Sviluppo di un visualizzatore di dati di telemetria satellitari applicato alla missione ESEO

Università degli Studi di Bologna – campus di Forlì
Dipartimento di Ingegneria e Architettura
Laurea Triennale in Ingegneria Aerospaziale

Candidato: Francesco Mazzeo

Prof. Relatore: Paolo Tortora

Anno accademico 2018 – 2019

INDICE

Introduzione

Il lavoro di tesi svolto consiste nello sviluppo di un visualizzatore 3D di dati di telemetria per il satellite ESEO. L'elaborato che segue si articola in 3 capitoli principali: il primo farà una veloce panoramica sulla missione ESEO, l'architettura del satellite e la sua orbita; il secondo capitolo invece descriverà la prima parte del lavoro che si è svolto sui dati di telemetria del satellite, la creazione di visualizzatori basati sulla piattaforma Grafana; infine il terzo capitolo tratterà lo sviluppo di un visualizzatore 3D progettato appositamente per ESEO, utilizzando il tool CesiumJS.

La filosofia alla base del lavoro è rendere i dati inviati dal satellite in un formato "human readable", ovvero in una rappresentazione che sia naturalmente comprensibile e analizzabile dall'uomo. Questo significa che i dati inviati dal satellite alla Ground Station devono essere convertiti e organizzati in una visualizzazione che permetta all'operatore di interagirci ed analizzarli. Per fare un esempio pratico di questa necessità, si consideri uno dei parametri che il satellite invia ad ogni passaggio sotto forma di sequenza di bit, la temperatura del Power System. Per l'uomo risulta impossibile leggere direttamente la stringa binaria ricevuta, e quindi avere un'idea del valore di quella temperatura, né tantomeno del suo andamento nel tempo. Se invece questi dati vengono convertiti in cifre decimali e successivamente organizzati all'interno di un grafico, risulterà molto più immediato comprenderne l'andamento e l'intensità. Specialmente nel secondo capitolo di questa tesi, questo processo è ampliamente descritto. Il risultato finale consiste in una serie di pagine su Grafana in cui i dati sono visualizzati sotto forma di grafici, tabelle e contatori.

Per quanto riguarda invece lo sviluppo del visualizzatore 3D, lo scopo rientra sempre nello stesso ambito. Realizzare un'applicazione in cui il satellite compare come un modellino 3D che ruota attorno al globo terrestre seguendo la sua orbita, permette di avere un'idea molto immediata sulla posizione, l'assetto e la velocità del satellite. Inoltre è stata implementata una funzione che, grazie ad un'architettura di tipo REST, scarica direttamente dal database i valori reali e aggiornati all'ultimo passaggio, degli angoli di assetto del satellite, e direttamente dal sito CelesTrack i Two Line Elements.

Il lavoro si è articolato come in Figure 1: Mappa concettuale. Per prima cosa si è studiato il funzionamento di Grafana, le tipologie di pannelli che permette di creare e le query necessarie ad interrogare il database. Successivamente sono state implementate sulla piattaforma le pagine di OBDH HK e TC, seguendo i documenti di sistema i ESEO. Fatto ciò si è passati quindi allo studio della libreria Cesium e del framework NodeJS per la progettazione del visualizzatore 3D che si è voluto caricare all'interno di un pannello di Grafana. È stato necessario sviluppare un server apposito per il programma ed un'architettura di tipo REST per scaricare gli angoli di assetto e i TLE aggiornati. Inoltre si è scaricato un apposito modulo di Node, SGP4, che implementasse delle funzioni necessarie al calcolo di posizione e velocità del satellite. Lo scenario è stato quindi costruito utilizzando la libreria JavaScript di Cesium e integrando all'interno dello script le funzioni di SGP4 e i dati provenienti dall'architettura REST. Infine il programma è stato caricato su un apposito pannello di Grafana e messo online.

Questo lavoro di tesi è risultato prezioso per l'acquisizione di conoscenze nel campo delle missioni spaziali, sperimentando da vicino come si lavora in un Mission Control Centre e come i dati di telemetria di un satellite vengono ricevuti e analizzati dagli utenti. Da notare inoltre come questo lavoro ha apportato un contributo tangibile all'interno di un importante progetto come ESEO, dato che le applicazioni sviluppate saranno utilizzate operativamente e potranno essere replicate per altre missioni future.

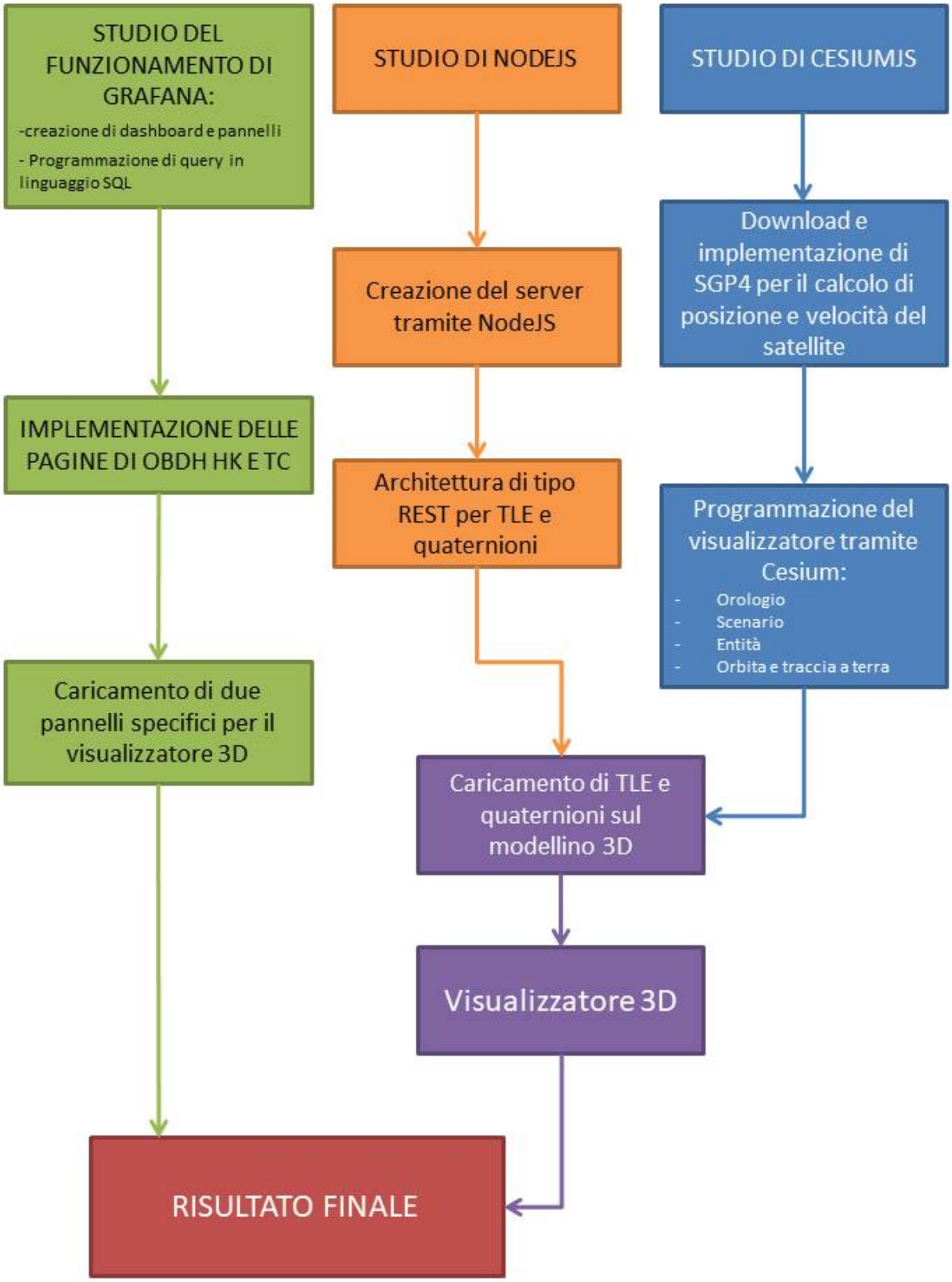

Figure 1: Mappa concettuale

1 ESEO

1.1 Missione

Figure 2: ESEO

ESEO (*European Student Earth Orbiter*) è un micro-satellite in orbita bassa (*LEO*) eliosincrona, progettato e sviluppato interamente da studenti di dieci università europee, tra cui l'Università di Bologna, e da SITAEL, che riveste il ruolo di partner industriale.

