InFrame: Design Document 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, Lucas-Kanade, Tracking, Embedded Systems.**

1. INTRODUCTION

AI-empowered 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 research group 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 relative success rates when 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 70% 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.

The following paper will extensively 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.

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 in under 1ms. 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 (FPS^{-1} seconds); furthermore, a period of 1ms was selected as an upper bound to represent a qualitative measurement of data transmission being "near instant." 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-design 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 uses cases, communication with this remote interface cannot be conducted through an online server since consistent WiFi connectivity cannot be relied upon outdoors.

To improve the user experience around setting up the device and initializing tracking, this design requires a maximum pair time between the camera system and its remote controller of up to 20 seconds; any amount of time beyond this would be too long to justify to users. Lastly, when the remote controller requests an object detected frame (e.g. an image capture with detected targets indicated / outlined), the frame data should reach and be displayed on the remote controller within 2 seconds, the upper bound of how long the system designers were willing to wait between requesting to select a target and actually being able to select one.

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, given a certain frame, the system should be able to detect the objects it was trained to detect with at least 70% accuracy (the lower bound of state-of-the-art object detection models).

2.2.2. Object Tracking

Using one of the bounding boxes resulting from object detection, the system needs to reliably and consistently track the object enclosed by said bounding box. Popular tracking methods such as Lucas Kanade tracking assume consistent lighting and small object displacements in between frames. As such, the requirement for the system's tracking capabilities is that in the absence of occlusions, object-shape-change and major lighting changes, a target should never be lost. Dealing with these cases is desirable, and this design will certainly strive to cover them by considering deep learning optimizations of these tracking algorithms, but they are outside of the scope of this project.

2.2.3. Use Case Coverage

InFrame is designed with two main use cases in mind, lecture recordings and action shots. As such, the computer vision pipeline should be able to support object detection and tracking in these cases. In order to do so, the system needs to be able to detect slow walking humans (i.e. lecturers) even as they turn around to face a blackboard, as well as fast moving entities such as soccer balls, skateboarders, and climbers. To achieve this, the system should use an object detection model trained on a dataset which includes these items as part of its classes.

In addition to the requirements outlined above, the time it takes to go from an input frame to a decision from the perception pipeline should be kept under 60 ms. This is because popular consumer electronics ratings agencies such as RTINGS.com claim 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. What this translates to is that the time it takes to take a frame, track the movement of an object with respect to the last frame and execute a motor command should all be kept below 60 ms[2].

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 certain set of requirements that need to be meant 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. At further distances the motors will be able to track targets moving faster. The motors must also be able to track the target anywhere around it for scenarios such as someone skateboarding. Therefore it is essential that it has continuous motion on the pan axis and be able to look above and behind itself within 180°. Meeting these speed requirements ensures that users will not be limited to 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.

2.3.3. Motion

As previously stated, continuous pan and a 180° tilt is necessary for an appropriate tracking range. The mechanical system must be able to support this without the potential for damage when the axes are moved to extreme angles. Therefore 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 must be easy to transport in a backpack. Access to the battery should allow for quick swapping. Like most photography equipment, it should be 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

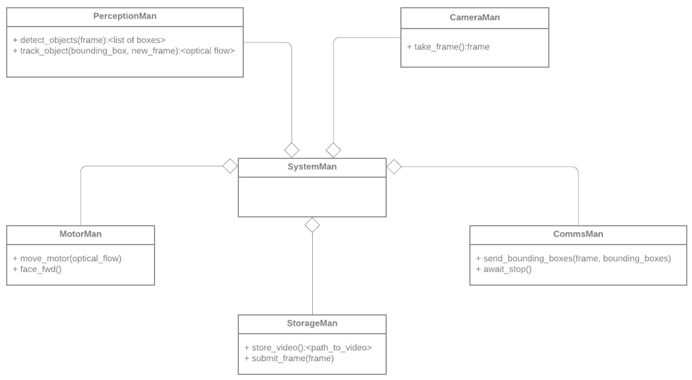

PerceptionMan
+ detect_objects(frame):<list of boxes>
+ track_object(bounding_box, new_frame):<optical flow>
CameraMan
+ take_frame():frame
SystemMan
MotorMan
+ move_motor(optical_flow)
+ face_fwd()
CommsMan
+ send_bounding_boxes(frame, bounding_boxes)
+ await_stop()
StorageMan
+ store_video():<path_to_video>
+ submit_frame(frame)

Figure 1: Software Interaction Diagram for Jetson CSM

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. To improve the overall speed of the system and allow for computer vision work to occur simultaneously with the rest of the interconnected system operations, the core system manager will be programmed using extensive concurrency features within Python, which will already be the language of choice for nearly all computer vision work and has more extensive embedded interaction libraries than other modern concurrency languages.

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 users request images, the system manager will first ping the camera manager to capture a frame, then 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. This path is on such potential user interaction story for the system; more such sequences will be outlined in further detail throughout Section 4.

