

Relazione Tirocinio

Michele Nardini

December 6, 2022

Part I

Periodo: 14 novembre 2022 - 8 dicembre 2022

Durata: 150h

Abstract

Il tirocinio da me svolto aveva come obiettivo la sperimentazione di tecnologie di Augmented Reality finalizzate alla visualizzazione di informazioni del prodotto riconosciuto, più precisamente l'obiettivo del progetto era la costruzione di una sessione di Augmented Reality che permettesse la visualizzazione di informazioni in real time(per esempio luminosità) delle luci riconosciute dalla fotocamera del dispositivo mobile.

1 Introduzione

Alexide si presenta come una piccola realtà composta da circa 15 persone con l'obiettivo di realizzare soluzioni software in vari ambiti commerciali, che vanno dal web design alla realtà aumentata.

Per i primi giorni il tutor mi aveva affidato come obiettivo preliminare lo studio delle tecnologie che sarei andato ad utilizzare nel progetto.

Fatto ciò io e il tutor abbiamo fissato degli obiettivi iniziali al fine di verificare la fattibilità del progetto. Tali obiettivi erano l'inizializzazione di una sessione di realtà aumentata e l'aggiunta in scena di un oggetto 3d. Superato con successo questo step abbiamo cercato una soluzione per il riconoscimento delle singole luci dell'ufficio al fine di poter ottenere i dati di una specifica luce.

Per questo progetto ho adoperato come ambiente di sviluppo Visual Studio Code, mentre per ottenere i dati delle luci dell'ufficio ho sfruttato il protocollo di messaggistica MQTT di cui parlerò più avanti.

2 Obiettivi

- Creazione di una session AR
- Configurazione della scena e degli oggetti da posizionare
- Sviluppo di un algoritmo di riconoscimento delle luci
- Ottenimento dati sulle luci mediante MQTT.

3 Studi preliminari

3.1 Argomento 1: Definizione Three.js, webXR, MQTT

3.1.1 Three.js

Three.js è una libreria JavaScript e un'API utilizzata per creare e visualizzare grafica computerizzata 3D in un browser Web utilizzando WebGL.

Ci sono però alcuni concetti da conoscere prima di poter iniziare a lavorare con questa libreria.

Scena: nel contesto di Three.js, una scena rappresenta l'ambiente 3D principale dell'applicazione. Qui è dove posizioni oggetti, luci, telecamere.

Render: il renderer è la parte responsabile del disegno effettivo della nostra scena sullo schermo. Grazie a un ciclo di rendering, disegneremo ripetutamente la nostra scena sullo schermo con un dato intervallo.

Telecamera: la telecamera rappresenta la vista della scena da parte degli utenti.

Modelli e mesh: Three.js offre la possibilità sia di caricare modelli 3D prefabbricati, sia di creare oggetti 3D da zero.

3.1.2 webXR

L'API del dispositivo WebXR fornisce l'accesso alle funzionalità di input (informazioni di posa da cuffie e controller) e output (visualizzazione hardware) comunemente associate ai dispositivi di realtà virtuale (VR) e realtà aumentata (AR). Ti consente di sviluppare e ospitare esperienze VR e AR sul web.

Per i telefoni: Abilita la realtà virtuale fornendo informazioni sulla posa e consentendo il rendering della scena WebGL affiancata per essere posizionata in un visore come Cardboard. Abilita l'AR utilizzando le funzionalità AR della piattaforma come ARCore per eseguire il rendering della scena WebGL nell'ambiente degli utenti come una finestra magica.

Per i desktop: I computer desktop possono utilizzare hardware VR tethered come Oculus Rift o HTC Vive per visualizzare la scena VR

3.1.3 MQTT

MQTT è un protocollo di messaggistica standard OASIS per Internet of Things (IoT). È progettato come trasporto di messaggistica di pubblicazione/sottoscrizione estremamente leggero, ideale per connettere dispositivi remoti con un footprint di codice ridotto e una larghezza di banda di rete minima. MQTT oggi è utilizzato in un'ampia varietà di settori, come quello automobilistico, manifatturiero, delle telecomunicazioni, del petrolio e del gas, ecc. Veniamo a conoscenza di alcuni termini utili utilizzati in questo protocollo.

MQTT client: un client MQTT è un qualsiasi dispositivo (da un microcontrollore fino a un server completo) che esegue una libreria MQTT e si connette a un broker MQTT su una rete.

