# Cameraman

Jae Song, Bhavya Jain, Thomas Li

Department of Electrical and Computer Engineering, Carnegie Mellon University

**Abstract—**Current method of broadcasting a live car race requires the labor of cameramen as well as of the production team. Beyond labor, it introduces an aspect of danger for cameramen near the track as well as possibilities of error due to human tracking and latency of feed switching. *Cameraman* is our approach to supplement, or even replace, this current form of live stream production, with the development of auto-tracking cameras and a feed selection algorithm**.**

***Index Terms*—**camera; car; livestream; OpenCV; track; tracking

## I. Introduction

Sports are a common and popular form of entertainment. That said, entertainment requires an audience, and for sports specifically, the medium between sports and sports audience is live streaming. This is especially true for racing sports. To capture footage of a race, multiple cameras along with cameramen are stationed at various locations around the track. These footage then get sent to the media center, where they determine the best footage of the race at a given moment and switch the main stream to that footage like circuit switching. However, there are certain challenges of this current approach. Cameramen are usually located near the track and swivel their camera as cars pass by, which could lead to accidental collisions with the car. Additionally, this requires a sizable production team of cameramen as well as operators.

*Cameraman* is our effort to approach this gap. *Cameraman* is a system of cameras that provides an autonomous live stream of a toy car racing around a track. Each camera will be attached to motorized stands and will provide an auto-tracked camera feed which the system receives and algorithmically determines and switches the live stream to the best feed at a given time. This project, when scaled up, will help the production team of a racing event as it reduces the labor, danger, and errors due to human involvement. It could also allow for better footage to be taken as auto-tracking cameras could be placed in locations that are closer to the action.

Our competing technology is the system that is currently in place, which consists of the production team of cameras, cameramen, and system operators. The advantages of our approach, as said before, is the removal of humans from dangerous filming locations and from camera and system labor that could be made autonomous. The goal of our project is to enable our system of auto-tracking of cars and auto-generation of live streams in a scaled down version of racing sports in the form of toy car racing.

‘

## II. Use-Case Requirements

To meet this general use case, we drafted several use-case requirements. First, the bottomline capability our system must have is the ability to track a car around the race track. The cameras placed around the track must be able detect and locate the car within its frames while the car travels around the track. Therefore, our first use-case requirement is that our system must be able to track a car traveling at 1.5m/s 2.5m/s for at least 95% of a lap. This speed range was determined by dividing the distance of a lap, 5.6m, by the time it took the car to travel one lap at the slowest setting and the fastest setting. The 5% leeway was determined because there are few areas in the track consisting of turns or obstacles that make it hard for cameras to see the car clearly.

A second use-case requirement for our system is that tracking our object car should not be distracted by other cars in the race. In other words, our system should not start tracking a different car in the middle of the race. This requirement should be met 100% of the time on the main stream for the span of a lap.

Another requirement we must ensure is that the stream should not lag too far behind the real time. Most sports production experts agree that 8 seconds of broadcasting latency is a safe level to reach. However, because our race is smaller in scale and our system is less complex, we will be striving for a faster latency. Therefore, our system will strive for a camera to stream latency of less than 500ms.

Our last few requirements are related to the quality of the stream. By nature of sports streaming, the standards of a “good” stream are qualitative. Of course in aspects of latency or even video quality, it is clear that lower latency and higher definition is the standard of a better stream. But in terms of angle of view, distance from track, and basis of feed switching, it is hard to quantitatively define the goal of our project. To make this more tangible, we can break it down into a few requirements. First, the stream must show the front of the car for over 75% of the time. This is because the front view of the car is the most desired and commonly filmed angle in races. However, a little bit of leeway is given because at times, especially at turn, we do want to capture the side/back of the car. Second, the car must be centered in the middle 50% of the screen for more than 75% of a lap. Lastly, the distance of the camera to the car must be as small as possible to capture the closest footage of the car. The constraint of this requirement will be that it must not break the other use-case requirements (ie. most of the track should still be captured between the four cameras).

We took into consideration the public health, safety, welfare as well as global, cultural, social, environmental, and economic factors when formulating our use-case and use-case requirements. First, safety is one of the reasons we are designing this product-for auto-tracking cameras to replace human cameramen, especially in areas of danger (close to the track). Another factor is the social aspect of this project. Since car racing is a form of entertainment, the audience as well as the racers are socially engaged to the sport. Therefore, we had to ensure a good quality of stream, as well as minimal interference with the actual race itself. Lastly, this product may have an impact on the economy of broadcasting and stream production. The autonomous system requires very little human labor to produce streams, so the cameramen and most of the production operators will not be needed as a result.

## III. Architecture and/or Principle of Operation

Our system design can be broken up into three main sections: hardware, computer vision, and feed selection.

### *A. Hardware*

Our assembly of camera stands consists of a camera module attached physically to the rotor of a servo motor. These stands will be placed around the track in optimal locations (furthered in system implementation), and thus require some material to stabilize on, like a wooden plank. The camera on each stand will run through the USB hub to connect to the PC.

We will be using an Arduino as the microcontroller for motor control. The Arduino will receive serial data for motor instruction from the PC via USB, run a motor control program, and control one or more of the servo motors attached on the camera stands.

### *B. Video Processing*

This module will receive the footage from each of the cameras and run object detection and tracking on OpenCV. We will be using data preprocessing to narrow down the search space for detection and tracking algorithms. We will use YOLOv4 as our object detection algorithm to detect the car as well as output its general location on screen and detection confidence level. We will be using GOTURN tracker as our tracking algorithm to track the motion of the car in the feed and output motor control instructions in parallel. It will also interface with the feed selection algorithm by providing the bounding box size.

