | 2.0.1 |
| pafy | 0.5.5 |

Table 1. Necessary Python Libraries and their Version Numbers.

Information on how to install all of these necessary libraries can also be found in the User Manual section of this paper.

For the sake of your understanding, we will go over a few of the more important libraries we used. First up is the NumPy library. NumPy adds support for large multi-dimensional arrays and matrices. We used NumPy to store data used in our computer vision implementations. Another very important library we used was imutils. This library adds a few convenience functions to aid in image processing. We used this in order to maintain the same pixel ratio when creating our Strided Convolution Neural Network. In order to create and use this neural network, we used both the Keras and TensorFlow libraries. TensorFlow allows for some basic machine learning operations. Keras acted as an interface for the TensorFlow library and made implementing our neural network incredibly simple. The OpenCV library is one of the most important libraries we used throughout this project's development. OpenCV aids in the use of computer vision and we made heavy use of this in our first pass of player identification and tracking. OpenCV allowed us to threshold our input video, capture the contours, and make educated guesses on where players were on the screen. OpenCV alone allowed us to achieve around 70% accuracy in player identification.

The requirements listed above are resources needed to develop and run our solutions on a local machine. Unfortunately due to hardware limitations, low performance should be expected when running the automated tracking with the neural network. We recommend excluding the use of the neural network when running locally, or using Plan B, in order to increase run speed and frames per second. Ideally, this program will sit on a server, most likely on AWS, that has access to greater computational power, allowing for all of the aspects of our program to function at a high speed.

## Design

As shown in Figure 3, the overall design for this project boils down to three main stages (the EDL generating and processing stages are considered to be a part of the same final stage): the live stage, player stats and highlights stage, and finally the post-processing stage. To make the code easier to understand, we have written each section as a stand alone item that can be executed and tested on its own. The team plans on wrapping these various pieces together later on in the development process, but as of April 29, 2021, the pieces will remain separate to allow for better modularity of the entire process.

### Live Stage

The first major component is the live stage. This phase of the software includes the entire hardware aspects of the ABC Drone project. A drone will be used to capture game video via a tether that will be connected to a computer. This computer will be running our GUI application, allowing a coach or other user to keep track of game stoppages as well as player stats throughout the game. Upon the completion of the game, the live stage will generate a cut game video that is free of gameplay stoppages. Most of the discussion throughout this paper is based on using a pre-recorded gameplay video, however, the intended use of this project will not produce a full, uncut video, instead only outputting a cut video consisting only of actual gameplay. Of course this is reliant on the quality of work the user does when operating the GUI application. Failure to stop and start play recording will result in inconsistent video recordings. This makes it crucial that the operator of the drone and the application remains attentive to the action taking place on the field.

The GUI application can be seen in Figure 2. The primary features of this application are the stopping and starting of gameplay recording, as well as the player roster and action buttons. Currently the roster and action buttons pull from text files in the project parent directory. This allows the application to scale according to team size and sport, giving the users customizable buttons to suit their needs. When initially running the application, the GUI and game feed will appear in separate windows. After the gameplay has finished, the application can be closed and the cut video will be generated. The output video from this phase will be heavily used throughout the remaining stages of the overall process.

**Player Statistics and Highlights**

The player stats and highlight phase of the project is optional and not needed for the execution of the final stage. The ABC Drone team recognized that a very valuable component of game film is the ability to create highlight clips. We believed that adding this feature would make our product much more appealing to both coaches and players. This phase of the project takes both the cut video and the key play file generated from the live capture phase. The key play file contains listings of all the important plays that took place during the course of the filmed game. Each time the user logs an action using the live GUI application, the action and player(s) involved are recorded into this file. When the user executes the player stats script, the program iterates through the roster file, creating a folder for each player on the roster. The program then runs through the key play file, looking for each player's involvement in any of the key plays. If the current player is involved in a key play, this will be logged in that player's statistics file and a highlight clip will be generated accordingly. After the execution of this phase, each player on the team will have their own folder containing a text file of all the key plays they were involved in, as well as clips showcasing those key plays.

**Pan, Tilt, and Zooming**

The final stage in the overall design process is the EDL generation and processing stage. As seen in Figure 3, there are two main ways to generate the EDL file. The first and more trivial is Plan B, the manual tracking route. Plan B consists of two key components: (1) the Plan B Python script and (2) the Plan B configuration JSON file. The Plan B configuration file contains important information for the execution of the Plan B program such as the input video, edit speed, and controller type. If the configuration file present in the project files is used, everything should work out of the box, with no changes needed. Executing the Plan B program will allow the user to PTZ around the cut video generated in live phase. These actions will be recorded in an output EDL file that will then be used in the EDL processing step of this phase.

The secondary way to generate the EDL file in this phase is to use the auto tracking software. The main file that drives this section is, conveniently, called driver.py. This is where the main loop of the auto tracking resides. After carefully analyzing the overall auto tracking process, we identified three critical components and broke those down into the following helper files:

- centermass_util.py
- fieldBound_util.py
- ptz_util.py

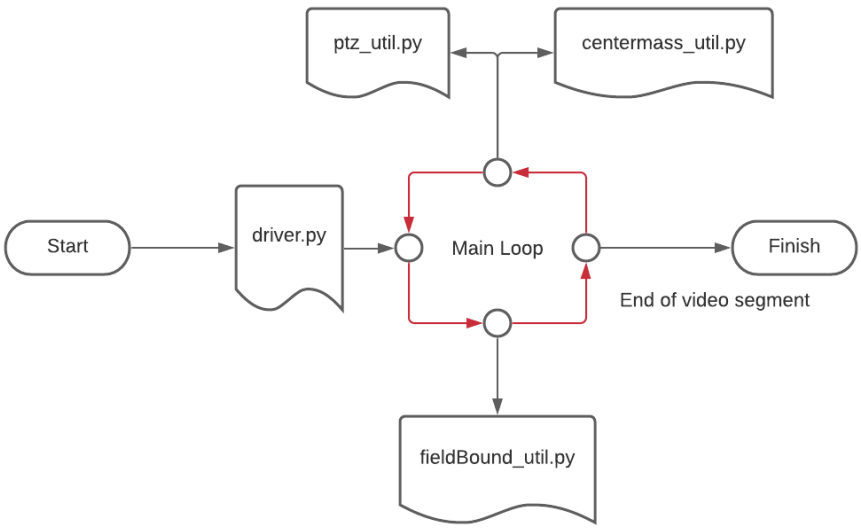


Figure 5. High Level Overview of Auto Tracking Design and Flow

A high level overview of the overall design of the auto tracking step can be seen in Figure 5. As you can see, as the main loop of driver.py is executed, the helper files are called at various parts along the execution process. This allows for certain complex components to be separate from the main driver file, making the driver code much cleaner and easier to understand.

When execution starts, the video is continuously played until a field boundary is set. The purpose of the field boundary is to restrict the portion of the screen where the computer will look for players. This helps to eliminate the detection of people on the sidelines that could factor into the mathematical calculations later on. The fieldBound_util.py is responsible for this functionality, allowing the user to click on the corners of the field, outlining the field of play. Once the field boundary is set, the program will then begin to track players. Every 10 iterations through our main loop, corresponding to 10 frames of the input video, our computer vision components will execute. This will cause the computer to look for what it thinks are players on the field using various CV2 functions as well as our Strided Neural Network. Once we have found all the players on the field, we then calculate what we are calling the center of mass, the (x,y) coordinate that represents the center of the players on the playing field. These mathematical calculations are done in the centermass_util.py file. Once this center of mass is found, we want to frame the video around that center of mass. In other words we want to pan to keep the center of mass in the center of our output video as well as zoom in so that the edges of the field boundary are as close to the edge of the

video as possible. This is done in the ptz_util.py file. Combining these 3 major components, with our computer vision solutions, we get a functional program that will automatically pan, tilt, and zoom to frame the action taking place on the field. For a more detailed look at how we go about accomplishing these various pieces, please refer to the Developer Manual section of this paper. The PTZ actions generated by the auto tracking program will also be logged in an output EDL file. This file, similar to the one generated by Plan B, will then be used in the EDL processing section of this stage.

The final step is to use the EDL file generated by either of the two methods just discussed to generate the output video. This output video will be cut and trimmed to exclude any stoppages in play, as well as be zoomed in to follow the action taking place on the field. The script responsible for this functionality takes in the cut video, generated in the live phase, as well as the EDL file as input. The program will then execute, resulting in the desired output video.

## Implementation

The overall implementation of this project is during the Spring 2021 school semester. Although there will be imperfections, we as a team know that this will not be the final product. Much more data collection will need to be done before we can really fine tune the code. The goal for this project is to successfully be able to autonomously generate an EDL file, then use that file to automatically create cut up tactical videos to be used in post-game analysis.

As of April 29, 2021 the drone team was still constructing our custom drone. Because we are looking to use a heavy duty drone and high quality camera, we are looking to customize the drone to allow for a charging and media transfer via a tether. We have successfully been able to capture some test footage using smaller drones, but we hope to have our drone fully functional by the end of May 2021 in order to capture some realistic footage to put through our software solutions.

Overall, the implementation of the software side or this project will be completed before the end of May 2021. We are hoping to demonstrate the project as a whole come the end of the semester, however due to the hardware requirements and the setbacks with our drone production, the timeline for a full scale demonstration is unclear. See the Testing, Evaluation, and Assessment section for more details on the testing and evaluation process.