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 so that the user can select a target and finally, a select target command to specify which target the system should start tracking.

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 2 below. However, it is worth noting that object detection will not be running in real-time, but only when a user wishes to change a tracking target (so that bounding boxes can be selected). As such, heavier networks that achieve greater accuracy such as YOLOv3 will also be considered. Because most of these models are trained using the COCO dataset, which includes labels that cover all of our use cases (people, sports balls, skateboards, frisbees, etc), it is expected that they would all achieve the same accuracy on our data that they report being able to achieve on their original reports.

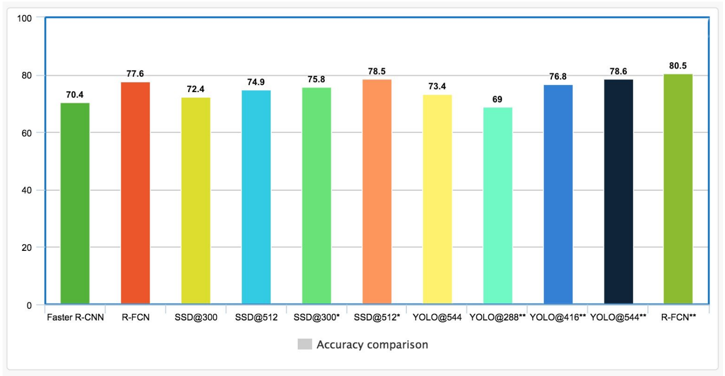


Figure 2: Accuracy comparison of state-of-the-art object detection models

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. In order to achieve this, InFrame extracts features from the object within the bounding box using Shi-Tomasi corner detection (wrapped under the goodFeaturesToTrack function of OpenCV). Then, it uses these points to calculate their optical flow using the Lucas-Kanade tracking method (wrapped under the calcOpticalFlowPyrLK function of OpenCV). It should be noted that Lucas-Kanade does not deal well with occlusions and changes of shape. As such, deep learning optimizations[4] of adaptive object tracking will also be considered to improve InFrame's tracking capabilities.

This design currently assumes that in the time that it takes for the user to select a target, the subject will not move (so that its location with respect to the bounding box that the user chooses based on the given frame does not change). This is of course not optimal and an attempt to perform multi-target tracking will be done for this period of time while the user chooses a target (i.e. track all the boxes with different IDs so that eventually the system just tracks the chosen box).

3.3. Hardware Design

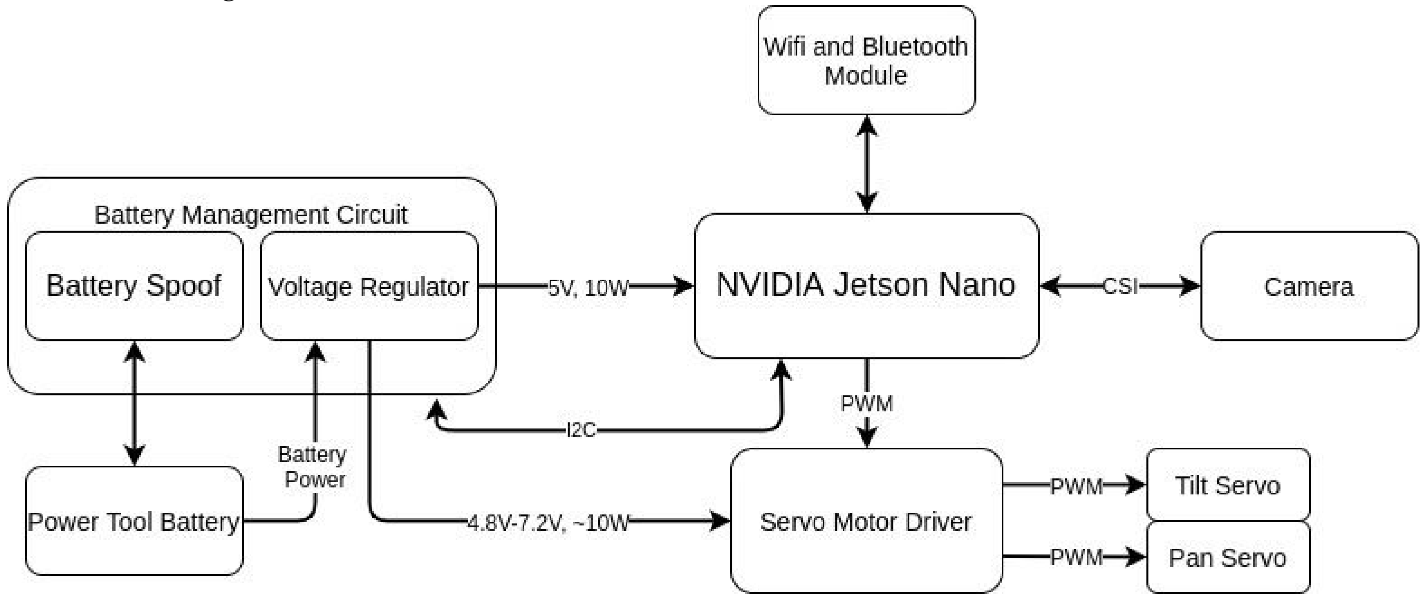


Figure 3: Hardware Interaction Diagram

3.3.1. Battery Management - Battery Spoof

A low power microcontroller which is able to perform authentication with the power tool battery and make it think it is connected to a tool instead of InFrame. This communication protocol will be observed from the battery and a power tool and then replicated by the spoofer.