### *C. Feed Selection*

Our feed selection module takes inputs from the tracking module with information of the tracked bounding box coordinates. The algorithm will determine which feed is the "best" at a given time (correlates to size of the detected car in the feed) and multiplex the output display from the four video sources using the algorithm. We will implement thresholding and hysteresis submodules to prevent switching back and forth of the stream and to make the stream contiguous.

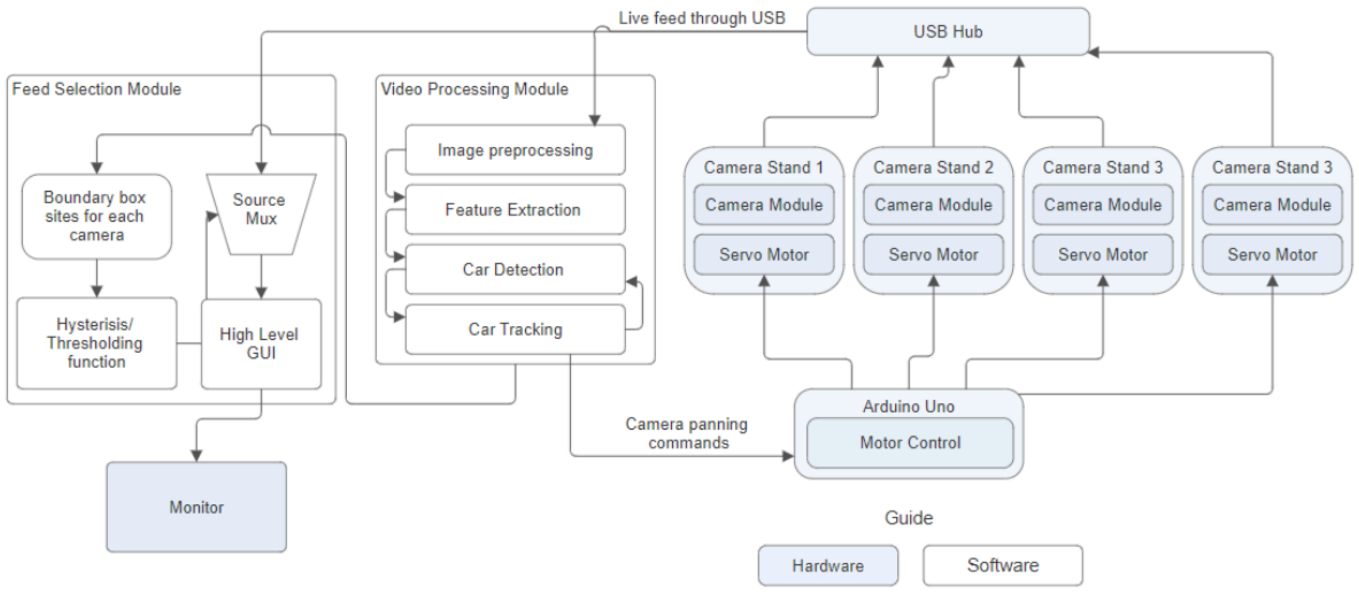

Fig. 1. Block diagram (overall system)
*A closer view can be found in Fig. 3. in the appendix*

## IV. Design Requirements

These design requirements, sectioned by the corresponding use-case requirements, must be met to ensure that our solution meets the 7 use-case requirements and sub-requirements.

*The system must track a car traveling at 1.5 m/s-2.5 m/s for at least 95% of a lap.*

Then an object tracking algorithm must provide control feedback to the motors that turn the cameras producing a rotational speed corresponding to the car's maximum speed of 2.5 m/s. The camera motors must be able to meet a maximum rotation speed of 70 degrees per second to track the turns of the car at 1.5 m/s. The cameras must be placed at a minimum distance from the track such that the camera motors do not exceed this maximum rotation speed.

*The closest possible footage of the car should be captured while meeting 95% coverage of the track.*

Then the cameras must be positioned in a way such that they each have a complementary view of each part of the track that minimizes distance between the camera and the track while providing 95% coverage. A feed selection algorithm must switch the camera source being displayed on the stream when the car moves between different areas of the track. The feed selection algorithm must select the camera feed which has the closest possible shot of the car to display.

*The front of the car must be shown for at least 75% of the lap.*

Then an object detection algorithm must accurately detect the car given real-time input data from the cameras. It must further accurately detect the front of the car, the side of the car, and the back of the car. The feed selection algorithm must switch cameras when the car travels between regions of the track covered by two different cameras, such that less than 25% of the frames from the camera before the switch show the back of the car. This meets the 75% requirement if it holds every time the camera is switched.

*The car must be centered in the middle 50% of the screen for at least 75% of the lap.*

Then an object detection algorithm must accurately bound the car within 10% of its actual location in the frame given real-time input data from the cameras. An object tracking algorithm must provide control feedback to the motors to achieve a position within 10% of the ideal position of the camera. The camera motors must execute positional controls commanded by the microcontroller within 5% of the actual command. This meets the +/-25% margin of error total.

*The end-to-end stream latency should not exceed 500 ms.*

Then the camera must provide the image processing module with an input frame within 100 ms of capture. The image processing module must provide the feed selection algorithm with an input frame within 200 ms of receiving it from a camera. The feed selection algorithm must not delay the communication of an input frame to the display while switching camera sources for streaming for more than 100 ms. The display must display an input frame within 100 ms of receiving it. This meets the 500 ms total.

*The above requirements should be met even with other cars racing alongside it.*

Then the image processing module must correctly detect the car for at least 98% of frames containing the car in each lap regardless of the presence of other cars in the frame to meet all the other requirements.

