הטכניון – מכון טכנולוגי לישראל

הפקולטה להנדסת חשמל

Networked Software Systems Laboratory

ספר פרויקט

Smart mobile car

אביב תשע"ח

Amid Shibli

Ashraf Yassin **Supervisors: Oren Kalinsky**

Table of content

Introduction

1. Project Goals

The project's main goal:

To implement a controlled smart car with smart camera that's controlled via android phone through AWS.

Secondary goals:

- The car motors controlled via application instead of joystick.
- The camera shall stream live video to the application with minimum delay.
- The camera shall have the ability to move by micro-servo engines.
- The user will have the ability to control the camera movement by 360 degree
- The user will have the ability to control the camera movement by the phone's motion sensors.
- The application will be user friendly.
- The implementation will use several AWS services.

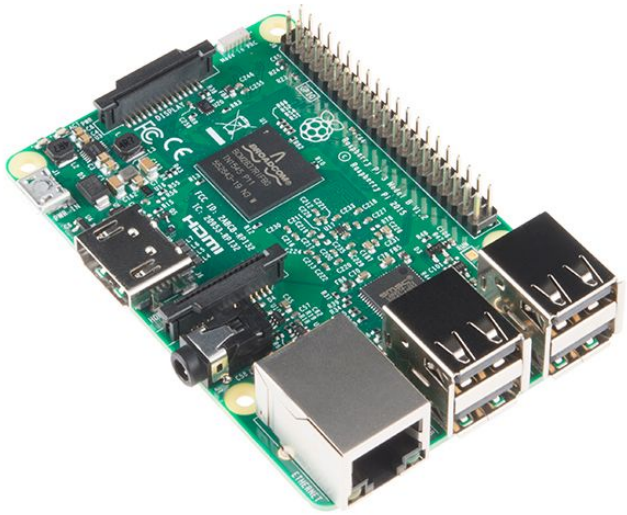
source code : https://github.com/ashrafyassin/IOT_Project

2. System Components

1. Raspberry Pi:

The Raspberry Pi is a series of small single-board computers.

In this project, we used Raspberry Pi 3 Model B: [Size: 85.60 mm × 56.5 mm × 17 mm



Specification:

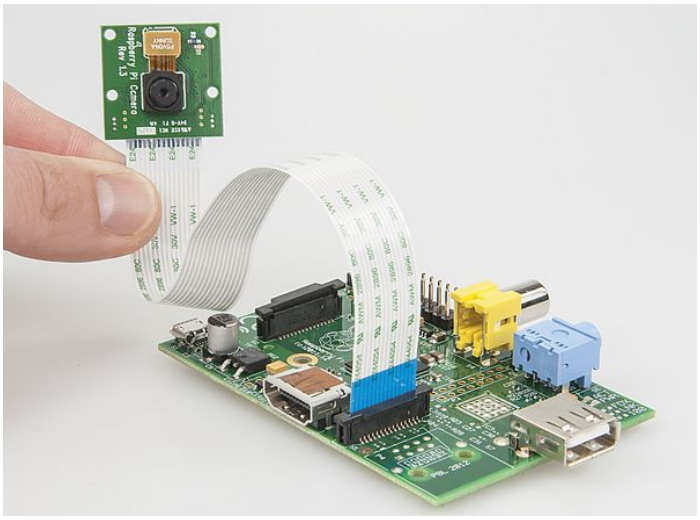
- CPU - 1.2 GHz 64-bit quad-core ARM Cortex-A53.
- Memory – 1GB.
- Four USB ports.
- 15-pin MIPI camera interface (CSI) connector (used with the Raspberry Pi camera)
- HDMI output.
- 10/100 Mbit/s Ethernet (8P8C) USB adapter.
- GPIO(General purpose input-output) 40 pins layout
- GPU - Broadcom VideoCore IV @ 250 MHz
- Operating systems - Raspbian, a Debian-based Linux operating system.
- Cost : ~ 35$

2. Raspberry PI Camera

The Raspberry Pi Camera Module is an official product from the Raspberry Pi Foundation.

The original 5-megapixel model was released in 2013, and an 8-megapixel Camera Module v2 was released in 2016.

In this project, we used Camera Module v2.1.



Specification:

- Size - 25 × 24 × 9 [mm]
- Weight - 3 [gram]
- Still resolution - 8 [Megapixels]
- Video modes - 1080p30, 720p60 and 640 × 480p60/90
- Sensor - Sony IMX219
- Video formats - raw h.264 (accelerated)
- Cost-~$25

3. Micro-servo Engines

A servomotor is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration.

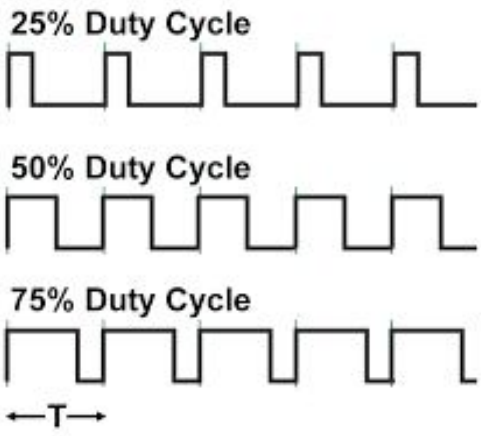
It consists of a suitable motor coupled to a sensor for position feedback. In this project, we used two TurnigyTM TG9e Eco Micro Servo.

Specification:

- Dimension: 23x12.2x29mm
- Torque: 1.5kg/cm (4.8V)
- Operating speed: 0.10sec/60 degree 0.09sec/60 degree(6.0V)
- Operating voltage: 4.8V
- Temperature range: 0-55C

The sensor gets its feedback from the head’s current position. As a signal, the servo gets Pulse-width modulation (PWM):

A square wave signal with a changing duty cycle (Pulse Width = Duty Cycle / Time [total period of the signal]).

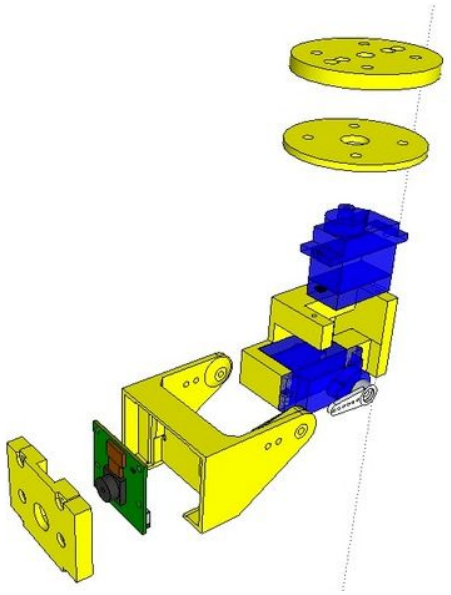


The engines are connected to the PI GPIO - General-purpose input-output (three pins each: V+, V - , Control) and get the signals directly from the PI.

4. Pan-tilt Mount

We took the pan tilt mount from the previous project ,where it control the camera movement through the micro-servo engines, they had to insert the different components into a structure that will hole them together and keep smooth movement.

A designated mount (found at Thingiverse.com) was printed through a 3-D printer:



The mount has two axis: X/Y, each axis is being controlled by a different motor. The camera is sitting inside the mount when its base is fixed.

