IR MAN
Final Document
for ECE Capstone Project

Max Bai, Shirley Zhang, Jiaqi Zou
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—IR MAN, a universal IR Controller on a raspberry pi connected to remote web server, and also as an interactive humanoid robot with 2 degrees of freedom (DOF) motion, computer vision (CV) based object detection and IR control capabilities, is designed to leverage home appliance use experience. The Internet of things (IoT) element of this project is driven by the convergence of multiple technologies such as machine learning, cloud services, commodity sensors, and real time embedded systems, hence achieving the goal of remotely controlling your household appliances via the internet. The following sections of this paper elicit the motivation, architecture design, technical requirements and testing metrics of this particularly innovative approach to deploying this IoT Smart Home System (IoTSHS). Using computer vision based approach, the system (IR MAN) can effectively identify the exact location of the specified household appliances and plan motion accordingly to point the IR emitter directly at it before emitting its corresponding IR commands. The proposed IoTSHS will be designed, programmed, manufactured, integrated and tested in the format of specifications in this paper.**

I. Introduction

The ”Internet of things” (IoT) is becoming an increasingly growing topic of conversation in the entire tech industry. With Broadband Internet becoming more and more accessible, the cost of connectivity is dramatically decreasing and mobile devices like smartphones has sky-rocketed its penetration to the market. All of these things are creating a ”perfect storm” for the IoT. However, many of our current household Infrared (IR) controlled appliances are ”traditionally dumb” devices that are not internet capable. Thus, the problem arises when we aspire to come up with a scalable approach to empower these devices with IoT capabilities under affordable cost and extensive add-on values.

Have you ever ran into the problem of not being able to find that remote control lying somewhere under the couch to turn on your TV? Or too many remote controls and you don’t know which one correspond to which device? Have you imagined being able to turn on your AC just before you got home on a hot summer day? With so many domestic appliances controlled by IR remote control, we decided to create IR MAN to make our life easier by designing a smart IoT IR Control Hub that allows your smartphone to remotely control all the traditionally dumb or non-internet-enabled IR domestic appliances. The solution proposed in this paper, IR MAN, aims to solve the aforementioned problems such that the remote control is truly remote and all home appliances are automatically upgraded to IoT devices with minimal cost. And more importantly, IR MAN is an interactive humanoid robot that actively locates the household appliances in your house using computer vision (CV) and plans motions accordingly to reorient the IR emitter in order to accurately shoot the IR command signals at it like a real Iron Man.

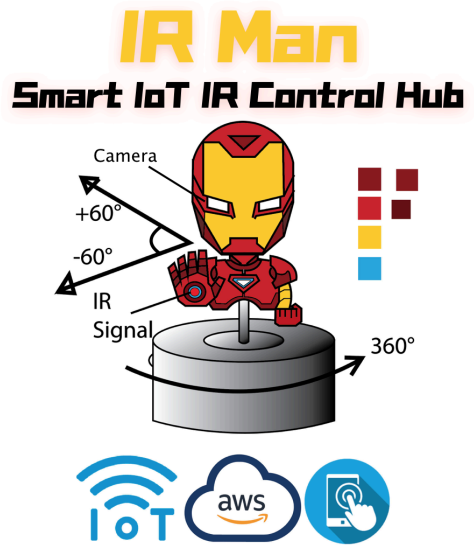

Fig. 1: Concept Design for IR Man

The out-of-the-box experience is like the following. When first logged into the webapp, the user is faced with option to either add more devices in the device list, or to calibrate the IR Man with existing (previously registered) devices. If the device has not been added before, the user can register a new device. Upon calibration, the IR Man is going to rotate 360 around the room while using computer vision to recognize and compute the location of each registered devices. Upon completion, IR Man uses that location data to command the motors to spin to the device and point the IR emitter at the IR device when a WebApp command is received. And the IR emitter will shoot out the IR signal received from WebApp to control the IR device in your home and listen for the next WebApp command. The flow chart in Fig 2 depicts an intuitive understanding of how the IR Man device work in a workflow manner.

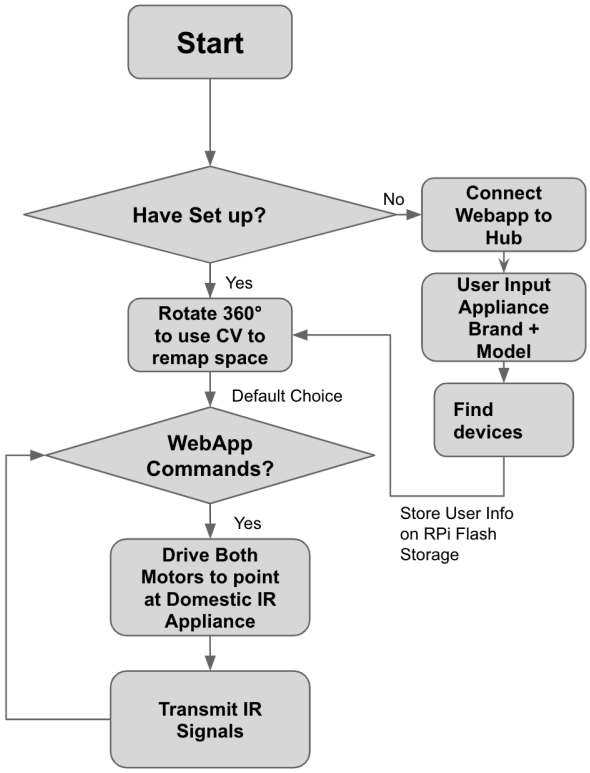

Fig. 2: Flow Chart IR Man

II. Design Requirements

A. Software Requirements

The main requirement of our product is latency and user experience: users need to be able to control IR devices successfully and quickly. With regard to this principle, we specify requirements for each components separately, and summarize them in a table attached below.

Component	Requirement
WebApp Latency	<500 (ms)
Device Locator Latency	<750 (ms)
Calibration Latency	<3 (mins)
Device Locator Accuray	>75%
Device Locator Output Position	± 10 degree error

One of the most crucial factor of user experience is the latency of IR MAN operation, which consists of the latency of inter-server communication and object detection pipeline. Therefore, the webapp to IR MAN messaging latency should be less than 500 (ms), which ensures that there is little to no delay between the time when user pressing a button on the webapp and the time when IR signal reaches the device. In addition,The latency between device locater server to IR MAN should be less than 750 (ms), considering the size of the images being transferred between these two platforms are of size (432, 768).

For the overall device locating pipeline, which includes the time for image transfer as well as object detection algorithm, we are aiming for less than 3 minutes, knowing that we plan to take 10 images for one round and the processing throughput of project detection model on CPU is around 1 image per second after heated to appropriate temperature, and that the first few images is going to take way longer.

Another feature that would affect user experience is the success rate of user operation, which depends on the device locator output. The accuracy of the resulting estimated device location needs to be within 10 degrees of the actual device location in order for the IR circuit to work. Moreover, for the object detection algorithm, we intend to achieve a validation accuracy of more than 75%.

B. Hardware Requirements

Component	Requirement
IR Circuit Success Rate	>90%
Motor Position Accuracy	± 10 degree error
Motor Rotation Time	<2 (s)

There are mainly two hardware components in our project: the IR circuit and the motors. For the IR circuit, we are aiming for greater than 90% success rate of controlling IR devices using our IR circuit under different circumstance, such as lighting and relative location of the IR device to the IR circuit.

For the motors, given a specific pose, the motors needs to be within 5 degrees of the correct pose for the success rate of the whole system. And for the sake of latency, time it takes from a random pose to reach the desired location should be under 2 seconds.

C. Overall Requirements

Component	Requirement
User Success Rate	>90%
User Latency	<3 (s)

In terms of the user experience of our whole system as a final product, we want to be able to achieve the following requirements. Time that it takes for any IR signal command received needs to be less than 3 seconds. And the average success rate of controlling all IR device needs to over 90%.

III. Architecture

A. Overall Architecture

The general design of the whole system is illustrated in the system architecture diagram in Fig 3. From the user end to the device end, we have the following major components:

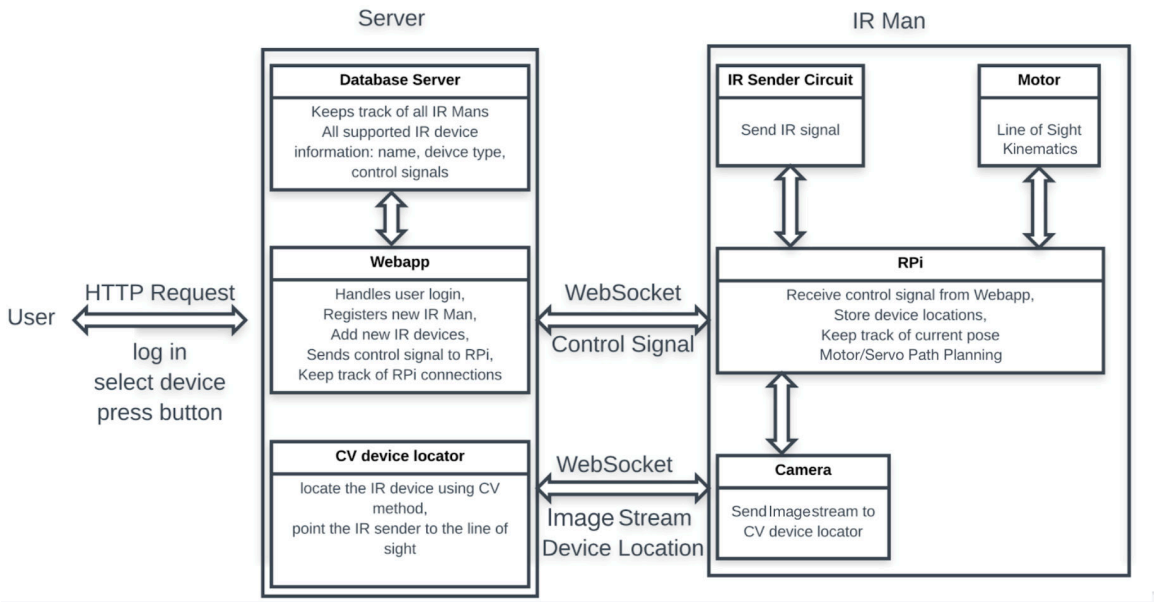

Fig. 3: System Architecture for IR Man

the webapp server which handles all the user interactions, the device locator server which handles computer vision inferences, and a Raspberry Pi that controls all the peripherals and websocket connections. Both of the two servers are running on the cloud. User can log into the webapp, find a list of all IR devices, and go to the control panel of a specific device and control the device just like a remote controller. The device locator server receives video stream from RPi and runs object detection algorithm as a backstage process. The Raspberry Pi will be interfacing with the camera modules, IR circuit and motors and motor controllers.