## Testing, Evaluation, & Assessment

Most of the testing and evaluation done for this project has been conducted internally. Our client has been able to provide us with a few files of sample game footage from multiple sports (primarily soccer and ultimate frisbee). Using these, we have been able to test and evaluate our progress as we've gone along. Much of our coding sessions and work has fallen into the cycle of: code, test, evaluate, repeat. Whenever subtle changes were made, we tested it to see if we like the changes, and then continue until we have completed the necessary steps to move forward with the project. Throughout this process, once larger steps are completed, we have met with Charles to showcase and discuss our current progress. During those discussions we highlight some of the key functionalities we have added, as well as some of our next steps. At the present time of writing this report, this methodology has worked extremely well for us and we do not plan to deviate from it. As we move forward towards a more complete product, we are hoping to hold a few recording sessions in order to obtain more sample clips. We have graciously received permission from New River United, a youth soccer organization local to Blacksburg VA, to come out and film some of their matches using our drone. The goal is to record this footage and use it to test the overall functionality of our project. We are hoping that we can capture footage and run it through our solution to auto generate the EDL file that can then be used to create the output video. These videos will then be made available to the teams and coaches accordingly.

Once a full scale test of the project as a whole has been conducted, some of the key things we as a team want to look at are: accuracy of tracking, smoothness of video, quality of video, as well as speed in which we can go from filming to generated tactical footage. We also plan to reach out to coaches a few days after returning their game footage to get their feedback. Coaches and teams such as New River United are our target audience and we want to make sure that we tailor our product to meet their needs and expectations.

Although we are optimistic about our chances to finish, test, and fully evaluate our project before the end of the semester, we recognize that unforeseen obstacles could set us back. We are still in a development stage, both in our striving for more accurate tracking as well as our final video cutting program. Additionally, we would also like to acquire much more testing video that we can use to generate more data on players versus not players. Doing so would allow us to train even more accurate AI which would in turn give us higher accuracy values when it comes to our player identification and tracking. Unfortunately issues with the hardware team and the drone have disabled our chances of getting this footage sooner. Although we can still move

forward with other aspects of the project, a polished product may not be feasible before this additional test footage is acquired.

As of April 29, 2021 we have been unable to record any new footage featuring the New River United soccer program. This has limited our ability to test our solutions and acquire feedback on its performance.

## User Guide

* All files including code and sample data can be found in the VTechWorks submission for this project.*

The following section serves to walk a new user through the installation and usage of various aspects of the project. At the time of writing this report, the easiest way to execute our project is to do so on a local level using the command line. The best and easiest way to execute the code for this project is through a Windows PowerShell instance. To open a PowerShell instance already in the project parent folder, first navigate to the folder using your system's File Explorer. Once there, hold down the Alt key and press the F key. This should open up a menu somewhere on your screen. From there, you can either press the corresponding key or select "Open Windows PowerShell". Upon doing so, a PowerShell window will open, already starting in the necessary folder.

### Dependency Installation

Throughout the development of this project, many additional libraries were used and needed to be installed. The current version of Python being used to execute all of the project code is Python 3.9.4, however any recent version of Python should do just fine. For more information on how to download and install Python, please visit their official website at www.python.org. Once Python is installed, the next step is to install all of the external libraries listed in Table 1. The easiest way to do this is to obtain the requirements.txt file from the project files. The requirements.txt file will be at the top level of the code file structure.  On the command line, use the following command:

*pip install -r requirements.txt*

This command will install each dependency needed to execute the project code. If any errors occur during this process, it is possible to manually install each library using the command:

*pip install <library name>*

**Parent Directory**

The parent folder for the project code, as seen in Figure 6, contains many important files as well as two sub-folders. For convenience we have gone ahead and placed both a raw video clip, clip1.mp4, as well as an already cut video, cutVideo.mp4, to ensure that all aspects of the project can be run out of the box. An explanation of each file and its unique purpose can be found in Table 2.

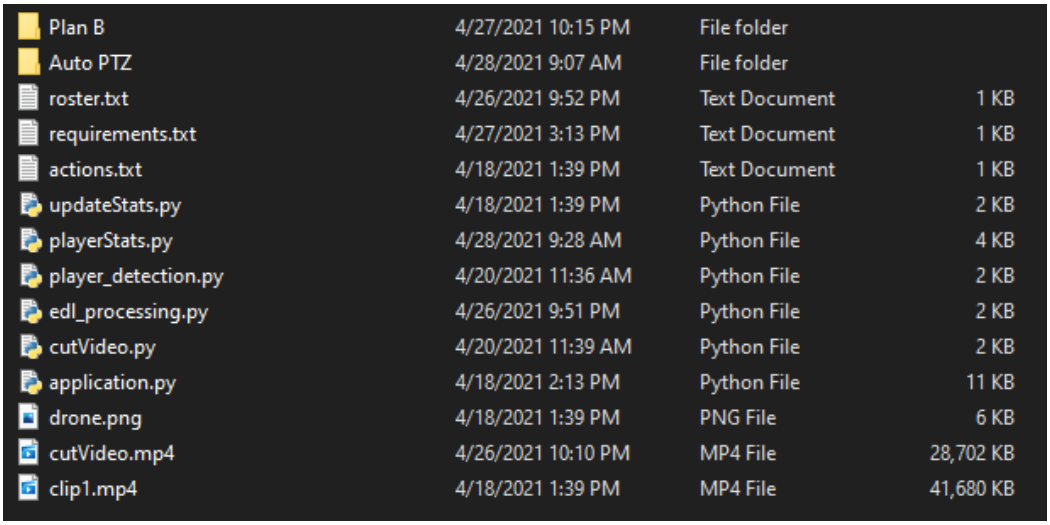


Figure 6: Parent Level Directory Files.

| **File** | **Explanation** |
|---|---|
| requirements.txt | List of library dependencies used in the project code. Used to easily install all necessary libraries. See the Dependency Installation of the User Manual for more details. |
| roster.txt | A text file containing all player names on the current team. Changes to this file will be reflected in both the player buttons on the GUI application, as well as in the subfolders produced during the player stat script. |
| actions.txt | A text file containing all game actions |

| | |
|---|---|
| | performed by players. Changes to this file will be reflected in the application GUI. |
| updateStats.py | This Python script is responsible for updating and keeping track of stats throughout the GUI application's execution. This script is never executed directly by the user, instead is called as a helper script from the application.py file. |
| playerStats.py | This script is responsible for the individual creation of the player stats and highlights. Creates a folder named Player, giving each player in the roster their own subfolder in this directory. Each player's unique files are then placed in their own folder. |
| gameFeedDisplay.py | This script is called by the GUI application and is responsible for displaying the game feed while the application is being run. |
| edl_processing.py | This Python file is the last element of the project hierarchy. It takes in the cut video from the applications, as well as an EDL file to produce the final cut output video. |
| cutVideo.py | This script contains helper features to the GUI application. The application uses this file to help export the cut video, removing all stoppages in play from the video file. |
| application.py | The GUI application script. Running this script will produce the GUI as well as kick start many of the other files in this directory. |

Table 2: Parent Directory File Explanations and Purpose.

As of right now, the GUI application pulls in video feed from clip1.mp4. As mentioned earlier in this report, the goal for this is eventually to be used on a live video feed, but for the purpose of local testing we will continue to use the sample clip. To start

the execution of the GUI application, use the following command on the Windows PowerShell instance opened earlier:

*python .\application.py*

If working correctly, two windows should open up: (1) the GUI application and (2) the video feed window. From here, you can stop and start play, update scores, and log key plays throughout the game. After the video has ended, you can exit out of the GUI application. This will start the process of video cutting. After a few seconds, the program will finish, printing valuable information to the command line. In the parent directory, 3 files will be created. The first and most important one is the cut video file, labeled cutVideo.mp4. The next two are the stats.txt and corrStats.txt files. The stats file contains all key play logging from the raw clip as well as the frame number where the associated key plays took place. The corrStats.txt file contains what are referred to as the corrected stats. When removing all of the stoppages from the original video clip, this changes the frame numbers of the entire clip. The corrected stats file contains the new frame numbers for the frames that are now associated with the key plays. For example, if a key play happened at frame 10, but we removed a stoppage in play from frames 5-7, the resulting video clip is now only 8 frames long. This changes the frame that the key play occurred at in the cut video. This correct stats file contains those corrections. With these corrected stats, we can now advance to the player stats and highlights phase of the program.

**Player Stats and Highlights**

The player stats and highlights portion of the project code also resides in the parent directory. The Python script playerStats.py is responsible for the various functions used to achieve the desired output. To begin this phase, execute the following command:

*python .\playerStats.py*

If done correctly, a Players folder will be created in the parent directory, containing a folder for each player present in the roster.txt file. An example of this can be seen in Figure 7.


Figure 7. Individual Player Folder Creation.

Each player sub-folder will contain a stat text file containing all key plays that this player was involved in. For each stat in this file, a paired video clip will also be created to showcase the play. Right now, the highlights are set to be 300 frames long. This is easily changed in the code of the program.

**EDL File Generation - Plan B**

The next step in the process is to generate the EDL file that will be used to generate the final cut video file. The first way to go about this is using Plan B, the manual tracking method. The entirety of the Plan B program resides in the Plan B folder in the parent directory. One important thing to note is that you will need to have some form of a game controller connected to your system. Failure to connect a controller will result in execution failure. We recommend using either a Xbox or Playstation controller as these seem to work the best, however any standard controller containing two joysticks should work. To begin executing, move into the Plan B directory. Here you will find the planB.py script. To execute this script use the command:

*python .\planB.py*

Once started, Plan B will fetch the cut video from the parent directory. You will also be greeted with a red controller testing screen. This will show you some of the buttons and axes the program recognizes on your controller. You may simply exit out of this window to continue. From here, Plan B will begin to run. You now have the ability to pan, tilt, and zoom around the video feed using the controller. On most controllers, the left analog stick will be used to pan and tilt the feed, while the right stick is used to zoom in and out on the video. Figure 8 illustrates this concept.