5. Devastator Tank

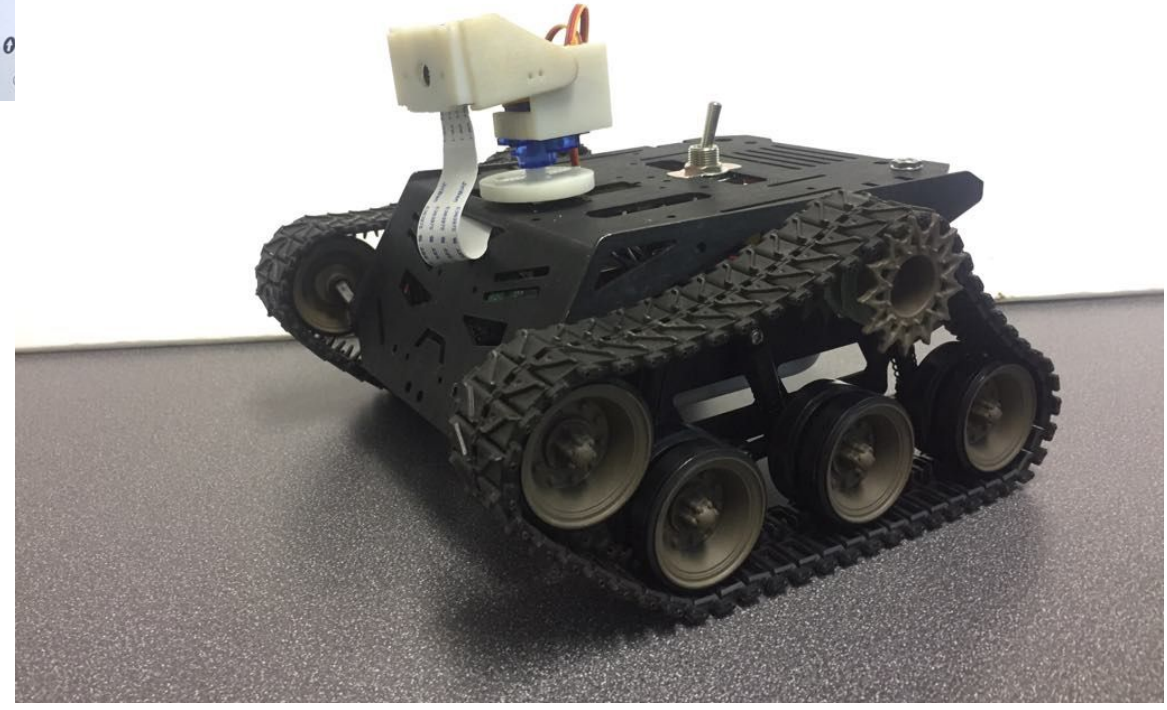
We bought the car from dfrobot website , DFRobot proudly presents Devastator Tank Mobile Platform. It is fully compatible with most popular controllers in the market such as Arduino, Raspberry Pi and so on.

The Devastator uses high strength aluminum alloy which makes it extremely solid and durable. The high speed motors and premium tracks also allows it to move swiftly everywhere. What's more, it benefits from a high performance suspension and enjoys an outstanding mobility across even the toughest terrains.

ability to add various sensors, servos, turntables and controllers with multiple mounting holes on the Devastator platform. It is perfect for hobbyists, educational, competitions and projects.

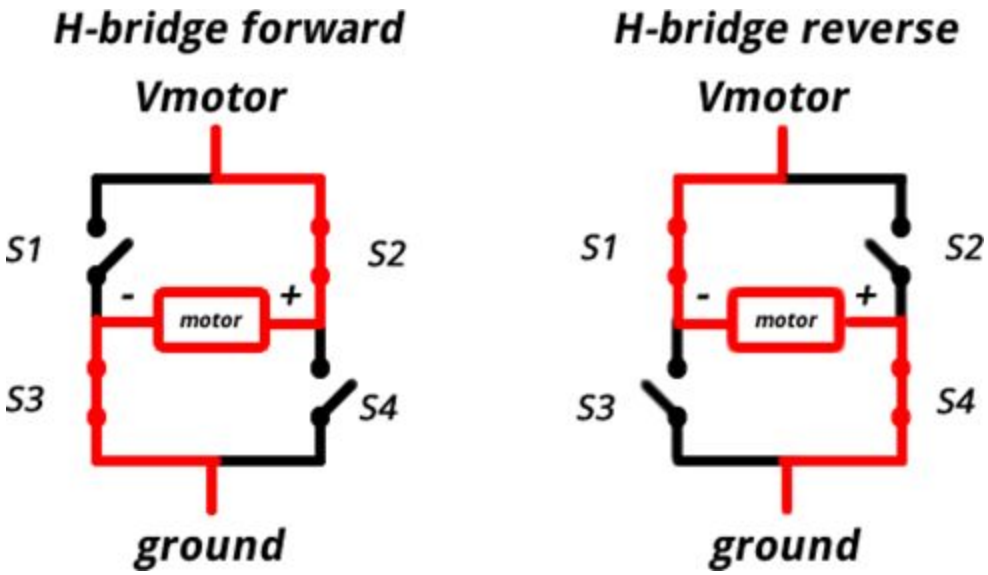
SPECIFICATION :

- Working Voltage：3-8V DC
- Driver wheel diameter : 43mm(1.69")
- Maximum Speed: 36cm/s
- Motor Specification：see Micro DC Geared Motor with Back Shaft
- Size：225*220*108mm(8.86*8.66*4.25")
- Weight：780g
- Cost : 70$

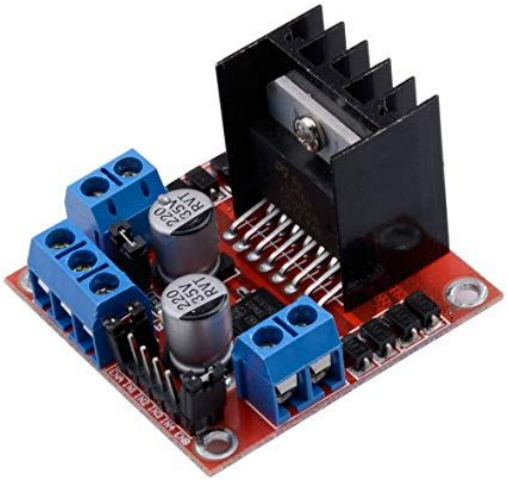


6. H-bridge

An H bridge is an electronic circuit that enables a voltage to be applied across a load in either direction. These circuits are often used in robotics and other applications to allow DC motors to run forwards or backwards.



The name is given because of the physical appearance of the circuit.