On the IR Man device, an RPi is used as the control hub for motor, camera and the IR Circuit. Upon first-time calibration, RPi sends frames from camera module to device locator server as the robot spins. The device locator gets back the corresponding device locations and RPi stores these locations locally. Upon a new button press, the RPi starts the motor to point to the specific device and then send the IR signal through the IR Circuit. The motor controls the pose of the robot and the IR Circuit sends any IR signal from RPi.

B. Operating Scenarios

The three major functionalities of IR Man are: (1) Calibration Command: let the IR Man scan the environment to locate IR devices. (2) IR Command: send an IR command to control a specific IR device. (3) Scenario Mode: user can send a predefined set of commands to IR Man such as ”quiet mode”, which mutes the TV and turns off the disco light.

To implement these three functionalities, the overall architecture consists of three major components: the webapp server, the device locator server and the client on the raspberry pi. The webapp handles all user interactions. The device locator server handles computer vision inference. The raspberry pi handles controlling the IR circuit, the motor and the servo. Both of the webapp server and the device locator server are running on AWS EC2.

Users access the system through the webapp server. Users can sign up and register an IR Man using an unique ID for each IR Man. Besides the three major functionalities described above, users can view all registered IR device and register new IR devices by the brand name and the device type.

The device locator server basically handles all the computer vision pipeline. It receives a set of images from RPi and a list of devices to look for. It outputs a device location map, which maps each device to its location relative to the IR Man: (θ_1, θ_2). θ_1 is the for degree for the base motor and θ_2 is the degree for the servo.

The raspberry pi receives commands from the webapp server and controls the IR Man to perform these commands. For calibration, it spins around to take pictures and sends these pictures to the device locator server to get the locations for each devices. For IR commands, it spins to the specific device location, the (θ_1, θ_2) pair, and send the IR signal. For scenario modes, it simply runs through the set of commands in similar manner.

C. Communication Protocol and Interfaces

There are several communication protocols between these three major components:

1) From webapp server to RPi: The webapp server sends three kinds of commands to RPi: calibration commands, IR commands and scenarios modes, as stated above. These commands are sent to RPi through websockets. Upon boot up, RPi sends a handshake message to the webapp server, which includes an unique ID for this IR Man. This ID is

also used when the user register the IR Man. When the user initiates a command, the webapp server finds the websockets connection by this unique ID and send the corresponding command content. The communication protocol includes an opcode to identify the command type, a timestamp and specific content of the request. For calibration, the opcode is 0, and the content is a list of all of the user's registered devices. For IR Commands, the opcode is 1, and the content is the device brand, device type and the button name. For scenario modes, each scenario mode corresponds to an unique opcode. These commands are sent to the RPi in JSON format.

2) Between RPi and device locator server: When user instruct the IR Man to search for device locations(calibration), RPi first asks the device locator server on AWS for transmission permission to ensure that no on-going process of the given IR MAN is running on the server. Upon permission, RPi sends 10 frame images to device locator server as following bundle: $(device\ \ ID, frame, frame\ \ number)$. This bundle identifies the specific IR MAN through device id and initializes data structure under this identity. The frame is sent in zip file in the format of numpy matrix for simplicity of processing. The frame number is used to calculate the corresponding position of this sent frame given that all frames are taken at a fixed degree with respect to the initial position. The calculated device positions are sent back to RPi as a dictionary wrapped in a json file. The keys of the dictionary are the device names, and the corresponding values are the device position stored in the format of (θ_1, θ_2), where θ_1 is the horizontal position of stepper motor and $theta_2$ is the position for servo motor.

3) Between RPi and peripheral devices: This is the embedded hardware communication pathway that connects all the peripheral devices of IR Man. The RPi connects to a camera, a motor controller that connects to a stepper motor, a servo, the IR circuit and (Human Input Devices) HID devices like monitor, keyboard and mouse. The camera is connected to the RPi through the RPi defined 16 pin CSI interface. The motor controller driver board is connected to RPi through the standard GPIO pins. The servo and IR circuit are connected to RPi through the PWM enabled GPIO pins. And finally, the HID devices are connected through HDMI and USB ports.

D. System Interaction

To understand how the system components interact with each other (Fig 4), we elaborate on three major functionalities of our system: (1) Calibration Command: user calibrate the IR Man to locate registered IR devices (2) IR Command: user presses a button on the control panel in the webapp to control the IR device. (3) Scenario Mode: a series of commands. We will examine how different system components interact with each other to achieve these three functionalities.

1) Calibration Command: Before user can use the IR Man to control any IR devices, the IR Man needs to find the device locations in the terms of (θ_1, θ_2) as the pose for the tow motors. This bootstrap step also needs to be done whenever the user moves the IR Man to a new position. When user presses the "calibration" button on the webapp, the webapp server sends the list of user's registered IR device to RPi to search for these devices in the surroundings. The RPi starts to spin the motor in several stops and send an image for every stop to the device locator server. The device locator server sends back specific location for each device to RPi. These locations are stored in a local file so that RPi "remembers" the pose for these devices for future uses.

2) IR Command: The user send the button press HTTP request to the webapp server. Upon receiving the request, the webapp server fetch the button information from the database server, which includes the device name, the device type, the protocol name, the button name, the button hexcode. The webapp server then find the user's IR Man device websocket connection by the IR Man device's unique ID. The button information is sent to the RPi over the websocket connection. Upon receiving the button information from webapp server, RPi extract the device name and find the predetermined device location, then it controls the motor module to move to correct pose for the device, so that the IR sender points to the device. Finally, RPi sends the control signal to the IR circuit.

3) Scenario Modes: There are four different scenario modes: quiet mode, leave home mode, party mode, and go home mode. Each mode consists of a list of different commands, that will be further explained in the system description section. Upon receiving the opcode for these modes, IR Man runs these IR commands in the same manner one by one.

IV. Design Trade Studies

A. MCU Trade Study

Since the MCU is the heart and brain of the IR Man, not only does it have to handle the controls of all the peripheral devices, it also needs to do a fair amount of data processing, as well as maintaining network connection with our Web server so that heavy duty computation tasks can be off-loaded to the cloud. Therefore, we did the following trade study between a Raspberry Pi, an Arduino Mega and a Lattice FPGA. All of the metrics are grouped into hard requirements and soft performances, where hard requirements are features that the MCU candidate has to meet in order to achieve our minimum feature requirements. Soft performance metrics are those that are good to have, such as better RAM configuration and support special Machine learning application on board. But those usually come at a cost of increasing budget. And refer to Fig 5 for the trade study analysis for our MCU selection.

B. IR Circuit Design Trade Study

A valid concerns has been raised during our design ideation and the peer review as we are aiming to achieve a 90%

