COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

Team 12

Madelyn Swelstad
Grant Dibala
Rachel Cavalier
Caleb Curran-Velasco

Final Revision June 15, 2024

CSCI 370 Summer 2024

Prof. Kathleen Kelly

Table 1: Revision history

Revision	Date	Comments
New	May 19, 2024	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done XIII. Team Profile References Appendix A – Key Terms
2nd	May 26, 2024	Completed Section: VI. System Architecture
3rd	May 30, 2024	Revised Section: Requirements
4th	Jun 2, 2024	Completed: VII. Software Quality and Test
5th	Jun 4, 2024	Completed: VIII. Project Ethical Considerations
6th	Jun 8, 2024	Completed Sections: IX. Project Completion Status X. Future Work
7th	Jun 9, 2024	Completed Sections: XI. Lessons Learned Revised Sections: IX. Project Completion Status
8th	Jun 12, 2024	Completed Sections: XII. Acknowledgements Revised Sections: All
9th	Jun 15, 2024	Revised Sections: VI. System Architecture

Table of Contents

I. Introduction

For this project, our team created a 3D city and traffic simulation in Unity. Within this city scene, there were many objects including buildings, cars, pedestrians, streets, and stop lights. We then placed two cameras above the scene to analyze the movement of vehicles around the scene. The goal of this project is to create software that can identify a robber vehicle and police cars from civilian vehicles and other 'noise', which then tracks the robber vehicle utilizing data collected from the cameras and predict its future movement. Finally, the software deploys police vehicles for interception of the robbers. In conclusion, the project aims to develop a city and traffic simulation in Unity, utilizing well-placed cameras to analyze vehicle movements, with the ultimate objective of implementing software capable of distinguishing and tracking suspect vehicles amidst an urban environment.

Northrop Grumman - a global aerospace, defense, and security company - is facilitating this project. For security reasons, the project's results are not being used in any greater systems within Northrop Grumman; rather, the company is using this project to introduce students to defense technology. Thus, there is no preexisting software to build upon and the software created for the project's duration will not be maintained after completion. If implemented, possible stakeholders for this project might be the Department of Defense, urban defense systems, and civilians.

II. Functional Requirements

- Simulation and Visualization Requirements:
    - The simulation shall utilize Unity for visualizing an urban environment and ground targets.
        - The environment shall be constructed using SimplePoly City's provided textures.
            - The environment shall include 1 robber vehicle.
            - The environment shall include 3 police vehicles.
            - The environment shall contain an arbitrary number of pedestrians (that don't interact with the vehicles/rest of the environment).
            - The environment shall contain an arbitrary number of "civilian vehicles" to make the simulation more realistic.
    - The simulation shall create dynamic scenarios and visualize heists, police interception, and vehicle movement.
    - The environment shall include spawn locations and escape points for hostile vehicles within the simulation environment.
    - The simulation shall be capable of customizing ground vehicles' locations, orientations, and trajectories within the environment.
- Detection Requirements:
    - The detection system shall implement algorithms/models that detect vehicles.
        - These algorithms shall include a Python implementation of a CNN - convolutional neural network.
        - These models shall include a self-trained AI model via Roboflow.
    - The detection system shall be able to distinguish between civilian, robber, and police interceptor vehicles.
    - The system shall implement a pre-trained object detector for efficient detection and classification.
    - The system shall accurately detect and classify vehicles 85% of the time.

- Tracking Requirements:
    - The system shall utilize OpenCV and implement a Python algorithm for real-time tracking of vehicles.
        - The system shall keep track of the state of each vehicle.
            - The system shall keep track of each vehicle’s appearance.
            - The system shall keep track of each vehicle’s velocity.
            - The system shall keep track of each vehicle’s ID number while the vehicle is inside the current camera’s frame.
            - The system shall keep track of each vehicle’s location.
        - The system shall prioritize tracking the robber car once it is identified.
- Prediction and Planning Requirements:
    - The system shall predict the future trajectories and destinations of vehicles.
        - The success rate shall be 85%.
    - The system shall plan interception strategies based on predicted vehicle movements.
- Interception Requirements:
    - The system shall develop interception strategies to capture robber vehicles.
        - The system shall utilize familiar algorithms such as A* and/or Dijkstra’s.
        - The interception shall be successful 85% of the time.
- Testing Plan:
    - The testing shall ensure proper detection of vehicles.
        - The testing shall include accurate classification and differentiation between vehicle types.
        - The testing shall include movement assessment such as attribute maintenance.
        - The identification testing shall yield a success rate of over 85%.
    - The testing shall verify that the robber car can be intercepted 85% of the time using the pathfinding algorithms implemented into the police cars.
    - Testing shall include at least 15 scenarios thoroughly testing each sub-requirement on the list (detection, tracking, prediction, sensors, interception).
        - Detection testing shall include Roboflow tools and graphs to determine how accurately the OpenCV Python script can detect vehicles and their classifications.
        - Tracking testing shall include human verification of correct cars being tracked and that identification numbers are maintained, even after view obstruction.
        - Prediction testing shall include verification that all possible turns that could be made by the robber car are attempted to be covered by police cars within the simulation.
        - Interception testing shall include test trials to see if police cars can catch the robber car.

