InFrame: Final Report
18-500 - ECE Capstone

Martinez, Diego | Kilinc, Ahmet | Mercier, Ismael
Electrical & Computer Engineering, Carnegie Mellon University

***Abstract*— Robotic-assisted photography is among the most exciting technological developments in the production industry since the invention of the camera itself. It provides new and unique ways to capture dynamic scenes and unlock new avenues for cinematographic creative potential. Current photography robots on the market range from tremendously expensive industrial robotic arms, such as Motorized Precision's Kira camera robot, to portable devices that allow for smooth motion across basic scenes. However, most photography robots have not yet leveraged breakthroughs made in the Computer Vision space in the last decade. For these reasons, this design blueprints *InFrame*, an intelligent, motorized photography assistant that uses state of the art object detection models and tracking algorithms to follow user-selected targets across a 3D space, constantly keeping them *InFrame*.**

***Index Terms*— Photography, Computer Vision, Robotics, Deep Learning, Object Detection, Tracking, Embedded Systems.**

1. INTRODUCTION

AI-powered robotic systems have become the cornerstone of technological feats unachieveable by any human, pushing forward the boundaries of what is considered possible across countless technology-supported fields. Specifically within the photography and production industry, countless high-precision robotic devices are now being utilized to improve cinematography capabilities, such as Motorized Precision's Kira camera robot or the Rhino Arc II. However, few such products available on the market make adequate usage of the many research breakthroughs in the Computer Vision field to enhance system operation while maintaining overall system affordability. The additional functionality that results from applying these Computer Vision breakthroughs enables powerful autonomy practical for even the most advanced users, while hardware affordability broadens the product's user population to even the hobbyist level.

To fill this technological gap in market-available development, this team has decided to build the InFrame system and improve the quality standard for affordable smart-photography systems everywhere. The system is designed to address two primary areas of use-cases: semi-autonomous lecture recording and action shot capturing. Many courses at institutions like Carnegie Mellon University record 1-3 hour long lectures where the autonomatable task of following the target professor across a classroom floor is currently being done by a human idling his or her time while occasionally making acute camera adjustments. Action shots, such as a diver jumping from a 3-meter board, are not even being filmed until the olympic levels, creating a market gap for all other levels of diving activity for users who want to film themselves either during competition or period of practice. For both of these application areas, InFrame is the simple solution that users have long awaited.

These use-cases can be itemized into goals delineated among three primary system components. Within the Perception subsystem, the outlined technology must conduct object detection, object tracking, and be relatively successful in dealing with occlusions blocking the target from the camera system's field of view. Within the Hardware subsystem, a mechanism must be built that can position a camera's frame, record image data, and host a computing device capable of processing computer vision models. Lastly, to control the full stack, a concurrent system controller alongside an intuitive user-oriented interface must be developed. To measure success, this design must support object detection at 80% accuracy and tracking that allows for selection and frame-centering of a target for the outlined use-cases without major occlusions and major lighting changes. Furthermore, to guarantee that users view the camera as tracking them in real-time, the design must minimize latency between their measurable movement inputs and camera tracking outputs to be within a 60ms upper threshold. The achievement of these two principal metrics translate to a successful implementation of the InFrame system's core functionality.

This report will detail the quantitative and qualitative system design requirements, the solutions designated for meeting these requirements, an overview of how the product's subsystems interact with one-another, an itemization of design metrics established for assessing each subsystem's successful operation, an overview of previously evaluated and eliminated design alternatives, and an overview of the project management process behind the development of the overarching InFrame system.

2. DESIGN REQUIREMENTS

For InFrame to successfully support the use-cases outlined by this design, a set of itemized technical goals and quantitative (wherever possible) requirements must be specified and met in order to guarantee a successful product. These collections of goals and requirements are delineated across the product's principal subsystems: System Control, Perception, and Hardware.

2.1. System Control Requirements

Streamlined and comprehensive control is integral to the maintenance of the physical InFrame system's semi-independent functionality while providing intuitive interfaces for users to interact seamlessly with the overarching product functionality. This project design works to solve this central goal by building a system that can operate as independently as possible once it has an objective stored locally, while generating a human-facing interface for users to intermittently provide these objectives without constant dependence from the remote device in between command delivery. Therefore, this report proposal delineates design goals and requirements between these two subsystems: a core system manager and a user-facing interface for inputting system commands.

Consistently, a crucial number will arise across all timing considerations in this project: 60 ms. This is because popular consumer electronics ratings agencies such as RTINGS.com have conducted extensive research to prove that input lags above 50 ms start to become noticeable for humans. Hence, when a target moves, for the system to be responsive to that "input" and appear to track in real-time, the duration of the entire perception stack should be kept only a bit above 50 ms since that number is meant for gaming and a bit above that would still appear to be happening in real-time. Thus, 60 ms becomes an ideal objective across the various project subcomponents and their integration[2].

2.1.1. Core System Manager

The core system manager's foremost goal is to provide a framework for managing the product's various subsystems simultaneously so as to avoid queuing subsystem tasks, an outcome of sequential design that would slow down the overall system. In order to accomplish this goal, the core system manager has a requirement to be programmed as a collection of concurrently operating managers, each of which pass information (e.g. image frames, received user-commands, motor rotation vectors) among one another. Subsequently, to improve the effective operating speeds (e.g. effective post-processing framerate), the core system manager must be centralized to reduce the latency between data transmission. To support this goal, the system should be able to transmit a single image frame from the camera to the Perception subsystem within the same 60ms threshold it takes object tracking to output camera adjustment directives to the motors, representing the threshold for measureable latency to be detected by the human eye. A single image frame was selected as it is the largest amount of data that the system manager would have to send from one subsystem to another within a short period of time, thereby not adding further latency to an already time-constrained system. Lastly, since the product will be utilized outdoors and thus typically without reliable WiFi connection, video storage cannot occur on the cloud and must be operated offline; the core system manager must therefore support offline video storage with a simple download process.

2.1.2. User-Interface for Command Delivery

Commanding the InFrame system through a user-intuitive medium is critical to the development of a streamlined user experience and, as a result, a well-designed product. For any use-case InFrame is designed to deliver, both a screen for displaying images and an ability to touch select specific locations on said screen are critical requirements for the reason that users interacting with the system will need to both view object-detected targets within the camera's field of view and select a specific intended target. With the unnecessary complexity integral with installing, populating, and polling an external touch screen, a remote interface surfaces as a clear requirement. With the added goal of supporting outdoor use cases, communication with this remote interface cannot be conducted through an online server (e.g. AWS instance) since consistent WiFi connectivity cannot be relied upon outdoors.

In order to improve the responsiveness of the system to user tracking directives, the system should be able to respond to a user's target selection and begin operation within 2 seconds of making this selection on their previously object-detected frame (e.g. an image capture with detected targets indicated/outlined). This threshold represents the upper bound of how long the system designers were willing to wait between requesting a target selection and expecting the physical system to be able to respond in some degree of affirmation.