È la terza missione (dopo *SSETI Express* e *YES-2*) del progetto ESA *(European Space Agency*) Education Satellite Programme che ha lo scopo di formare studenti e neolaureati nel campo della progettazione di satelliti e missioni spaziali.

Il lancio è avvenuto il 3 dicembre 2018 alle 19,34 (ora italiana) dalla Vandenberg Air Force Base in California (USA), a bordo del vettore SpaceX Falcon 9, come parte della missione SSO-A.

Il satellite resterà attivo in orbita per 6 mesi, al termine dei quali la sua missione potrà essere rinnovata per altri 12.

Gli obiettivi della missione ESEO consistono in esperimenti scientifici, test di varie tecnologie sviluppate da studenti oltre che un costante monitoraggio delle proprie attività.

I principali sono i seguenti:

- Scattare foto alla Terra e ad altri corpi celesti, tramite l'utilizzo della microcamera uCAM, che opera nello spettro del visibile;
- Misura dei livelli di radiazione in un' orbita LEO e i suoi effetti sui componenti del satellite.
  Questo è possibile grazie ad appositi sensori LMP e TRITEL;
- Testare varie tecnologie sviluppate da team universitari, quali:
  - un sistema di comunicazione a banda-S (HSTX);
  - un ricevitore GPS per la determinazione della posizione del satellite sull'orbita LEO;

  - un sistema DOM (*De-Orbit Mechanism*);
  - il software ADE (*Attitude Determination Estimator*) per verificare l'efficienza di quattro differenti algoritmi per la determinazione dell' assetto e posizione del satellite;
- Consentire a radioamatori di connettersi al satellite tramite la banda VHF, grazie al payload AMSAT.

Il satellite di dimensioni 33x33x66 centimetri, a base quadrata, ha un peso di circa 50 kg ed è composto da due moduli principali: BM (*Bus module*) e PM (*Payload Module*) contenenti tutti i sottosistemi e i payload.

1.2 Architettura del satellite

1.2.1 Bus Module

Il BM comprende i sottosistemi di ESEO, che sono situati all'interno di cassetti (*Tray*) in alluminio, necessari a separare i vari componenti ed irrigidire la struttura.

Vi sono in totale 6 cassetti, più i pannelli laterali e la piattaforma superiore.

- **Tray 1**: contiene i micro-propulsori dell'AOCS (*Attitude and Orbital Control System*) necessari per manovrare il satellite;
- **Tray 2**: contiene MWM e MWR (*Momentum Wheel Main/Redundant)* e una coppia di MT (*MagnetoTorquer)* lungo l'asse –Y;
- **Tray 3**: contiene i componenti principali del PS (*Power System)* ovvero il PMB *(Power Management Board)*, PDU *(Power Distribution Unit*) e BPs (*Battery Packs*);
- **Tray 4**: contiene l'OBDH (*On-Board Data Handling)* ossia la sede centrale delle connessioni tra i vari componenti e sottosistemi;
- **Tray 5**: contiene una coppia di Magnetometri e il sistema GPS (*Global Positioning System*);
- **Tray 6**: contiene i sottosistemi di telemetria e telecommando (TMTC);
- **Pannelli laterali**: contiene le SA (*Solar Arrays*), celle per la produzione di energia solare;
- **Piattaforma superiore**: contiene i SS (*Sun Sensors*), l'antenna UHF, GPS e le antenne AM-SAT;

1.2.2 Payload Module

Come nel Bus Module, i payloads sono contenuti in supporti d' alluminio che hanno appunto il compito di separare e proteggere i sistemi che in questo caso sono le strumentazioni necessarie per il compimento degli obiettivi della missione.

Sulla piattaforma inferiore sono situati tutti i payload che necessitano di un accesso esterno per il puntamento di Nadir, come LMP, uCAM, TRITEL e HSTX. Anche gli ES (Earth Sensor) sono situati sulla piattaforma inferiore dato che necessitano di una visuale sulla Terra.

Ogni payload e sottosistema, identificato da un prefisso, è elencato nella Tab.1 sottostante.

Equipment/Payload	Prefix
OBDH equipment	OBD
TMTC main	TTM
TMTC redundant	TTR
Power Management Unit main	PMM
Power Management Unit redundant	PMR
AOCS algorithms	ACS
Sun sensor main	SSM
Sun sensor redundant	SSR
Earth sensor	ESE
Magnetometer main	MMM
Magnetometer redundant	MMR
Magnetic Torquer main	MTM
Magnetic Torquer redundant	MTR
Momentum Wheel main	MWM
Momentum Wheel redundant	MWR
TRITEL	TRI
Langmuir Probe	LMP
uCAM	CAM
De-orbit mechanism	DOM
AMSAT-UK	AMS
S-Band transmitter	STX
GPS receiver	GPS
TT-Deflt attitude determination experiment	ADE
High Priority Command	HPC

Table 1: Sottosistemi e Payloads

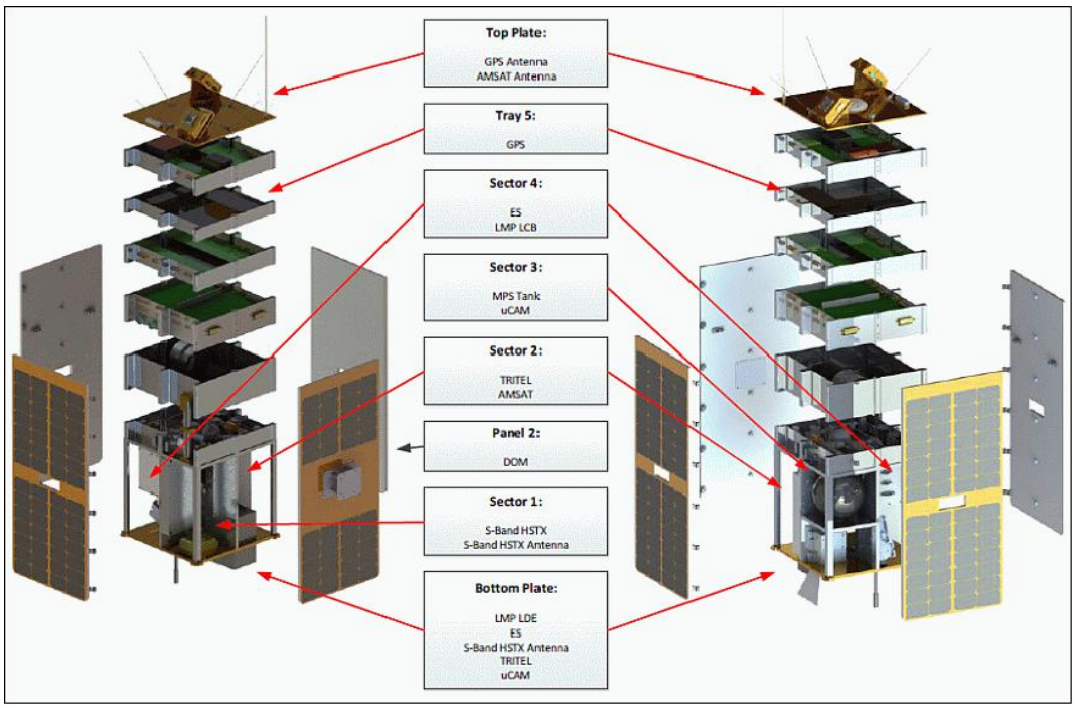

Figure 3: Sottosistemi di ESEO

1.3 Orbita

1.3.1 Descrizione dell'orbita target di ESEO

L'orbita scelta per la missione ESEO è un'orbita circolare bassa (LEO) eliosincrona.