III. Non-Functional Requirements

- We shall utilize free software when available and keep costs as low as possible when necessary.
- Teammates shall create work to the best of their ability, utilizing available research, technical support, and advisors to guide decisions.
- The team shall communicate frequently with the client such that feedback is continuously received and the project stays on track with the client’s visions and needs.
- The software shall run efficiently on any standard machine, providing timely responses.
- The code shall be well commented and documented.
- The code shall follow best security practices.

- Sensors Requirements:
    - The environment shall simulate virtual cameras for optical sensors with customizable parameters.
        - Each intersection in the environment shall be covered by at least one camera, and each camera shall cover at least one intersection.
    - The system shall utilize OpenCV for real-time processing of sensor data.
    - Each camera shall have the following specifications:
        - The cameras shall have a resolution of 2880x1612.
        - The cameras shall keep track of their location and rotation with reference to the main simulation camera
- Delivery Plan:
    - We shall deliver the product through live demonstrations conducted on-site for the client team and Northrop Grumman associates
    - We shall regularly provide updated demos to showcase progress and gather feedback, maintaining consistent communication with Northrop Grumman.

IV. Risks

- Experience in Unity
    a. Likelihood - Likely: This is a middle range risk for us because Unity as described to us by our client is "easy to pick up, but difficult to master". When it comes to the high level processes we might have to do later, our little experience in Unity might pose an issue
    b. Impact - Minor: Our clients have offered their assistance in issues we might face with Unity as they have worked with the software before and are familiar. The impact of any issues that arrive is expected to be lower because we have access to help and should only extend to the simulation instead of the full scale project.
    c. Risk mitigation plan - Research: Our whole team plans to spend multiple hours individually in sprint 1 to watch either Youtube videos specific to Unity and simulation (or Polycity Asset) to familiarize ourselves with the software enough to get started on building our simulation.
- Little experience in training a model, especially with computer vision
    a. Likelihood - Likely: As of right now, we are expecting a possibility of having to build our own model for detection, however it is possible we can adapt one to our data. Either way, we all have to learn how the model will function and apply to the simulation and predictions.
    b. Impact - Moderate: With whichever route we choose, the predictive model that we use will drive a lot of the functionality of the project. Therefore, it has to be highly functional especially if our goal requirements include specific accuracy goals.
    c. Risk mitigation plan - Our team has plans to research how to best fit our data to a LLM or ML, and to consult with the client in the face of any issues. The client has recommended that we use OpenCV for the project, so we are looking into how to integrate our data from the Unity model with OpenCV.
- All members have experience in C++, Java and Python, but only one member has experience with C#
    a. Likelihood - Likely: We will need to use Unity for our simulation for the project, and will likely have to use a decent amount of C# even though we can use assets and libraries.
    b. Impact - Minor: We anticipate a decently smooth transition into C# with our knowledge of other languages, but if there are any significant issues with our coding it does have the potential to impact the entire simulation - and therefore the whole project.

    c. Risk mitigation plan - Our team has planned to do code reviews with the client as well as with each other in subgroups. so that we can try to prevent any large scale errors early on. We also plan to use internet resources to get familiar with C# before beginning coding.

- Decision between training our own model for detection based on our data versus adapting an existing model
    a. Likelihood - Very Likely
    b. Impact - Major: The Computer Vision portion of the project is expected to be one of the largest portions in both production time and functionality. Choosing the best option to proceed with for detection is crucial to the project and the simulation's success.
    c. Risk mitigation plan -We have team members attending the Sagemaker seminar to get information on AI and try to get input on whether it will be more feasible to build or adapt a model. We have already discovered a way to export camera frame data from Unity, so our main concern is compatibility of the visual models versus our own simulation.

- Connecting work done across different platforms and languages
    a. Likelihood - Very Likely: Since the project involves multiple software that utilize different languages, we will have to ensure that the languages can communicate with each other while maintaining the integrity of both systems.
    b. Impact - Moderate: This is something that has to be done for the project to have all of its functioning components. The simulation without computer vision input would just be a system of moving vehicles. In order to function fully with detection, tracking, and interception, Unity and OpenCV will have to communicate with each other.
    c. Risk mitigation plan - Our team is investigating the use of APIs, sockets, or other various forms of integration for the multiple systems we will be using. This decision process is partially informed by the data from OpenCV's needs.

V. Definition of Done

Features:

- Unity city simulation depicting a variety of objects with irregular movement, including civilian, robber, and police vehicles
- Data collected from simulated cameras detect, track, and predict movement of robber vehicles
- Police are to pursue robber vehicles systematically
    - Success goal of model police intercepting robbers 85% of the time within reasonable speed and environment limits on police vehicles

Tests:

- Unit tests for individual components within Unity
- Functional testing for AI training and integration with simulation
    - Proper detection and classification of vehicles
    - Proper trajectory tracking and prediction
- Performance testing of simulation
    - Timing results based on variable factors such as vehicle speeds/traffic/distance

Product was demoed live on site for the client team and any other Northrop Grumman associates who attend. This was around the same time the final presentation was presented in class. Update demos were given weekly as well.