Project schema

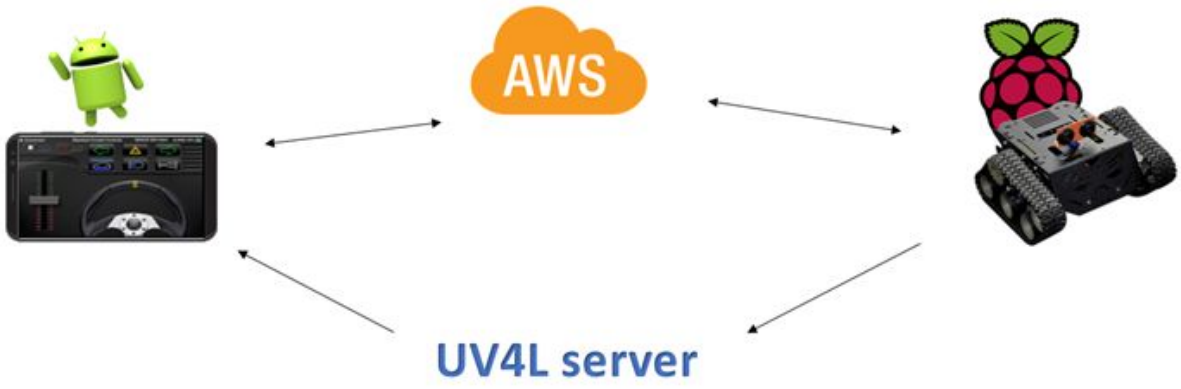


Figure 1

The tank connected to the aws , the android user publish to the aws , aws republishing the command to the tank controlling the motor .

The RP3 streaming live video to the application via UV4L server .

AWS

AWS (**A**mazon **W**eb **S**ervices) is a major component in this project and contain vast services that were used in order to implement it:

1. IOT

The Internet of things (IoT) is the network of physical devices, vehicles, home appliances and
other items embedded with electronics, software, sensors, actuators, and network connectivity,
which enables these objects to connect and exchange data.
Through AWS IOT, few components were registered:

- Raspberry PI
- AWS EC2 instance
- Android application

The different components can communicate after registration.
In this project, the communication was through MQTT protocol:

MQTT (MQ Telemetry Transport or Message Queuing Telemetry Transport) is an ISO standard
(ISO/IEC PRF 20922) publish-subscribe-based messaging protocol. It works on top of the TCP/IP

protocol. It is designed for connections with remote locations where a "small code footprint" is
required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a
message broker

2. EC2

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the
Amazon Web Services (AWS) cloud. Using Amazon EC2 eliminates your need to invest in
hardware up front, so you can develop and deploy applications faster. You can use Amazon EC2
to launch as many or as few virtual servers as you need, configure security and networking, and
manage storage. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.
In this project, an AWS EC2 instance was set, in order to run the gateway and video processing
script with no hardware dependency.
In that way, the video stream can be reached from anywhere and can communicate with other
relevant AWS services

3. Cognito

Amazon Cognito is an Amazon Web Services (AWS) product that controls user authentication and access for mobile applications on internet-connected devices. The service saves and synchronizes end-user data, which enables an application developer to focus on writing code instead of building and managing the back-end infrastructure. This can accelerate the mobile application development process.

In this project, AWS Cognito service is used as a sign up method for the application user.

4. IAM

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources. When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account.

In this project, AWS IAM service is used in order to secure the AWS resources and to give

permissions only to specific components for specific operations

5. Route53

Amazon Route 53 effectively connects user requests to infrastructure running in AWS – such as Amazon EC2 instances, Elastic Load Balancing load balancers, or Amazon S3 buckets – and can also be used to route users to infrastructure outside of AWS. You can use Amazon Route 53 to configure DNS health checks to route traffic to healthy endpoints or to independently monitor the health of your application and its endpoints. Amazon Route 53 Traffic Flow makes it easy for you to manage traffic globally through a variety of routing types, including Latency Based Routing, Geo DNS, Geo Proximity, and Weighted Round Robin—all of which can be combined with DNS Failover in order to enable a variety of low-latency, fault-tolerant architectures. Using Amazon Route 53 Traffic Flow simple visual editor, you can easily manage how your end-users are routed to your application's endpoints—whether in a single AWS region or distributed around the globe. Amazon Route 53 also offers Domain Name Registration – you can purchase and manage domain names such as example.com and Amazon Route 53 will automatically configure DNS settings for your domains.

6.Load Balancer

Load balancing is a technique commonly used by high-traffic Web sites and Web applications to share traffic across multiple hosts, thereby ensuring quick response times and rapid adaptation to traffic peaks and troughs. The Elastic Load Balancing service from Amazon Web Services with Secure Sockets Layer (SSL) support makes it easy to add secure load balancing for Bitnami applications running on AWS.

A load balancer distributes workloads across multiple compute resources, such as virtual servers. Using a load balancer increases the availability and fault tolerance of your applications.

You can add and remove compute resources from your load balancer as your needs change, without disrupting the overall flow of requests to your applications.

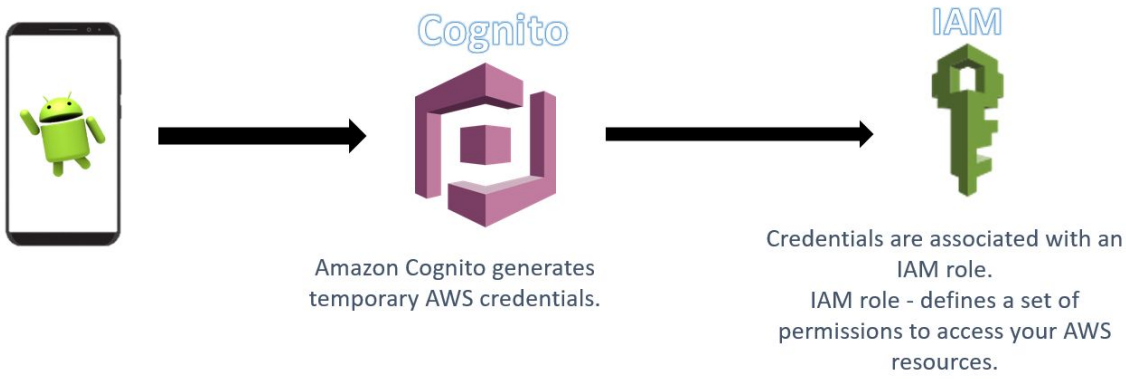
You can configure health checks, which are used to monitor the health of the compute resources so that the load balancer can send requests only to the healthy ones. You can also offload the work of encryption and decryption to your load balancer so that your compute resources can focus on their main work.