La caratteristica principale di questo tipo di orbita è quella di mantenere un angolo tra il piano orbitale e l'asse satellite-sole costante. Per questo motivo l'orbita eliosincrona è utilizzata per missioni di '*Remote sensing*' perché garantisce sempre una condizione di illuminazione ideale per osservare la Terra. Per realizzarla è necessario avere un'orbita circolare (quota costante) e che ruoti il suo piano orbitale di 0,98° al giorno

Si può dimostrare che tale effetto si ottiene sfruttando la perturbazione J2 (asfericità della terra) con un'orbita circa polare (inclinazione i tra 96° e 98°).

$$\Delta\Omega_{tot} = -\frac{3}{2} J_2 \sqrt{\mu} R_t^2 a^{-\frac{7}{2}} \cos i \cdot 86400$$

$$con\ \Delta\Omega_{tot} = 0{,}98\,^{\circ}/_{gg}$$

Sono riportati in Table 2**: Parametri orbitali ESEO** i parametri orbitali dell'orbita di ESEO.

Parametro	Misura
Semiasse maggiore	6946 km
Eccentricità	0.00144670°
Inclinazione	97.47884°
Argomento del nodo ascendente	~155°

Table 2: Parametri orbitali ESEO

1.3.2 Visualizzazione orbita: SGP4 e TLE

SGP4 – Simplified General Perturbation #4

Sebbene il moto di un satellite attorno ad un corpo celeste sia sostanzialmente descritto da un moto kepleriano (problema dei due corpi ristretto), esistono anche forze di perturbazione di ordine inferiore, che ne modificano la traiettoria.

Alcuni esempi di perturbazione possono essere il fenomeno di 'drag atmosferico', l'asfericità del corpo attorno a cui il satellite ruota, la presenza di altri corpi, la radiazione solare ecc.

SGP4 è un modello semi-analitico per la determinazione della posizione e della velocità di un satellite che tiene conto di una parte di queste perturbazioni e fornisce una misura abbastanza accurata dei due vettori e dei parametri orbitali che ne derivano.

Solitamente con SGP4 ci si riferisce ad un set di cinque modelli di perturbazione: SGP, SGP4, SGP8, SDP4 (*Simplified Deep space Perturbation model*) e SDP8.

La distinzione principale tra SGP4 e SDP4 è che il primo modello viene utilizzato per orbite con periodo orbitale T < 225', mentre il secondo per T > 225'.

L'accuratezza di questo modello dipende dall'orbita considerata (Table 3: Accuratezza del modello SGP4).

Orbita	Precisione [km]
LEO, NSO	~ 1
MEO	$1 \div 2$
GEO	$2 \div 4$
HEO, GTO	$4 \div 6$

Table 3: Accuratezza del modello SGP4

L'algoritmo di SGP4 calcola, a partire dai TLE dell'orbita un vettore di questo tipo

$$X = (t_0, B^*, i_0, \Omega_0, e_0, \omega_0, M_0, n_0)$$

dove:

- t_0: periodo orbitale;
- B^*: coefficiente di drag atmosferico;
- i_0: inclinazione;
- Ω_0: argomento del nodo ascendente;
- e_0: eccentricità;

- ω_0: argomento al perigeo;
- M_0: anomalia media;
- n_0: moto medio (rivoluzioni al giorno);

Per questa tesi, lo script utilizzato per il calcolo di SGP4, è stato scaricato come modulo di Node.js al link in bibliografia (Download modulo SGP4).

Lo stesso codice è stato poi modificato e adattato per il calcolo dei parametri di ESEO.

TLE – Two Line Elements

Un set TLE (*Two Line Elements*) è un set di parametri orbitali di un satellite in orbita attorno alla terra, codificati in due linee di 69 caratteri. Sono utilizzati dai vari modelli SGP per calcolare e predire posizione e velocità del satellite. Gli unici caratteri consentiti nei TLE sono numeri da 0-9, lettere A-Z, il segno +, - e lo spazio.

```
ESEO

1 43792U 18099AL  19146.18212367 -.00000357  00000-0 -26939-4 0  9993
2 43792  97.7446 218.5326 0014467  52.7760 307.4778 14.95498635 25670
```

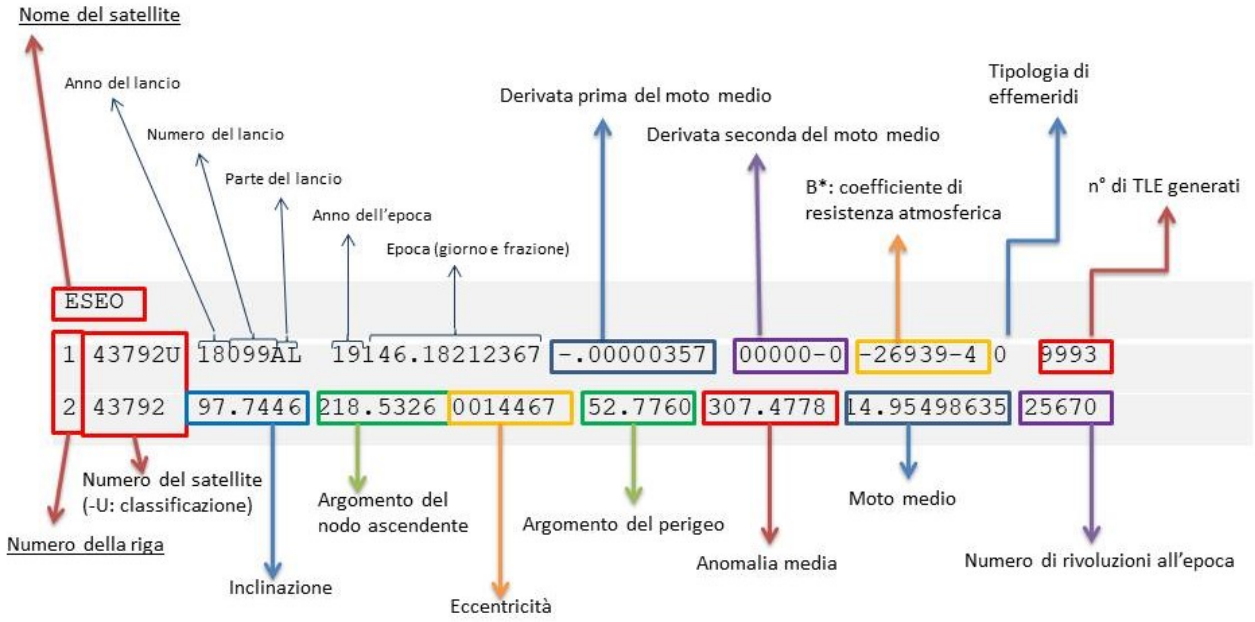

Figure 4: Two Line Elements

I TLE sono generati e mantenuti dal NORAD e scaricabili dal sito CelesTrack (CelesTrack ESEO).

1.4 Protocolli di telemetria e telecomando

Il protocollo di comunicazione tra ESEO e la GS (Ground Station) è un derivato del noto protocollo AX.25 (Link Access Protocol for Amateur Packet Radio). Viene utilizzato dal TMTC di bordo e dalla GS per inviare/ricevere telecomandi, dati di telemetria e acknowledges/rejections (ok/rifiuto).

Le informazioni vengono inviate in piccoli pacchetti di dati chiamati "*Frames*". Nell' AX.25 esistono 3 tipologie di frames:

- Information Frames (I frames)
- Supervision Frames (S frames)
- Unnumbered (U frames)

Ogni frame è composto da diversi sottopacchetti, chiamati *"Fields"* (campi). U e S contengono 5 campi, per un totale di 21 e 22 bytes, invece I può avere una lunghezza variabile in funzione del suo *info field*, che va da 128 a 151 bytes. Di seguito le tabelle di struttura dei 3 frames.