MQTT broker: Il broker è responsabile della ricezione di tutti i messaggi, del filtraggio dei messaggi, della determinazione di chi è iscritto a ciascun messaggio e dell'invio del messaggio a questi client sottoscritti.

4 Implementazione delle funzionalità

4.1 Creazione di una sessione AR

```
1
2 navigator.xr.requestSession( 'immersive-ar', sessionInit ).then(
    onSessionStarted );
```

Listing 1: Richiesta sessione AR

```
1
2 async function onSessionStarted( session ) {
3
4     session.addEventListener( 'end', onSessionEnded );
5     renderer.xr.setReferenceSpaceType( 'local' );
6
7     gl = renderer.getContext();
8     await renderer.xr.setSession( session );
9
10    currentSession = session;
11    currentSession.requestAnimationFrame(onXRFrame);
12 }
```

Listing 2: Creazione sessione AR

4.2 Creazione di una semplice scena AR

```
1 const scene = new THREE.Scene();
2 const camera = new THREE.PerspectiveCamera( 75, window.innerWidth /
    window.innerHeight, 0.1, 1000 );
3
4 const renderer = new THREE.WebGLRenderer();
5 renderer.setSize( window.innerWidth, window.innerHeight );
6 document.body.appendChild( renderer.domElement );
7
8 const geometry = new THREE.BoxGeometry( 1, 1, 1 );
9 const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } )
10 ;
11 const cube = new THREE.Mesh( geometry, material );
12 scene.add( cube );
13
14 camera.position.z = 5;
15
16 function animate() {
17     requestAnimationFrame( animate );
18
19     cube.rotation.x += 0.01;
20     cube.rotation.y += 0.01;
21
22     renderer.render( scene, camera );
23 };
24
25 animate();
```

Listing 3: Creazione di un cubo che apparirà una volta avviata la sessione AR



Figure 1: Creazione di un cubo e posizionamento davanti la fotocamera

4.3 Creazione della scena principale

Per la creazione delle singole mesh corrispondenti alle luci abbiamo preso le misure per il posizionamento di quest'ultime partendo dall'entrata principale dell'azienda e suddividendo l'ufficio in zone.

```
1 initMeshInScene(){
2   let count=0;
3   for (let c = 0; c < this.col; c++) {
4     for (let r = 0; r < this.row; r++) {
5       const lampMesh = createMesh(new THREE.Vector3(this.
        rowMeasure[r]*this.dist, this.height, this.
```

```

        colMeasure[c]*this.dist));
6      this.group.add(lampMesh);
7      this.lamps[count] = new Lamp(lampMesh, this.lampsName[
        count]);
8      count++;
9    }
10  }
11 }

```

Listing 4: Creazione della scena con posizionamento di mesh per il riconoscimento delle singole luci.



Figure 2: Creazione della scena

4.4 Associazione luci e mesh in scena

Per l'associazione delle mesh alle corrispondenti luci ho creato una semplicissima classe durante l'istanzamento di ogni mesh.

```

1 class Lamp{
2
3   constructor(mesh, name){
4     this.mesh=mesh;
5     this.name=name;
6   }
7
8   getName(){
9     return this.name;
10  }
11
12  getMesh(){
13    return this.mesh;
14  }
15
16 }

```

```
17  
18 export {Lamp};
```

Listing 5: Classe luci.

4.5 Creazione elemento HTML per la visualizzazione dei dati sulle luci

Create la scena, e effettuata l'associazione tra luci e mesh era il momento di creare l'oggetto html per la visualizzazione dei dati e la sua aggiunta poi in scena.