## V. Design Trade Studies

***Hardware Components***

*A. Servo motor vs stepper motor*

One decision we had to make was whether to use a servo motor or a stepper motor as our camera panning motor. There were many aspects we had to take into consideration before making the decision to use the Beffkipp 9g micro servo motors. First one is the accuracy of the motor. Stepper motors use an open-loop control system, which means that there is no feedback to the system when holding a motor position. Servo motors, on the other hand, use a closed-loop control system, meaning there is constant feedback to adjust and control the motor position. Servo motors are generally used to provide more accurate and reliable positioning. Another aspect was the speed of the motor. Steppers operate in discrete steps and their speed is limited by the rate they can switch from one step to next. However, servos can operate at higher speeds with greater precision because of their closed-loop control systems. Then what is the downside of using servo motors? Servos are generally more expensive than steppers. However, our project does not require a big budget, which means we can spare more money on servos rather than steppers to provide more precise and faster camera panning.

*B. Microcontroller*

Another decision we had to make was choosing a microcontroller for our motor controls. Arduino Uno Rev3 is the microcontroller we decided on for several reasons. First, for our application, we just needed simple control over four servo motors. Therefore, everything Arduino Uno provided in terms of performance, usage, and I/O fulfilled our requirement. Arduino is very cost-effective compared to other microcontrollers and it is widely known for its ease of use. All of the team members have previous experience with Arduino so that was definitely a factor in our decision. Other microcontrollers provide better low-level programming, but again, the high-level programming that Arduino provides is enough for our application.

*C. Racetrack*

Choosing a racetrack to build our system around was another decision we had to make. We decided to choose the Wupuaait Slot Car Race Track Set because we needed one that would be most similar to an actual car race (ie. no overhead loops, not too many turns). This track offers a base configuration that is large enough to build a system around (~0.5m x ~1m). It is also configurable, meaning that we can reconfigure the track and test our system on different configurations, which is a reach goal. The speed that the cars hit on this track was 1.5m/s - 2.5m/s, which is fast for a toy car, but not too fast that our motors cannot catch up to its speed.

*D. Camera and USB hub*

Another consideration was choosing the camera and deciding to use a USB hub. First, we needed cameras that are attachable to our micro servo motors, which means it would likely have to be a camera module that is < 2in in length and width. We also needed the protocol to be USB since we needed to connect them to our PC. Additionally, we needed the camera cable to run at least 0.5 meters from the camera to the hub. Therefore, we decided on the Arducam 1080p USB camera for computers. The camera more than fulfills our performance requirements in terms of quality (1080p) and size (1.5in x 1.5in, 1 meter cable). But our only worry was the price point of $34.99 per camera. However, looking at our budget, we had room to spend on these cameras in order to gain good quality footage, as well as ease of interfacing to our PC.

Because our PC does not have four USB-A ports, we needed to purchase a USB hub. We chose to get the Syntech USB-A to USB-C hub. This takes in four USB-A ports and has one USB-C(USB-A with an adapter) output port. Each port supports up to 480Mbps, which is enough for our cameras to send footage - an average footage requires 5-15Mbps. We also needed the cable of this hub to be at least 0.5 meters to connect the hub located in the middle of the track to the PC located outside the track, and the cable of the Syntech hub is 2 feet(~0.6 meters) in length.

***Software Components***

*A. OpenCV*

In our project, the utilization of OpenCV is highly advantageous due to its extensive set of computer vision functionalities tailored to our specific requirements. OpenCV offers a comprehensive suite of image processing and computer vision algorithms, making it an indispensable tool for tasks such as image preprocessing, feature extraction, and object tracking. Its rich collection of functions enables us to efficiently implement various components of our system, including noise reduction, blurring, contrast enhancement, and color-based segmentation. Moreover, OpenCV's compatibility with diverse camera hardware and video formats ensures seamless integration with our camera setup, allowing us to capture, process, and analyze video footage with ease. Additionally, OpenCV provides robust support for real-time processing, enabling our system to achieve low latency and high performance even under demanding conditions. Furthermore, its extensive documentation and community support facilitate rapid development and troubleshooting, streamlining the implementation process and enhancing the overall efficiency of our project. In summary, OpenCV serves as a versatile and powerful toolset that significantly enhances the capabilities of our system, making it the ideal choice for our computer vision tasks.

*B. YOLOv4 for detection*

In our project, the choice of YOLOv4 for car detection is highly appropriate compared to other detection methods due to its exceptional balance of speed, accuracy, and efficiency. YOLOv4 (You Only Look Once version 4) is a state-of-the-art object detection algorithm known for its real-time performance and robustness. Unlike traditional methods that rely on region proposal networks (RPNs) and multiple stages of processing, YOLOv4 adopts a single-stage architecture, enabling faster inference and lower latency. This makes it ideal for our system, where quick detection of the car is paramount, especially considering the high speeds at which the car may be moving on the track. Furthermore, YOLOv4 offers excellent integration capabilities, allowing seamless integration with our tracking and camera control algorithms. Its unified architecture simplifies the deployment process and facilitates efficient resource utilization. Additionally, YOLOv4 demonstrates superior performance in terms of accuracy, thanks to advancements in model architecture and training techniques. This ensures reliable detection of the car under various conditions, including changes in lighting, weather, and background clutter. Compared to other detection methods such as Faster R-CNN or SSD (Single Shot MultiBox Detector), YOLOv4 stands out for its superior speed-accuracy trade-off, making it the most suitable choice for our project where quick detection, low latency, and good integration are paramount.

*3. GOTURN (Generic Object Tracking Using Regression Networks) for tracking*

In our project, the selection of GOTURN (Generic Object Tracking Using Regression Networks) for car tracking is highly appropriate considering its unique capabilities tailored to our specific requirements. GOTURN is a deep learning-based tracking algorithm designed to track objects across frames in a video sequence with high accuracy and efficiency. Unlike traditional tracking methods such as correlation filters or optical flow, GOTURN employs a convolutional neural network (CNN) architecture to learn object appearance changes and motion patterns directly from data, making it robust to variations in lighting, speed, and object scale. This is particularly advantageous for our system, where the car may resize and rotate with respect to the camera while navigating the track at high speeds. GOTURN's ability to adapt to such dynamic changes ensures reliable and consistent tracking performance under challenging conditions.