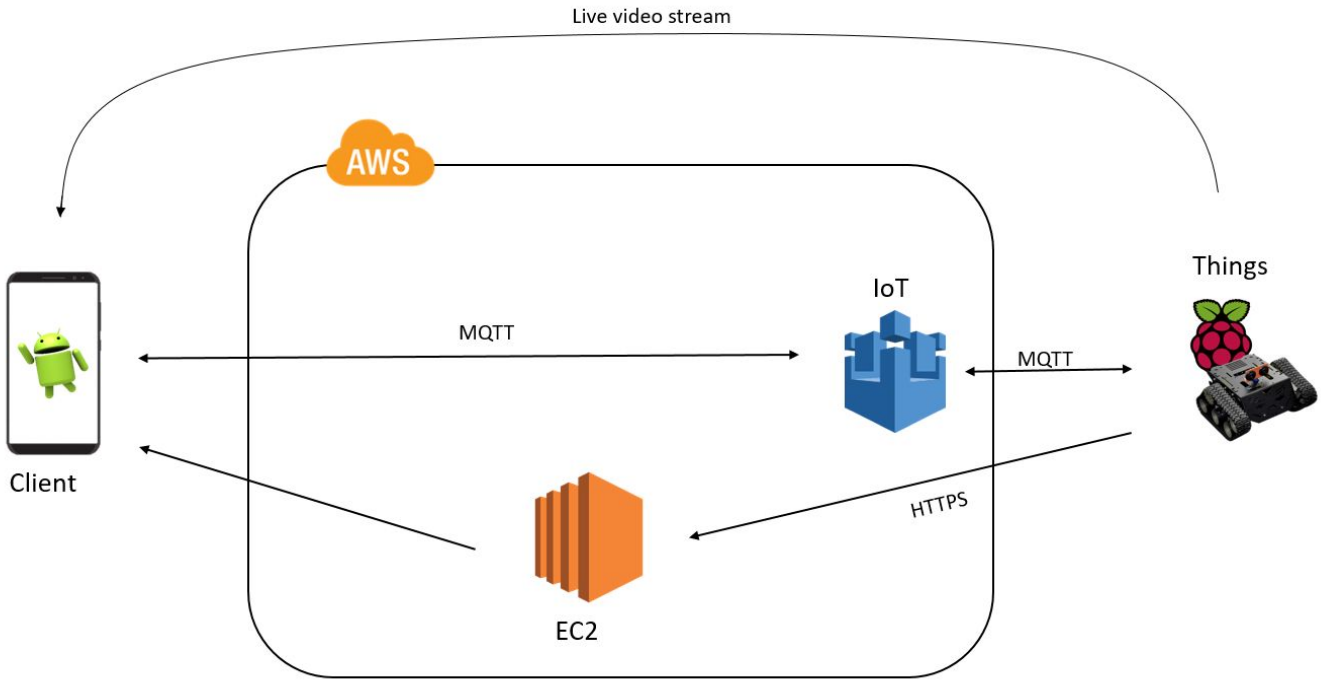
7.ACM

AWS Certificate Manager is a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with AWS services and your internal connected resources. SSL/TLS certificates are used to secure network communications and establish the identity of websites over the Internet as well as resources on private networks. AWS Certificate Manager removes the time-consuming manual process of purchasing, uploading, and renewing SSL/TLS certificates.

With AWS Certificate Manager, you can quickly request a certificate, deploy it on ACM-integrated AWS resources, such as Elastic Load Balancers, Amazon CloudFront distributions, and APIs on API Gateway, and let AWS Certificate Manager handle certificate renewals. It also enables you to create private certificates for your internal resources and manage the certificate lifecycle centrally. Public and private certificates provisioned through AWS Certificate Manager for use with ACM-integrated services are free. You pay only for the AWS resources you create to run your application. With AWS Certificate Manager Private Certificate Authority, you pay monthly for the operation of the private CA and for the private certificates you issue.

System architecture





Live Video Streaming

The live video streaming feature is implemented through UV4L [1.2.6] and Janus Gateway
[1.2.7].
[UV4L configuration files, available: https://github.com/ayalgenzer/SecurityCamera/tree/master/Rasbperry_PI/UV4L_configuration]
In order to watch the video streaming consecutively we needed to convert the camera output into a device file.
In Unix-like operating systems, a device file is an interface for a device driver that appears
in a file system as if it were an ordinary file. They allow software to interact with a device driver using standard input/output system calls, which simplifies many tasks and unifies userspace I/O mechanisms.
UV4L supplies the conversion of the video output into a device file in the next path: /dev/video0.
A gateway is a link between two computer programs or systems such as Internet Forums. A
gateway acts as a portal between two programs allowing them to share information by communicating between protocols on a computer or between dissimilar computers.
UV4L also supplies many tools that can be used as gateways for the video streaming.

In order to access the device file (video streaming) from the application that is outside of the
Raspberry PI/Camera LAN and avoid the LAN security, we had to use a gateway so we chose
Janus Gateway.
UV4L supplies built-in tool to use the Janus Gateway plug-in "video room".
The "video room" is a plug-in of Janus Gateway that implements a simple videoconferencing
application that can be watched through HTTP.
The implementation is written in JS and the display interface is an HTML.
Janus Gateway uses WebRTC technology in order to stream the video.
Since WebRTC, technology is a peer-to-peer connection and the two peers are probably in a
different LAN there is a good chance that neither has an IP address and port that the other can
send directly to.
Therefore, ICE technique is used by Janus Gateway.

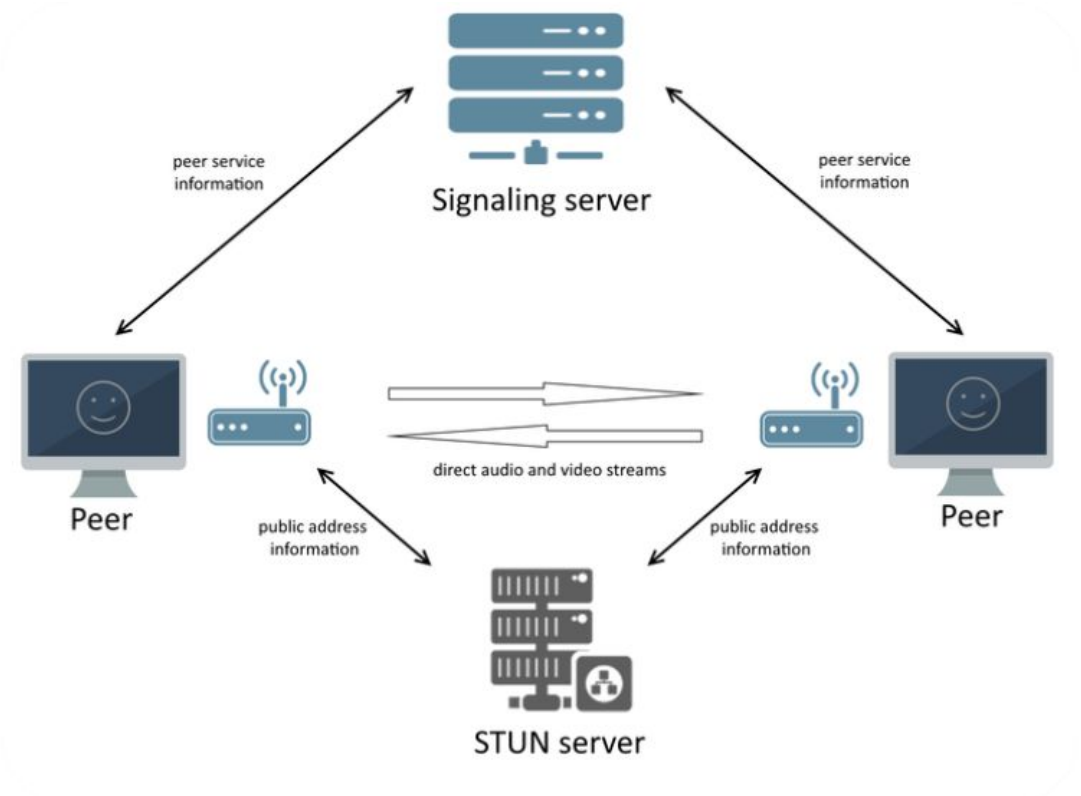
> **I**nteractive **C**onnectivity **E**stablishment (ICE) is a technique used in computer networking to
> find ways for two computers to talk to each other as directly as possible in peer-topeer networking. This is most commonly used for interactive media such as Voice over Internet
> Protocol (VoIP), peer-to-peer communications, video, and instant messaging. In such
> applications, you want to avoid communicating through a central server (which would slow down
> communication, and be expensive), but direct communication between client applications on

the Internet is very tricky due to network address translators (NATs), firewalls, and other network barriers