VI. System Architecture

Our architecture diagram is below in Figure 1, depicting our plan for the flow of information in our project. Our project starts out with a Unity simulation of the city we want to work in. The simulation then sends pictures from two bird's-eye-view cameras above the simulated city to a socket connected to a Python script. This script utilizes OpenCV and YOLOv8 to process the images, detecting vehicles and classifying them as either a "Car", "Police Car", or "Robber Car". Our results from the YOLOv8 identification process then get sent back to the Unity simulation, where cars are tracked according to the bounding boxes. This tracking allows us to predict future trajectories for the vehicle, thus giving us an idea of where police cars should go to intercept the robber cars. The interception algorithm sends police to each intersection the robber car could go to, and theoretically, they intercept this robber before they leave the city. This diagram has been updated to better reflect the final design in Figure 3.

Our project is very linear, meaning a data flowchart is the best fit for our process diagram, seen in Figure 2. Once pictures get sent from the simulation via a C# / Python socket, they are received by the Python script that contains both OpenCV and YOLOv8 to detect vehicles. For our purposes, we used a User Datagram Protocol socket between the two software, and converted the render textures of each Unity camera png into byte arrays and chunks for sending. Each frame received over the socket is then fed into the YOLOv8 detection model with a local API call. This script recognizes the vehicles and classifies them as mentioned above, with the use of YOLO's convolutional neural network capabilities. They will be given a bounding box and a confidence interval that they are being identified correctly based on the model's precision score. This information is then sent to another Python script that tracks them as they move throughout the city, done through the use of a centroid tracking algorithm. The centroid tracking takes in the bounding boxes of each detected vehicle, and throughout different frames associates objects together with minimized centroid distances. This algorithm will also keep track of where vehicles have been, and will track the velocity, location, and direction of each vehicle. It should also be able to distinguish each vehicle from another, even when the vehicle disappears for a bit behind an obstacle. The information stored by the tracking algorithm will be sent to the prediction algorithm, which will use previous information and AI to predict a path the car will follow. Specifically, for our prediction algorithm the 2D pixel coordinates of the centroid are sent over a secondary socket, and then raycasted into universal 3D coordinates. The prediction script takes note of this location, and uses segmentation within Unity to calculate the possible intersections that the robber vehicle will go to next. These possible paths get sent to the interception algorithm, which relays back to the Unity simulation how police cars should be dispatched and updates constantly with the received and processed coordinates. This diagram has also been updated to depict the final design in Figure 4.