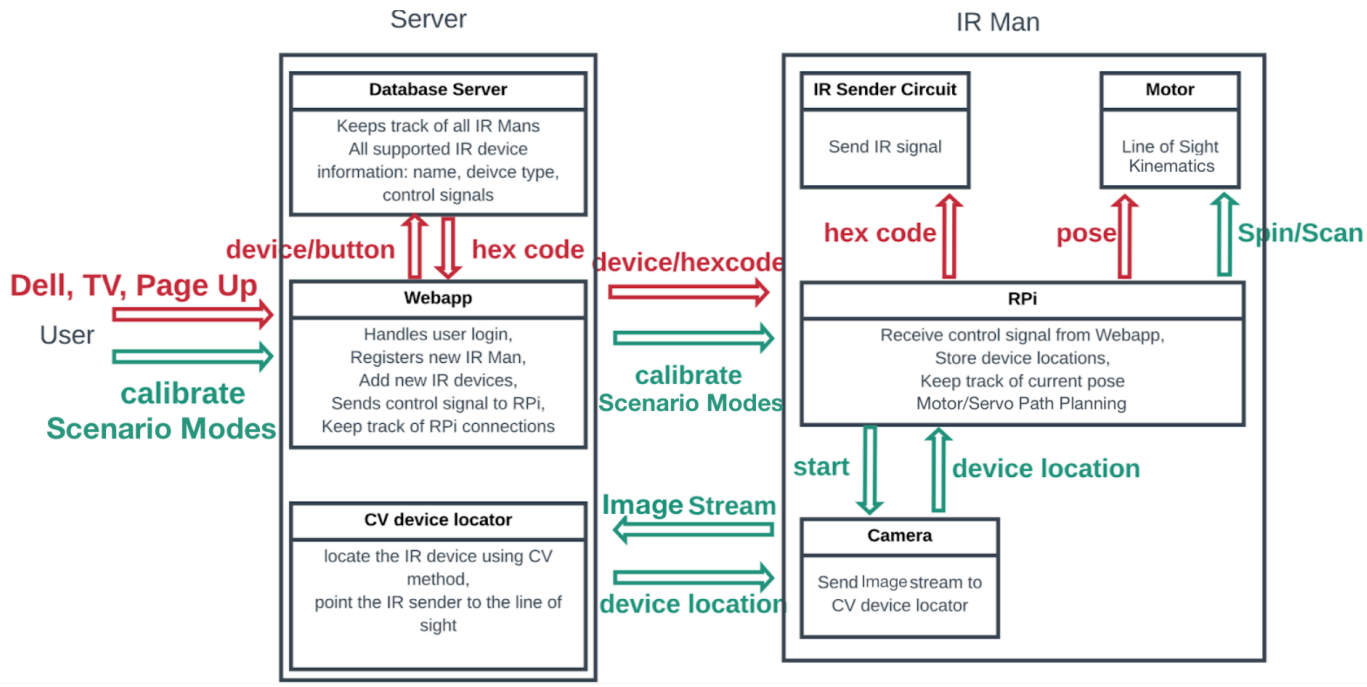

Fig. 4: System Interaction for IR Man

Component Alternatives		Main Computing Unit (MCU)			
Metrics	Scale	Raspberry Pi 3B+	Arduino Mega	Lattice FPGA	
Processor (Clock Speed & Addressing Mode)	1=slow clock speed 10=fast clock speed	7	3	10	Hard Requirements
Unit Price	1=Expensive 10=Cheap	6	9	1	Soft Performances
RAM (Size and Speed)	1=Low RAM 10=Big RAM	7	1	8	
I/O Accessibility (Frequently Used High Speed Interface)	1=Poor I/O selection 10=Rich I/O selection	9	3	9	
Programming Interface (Language and Environment Setup)	1=Hard to Program 10=Easy to Program	8	9	1	
Power Consumption	1=Very Power Impact 10=Fair Power Impact	5	10	8	
GPIO Accessibility (Current Draw & Total # of GPIOs)	1=Not convenient 10=Very convenient	7	8	7	
Advance Application Features (Supports CV and ML applications)	1=Not supported 10=Very supportive	8	3	10	
Network Connectivity	1=Not supported 10=Very supportive	10	1	5	
Total		67	47	59	

Fig. 5: MCU Trade Studies

accuracy for the IR circuit specifically. Some peers mentioned that having a ring of IR emitters to broadcast IR signals instead of having having the IR Man turn to the direction of a specific device. Although a ring of IR emitters would cut down system complexity by a lot, it is at risk of signal cross interference and uniqueness in design. Firstly, a lot of IR controlled devices of the same brand might be controlled with the same IR signal. For example, it might not be user's intention to turn on all of his Dell devices at the same time. Therefore, a ring of IR emitters broadcasting IR signals in all direction would cause unwanted behaviour of the system. Thus, we want to have the IR Man turn to the specific location of the IR device and pinpoint that control signal without affecting other IR devices. Another concern is that having the IR Man drive its motors based on computer vision is a unique solution to the problem and we want to keep it that way since it is an interesting engineering problem to solve with lots of added value like line of sight, product differentiation and interactivity.

C. webapp server to RPi connection

For the connection from the webapp server to various RPi, we chose to use websockets, because websockets enable the server and client to send messages to each other at any time, after a connection is established, without an explicit request by one or the other. This is crucial to our system because we are building a real time system: the webapp server could be sending requests to RPi at any time. In a challenge-response system, there is no way for clients(RPi) to know when new request is available for them. The only similar implementation is RPi polling the webapp server periodically, but that's not exactly what we want. Thus, this connection is established through websocket.

D. Camera Selection Trade Studies

We have also performed a trade study (Fig 6) on the camera selection since a huge portion of the feature is based on computer vision, thus making the choice of camera very important. When picking a camera, we first narrowed it down to camera modules that are supported by the MCU platform we are using, which is Raspberry Pi. Thus only cameras with CSI interface was considered. And specific hard requirements that we looked into for cameras are pixel density, video fps, field of view angle, and whether it supports night vision.

After comparing all three camera modules side by side, the result is pretty clear tha Dorhea camera wins. Mainly because it has an embedded photo-resistor sensor and can automatically switch between day vision and night vision. In addition, there are photosensitive infrared lights module that can be extended on Dorhea camera, which is huge plus especially because our IR Man is expected to work at all time of the day to better combat the variance in ambient lighting environment.

Component Alternatives	Camera			
Metrics	Scale	Raspberry Pi Camera Module V2 8 Megapixel1080p	Dorhea Raspberry Pi Camera Module Automatic IR Switching Day/Night Vision	Raspberry Pi Camera Module 5MP Wide Angle 160 Degree Fisheye Lens Camera OV5647	
Pixel Density for still Pictures	1=poor resolution 10=good resolution	8	7	2	Hard Requirements
Unit Price	1=Expensive 10=Cheap	9	8	7	Soft Performances
Video Frame Rates	1=Low fps 10=High fps	10	7	7	
Field of View	1=Poor FOV 10=Wide FOV	7	5	9	
Night Vision	1=Poor vision when dark 10=Good vision when dark	6	10	3	
Flexible Focal Length	1=Not adjustible 10=Fully adjustible	1	10	1	
Camera Extension Modules (Allows extra sensors to add on to current camera module)	1=Not extendible 10=Very modular	5	10	5	
Total		46	57	34	

Fig. 6: Camera Trade Studies

E. Image Transmission from RPi to device locator

In order to find the device locations, IR Man needs to look for these devices in its surroundings. Our solution is to let IR Man spin 10 steps, which has a gap of 36 degrees between per step. Before each step, the IR Man stop for a moment to take one photo and send it over to the device locator server. We decided to send 10 images from RPi to device locator instead of sending a complete video stream over the concern of latency, given that user experience is what we really care about this product. The number of 10 is carefully picked after we measured the field of view of the camera we chose. Given that the camera has a horizontal view of approximately 45 degrees, turning 10 times made sure that for each device, there will at least be 1 image that completely captures it. The reason that we are not taking more images for the sake of accuracy is that image inference through the object detection model is a main source of latency.

F. Webapp Server and Device Locator Server

Instead of using one server to handle everything, we decided to have two servers: the webapp server and the device locator server. We choose to separate these two servers because they have fundamentally different functions and there is no communication between these two servers according to our system design. It is also more elegant in that the two servers are running in different environments. This is also a more scalable design: once the system grows larger, we can just scale these two kinds of servers separately.

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instance types comprise varying combinations of CPU, memory, storage, and networking capacity and give us the flexibility in choosing the appropriate computation resources for our IR Man. Each instance type includes one or more instance sizes, allowing us to scale your resources to the requirements of your target workload. And please refer to Fig 7 for our analysis for server selections.

We adopted a t2.micro since Amazon has a free first 750 hours of usage and the performance of t2.micro is great for hosting a web application. We have tested the network latency of t2.micro server and the latency stabilizes at around 20ms, which is also acceptable for our application. We have also identified g4dn.xlarge, which is a GPU server that will allow us to do computer vision, image training. Consider that we are only going to train our model for so many times, the GPU server is not going to be up for the entire time and we use it on-demand when training is required. So the value of a GPU instance is worth its price.

G. Front End User Interface

We choose to develop our front end user interface on a webapp instead of an iOS or Android app because we want to have a platform independent solution. Although a webapp possesses a lot of limitation in terms of performance and feature access through user's native device, and might not provide the best possible user experience, we still choose this to be our approach for the final front end user interface because it is relatively easy and fast to deploy. Given the limited time and resources we have for this project, the best way, under our careful consideration, to support all the user features for the largest audience is through a webapp.

H. Robotics Motion & Dynamics

We decided to incorporate the robotics motion and dynamics feature in this project early on in ideation phase mainly for two reasons. There has been found lot of universal IR devices on the consumer market, and when we were asked the question, "what makes your project different from that of the others", we immediately delved into a brainstorm process for product differentiation. Since our project has to do with IR signals, and all of the team members are Marvel's fan, we thought of using Iron Man's figurine as our chassis and re-engineer Iron Man's gauntlet glove into a IR emitter device, and call it the "IR MAN" instead. The clever pun led us into an elaboration of robotics motion and dynamics. Some peers have questioned the purpose and given us feedback about the robotics part. There are indeed many ways to place the IR diodes, namely put a ring of IR diodes on the chassis to send out the same IR signal in all directions, but we decided to put it in the center of IR MAN's gauntlet and plan motions accordingly to aim the IR emitter directly at the target device. This would solve the line of sight issue, as well as reducing power consumption of the system. This robotics approach adds another layer of computer vision and dynamics control complexity to the project. However, not only does it make IR MAN unique, but also makes it more fun to interact with. In addition, the decision to use a servo as opposed to a second stepper motor is because of weight concerns. The servo is relatively light and can be easily mounted on the body of IR Man. This approach we used proved to be realistic in the end as we happened to deal with IR devices which shares the same IR communication codes. By implementing this seemingly over-complicated design, we were able to solve this issue by deliberately pointing to the exact device which can not be achieved by normal IR hub design.