ICE technique interact with several servers:

- Signaling Server

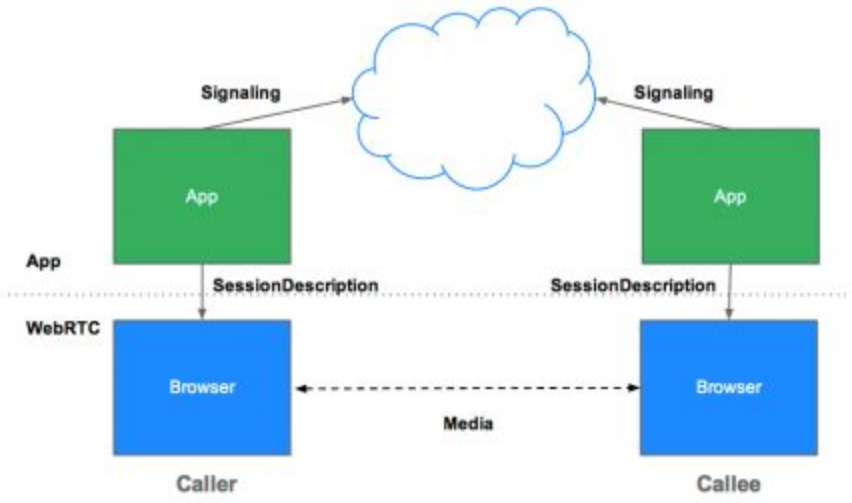
  A signaling gateway is a network component responsible for transferring signaling messages (i.e. information related to call establishment, billing, location, short messages, address conversion, and other services) between Common Channel Signaling (CCS) nodes that communicate using different protocols and transports.



Protocol conversion gateways can also convert from one network operational paradigm to another.

In this project, as a part of UV4L support of Janus Gateway, the signaling server is already

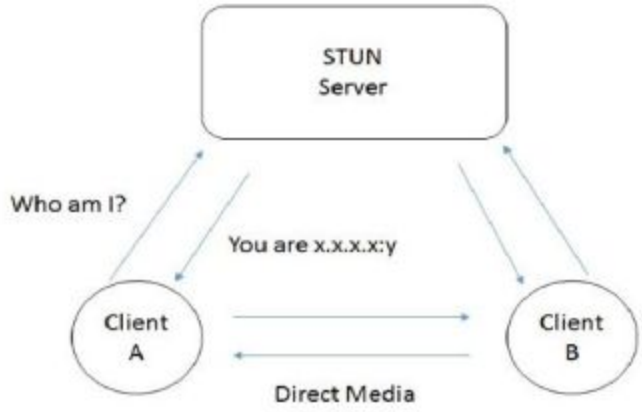
implemented.



• STUN (Session Traversal Utilities for NAT) Server

Session Traversal Utilities for NAT (STUN) is a standardized set of methods, including a
network protocol, for traversal of network address translator (NAT) gateways in applications of
real-time voice, video, messaging, and other interactive communications.
STUN is a tool used by other protocols, such as Interactive Connectivity Establishment (ICE),
the Session Initiation Protocol (SIP), or WebRTC. It provides a tool for hosts to discover the
presence of a network address translator, and to discover the mapped, usually public, Internet
Protocol (IP) address and port number that the NAT has allocated for the application's User
Datagram Protocol (UDP) flows to remote hosts. The protocol requires assistance from a third-party network server (STUN server) located on the opposing (public) side of the NAT, usually the
public Internet.

In this project, a google STUN server was used- 19302:com.google.l.stun.

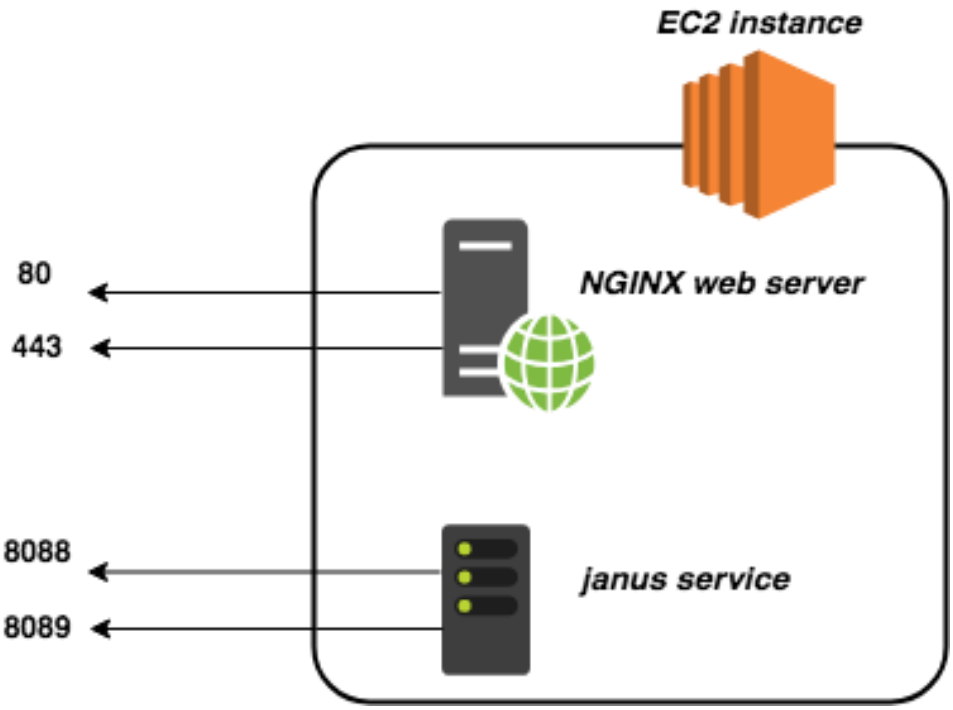


In this project, we edited the "video room" plug-in's code to match it to our own demands:

- Janus gateway and the video room plugin were installed on an EC2 instance in order to give a public IP and access from anywhere.
- The application has a unique room.
- Only two participants can join the room: one streamer (camera) and one listener (mobile application).
- The listener access has immediate access to the video with the right IP and configuration through HTTPS.
- Janus Gateway's plug in, "Video Room", is using VP8 codec for the video
  VP8 is an open and royalty free video compression format owned by Google and created by On2 Technologies as a successor to VP7.
  Opera, Firefox, Chrome, and Chromium support playing VP8 video in HTML5 video tag.

According to Google VP8 is mainly used in connection with WebRTC and as a format
for short looped animations, as a replacement for the Graphics Interchange Format (GIF).
VP8 can be multiplexed into the Matroska-based container format WebM along with Vorbis and Opus audio.

We hard coded the right configuration for login into the mobile application code so the access
can be instant (after the authorization of sign-in of course).



The communication and synchronization is implemented through AWS IOT with MQTT messages.
When the PI is connected, a startup script runs:

- Starts UV4L.
- Sets the right streaming configuration through Janus Gateway.
- Starts the PIGPIO plug in (controls the servos).
- Runs an MQTT listening script.