2.2. Perception Requirements

Perception lies at the core of InFrame's functionality. In order to have a fully functioning system, we need an accurate and efficient computer vision pipeline capable of object detection and object tracking.

2.2.1. Object Detection

State-of-the-art object detection models such as YOLOv2, SSD ResNet-18 and SSD Mobilenet-V2 achieve accuracies between 69% and 85%[1]. As such, the system requirement the team defined is to meet at least that level of accuracy on InFrame's outlined use cases.

2.2.2. Object Tracking

Since tracking is such an inherent part of the system, a requirement for the system is to never lose the target under normal circumstances. In any case, this translates to a success rate of at least 99% under normal circumstances with consistent lighting and small object displacements in between frames. Occlusions, objects changing shape and major light changes are certainly desirable but are not required.

2.3. Hardware Requirements

Hardware is the aspect that really makes InFrame come to life. As such, it is imperative that each aspect of the hardware has a specific set of requirements that need to be met to ensure a meaningful user experience. Furthermore, each aspect of these requirements must be tied back to the end user-experience so as to ensure the highest quality product.

2.3.1. Tracking Speed

The pan and tilt motors must support moving fast enough to track a professor walking from 2.5m away. At average human walking speeds this is 40° per second. Meeting these speed requirements ensures that users are not limited on how fast the tracking can be.

2.3.2. Battery Life

InFrame must be able to record continuously for the duration of a lecture or presentation. Most presentations or meetings will have at least one intermission in a 3 hour period. Therefore the target runtime is 3 hours. If users require a longer filming session the batteries could easily be swapped. It should also be noted that the battery life for most camera systems is around 3 hours, when photographers need to film for longer periods of time they swap out the battery on their camera. This requirement here is to have a user experience analogous to that of working with a traditional camera system. Although the battery capacity was not tested it was calculated and in this way proved to be sufficient.

2.3.3. Motion

The intentions were to enable InFrame rotate continuously to track targets anywhere around it. Due to manufacturing complications as a result of COVID-19 it was decided to leave out this capacity. InFrame is capable of 360° rotation, but it cannot do so continuously. The mechanical system must be able to support this without the potential for damage when the axes are moved to extreme angles. Therefore there the system was designed so there cannot be collisions between parts and the wires may not get tangled or twisted while moving. Creating a system that will not destroy itself over time is a requirement for any product.

2.3.4. Profile

InFrame is easy to transport in a backpack. Access to the battery allows for quick swapping. Like most photography equipment, it is tripod mountable. Again, a requirement for a quality user experience is to feel similar to standard photography equipment.

3. SYSTEM DESCRIPTION

The aforementioned requirements are the driving factor behind InFrame's design. Now, this report will outline the proposed design to meet these requirements to build a successful product.

3.1. System Control Suite

3.1.1. Core System Manager Design

The core system manager will need to maintain concurrent control of the camera products various subsystems, which include camera control, motor control, perception (object detection and object tracking), video storage, and communication with the remote control interface. To minimize the latency of data transfer between any two subsystems, such as moving image data from the camera control to the perception module, this design centralizes this compute on the Jetson device itself. Given that the Jetson will be host of the system's heaviest computing load, the perception operations of object tracking, this design will attempt to consolidate all other operations that depend upon the output of this computer vision work within this device. In order to meet the multifunctionality of the CSM's design and operate its core image processing pipeline while also sending / listening for Bluetooth data and marking data for future compilation, the system must be programmed to operate concurrently. Moreover, since the standard for conducting computer vision work is Python, which already has more extensive embedded interaction libraries than other concurrency languages, the CSM is programmed using Python and its multithreading libraries.

3.1.2. Remote Control Interface

As was outlined in Section 2.1.2, it is imperative for an intuitive target selection that InFrame features a remote touch screen display. Given the ubiquity of smartphone devices, the choice becomes clear. By pairing the InFrame system over Bluetooth to a user's smartphone device, InFrame becomes much more cost-efficient, user-friendly and easy to use. Furthermore, iOS was chosen due to iPhones' competitive advantage in the photography space and the fact that the InFrame team members all have iPhones.

The remote control interface is meant so that the user can send three types of commands: A *start/stop* command to start recording frames or stop to save the resulting video, a *detect objects* command so that the system may perform object detection on the current frame and send both the frame and the bounding boxes above a certain confidence threshold to the phone, a select target command to specify which target the system should start tracking, and a *terminate* command to signal the CSM to fully shut down.

3.1.3. Principle of Operation

To start up the main operational sequence, the core system manager will listen for and retrieve command information from the communication module, to which the remote controller will send system commands. When the user requests an image, the system manager will ping the camera manager to capture a frame, will forward this frame to the perception module where it can run object detection to outline potential targets, and lastly forward this information to the communication module for

transmission back to the remote interface for the user to select a tracking target. When the user communicates with the CSM to select a target to track, the comms module will retrieve the data, which will be parsed in a separate thread run by the main system manager, which will first take another picture using the camera. This frame will be passed to the perception module, where object tracking of the selected target will be conducted and a set of optical flow vectors, which indicate required camera adjustments, will be generated. This data is then delivered to the motor manager, where they will be reformatted into angular rotation and used to readjust the physical camera system to keep the target in the center of the frame. The image taken by the camera in this iteration will also be sent to the storage module, where it will be stored for future compilation. This cycle will repeat until the user requests to end the current filming term, at which point the main system manager will compile the stored frames into a video format and once again restore the system to a state prepared for the start of another tracking and filming term. Portions of this user interaction are outlined in further detail in Section 4, which also contains a description of the API this team has designed to enable the CSM and remote user interface user to communicate with one another.

3.2. Perception Pipeline

The computer vision pipeline is mainly composed of two subcomponents: Image pre-processing, object detection, and object tracking. The image pre-processing component downsamples an image so that it fits into the object detection model. The object detection component detects objects in an image above a certain confidence threshold. The object tracking component, given a bounding box, tracks the movement of the object within the bounding box from one frame to another.

3.2.1. Image Pre-Processing

The InFrame system features a Raspberry Pi Camera V2, which is set to operate at 720p60. This is because the main camera is used not only for object detection and tracking but also for actually recording video, so a high quality (for video playback) and a high frame rate (for slow motion capabilities in post-processing editing of action shots and small displacements in between frames for accurate tracking) is desirable. However, 720p (1280x720 pixels) is a much higher resolution that is needed for image inference, so images are downsampled to fit the object detection model's input.