Component Alternatives	AWS EC2 Instance					
Instance	t2.nano	t2.micro	t2.small	t2.medium	t2.large	g4dn.xlarge (GPU)
vCPU*	1	1	1	2	2	4
CPU Credits / hour	3	6	12	24	36	N/A
Mem (GiB)	0.5	1	2	4	8	125 GB NVMe SSD
Storage	EBS-Only	EBS-Only	EBS-Only	EBS-Only	EBS-Only	EBS-Only
Network Performance	Low	Low to Moderate	Low to Moderate	Low to Moderate	ow to Moderat	High
Price	$0.0058/hr	Free for 750hrs $0.0116/hr	$0.023	$0.0464	$0.0928	0.526/hr

Fig. 7: AWS EC2 Instance Trade Studies

Regarding this design, we performed a trade study (Fig 8) of different kinds of motors to achieve our goal of robot kinematics and dynamics system.

Component Alternatives		Motor			
Metrics	Scale	DC 6V Gear Motor with Long M355MM Lead Screw Thread Output Shaft 30/60/100/150/200/300/400/500RPM(6V 200RPM)	NEMA17 Stepper Motor High Torque Bipolar DC Step Motor Kit by MOTOU	Usongshine Nema 23 CNC Stepper Motor 2.8A 56N.cm Nema 57 Stepper Motor CNC Stepping Motor for DIY CNC Mill (23HS5628-8mm)	
Torque	1=poor torque 10=high torque	5	8	10	Hard Requirements
Power Requirement	10=Low 1=High	10	8	7	Soft Performances
Interface Difficulty	10=Easy 1=Difficult	6	10	8	
Price	1=Expensive 10=Cheap	10	8	6	
Size/weight	1=Huge/heavy 10=Small/light	10	9	6	
Total		41	43	37	

Fig. 8: Motor Trade Studies

I. Image Pre-processing

Knowing that the input of camera feed is fixed at 1080p (1920×1080 pixels), and that the input for image inference does not require resolution at this level, we decide to divide the image pre-processing phase into 2 sections, and perform them on different platform. Pre-process of a frame before inference typically consists of resizing the image and padding the rims of the image into a square. The previous phase shrinks the size of an image while the latter part expands the size by deliberately changing the image into a square to fit the fixed input size of the model. Therefore, knowing that all frames are to be transferred from RPi to a remote server, we decide to run image resizing on RPi before image transfer, and image padding on device locator server, such that the image to be transferred is in its smallest size, and that the transferred speed is maximized under limited network bandwidth. The size of transferred image is fixed at (432, 768) in the end given the limit of websocket's buffer size.

J. Computer Vision & Neural Network Framework

Due to the limited computing power on the RPi, we decided to run our computer vision module on a remote server. The object detector model we chose is YOLOv3. Reason that we pick this model out of many others is that it balances accuracy and inference fps better than the other popular models.

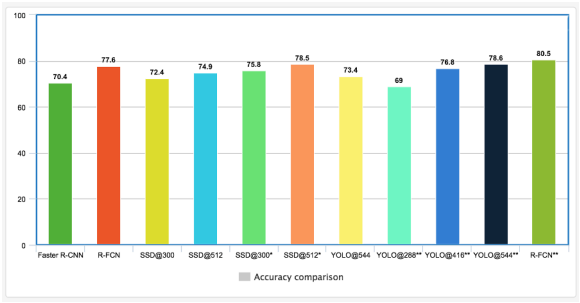

Fig. 9: Accuracy of popular models compared upon PASCAL VOC 2007 and 2012

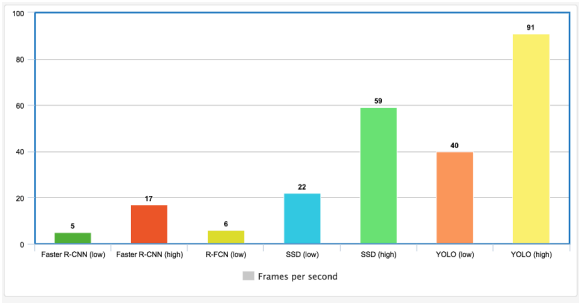

Fig. 10: Inference FPS of popular models compared upon PASCAL VOC 2007 and 2012

It is shown in the graph that validation accuracy doesn't deviate much between models, but YOLOv3 has a large space between its upper and lower limit of inference speed, which means we would be able to tune our image size and control the trade off between speed and accuracy. To enhance the accuracy of the model, we tried different approaches for dataset collection and labeling, and the method we used in the end was to only take photos that record the IR receiver side of the device, and only label the recognizable part that carries the IR receiver rather than the full device. This approach significantly improved the converging speed and accuracy of our model, and made sure that the calculated center of device is always subject to the position of IR receiver.

V. System Description

A. Device Locator Implementation

1) CV Object Detection Model: The object detection model framework, as mentioned before, was YOLOV3 trained from the weights of Darknet. Initially we were collecting dataset with no control of the facets of device being included in the image. This crude approach turned out to be a failure, especially for devices like AC which has multiple surfaces with different attributes: the model would not converge during training. After carefully studying the characteristics of the image to be sent from RPi to device locator server, we found that these images only include the front of the device where the IR receiver locates. We therefore modified our training dataset such that our model only cared about the region where IR receiver resides.

2) Data Post-processing & Location Parsing: This step is to be performed only after all frames had been fed into the neural network model and output data collected. Since each frame has its corresponding motor positions(θ) transferred together, we can find the median value of those motor positions where a specific device (outputs with the same tag) shows up, and can take it as the correct orientation for this device. Note that we will only take outputs with confidence bigger than 0.5 into calculation. This threshold is often chosen as a standard approach in industry. Refer to Figure 12 for the image post-processing pipeline.

3) CV Device Location Calculation: Since the camera is mounted on a base rotating about Z-axis, in the calibration phase, the camera is going to rotate a 360° and take finite amount of pictures at each stop. Let the number of pictures taken be N. Granted that the camera rotates in clockwise, each image $m_i, \forall i \in N$ will cover $\frac{360^\circ}{N} \cdot i$ field of view region.

On the location locator server, each image m_i has bounding boxes around the target devices. We first find the centroid of the bounding box using our CV pipeline. C_x, C_y are the centroid to the polygon with vertices $(x_0, y_0), (x_1, y_1), ..., (x_{n1}, y_{n1})$.

$$C_{\mathrm{x}} = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1}) (x_i y_{i+1} - x_{i+1} y_i)$$

$$C_{\mathrm{y}} = \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1}) (x_i y_{i+1} - x_{i+1} y_i)$$

where A is

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i)$$

Next, we calculate the relative position of the centroid (C_x, C_y) to the image m_i. Since we know that m_i will represent the range of FOV between $\left[\frac{360^\circ}{N} \cdot i, \frac{360^\circ}{N} \cdot (i+1)\right)$. The absolute location in θ that correspond to the target device is then

$$\theta = \frac{360^\circ}{N} \cdot i + C_x$$

If our hyper-parameter N, the number of stops (number of photos) for calibration is granular, then multiple photos can have the same target device in the FOV. In this case, we can take the sum average of their centroids to produce a more accurate result reflecting the absolute location of that device.

In the case where multiple target devices appear in the same FOV, we simply run the aforementioned find centroid algorithm in parallel to find the device location concurrently as shown in Fig 11

4) Complete CV Pipeline: The complete design of the device locator server consists of 2 threads: one handles the connection over websocket(connection thread) and the other handles the object detection algorithm(CV thread).
With the server being run, both threads will be initiated before handling any connection request. Upon a call of calibration, the connection thread firsts asynchronously receives the permission request from RPi. The connection thread checks whether there is any ongoing calibration running for this IRMan on server, and if there is not, connection thread sends the permission back to RPi. RPi, receiving permission, will start transmitting 10 image packets over websockets to connection thread. Connection thread un-zip them and puts them separately into a image processing queue for CV thread to retrieve image from.

Fig. 11: Computer Vision Object Detection Result

The CV thread, running in back ground and constantly checking the content of image-processing queue, retrieves image from the processing queue and inputs the image through the object detection model. The output of each image contains the detected device on image and it location on image. This information then goes through the device location calculation described above and the result location is stored into a dictionary. CV thread actively keep track of the number of images being inferred for each calibration command, and if the output of all calibration frames (10 in our design) have been collected, it then push the assembled result in to the result queue, which is to be handled by the connection thread.

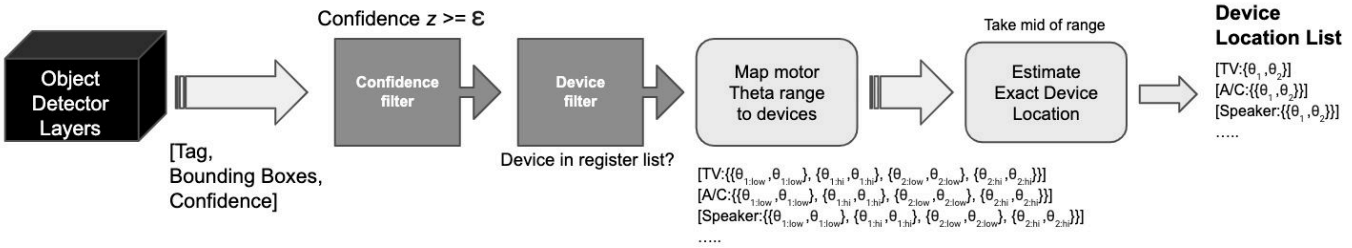

Fig. 12: Image Post-processing Pipeline

The connection thread, while asynchronously waiting for incoming messages from RPi, also actively cheking the content of result queue. These two coroutines are managed by the python async library. If the result queue is not empty, the connection thread process the content and compose them into a result package which is to be sent back to the corresponding RPi as the calibration result. The package consists of a dictionary, with the keys being the device names and the values being the device locations.