3.3.2. Battery Management - Voltage Regulator

Once authentication is complete more power can be drawn from the battery. The microcontroller responsible for spoofing will connect the 5V power regulator to the Jetson over a MOSFET. At this point the Jetson can turn on the Servo Motor Driver by sending a I2C command back to the battery manager enabling it to turn on the switching power regulator for the servos. The purpose behind this is to allow for greater flexibility with how the servos are powered and enable easy testing of what power settings achieve the greatest performance.

3.3.3. 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.

3.3.4. Servo Motor Driver

This PCB 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.

3.3.5. 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.

3.3.6. 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.

3.3.7. 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 iOSDevice.

3.3.8. 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

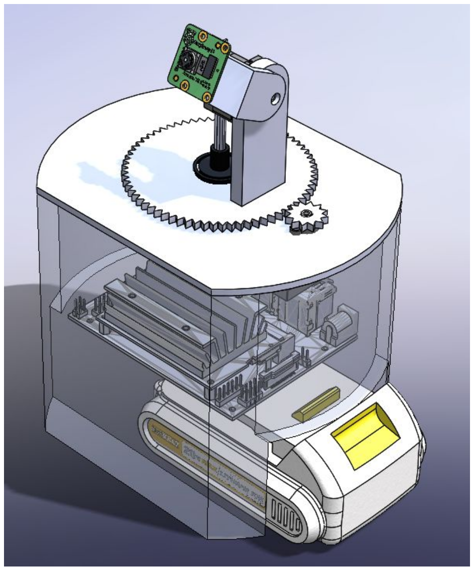


Figure 4: CAD Model of InFrame

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. So as to not tangle wires with continuous rotation there will be a slip-ring connecting the tilt servo and the camera connections to the rest of the system. 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”x6”x9”. 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 managers to actionable movement and InFrame can effectively track a subject in 3D space.

5. METRICS & VALIDATION

5.1 Subsystem Testing

5.1.1. Object Detection

In order to determine if object detection is functioning properly, a suite of testing scenarios designed around the use cases that InFrame is meant to support will be created. In each scenario, according to the system's requirements, a successful test would accurately detect at least 70% of the intended objects. In order to cover a lecture recording, static input frames with multiple people at distances of 5m, 10m and 15m from the camera will be fed into the perception pipeline. In order to cover a broad range of action shots, static input frames featuring skateboards, various types of sports balls, frisbees and cars will be used. As long as these objects are not very close together (since object detection has a threshold for how close these objects can be), success in this space means detecting at least 70% of these objects in a frame.

5.1.2. Object Tracking

Object tracking is a critical aspect of InFrame's functionality (its raison d'etre, if you will) and as such will be strictly tested. A test suite featuring fast moving objects will test the camera's input FPS so that displacements in between frames are small enough to be accurately tracked. In addition, these tests will test that the tracking speeds between frames are fast enough to keep up with the input FPS and achieve the 60 ms requirement that was mentioned in section 2.2.3. In addition, there will be tests where the target object changes a bit in shape as it moves (i.e. a climber on a rock wall) so that the tracking algorithm's generalization capabilities can cover that. Finally, a set of tests will test the system's capabilities to deal with occlusions and/or major lighting changes. It is not required to deal with these to a great extent, but these tests will inform the team of the different circumstances in which InFrame would not work well.

5.1.3. Battery Management Circuit

For the Battery Management Circuit to work properly it must first be able to authenticate the connection with the battery and then maintain it for an extended period of time. This subsystem must also be able to supply enough power for InFrame to function properly. After authentication has been proven to work two more tests will be conducted. First, a dummy load, similar to the highest power demand of InFrame, will be applied to the battery manger and monitored until the battery is drained to the safe minimum. It is important to do this test before connecting it to the Jetson to ensure that there will not be any unsafe behaviour that could damage the Jetson. During this period of time any voltage or power drop-off will be noted and software adjustments will be made to compensate for this. Once safety has been proven it will be possible to connect the battery manager to the rest of the system and measure the true battery life. At this point the tracking should be run on the system for as long as possible to get an accurate representation of what the battery life is.

5.1.4. Motor Control

Motor testing is twofold, firstly it must be proven that they can be controlled by the Jetson and powered through the battery manager. After this has been proven it is imperative that motor speed is tested. This will require the entire mechanical system to be complete to prove that each of the components can be moved at the required speed. Once fully assembled axes movement will be timed to ensure that they can move fast enough. Using the variable switching regulator on the Battery Management Circuit will allow for a performance comparison at different voltage levels.