First Byte				
Flag	**Address**	**Control**	**FCS**	**Flag**
0x7E7E	14 Bytes	1 Byte	2 Bytes	0x7E7E

Table 4: U frame

First Byte				
Flag	**Address**	**Control**	**FCS**	**Flag**
0x7E7E	14 Bytes	2 Bytes	2 Bytes	0x7E7E

Table 5: S frame

First Byte						
Flag	**Address**	**Control**	**PID**	**Info**	**FCS**	**Flag**
0x7E7E	14 Bytes	2 Bytes	1 Byte	N Bytes	2 Bytes	0x7E7E

Table 6: I frame

- *Flag field:* composto da 2 byte, delimita gli estremi del frame con una particolare sequenza di bit: 0111111001111110. Si trova all'inizio e alla fine del pacchetto;
- *Address field:* è composto da 14 byte, di cui 7 contengono la sorgente del pacchetto e 7 il destinatario;
- *Control field:* definisce il tipo di frame ed è composto da 2 byte nell' I frame ed S frame, e 1 byte nell'U frame;
- *FCS (Frame Check Sequence):* contiene un numero di 16 bit generato contemporaneamente da sorgente e destinatario, ed è necessario per assicurare l'integrità del dato.
- *PID (Protocol Identifier):* indica quale protocollo del layer 3 è stato utilizzato. Nel caso di ESEO nessuno, quindi la sua sequenza sarà 11110000 (0xF0 in esadecimali).
- *Info field:* contiene i dati di TC (Telecommand) e HK (Housekeeping) inviati.

Segue una descrizione di come il protocollo AX.25 è stato implementato all'interno di ESEO per gestire la comunicazione tra i due sottosistemi OBDH (*On-Board Data Handling)* e TMTC *(Telemetry and Telecommand)*. Per quanto riguarda la comunicazione tra Ground Segment e TMTC, il protocollo è analogo ma opera via onde radio.

L'interfaccia utilizzata dall'OBDH ha le seguenti funzionalità:

- ricevere TC standard dal TMTC. Tali comandi vengono generati nella Ground Station ed inviati con lo stesso procedimento al TMTC del satellite;
- trasmettere i risultati di un TC (ex. Inviare un parametro che è stato richiesto);
- ricevere e trasmettere risultati di un time – tagged TC;
- trasmettere i dati dei beacon;
- richieste di TC dedicate:
    - trasmettere *HK pages* complete
    - trasmettere *HK history* di determinati dati (per un massimo di 128 bytes)
    - trasmettere *PS high rate history*
    - trasmettere report sullo stato della strumentazione
    - immagini dell'Earth Sensor
    - dati e info sulla Power Unit *(Power System Main Bus Data)*

Il pacchetto di dati standard inviato da e per ESEO ha una struttura come quella riportata in Table 7: Pacchetto standard scambiato tra OBDH e TMTC.

First byte					
Start Flag	**Seq. number**	**Class**	**Length**	**Data**	**CRC**
0xAA	1 byte	1 byte	1 byte	M bytes	2 bytes

Table 7: Pacchetto standard scambiato tra OBDH e TMTC

Descrizione dei vari campi:

- *Start Flag:* sequenza di bit che delimita l'inizio del pacchetto;
- *Sequence number:* identifica il pacchetto ed è utilizzato nelle comunicazioni di acknowledge/rejection con il trasmettitore di terra;
- *Classe:* definisce la classe del pacchetto. Le possibili classi sono elencate in Table 8: Classi del pacchetto standard.

Class value	Packet definition	Direction
0x00	Standard TC	OBDH ← TMTC
0x01	Time tagged TC	OBDH ← TMTC
0x02	TC Acknowledge/rejection message	OBDH → TMTC
0x03	Beacon of type 1: general	OBDH → TMTC
0x04	Beacon of type 2: power system	OBDH → TMTC
0x05	Beacon of type 3: OBDH	OBDH → TMTC
0x06	Beacon of type 4: AOCS and sensor/actuators	OBDH → TMTC
0x07	Beacon of type 5: FDIR-TMTC	OBDH → TMTC
0x08	Beacon of type 6: payload	OBDH → TMTC
0x09	HK data page	OBDH → TMTC
0x0A	HK history segment	OBDH → TMTC
0x0B	PS high rate history segment	OBDH → TMTC
0x0C	Death report	OBDH → TMTC
0x0D	ES IR image	OBDH → TMTC
0x0E	TMTC Acknowledge/rejection message	OBDH ← TMTC
0x0F	PS main bus high rate history data	OBDH → TMTC

Table 8: Classi del pacchetto standard

- *Length:* lunghezza del campo *Data*;
- *Data:* è il dato che viene scambiato, può variare in base alla classe del pacchetto;

- *CRC:* è un numero intero a 16 bit e controlla l'affidabilità del pacchetto dopo che la ricezione/trasmissione è avvenuta. Controlla che il dato non sia stato corrotto, in accordo con il protocollo di comunicazione AX.25.

Di seguito viene fatta una descrizione più precisa del campo *Data*, che corrisponde nell'AX.25 al campo che era stato denominato *Info* dell'I frame.

In base al tipo di dato, il campo assume una struttura particolare.

- **Standard TC**

Standard TC Data field			
Address	**Sub-address**	**Type**	**Register value**
1 byte	1 byte	1 byte	4 bytes

Table 9: Standard TC Data field

Dove:

- *Address* indica l'emittente del dato;
- *Sub-address* indica lo strumento selezionato per quel comando;
- *Type* indica il tipo di comando che si vuol dare. Esistono due tipo di TC e sono SET e GET;
- *Register Value* indica il valore del dato, rappresentato in una "little endian representation" (particolare modalità per immagazzinare dati di dimensioni superiori ad 1 byte).

- **Time Tagged TC**

Time Tagged TC Data field					
Address	**Sub-address**	**Type**	**Register value**	**Day**	**Sec**
1 byte	1 byte	1 byte	4 bytes	3 bytes	3 bytes

Table 10: Time Tagged TC Data field

Dove:

- *Address, Sub – Address, Type e Register values* sono analoghi allo Standard TC;
- *Day e Seconds* forniscono un'informazione su tempo a cui è stato attutato il TC. In particolare Day rappresenta i giorni passati da J2000, mentre Sec rappresenta i secondi del giorno attuale.

- **TC acknowledge/rejection message**

Acknowledge TC Data field			
Type	**Address**	**Sub-address**	**Register value**
0x01	1 byte	1 byte	4 byte

Table 11: Acknowledge TC Data field

Rejection TC Data field	
Type	**Error message**
0x10	2 bytes

Table 12: Rejection TC Data field

Il *type field* nel caso di rejection TC rappresenta il tipo di rejection, il rispettivo messaggio di errore è riportato in Table 13: Messaggi di error.13.

Error coding	
0x0000	No error
0x0001	CRC error
0x0002	Invalid address: out of range
0x0004	Invalid address: equipment/payload is OFF
0x0008	Invalid address/type combination
0x0010	Invalid time
0x0020	Time tagged of type GET is invalid (not implemented)
0x0040	Destination equipment timeout/not answering
0x0080	Sequence number error (not implemented)
0x0100	Unable to process TC request

Table 13: Messaggi di errore

- **Beacons**

Beacon of type x Data field
Data field
122 bytes

Table 14: Beacon Data field

Composti da 122 bytes, e vi sono 6 differenti Beacon in ESEO:

1) Beacon 1: general
2) Beacon 2: Power System
3) Beacon 3: OBDH
4) Beacon 4: AOCS e sensori/attuatori
5) Beacon 5: FDIR e TMTC
6) Beacon 6: Payloads

- **HK Data Pages**

HK page TC Data field	
Page number	Page content
1 byte	Description

Table 15: HK Page TC Data field

- *Page number* indica il numero della pagina (da 1 a 28);
- *Page content* è una breve descrizione del contenuto della pagina.

1.5 Ground Segment e Archiving

Il database utilizzato per la missione ESEO è implementato in MySQL. MySQL è un software RDBMS (Relational Database Management System) open source, finalizzato alla gestione di database, che utilizza un linguaggio di tipo SQL (Structured Query Language) per interagire con la banca dati.

Nel caso di ESEO i dati sono organizzati in tabelle alle quali è possibile accedere con una specifica query. Quelle create sono le seguenti:

- TC list
- TM history
- TM parameters
- TC history
- HK pages
- Beacon
- TTC schedule
- TC queue
- Acknowledge message

TC list

Contiene la lista in ordine alfabetico di tutti i possibili telecomandi che possono esser lanciati dal MCS *(Mission Control Centre)*. È possibile, con una specifica query, visualizzare sull'UI *(User Imterface)* tale lista, divisa per sottosistemi e payload, oppure ottenere il nome di un particolare TC inserendo come campo di ricerca il suo TC address.