3.2.2. Object Detection

The object detection component of the perception pipeline will make use of a pre-trained neural network architecture designed for object detection and OpenCV's Deep Neural Networks module to load said network. The model that it will use is SSD Mobilenet-V2, which is an architecture designed by Google AI for on-device mobile vision applications. It is a lightweight model that achieves 39 FPS on object detection tasks when running on a Jetson Nano using 300x300 images[3]. It also achieves very similar accuracy benchmarks as other state-of-the-art object detection algorithms, as shown in figure 1 below.

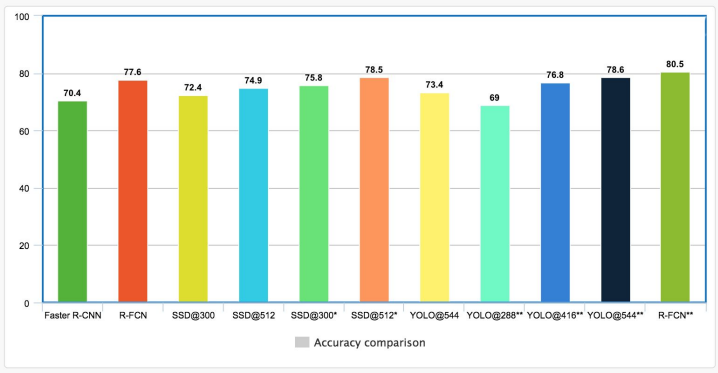


Figure 1: Accuracy comparison of state-of-the-art object detection models

3.2.3. Object Tracking

After a bounding box has been selected by the user, the perception pipeline enters the object tracking stage. Here, the system determines the optical flow from one frame to another in order to track the movement of the object within the bounding box. Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of an object or a camera. It is a 2D vector field where each vector is a displacement vector showing the movement of points from the first frame to the second. By using optical flow, the system can determine the direction and magnitude of movement of an object from frame to frame and use that to drive to motors to track the subject. InFrame calculates optical flow by calculating the vector from the center point of the previous bounding box to the center point of the next.

In order to track objects in real-time on the Jetson Nano, several tracking algorithms were implemented and considered. Ultimately, because the Jetson Nano's image comes pre-packaged with an inference optimization library called TensorRT which achieves speedups of up to 40x, object detection itself proved to be the best tracker. As such, once the user selects a target, the system saves the class ID of that target (i.e. human, dog, skateboard, etc.) and looks for that same class ID in future frames' detection results. In InFrame, this means that there can only be one instance of each class per frame, otherwise it would randomly switch between them. This can be easily improved by gathering features from the target bounding box, comparing them to the features in each potential bounding box for all the objects pertaining to that class and choosing the one that minimizes the difference, but the team did not have time to implement this improvement.

3.3. Hardware Design

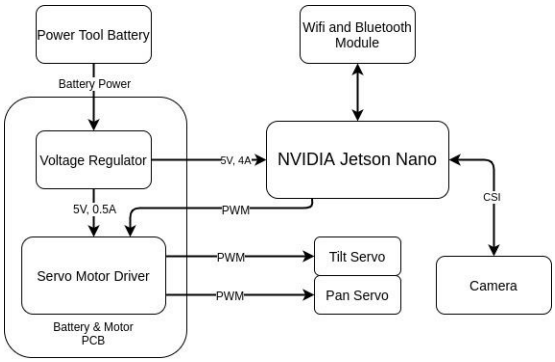


Figure 2: Hardware Interaction Diagram

3.3.1. Battery Management - Voltage Regulator
The 18V from the power tool battery is converted to necessary 5V 4A voltage for the Jetson and the 4.8-7.2v needed for the servo motors using a switching regulator. This is done with a custom PCB that was not manufactured due complications with the pandemic.

3.3.2. Power Tool Battery
A power tool battery has been selected because it will be easier and safer to work with than just LithiumPolymer batteries since it already comes with the protective charge and discharge circuitry. A Milwaukee Tool 18V 6.0Ah battery has been selected because it provides sufficient power for the desired runtime while still remaining economical. Standard photography equipment uses interchangeable batteries so in doing the same InFrame shares a similar user experience.

3.3.3. Servo Motor Driver
This circuit will be a simple logic level shifter that will allow control of the 4.8V-7.2V servo's with the 3.3V Jetson gpio pins. An optocoupler is used to isolate and protect the Jetson from any potential voltage spikes coming from the servo motors. This board will also have a power passthrough from the battery manager to the servo motors. This circuit will be on the same PCB as the voltage regulators.

3.3.4. Servos
This option was chosen due to the ease of controlling a servo motor since they require less hardware and simpler control signals than other motors. In addition to this, servo motors offer greater torque for their size as a result of their internal gearing system. For panning motion, the system will use a continuous rotation metal geared servo motor that will enable 360° rotation. For tilting, the system will use a 180° metal geared servo which will allow movement without worry about collisions since it cannot go past the set limit.

3.3.5. Camera
The system will use a 1080P30FPS camera that will be easily integrated with the Jetson through a CSI MIPI interface that is built-in. This camera has a high enough FPS where we can constantly keep the GPU running the tracking algorithm and also film a quality video. 1080P30FPS is the typical "high quality video" resolution seen in most online streaming. Providing the highest quality possible is a key aspect of any photography equipment.

3.3.6. WiFi and Bluetooth Module
The system will feature an Intel 8265NGW Dual Band WiFi and Bluetooth 4.2 module that will enable wireless communication for the Jetson. It plugs directly into the Jetson's M.2 connector and works with supported software. This will allow for easy Bluetooth communication between InFrame and the iOS Device.

3.3.7. Nvidia Jetson Nano
The Jetson is the most affordable embedded single board computer that offers high computer vision performance with its NVIDIA GPU. See section 6.1 for more details.

3.4. Mechanical Design

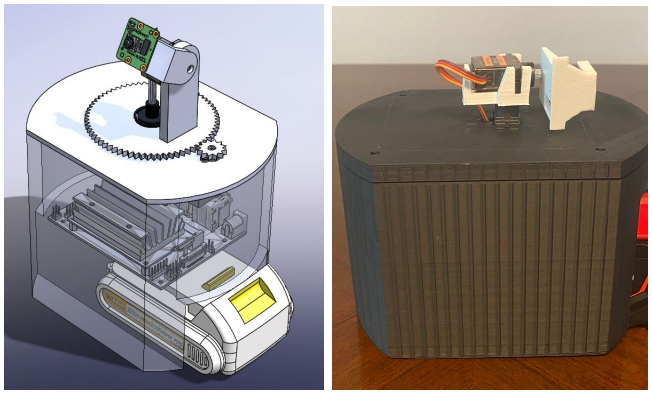


Figure 3:Physical System Design

3.4.1. Motion
To properly track a target, InFrame must be able to pan and tilt the camera. This motion will be achieved with servo motors. A continuous rotation servo motor will allow InFrame to track targets in full 360° motion. Due to pandemic complications, a slip ring was not integrated into the panning mechanism and therefore it is unable to turn continuously and is restricted to just 360°. A 360° tilt will not be possible because the tilt arm would collide with the rest of the system so it has been limited to 180° tilting. To be able to track a person moving at average walking speeds from 2.5m away (test case for a lecture) InFrame must be able to move at 40° per second. This is easily achievable with most servo motors.