Compared to other tracking algorithms and techniques, GOTURN offers several key advantages tailored to our project's requirements:

- Accuracy: GOTURN leverages deep learning to learn and adapt to the specific appearance variations and motion patterns of cars on the track. This results in more accurate and robust tracking performance, crucial for ensuring precise localization of the car even amidst changes in lighting conditions, speed, and object scale.
- Robustness: In our dynamic tracking environment, where the car may resize and rotate with respect to the camera while navigating the track at high speeds, GOTURN excels in handling such variations with resilience. Its ability to cope with object scale changes, rotations, and occlusions ensures uninterrupted tracking even in challenging scenarios.
- Speed: Despite its sophisticated deep learning

architecture, GOTURN maintains remarkable computational efficiency, enabling real-time tracking performance. This swift response is essential for our system's requirements, ensuring timely adjustments to the car's position without introducing any noticeable latency.

- Adaptability: GOTURN's learning-based approach allows it to adapt and generalize to different car appearances and track conditions without the need for manual parameter tuning. This adaptability enhances the system's versatility, allowing it to track cars effectively under varying environmental conditions and track configurations.

The utilization of the GOTURN algorithm for object tracking is highly advantageous compared to other tracking algorithms and techniques, such as Kalman filters, particle filters, and optical flow-based methods. While these traditional tracking algorithms have their merits, they often struggle to handle complex scenarios where the tracked object undergoes significant scale and rotation variations, as is the case with our moving car on the track. Kalman filters, for instance, rely on linear dynamical models and assume Gaussian noise distributions, which may not accurately capture the nonlinear dynamics and uncertainties associated with object motion in our dynamic environment. Similarly, particle filters suffer from high computational costs and may struggle to maintain accurate tracking under rapid changes in object appearance or occlusion. Optical flow-based methods, while effective for motion estimation, may struggle with object occlusion and scale changes, leading to tracking drift or loss.

## VI. System Implementation

### ***Hardware***

### *A. Camera stand assembly*

On each camera stand, the components we plan to attach are the Arducam camera module and the micro servo motor. We plan on using screws to physically attach the camera module to an axis perpendicular to the motor that will be attached to the rotor. This is to enable panning of the camera. Now to make the stand stable and stationary, the ends of the servo will be grounded via screws to a heavier object (wooden planks). The simplicity of our camera stand design allows for portability as well as usability, especially with the USB cables and other wires running through them. We will be making four of such camera stands.

### *B. Location of camera stands and communication to PC*

We will be spreading our camera stands around the track to capture the maximum percentage of the track. Finding these optimal locations will take trial and error, but our approach will be to view different camera angles and determine the ones that are comparable to real broadcasted shots from F1 (ex. at the end of straightaways, at turns, etc…). Additionally, there will be a one way, USB 2.0 communication from the camera to the PC to share the footage the camera captures. Therefore, we require four USB ports on the PC. To support this in our PC which does not have four ports, we will use a USB hub with four USB-A input ports and one USB-C (or A using an adapter) output port. The camera USB cables will meet at the center of the track, where the hub will be.

### *C. Communication from PC to Arduino*

We will be using the USB 2.0 protocol as the communication method between the PC and the Arduino. This communication will be solely one way, as the only information that is sent from the PC to the Arduino is regarding the movement of specific motors to guide the tracking. On the PC side, since we are running OpenCV on Python, we will use Python's Serial library to send the following data on motor movement: 1. Which motor? 2. Which direction? 3. How much? (in angles or steps). This transmission will be in parallel with OpenCV calculations on car location relative to the screen. On the Arduino side, we will: 1. wait for data from PC 2. move motor "n" in direction "R" by x (angles or steps) 3. repeat forever. The specific cable we are using is the USB 2.0 A/B cable.

### *D. Communication from Arduino to motors*

The Arduino Uno will be placed in the middle of the track with the USB A/B cable running from it to the PC outside the track. This Arduino will lie on a breadboard and will control all four servos via GPIO. Each servo cable has 3 pins: negative, positive, and signal. The negative pins will be on the GND of Arduino and the positive pins will be on 5V supply powered by the Arduino. The signal pins of the servos will be on the PWM Digital outputs.

We will be using the Servo library installed on Arduino. The main functions we will utilize are attach(), servo.write(), and delay(). Servo.attach(pin) allows us to attach a servo on a specific pin. Servo.write(pos) allows us to write a specific angle of position to that servo. delay(time) allows us to delay time in milliseconds before moving to the next line of code. The gist of our Arduino code will be: 1. setup servos with designated pins 2. loop until data from PC arrives 3. use a combination of servo.write() and delay() to move the motor to a position at a controlled speed.

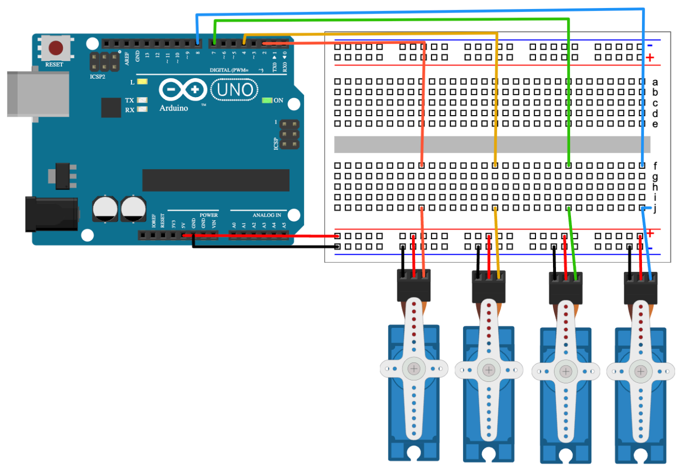

Fig. 2. Arduino to servo connections

***Detection and Tracking Module***

Live feed off of every camera will be relayed to the processing system. The aim here is to track the camera and relay car position information to the camera to allow for smooth panning. Given that the sequence of camera positions is known and since the car must pass through the cameras in order, only the video feed of the current tracking camera and the next camera will be processed. The acquired footage will undergo the following:

*A. Data preprocessing:*

By extracting relevant features, such as the color of the toy car, preprocessing helps narrow down the search space for the detection and tracking algorithms. This focused approach reduces the amount of computation required to analyze each frame, leading to faster processing times and lower latency.