Field	prefix	address	type	description
Type	char(3)	varbinary(2)	binary(1)	varchar(100)

Table 16: TC list

Il campo *Prefix* indica il sottosistema selezionato, *address* identifica in modo univoco il TC da inviare, mentre *Type* indica il tipo di TC. Vi sono tre tipi di TC selezionabili: SET (0x01), GET (0x10) o entrambi.

TM history

Registra tutti i pacchetti di TM (Telemetry) ricevuti dal satellite e li visualizza nella MCS UI a partire dall'ultimo ricevuto. La tabella può essere filtrata per *start_date, class* o *last result,* e, con una specifica query, è possibile visualizzare il numero totale di pacchetti che sono stati ricevuti.

Field	class	info string	received at
Type	binary(1)	varbinary(128)	timestamp

Table 17: TM history

I campi della tabella sono i seguenti:

- *Class:* indica il tipo di pacchetto ricevuto (vedi Tab.18);
- *Info string:* contiene solo il campo *info* estratto dal frame inviato;
- *Received at:* data e orario di ricezione del frame, in UTC formato YYYY – MM – DD HH:MM:SS.

Class value	Packet definition
0x02	Acknowledge/rejection message
0x03	Beacon of type 1: general
0x04	Beacon of type 2: power system
0x05	Beacon of type 3: OBDH
0x06	Beacon of type 4: AOCS
0x07	Beacon of type 5: sensor/actuator
0x08	Beacon of type 6: payload
0x09	HK data page
0x0A	HK history segment
0x0B	PS high rate history segment
0x0C	Death report
0x0D	ERS Image

Table 18: Definizione dei Class field

TM Parameters

Contiene la definizione di tutti i dati di HK e viene utilizzata nel MRT (*Mission Report Tool)* per selezionare, visualizzare e, se richiesto, scaricare i parametri di telemetria desiderati.

Field	prefix	label	data_type	description	default_value
Type	char(3)	Varchar(50)	enum	Varchar(150)	float
Field	min_value	max_value	scaling_divide	scaling_add	units
Type	float	float	float	float	enum

Table 19: TM Parameters

- *Label:* nome del parametro di HK, visualizzato nella lista del MRT;
- *Datatype:* tipo di dati del parametro (vedi Table 20: Tipo di dati parametri HK);
- *Description:* breve descrizione del parametro;
- *Min_value* e *Max_value:* valori minimi e massimi accettabili per quell parametro.
- *Scaling_divide* e *Scaling_add:* valore da dividere o aggiungere al dato ricevuto per ottenere il valore corretto e calibrato del parametro;
- *Units:* unità di misura.

Data type	Description
U8	Byte unsigned integer
I8	Byte signed integer
U16	Word unsigned integer
I16	Word signed integer
U24	Unsigned 24 bit integer
U32	Long unsigned integer
I32	Long Signed Integer
SGL	Single-precision, floating point
U64	Quad unsigned integer

Table 20: Tipo di dati parametri HK

TC history

Contiene una lista di tutti i pacchetti TC trasmessi, e può essere visualizzata e filtrata per dato o sottosistema nella MCS UI (interfaccia grafica della ground station). Contiene informazioni anche sullo status dei sottosistemi e dei payload.

Field	address	prefix	description	info string	sent at	status
Type	varbinary(2)	char(3)	varchar(100)	varbinary(15)	timestamp	enum

Table 21: TC history

- *Address*: identifica in modo univoco il TC da inviare;
- *Prefix*: indica il sottosistema/payload a cui è riferito il TC;
- *Description*: breve descrizione del TC;
- *Info string*: è il campo che contiene il TC stesso, in notazione esadecimale;
- *Sent at*: indica la data di invio del TC;
- *Status*: fornisce informazioni sullo status del comando (vedi Tab. 22).

Status	Description
ACK	The TC has been successfully received by the S/C
NAK	The TC has not been received by the S/C or the ACK/REJ message has not been received by the G/S
REJ	The TC has been received by the S/C but rejected

Table 22: Status

HK Pages

I dati di telemetria delle classi HK pages, HK history, PS high rate history e Beacon sono immagazzinati insieme in varie tabelle, suddivise in base al sottosistema/payload a cui appartengono e nominate con il prefisso del sottosistema/payload. Il valore di ogni parametro è inserito nella tabella così come viene inviato dal satellite, quando poi viene richiesto al database dalla UI, il dato viene calibrato con specifici coefficienti che vengono moltiplicati o aggiunti.

Generalmente la prima colonna delle tabelle contiene il tempo di ricezione del parametro, la seconda colonna indica se il parametro è di tipo HK history, data page o beacon, mentre le ultime due colonne contengono il *timestamp* (marcatura temporale) e la sorgente del parametro. Per quanto riguarda le HK pages, per alcuni sottosistemi è stato necessario separare i parametri in diverse pagine, a causa del loro numero elevato. In Table 23: HK Pages sono mostrate le pagine inerenti ad ogni sottosistema. L'implementazione di queste pagine comprende una parte di questa tesi, e se ne discuterà in modo più approfondito nel 2 Implementazione delle pagine di OBDH HK e TC su Grafana.

Equipment (table)	Page (view)
obd	page1
	page2
acs	page3
	page4
pmm	page5
	page6
tta	page9
ttb	page10
ssm	page11
ssr	page12
ese	page13
mwr	page14
mwm	page15
mps	page16
	page17
mmm	page18
mmr	page19
mtm	page20
mtr	page21
tri	page22
lmp	page23

ucam	page24
amsat	page25
hstx	page26
gps	page27
ade	page28
scam	page29

Table 23: HK Pages

Beacons

I beacon sono pacchetti di dati contenenti parametri di telemetria del satellite trasmessi automaticamente ad intervalli regolari, che vengono resi disponibili e condivisi anche con i radioamatori.

La struttura dei dati dei beacon è simile a quella delle HK pages e, come già citato nel paragrafo 1.4 Protocolli di telemetria e telecomando, ne esistono 6 classi. Qualora un beacon contenga dati che provengono da più sottosistemi/payload allora viene creata una tabella apposita; se invece quei parametri sono già stati implementati un una tabella precedente, si rende possibile richiamare quel parametro ricercando sia la tabella, che il beacon.

Time – Tagged schedule

Contiene una lista di telecomandi Timetagged (cioè che non vanno eseguiti istantaneamente ma ad un tempo preciso) approvati, inviati dalla GS al satellite. Ogni comando è marcato con il suo tempo di esecuzione. La struttura di questa tabella è descritta di seguito.

Field	number	execution_time	address	value	status
Type	tinyint unsigned	timestamp	binary(2)	binary(4)	enum

Table 24: Time - Tagged schedule

- *Number* identifica la riga e quindi in modo univoco anche il TTC fino alla fine della sua esecuzione;
- *Execution time* identifica il tempo di esecuzione del comando, e se questo sta ancora procedendo;
- *Address* identifica in modo univoco il comando da inviare;
- *Value* indica il valore richiesto per quel parametro, in caso il comando sia di tipo SET;
- *Status* indica lo status del comando, perciò se esso è o non è in esecuzione o se è stato completato;

TC Queue

Quando le operazioni vengono svolte online, i TC vengono salvati in una lista che, una volta tornato online, viene scaricata e salvata nella tabella TC Queue del database. La struttura della tabella è la seguente:

Field	seq	address	type	description	info	status
Type	mediumint	varbinary(2)	binary(2)	varchar(100)	varbinary(15)	enum

Table 25: TC Queue

- *Seq (sequence)* è un numero intero che indica la posizione del comando sulla lista;
- *Address* identifica in modo univoco il parametro modificato con il TC sul satellite;
- *Type* è il tipo di comando (SET, GET o entrambi);
- *Description* è una breve descrizione del TC;
- *Info* contiene il valore richiesto dal TC;
- *Status* è lo status del TC (vedi Table 22: Status);

Acknowledge message

Questa tabella è utilizzata dal software di ESEO per svolgere procedure di volo automatiche. Non è strettamente necessaria per il completamento della missione, ma velocizza l'upload dei TC nel MCS e alleggerisce il carico di lavoro degli operatori. Tramite questa tabella possono essere visualizzati alcuni dati di telemetria e comandi di tipo SET/GET che sono stati attuati in automatico sul satellite. La struttura della tabella è la sguente:

Field	register	address	updated_at
Type	binary(4)	binary(2)	timestamp

Table 26: Acknowledge message

- *Register* contiene il valore attuale del parametro sul satellite (in notazione esadecimale)
- *Address* identifica in modo univoco la variabile;
- *Updated_at* contiene il tempo a cui è stato aggiornata quella variabile.