3.4.2. Profile
To be able to comfortably transport Inframe in a backpack, the system should be limited to a maximum of 6.25" x 4.5" x 5.5". There will also be an easily accessible battery slot to be able to quickly swap batteries. In addition, inFrame will also feature a standard tripod screw so that users may mount it to a tripod like most camera systems.

4. SYSTEM INTERACTION

A detailed diagram of the way that information is passed across the different components of the system is included in Appendix 1 (section 10.1). It outlines the different serial communication protocols used to communicate between the software component in charge of initializing and managing a specific hardware component and said hardware component, as well as the main purpose of each of the different subcomponents of the entire system. In addition, on the right side of the diagram, there is a detailed look at the different commands that can be sent from the iOS device to interface with the InFrame system.

This diagram gives a very general overview of how different parts of the system work together to create the fully functioning InFrame system. In order to illustrate two common uses of InFrame, this report will now dive into how the target selection and target tracking scenarios work.

4.1 Sequence Diagram for Tracking Target Selection

A detailed diagram of the target selection sequence is included in Appendix 2 (section 10.2). Here, a user uses the iOS device to request the Jetson Nano to perform object detection. Then, the iOS app draws the resulting bounding boxes on top of the current frame. Finally, a user can select one of these bounding boxes and its ID is sent back to the Jetson to move forward with tracking.

4.2 Sequence Diagram for Object Tracking

A detailed diagram of the object tracking sequence is included in Appendix 3 (section 10.3). Here, after a user has selected a bounding box and the Jetson Nano received the appropriate bounding box ID, it can continuously use the camera and perception managers to keep track of the object within the bounding box and compute the optical flow from frame to frame. This optical flow can then be translated by the motor manager to actionable movement and InFrame can effectively track a subject in 3D space.

4.3 Communication Protocol: CSM & Remote Interface

In order to clarify and organize interactions between the CSM computing host and the Remote Interface, the following API was designed.

Remote Interface → CSM

"Start" - Signals the CSM to begin its target selection process by taking an image of its current view, annotating the image with detected target options, and returning this frame and detection information.

"Select:LEFT;TOP;RIGHT;BOTTOM" - Sends the CSM the boundaries of the intended target's bounding box border locations. Each element indicated in all capitalization represents an integer value of the bounding box border's location in terms of a pixel value on the image. This command starts a CSM target tracking and filming term.

"Finish" - Signals the CSM to end its current target tracking term, compile the frames it has accrued, and prepare for the next starting instruction from the user.

"Terminate" - Signals for the CSM to terminate all operations, compile its current footage (if the term has not yet been terminated), and shut the system down.

CSM → Remote Interface

Detections (Format: .JSON)

```
{
 "frame":
        { filename, image, width, height},
"detections":
        [ {Instance, ClassID, Confidence, Left, Right, Top,
        Bottom}, ...]
}
```

5. METRICS & VALIDATION

To measure the system's overarching ability to meet its outlined goals, the following section details the testing methods and resultant degrees of success for each requirement pertaining to InFrame's subsystems.

5.1 Subsystem Testing

5.1.1. Object Detection
In order to gauge the effectiveness of the object detection model of choice, a suite of tests cases using images found online was put together. This suite was designed around the most common use cases that InFrame is meant to support and is therefore spread out evenly across 3 common use cases: 40 images of lecturers, 40 of runners and 40 of skateboarders.

By using a binary success metric here (successful if it detected the target in each image), the system achieved 92.5% accuracy on lecturers, 77.5% accuracy on runners and 85% accuracy on skateboarders when running the SSD Mobilenet-V2 object detection model. By averaging these results, an average of 85% accuracy is obtained, which successfully meets the target requirement of 80%.

5.1.2. Object Tracking

Object tracking is a critical aspect of InFrame's functionality (its raison d'etre, if you will). Because of this, the team set out to meet the lofty requirement of never losing the target under normal circumstances (or a success rate greater than at least 99% to account for some minor errors), which means no occlusions, lighting changes or object shape changes. This metric was tested by building a suite of 10 test sequences, each one minute long and shot using a frame rate of 60 FPS -- meaning a total of 1200 frames to track. This test suite includes 6 "normal" test cases, as well as 4 "reach" tests. The latter tests include occlusions, lighting changes and major object shape changes and as such aren't required but are good to have to measure the extent of the system's capabilities.

By having the software draw bounding boxes around the target in each frame, successful tracking was manually verified on each frame. Out of the 720 frames of the 6 easy tests, 718 were successfully tracked, which translates to a 99.7% success rate. For the reach tests, out of the 480 frames, 417 were successfully tracked (86.9%)! Again, the requirement for the reach tests was essentially 0% because these tests include very major occlusions, very dark environments quickly turning very bright, objects completely changing shape, and other very hard-to-track scenarios, so such a high accuracy here is very impressive.

Finally, the last requirement for object tracking is to be able to track any possible target that is within our outlined use cases. This requirement however isn't met in the team's final implementation of the InFrame system. This is because the final optimization done on the perception pipeline doesn't use bounding box features to track the target. Rather, it saves the class itself (human, dog, etc) and looks for that same class at each frame. If at any certain frame there's more than one instance of a class, the system would randomly track one of them, so there can currently only be one of each class per frame. This can be improved by saving bounding box features and looking for the most similar bounding box within a certain class (i.e. the system looks for a human but also the human that looks most like the one selected initially) but the team did not have time to implement this last feature.

5.1.3. Battery Management Circuit

Unfortunately, due to COVID19 complications the battery system was not tested. These are the proposed tests. Once the voltage regulator has been manufactured it would then be connected to the battery and a 15W dummy load would be applied and the output power would be recorded over time to see what sort of dropoff the battery and regulator system produce. These measurements would be taken on the battery voltage and the regulator voltage. It is expected that the battery voltage will drop but the regulator would maintain This needs to be proven. The total time the battery is able to sustain this load without discharging the lipo cells to their dead voltage would be recorded. This needs to be proven to be at least 3.5 hours. Once it has been determined that there are no strange voltage spikes in this system the Jetson and motors may be connected and again timed to ensure the battery lasts as long as expected.

5.1.4. Motor Control

The motor's speed was tested to prove they are fast enough to track targets. The motors were timed using software timers and the angle turned was measured for several operating points. A maximum operating speed of 100°/s was measured. Motor accuracy was determined by moving the motors clockwise and counterclockwise 10 times and the offset angle was measured. The motors were within 5-7° of the target for each test. This met our 10° accuracy requirement.