Once the Raspberry PI has finished setting up the camera (with UV4L right configuration), it
sends a MQTT message with the topic "OpenJanus".
The EC2 instance is always listening for MQTT messages and once it gets the topic "OpenJanus", it begins its initialization:

- Delete old session's files.
- Initialize global variables.
- Starts video room session (once it opens, it immediately gets the video streaming)
- Starts processing the video (for recording).

Once the set-up has been completed, the mobile application can access to the live video
streaming through HTTP and directed with NGINX.
Nginx is a web server , which can also be used as a HTTP and reverse proxy server, a mail
proxy server, and a generic TCP/UDP proxy server, load balancer and HTTP cache.
Nginx is free and open source software

HTTPS

We got to the point where we can stream the live video from the pi to the janus room, and we wanted to access this video from the android app, but since the web view that we are using in our app is a chrome view, we are accessing the video from chrome browser, and since chrome now is more strict for sensitive content like video and audio, it does not allow accessing them in an insecure way so it demand accessing using HTTPS only, so we had to secure our connection to this stream.

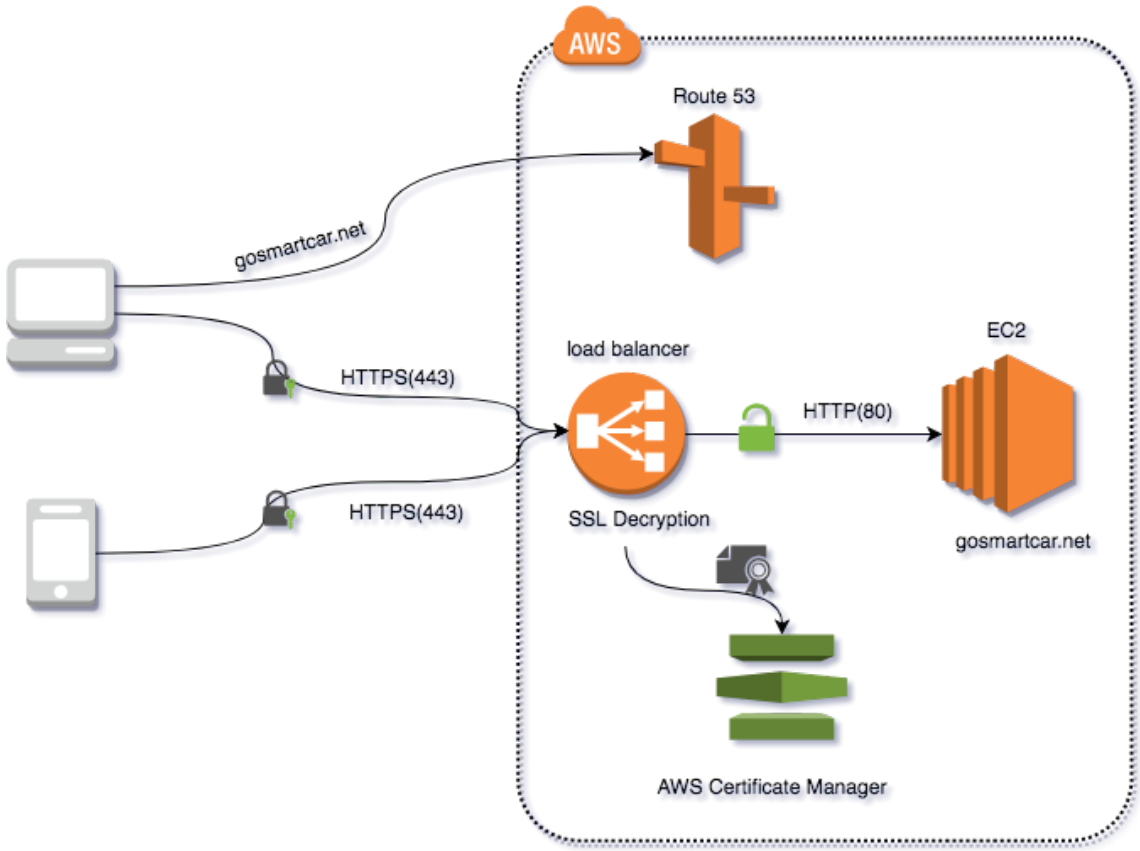
To conclude the steps that we did in order to complete this we had to do:

1.register a domain name.

2. Create SSL certificate for this domain name.

3.create load balancer

4.decrypt the traffic from client to the ec2 in the load balancer.



Explanation for each step:

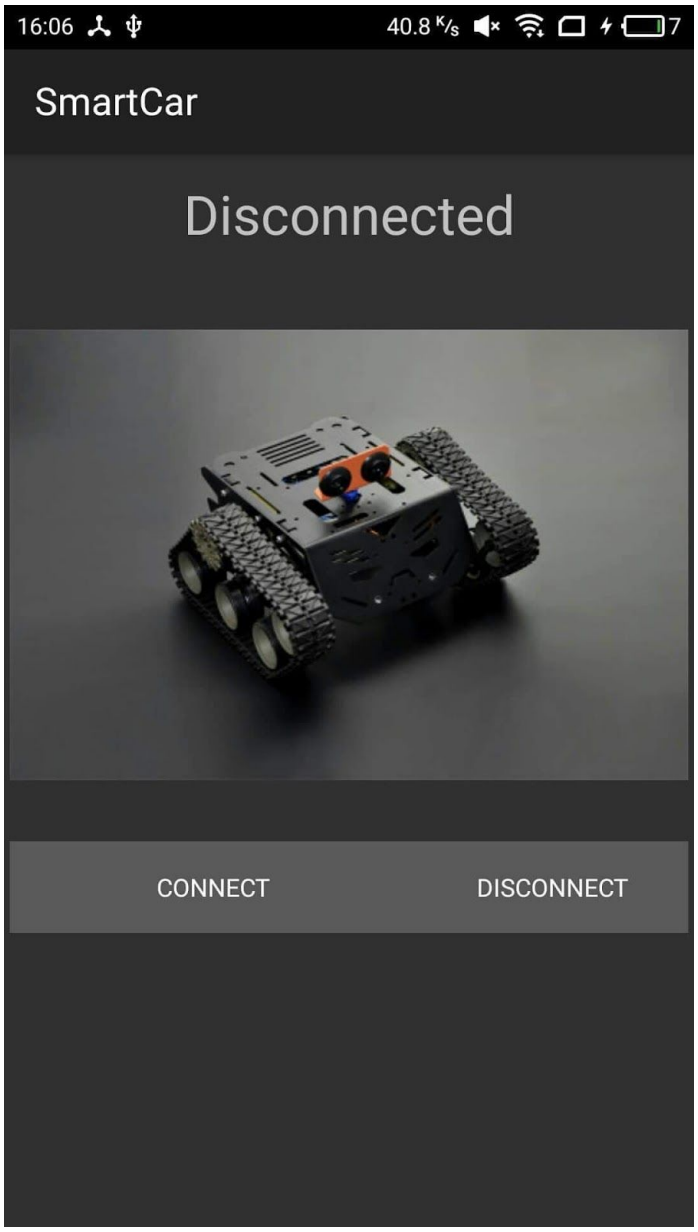
1.we have registered the domain name “gosmartcar.net” using the Route53 Service, which was easy to use.this step was necessary in order to obtain SSL certificate.