1. Color-based segmentation aims to isolate regions in the image that correspond to the toy cars. Utilizing the HSV color space, the implementation segments the image using a color mask tailored to the target color of the toy cars (e.g., red). Color-based segmentation effectively narrows down the search space for the toy cars, focusing computational resources on regions likely to contain the objects of interest.
2. Smoothing is employed to create a more uniform appearance of the image by reducing abrupt changes in pixel intensity, thereby improving the continuity of the tracked object. A blur filter is applied to the image to achieve smoothing, with the flexibility to adjust the kernel size based on the desired level of smoothing.
3. Contrast Limited Adaptive Histogram Equalization (CLAHE) is applied to the L channel of the LAB color space, enabling localized contrast enhancement while preserving image details.
4. Noise reduction aims to mitigate the adverse effects of unwanted signal fluctuations in the input frames, which can obscure the true features of the toy cars and impede accurate tracking. This implementation offers various noise reduction methods, including Gaussian blur, median filtering, and bilateral filtering.

*B. Object detection:*

The input to the object detection module consists of preprocessed images with the toy cars highlighted in red. This preprocessing step ensures that the objects of interest are distinctly delineated, enhancing the YOLOv4 algorithm's ability to detect them accurately. The YOLOv4 model is seamlessly integrated into the system using the TensorFlow deep learning framework. Upon receiving the preprocessed images, the YOLOv4 algorithm is executed to detect toy cars within the camera frames. The algorithm outputs bounding box coordinates and confidence scores for the detected objects, providing essential spatial and confidence information for subsequent processing. To ensure robust detection, a confidence threshold is applied to the confidence scores associated with each detected object. Detections with confidence scores below a predetermined threshold are filtered out, effectively reducing false positives and enhancing the overall accuracy of the detection process. Following confidence thresholding, non-maximum suppression (NMS) is applied to eliminate redundant bounding boxes and retain only the most confident detections. This post-processing step further refines the detection results, ensuring that only the most salient object detections are considered for further processing.

*C. Object tracking:*

Upon receiving the bounding box coordinates of the detected toy car from the object detection module, the object tracking algorithm initiates tracking using the GOTURN tracker. The bounding box serves as the initial region of interest (ROI) for the tracker, providing spatial context for the location of the toy car within the camera frame. As the toy car moves within the camera's field of view, the GOTURN tracker continuously updates the position of the bounding box to maintain tracking. The tracker adjusts the bounding box coordinates based on changes in the object's position and appearance, ensuring accurate and robust tracking performance. Once the toy car enters the middle of the camera frame, the tracking algorithm provides movement information to adjust the camera's orientation and preserve the car in the center of the frame. This involves determining the direction and magnitude of camera movement based on the position of the car within the frame. The camera's movement is constrained within predefined limits to prevent excessive panning and maintain a smooth tracking experience. Upon reaching the pan limit or when the next camera starts tracking the toy car, the tracking algorithm initiates a handover process to seamlessly transition tracking responsibilities to the next camera in the sequence.

***Feed Selection Module***

The feed selection module will accept as input the boundary box dimensions produced by the video processing module in its analysis of frames captured by each camera. It will also accept a corresponding classification from that module on whether each of these frames contains a front view, a side view, or a back view of the car. Additionally, it will accept the raw frames produced by each camera and forward a subset of these frames to the monitor to be displayed through the High-Level GUI library.

*A. Feed comparison*

For each set of frames from a single time-step, we will determine which camera has the closest possible shot of the car from the boundary box dimensions produced by the video processing module. We will take into account whether the camera sees the front, the side, or the back of the car, and assume that within each category, the largest bounding box corresponds to the closest possible shot of the car. We will determine the best camera view as the front view with the largest bounding box and the worst view as the back view with the smallest bounding box, or no box if car is undetectable, and everything in between.

*B. Hysteresis*

From frame to frame, keep track of which camera has the best view of the car. If a camera maintains continuously the best view of a car for a set threshold number of frames (approximately 3 frames, 100ms for 30fps), we will switch from the previously selected source to this camera as the new source.

*C. Source forwarding*

We will display the frames from the selected source on the monitor by repeatedly calling High-Level GUI library functions on those frames. As soon as the selected source changes, we will switch to displaying the frames from the new source.

## VII. Test, Verification and Validation

***Use-Case Tests***

*Tests for Use-Case #1: Ensure the system can track a car traveling at 1.5m/s-2.5m/s for at least 95% of a lap.*

Record video footage of the toy car moving at different speeds within the specified range. Analyze the tracked trajectory to verify the system's tracking capability. There are 4 main track turns that are seen on major circuits that will be analyzed. The each correspond to different levels of camera panning with different speeds of tracking: 1. straightaways 2. sweepers (long and gradual turn) 3. hairpin turns (the tightest possible turn) 4. chicanes (slightly S shape straightaways.

*Tests for Use-Case #2: Verify that tracking the object car is not distracted by other cars in the race.*

Our track includes two slots. Introduce another car to the other lane and run the race at the same time. In the span of a lap, analyze the stream and calculate the percentage of the stream, if any, is tracking the other unwanted car instead of the main car. This percentage should be 0%, 2% at maximum.