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 will first be 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 a "Stop Recording" Bluetooth message on the Jetson-end will be tested while the various other subsystems are running on concurrent threads.

5.1.7. Image Transmission

With the size of a typical InFrame image being significantly greater than that of a small "Start," "Stop," or "Track ID:X" command, 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. If the image can reliably (over 5 repetitions) be received and displayed on the iPhone within a 2 second period, then this test will be considered successful.

5.2 Integration Testing

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 setup 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 setup 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

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. 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 User Interface/System Control

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.

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.

6.3 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.

6.4 Object Detection/Tracking

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 building and maintaining all system control, ranging from multithreading Jetson submodules to implementing the remote iOS user-interface.

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. It is likely that more parts will be necessary as further development is made but their cost will be but a fraction of the remaining funds. Replicating InFrame in the future would cost less since one time expenses such as the IOS developer account would not be necessary and because many of these parts are bought in multiples and could be used to make more than one system effectively amortizing the costs at scale.

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
Dev Sub	iOS Dev Sub	$100
Slip-ring	18 wire slip-ring	$19.26
3D Printing Material	Matte black plastic	$26.99
WiFi/Bluetooth Module	Intel 8265NGW	$30

Figure 5: Parts List & Overall Budget

7.4 Risk Management

7.4.1. Hardware Risks

The biggest risk factor within the hardware is the Battery Management Circuit. If for whatever reason it proves to be too difficult to spoof the authentication a fall back will be necessary. Fortunately, there is an off the shelf product that does the authentication with the battery and outputs power, Milwaukee 49-24-2371 M18 Power Source. This part's output will then need to get regulated into the necessary voltages for the system. Another potential risk factor is that the selected servo motors either wont be strong or fast enough to meet the requirements. This could easily be solved by purchasing higher power servo motors such as the B13 DLM.

7.4.2. System Control Risks

Although the Picamera, the system's primary image-recording device, is advertised as able to record up to 30FPS at 1080p and 60FPS at 720p, it may only be able to do so in video mode, restricting individual frame access until the end of a recording period when the frames are converted into an MP4. This limitation would mean that when taking single images and passing them through the CSM, a high enough effective FPS to support target tracking may not be achieved. For these reasons, if taking single images cannot be done fast enough, lower resolution images might have to be used instead or a faster camera module, such as the 16MP 4K MIPI Arducam Module supporting 720p120fps, might have to be purchased. Ideally, the latter would be chosen so high quality footage can still be recorded.

7.4.3. Perception Risks

The system's perception pipeline uses the lightest object detection model to maximize performance of the most computationally intensive task. If this model (SSD Mobilenet-V2) for 300x300 size images does not meet the accuracy requirements of above 70% on this design's intended use cases, the same model with bigger input images will be considered. If that still does not suffice, a heavier model such as YOLOv3 will be considered. A final contention plan if nothing seems to work well enough on the Jetson is to run the perception pipeline on an AWS EC2 instance. The approximate latency to the nearest AWS server is about 20 ms, which is well within the 60 ms upper bound for the CV latency that was outlined in the CV requirements section. Furthermore, if tracking itself does not work accurately due to minor lighting changes or slight occlusions having a big effect on performance, rather than using OpenCVs implementation of LK tracking, an implementation from scratch of a deep learning optimization of LK tracking will be used to improve performance.

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.