Figure 1: System Architecture

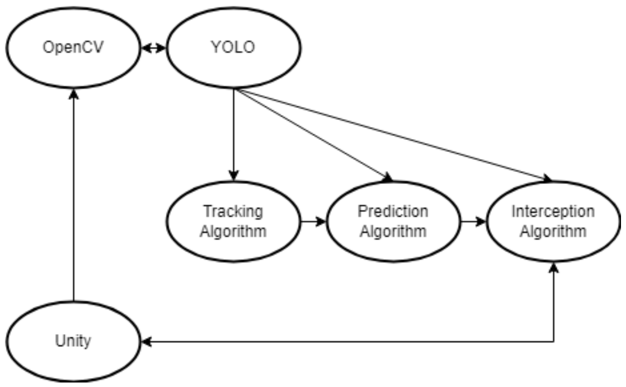


Figure 2: Data Flow Process Diagram

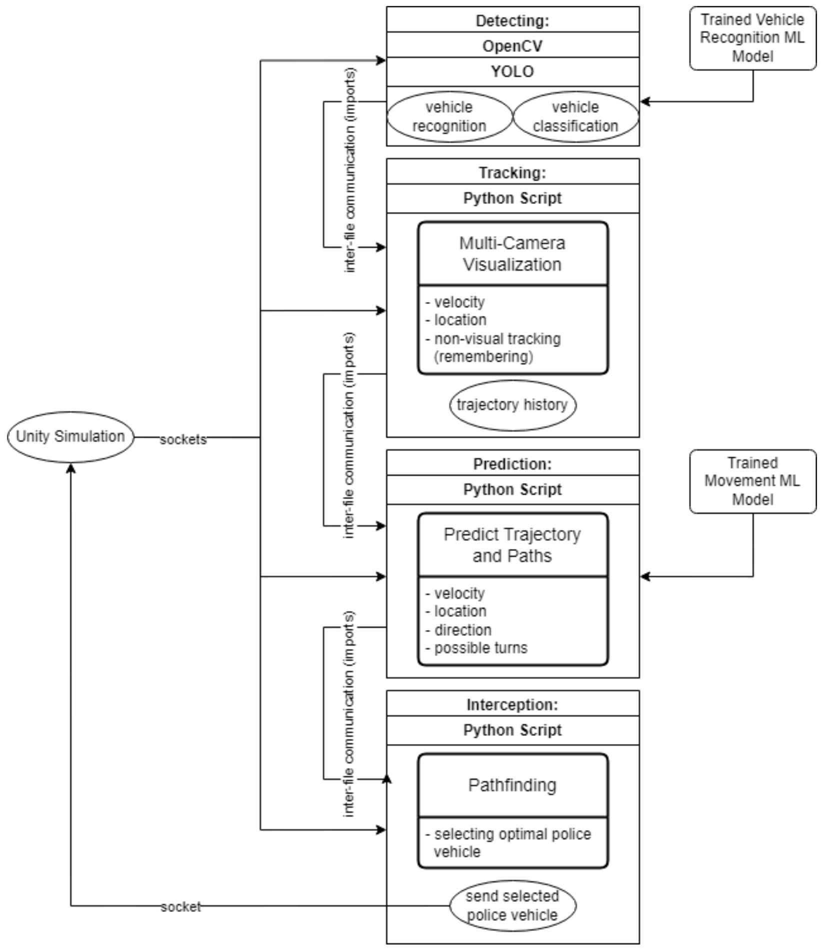

Detecting:
OpenCV
YOLO
vehicle recognition
vehicle classification
Trained Vehicle Recognition ML Model
inter-file communication (imports)
Tracking:
Python Script
Multi-Camera Visualization
- velocity
- location
- non-visual tracking (remembering)
trajectory history
Unity Simulation
sockets
inter-file communication (imports)
Prediction:
Python Script
Trained Movement ML Model
Predict Trajectory and Paths
- velocity
- location
- direction
- possible turns
inter-file communication (imports)
Interception:
Python Script
Pathfinding
- selecting optimal police vehicle
send selected police vehicle
socket

Figure 3: Updated System Architecture Diagram

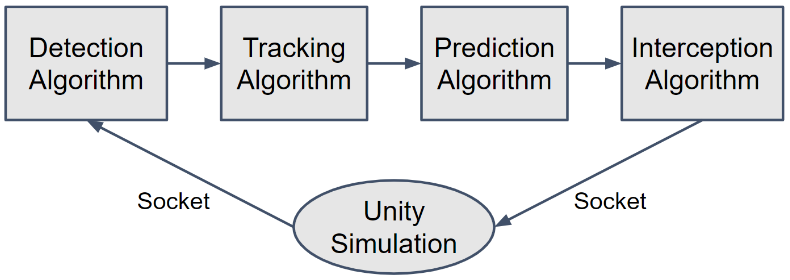
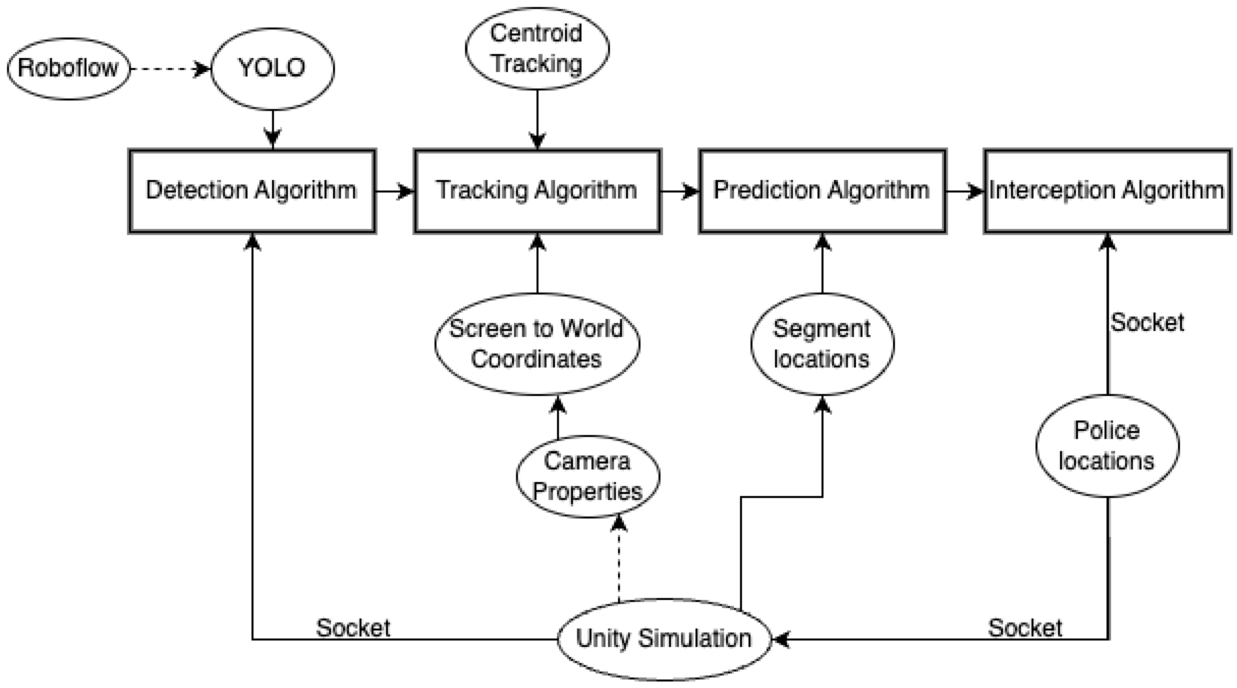


Figure 4: Updated Data Flow Process Diagram



VII. Software Test and Quality

Simulation Requirements:

- **Purpose of test:** Ensuring that the city environment created accurately represents a real-world scenario. Must be customizable.
- **Description:** Human verification; correct function of all entities in simulation (pedestrians, cars, traffic lights, obstacles, accidents)
- **Tools needed:** Unity simulation
- **Threshold for acceptability:** 100% - our simulation MUST function properly

- **Edge cases:** Cars leaving the simulation, traffic jams/accidents, left turn yields
- **Results:** A fully functional simulation that works correctly and consistently and resembles a real city.