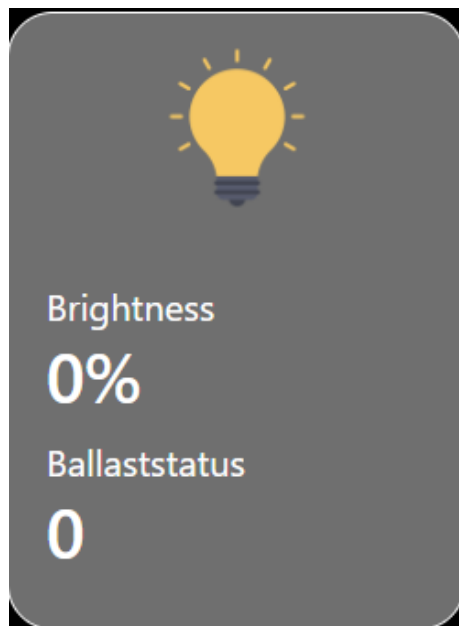


Figure 3: Elemento html per visualizzazione dati lampada



Figure 4: Visualizzazione dati lampada in scena

Per la visualizzazione dell'elemento html in scena c'erano più strade, in comune accordo con il tutor abbiamo deciso di sfruttare lo script html2canvas. Lo script permette di acquisire "screenshot" di pagine web o parti di esse, direttamente sul browser dell'utente. Lo script attraversa il DOM della pagina su cui è caricato. Raccoglie informazioni su tutti gli elementi presenti, che poi utilizza per costruire una rappresentazione della pagina. In altre parole, in realtà non acquisisce uno screenshot della pagina, ma ne costruisce una rappresentazione basata sulle proprietà che legge dal DOM. Acquisito lo screenshot l'ho convertito in una texture in modo tale da applicarlo ad una mesh di tipo plane e visualizzarlo in scena.

```

1 export function createTextLamp(mesh){
2   html2canvas(document.querySelector("#card-light"),{
3     backgroundColor:null}).then(canvas => {
4     const geometry = new THREE.PlaneGeometry(1, 1.2);
5     const texture = new THREE.CanvasTexture(canvas);
6     texture.wrapS = THREE.RepeatWrapping;
7     texture.wrapT = THREE.RepeatWrapping;
8     const plane = new THREE.Mesh(geometry, new THREE.
9       MeshBasicMaterial({
10        map: texture,
11        transparent: true
12      }));
13     plane.material.side = THREE.DoubleSide;
14     mesh.add(plane);
15     plane.position.set(0,0,1);
16     plane.userData = { URL: "../consumi.html" };
17   });

```

Listing 6: Funzione che acquisisce il dom dei dati della lampada

Oltre alle informazioni della lampada, cliccando sul plane appena creato, si viene reindirizzati ad una pagina(al momento statica e temporanea) dove è possibile visualizzare il consumo giornaliero.



Figure 5: Visualizzazione consumi

4.6 Utilizzo MQTT per l'ottenimento dei dati sulle lampade

Per poter ottenere i dati attraverso MQTT ho dovuto creare un client mqtt che si collegasse al broker aziendale contenente tutti i dati sui dispositivi domotici, tra cui quelli delle lampade.

```
1
2 topic = 'SolidRules/Machine/INTESIS_DALI/Status';
3 hostname='wss://192.168.0.7:9004/ws';
4
5 options = {
6     // Clean session
7     clean: true,
8     connectTimeout: 4000,
9     protocol: "wss",
10    // Authentication
11    //useSSL: true,
12    rejectUnauthorized: false,
13 }
14
15 const client = mqtt.connect(this.hostname, this.options);
16
17 client.on('connect', () => {
18     console.log('Connected')
19 })
```

Listing 7: Creazione client MQTT e connessione al broker MQTT

Affinchè il client possa ricevere o mandare dei messaggi al broker, quest'ultimo deve iscriversi ad un determinato topic.

```
1
2 client.subscribe([this.topic], () => {
3     console.log('Subscribe to topic '${this.topic}')
```

Listing 8: Iscrizione al topic

```
1
2
3
4 class MqttClient{
5
6     dataLamps = null;
7     topic = 'SolidRules/Machine/INTESIS_DALI/Status';
8     hostname='wss://192.168.0.7:9004/ws';
9     needToUpdate=false;
10
11
12     options = {
13         // Clean session
14         clean: true,
15         connectTimeout: 4000,
16         protocol: "wss",
17         // Authentication
18         //useSSL: true,
```

```

19     rejectUnauthorized: false,
20   }
21
22   constructor(){
23
24     const client = mqtt.connect(this.hostname, this.options);
25
26     client.on('connect', () => {
27       console.log('Connected')
28       client.subscribe([this.topic], () => {
29         console.log('Subscribe to topic `${this.topic}`')
30       })
31     })
32
33     client.on('message', (topic, payload) => {
34       console.log('Received Message:', topic, payload.
35         toString())
36       this.dataLamps = JSON.parse(payload.toString());
37       this.needToUpdate = true;
38     })
39
40     getData(){
41       return this.dataLamps;
42     }
43
44     getNeedToUpdate(){
45       return this.needToUpdate;
46     }
47
48     setNeedToUpdate(needToUpdate){
49       this.needToUpdate=needToUpdate;
50     }
51   }
52 }
53
54 export {MqttClient};