*Tests for Use-Case #3: Achieve a camera-to-stream latency of less than 500ms.*

This requirement deals with the latency from real-time race that the camera captures to the footage that appears on the stream. To test this requirement, we plan to use the starting line as a marker for the car. When the car in the stream passes through the marker, we can signal the camera by throwing a flag to the screen. Now we can replay the footage on the stream on slow motion and check the latency from when the car passes the marker to when the flag is thrown. This latency will correspond to the camera to stream latency, and should be under 500ms.

*Tests for Use-Case #4: Ensure good quality of stream*

Ensure good quality of the stream with the front of the car visible for over 75% of the time and ensure the car remains centered (middle 50% area of the screen) for more than 75 percent of a lap. Analyze the position of the car within the frame throughout the lap and verify compliance with the centering requirement. An easily identifiable marker will be placed in the front of the car. Gathering the race footage we can measure the amount of time the front of the car was visible on screen. Here car front view time/front detectable time and car in center of frame time/car in detectable zone time should be more than 75 percent. This is in accordance with car racing streaming standards that prefer to show the fronts of the cars and keep the cars centered in the shot.

***Additional Tests***

*Detection to motor latency Test:*

This latency is crucial for ensuring real-time responsiveness in tracking the car's movement. The latency measurement encompasses the time it takes for the camera to pan correctly in response to the detected movement of the car, as determined by the GOTURN algorithm. When the algorithm detects that the car is moving out of the specified region of interest, it signals the camera to adjust its position accordingly to keep the car within the frame. The camera's pan speed is critical in maintaining the car within the area of interest and minimizing any lag between the detection and the camera's response. To measure latency, we utilize the time module in Python, which allows precise timing of events. The process involves time-stamping the moment when the algorithm detects the car's movement and time-stamping the moment when the camera completes the pan adjustment. The difference between these two timestamps represents the latency between the detection and the camera's response. By repeating this process for multiple instances of car movement detection, we gather a dataset of latency measurements. We are hoping for a latency of 5ms, a bit of room since the camera may lag behind the moving car as long as it keeps it in frame.

*Scalability Test:*

The scalability test evaluates the system's ability to handle the addition or removal of cameras placed anywhere on the track while maintaining optimal performance and functionality. We will test how the system performs with different levels of separation between the two cameras - whether they are stuck together or on opposite parts of the track. The system should also handle blind spots in the track, if too few cameras are placed, without breaking the stream.

## VIII. Project Management

*A. Schedule*

Our weekly schedule is provided in detail in Fig. 4. of the Appendix.

*B. Team Member Responsibilities*

For the scope of the project, we decided to split the responsibilities into three sections of hardware, computer vision, and feed selection algorithm. Jae's primary responsibility will be to take charge of the hardware tasks of this project, including camera stand assembly, determination of camera stand locations, servo motor control, and Arduino programming. He will also take the lead in interfacing Python with Arduino to allow communication from computer vision to the Arduino for motor control. His secondary responsibility will be to take initiative in integration and testing approaches after finishing hardware tasks, as others may need more time to refine their parts. Bhavya's primary responsibility is car detection and tracking using OpenCV. His tasks include detection of cars using libraries, tracking of cars and sending location information to Arduino, real-time multithread car tracking for four cameras, and sending relevant information for feed switching to the feed selection module. His secondary responsibility will be updating our block diagram and schedule, as well as communicating with Thomas on the interface between OpenCV and feed selection. Lastly, Thomas' primary responsibility is feed selection algorithm development. He will be responsible for designing an algorithm to reasonably switch feeds between the four cameras to provide the best stream. He will communicate with Bhavya to attain the data points he needs to make these feed switches. His secondary responsibility will be to communicate to Jae and Bhavya continuously on the things that can be done on the hardware and computer vision part to improve the quality of the stream.

### *C. Bill of Materials and Budget*

Bill of Materials and Budget is broken down in Table I of the Appendix.

### *D. Risk Mitigation Plans*

One of the risks we may encounter is that the race car moves too fast around the track for our system to continuously track. This could occur if the motors are too slow or if the feedback control incurs too much processing time. In order to account for this risk, we will plan levels of distances for the cameras from the track so that we can move the cameras farther away if the close-distance configuration is not possible, resulting in lower motor rotational speed requirements and motor feedback processing time requirements.

Another risk we may encounter is that one of the cameras detects the race car on the wrong side of the track and the system erroneously switches to that camera to follow the race car. In this case, because a real racetrack would involve longer distances and obstructions and this risk is due to unique toy racetrack conditions, we plan on building obstructions around the track to simulate real racetrack obstructions and prevent the cameras from detecting the car from the other side of the track.

Another risk we may encounter is that the camera motors stutter during tracking due to unforeseen mechanical or control challenges with our current implementation plan. In order to account for this risk, we plan on researching a PID motor control module that could be implemented if necessary to ensure the motor tracks properly.

## IX. Reach Goals

Beyond our MVP, we have reach goals that we will strive for if time allows. First is our system's flexibility to different camera stand locations. While our specifications will work with stationary camera stands at specific locations around the track, our goal is to remove the dependency of our system on knowing where the camera stands are around the track. This will allow for different angles of footage and therefore enhance the stream controllability. Another reach goal is to accommodate for different track configurations. Similar to the previous goal, since our track is configurable, if time allows, we would like to remove the dependency of our system on knowing what the track is. This will increase the usability of our product. Lastly, we would also like to take the zoom of the camera and incorporate it into our footage. For example, when a car is far away, our system would be able to zoom into that car, and zoom out as the car approaches the camera. This is how cars are captured in real racing, and we look to implement it into our system if time allows.