5.1.5. Camera

Testing the camera is as simple as proving that the desired resolution can be achieved at the required frame rate. It is important to determine if there are any dropped frames or similar issues. This can be done by running the camera at the desired settings for a period of time and making sure the images look as they should.

5.1.6. Bluetooth Commands

Data transmission of user-inputted commands from the iOS interface to the Jetson CSM is critical to the utilization of the remote user-interface. To ensure that these commands are going through properly, the system's communication was first tested under ideal conditions with no active threads controlling InFrame's other subsystems by sending a Bluetooth message to the Jetson and viewing if its contents can be printed on its terminal. Then, to better model expected conditions, the transmission and retrieval of the Bluetooth message signaling the CSM to stop recording was tested while the various other subsystems are running on concurrent threads. These tests all yielded positive results, with the CSM system functioning robustly and exactly as intended.

5.1.7. CSM Data Flow

The primary purpose of the CSM is to coordinate all interactions between the various system components and act as the product's control flow operator. For this reason, the CSM has to live within the same timing objectives as the overall system. Specifically, the CSM should, independently,

be able to transfer data from the camera manager to the motor manager within the 60 ms upper bound this report has established early in Section 2.1. To accomplish this task, without any heavy Perception code running, the CSM was simulated repeatedly retrieving data from the camera module, pushing it to the motor module, and recording the time it takes to accomplish this cycle. The results of this experiment yielded an independent operating time of $3.76*10^{-3}$ms between passing sample data in through the Camera module (without operating the physical system itself) and this data being sent to the Motor module without the intermediary Perception processing (this is taking place while both the Communications and Storage modules are running concurrently). This result is tremendously encouraging and far below the ideal 60ms objective!

However, as a follow up, a second experiment was conducted to test the CSM under concurrent loads more closely representing its expected working conditions. Specifically, the same time interval was calculated once again; however, in this test, the Perception code was additionally being run. This test was done with 10 seconds of filming, over 5 iterations. Overall, this test reports 170ms of time between each frame.

This second experiment posed significantly slower operation than the previous, going beyond the 60ms objective the project had set. After further experimentation, it was found that the primary reason for this slowdown is Python's principal means of running concurrent code: the Python Global Interpreter Lock (GIL). This data structure implements what can only be described as a rudimentary concurrency model, the likes of which could be designed by a university student taking their first distributed systems class.

After researching means of improving the system, this team has found two possible solutions. The first solution would be to rewrite the entirety of the CSM in a more concurrency friendly programming language, such as Golang, whose planning and structure supports the kind of multithreading work this project demands. This method would additionally require establishing methods for modularizing the Perception code so as to be called from a compiler-based language like Golang. The second solution would be to implement multiprocessing, an alternative Python library that is often used by developers to represent parallel systems in the language. Although this approach would not be the optimal approach, it will likely improve the higher-load operating speeds the CSM will experience. Unfortunately, with the scheduling delays brought upon by COVID-19, fully implementing and debugging either of these approaches did not become possible. Preliminary code was written for the multiprocessing alternative; however, this team did not consider this prototype to be robust enough for merging into the project's final master branch.

In addition, it should be acknowledged that a secondary source of slowdown in the fully functioning system is the operation of the physical camera. Beyond the unavoidable delay in between the camera is able to take individual frames, driving the device seems to take a measurable amount of the Jetson's resources. Beyond a typical approach of scaling up the hardware by purchasing a better camera, this report could improve upon this camera utilization by creating an independent PCB driving the camera or designing a new camera system entirely for the more specialized purpose of recording and downloading single frames with as little wait-time as possible. With both approaches drastically expanding the scope of the project, these alternatives would need to be accomplished over a longer period of time than a single Capstone course.

5.1.8. Image Transmission

With the size of a typical InFrame image being significantly greater than that of the String instruction commands, the transmission of Bluetooth commands will be a relatively lighter case compared to that of a full frame. The testing of this functionality will be done simply by serializing and delivering an image, a collection of pixel coordinates (representing all possible object-detected outline boxes), and a collection of ID values associated with each box via Bluetooth from the Jetson to a paired iPhone. This test was conducted for over 5 repetitions to be received and displayed on the iPhone within a target 2 second period. These tests resulted in an average measured time of 443 ms across the entire testing suite. Therefore, the testing of image transmission is classified to be a success.

5.2 Integration Testing

As a result of COVID19, the team's integration efforts were significantly impacted. Integration of the full system was achieved on a software level, but the team was not able to fully integrate the software stack with the system hardware. The extent of this integration was to a point in which a clear set of inputs and outputs is verified to/from the software so that integrating the hardware is a matter of simply plugging things in. However, it was not possible to perform full system testing.

Nevertheless, the team did everything in their power to test integration. Lecture recordings were mimicked by having Diego running the full software stack on his Jetson and sending the optical flow output, as well as a video recording of him moving from side to side to Ismael so he could test the motor's response to said movement. In order to verify successful functionality under this limited test, the team used the optical flow output from Diego's movement on the motors that Thor had, looking to see if the camera moved in ways consistent with Diego's movement. This test proved to be

successful and the team is confident that they could integrate software and hardware easily in the future, allowing them to properly test the full system as outlined below.

5.2.1. Lecture Recording

When the project is completed final testing may be performed. To test usability and functionality, someone not part of the team will be performing these tests. First this test user will set up InFrame in a lecture at the front of the class. They will have to go through the initialization process of picking a target, the professor, and then starting the recording and offloading of the video.

5.2.2. Action Shot: 3m Diver

To test the athletic event scenario a user will set up InFrame to track him/herself as they dive off of a 3m diving board and then recover the footage after the dive. To decide how to best improve and move forward with InFrame multiple test users will be selected so that their feedback may be used.

6. DESIGN TRADE STUDIES & TRADEOFFS

6.1 Main Compute

The largest bottleneck for InFrame is the image processing pipeline. It is imperative that the computer vision be performed as quickly as possible to leave enough time for motors to update within the 60ms alloted for each cycle of tracking. Additionally, this computation has to be hosted locally to support outdoor use cases. The decision to use a Jetson as opposed to other popular single board computers such as the Raspberry Pi comes down to how much processing power each board has. The NVIDIA Jetson Nano outperforms any RPI board substantially in a computer vision application. This is because the Jetson has a GPU which the RPI lacks. There are not many SBC's that offer GPU capabilities and the Jetson Nano is the only one reasonably within the project budget.