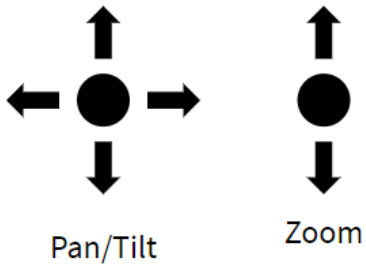


Figure 8. Plan B Controls Diagram.

Once the video clip is over, Plan B will exit, producing two files. The important file is the EDL text file. This contains all the information needed to recreate the PTZ actions made during the Plan B execution. You will need to copy or move this file to the parent directory in order to complete the final step.

**EDL File Generation - Auto Tracking**

The second way to generate the necessary EDL file is to use the auto tracking solution. All files necessary for this section can be found in the Auto PTZ folder in the parent directory. You should see 4 files at this level: driver.py, centermass_util.py, fieldBound_util.py, and ptz_util.py. These files are the main scripts behind the auto tracking execution. Also in this directory you will see a ML subfolder. This folder contains all the files and subfolders that make up the Strided Convolutional Neural Network, including the data generation program and the model creation and training program. More information on how to use and run the neural network will be given at the end of the User Manual. It should also be noted that the current state of the program does not include the use of the neural network. This is largely due to the fact that the frames per second will dramatically decrease while using it. Although this will decrease our tracking accuracy, it will greatly speed up the execution of the overall program. More information on how to include the neural network will be provided in the Developer Manual. To start the auto tracking program, use the command:

*python .\driver.py*

When the program begins, it will pull the cut video from the parent directory and two windows will appear: one titled “Feed” and the other titled “crop”. The Feed window will feature a green grid overlaid on the raw input video file. The Crop window will feature the same video, but without the green grid. The Crop window will reflect the pan, tilt, and zoom operations performed by the program. This will be mirrored by a blue box

present on the original Feed window later. Figure 9 shows both the Feed window and the Crop window. Note the Crop window is in front of the Feed window in this instance.

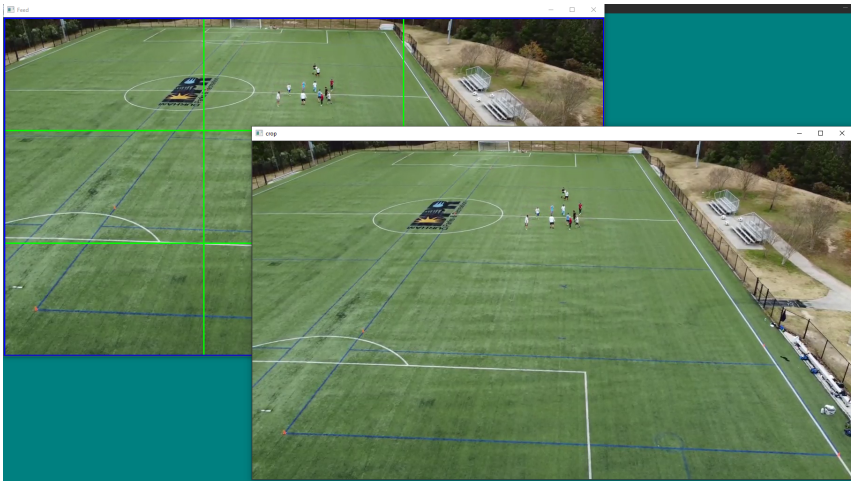

Figure 9. Start Auto Tracking Sample Run.

You may notice that nothing appears to be happening. Don't worry. This is by design. We did not want any form of tracking to take place until the field boundary was set up. That is the next step. Mapping the field boundaries is relatively simple and only requires four clicks. To do so, click on the Feed window to ensure it is the active window on your screen. From there, click on the four corners of the field that outline the playable space in your video. In this case, the field can be seen marked by dark blue lines and orange cones. Upon clicking, a light blue dot will appear on the video; this indicates a placed field corner. Continue placing the field corners until you have identified all of the corners. Do not worry about the order in which you place the corners. The program uses some basic math to determine which corner is the bottom left, top left, and so on. Once you place the final fourth corner, you will see the program come to life. A light blue line will appear outlining the entire field. This indicates the field boundary. Anything inside this shape will be valid space in terms of player tracking. You will also see red boxes starting to form around the players. This is to indicate the players the program has identified and is using as a part of our center of mass calculation. The center of mass of the players will be shown via a yellow dot that will periodically move around the screen as the game progresses. The last thing you will notice is a big blue box that begins to move around the screen. If you watch closely, you will notice that the green grid moves with this box. This box represents the Crop window feed. Looking closely you will see that the cropped feed matches the blue box exactly. An example of the field boundary, as well as the additional pieces just mentioned can be seen in Figure 10.

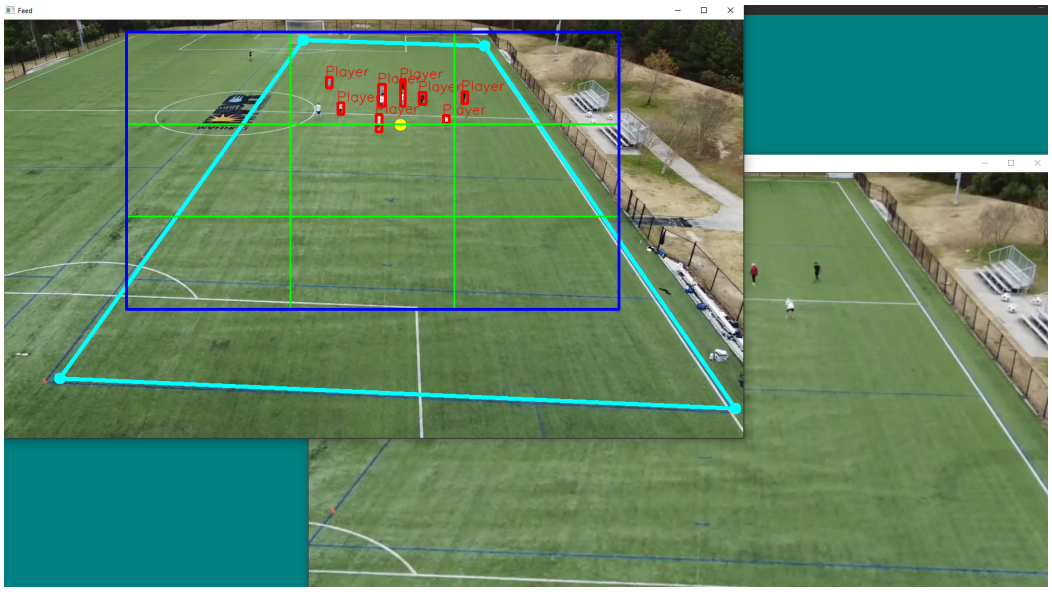


Figure 10. Auto Tracking Field Boundary Definition.

One key thing to note about the field boundary is that it is responsive. Due to errors in filming or weather conditions, the field boundary may not stay exactly lined up throughout the entire video. Because of this, we included an easy way to change the field boundaries during the execution process. To change a field boundary corner, simply click on the dot you would like to change. If done correctly, the dot's color should change to red, indicating that it is ready to be placed somewhere else. At this point, simply click wherever you would like to replace the field boundary. After doing so the field boundary should update accordingly. In the future, we as a team hope to include ways to automatically do this, however this is a manual process as of right now. An example of field boundary replacement can be seen in Figure 11.

Figure 11. Moving a Field Boundary Marker.

At this point, you can sit back and watch the program run. It should execute on its own until the full video clip is played. Other than requiring you to occasionally fix the field bounds, the program is fairly self sufficient. As you watch the Cropped video feed, you will begin noticing a pattern governing when it moves. If you draw your attention to the green grid lines inside the blue box, on the Feed window, you will notice that it moves whenever the center of mass dot leaves the center green box. This is by design and is still a work in progress. We are unsure of the exact approach we want to take when it comes to framing the video. Because of this, we are waiting to get some feedback from coaches and players on what would make the tactical footage the most useful. After we receive some feedback, we plan to integrate a much more sleek and effective pan, tilt, and zoom method.

After the completion of the auto tracking program, an EDL file will be created. Similarly to the Plan B EDL, this file contains all the necessary information to recreate the PTZ actions specified by the auto tracking software. To continue to the last step of the process, copy or move this file into the project's parent directory.

**EDL Processing**

The final stage of this process is to use the newly generated EDL file to create the final cut video. The script responsible for this is the edl_processing.py file in the project parent directory. To work correctly, both the edl.txt and cutVideo.mp4 files will need to be at the same location as the script. To execute, simply use the command:

*python .\edl_processing.py*

The program will then begin executing. After completion, the program will output the time it took to export the video, and a finalVideo.mp4 will be created in the parent directory. This video will feature all the stoppage cuts, as well as reflect the PTZ actions defined in the edl file. This video file can then be used however coaches and players see fit!

For a full demonstration of the outlined process please check out the demonstration video that can be found at Connor's YouTube channel (Bartal, 2021), or in the VTechWorks submission.