```

Listing 9: Classe completa

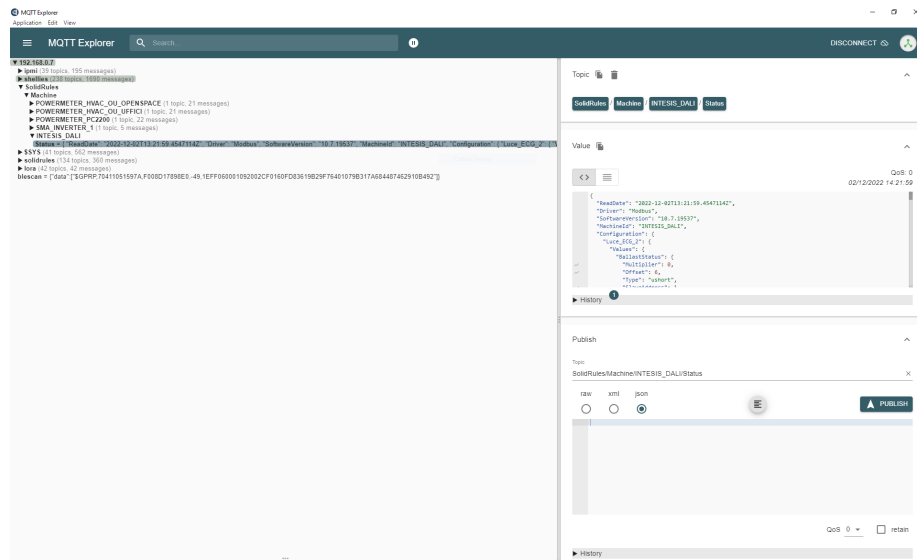


Figure 6: Broker MQTT

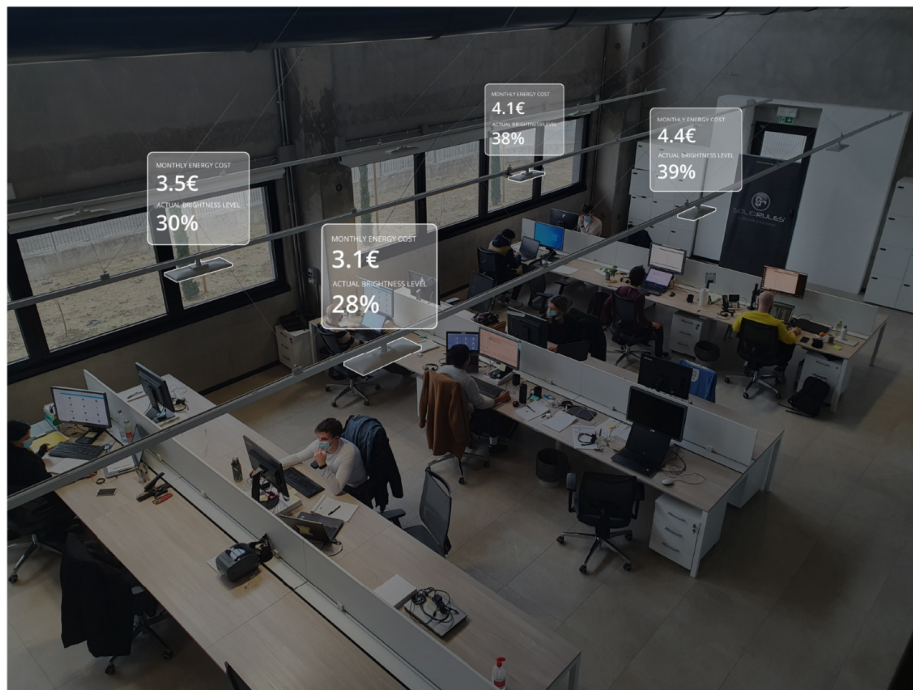


Figure 7: Aspettative



Figure 8: Realta'