B. Webapp Implementation & Deployment

The webapp server handles all user interaction. We need to handler user authentication, adding new devices, choosing devices and button press on specific devices. The routes and their functionalities are listed in the table below:

ROUTE	METHOD	USAGE
/devices	GET	list of registered IR devices
/devices/new	GET/POST	register new devices
/calibrate	GET	send calibration command to IR Man
/devices/:device	GET	show the controller for a specific device
/devices/:device/:button	POST	send IR command to IR Man
/register	POST/GET	register new user and IR Man
/login	POST/GET	user authentication
/logout	GET	user logging out
/	GET	landing page

For the database we have the following schemas stored: User, UserDevice and Device. The User object keeps track of its own IR Man device by an unique ID. The Device object keep tracks of the device information: a list of buttons with their names and its specific signal encoded in hexcode. The UserDevice object keeps track of its nickname and the base Device object as a reference. Each User object has a list of userDevice objects. For example, 5 users might be using the same Dell TV, and there is only one dell TV Device instance to keep track of all the Dell TV's buttons. The 5 users have 5 different UserDevice instances that represents each user's dell TV.

The webapp also keeps track of all connections with the IR Man devices in a hashmap, which maps each IR Man's ID to its websocket connection. Upon sending requests to a certain user's IR Man, we look for the user's IR Man ID in the map and send the request to the corresponding connection.

As for implementation, we used Express.js and Node.js as the web framework and MongoDB as the database server. For front-end, since we don't have dynamic content, we simply used HTML/CSS and ejs. For user authentication, we used Passport.js to handle user registration and logging in/out. The webapp server is deployed to AWS EC2. We wrote all the routing logic, front-end inter- faces, back-end logic, the communication protocl to RPi by ourselves.

The interaction between the user, the webapp and the RPi is now of a more complete picture described in Fig 13. Our front-end user interface looks like the Fig 14, which is the displaying the all the devices for an user. The authenticated user can also control registered devices on this portal.

C. RPi Embedded Software & Hardware

The RPi client controls the IR Circuit, servo, motor and the camera. Please refer to the schematics diagram in Fig 15 for all electrical connections.

1) Camera Control: The camera module we adopted is Dorhea Raspberry Pi Camera Module . The connection to RPi is very straight forward, which is to connect the 15 pin FPC directly with the dedicated CAM socket on Raspberry Pi. It attaches to Pi by way of one of the small sockets on the board upper surface and uses the dedicated CSI interface, designed especially for interfacing to cameras. With 8 megapixels of still resolution and 1080p30, 720p60 and 480p90 for videos. Its horizontal field of view of 62.2 degrees and vertical field of view of 48.8 degrees will allow a better accuracy for our device locator algorithm since the accuracy was previously

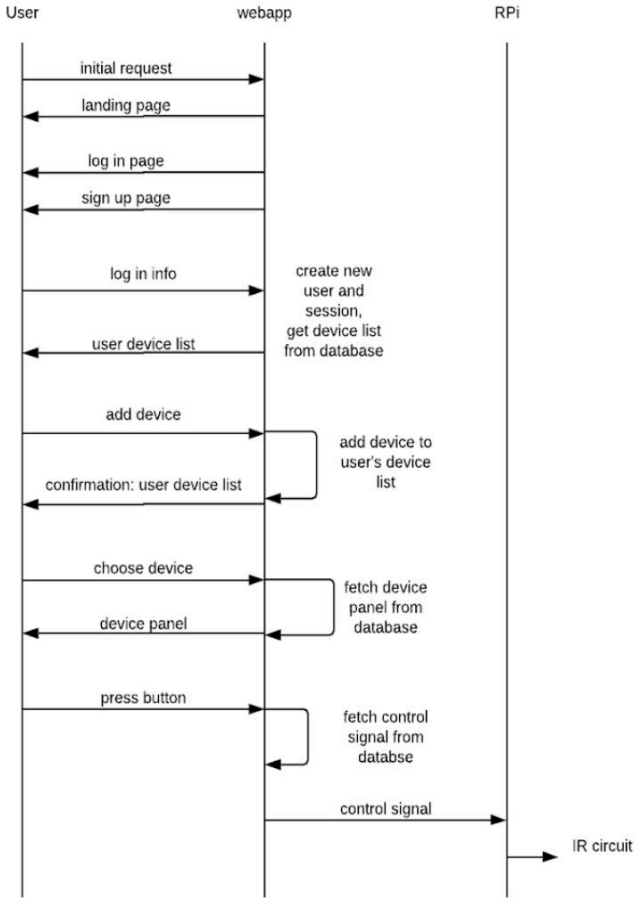

Fig. 13: User Interaction Diagram for the webapp

limited by the number of frames provided, dictated by the network bandwidth. However, with wider field of view, the possibility of target objects appearing in each frame is larger, thus reducing the amount of frames to be transmitted over the network.

2) IR circuit control: The IR circuit is composed of IR LEDs and MOSFETs. The MOSFET is a voltage controlled current sources that dictates the I_D that flows through it. The ON and OFF of the MOSFET is controlled by one of the programmed GPIO pins on Raspberry Pi. The specific ON/OFF PWM signal is generated using an open source IR signal database called LIRC that collected almost all brand's IR signal protocols. The protocols are enclosed in some hex codes representing bitmaps of ON/OFF levels in a given duty cycle. This database works with LIRC [1] (Linux Infrared Remote Control) package on RPi, which is capable of wrapping up the above process with command line arguments. We will be using an Open source IR remote control library: LIRC to controls the IR circuit. The IR Signals can also be retrieved from this library.

3) Motor and Servo Control: In order to support the required motion of IR MAN, we need our RPi to control both a stepper motor and a servo. After the aforementioned

Fig. 14: WebApp User Interface

computer vision based device locator algorithm returns the device location in terms of (θ_1, θ_2). θ_1 represents the target position of the motor in the chassis base. θ_2 represents the target position of the motor in the arm of IR MAN. Our RPi will then drive the motor/servo to the target position specified by (θ_1, θ_2). To control the stepper motor from RPi, we acquired a separate controller board called "Easy Driver" from SparkFun. The driver is essentially an H-bridge that controls the spin direction and step size for the stepper motor's voltage level using the following input pins.

- MS1 – Logic Input for the stepper step size. See truth table below.
- MS2 – Logic Input for the stepper step size. See truth table below.

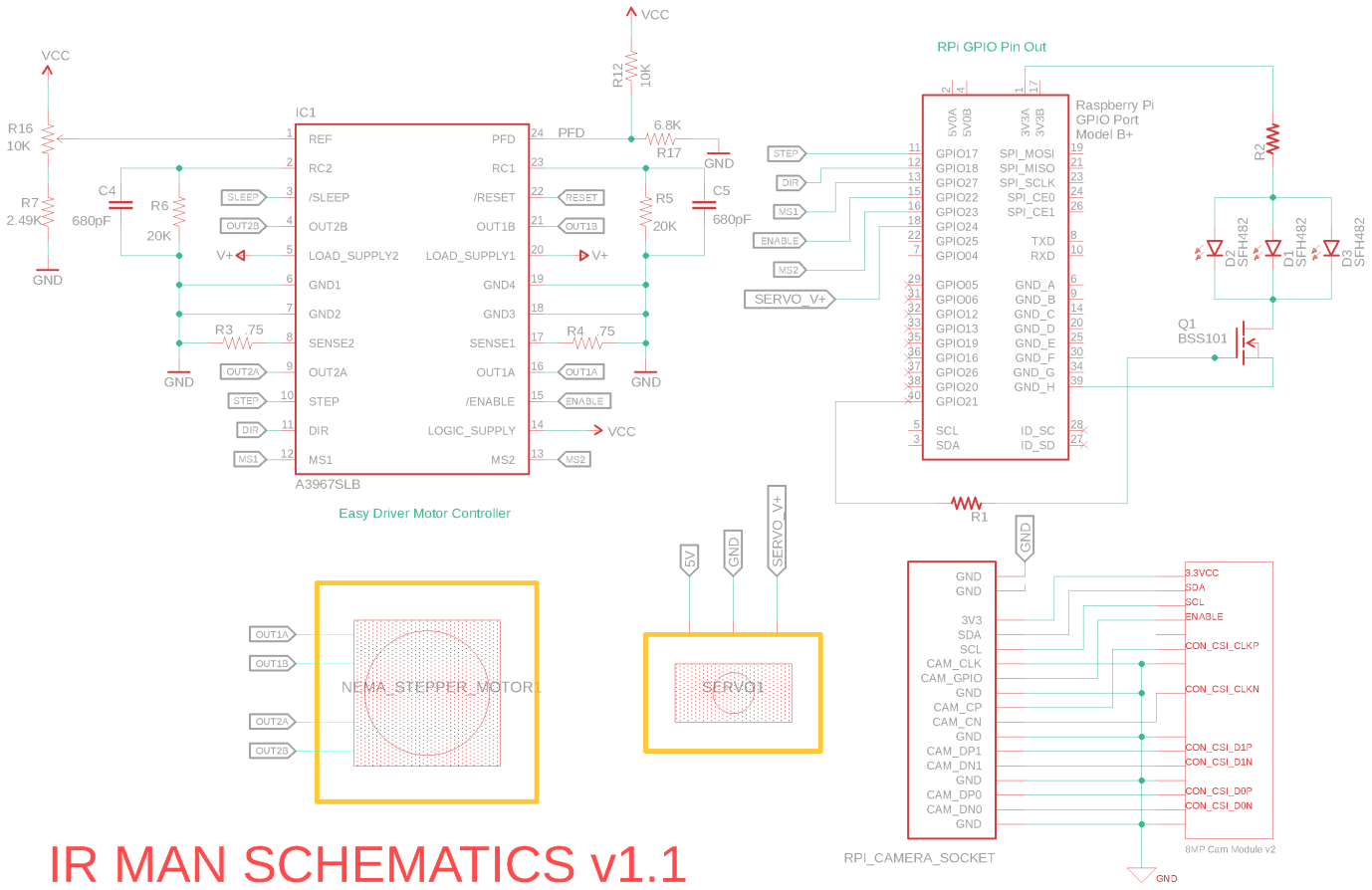

Fig. 15: Schematics for IR Man

MS1	MS2	Micro-step Resolution
L	L	Full Step (2 Phase)
H	L	Half Step
L	H	Quarter Step
H	H	Eighth Step