**AI Data Generation and Training Programs**

In this subsection, we will cover how to execute and use both the data generation program as well as the model generation and training program found in the ML folder under the Auto PTZ directory. Together these two scripts are used to generate sample data to be used to train machine learning algorithms. As we continue, we will be assuming you have a basic understanding of what machine learning is and how it works. If you need a brief explanation or a refresher on how this process of data generation and training works, please refer to the Wikipedia article on machine learning for a general overview (Machine Learning, 2021).

The key to a good machine learning solution is a lot of data. Unfortunately due to reasons explained earlier in this report, we were unable to come up with an abundance of data. However, we were able to use the limited sample clips we had available to generate some pretty decent models. To begin the process, make sure you enter into the data_create folder within the ML directory. Inside the data_create folder you will find a few things. First there are two files, click_extractor.py and data_combine.py. The click extractor is how we generate data to train and create our machine learning models. Our goal was to create a quick and easy way to generate lots of test data. To execute this program use the command:

*python .\click_extractor.py -v ..\..\..\clip1.mp4*

The -v indicates the path to the video file you would like to extract data from. Upon entering the command, you will be presented with something similar to Figure 12.


Figure 12. Start of Data Collection Program.

When the program starts, it will not automatically start playing. This is because we want you to be able to generate tons of data frame by frame. The basic idea behind this is the difference between a left and right click of your mouse. We needed an easy way to not only gather the data, but also label the data. This is where left and right clicks come in. As highlighted in Figure 13, we used a left click to indicate a 60x40 pixel image containing a player, and a right click to be a similar image that doesn't contain a player.

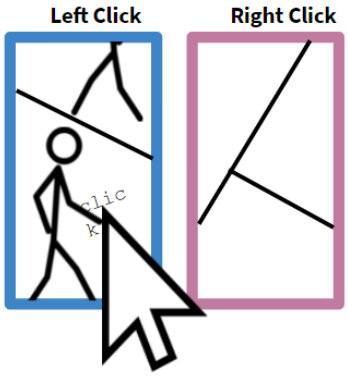


Figure 13. Data Collection Proof of Concept.

Using this data collection method, you can then work your way through video clips generating tons of data per frame. On the starting frame, left click on top of players, and right click on any space not occupied by players. The ideal scenario is to take a similar number of player versus non-player samples to be able to train the neural network. In the top left corner you will see the total number of each set you have created. Taking samples from areas that have a similar shape, size, color, to players will help the neural network differentiate players against other similar objects and patterns. Aim to include variety and take multiple samples of the same object with various framing. This way, we can optimize the decision making process of the neural network due to the abundance of data. We give the user the ability to choose the non-player samples to create more data for the neural network if it is having a particularly hard time with a certain identification instance. See Figure 14 as an example.

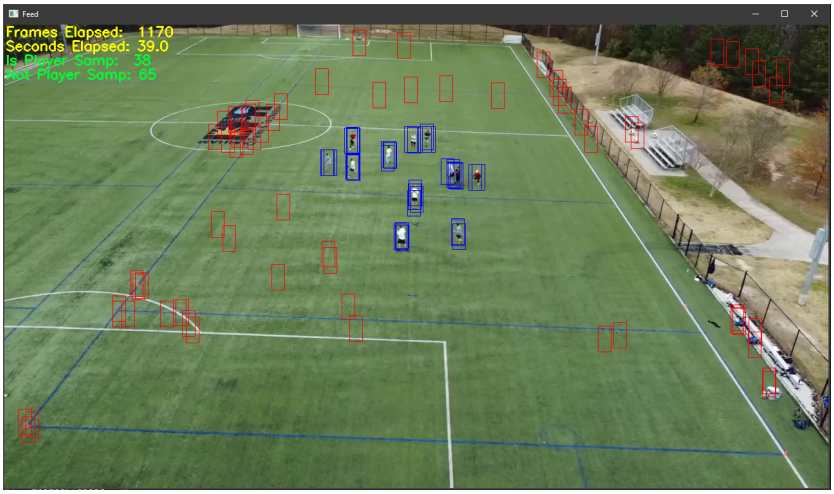


Figure 14. Sample AI Data Collection.

Once you are satisfied with your data collection for a particular frame, hit the ‘s’ key to advance forward by 30 frames. From here, repeat the process to generate more data. This process will continue until you have reached the end of the video segment. We would recommend doing this on more than one video clip in order to have a wider variety of data. If at any point you wish to quit out of the program, you can do so by hitting the ‘q’ key on your keyboard. Upon quitting out of the sequence or finishing the video clip, the program will then show you all of the data points it collected. An example of one of these photos can be seen in Figure 15. You may cycle through all the data points simply by pressing the spacebar, or you may quit the program by again pressing the ‘q’ key.


Figure 15. Sample Data Picture

Once you have created the data, the pictures as well as the answer key will be stored in the xData and yData folders. Again, we recommend doing this for multiple clips to create a variety of data for the training program.

The next step of the process is to combine all of our data files into one using the data_combine.py Python script. This can be executed using the following command:

*python .\data_combine.py -x .\xData\ -y .\yData\.*

You will notice that this program takes two arguments. The -x argument expects the file path to the folder containing all of our photos, or x, data, and the -y argument expects the file path to the answer key, or y data. This program will cycle through all of the files in each directory, combining them into one master x data file and one master y data file, placing them in the Aggregated folder. If everything works correctly, the program will print out the size of your master data and you will see two npy files in the Aggregated folder. This is now the data we will use to create our model.

To begin our model creation, make sure to navigate back to the ML folder. The two important files here are the stridednet.py and the run_strided.py files. The strided net Python file contains the entire makeup of the neural network. It is here that the model is actually constructed. The run strided script uses the strided net file to construct the neural network and then train it using the data set we provide. This script takes a number of command line arguments and switches. A full list of them can be found in Table 3.

| **Command Line Switch** | **Purpose / Use** |
|---|---|
| -x | Used to specify where the aggregated x data is. This switch is mandatory. |
| -y | Used to specify where the aggregated y data is. This switch is mandatory. |
| -e | Used to specify the number of epochs used for training the network. This switch is optional and the default value is 50. |
| -p | Used to specify the path of where to save the accuracy/loss plot. This switch is optional and the default value is plot.png. |
| -s | Used to specify the path of where to save the model. This switch is optional and users will still be asked if they would like to save the model after the execution of the program. |

Table 3. Model Creation Script Command Line Switches.

The very basic command that can be used to run the program is:

*python .\run_strided.py -x .\data_create\Aggregated\x_agg1.npy -y .\data_create\Aggregated\y_agg1.npy.*

This is a fairly long command so make sure you take advantage of tab completion on the command line. If everything works correctly, the program will come to life. At first you will see a lot of text; ignore this, it is not important. After a few moments you should start seeing the epochs being run. As we didn't specify a number of epochs to run, the default number of 50 will be used. This training may take some time depending on the size of your data sets. As it runs, you can monitor the progress of your model through the statistics printed out to the screen.

```
Epoch 1/50
4/4 [==============================] - 2s 155ms/step - loss: 1.7801 - accuracy: 0.5219 - val_loss: 1.1595 - val_accuracy: 0.5000
Epoch 2/50
4/4 [==============================] - 0s 29ms/step - loss: 1.9179 - accuracy: 0.4283 - val_loss: 1.1354 - val_accuracy: 0.5800
Epoch 3/50
4/4 [==============================] - 0s 30ms/step - loss: 1.8427 - accuracy: 0.5406 - val_loss: 1.1345 - val_accuracy: 0.5400
Epoch 4/50
4/4 [==============================] - 0s 28ms/step - loss: 1.4758 - accuracy: 0.5869 - val_loss: 1.1355 - val_accuracy: 0.5400
Epoch 5/50
4/4 [==============================] - 0s 29ms/step - loss: 1.4143 - accuracy: 0.6430 - val_loss: 1.1289 - val_accuracy: 0.5400
Epoch 6/50
4/4 [==============================] - 0s 31ms/step - loss: 1.5452 - accuracy: 0.5771 - val_loss: 1.1162 - val_accuracy: 0.5600
Epoch 7/50
4/4 [==============================] - 0s 31ms/step - loss: 1.7504 - accuracy: 0.5500 - val_loss: 1.1024 - val_accuracy: 0.5800
Epoch 8/50
4/4 [==============================] - 0s 30ms/step - loss: 1.5116 - accuracy: 0.5854 - val_loss: 1.0938 - val_accuracy: 0.6200
Epoch 9/50
```

Figure 16. Sample Epochs from Model Creation

After the model has been created, you will be prompted about testing the model with select photos. This is a space for you to see the effectiveness of the model on a sample set of your own choosing. If you would not like to test the model at this time, or have no pictures ready, simply type 'n' and hit enter. If you would like to test, simply type 'y' and hit enter. You will then be prompted to enter the path to photos you would like to use. It should be noted that the images must be .npy pictures.

After this you will then be prompted to save the model. Handle this situation similarly. If you select yes, the program will then ask you for a few notes regarding the model. This is your chance to type in a few words describing its performance. These notes will then be appended to the file name, allowing you to easily identify models later on. Once you are happy, hit enter and the program will begin to export the model to the saved_models folder.

```
Do you wish to test this model with select pictures? (y/n) n

Do you wish to save this model? (y/n) y

A word or two on performance of model: sample run.
Saving saved_model/stridedModel_samplerun._1
2021-04-07 12:12:04.540414: W tensorflow/python/util/util.cc:348] Sets are not cu
sider avoiding using them.

Exiting...
PS D:\Desktop - HDD\Senior-Year-Files\Spring\Capstone\AWS Stuff\Code\ML>
```

Figure 17. End of Model Creation Prompts.