Detection Requirements:

- **Purpose of test:** To ensure that cars can be correctly detected and correctly classified as a car, police car, or robber car.
- **Description:** Roboflow provides data on detection occurrence and confidence, we use these to decide whether or not a detection is valid or not, then decide if the detection was correct in classifying the type of vehicle (car, police car, robber car).
- **Tools needed:** Unity simulation, OpenCV, YOLOv8 (Ultralytics), Roboflow, UDP socket
- **Threshold for acceptability:** 85% - this subsection of the product requires us to use AI models, which will never be 100% accurate. Therefore, any number above a certain point cannot be assumed to be ever reached since AI is simply not reliable enough. Our team felt that 85% is a good estimate.
- **Edge cases:** Vehicles overlap in camera view, vehicle is only partly in view, vehicle goes behind an obstacle (building, pedestrian, tree, etc.).
- **Results:** A percentage of correctly classified cars that is above our goal.

Tracking Requirements:

- **Purpose of test:** To ensure that the model is not only detecting the vehicle but continuously recognizing it as the same object ID as it moves through the simulation, giving it a unique ID number.
- **Description:** This will be a visual test of 10 minutes of collected data. For a vehicle that is being tracked, instead of just displaying its centroid and ID, we will have the centroid leave a trail. This will be done by appending the position to a persistent between-frame list, with an array that acts with a rolling window, dropping the last value when adding new ones within a certain threshold. Because this will be tied to the vehicle's 'object id', if the id (ie recognition) changes for even one frame, the "trail" will disappear. This will show us how the centroid tracking can be improved for the object's continuity.
- **Tools needed:** YOLO and OpenCV, alterations of the code that draws the bounding boxes and centroids, and camera output.
- **Threshold for acceptability:** This is a repeatable scenario, meant for improving the success of our tracker. We don't have a defined goal of the length of the path based on the randomization of our simulation, but we are hoping to achieve the same object identification leaving a continuous "trail" throughout the entire time it appears on a camera, before leaving.
- **Edge cases:** There tend to be some issues with recognizing small parts of a car as its own, individual car, based on the model's training. There also tend to be issues when two objects cross each other's visual paths on the cameras or disappear behind objects (like people, trees, buildings). Ideally, these can be dealt with with filters and color matching, but will still likely affect some of the tests.
- **Results:** A consistent detection of one vehicle so that it can be tracked throughout the simulation. This is necessary for the prediction element as we map its past trajectory.

Prediction and Planning Requirements:

- **Purpose of test:** Observe whether or not the prediction algorithm's calculated/predicted trajectories for various objects match the actual trajectories of these objects
- **Description:** Once spotted, the robber car's path will be recorded and we will have algorithms to calculate, based on motion models, where the car could be in N seconds. In N seconds, we will then

check the cameras for the robber car in the possible locations. If the car is in any of these locations, then the test is successful. This is a repeatable situation test.
- **Tools needed:** Path Algorithm, successful detection and tracking (YOLO V8 and OpenCV), camera evaluation at the calculated coordinates
- **Threshold for acceptability:** As close to 100% is desired, but we consider 85% success (of the car being in one of the predicted locations) to be acceptable.
- **Edge cases:** The simulation has experienced traffic before in some odd situations, so it is possible that the vehicle will not follow its typical motion model. It is also possible that the algorithm will cover many path options, but not all. In these cases, it is possible that the robber vehicle could be in any number of places that aren't the predicted locations in N seconds
- **Results:** A functional predictive model, which is what is needed in order for interception to be successful at a high percentage.

Interception Requirements:
- **Purpose of test:** To ensure that our pathfinding algorithms can produce the most efficient interception strategy, and then send that strategy over to the police cars in the simulation.
- **Description:** We will keep track of the percentage of times a robber car gets stopped by a police car.
- **Tools needed:** Unity Simulation, UDP socket, pathfinding algorithm(s)
- **Threshold for acceptability:** 85% - this number is our favorite. In this case, we accept the algorithms if the robber car gets caught around or above 85% of the time.
- **Edge cases:** Car accidents, traffic, unpredictability of robber car
- **Results:** A pathfinding algorithm that can send information to the simulation's police cars and fairly regularly catch the robber car.

VIII. Project Ethical Considerations

ACM Principle 2.1 - Strive to achieve high quality in both the processes and products of professional work
We keep this principle in mind because our project requires us to provide a product that will serve the general public. Since this is the case, our product must be of high quality.