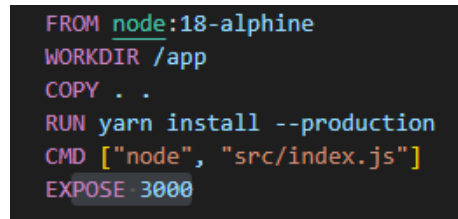
5 Bonus

5.1 Docker

Docker è una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità. Docker raccoglie il software in unità standardizzate chiamate container che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime. Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, tenendo sempre sotto controllo il codice eseguito. L'esecuzione di Docker in AWS offre a sviluppatori e amministratori un modo altamente affidabile e poco costoso per creare, spedire ed eseguire applicazioni distribuite su qualsiasi scala. Un container è un processo in modalità sandbox sulla tua macchina che è isolato da tutti gli altri processi sulla macchina host. Tale isolamento sfrutta gli spazi dei nomi del kernel e i cgroup, funzionalità presenti in Linux da molto tempo. Docker ha lavorato per rendere queste funzionalità accessibili e facili da usare.

5.1.1 Creare un container

Per creare l'immagine del container, dovrai utilizzare un file Dockerfile. Un Dockerfile è semplicemente un file di testo senza estensione di file. Un Dockerfile contiene uno script di istruzioni che Docker utilizza per creare un'immagine del contenitore.



```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

Figure 9: Sintassi DockerFile'

Ogni Dockerfile deve iniziare con l'istruzione FROM. L'idea alla base è che hai bisogno di un punto di partenza per costruire la tua immagine. L'istruzione RUN indica a docker di eseguire le istruzioni specificate, come per esempio l'installazione di dipendenze. L'istruzione COPY copierà il codice sorgente mentre l'istruzione EXPOSE informerà a DOCKER su che porta di rete sarà in ascolto.

```
1 $ docker build -t getting-started .
```

Il comando docker build utilizza il Dockerfile per creare una nuova immagine del contenitore. Dopo che Docker ha scaricato l'immagine, le istruzioni del

Dockerfile vengono copiate nell'applicazione e utilizzate attraverso yarn per installare le dipendenze dell'applicazione. Fatto tutto ciò ora potrai avviare la tua applicazione attraverso il container docker attraverso il seguente comando:

```
1 $ docker run -dp 3000:3000 getting-started
```

Ora dopo qualche secondo di attesa, potrai vedere la tua app su <http://localhost:3000>.

5.1.2 Rimuovere un container

Per poter rimuovere un container creato sono necessari 3 step:

- Ottenere l'id del container da eliminare

```
1 $ docker ps
```

- Arrestare il container in esecuzione

```
1 $ docker stop <the-container-id>
```

- Rimuovere il container

```
1 $ docker rm <the-container-id>
```

6 Risultati

In definitiva siamo riusciti ad implementare tutte le funzionalità che erano richieste dal progetto, anche se effettuando dei cambiamenti sulle modalità, in particolare abbiamo accantonato l'idea iniziale di riconoscere ogni singola lampada mediante un algoritmo di Machine Learning che purtroppo sarebbe stato molto complicato e oneroso da implementare a causa della grande similitudine delle immagini che sarebbero state adoperate per allenare l'intelligenza artificiale e ciò non avrebbe permesso un efficiente riconoscimento delle immagini.

7 Commenti

Questa esperienza in definitiva mi ha insegnato molte cose, da quelle più personali a quelle più professionali. Partendo da quelle professionali, sono molto contento e soddisfatto di essere entrato in contatto con questa tecnologia che è in continuo sviluppo ogni singolo giorno e che può essere adoperata in ogni ambito, che sia esso commerciale, videoludico, etc. Ho imparato molte cose anche grazie ai Webinar che sono stati fatti in ufficio, dove ho fatto conoscenza di Docker, un software che permette di creare, testare e distribuire applicazioni con la massima rapidità. A livello personale ho imparato molte cose, ho affrontato molti ostacoli e ne ho superati altrettanti. Essendo la mia prima vera e propria esperienza lavorativa da sviluppatore i primi giorni per me sono stati abbastanza duri, avevo paura di non essere all'altezza o adeguato a questo progetto, ma ho capito che l'unico modo che avevo per superare questo ostacolo che da solo mi ero creato era quello di chiedere aiuto proprio a chi mi ha voluto qui e che non dovevo aver paura di chiedere; questo è sicuramente l'insegnamento più grande che mi porterò dietro. Il clima all'interno dell'azienda è ottimo, ciò grazie ai buoni rapporti che si sono creati con i colleghi, basati sul reciproco aiuto.