At this point, you have successfully created a neural network model that can be used to increase player tracking accuracy. One last thing to note is that some statistics for the new model will be saved in plot.png. As you can see in our sample plot in Figure 18, the more epochs we went through, the higher the overall accuracy of the model was. After a few sample test runs, we were able to generate several models that had accuracy percentages in the lower nineties.

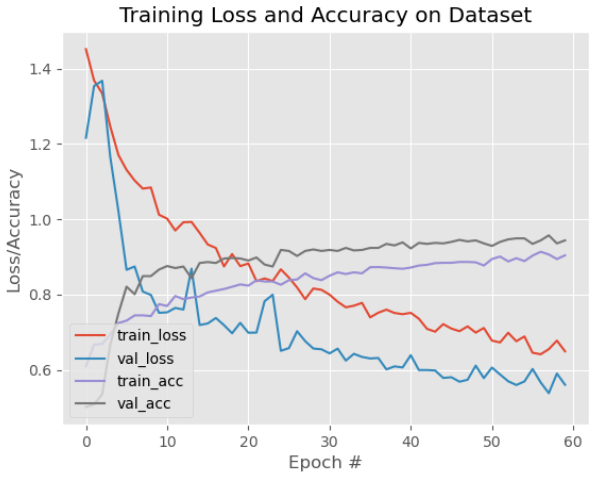


Figure 18. Example Accuracy Plot from a Model Creation Sample Run.

## Developer Guide

All of the files needed to follow along with the Developer Guide can be found on our GitHub. See the README to download the clip that we use in this report.

GitHub Link: [https://github.com/jaredcoop/Sports-Drone-VTechWorksFinal](https://github.com/jaredcoop/Sports-Drone-VTechWorksFinal)

Minimum files needed for this project can also be found in the VTechWorks submission for this project.

**Breakdown of Digital Pan, Tilt, Zoom (PTZ)**

The Digital PTZ functionality is the process of creating an EDL without manual input by considering the center of mass of contours in the frame. Inside of the Auto PTZ folder, there is *driver.py* which will be run to process a video for which an EDL is to be created. There are three helper objects which help the driver file do its job: *fieldBound_util.py, ptz_util.py,* and *centermass_util.py.* To understand how Digital PTZ works, these three helper files must be understood first.

**fieldBound_util.py**

```
class FieldBound:
    #in form of (x, y, color-value) ** going to have 4 points max
    points = []
    isAdjusting = False
    pointsSorted = False
    fieldContour = None
    upperR, upperL, lowerL, lowerR = None, None, None, None

    def mouse_click(self, event, x, y, flags, params):
        if event == cv2.EVENT_LBUTTONDOWN:
            if self.isAdjusting:
                for num, p in enumerate(self.points):
                    px, py, pcolor = p
                    if pcolor == (0, 0, 255):
                        #changing x and y position after adjustment
                        self.points[num] = [x, y, (255, 255, 0)]
                        self.isAdjusting = False
                        self.pointsSorted = False
            else:
                if len(self.points) > 0:
                    for num, p in enumerate(self.points):
                        px, py, pcolor = p
                        if (abs(px-x)**2 + abs(py-y)**2)**0.5 < 10:
                            self.isAdjusting = True
                            self.points[num][2] = (0, 0, 255)
                if len(self.points) < 4 and not self.isAdjusting:
                    self.points.append([x,y, (255, 255, 0)])
```

Figure 19. mouse_click method from fieldBound_util.py.

The way the FieldBound class creates the boundary for the field is by holding the corners of the field in a global *points* variable. Each point will hold an x, y, and BGR color value. There will only ever be four points or less in this global list. The mouse_click method in Figure 19 demonstrates what happens when the user clicks on the screen. If the user clicks on the video, a dot is created. With each click, a new dot will be created until there are a total of four dots. If the user clicks on a dot, the computer knows that the user seeks to move that dot and turns the point red. The next click will assign a new x, y position to the red dot and change it back to a teal color. Resources for understanding how clicks are registered in OpenCV can be found on YouTube (Mhproductionhouse, 2019).

```
69          #·Draw·lines·when·there·are·4·corner·points·selected
70          if·len(self.points)·==·4:
71              height,·width,·channels·=·image.shape
72              pts·=·np.array([self.upperR,·self.upperL,·self.lowerL,·self.lowerR],·np.int32)·
73              pts·=·pts.reshape((-1,·1,·2))·
74              cv2.polylines(image,·[pts],·True,·(255,·255,·0),·5)
75              blackBackground·=·np.zeros((height,·width,·3),·np.uint8)
76              self.fieldContour·=·cv2.drawContours(blackBackground,·[pts],·-1,·(0,255,0),·cv2.FILLED)
77              if·show:
78                  cv2.imshow('Contour',·self.fieldContour)
79      
80              #·bool·fieldContourSet,·image·fieldContour
81              return·(True,·self.fieldContour)
82          else:
83              return·(False,·None)
```

Figure 20. Snippet from draw_bounds method in fieldBound_util.py.

Once the points have been sorted, they are used to create a polylines object which draws a green polygon in the shape of the field on a separate window with the same size as the original video (line 76 in Figure 20). The computer is able to determine if an object is within the field of play by referencing this separate window and checking if the x,y position contains a green pixel.

**ptz_util.py**

```
6  class PTZ:
7
8      width, height = -1, -1
9      centerFrameX, centerFrameY = -1, -1
10     radiusFrameX, radiusFrameY = -1, -1
11     minX, maxX = -1, -1
12     minY, maxY = -1, -1
13     zoomScale = 1
14
```

Figure 21. Code Snippet from ptz_util.py.

The main purpose of the PTZ class is to keep track of important positioning information. The explanation of the variables is as follows:

| | |
|---|---|
| **width, height** | **Width and height of the original frame** |
| **centerFrameX, centerFrameY** | **x, y coordinates of center of zoomed (PTZ) frame** |
| **radiusFrameX, radiusFrameY** | **distance from center of frame to x and y boundaries** |
| **minX, maxX** | **the range of pixels on the x-axis of the original frame** |
| **minY, maxY** | **the range of pixels on the y-axis of the original frame** |
| **zoomScale** | **how zoomed-in the frame looks, with 1 being fully zoomed-out** |

Table 4. Variables of the PTZ Class.

```
26      def ptz(self, image, centerMass):
27
28          centerMassX, centerMassY = centerMass
29
30          # PAN TILT ZOOM SECTION
31          thirdHeightCropped = (self.maxY-self.minY)//3
32          thirdWidthCropped = (self.maxX-self.minX)//3
33
34          #If in top third of screen
35          if centerMassY < self.minY + thirdHeightCropped:
36              #PAN
37              self.centerFrameY -=1
38              #ZOOM
39              if self.radiusFrameX > int((self.width/6)):
40                  self.zoomScale += 0.005
41
42          #If in bottom third of screen
43          if centerMassY > self.minY + thirdHeightCropped*2:
44              self.centerFrameY +=1
45
46              if self.radiusFrameX < int((self.width/2)):
47                  self.zoomScale -= 0.005
48
49          #If in left third of screen
50          if centerMassX < self.minX + thirdWidthCropped:
51              self.centerFrameX -= 1
52
53          #If in right third of screen
54          if centerMassX > self.minX + thirdWidthCropped*2:
55              self.centerFrameX +=1
56
```

Figure 22. Snippet from PTZ method of ptz_util.py.

The main functionality of the *PTZ* method is to determine how the frame will be manipulated depending on the center of mass of the contours. The method takes in the center of mass and changes the global variables in Table 4 accordingly. The variables in lines 31-32 of Figure 22 can be changed to determine the sensitivity of the movement of the center of mass. The variables being changed under each if-statement determine how much the frame is going to move. The *PTZ* method then returns a tuple of (minX, maxX, minY, maxY) which represents the bounds of the new cropped frame.

**centermass_util.py**

```
18    def gather(self, x, y, w, h, modifier=1.05):
19        #Total XY mass of players
20        self.totalX+= (x+(w/2))
21        #Modifier to favor a lower center of mass
22        self.totalY+= (y+(h/2))**modifier
23        self.numContours += 1
```

Figure 23. Snippet from centermass_util.py.

The *gather* method from centermass_util.py keeps track of the total amount of "mass" of the contours. The **modifier** parameter is important as it enables the user to define an amount to "favor" players lower in the frame to ensure that all players are in view. If a player is at the closest part of the screen, he should be in view (zoomed out) so we assign a higher value to his mass in the y-direction using the modifier (line 22 of Figure 23).

**driver.py**

The driver.py takes the other three utility scripts and uses them to digitally pan, tilt, and zoom on given footage. The tracking used is based on what was learned from Adrian Rosebrock (Rosebrock, 2015) and the GitHub user KananVyas (KananVyas, 2018). From these two sources, it was found that the fastest way to identify players was by looking at their contours and analyzing the size, position, and color (see player_detection.py from KananVyas).

```
11  #READING / VIDEO PROCESSING
12  vidcap = cv2.VideoCapture('../cutVideo.mp4') #change video file here
13  success,image = vidcap.read()
14
15  height, width, channels = image.shape
16  numLoop = 0
17
18  fieldBound = FieldBound()
19  ptz = PTZ(width, height)
20  centerMass = CenterMass(width, height)
```

Figure 24. Snippet from driver.py.