ACM Principle 2.3 - Know and respect existing rules pertaining to professional work
Since the company we are working with is an international defense company, they must keep their information private, and therefore there are things we are not allowed to know about and there are things we cannot share.

ACM Principle 3.1 - Ensure that the public good is the central concern during all professional computing work
This project is centered around keeping cities safe, so we must keep the public's well-being in mind while doing this project. We must simultaneously make sure that in the cases of surveillance, the "good" effects of the project outweigh the "bad". The privacy concerns behind detection and tracking must be considered against the benefits of being able to do this autonomously with computer vision.

IEEE Principle 2.06 - Identify, document, collect evidence, and report to the client or the employer promptly if, in their opinion, a project is likely to fail, to prove too expensive, to violate intellectual property law, or otherwise to be problematic

We have had to notify our client that we do not see certain aspects of the project as possible anymore, and we have shown them why and given a solution instead of the first solution that we proceeded with.

IEEE Principle 3.02 - Ensure proper and achievable goals and objectives for any project on which they work or propose
To keep this project moving, we must set reasonable milestones along the way. This allows us to stay motivated and means we will reach our goals instead of constantly reducing expectations.

IX. Project Completion Status and Results

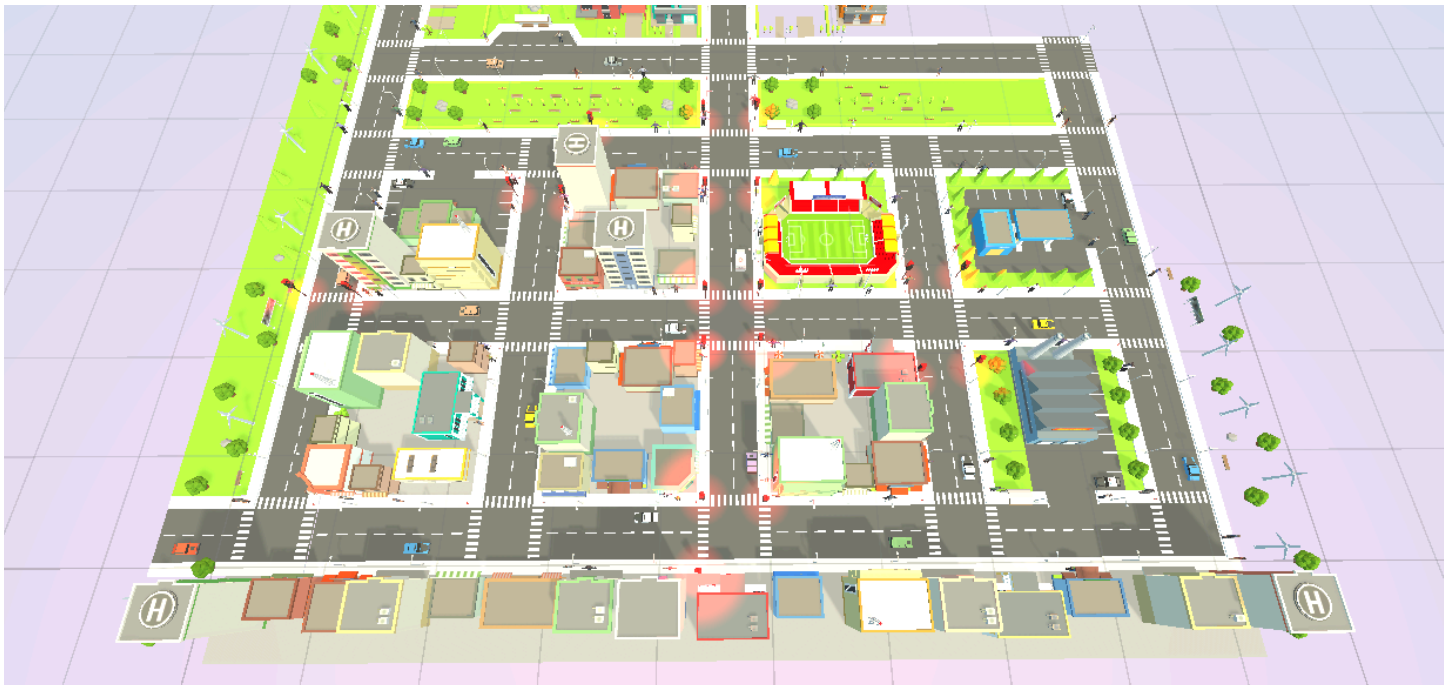
This project contains many subsystems, each building upon another to function properly. We began the project immediately constructing a simulation good enough to begin working on other subsections (this simulation was continuously updated and improved upon as the project continued, starting out containing just buildings and cars and eventually containing traffic lights, improved vehicle AI, and pedestrians). As soon as we got enough of a baseline simulation to be able to detect cars, we began working on our detection system, which included an AI model that we hand-trained. This model learned to classify cars that it “saw” in the simulation as either a “Car”, “Police Car”, or “Robber Car”. However, the detection system itself did not look at the simulation, it was instead handed screenshots taken inside the simulation via a socket. Our project uses a multitude of sockets to connect the Unity simulation with the Python scripts that control necessary external procedures. Once the vehicle detection was completed, we began working on the tracking, prediction, and interception algorithms simultaneously. The tracking algorithm again takes screenshots from a socket, then associates any cars in the pictures with a unique ID number that is maintained by the algorithm until the car leaves the simulation. It also notes the direction, location, and speed of the car. The cameras feeding these sockets are bird’s-eye-view cameras that constantly take pictures of the simulation below. These cameras are calibrated to the simulation using a Universal Coordinate System, which allows a translation between the coordinates in the cameras’ frame to the coordinates in the Unity simulation. The pictures taken by the cameras are then given to the tracking algorithm to perform these necessary calculations, and ultimately assign each car a coordinate value representing its location in the environment. This information gets sent via file writing and reading to the prediction algorithm, which takes note of where the car has been recently and where it could go in the future. We specifically curated this algorithm to predict the next intersections the car could go to once it passes through the one it is heading towards. This matters most in the case of the robber car, whose information is sent to the police cars in the Unity simulation after being processed by the interception algorithm, which sends a police car to each of the (at maximum) three intersections the robber car could next go to.