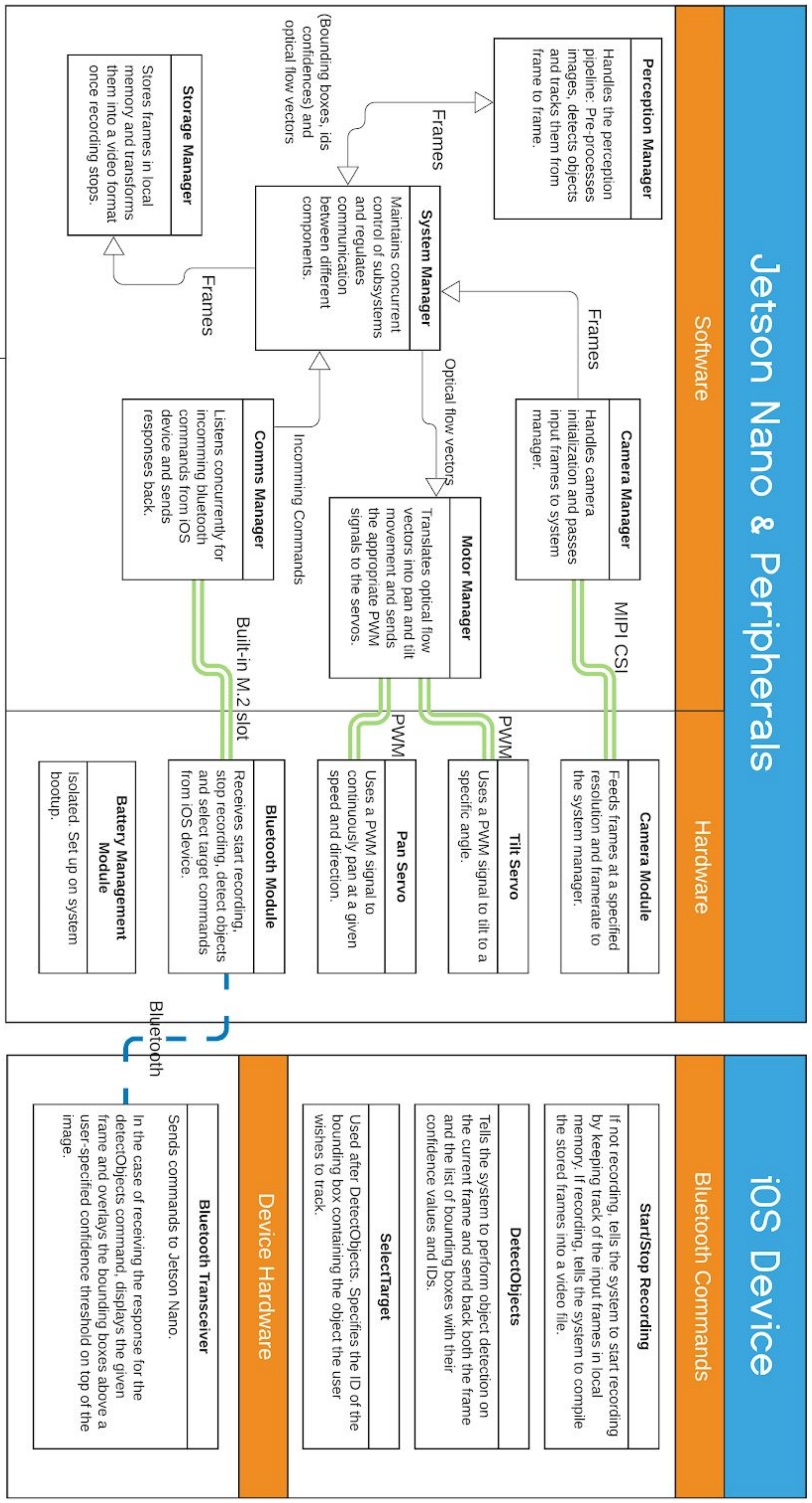
Figure 6: Rhino Arc II

9. REFERENCES

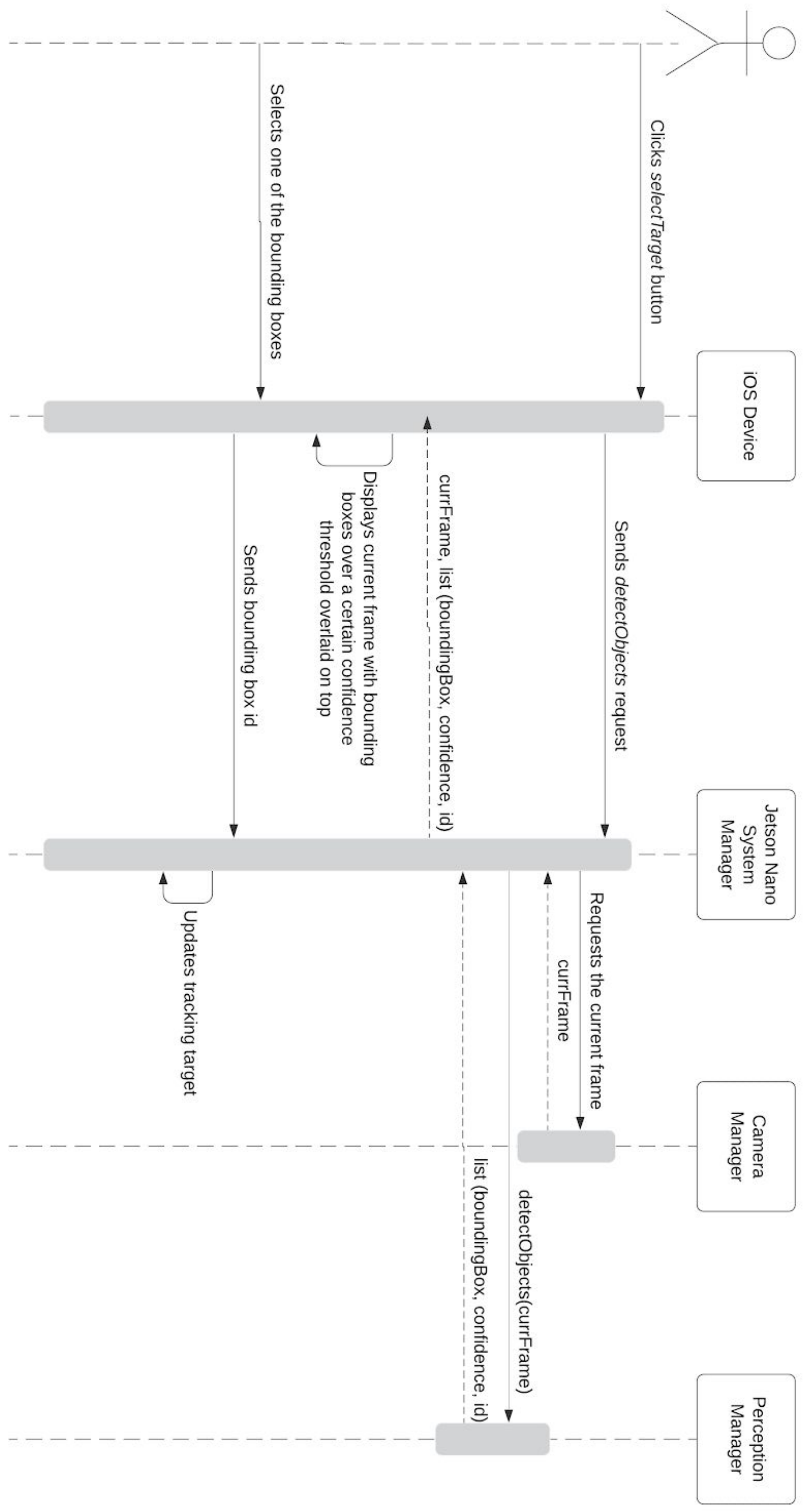
1. Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object Detection With Deep Learning: A Review. IEEE Transactions on Neural Networks and Learning Systems.
2. Demers, C. (2019, February 28). Input Lag of TVs. Retrieved from https://www.rtings.com/tv/tests/inputs/input-lag.
3. Jetson Nano: Deep Learning 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