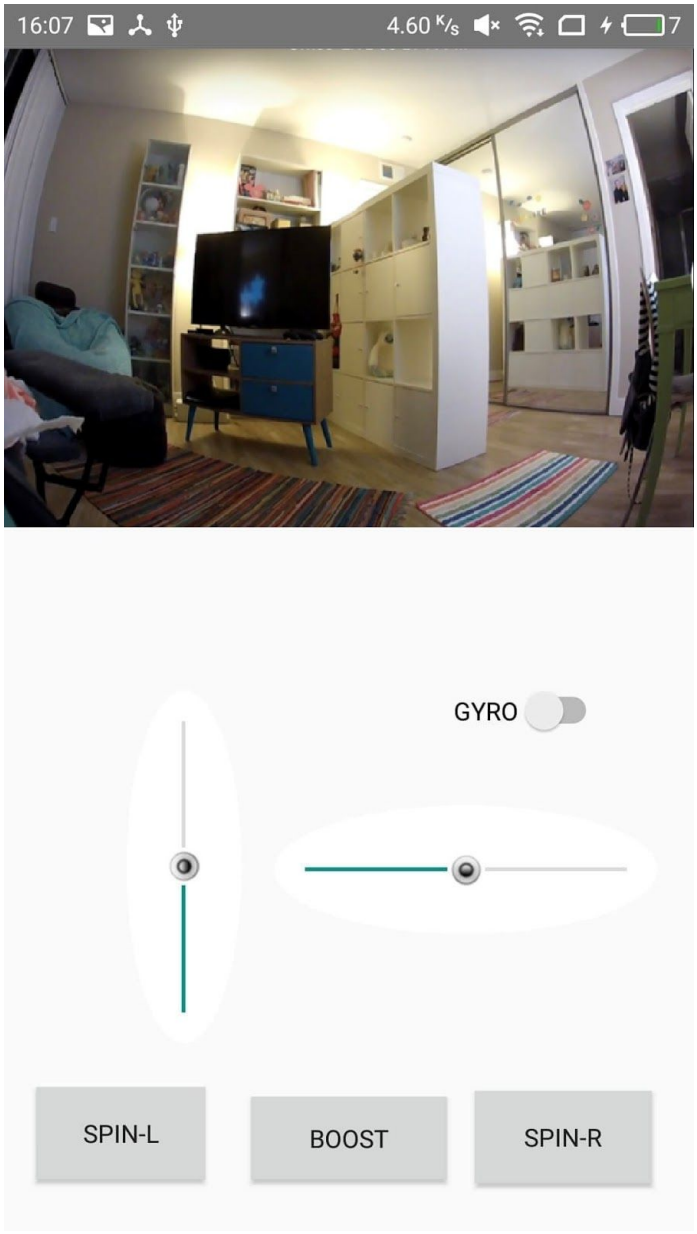
2.we had to create SSL certificate for our domain name “gosmartcar.net” and we’ve done that using the AWS Certificate Manager Service (ACM), that would generate for us an SSL certificate but we have to verify that we are the legitimate owners of this domain, so there was two options:
First by sending the owner an email to verify him, second is by adding a record to the DNS that we control after register the domain name, and the ACM service can do this for us, and we can now obtain a SSL certificate to use for our https connection.

3. In order to use the SSL certificate we had to create a load balancer since its not possible to copy them to the ec2 instance and using them there, but the ACM integrates with Elastic Load Balancing so that we can deploy the certificate on our load balancer, and like that we had to create the load balancer which clients who wish to access “https://gosmartcar.net” will reach him and he then by his rule will decrypt this traffic using the SSL certificate from the ACM and send it decrypted to our ec2 instance.

4. To decrypt the https traffic we have to first off all open the HTTPS port 443 on the load balancer and janus encrypted port 8089, and then use the SSL keys to decrypt this traffic and forwarded it to port 80 on the ec2,or 8088 if it was for janus service. And like that we handle a secure connection from the client how can now access the janus service securely.

Application API





Application primary components:

Web View:

Since we had to view the video from html page we had to use the web view

```
// web view set up
web_video = (WebView) findViewById(R.id.video);

web_video.getSettings().setJavaScriptEnabled(true);

web_video.setWebViewClient(new InsideWebViewClient());
web_video.setWebChromeClient(new WebChromeClient(){
    @TargetApi(Build.VERSION_CODES.LOLLIPOP)
    @Override
    public void onPermissionRequest(final PermissionRequest request) {
        request.grant(request.getResources());
    }
});

web_video.loadUrl("https://gosmartcar.net/SecCam.html");
```

We set JavaScriptenabled to true because we use html and java script content.
One should be careful with this enabling, since this my cause XSS(cross-site-scripting) attacks, but since we had written the java scripts by ourselves we can be more relaxed.

We had to grant the permissions at runtime, since only declaring the permission in the manifest is not enough for streaming media related stuff .

SeekBar:

To control the motors speed and direction we used the seek bar which range from 0 to 10 :

```
speedbar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener(){

    @Override
    public void onProgressChanged(SeekBar seekBar, int progress,
                                  boolean fromUser) {
        final String topic = "car";
        final String msg = "{\"cmd\": \"Speed\" , \"val\" :"+ Integer.toString(progress) + "}";

        publishAWS(topic,msg);

    }
```

After getting the seekbar value we compose a json message and send it to the AWS with the “car” Topic.

The same thing is for the direction Seekbar:


```
dirbar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener(){

    @Override
    public void onProgressChanged(SeekBar seekBar, int progress,
                                  boolean fromUser) {
        final String topic = "car";
        final String msg = "{\"cmd\": \"Dir\" , \"val\" :"+ Integer.toString(progress) + "}";

        publishAWS(topic,msg);
    }
```

Challenges Encountered

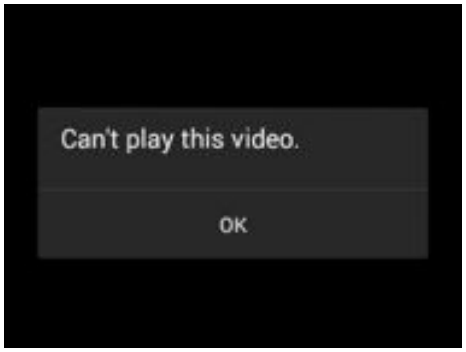
•Dealing with AWS

In our project, we aimed for Android application developing, that led us to rely on the AWSAndroid-sdk and its code samples on GitHub. The problem was that the "pubsub" code sample was committed to the code samples repository only few months ago and unlike other services there was no documentation for how to code/ use API, so the only reliable source for android MQTT enabling was that code sample which had errors in it and was not working.