2 Implementazione delle pagine di OBDH HK e TC su Grafana

2.1 Organizzazione dei dati

Grafana è una piattaforma open – source che consente di visualizzare ed organizzare graficamente dati provenienti da diverse sorgenti. La piattaforma è organizzata in *dashboard*, ovvero in pagine che racchiudono una serie di dati ed informazioni riguardanti un certo ambito, tabella, database, sottosistema ecc. Grafana è stata utilizzata per visualizzare i dati di ESEO contenuti all'interno del database nelle tabelle dei 6 beacon (più uno di emergenza) e delle pagine di OBDH HK (*HouseKeeping*) e TC (*TeleCommand*) . Il grande vantaggio di utilizzare una piattaforma come Grafana per organizzare i dati di un database è quello di renderli "human readable". Questo rende i dati più comprensibili, permette di visualizzarne in modo chiaro l'andamento e l'attività.

Le varie tabelle del database sono organizzate in una serie di dashboard apposite.

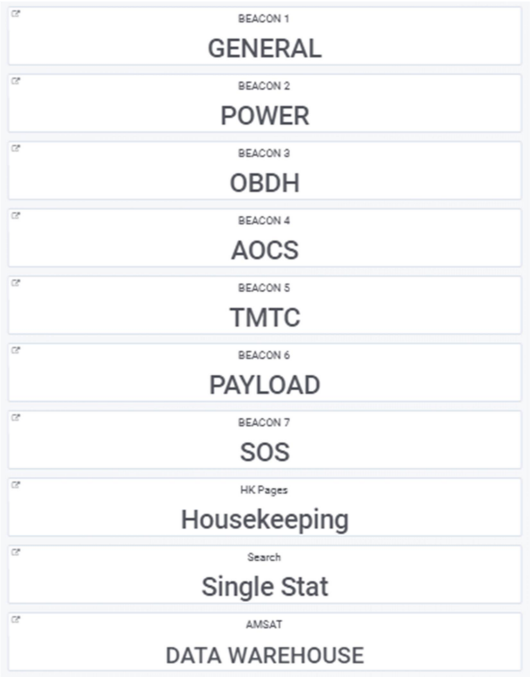

Figure 5: Beacons

Figure 6: HK Pages

La homepage del sito di grafana per ESEO consiste in una visualizzazione dei principali dati di telemetria del satellite.

È presente un conteggio dei giorni passati dall'ultimo reset, l'attuale modalità attiva sulla piattaforma di OBDH, un visualizzatore 2D per la posizione del satellite sulla terra, una serie di grafici e tabelle per monitorare lo status e l'attività dei principali sottosistemi del satellite ed infine un visualizzatore 3D dei dati di telemetria sviluppato su Cesium. Quest ultimo sarà l'argomento principale di questa tesi e verrà trattato in modo approfondito nel capitolo 3 Sviluppo di un visualizzatore 3D di dati orbitali.

Per analizzare singolarmente ogni parametro è stato inserito un drop – down menu nella pagina principale che consente di scegliere la dashboard che si vuole monitorare o modificare.

All'interno, le informazioni contenute nelle dashboard, vengono suddivise in *pannelli* . Grafana mette a disposizione una vasta gamma di pannelli che possono risultare più o meno adatti ad organizzare i dati di un singolo parametro.

2.2 Creazione dei pannelli e visualizzazione dei dati

Tra i vari pannelli, quelli che sono stati maggiormente utilizzati sono *graph*, *singlestat* e *discrete*. Ogni pannello ha un suo editor, che consente di modificarne gli aspetti visivi (colori, scala, dimensioni, titolo ecc….) e realizzare la query in linguaggio SQL necessaria a prelevare il parametro dal database.

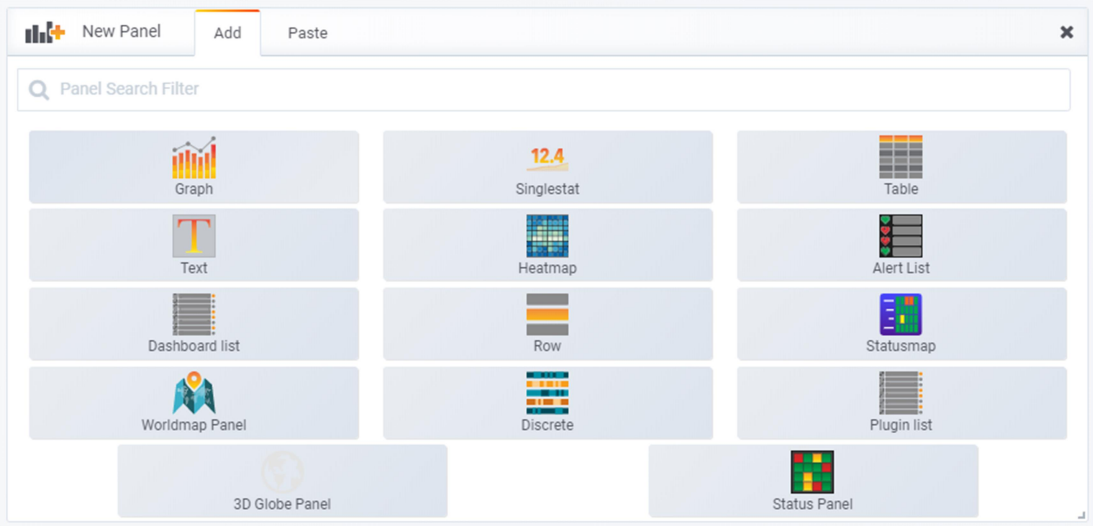

Figure 7: Pannelli

Graph

Il pannello *graph* consente di realizzare un grafico cartesiano dell'andamento di un certo parametro. I vari pacchetti di dati relativi ad esso vengono ordinati in base al tempo e visualizzati sul pannello. Infatti, quando ESEO invia *frames* alla GS, ogni pacchetto di dati contiene anche un'informazione su quando è stato registrato e prelevando quest'informazione dal database i valori possono essere ordinati.

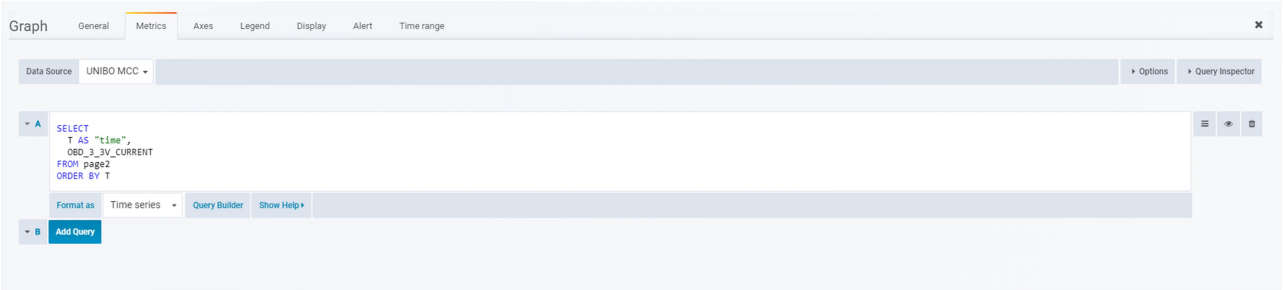

Figure 8: Graph editor

Di seguito sono descritte le varie sezioni dell'editor:

- *General:* titolo e breve descrizione del grafico;
- *Metrics:* è la sezione che consente di scrivere la query per selezionare il giusto parametro dalla tabella del database. In alto come 'datasource' è necessario selezionare il database da cui prelevare i dati (in questo caso si chiama "UNIBO MCC", ed è stato impostato in una sezione apposita del server Grafana). La query si costruisce nel seguente modo: si utilizza un comando *SELECT* (istruzione necessaria per recuperare i dati memorizzati in un database) seguito dall'assegnazione di T come tempo e dal nome del parametro. Si definisce la tabella da cui prelevare lo stesso e in che modo ordinare i vari dati (in base al loro tempo). È possibile aggiungere più query per visualizzare più parametri sullo stesso grafico; per questioni di coerenza i parametri visualizzati nel medesimo grafico dovranno avere la stessa unità di misura;
- *Axes:* per definire l'unità di misura, la scala, il numero di decimali e modificare a piacimento gli assi;
- *Legend:* inserire una legenda al grafico;
- *Display:* modificare la visualizzazione del singolo dato (linee, punti, spessori ecc…);
- *Alert:* è possibile inserire un messaggio di allerta in caso un dato superi un certo valore consentito;
- *Time range:* il range temporale di dati che si vogliono visualizzare.

In Figure 9: Angular rates vi è un esempio di come vengono visualizzati i dati sul grafico.

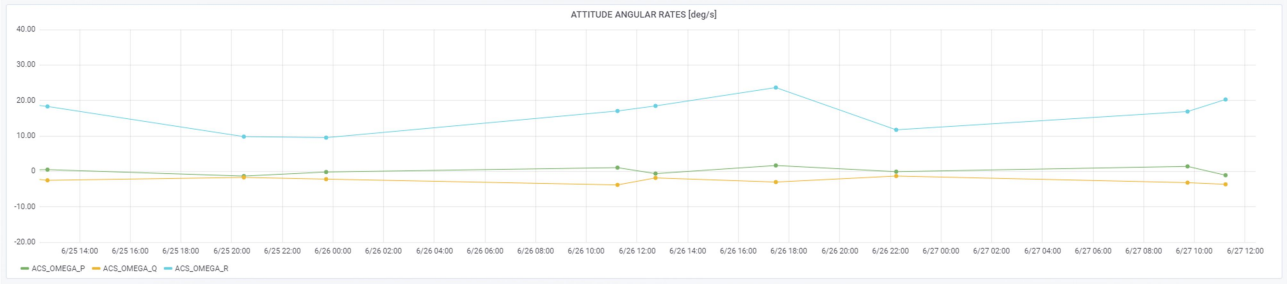

Figure 9: Angular rates

Singlestat

Il pannello *singlestat* è utilizzato principalmente per contatori e parametri numerici, di cui si vuole visualizzare l'ultimo aggiornamento e non il suo andamento. È utilizzato per esempio per conteggiare i giorni passati da J2000, il numero di errori accorsi ad un certo sottosistema ecc…

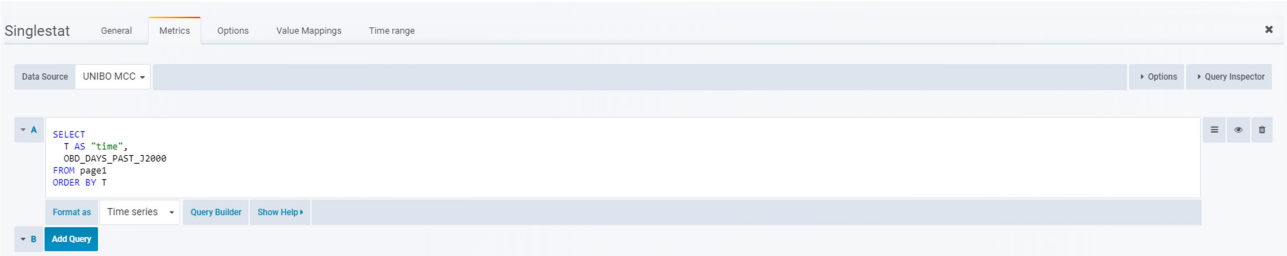

Figure 10: Singlestat editor

Di seguito sono descritte le varie sezioni dell'editor:

- *General:* titolo e breve descrizione del grafico;
- *Metrics:* la query per un *singlestat* viene costruita allo stesso modo del *graph;*
- *Options:* consente di modificare l'unità di misura, il tipo di dato (medio, minimo, massimo, attuale ecc…), decimali, dimensioni e colori;
- *Value mapping:* altra sezione molto importante dell'editor del pannello, largamente utilizzata per il *discrete* e descritta meglio in seguito;
- *Time range:* range temporale di dati che si vogliono visualizzare.

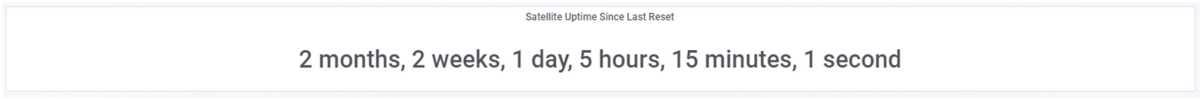

Figure 11: Tempo dall'ultimo reset

Discrete

Il panello *discrete* consente di visualizzare lo stato di parametri che possono assumere solo un insieme limitato di valori, anziché variare nel continuo (ad esempio semafori on/off o flag di errore OK/ERROR o ancora modalità operative di un sottosistema). In questa specifica applicazione *discrete* è stato utilizzato anche per quelle variabili che contengono più parametri al loro interno, e che vanno ulteriormente spacchettate dopo che sono state lette dal database (nel formato dati di ESEO alcune variabili di N bit hanno al loro interno N parametri diversi che vanno letti e plottati separatamente).

Il segnale inviato dal satellite, infatti, non è altro che una sequenza di bit, che viene salvata dal database come serie di 0 e 1. In alcuni segnali, ogni bit, o sequenza di bit, ha un significato ben preciso e può indicare, come già detto in precedenza, un particolare stato di un sottosistema.

Figure 12: Discrete editor

Di seguito sono descritte le varie sezioni dell'editor:

- *General:* titolo e breve descrizione del pannello;
- *Options:* dimensioni delle righe ed altre caratteristiche grafiche;
- *Metrics:* sezione in cui viene scritta la query in linguaggio SQL per recuperare i dati dal database. Nel caso in Fig.NUMERO è stato necessario assegnare una variabile ad ogni bit, perché ognuno rappresentava lo stato acceso/spento di un sottosistema. La riga di codice utilizzata, e ripetuta per ogni bit, è la seguente:

```
CONVERT(SUBSTRING(LPAD(BIN(OBD_EQUIPMENT_STATUS),32,'0'),30,1),UNSIGNED INTEGER) as "PMB"
```

- BIN(decimale) restituisce il valore binario di un numero in base dieci, equivale a CONV(numero,10,2);
- LPAD(stringa, lunghezza, sottostringa) restituisce una stringa dopo averla riempita da sinistra di tante sottostringhe fino a raggiungere la lunghezza richiesta;
- SUBSTRING(stringa, posizione) restituisce una stringa estratta da un'altra stringa più lunga, a partire da una determinata posizione;
- CONVERT converte i valori da un tipo di dati ad un altro;

In certi casi è stato necessario estrarre, da un'unica sequenza di bit, più sequenze più piccole, ognuna rappresentante un numero, in codice binario, che può poi essere associato ad una determinata condizione nella sezione *mappings*;

- *Legend:* legenda della tabella;
- *Colors:* è possibile assegnare ad ogni valore del *mapping* un certo colore in modo da rendere più immediata la comprensione dello stato del parametro;
- *Mappings:* questa sezione è molto utilizzata nel pannello *discrete*, ma può essere implementata anche nei *singlestat*. Una volta estratti i valori di ogni bit, o sequenza di bit, si può assegnare ad ogni valore un determinato messaggio visualizzato. Ad esempio se si vuole visualizzare lo stato acceso/spento di un sistema, si assegna al valore 0 → OFF e al valore 1→ON. In questo modo se il bit riguardante quel determinato sottosistema è pari a 0, nel *discrete* sarà visualizzato OFF, viceversa con 1. Allo stesso modo se abbiamo più valori, dati da una sequenza di bit, possiamo assegnare a ciascuno un certo messaggio, come si fa ad esempio per indicare la modalità attiva sull'OBDH. I valori estratti da mappare sono descritti nei documenti di sistema di ESEO;
- *Time range:* range temporale di dati che si vogliono visualizzare.