Features Implemented and Performance Notes:

- Simulation - The simulation has provided a great environment to test and work with, giving us data to feed AI models with and following the commands given by our Python scripts. See Figure 5 below.
- Detection System - Following the hours of work annotating pictures of our simulation by hand, our model worked great, achieving upwards of 85% accuracy on every sample we gave it. Finishing and fully implementing the detection system truly allowed us to move on much further into the project, starting work on practically every other algorithm at once, if not a few days after each other.
- Sockets - The sockets have proved to be quite a learning experience, requiring multiple days of dedicated work to function properly. However, once it was correct, it has proven to be the most useful tool we have for information flow.

- Tracking System - Our tracking system gave us plenty of issues when we were first trying to implement it. What gave us the most trouble was reassigning cars a new ID number many times but it also had a multitude of other buggy behaviors.
- Cameras and Screenshots - We tried so many different configurations of cameras for this project, failing with many attempts. At one point, we had 17 cameras in the simulation and have since brought it down to two. We toyed around with Non-Overlapping Multiple Camera Calibration but ended up getting in too deep and had to pull away from it and go with simpler options.
- Universal Coordinate System - Getting the UCS implemented took quite a lot of work, but ended up being the solution to a lot of problems we were running into with tracking. Figuring out ray projection and certain aspects of the relationship between the cameras' coordinate systems and the UCS (like the aspect ratio affecting the projection math, or the x-coordinate being off by about 27 and the z-value (distance from the camera to projected location) being negated, but only for certain cameras) was a struggle, and it all seemed almost nonsensical.
- Prediction Algorithm - Most of the issues for the prediction algorithm came from the coordinates being received correctly over the socket, and some strange set-up within the segmentations of the road.
- Interception Algorithm - Cop cars in our interception algorithm were initially going to completely incorrect segments, and sometimes a certain cop car would not be directed towards a target segment at all.

Figure 5: Overhead View of Unity Simulation



Features Considered but not Implemented:

- Non-Overlapping Multiple Camera Calibration - When dealing with stereo pair calibrated cameras, the projection math is most certain when you are calculating the three-dimensional coordinates of the same object in the same position, just from different angles. However, if there was no overlap between the cameras, projection to three dimensions would require a lot more individual information about the camera's parameters, location, disparity, baseline, and linear depth. This is something that we attempted to do, but were ultimately unable to make work within the scope of our project.

- Particle Filter - Particle Filtering would tie into our algorithm for interception as a framework for the probability densities of where a vehicle might be at one time. Each particle would be a hypothesis of the position and velocity, and the cop vehicle's movement would be directed onto paths in which their radius touches the most particles.

X. Future Work

If a team of entirely new students were to pick up this project where we had left off, we would recommend more than anything knowledge of computer vision. Computer vision software such as YOLOv8 and OpenCV have played a monumental role in the success of our project, and before their implementation came a lot of research and test runs (and failures). However, competence in Python, C#/Unity, and socket work would also be highly advised. Along with this, these students must have access to the Roboflow model and the local API calls utilized in our code, the GitHub repository that is being used to backlog our work, every Unity extension we have implemented, and access to local sockets on their machine.

If granted more time, we would've looked deeper into the Non-Overlapping Multiple Camera Calibration and the Particle Filter, both of which we had already considered. We also would have created more simulations for variable scenarios, maybe containing different road patterns or traffic systems. We considered implementing different camera resolutions or cameras with certain features like blurriness, glare, or greyscale, but this would've required training new models for these cameras as well, and we did not have enough time to commit to that. We started the project intending to make the robber car visibly indistinguishable from a civilian car and have its defining feature(s) associated with its interaction with the simulation, such as velocity. Implementing some or all of these features would certainly prove difficult, most likely talking about a month of extra work, but would greatly improve the overall extravagance of this project that our time constraint forced us to lack.