6.2 Core System Manager

On the core system manager (CSM) end, one major point of discussion was rooted in the language that the Jetson Nano would be programmed in. With the heavily concurrent design requirements for the CSM, another language considered for deployment was Golang, a modern programming language designed to support scalable distributed systems. Although developing in Golang would facilitate debugging, improve code clarity, and overall make the CSM codebase more manageable, Golang's primary shortcoming is its relative lack of GPIO and embedded interfacing support when compared to Python, for which heavily supported libraries for GPIO (interfacing with motors) and CSI (interfacing with camera) have already been developed. Furthermore, since the computer vision work on the perception manager must already be written in Python due to the powerful CV-based libraries it supports, consolidating the full software stack to a single programming language improves the quality and consistency of the overall codebase. Lastly, as described earlier in Section 5.1.7, research was conducted to replace the concurrency libraries to be used in Python with a language supported multiprocessing framework. This post-implementation research found the multiprocessing approach to likely improve the independent speeds achieved by the CSM; however, due to COVID-19 scheduling delays, there was not enough time to robustly implement this alternative approach to CSM control flow.

6.3 Remote Interface

In the process of selecting the medium for the system's remote control interface, several options were first considered before deciding upon an iOS framework. First, a webapp was evaluated as a potential remote control alternative due to the project team members' past experiences doing such development and the opportunities it offers for livestreaming footage over WiFi onto an AWS instance. However, with an intended use case of outdoor filming and the resulting inability to rely on a consistent WiFi connection, the possibility of webapp system control was abandoned in favor of a hardware device that can communicate with the camera system directly without a middleman remote server. With the additional need for an easily interfaceable touch screen, the options were limited to developing an iOS or an Android user interface. In selecting between these two devices, although Android development is facilitated with its no-investment development cost as opposed to the iOS development-subscription system, all InFrame team members have iOS devices, wish to use this device post-development, and have space within the project budget to afford this relatively low subscription cost.

6.4 Wireless Communication

As the need for a remote user interface arose, selecting a wireless communication protocol (WCP) for data transfer between the interface and the core system manager became a critical portion of the design. Although Bluetooth eventually arose as the optimal WCP, this design process additionally evaluated communication over a WiFi protocol. This alternative would allow for higher image transfer rates at roughly 11Mbps, while Bluetooth only offers roughly 800Kbps. This boost in speed would have been utilized to enable WiFi-based live streaming of camera feed and camera control. However, this alternative was found to be unnecessary as the project team classified livestreaming as an additional feature to be beyond the project scope and intended MVP. Additionally, the process of setting up the Jetson Nano as a WiFi-connectable device would, on the user's end, staple an additional period of setup time that would exceed the 20 second requirement this project is committing to. For these

reasons, WiFi-hosting on the Jetson Nano was abandoned in favor of a lighter, more low-power Bluetooth communication protocol. As a result, users view possible targets to follow on an image snapshot representing what the camera system can see and detect at the time when the user requests to start the target selection process.

6.5 Object Tracking

The most interesting tradeoff decision for the perception pipeline was the choice of tracker. There are currently many different tracking algorithms available in OpenCV, all with varying degrees of accuracy, speed and even ability to detect failure. The two most worth noting though are those at the opposite ends of the "tracking spectrum". The fastest tracker available is the Minimum Output Sum of Squared Error (MOSSE) tracker, which uses adaptive correlation for object tracking. It is robust to variations in lighting, scale, pose, and non-rigid deformations. While it does run very fast, achieving an average frame rate of 5.98 FPS on the Jetson Nano, it doesn't work very well. In a suite of 20 tests, it lost the target 19 times.

On the other side of the spectrum, there is the Discriminative Correlation Filter with Channel and Spatial Reliability (CSRT) tracker, which uses a spatial reliability map to adjust filter support to the part of the selected region from the frame for tracking. It runs much slower than MOSSE, achieving only 1.19 FPS on the Jetson Nano, and while it does work somewhat better, it is still not reliable enough to be used as the sole tracker in a system whose main purpose is tracking -- losing the target in 14/20 of the test cases.

Now, because even the fastest tracker was still about 3x slower than out 60 ms (16.6 FPS) requirement, the team looked into using the fastest tracker available (MOSSE) and improving it by running object detection every certain number of frames to "reset" the bounding box before it accumulates error and loses the target. This improvement worked very well, achieving an average frame rate of 4.83 FPS (a slow down of only about 1 FPS from standalone MOSSE), but losing the target on only 1 out of our 20 test cases.

Near the end of the semester, it was discovered that the Jetson Nano comes pre-packaged with an SDK called TensorRT, which optimizes neural network inference performance up to 40x. By switching the inference code to use the libraries from that SDK, object detection was running at an average speed of almost 14 FPS, with a max of nearly 35 FPS. Because of this significant speedup to the object detection part of the pipeline, the detection model itself was used as the tracker (the whole point of using object tracking is that it's supposed to be faster than running object detection at every frame). This improvement not only resulted in significant speedup of the perception pipeline, but by switching from tracking algorithms to deep learning models, occlusions, lighting changes and many or reach cases started to work much better than expected. In our suite of 20 test cases, the object detection tracker *never* lost the target.

6.6 Object Detection

The choice of object detection model boiled down to the fastest and lightest model which could meet the 80% accuracy requirement on the system's use cases. Running heavy models on more capable cloud instances was considered but quickly discarded since the system should be independent of a Wide-Area Network so that it can be used outdoors. The fastest object detection model on the Jetson Nano was SSD MobileNet-V2 and as outlined in section 5.1.1, it achieved an accuracy of 85% on our use cases. Other heavier models such as ResNet-18 were considered and actually did perform a bit better in terms of accuracy but ran inference so slowly on the Jetson Nano that they were not viable options.

While going over the design of the perception pipeline, a major point of discussion was on whether InFrame's tracking capabilities should be based on object or face detection. Object detection works with high-level features such as shape and relative size, while face detection depends on more minute details such as colour, structure and shading. However, some research suggests that object detection could be generalized to distinguish between different faces as well. Furthermore, at ranges of up to 15m from the camera, faces start to become very similar. As such, it was decided that InFrame would use state-of-the-art object detection to differentiate between different objects. Rather than recognizing a person and assigning an ID to them as you would with facial detection (in such a way that you could say "track Bob"), InFrame detects everything as an object and leaves it up to the user to select what the target is (by using bounding boxes rather than IDs).

7. PROJECT MANAGEMENT

7.1 Schedule

A detailed look at InFrame's team schedule is included in Appendix 4 (section 10.4).

7.2 Team Member Responsibilities

Diego Martinez is responsible for the design and implementation of the computer vision software and its integration with the rest of the system.

Ismael Mercier is responsible for all electronics hardware as well as mechanical hardware.

Ike Kilinc is responsible for the core system manager, the remote interface on iOS, all data transfer and coordination across the system, and pushing forward software integration.

7.3 Budget

Attached below are all the parts identified as necessary for InFrame. The total cost is $406.41 which is well within the allocated budget.