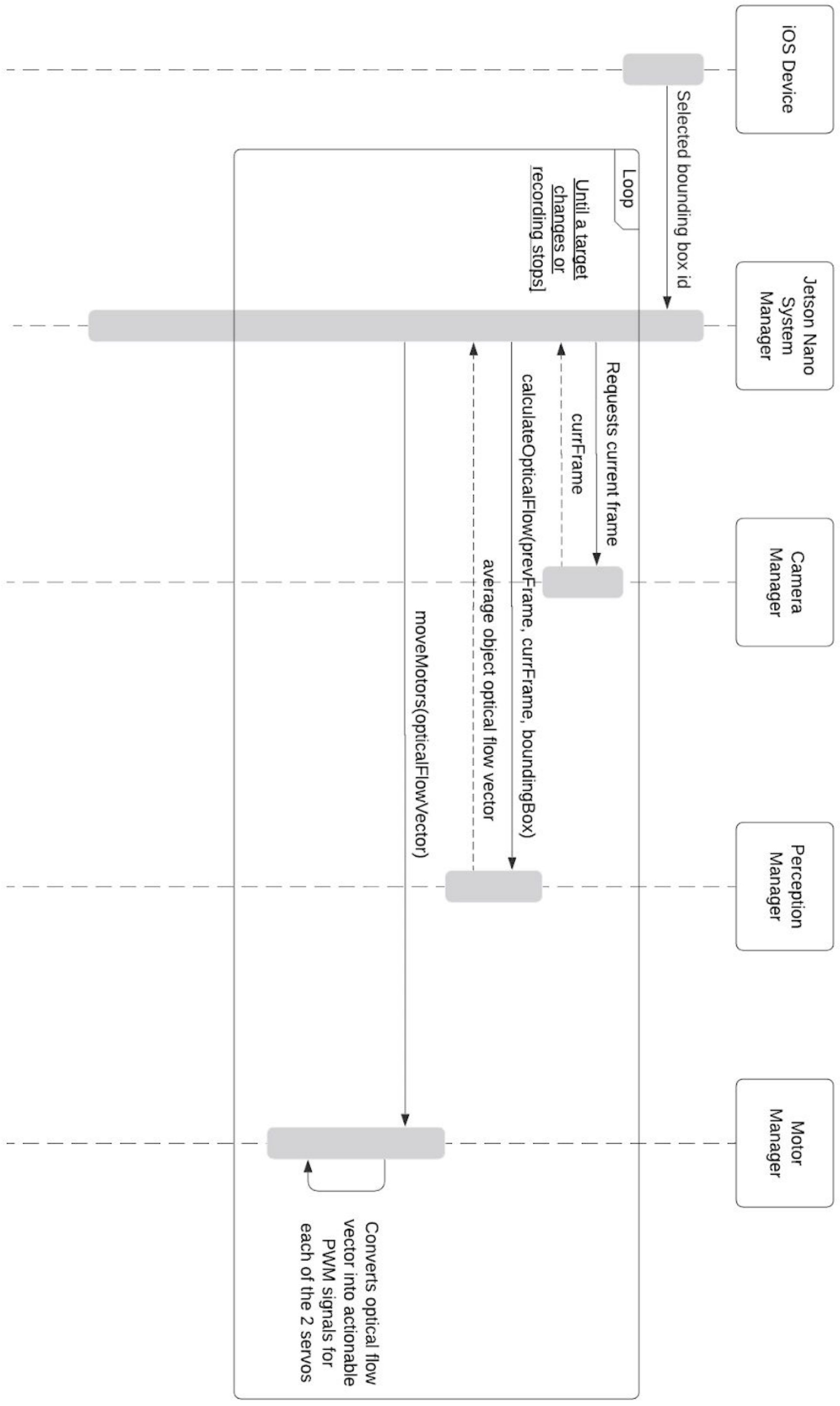
10.1 Appendix 1: System Communication Diagram