- STEP -Logic Input. Any transition on this pin from LOW to HIGH will trigger the motor to step forward one step. Direction and size of step are controlled by DIR and MSx pin settings.
- DIR -Logic Input. This pin determines the direction of motor rotation. Changes in the state from HIGH to LOW or LOW to HIGH only take effect on the next rising edge of the STEP command.
- ENABLE -Logic Input. Enables the FET functionality within the motor driver. If set to HIGH, the FETs will be disabled, and the IC will not drive the motor. If set to LOW, all FETs will be enabled, allowing motor control.

Eventually, we adopted using Eighth Step to ensure that the stepper motor is outputing the maximum amount of torque. And the trade-off is that the motor is now 1/8 of the speed comparing to the full step mode. The servo is controlled by RPi's PWM GPIO pins output signals.

In order to "remember" the device locations, RPi write all the device location map received from the device locator server in a local file. Upon booting up, RPi would read this file and restore the device location map. Furthermore, for the device location to be accurate, RPi also needs to "remember" its own pose, so $\theta 1$ is also written to a local file every time the IR Man moves. Upon booting up, RPi reads this file and retrieve its current pose.

4) Scenario Modes: There are four predefined scenario modes: quiet mode, go home mode, leave home mode, and party mode.

Quiet mode has the following commands: Disco Light: Off, AC: night mode, TV: Mute. Party mode has the following commands: Disco: Strobe, TV: Volume Up. Go home mode: AC: Power, TV: Power, Disco Light: White. Leave home mode: AC: Power, TV: Power, Disco Light: Power. Upon receiving the opcode for these modes, RPi runs the list of commands one by one.

D. Mechanical System

The IR Man mechanical system consists of the upper body of the IR Man and the rotating base. Since the camera is connected to the RPi with the flex cable that cannot be twisted, the RPi has to be spinning with the IR Man. Thus, we put every electrical components on the rotating base except for the motor that is driving the rotating base. We also decided to use a sprocket/chain system with a slip ring to drive the rotating base because the wires connecting the RPi and the motor could get twisted after the IR Man rotates a few cycles. The slip ring makes sure that the wires are not twisted, and the sprocket/chain makes sure that there is no slip between the motor and the rotating base.

The motor is secured by using a motor bracket and a L-shaped pattern bracket. The motor drives a small sprocket. The large sprocket is secured on a steel tube, which is supported by two pillow block bearing. The slip ring is secured at the bottom of the steel tube to feed wires from the motor, through the center of the steel tube, to the RPi on the rotating base. In Fig 16 shows our progression from concept to design, then to manufacture.

We 3D printed the IR Man upper body and secured it on top of the rotating base by a carved wood base. The servo is embedded into a cutout we made on one side of the upper body and the arm is secured to the servo with hot glue gun. The camera is placed on the chest of the IR Man and we have two parallel IR sensor secured on the arm in order to increase the signal strength and area that the signal covers.

VI. System Validation & Results

For testing our components and the final product against the our requirements, we designed the following test and validation methods for each subsystems. And the results are reported in the following sections.

A. WebApp Metrics Validation

WebApp: We recorded the latency of server to RPi by inputting 100 IR MAN command messages sent from server(webapp side) to 10 concurrent simulated RPi clients. Each command consists of gathering information from database and sending the message over network. For responsive UI, we want the latency of WebApp be smaller than 500ms to pass the test. According to our system design, this latency is subject to the network bandwidth, message size and database access time. The average latency we got for the final system layout is 24.41ms, with the difference between max and min be around 6.5 ms (very small). According to our result, the performance definitely goes better than the initial bar we set.

IR Circuit: The success rate of Infrared circuit is recorded for distances from 4 to 10 feets, with each IR command operated for 3 times at each distance. The test input is the IR signal sent from our IR MAN. We would like the overall success rate to be 90% within the range of 4 to 10 feet since typically people control home devices within this distance. This success rate is subject the the strength of IR emission and many environmental factors. For our 2-diode structure, we found that IR worked 100% for 4-9 feet under different lighting conditions. However, the success rate went to 75.2% for 10 feet specifically. It seems that 10 feet is the limit of IR capacity. Nevertheless, the average success rate is 96.5% which exceeds 90%.

B. Motor Metrics Validation

Motor: We tested the motor position accuracy for stepper motor and servo motor respectively by giving each 8 positions and checking the difference between their rotation and the expected result manually. We are not worrying about the built up error for consecutive rotations as both stepper motor and servo motor are not affected by that. The input of test is just the expected location, and a passed test is the average error being less than 5 degrees. The stepper motor turned out to be very accurate, with no observable error. Servo motor has a maximum of 2 degree off and a minimum of 0. The average error for servo motor is 0.375 degrees. Test passed.

To test the time it takes for the motor to rotate to any location, we first record the time it takes for a full rotation, and then test the motor with 5 different device layouts. The devices are located without overlap and at least 10 degrees in between. At first we expected the time it take to be less than 2 seconds to be a passed test. However, this number could not be reached for real-world implementation. By the nature of stepper motor, faster the motor spins, smaller the torque it outputs. If we want to move IR Man(which is heavy, over 750 grams), we need the motor to spin slower. The number we got in the end is less than 3 seconds for each spin given our layout of devices.

C. Device Locator Metrics Validation

Device Locator: To test the device locator accuracy, we collected 140 photos with 0-4 devices on them as the validation input. The initial requirement for validation accuracy of our object detection model is set at 75%. This is because the accuracy of YoloV3 model trained on standard dataset is around 72%, and since we are only training 4 classes, we expected a higher accuracy. The result we got was around 83.5%, with no false negative but a few false positives. It is also shown in the results that our model does not work very well under complicated environment with many interference. This is probably because the variety of our training images is not enough, which should not be a problem in our scope.

To test the single image transfer latency between device locator server and RPi, we record the time for single image transfer of camera-sized images for 10 times. Considering the size of image and network bandwidth of websocket, we set the goal of passed test to be less than 750 ms. The result we got was an average of 612 ms, with max and min being 635 ms and 607 ms.

The error of the output position of device locator is quantified by collecting 10 samples of position output and checking that with the actual center of the device. We would like the