## X. Related Work

We have identified currently existing products on the market which have a similar purpose and nature to our product, though with slightly different applications and technical challenges. One of these products is Streamline - United Sports Services. This product, similar to ours, seeks to deliver a real-time livestream of a race with multiple camera angles and autonomous camera switching. They use a computer, cameras, and cabling, like us. They differ from our project notably by focusing on pool championships, in which humans race back and forth along a pool line instead of around a race track which has more variability in geometry. They also differ from our project by using fixed cameras instead of CV-based tracking cameras.

Another one of these products is Multi-Angle Remote Production - PlaySight. This product, similar to ours, seeks to deliver multi-angle production of a livestream with smooth camera switching. They differ from our project notably by not having autonomous switching between multiple camera angles, though they do have multiple camera angles and CV-based camera tracking separately. They also focus on sports with human players, while we focus on race cars on a race track which involves higher speeds and longer distances to be covered.

## XI. Summary

Overall, our system fills in the gap of human labor, error, and danger in the production of live streams by developing auto-tracking cameras as well as generating a live stream autonomously. The danger of cameramen having to stay near the track to capture good footage is no longer a factor in our system. Additionally, the labor and possible errors that come with the production of the stream using these footages are also mitigated by our system. Some challenges that are within discussions include the system's ability to track the car at high speeds, quantifying and improving the quality of stream (feed switching aspect), and our reach goal of achieving flexibility to different camera stand locations as well as different track configurations. Although they are challenges, we hope to overcome them in order to create a useful and impressive produce, as well as to enjoy the process of this project.

## Glossary of Acronyms

| | |
|---|---|
| CLAHE | Contrast Limited Adaptive Histogram Equalization |
| CV | Computer Vision |
| F1 | Formula One |
| GND | Ground |
| GOTURN | Generic Object Tracking Using Regression Networks |
| GPIO | General Purpose Input/Output |
| HSV | Hue, Saturation, Value |
| I/O | Input/Output |
| LAB | Lightness, a coloring, b coloring |
| MVP | Minimum Viable Product |
| NMS | Non-maximum Suppression |
| PC | Personal Computer |

PID - Proportional-Integral-Derivative
PWM - Pulse Width Modulation
ROI - Region of Interest
USB - Universal Serial Bus
YOLOv4 - You Only Look Once version 4

## References

1. J. Tang, “The choice between servo motors and stepper motors,” *Orientalmotor.com*, 01-Dec-2021. .
2. “OpenCV: Getting started with videos,” *Opencv.org*. [Online]. Available: https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html. [Accessed: 02-Mar-2024].
3. “OpenCV: High-level GUI,” *Opencv.org*. [Online]. Available: https://docs.opencv.org/3.4/d7/dfc/group__highgui.html. [Accessed: 02-Mar-2024].
4. “OpenCV: Creating Bounding boxes and circles for contours,” *Opencv.org*. [Online]. Available: https://docs.opencv.org/4.x/da/d0c/tutorial_bounding_rects_circles.html. [Accessed: 02-Mar-2024].
5. “Does performance differ between Python or C++ coding of OpenCV?,” *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/13432800/does-performance-differ-between-python-or-c-coding-of-opencv. [Accessed: 02-Mar-2024].
6. M. Murphy, “Streamline,” *United Sports Services*, 12-Oct-2020. [Online]. Available: https://unitedsportsservices.com/services/streamline/. [Accessed: 02-Mar-2024].
7. “Multi-Angle Remote Production,” *playsight*, 05-Jul-2023. [Online]. Available: https://playsight.com/multi-angle-remote-production/. [Accessed: 02-Mar-2024].
8. S. Mallick, “GOTURN : Deep Learning based Object Tracking,” *LearnOpenCV – Learn OpenCV, PyTorch, Keras, Tensorflow with code, & tutorials*, 22-Jul-2018. [Online]. Available: https://learnopencv.com/goturn-deep-learning-based-object-tracking/. [Accessed: 02-Mar-2024].

*Appendix*

Fig. 3: Closer View of Block Diagram (overall system)