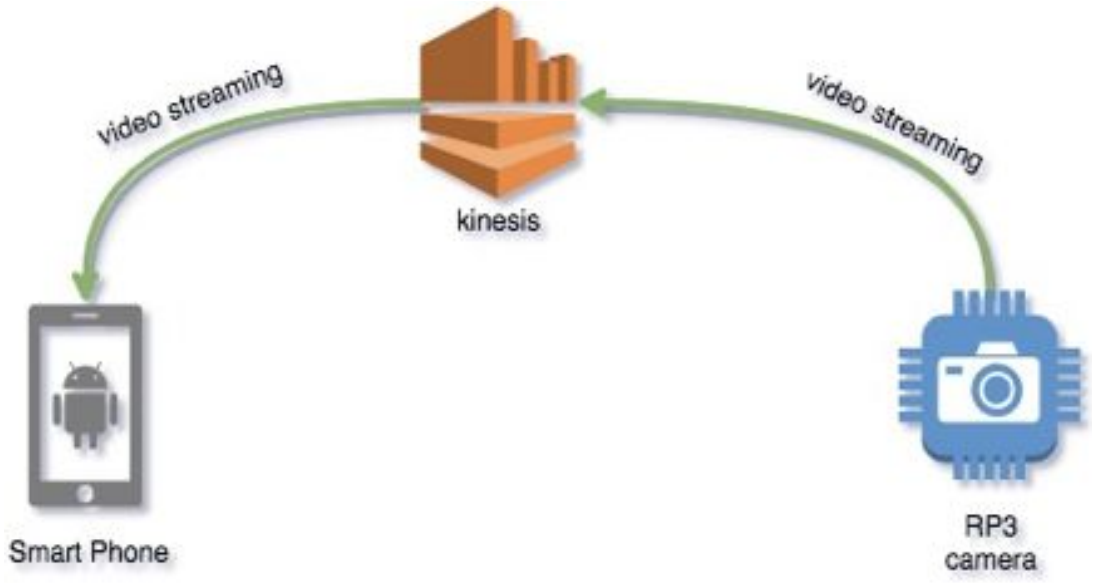
Another issue was that the code sample works with the basic default certificate and keystore file stand alone, and for integrating this code to our project, we needed to generate our own keystore file.

•Streaming live video

First try is streaming to web browser and it was fine and we get to ~100 ms delay, but when moved to stream through the android phone the video start freezing, we didn't know why for sure why in the beginning but later we found a out that android does not fully support h264 format, so we had to change the format to Mjpeg where we got with it ~200 ms.



Second try is to streams the video to the aws kinesis and then the android request the stream from the kinesis, from one side we used reliable and secure service which was a safe way to stream , but in another hand it wasn't good enough we got ~4 sec delay because of that we started to search for another way to stream .



Final attempt We needed fast way to transmit the video with minimum latency which is Peer to peer streaming using UV4L utility for Janus Gateway plugin called "Video Room" which have a built-in VP8 codec, which is supported by Google Chrome and therefore, the Android Web viewer.

- Burn raspberry pi :

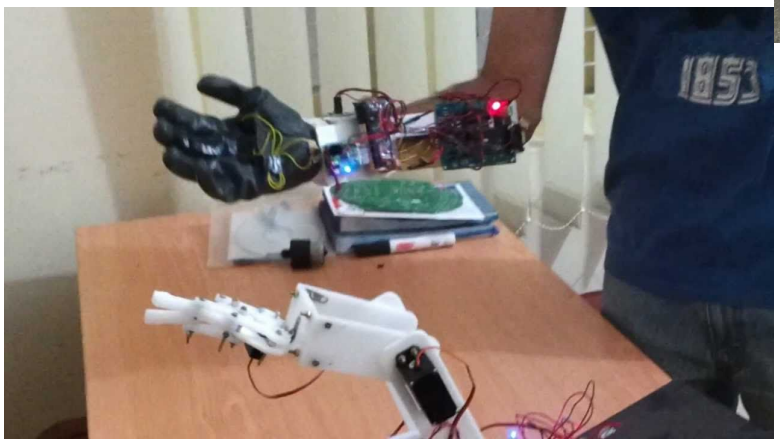
  At the first of work we used the GPIO pins power of the RP for the servo and the motor, it work well but after long work the RP suddenly burned due to high current we pull to the servo.
  The GPIO pins are natively 3.3V, so 5V devices MUST NOT be attached directly without some sort of voltage conversion. The pins can provide up to 16mA current.

Summary

In this project, we implemented a system with many different components.
We experienced new fields and acquired knowledge.
The IoT is a developing and relevant technology. We handled a network of physical devices
including an Android based smartphone, Raspberry PI and learned to manage it through AWS.
We learned about the different services of AWS and the amount of capabilities it includes.
We handled with streaming media and video in particular, with mobile application development,
Raspberry PI development, and many other technologies that we first met and had to face several
issues and constraints within almost every one of these technologies.
Most of all, we experienced the birth of a major project that we decided to take with no prior
knowledge. We searched for the right sources and had to conclude and interpret the existing
material into our own vision. We had to plan the work, the handle a schedule and to integrate
several sub-systems into one major functional system.
The knowledge acquired with this project and the challenge of facing new and unfamiliar subjects,
was exciting and enriching. The satisfaction from getting a finished physical product after months of hard learning and working was worth it all

Continuing projects

A glove wearing in hand that connected with the android app that can remotely control artificial hand on the tank



Bibliography

[1]The car we bought : [https://www.dfrobot.com/product-1219.html]

[2] Andrew Prokop, "Understanding WebRTC Media Connections: ICE, STUN and TURN" August 2014. [https://www.avaya.com/blogs/archives/2014/08/understandingwebrtc-media-connections-ice-stun-and-turn.html]

[3] "Raspberry PI, Wikipedia" -[https://en.wikipedia.org/wiki/Raspberry_Pi]

[4] "Raspberry PI documentation" -[https://www.raspberrypi.org/documentation/]

[5] "Tilt, Wikipedia" - [https://en.wikipedia.org/wiki/Tilt]

[6] "Motion sensors, Wikipedia" - [https://developer.android.com/guide/topics/sensors/sensors_motion.html]

[7] "VP8, Wikipedia" - [https://en.wikipedia.org/wiki/VP8]

[8] Micro-servo Specification - "Turnigy™ TG9e Eco Micro Servo" - [https://hobbyking.com/en_us/turnigytm-tg9e-eco-micro-servo-1-5kg-0-10sec-9g.html?__store=en_us]

[9] "Pigpio library" - [http://abyz.me.uk/rpi/pigpio]

[10] T. Zalophus, " Tilt Camera Mount - V1" - [https://www.thingiverse.com/thing:350229]

[11] "AWS EC2". [https://aws.amazon.com/documentation/ec2/]

[12] "AWS ACM". [https://docs.aws.amazon.com/acm/]

[13] "AWS IAM". [https://aws.amazon.com/documentation/iam/]

[14] "AWS IoT". [https://aws.amazon.com/documentation/iot/]

[15] "AWS Cognito". [https://aws.amazon.com/documentation/cognito/]

[16] "AWS Route53".[https://aws.amazon.com/route53/

[17] "AWS Loadbalancer".
[https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-elasticloadbalancingv2-loadbalancer.html]
[18] "Gateway, Wikipedia" - [https://he.wikipedia.org/wiki/Gateway]
[19] "STUN server, Wikipedia" -[https://en.wikipedia.org/wiki/STUN]
[20] "Signaling server, Wikipedia" - [https://en.wikipedia.org/wiki/Signaling_gateway]
[21] "Nnginx documentation" -[https://nginx.org/en/docs/]
[22] "UV4L documentation" - [https://www.linux-projects.org/documentation/]
[23] "Janus Gateway documentation" - [https://janus.conf.meetecho.com/docs/]
[24] "WebRTC" - [https://webrtc.org/]