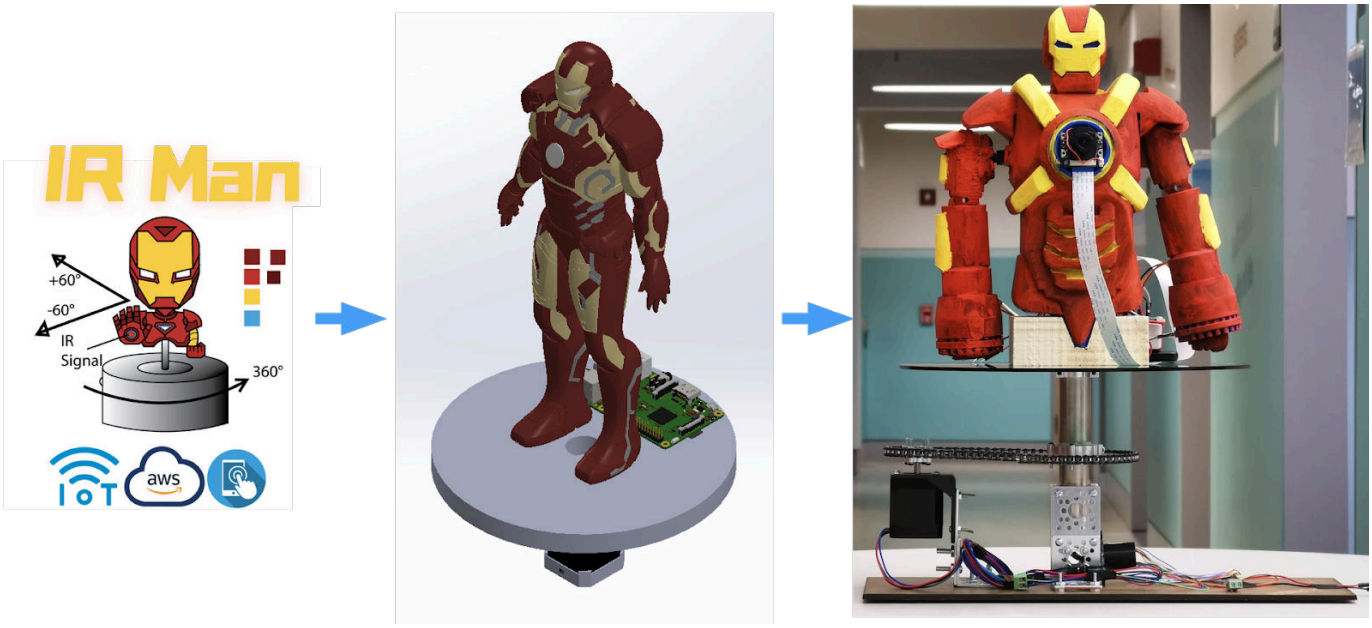

Fig. 16: From Concept to Design to Manufacture

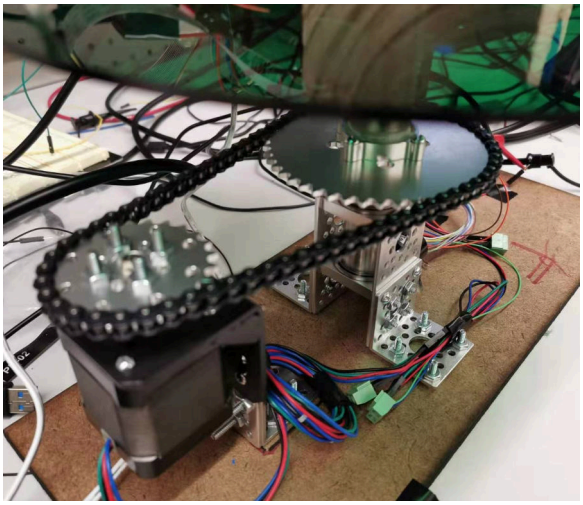

Fig. 17: Mechanical Base of IR Man

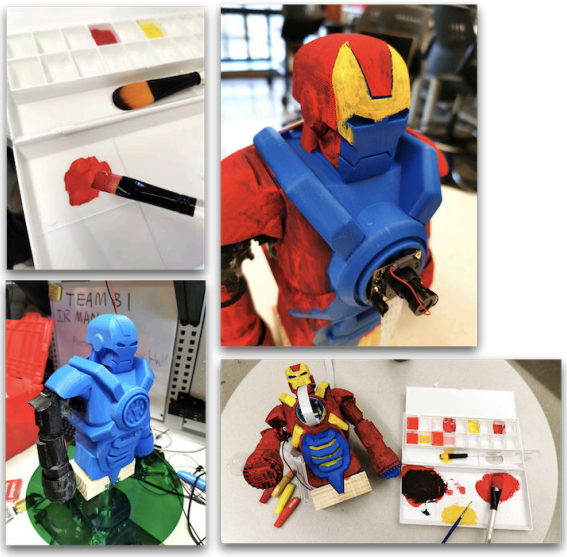

Fig. 18: 3D Printed IR Man & Paint

output of device locator to have an error of less than 10 degrees such that IR may still work in this range. The average error is 2.74 degrees and maximum of error is 6.5 degrees. This is because in certain frames only part of the device is captured, and the calculated center of device can therefore be slightly shifted. Nevertheless, the result is within our requirement.

Latency of a complete calibration is recorded starting from the user sending command and ending at IR MAN receiving the device positions from device locator server. We would like the total calibration time to be under 3 minutes given that a full calibration included latency of both software and hardware operations. We ran complete calibration for 10 times and the maximum time it took was 103 seconds. This seems to be a special case since all other results are around 30 seconds. The extra time was due to a especially crammed CPU workload.

Component	Requirement	Testing Method	Result
WebApp Latency	Server to RPi latency under 500ms	Timestep, stress tested with a mock web app server sending 100 messages to 10 concurrent simulated raspberry pi clients	Avg latency of 24.41ms Maximum 26.07ms Minimum 20.49ms
IR Circuit Success Rate	success rate of 90% within range of 4-10 feets	Collect success rate of all types of command signals from 3 different devices/protocol under distances of 4 to 10 feet	100% for 4-9 feet 75.2% for 10 feet 96.5% overall

Fig. 19: WebApp Testing Results

Component	Requirement	Testing Method	Result
Motor Position Accuracy	± 5 degrees of correct pose	Feed in 8 expected positions and manually check the result	Accurate for stepper motor Avg error 0.375 degrees for servo motor (Max 2, min 0)
Motor Rotation Time	Time to specific pose < 1s	Record time it takes for a full rotation and rotations with 5 different layouts (no overlap, minimum 10 degrees in between)	Takes <3 seconds

Fig. 20: Motor Testing Results

Component	Requirement	Testing Method	Result
Device Locator Accuray	Image validation accuracy > 75%	Manually set up 140 validation images, run through model and collect result	83.5% accuray in unmonitored environment
Device Locator Latency	Image transfer latency < 750 ms	Record time for 10 image transfer of camera-sized images over websocket	Avg 612 ms Max 635 ms Min 607 ms

Fig. 21: Device Locator Testing Results 1

Component	Requirement	Testing Method	Result
Device Locator Output Position	± 10 degree of the correct pose for each devices	10 test calibration runs (10 images) in different environment with TV, Disco lights, AC	Avg 2.74 degrees Max 6.5 degrees
Calibration Latency	Total calibration time < 3 mins	10 complete calibration runs in different environment with TV, Disco lights, AC	Worst case 103 seconds. Normally avg 33 seconds.

Fig. 22: Device Locator Testing Results 2

We consider this a passed test.

D. Overall Metrics Validation

Component	Requirement	Testing Method	Result
User Success Rate	Success rate > 90%	Run 30 IR commands on test devices with correct pose information	86.67% (26 out of 30)
User Latency	Average Latency < 2s	Run 30 IR commands on test devices with correct pose information	Average total time: 3.107 s

Fig. 23: System Testing Results

Device Operation: We are testing the success rate of user device operation command by running 30 IR commands on user webapp side with the correct device position information manually stored. We would like the success rate to be over 90%. The actual result we got is 26 commands out of 30 working successfully, and the percentage is 86.67%. While statistically we didn't fulfill our requirement, we are aware that 90% is just 27 out of 30 and that certain commands on disco light is interfered by the lights it emits. We consider our result as slightly off the requirement but not necessarily failed the test.

The latency of user device operation command is tested in the same way as we test the success rate, where we record the time it take for an operation to be completed. Initially we set the bar to be 3 seconds. The result we got was an average of 3.107 seconds. This is associated with the motor speed we discussed previously, and overall we think IRMan is moving rather smoothly and actively in response to user command. Therefore, although we did not pass the test, we consider IRMan still in a good shape.

VII. Related Work

We are well aware that the concept of "smart home hub" or "universal IR controller" is not new. In fact, the industry has a selection of universal IR hub in the market right now. (e.g.

Fig. 24: Broadlink RM Mini3 Black Bean Universal Remote

Broadlink RM Mini3 Black Bean Universal Remote, as shown in Figure 24) However, we consider our idea of integrating the hub into a 2-DOF robot as rather pristine in that it solves the problem of line of sight, as well as to avoid mistakenly controlling unwanted devices, which lingers for the existing designs of IR hubs. The design of ID from Iron Man also adds a layer of fun and interactivity into this home device, and can be meaningful to Marvel fans like us.

VIII. Project Management

A. Job Distribution

Max mainly worked on the webapp server and the mechanical structure of the IR Man. Shirley worked on the training the machine learning model and finishing the device locator server. Jiaqi mainly worked on the RPi client code: motor/servo control, IR circuit and electrical integration. Please refer to Appendix on the last page for our project management Gantt Chart.

B. Budget & Bill of Materials

Please refer to Fig 25 for our final bill of material. And in Fig 26 shows our AWS EC2 usage.

Name	Quantity	link	price	note	Total	Ordered	Explanation
motor mounting bracket	1	https://www.amazon.com/STEPPER	5.66		5.66	Yes	mounting the motor
set screw hub	1	https://www.servocity.com/770-set-s	4.99	5mm (SKU: 545572)	4.99	Yes	secure the motor shaft to the hub sprockets
hub sprocket(small)	1	https://www.servocity.com/0-770-alu	4.79	24T (SKU: 615106)	4.79	Yes	drive the large hub sprockets by chain
flat pattern bracket	1	https://www.servocity.com/flat-dual-c	1.79		1.79	Yes	support the motor/motor mounting bracket
90 pattern bracket	1	https://www.servocity.com/90-single-	1.89		1.89	Yes	secure the flat pattern bracket to the ground
Rotating Base	1	Ansys Maker Space	0.99	Laser Cutted	0.99	No (Manufactured)	Acrylic Plate
pillow block	2	https://www.servocity.com/1-000-bor	9.99		19.98	Yes	support the rotating tube
flat pattern bracket	2	https://www.servocity.com/flat-dual-c	1.79		3.58	Yes	secure the pillow block
90 pattern bracket	3	https://www.servocity.com/90-single-	1.89		5.67	Yes	secure the flat pattern bracket to the ground
metal tube	1	https://www.servocity.com/1-00-stair	6.69	Length: 6.00 (SKU:635142)	6.69	Yes	support the rotating top surface
hub sprocket(large)	1	https://www.servocity.com/1-50-alum	7.19	48T (SKU:615126)	7.19	Yes	drive the large hub sprockets by chain
tube mounting clamp	2	https://www.servocity.com/1-bore-bc	5.99		11.98	Yes	secure the top surface to the steel tube
tube clamping hub	1	https://www.servocity.com/1-bore-cla	6.99		6.99	Yes	secure the large hub sprockets to the steel tubing
chain	1	https://www.servocity.com/0-250-cha	5.99		5.99	Yes	drive the large hub sprockets by chain
Slip Ring	1	https://www.amazon.com/GeeBat-C	14.99	12 Wires	14.99	Yes	connect motor wire to RPi
Rasberry Pi 3	1	https://www.amazon.com/ELEMENT	$49.49		$49.49	No (inventory)	Choice of MCU
RPi Camera Modules	1	https://www.amazon.com/AuviPal-R	$15.99		$15.99	No (inventory)	Camera Sensor
Step Motors	1	https://www.amazon.com/Stepper-M	$12.99		$12.99	No (inventory)	Choice of kinematics
Easy Driver	1	https://www.sparkfun.com/products/	$14.95		$14.95	Yes	Controller board to control motor
Servo	1	https://www.amazon.com/J-Deal-Mic	$11.78		$11.78	No (inventory)	2nd Choice of kinematics
IR Man 3D Printing	200	Ansys Maker Space	$0.50		$100.00	No	ID Design 3D printed at Ansys Maker space
AWS Server (t2.micro + GPU)	1	https://console.aws.amazon.com/co	$50.31		$50.31	No	GPU Server for CV image training
Total					**$358.68**		