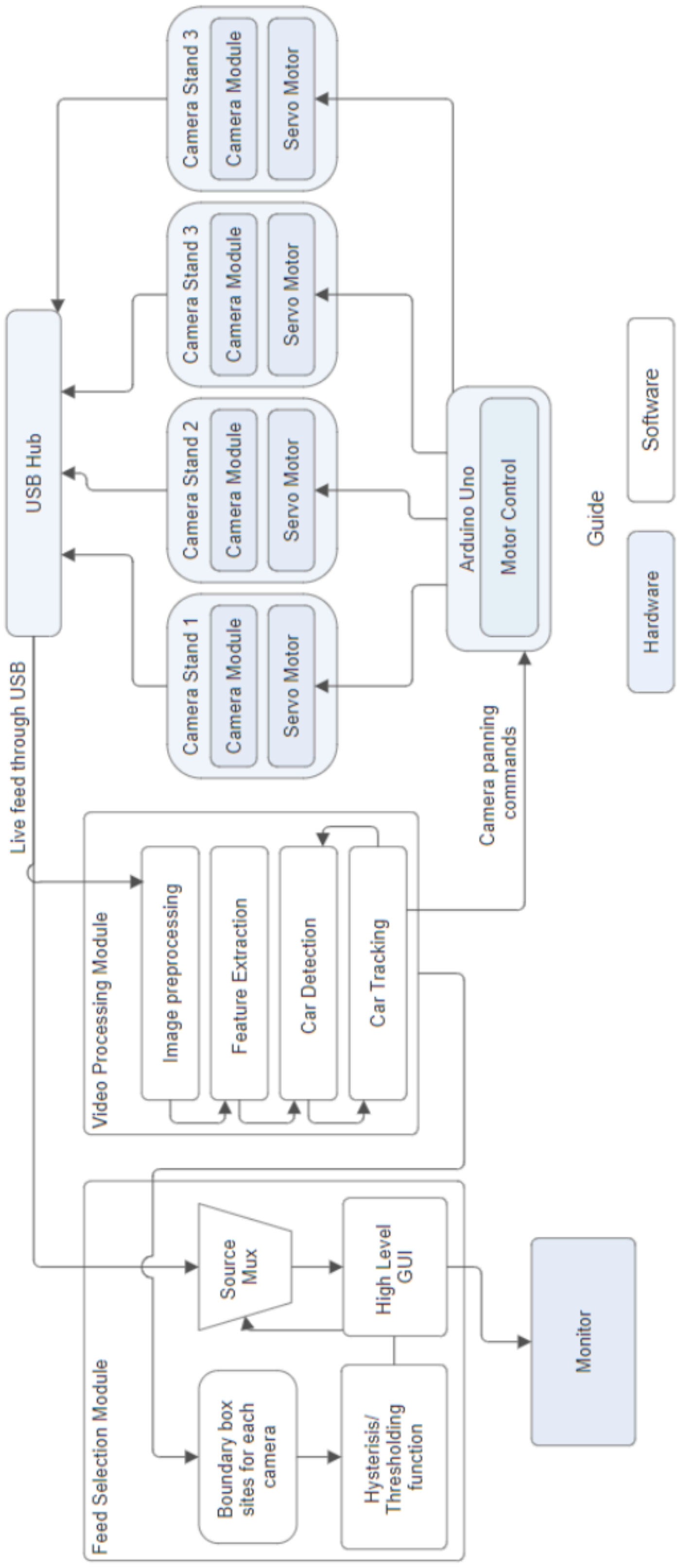

Fig. 4. Schedule (from section VIII.A)

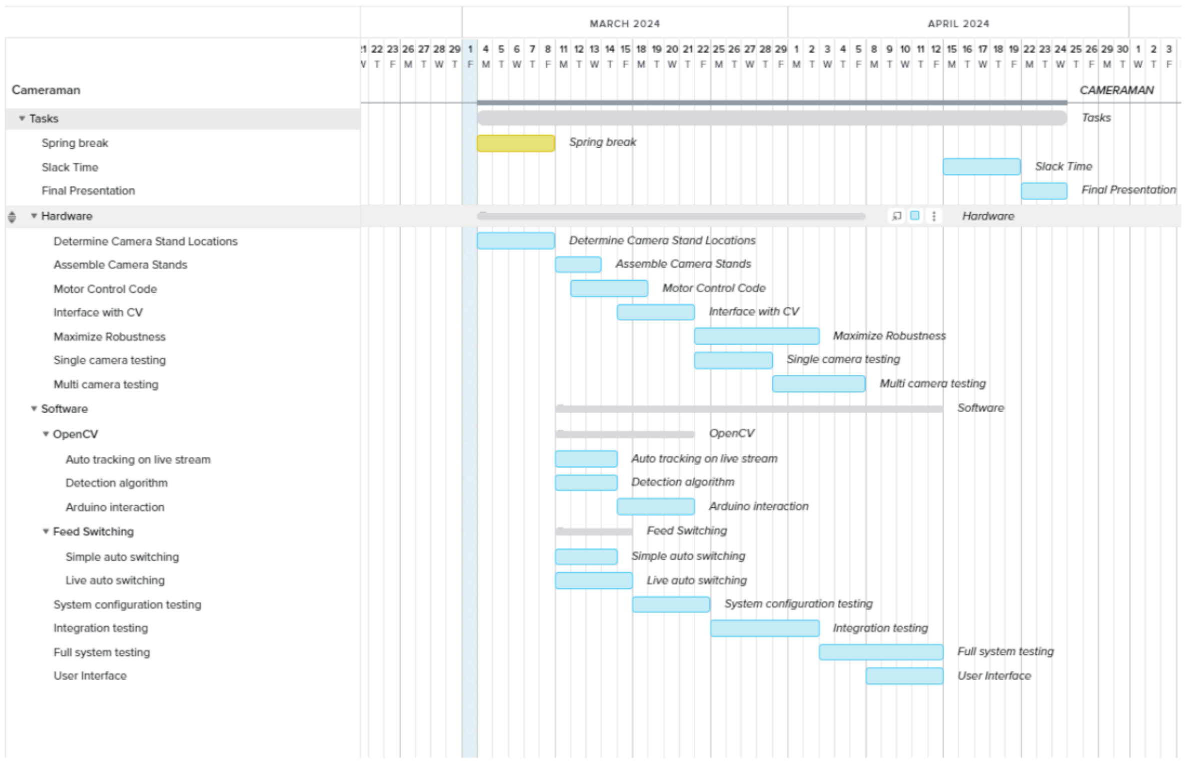

Table I. Bill of Materials (from section VIII.C)

| Description | Model # | Manufacturer | Quantity | Cost (per unit) | Subtotal |
|---|---|---|---|---|---|
| Slot Car Race Track | A64-8A | Wupuaait | 1 | $57.56 | $57.56 |
| Arduino Uno Rev3 | A000066 | Arduino | 1 | $32.61 | $32.61 |
| 4-pc 9g Micro Servos | SG90 | Beffkkip | 1 | $8.68 | $8.68 |
| 3220-Point Solderless Breadboard | 20812 | Jameco | 1 | $0 | $0 |
| USB 2.0 Cable Type A/B | M000006 | Arduino | 1 | $0 | $0 |
| Arducam 1080P USB Camera | B0205 | Arducam | 4 | $38.01 | $152.04 |
| USB C to USB Hub 4 ports (2ft) | M20C | Syntech | 1 | $20.62 | $20.63 |
| | | | | **Total** | $271.52 |