Firstly, in the driver, the video path must be input (line 12 of Figure 24), and the helper objects must be instantiated with the correct parameters (lines 18-20 of Figure 24).

```
30  #MAIN WHILE LOOP FOR PROCESSING
31  #Read the video frame by frame
32  while success:
33      #PAN TILT ZOOM
34      minX, maxX, minY, maxY = ptz.ptz(image, (centerMass.X, centerMass.Y))
35      thirdWidthCropped, thirdHeightCropped = (maxX-minX)//3 , (maxY-minY)//3
36
37      #IF GETTING ~!ssize.empty() ERROR, the min/max X/Y are out of bounds (check zoomScale)
38      #The zoomScale + int rounding causes bounds to go negative
39      croppedFrame = image[minY:maxY, minX:maxX]
40      resized = cv2.resize(croppedFrame, (width, height))
```

Figure 25. Snippet from driver.py.

The while-loop on line 32 of Figure 25 reads the input video frame by frame until there are no frames left, with each frame being assigned to the image variable. The PTZ method from ptz_util.py is called to get the bounds of the frame. Lines 39-40 of Figure 25 take the bounds and create a new frame to be displayed.

```
43      statoutput = str(minX) + "," + str(maxX) + "," + str(minY) + "," + str(maxY) + "\n"
44      textEDL.write(statoutput)
```

Figure 26. Snippet from driver.py.

Figure 26 shows the bounds of the frame being written to the EDL file to be read at a later point to create the final video.

```
82      for i, c in enumerate(contours):
83
84          x,y,w,h = cv2.boundingRect(c)
85
86          #Detect players
87          if (h > (1.2*w)) and (w>2) and (w < 60) and fieldContourSet and (fieldContour[y][x] == (0, 255, 0))[1]:
88
89              player_img = image[y:y+h,x:x+w]
90              player_hsv = cv2.cvtColor(player_img,cv2.COLOR_BGR2HSV)
91
92              centerMass.gather(x, y, w, h)
93
94              cv2.putText(image, 'Player', (x-2, y-2), font, 0.8, (0,0,255), 1, cv2.LINE_AA)
95              cv2.rectangle(image,(x,y),(x+w,y+h),(0,0,255),3)
96
97          #every 10th frame, update Center of Mass
98          if numLoop % 10 == 0 and centerMass.numContours > 0 and i == len(contours)-1:
99              centerMass.update()
```

Figure 27. Snippet from driver.py Showing Processing of Contours.

Once the contours have been found, we only process the ones that have a certain dimension and are within the field boundaries (line 87 of Figure 27). The center of mass is gathered with each contour (line 92 of Figure 27), but is only updated every 10 frames (line 98-99) to keep the center of mass from updating too quickly and amplifying artifacts that may skew the calculation. This process repeats until the entire video has

been processed, and there is then an EDL file with the correct frame sizes to output a panned, tilted, and zoomed video.

Note: Running the edl_processing.py script with the correct EDL file that was just created from driver.py will yield your finished video.

**Digital PTZ with CNN**

Digitally creating the PTZ EDL can also be created with a machine learning model instead of analyzing contours. This method is much slower, but also more accurate; this process could theoretically sped up significantly by the use of GPU acceleration. GPU acceleration was achieved for OpenCV after months of attempting to find a straightforward path to do so, using the tutorial from The Coding Bug (The Coding Bug, 2021). Unfortunately, doing so led to ruining some dependencies and not being able to run TensorFlow which is required to use the CNN. It is also worth mentioning that it would be ideal to have both OpenCV and TensorFlow utilizing the GPU for maximum efficiency, which has not been achieved yet.

Continuing with the CNN method of identification requires that there is already a saved CNN model on hand, and is read into the script with the keras.models.load_model(<path to model>) function.

```
102         for i, c in enumerate(contours):
103             
104             x,y,w,h = cv2.boundingRect(c)
105             if (fieldContour[y][x] != (0, 255, 0))[1]:
106                 continue
107             #grab 40x20 around player
108             contour_img = image[y:y+h,x:x+w]
109             if w <= 20 or h <= 40:
110                 y_mid = int(y + (h/2))
111                 x_mid = int(x + (w/2))
112                 contour_img = image[(y_mid - 20):(y_mid + 20), (x_mid - 10):(x_mid + 10)]
113                 cv2.rectangle(image,(x_mid-10,y_mid-20),(x_mid+10,y_mid+20),(0,0,255),3)
114             elif w <= 30 and h <= 60:
115                 contour_img = resize(contour_img, (40, 20, 3))
116                 
117                 cv2.rectangle(image,(x_mid-10,y_mid-20),(x_mid+10,y_mid+20),(0,0,255),3)
118             else:
119                 continue
```

Figure 28. Snippet from ptz_cnn.py.

Similarly to Figure 27, Figure 28 shows the first steps within the for-loop that iterates through all of the found contours. Lines 105-115 of Figure 28 are taking a contour inside of the field bounds, and if they match a certain size, the pixels surrounding the contour are extracted and saved in the variable contour_img.

```
121             #let the model make a prediction
122             model_prediction = model.predict(np.array([contour_img]))
123             predict_list = (model_prediction.tolist())[0]
124             is_player = (predict_list[0] < predict_list[1])
```

Figure 29. Snippet from ptz_cnn.py.

The variable contour_img is then passed into the model.predict function (line 122 of Figure 29) where it can then predict if the contour is a player. If the contour is a player, then it can be factored into the center of mass as in the last few lines of Figure 27.

**Creation of Data for CNN Model Training**

The script /Auto PTZ/ML/data_create/click_extractor.py is what will be used to create the dataset necessary for training a CNN to identify players on a field. The mouse_click function found on line 51 of click_extractor.py is what will extract a 20 x 40 pixel picture and assign a label to the picture of *player or not player*.

```
61  ▼   if event == cv2.EVENT_LBUTTONDOWN and isInBounds(x, y):
62          numIsPlayer += 1
63          player = copyImg[y-h2:y+h2,x-w2:x+w2]
64          imgList.append(player)
65          labelsList.append(1)
66          isPlayerList.append((x, y))
67          selections.append(1)
68          image[40:90,0:300] = copyImg[40:90,0:300]
```

Figure 30. Snippet from click_extractor.py.

Figure 30 shows the left-click action if the click does not produce an out-of-bounds image (see method isInBounds on line 48 of click_extractor.py). The use of the copyImg variable on lines 63+68 of Figure 30 is to keep the GUI interface from affecting the images being put in the dataset for training. The right-click action is nearly identical to the left-click action, just assigning a label of notPlayer and displaying a red rectangle. The spacebar can be clicked to invoke the remove_last_selection method which will remove the last selection in case there was a misclick. Lines 130-174 of click_extractor.py define the actions for each of the keys:

q - quit
a - move ahead 15 frames
s - move ahead 30 frames

d - move ahead 60 frames
f - move ahead 150 frames
space - remove last click

Once all of the selections have been made, the program will display each selection one-by-one, which can be cycled through using the spacebar (lines 181-195 of click_extractor.py). The program will then go ahead and automatically save the pictures to the xData folder, and the labels to the yData folder (lines 198-212 of click_extractor.py).

Call the data_combine.py script with parameters of your xData folder and yData folder to create a dataset that combines x and y data from multiple clips.

Much of the data formatting and creation process was inspired by a tutorial from user Computer Science on YouTube (Computer Science, 2020). This resource was invaluable for understanding how to create the data collection application so that it would integrate seamlessly with the CNN structure.

**Creation of CNN Model**

Much of the knowledge required to understand how CNN's operate was learned through the YouTube channel Computerphile (Computerphile, 2016). This video gives a solid background to help understand the basics of what is happening behind the scenes which allowed problem solving to occur more quickly. In the ML folder, there will be stridednet.py which defines the structure of the neural network. The type of neural network that was settled on was the strided net model, which was built on the tutorial of Adrian Rosebrock at pyimagesearch.com (Rosebrock, 2018) and modified for better performance. There was not much rhyme or reason for the numbers that define the shape of the neural network as they were chosen with iterative testing. The way the performance of the structure was compared was by looking at a few key metrics seen in this GitHub post (Prosti, 2019). This article by Tanishq Gautam explains the metrics slightly more in-depth along with covering more basics about the Keras and TensorFlow models (Gautam, 2020).

To begin creation of the CNN model, call the run_strided.py script with the flags leading to the aggregated x and y data.

```
52  if len(import_images) != len(import_labels):
53      print('NOT SAME LENGTH')
54  # loop over the image paths
55  for i, img in enumerate(import_images):
56      # extract the class label from the filename
57      label = import_labels[i]
58      # if the label of the current image is not par
59      # are interested in, then ignore the image
60      if label not in LABELS:
61          continue
62      # load the image and resize it to be a fixed 9
63      # ignoring aspect ratio
64      image = import_images[i]
65      # image = cv2.resize(image, (40, 40))
66      # update the data and labels lists, respective
67      data.append(image)
68      if label == 1:
69          labels.append(np.array([1, 0]))
70      else:
71          labels.append(np.array([0, 1]))
72
73  labels = np.array(labels)
74
```

Figure 31. Snippet from run_strided.py.

The script will see if the x and y data are matching in length (line 52 of Figure 31) before proceeding. This strided net model uses binary labeling, so labels are represented by arrays of 1's and 0's. Lines 68-73 of Figure 31 convert the y data to labels that the model will be able to understand.

```
76  # convert the data into a NumPy array, then preprocess it by scaling
77  # all pixel intensities to the range [0, 1]
78  data = np.array(data, dtype="float") / 255.0
79
80  (trainX, testX, trainY, testY) = train_test_split(data, labels,
81      test_size=0.25, stratify=labels, random_state=42)
```

Figure 32. Snippet from run_strided.py.