XI. Lessons Learned

- Unity provides a great environment for testing, but it also has a lot of surface-level functionality that is easy to use, but difficult to master. The ability to attach multiple scripts for socket, camera, or calculation-based functionality was essential for us to be able to achieve detection and interception. It can be very difficult, however, to debug because of all the (literal) moving parts and components. One recurring issue is that we fix some sort of bug in a script like the vehicle's AI, and then sometime later the bug would reappear for seemingly no reason. Despite the difficulties with the complexities of Unity, its complexities are also exactly the reason we needed it for this project. With this newfound, much more high-level understanding of Unity, we can now all comfortably use Unity to perform similar tasks such as modeling in the future.
- Python's versatility carried out much of the functionality of our project. We were able to call our API of the model locally, run OpenCV for object detection on our PNG stream, and run two UDP sockets for the incoming flow of Unity camera frames and outgoing pixel coordinates of the centroids. We were also able to apply an entire centroid tracking script. All of the libraries available for Python make a multifaceted project like this a lot more feasible. This knowledge of Python's powerful tools makes it a great resource for larger future projects that involve data transfer and/or analysis (almost everything ever).

- Sockets provided the essential integration point between all of our applications and scripts. With Unity behaving as a black box, we had to extract a camera stream for use in our various Python scripts. Simply understanding what a socket is and how to set one up for a client and host (in both Unity/C# and Python) was the most difficult part. Something that came in handy when working with the sockets was to build up functionality from the bottom. Before trying to do anything complex, make sure that you can receive a small form of data first. After this, exception handling comes in handy during the debugging of added functionality. Finally, the customizability of each socket is the greatest appeal of their use after getting to work with them. Sockets will forever continue to pop up depending on career paths, so being familiar with them now is very useful. Efficient data transfer methods are always great to have in the back of your mind.
- Computer vision is a fascinating subject, and luckily we had access to tools that were very straightforward to understand. Complex topics in computer vision, on the other hand, are not straightforward and involve a lot of technical math. When researching specifics of computer vision for our project, we dealt with a lot of abstract and high-level information, as well as a lot of the resources for our purposes would come from forums instead of direct and complete resources. Computer vision applications are useful and often streamlined, but it is a vast and complex field that would take a lot to master. Nevertheless, this was a main component of our project for detection, and YOLO, Roboflow, and OpenCV were ideal for our purposes. Now that we have a baseline knowledge of what computer vision is capable of, we are much more prepared for future projects that require cameras or image processing.

XII. Acknowledgments

To our advisors and clients - Thank you for your support throughout this project. As a team, we all greatly appreciate all of the advice, words of encouragement, and praise given to us. Your continued belief in us kept us in a great mindset to keep working.

To Kathleen Kelly - The weekly meetings with you were meetings we all looked forward to going to; we would always walk out with discussion topics that overall improved our team dynamic and eased our minds. Your encouragement and admiration of the project kept us wanting to produce a high-quality product.

To the Northrop Grumman Team - Thank you for constantly pushing us to do our best work. Your constant suggestions and insights gave us great starting points for certain aspects of the project, and even though some of your suggestions didn’t make it into this version of the project, if we were to continue working we certainly would implement them because they were all great suggestions.

XIII. Team Profile

Madelyn Swelstad
Junior
Computer Science
Hometown: Grand Junction, Colorado
Work Experience: yoga instructor, math tutor
Interests: Meditation, philosophy, international travel, reading, handstands, baking, cooking, guitar, art, happiness

Grant Dibala

Sophomore
Computer Science
Hometown: Houston, Texas
Work Experience: swim coach, lifeguard, equipment manager, sales attendant
Interests: rock climbing, swimming, water polo, running, music, math, nature, rocks

Caleb Curran-Velasco
Senior
Computer Science
Hometown: Colorado Springs, Colorado
Work Experience: Engineering Technician, Mover and Packer
Interests: Volleyball, Basketball, Soccer, Muay Thai, Video Games

Rachel Cavalier
Junior
Computer Engineering
Hometown: Arvada, CO
Work Experience: Server, Cashier, Data Center Technician
Interests: Kung Fu, Hip Hop, Fostering Cats, Reptiles, Basketball

References

Cherubini, Matthieu. "Unity Traffic Simulation." *GitHub*, 4 Mar. 2019, github.com/mchrbn/unity-traffic-simulation/blob/master/README.md.

VenCreations. "Simplepoly City - Low Poly Assets: 3D Environments." *Unity Asset Store*, 6 Apr. 2017, assetstore.unity.com/packages/3d/environments/simplepoly-city-low-poly-assets-58899.

Appendix A – Key Terms

Term	**Definition**
MOT	*Multiple Object Tracking: the practice of identifying multiple targets from incoming data and recording trajectories for each*
MCT	*Multiple Camera Tracking: tracking using multiple overlapping or non-overlapping cameras*

UCS	*Universal Coordinate System: the coordinate system implemented into the Unity simulation to allow multi-camera calibration with the environment*
ML	*Machine Learning: the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyze and draw inferences from patterns in data.*
AI	*Artificial Intelligence: the theory and development of computer systems able to perform tasks that normally require human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.*