Jetson Nano & Peripherals
Software
Hardware
Perception Manager
Handles the perception pipeline: Pre-processes images, detects objects and tracks them from frame to frame.
Frames
(Bounding boxes, ids confidences) and optical flow vectors
Storage Manager
Stores frames in local memory and transforms them into a video format once recording stops.
System Manager
Maintains concurrent control of subsystems and regulates communication between different components.
Frames
Frames
Optical flow vectors
Incomming Commands
Camera Manager
Handles camera initialization and passes input frames to system manager.
Motor Manager
Translates optical flow vectors into pan and tilt movement and sends the appropriate PWM signals to the servos.
Comms Manager
Listens concurrently for incomming bluetooth commands from iOS device and sends responses back.
MIPI CSI
Built-in M.2 slot
PWM
PWM
Camera Module
Feeds frames at a specified resolution and framerate to the system manager.
Tilt Servo
Uses a PWM signal to tilt to a specific angle.
Pan Servo
Uses a PWM signal to continuously pan at a given speed and direction.
Bluetooth Module
Receives start recording, stop recording, detect objects and select target commands from iOS device.
Battery Management Module
Isolated. Set up on system bootup.
Bluetooth
iOS Device
Bluetooth Commands
Device Hardware
Start/Stop Recording
If not recording, tells the system to start recording by keeping track of the input frames in local memory. If recording, tells the system to compile the stored frames into a video file.
DetectObjects
Tells the system to perform object detection on the current frame and send back both the frame and the list of bounding boxes with their confidence values and IDs.
SelectTarget
Used after DetectObjects. Specifies the ID of the bounding box containing the object the user wishes to track.
Bluetooth Transceiver
Sends commands to Jetson Nano.
In the case of receiving the response for the detectObjects command, displays the given frame and overlays the bounding boxes above a user-specified confidence threshold on top of the image.

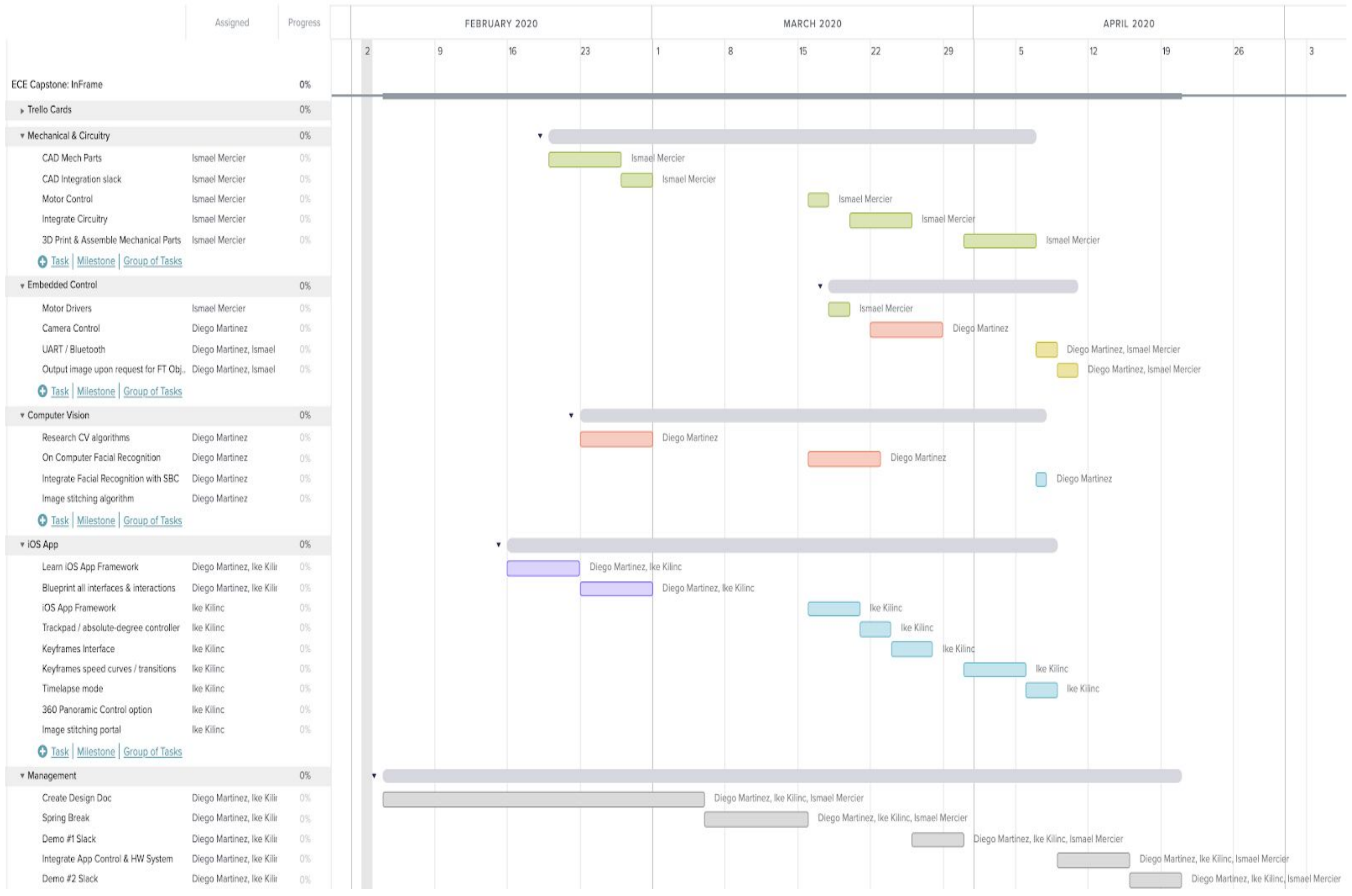
10.2 Appendix 2: Target Selection Sequence Diagram