It is crucial that the data be in the range of 0-1 or the prediction results will be highly inaccurate and even produce errors. Since our data was pixel values from 0-255, it was adjusted in line 78 of Figure 32. Line 80 of Figure 32 randomly selects the training and

testing data split for fitting the model - in this case, there was a 75-25 train to test split. Line 100 of run_strided.py determines the batch size and number of epochs when fitting the data.

```
manTest = input("\nDo you wish to test this model with select pictures? (y/n) ")
if manTest == 'y':
    import glob
    foldPath = input("\nType the path of the folder with the .npy pictures you wish to open: ")
    files = glob.glob(foldPath + '/*.npy')

    for pic in files:
        loadPic = np.load(pic)
        loadPic = np.array(loadPic, dtype="float") / 255.0
        predictions = model.predict(np.array([loadPic]))
        predict = predictions.tolist()[0]
        isPlayer = "Yes Player" if predict[0] > predict[1] else "No Player"
```

Figure 33. Snippet from run_strided.py.

After the model has been created, there will be an opportunity to test it against a folder of test pictures. The pictures must be in *.npy format (can be created using the same method as click_extractor.py). Figure 33 shows the prediction process on the given test pictures (cycle through the pictures using the spacebar). The program will then give an opportunity to save the model.

**GUI and Player Highlight Creation**

In the main folder, there will be the Python script application.py. Although this is something that will be used live in its final form, the current proof of concept is used in conjunction with a saved video clip. The script will produce a GUI that can be used to record stoppages and stats in a clip. The GUI can be seen in Figure 2. Before running the script, update roster.txt and actions.txt to reflect the players and stats that need to be recorded in the clip. In order to run the GUI and watch the video clip at the same time, each of these processes run on a thread.

**application.py**

This script takes the inputs of the buttons on the GUI and writes them to a text file. Stoppages in play are written to the EDL file while the stats are written to the stat file. Each entry has a description of the statistic followed by the frame number.

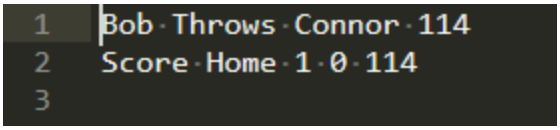


Figure 34. Example from stats.txt.

```
54      if len(action_list) == 3 and action_list[1] == 'Throws':
55          stat_file.write(action_list[0] + ' ' + action_list[1] + ' ' + action_list[2] + ' ' + str(getFrame()) + '\n')
56          updateScores('home', 1, button_play_start)
57          action_list.clear()
```

Figure 35. Snippet from application.py.

The global variable action_list holds the string to be written to the text files. Once a certain length or combination of words is met, the string is written to the file (Figure 35). The global variable disable_buttons holds the list of buttons that need to be disabled whenever there is a stoppage in play to avoid a user inputting an incorrect stat and ruining the timeline.

```
216 thread = Thread(target = execute_play_detection)
217 thread.start()
218
219 def on_closing():
220     forceQuit()
221     root.destroy()
222     
223     stat_file.close()
224     edit_file.close()
225     
226     cuts = updateStats()
227     print(cuts)
228     print("from app")
229     createCutVideo(cuts)
230
231     
232
233 root.protocol("WM_DELETE_WINDOW", on_closing)
```

Figure 36. Snippet of application.py.

On close of the thread, the stats text file needs to be updated to reflect the cuts that will be made to the video. If there is a stoppage in play from frame 30 to 60, and there is a stat at frame 75, the stat will need to be updated to reflect frames 30-60 being cut out, which would make the stat now at frame 45. This process happens at line 226 of Figure 36 and returns a list of cuts that the next script will use to create a video without any

stoppages in cutVideo.py (line 229 Figure 36). The call to updateStats at line 226 of Figure 36 also creates a new stat file with the correct frame-stamps.

**cutVideo.py**

```
15      fourcc = cv2.VideoWriter_fourcc(*'mp4v')
16      out = cv2.VideoWriter('cutVideo.mp4', fourcc, 30, (width, height))
17
18      numFrame = 0
19      while success:
20          numFrame += 1
21
22          include = True
23
24          for low, upper in cuts:
25              if numFrame < low:
26                  continue
27              elif numFrame >= low and numFrame <= upper:
28                  include = False
29
30          if include:
31              out.write(image)
```

Figure 37. Snippet of createCutVideo method from cutVideo.py.

The createCutVideo method in cutVideo.py takes in a list of cuts that is provided by the call to updateStats in Figure 36. The script then writes each frame that is not within those cuts (line 31 of Figure 37) to a new file given by line 16 of Figure 37. The output is a video with no downtime.

Note: The output process was slow, and there was hope that OpenCV GPU acceleration would decrease the processing time. Using an NVIDIA GeForce GTX 965M on a Surface Book, a test output of speeding up a video 2x resulted in a decrease of only 2 seconds, from 27 seconds without GPU to output a video to 25 seconds with GPU.

**playerStats.py**

The playerStats script works similarly to cutVideo.py, except it draws frame data from the corrected stats file. It reads each stat and determines the player involved and a five second window around the given frame for the stat. All stats for each player get written to one highlight video that gets placed in an individual player folder.

## Methodology

The thought-processes of this project led to four main goals:

- Autonomously Tracking Players on Footage
- Focusing on Action and Decreasing Wasted Space in Frame
- Creating a Simple Way to Edit Footage
- Having an Easy Way to Record Stats

The first two bullets are referring to the Digital PTZ and CNN endeavors, while the latter two refer to the EDL and video creation processes. The following indicates the order in which each was achieved and/or the order of importance of each feature.

### Autonomously Track Players on Footage

This task is to identify players in a video clip; this is shown by placing a bounding box around them.

1. **Import Video**
2. **Choose Identification Method**
    a. Contour - Looking at the shape and position of contours on the field can give a good idea where players may be - not always the most accurate, but one of the fastest methods of identification.
        i. Mask out field
            1. Determine best RGB Range
        ii. Apply threshold
            1. Determine best threshold
        iii. Get Contours
        iv. Analyze shape/position
        v. Player guess
    b. CNN - Running samples of a frame through a convolutional neural network is more compute intensive, but generally yields a higher accuracy. To get started it is essential to get a good dataset, which we have to create ourselves. We determined the best process to do this was to create our own application that allows us to click-to-extract picture samples while simultaneously giving them a label.
        i. Get Data
            1. Create Data Collection Application
                a. Determine best size of each data sample
                b. Determine method of collection

        i. Auto-Collect
        ii. Manual Collect
            1. Left/Right Click for label
    c. Implement Navigation around clips
    d. Add visual cues to help user
        i. Selection area
        ii. Key/Legend
    e. Add error-proof features
        i. Overwriting files
        ii. Crashing at end of clips
        iii. Wrong label selection
2. Create Data Aggregation Application - When taking datasets from various video clips, it is important that we can aggregate them together to make it easier to feed to the CNN for training.
    a. Take data from multiple clips and combine into one file
        i. Make sure all is still in order
3. Build CNN
    a. Choose model
    b. Modify to fit data size
    c. Implement system to review model performance
        i. File selection
        ii. GUI
        iii. Navigation
    d. Graph creation
4. Train model
    a. Determine best parameters
    b. Run and iterate

3. **Apply identification to frames of footage -** Once the identification process is solidified, we have to plug it into the code that runs the video frame by frame. We don't always want to identify the players in every single frame because that is compute intensive, instead, it is good to find a middle ground that strikes a balance.
    a. Determine frequency of identification

## Focus on Action and Waste Little Space for Viewer

This task was a large focus at the beginning of our semester, but has since been shifted to second priority. With this task, the idea is to frame the action of the game in the most appealing and engaging way possible - with the action taking up almost all of the frame.

1. Determine method for keeping action on screen - There are many ways to attempt to catch all the action, but the two most sensible options were Center of Mass (CoM) and Highest/Lowest player bounding. Center of mass proved to be the most reliable with our current level of detection.
    a. Highest/Lowest Player on Screen
    b. Center of Mass
        i. Find solution that ensures outliers from CoM remain on screen
            1. Exponentially favoring lower players on screen by assigning them slightly higher weight (y-position ^ 1.05 by default)
        ii. Keep center of mass from changing drastically each frame
            1. Update CoM every *n* frames by taking average over *n* frames
2. Implement field bounds on identification - It is a waste of compute resources to be identifying objects off the field of play, so we needed a way to eliminate the computations outside of the field of play. Creating a quadrilateral shape over the field and comparing pixel colors became the solution of choice.
    a. Create quadrilateral outline around field
        i. Using mouse input
            1. User-proofing by analyzing position of points and sorting
    b. Determine if contours inside of outline
        i. Create new hidden reference frame of same frame-size with green pixels representing field
        ii. Compare x,y position of contours on main-frame to x,y position in reference-frame and check if reference frame pixel is green
    c. Only start identification once field boundary is set and only identify within boundary
3. Split screen into grid to assign Pan/Tilt/Zoom movements to each section
    a. Calculate center of mass and call appropriate movement when CoM is in certain grid sections
        i. Bottom Third
            1. Zoom Out
        ii. Middle Third
            1. Properly Framed
        iii. Top Third
            1. Zoom In

b. Determine level of movement that still allows for smooth movement and least distortion as possible

*Create Simple Way to Edit Footage / Have Footage Quickly Processed*

This task is to let the user send a video through an application that will enable them to quickly make edits and PTZ through sports footage in a fraction of time that it would typically take on standard software. The preferred method of input for this editing is a video game controller.