Part	Name	Cost
Tilt Servo	Maxmoral 2pcs MG90S 9g Metal Gear Pro Servo	$9.99
Pan Servo	Metal Gear Micro Servo / Continuous Rotation	$16.99
Camera	Raspberry Pi Camera Module V2 - 8MP, 1080p	$28.20
Battery	Waitley M18 18V 6.0Ah Replacement Battery	$42.98
Central Compute	Jetson Nano Developer Kit	$99
Battery Adapter	Milwaukee 49-24-2371 M18 Power Source	$33
3D Printing Material	Matte black plastic	$26.99
WiFi/Bluetooth Module	Intel 8265NGW	$30

Figure 4: Parts List & Overall Budget

8. RELATED WORK

The Rhino Arc II is a 4-axis motorized system designed for amateur to professional photographers that was recently crowdsourced on Kickstarter. Much like InFrame, it too has pan and tilt motors, interchangeable batteries and a remote control interface. However, the key difference between the Arc II and InFrame is that InFrame has object tracking capabilities and is itself an end-to-end solution that does not require an external camera.

The Arc II does have some very elegant features and modes of operation that InFrame could benefit from. These include keyframe support and variable speed curves in between keyframes, the ability to adjust zoom and focus and a built-in interface to control the system alongside the remote interface. Because of this, InFrame is designed in a scalable and maintainable way so that adding features like these in the future is straightforward to achieve.