Fig. 25: Bill of Materials

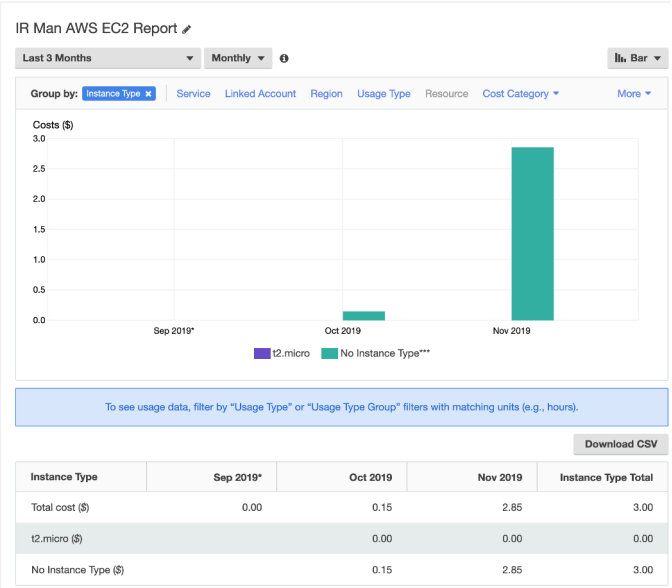

Fig. 26: AWS Usage

IX. Conclusion

A. Future Work

Up till now, our implementation of IR MAN only supports manipulating 4 different types of IR devices. The servers are only interacting with one IR MAN and the messages are not well-encrypted for security. For future work, the first thing we may do is to upgrade the IR code database such that it supports a wider range of IR devices. The transmission of IR codes is handled by the LIRC library, which already incorporated a wide selection of existing IR devices. For those that are not recorded, their IR codes can be easily collected through IR receivers and LIRC library. We consider this a very scalable feature of our database. The next step will be enabling servers to work with multiple IR MANs. In fact, this feature has already been implemented within our system design, and we did not test it because we consider this feature as not required for the scope of this course. Regarding the encryption of messages, what we have now is including the id of IR MAN as a basic proof of message validity. What we may do in the future is to develop a unique hashing algorithm which enhances the security of our messaging protocol.

To learn more about IR Man, the development process, watch the project video, see more pictures and status updates, please check out our project home page [4]. And stay tuned for any future updates.

B. Lessons Learned

The first lesson we learned is that do not ever be unclear in what we are about to do. To finish a project in time, we should always have a clear structure in mind even if we are changing our design in between. This does not imply that we should always know the details of any specific implementation, but that we should be clear about what are the portions that are left to be filled in.

Another important lesson we learned is that the initial goal should be set high, since reality can be fun of surprises and in most cases, people will not be able to finish everything they planed for in the beginning. We we do not aim high enough, then we might not be able to complete even a reasonably satisfying product.

Finally, we also realized that integration can take much more time than we expected. One approach we found very helpful is to keep a spread sheet which specifies the variables and protocols to be shared between different modules. A well-maintained sheet can significantly shorten the integration difficulty.

APPENDIX A
IR MAN GANTT CHART

Refer to Figure 27 for the Gantt Chart.

ACKNOWLEDGMENT

Thanks CMU ECE department for giving us the opportunity to work on a capstone project and transform our knowledge into real world applications. Thanks Prof. Bill Nace for providing the course logistics and framework for us to work efficiently on our project. Thanks Prof. Vyas Sekar and Prof. Shawn Kelly for providing us lots of technical support and advice during our development. Thanks Ranganath (Bujji) Selagamesetty for providing us technical supports and logistical supports and being a good TA/friend.

REFERENCES

[1] LIRC: Linux Infrared Remote Control Webpage
http://lirc.sourceforge.net/remotes/
[2] Jonathan Hui: Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)
https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359
[3] Google Spreadsheet: Bill of Material for IR Man)
https://docs.google.com/spreadsheets/d/1_w5oPxHe_FYA4SagRiQoSDFtwuDNIX81X4zk4ygqV1Y/edit
[4] IR Man Home Page:
http://course.ece.cmu.edu/ẽce500/projects/f19-teamb1/

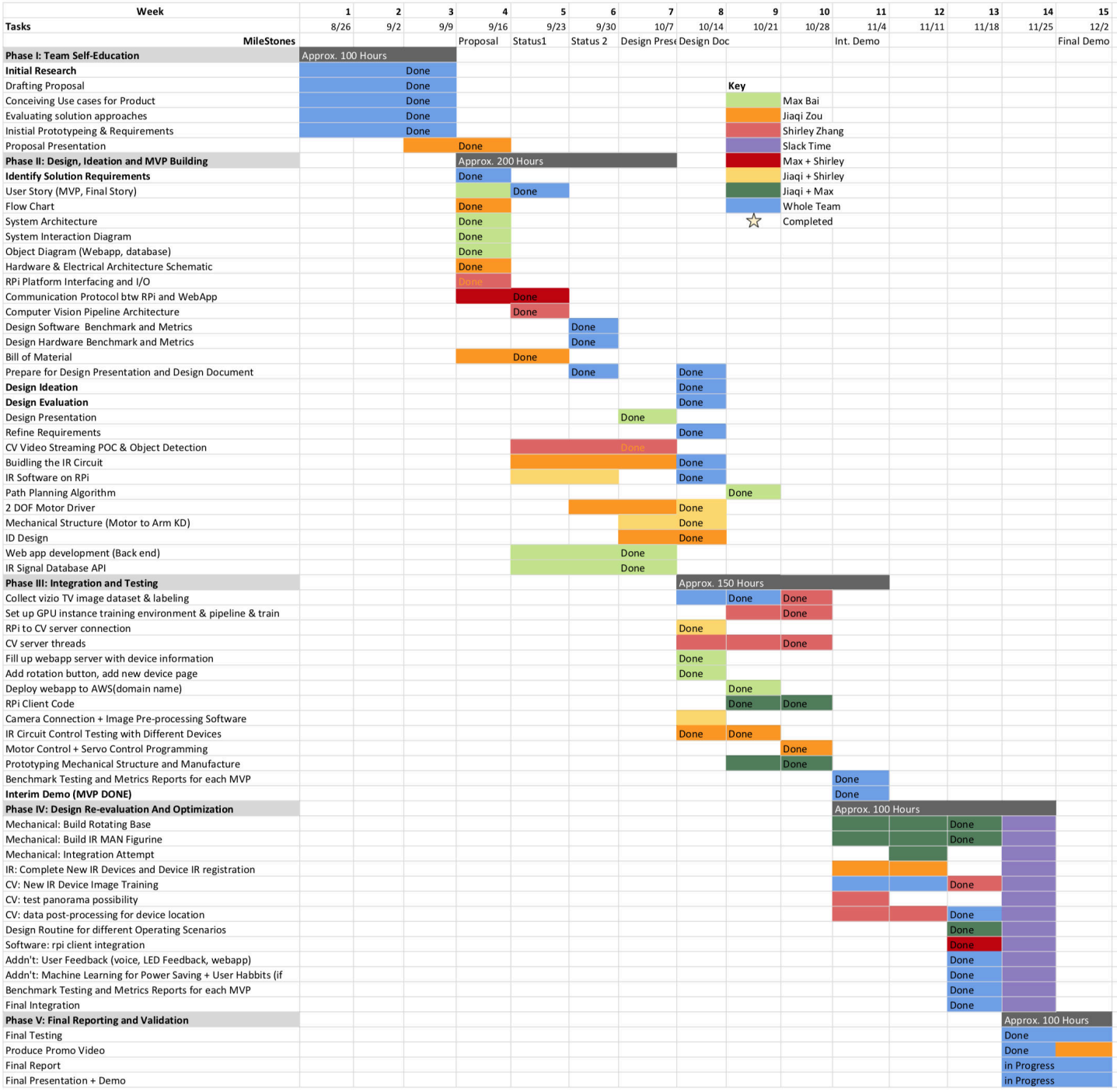

Fig. 27: Gantt Chart for IR Man