1. Import Video
2. Create Application to create Edit-Decision-Lists (EDL files) - The application works by logging each edit with the controller via text in a JSON file (EDL). This EDL is then read by the program and the new video is computed using the edits from the EDL.
    a. Determine Input
        i. Mouse + Keyboard
        ii. Controller
            1. Determine which type of controller
            2. Map buttons to inputs
                a. Pan/Tilt/Zoom
                b. Cut
                c. Important Event
    b. Create Key/Legend GUI to help with navigation
    c. Convert inputs to JSON format
        i. Determine best structure of JSON
    d. Export JSON
3. Send EDL and Video File to be Processed - Locally computing the final video with the EDL file is fairly slow, so hardware optimized for this work will greatly accelerate the process.
    a. Locally

## Easy Way to Record Stats

Through interviews, our team found that coaches highly value being able to easily maintain a log of statistics. The thought for this task is to create a GUI (a concept for future mobile app) that allows coaches to easily track the stats while streamlining the process greatly from a pencil and clipboard. The GUI/stats will integrate with the EDLs

from the previous task to greatly increase the speed of which the coaches are able to post-process their video.

1. Create GUI with Stat tracking features - The GUI will have minimal buttons but allow coaches to quickly log stats and game stats for reference after-the-fact. All GUI inputs will be saved into the same EDL file-format.
    a. Clicks on GUI register in EDL
    b. GUI buttons will track
        i. Stoppages
        ii. Player Stats
        iii. Score
    c. Determine how each record should affect editing
        i. Show Text
        ii. Cut Video
    d. Give option to export Stat-Tracking EDL or combine with Positioning EDL later
        i. EDL from GUI can be standalone, or integrate with post-process EDL
            1. Must sort combined EDL if integrating
    e. Keep list of player stats and timestamps when going through EDL
2. Create additional GUI that displays timestamps of selected players for easy location of highlights - An additional program will be created that will allow players and coaches to select their names and see the timestamps and videos where they make a notable play (according to the statistics entered by the coach)

## Future Plans

Despite the Spring 2021 semester coming to a close, there are still plans for future iterations of this project. One of the biggest things we would like to improve upon is streamlining the overall post-processing operation. If you followed the User Manual, or took the time to check out the demo video, you would likely notice that there are a lot of moving parts that the user needs to account for when operating our software. There are various different scripts as well as multiple directories the user must sift through to complete the execution of the program. This is something we are hoping to improve upon in the future. We would like the experience to only require the execution of a single script, possibly even all through a direct GUI.

Another improvement would be to get the integration of the neural network into the auto tracking programs to a functional level. The auto tracking in place right now gets us to about 70-75% accuracy. Including the machine learning algorithm brings that accuracy up to around 90%, however, it comes at a great cost to overall performance. We are hoping that a solution will be found to successfully integrate the two pieces to allow for better tracking and subsequently better output videos. As mentioned earlier in this report, the team is hopeful to use AWS as a platform to achieve this integration, however due to the high cost associated with this move, the idea has been pushed off to a later date.

An additional aspect of the project that needs improvement is the code organization and documentation. Each Python script is documented well enough to understand the flow of each program and provides a good idea of the thought process when developing each script. The current flaw is the lack of documentation relating all of the scripts together, as this is not something that we have made clear through a centralized source. Right now, the best instructions for operating the applications will be this report. This is something we are looking to tackle in the very near future. We are hoping by doing this we will promote the increase in the project's lifespan as it will be more likely another group will pick up right where we left off. Another element along the lines of code documentation is prettifying our GUI. Right now, the GUI functions more as a proof of concept. It is not the prettiest thing in the world and we would like to improve upon its features and aesthetics before pushing it out as a product.

Looking towards the evolution of the proof of concept GUI, we would like to turn it into a companion app for our application that runs on mobile platforms. The vision for this app would allow users and coaches the freedom to keep stats and control the recording and cutting software from anywhere near the field. We feel that this would dramatically improve the desirability of our product since almost everyone nowadays has some form of smartphone. This plan is very far off in the future as there are many steps needed before we can even begin to discuss this as a possibility.

## Lessons Learned

As with any project, it is important to reflect on some of the hardships you face along the way, for it is these hardships that teach you something and allow you to grow. One of the most important things we learned was the importance of documentation. There were several instances over the course of the semester where we returned to some code we had written weeks prior. In those moments it often took us a little longer than it should have to fully grasp what and how we were doing something. Had we

taken the little extra time to document our code and explain how it works, we would have saved ourselves more time in the long run.

Branching off of documentation, we also learned the importance of code organization. There were a few moments of miscommunication that led to some messy code and duplicate files. This ended up creating some problems for us as we lost the ability to fully grasp all the code we had written. Due to this, we ended up spending an entire day rewriting much of the code we had already written. Doing this was a necessary step as it allowed us to clean up our code and better understand how it truly functions, but it did result in a lost day.

An additional lesson learned, is that video manipulation involves a lot more math than we originally realized. We had to learn a little bit about video aspect ratios, resizing methods, and how to best manipulate frames in order to complete this project. Of course there are many external libraries containing helper functions to aid in this process, but nonetheless, we still had to do a lot more math than we anticipated. This mainly came into play when it came to the PTZ actions taking place over the top of an image.

The last, and probably the most important lesson learned was the ability to adapt at a moment's notice. About half way through the project, a major shift occurred. As mentioned earlier on in this report, the focus for the project was shifted from live streaming and tracking to a more post-processing and game clip editing approach. Despite being able to still use some of the code previously developed, we still had to quickly shift gears to meet the needs of this new target goal. Despite being a largely academically driven project, the work completed and lessons learned will certainly prove to be extremely beneficial in the real world.

## Acknowledgements

The ABC Drone Capstone team would like to thank Charles Kerr for all of his help throughout the course of this project. He proved to be invaluable for his insight and knowledge of the non-technical world surrounding this project. Charles was a great leader and mentor and everyone involved would like to thank him for all his hard work.

We would also like to thank Dr. Fox and all of the supporting TA’s as a part of the Multimedia, Hypertext, and Information Access Capstone course. Their support and willingness to help will forever be appreciated. We wish them all the best of luck and hope they consider this project as an iterative opportunity for students in the future.

## References


1. A. Rosebrock, “Keras Conv2D and Convolutional Layers,” *PyImageSearch*, 31-Dec-2018. [Online]. Available: https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/. [Accessed: 10-May-2021].

2. A. Rosebrock, “OpenCV Track Object Movement,” *PyImageSearch*, 21-Sep-2015. [Online]. Available: https://www.pyimagesearch.com/2015/09/21/opencv-track-object-movement/. [Accessed: 10-May-2021].

3. B. Swanson, “Youth Sports Participation By the Numbers,” *ACTIVEkids*, 29-Jun-2017. [Online]. Available: https://www.activekids.com/football/articles/youth-sports-participation-by-the-numbers?page=2. [Accessed: 10-Apr-2021].

4. C. Bartal, “ABC Drone Software Team Post-Processing Demo,” *YouTube*, 28-Apr-2021. [Online]. Available: https://youtu.be/05lisTxoJLg. [Accessed: 13-May-2021].

5. The Coding Bug, “Build and Install OpenCV With CUDA GPU Support on Windows 10 | OpenCV 4.5.1 | 2021,” *YouTube*, 21-Jan-2021. [Online]. Available: https://www.youtube.com/watch?v=YsmhKar8oOc&list=PLOKaL5oBTZ2Ur86dcE7qLfjXK3qnpyuMQ&index=18&t=5s. [Accessed: 10-May-2021].

6. Computer Science, “Classify Images Using Python & Machine Learning,” *YouTube*, 05-Jun-2020. [Online]. Available: https://www.youtube.com/watch?v=iGWbqhdjf2s&list=PLOKaL5oBTZ2Ur86dcE7qLfjXK3qnpyuMQ&index=12. [Accessed: 10-May-2021].

7. Computerphile, “CNN: Convolutional Neural Networks Explained - Computerphile,” *YouTube*, 20-May-2016. [Online]. Available: https://www.youtube.com/watch?v=py5byOOHZM8&list=PLOKaL5oBTZ2Ur86dcE7qLfjXK3qnpyuMQ&index=15. [Accessed: 10-May-2021].

8. KananVyas, “KananVyas/PlayerDetection,” *GitHub*, 21-Jul-2018. [Online]. Available: https://github.com/KananVyas/PlayerDetection. [Accessed: 10-May-2021].

9. “Machine learning,” *Wikipedia*, 10-May-2021. [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning. [Accessed: 13-May-2021].

10. Mhproductionhouse, “Detecting Clicks on Image [7] | OpenCV Python Tutorials for Beginners 2020,” *YouTube*, 20-Oct-2019. [Online]. Available: https://www.youtube.com/watch?v=DaQoorJQSZQ&list=PLOKaL5oBTZ2Ur86dcE7qLfjXK3qnpyuMQ&index=6&t=132s. [Accessed: 10-May-2021].

11. “OBS Studio,” *OBS*, 2012. [Online]. Available: https://obsproject.com/. [Accessed: 29-Apr-2021].

12. *OpenCV*, 26-Feb-2021. [Online]. Available: https://opencv.org/. [Accessed: 29-Apr-2021].

13. Prosti, “Training Loss and Validation Loss in Deep Learning,” *Stack Overflow*, 02-Jun-2019. [Online]. Available: https://stackoverflow.com/questions/48226086/training-loss-and-validation-loss-in-deep-learning. [Accessed: 10-May-2021].

14. T. Gautam, “Image Classification in Python with Keras: Image Classification,” *Analytics Vidhya*, 16-Oct-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/create-image-classification-model-python-keras/. [Accessed: 10-May-2021].