iOS Device
Jetson Nano System Manager
Camera Manager
Perception Manager
Clicks *selectTarget* button
Sends *detectObjects* request
Requests the current frame
currFrame
detectObjects(currFrame)
list (boundingBox, confidence, id)
currFrame, list (boundingBox, confidence, id)
Displays current frame with bounding boxes over a certain confidence threshold overlaid on top
Selects one of the bounding boxes
Sends bounding box id
Updates tracking target

10.3 Appendix 3: Target Tracking Sequence Diagram


iOS Device
Jetson Nano System Manager
Camera Manager
Perception Manager
Motor Manager
Selected bounding box id
Loop
Until a target changes or recording stops]
Requests current frame
currFrame
calculateOpticalFlow(prevFrame, currFrame, boundingBox)
average object optical flow vector
moveMotors(opticalFlowVector)
Converts optical flow vector into actionable PWM signals for each of the 2 servos

10.4 Appendix 4: Team Gantt Chart for Project Management



	Assigned	Progress
ECE Capstone: InFrame		0%
▸ Trello Cards		0%
▾ Mechanical & Circuitry		0%
CAD Mech Parts	Ismael Mercier	0%
CAD Integration slack	Ismael Mercier	0%
Motor Control	Ismael Mercier	0%
Integrate Circuitry	Ismael Mercier	0%
3D Print & Assemble Mechanical Parts	Ismael Mercier	0%
Task | Milestone | Group of Tasks		
▾ Embedded Control		0%
Motor Drivers	Ismael Mercier	0%
Camera Control	Diego Martinez	0%
UART / Bluetooth	Diego Martinez, Ismael	0%
Output image upon request for FT Obj..	Diego Martinez, Ismael	0%
Task | Milestone | Group of Tasks		
▾ Computer Vision		0%
Research CV algorithms	Diego Martinez	0%
On Computer Facial Recognition	Diego Martinez	0%
Integrate Facial Recognition with SBC	Diego Martinez	0%
Image stitching algorithm	Diego Martinez	0%
Task | Milestone | Group of Tasks		
▾ iOS App		0%
Learn iOS App Framework	Diego Martinez, Ike Kilir	0%
Blueprint all interfaces & interactions	Diego Martinez, Ike Kilir	0%
iOS App Framework	Ike Kilinc	0%
Trackpad / absolute-degree controller	Ike Kilinc	0%
Keyframes Interface	Ike Kilinc	0%
Keyframes speed curves / transitions	Ike Kilinc	0%
Timelapse mode	Ike Kilinc	0%
360 Panoramic Control option	Ike Kilinc	0%
Image stitching portal	Ike Kilinc	0%
Task | Milestone | Group of Tasks		
▾ Management		0%
Create Design Doc	Diego Martinez, Ike Kilir	0%
Spring Break	Diego Martinez, Ike Kilir	0%
Demo #1 Slack	Diego Martinez, Ike Kilir	0%
Integrate App Control & HW System	Diego Martinez, Ike Kilir	0%
Demo #2 Slack	Diego Martinez, Ike Kilir	0%