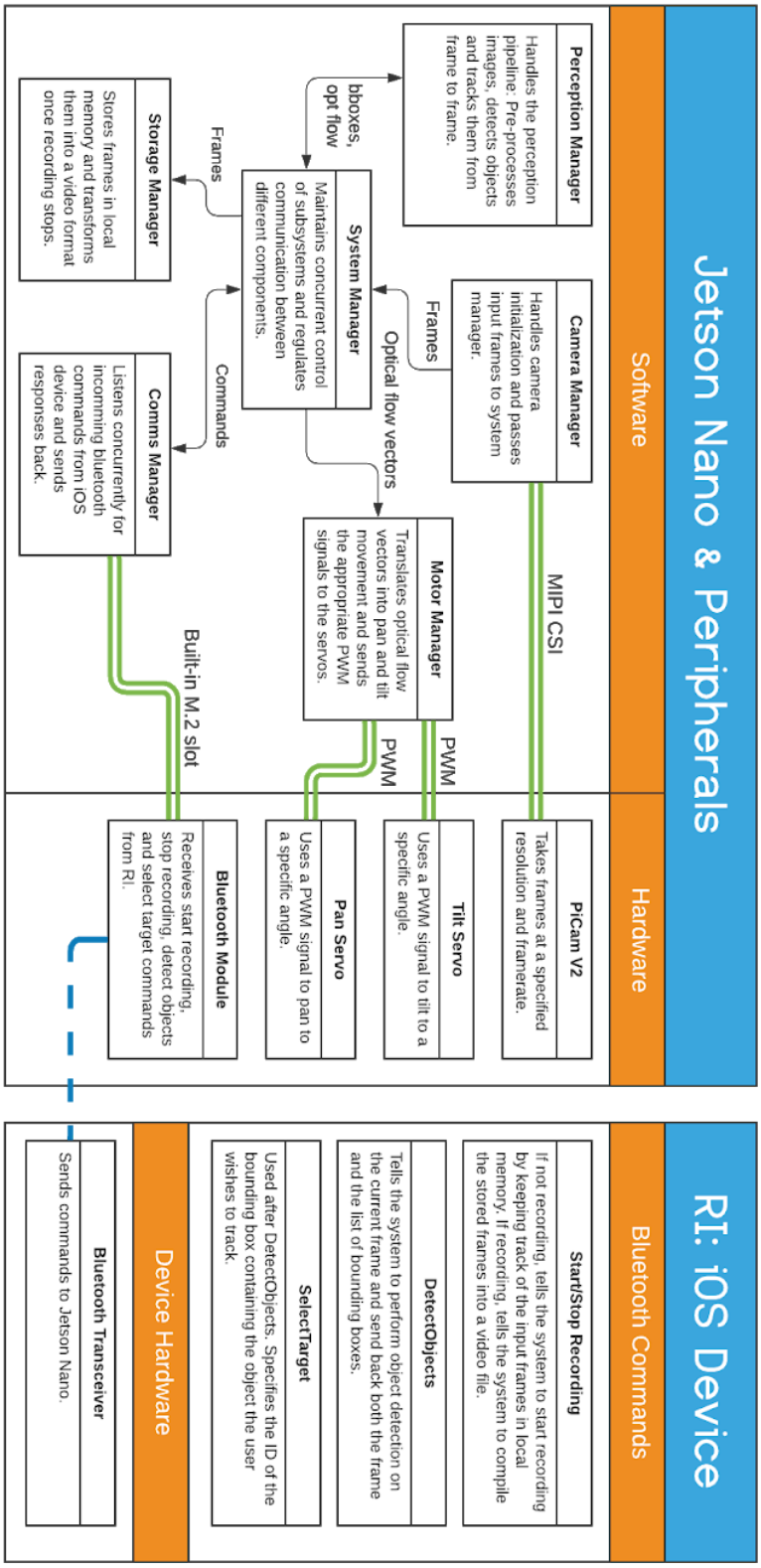
Figure 5: Rhino Arc II

9. REFERENCES

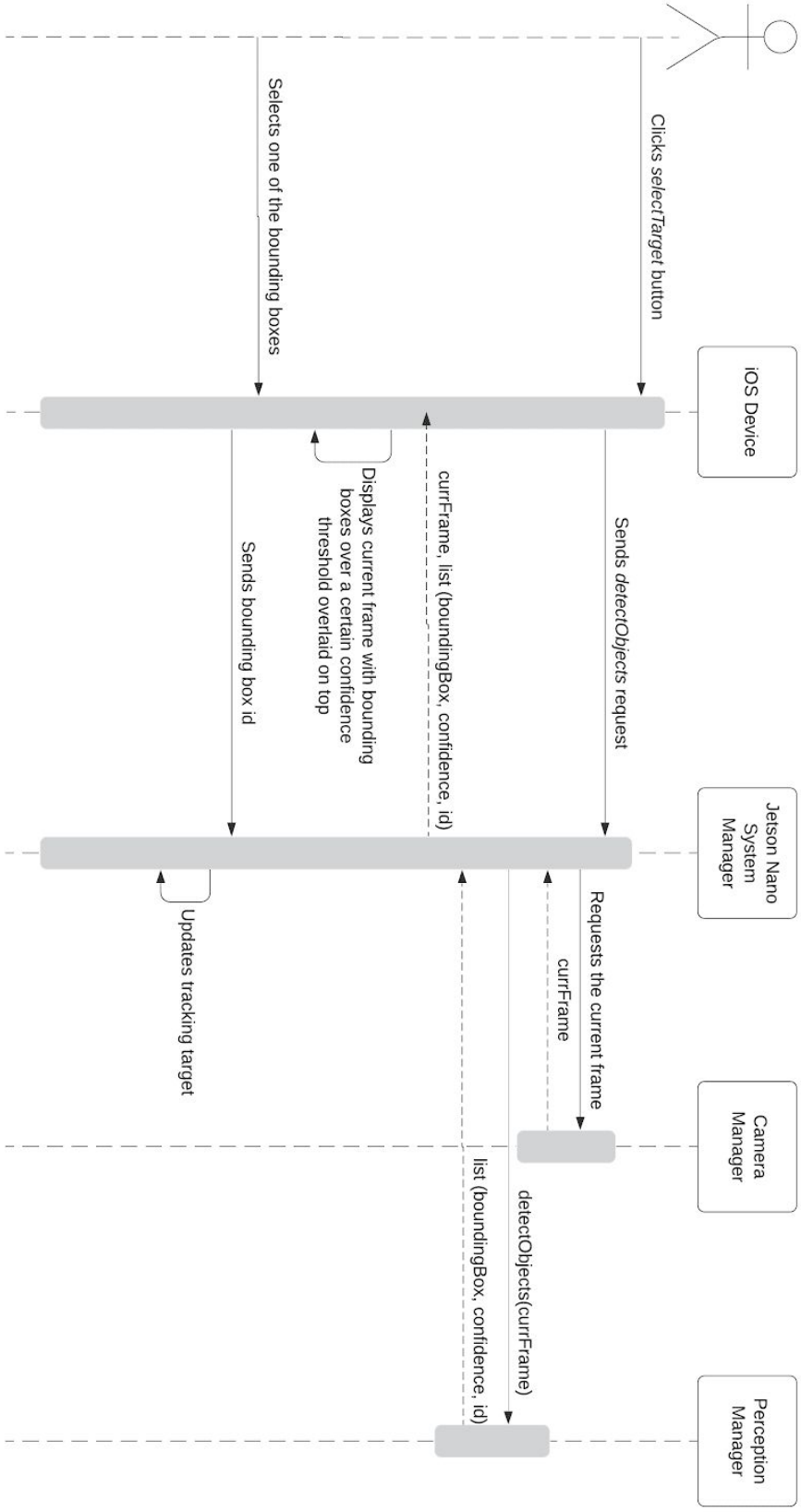
1. Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object Detection With Deep Learning: A Review. IEEE Transactions on Neural Networks and Learning Systems.
2. Demers, C. (2019, February 28). Input Lag of TVs. Retrieved from https://www.rtings.com/tv/tests/inputs/input-lag.
3. Jetson Nano: Deep Learning Inference Benchmarks. (2019, April 25). Retrieved from https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks.
4. Wang, C., Galoogahi, H. K., Lin, C.-H., & Lucey, S. (2018). Deep-LK for Efficient Adaptive Object Tracking. 2018 IEEE International Conference on Robotics and Automation (ICRA).

10. APPENDIX

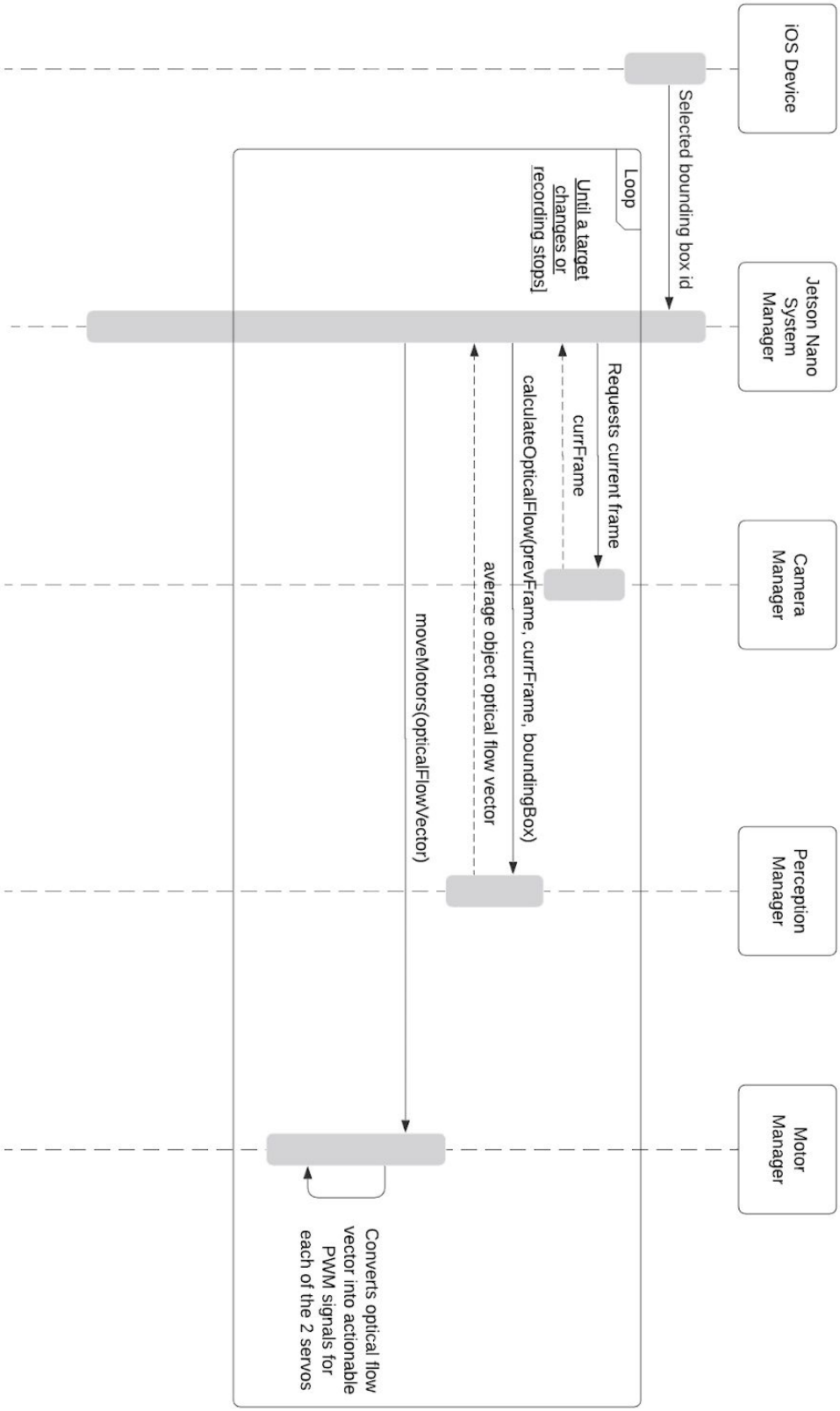
10.1 Appendix 1: System Communication Diagram



10.2 Appendix 2: Target Selection Sequence Diagram



10.3 Appendix 3: Target Tracking Sequence Diagram



10.4 Appendix 4: Team Gantt Chart for Project Management