Il *discrete* prende tutti i dati inviati nell'arco temporale selezionato e ne fa una "storia". Ciò permette di visualizzare i periodi di accensione/spegnimento di un sottosistema o quando questo ha dato un errore o ancora quale modalità è stata attivata e per quanto tempo. Ad esempio, nelle figure sottostanti, è possibile vedere lo stato di alcune flag di attivazione e soprattutto i passaggi da uno stato all'altro, fornendo un'informazione immediata all'utente.

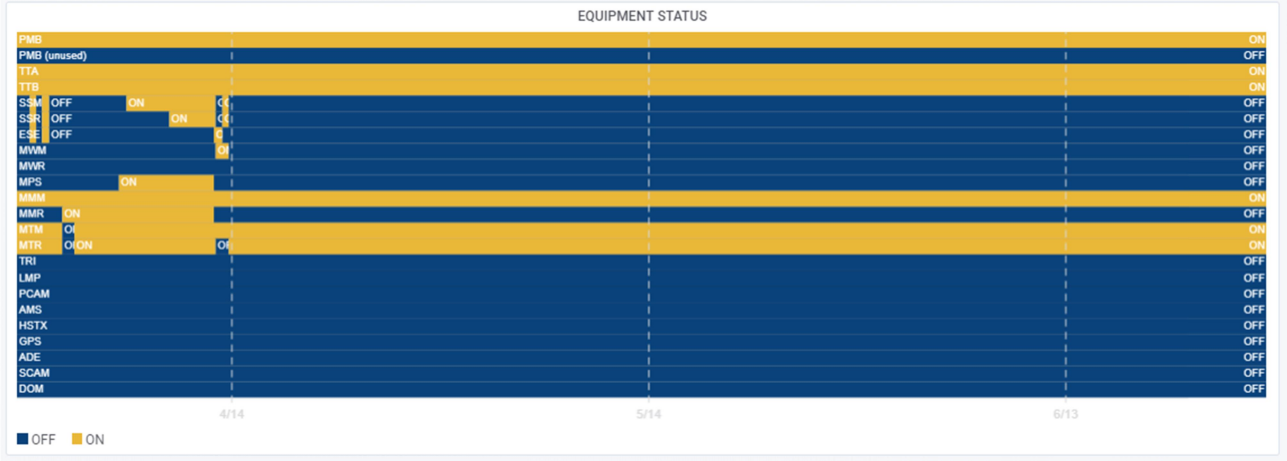

Figure 13: Equipment status

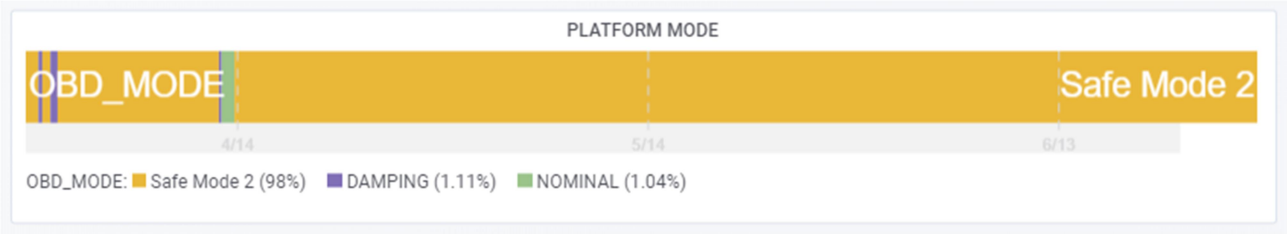

Figure 14: OBDH Platform mode

Il processo di creazione dei pannelli, scrittura delle query e organizzazione dei dati del database è stato svolto per tutti i parametri contenuti nelle pagine di OBDH HK e TC, le quali sono state poi messe online sulla piattaforma Grafana di ESEO.

3 Sviluppo di un visualizzatore 3D di dati orbitali

3.1 Node JS

3.1.1 Introduzione a Node Js

Node JS è un framework che consente di scrivere applicazioni in JavaScript lato server.

Node.js ha una libreria di diversi moduli JavaScript, che possono essere scaricati attraverso una semplice funzione e messi a disposizione come componenti già pronti all'interno di un'applicazione. Un esempio è il modulo SGP4, che è stato largamente utilizzato per questa tesi. Inoltre è possibile installare moduli aggiuntivi con il sistema di gestione dei pacchetti NPM (Node Package Manager), utilizzando il semplice comando: `npm install -nome del modulo-.`

Il vantaggio nell'utilizzare questa tecnologia deriva dal fatto che Node utilizza un **modello di I/O (Input/Output) asincrono basato sugli eventi**, modello particolarmente adatto ad essere programmato in JavaScript e per le applicazioni web. Tra le attività di I/O (Input/Output) caratteristiche di un server ci sono, ad esempio, eseguire o rispondere a delle richieste su dei dati, interagire con un database, instaurare collegamenti tra più componenti all'interno di un programma ecc… Nei linguaggi di programmazione, come C o Java, le operazioni I/O vengono eseguite in modo sincrono, portando a termine un compito dopo l'altro. Pertanto il sistema di I/O si blocca fino a che il compito attuale non viene portato a termine. Node.js utilizza invece un I/O asincrono, nel quale vengono delegate le operazioni di scrittura e di lettura al sistema operativo o al database. Questo permette di eseguire un grande numero di operazioni I/O parallelamente, senza che il programma si blocchi ogni volta che deve completare un'operazione. Ciò rende le applicazioni basate su Node.js e JavaScript un grande vantaggio in termini di velocità.

Questo ambiente è stato quindi scelto per lo sviluppo del visualizzatore di dati orbitali, in cui è stato necessario costruire un server che interroghi costantemente il database MySQL per ottenere il valore reale degli angoli d'assetto del satellite.

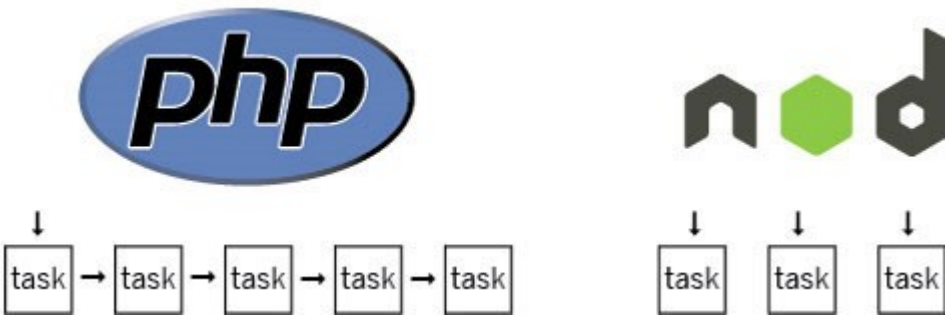

Figure 15: Funzionamento di NodeJS

3.1.2 Creazione del server

Per lo sviluppo del progetto di tesi è stato necessario progettare un piccolo server su cui far girare il programma in Cesium (vedi 3.2 Cesium). Utilizzando come struttura logica di supporto NodeJS, sono stati installati i seguenti moduli:

- express
- compression
- fs
- url
- request promise
- mysql

Principalmente i moduli più utilizzati all'interno dello script sono stati *express* (che consente di generare il server), *mysql* (che consente di utilizzare un database MySQL tramite Node), *request promise* (consente di fare una richiesta al database, per i quaternioni) e *url* (consente di caricare una pagina web ed estrarne l'html).

Per instaurare la connessione tra il server e il database si utilizza una funzione del modulo *mysql* chiamata *createConnection*, e il codice è il seguente:

```
2   var con = mysql.createConnection({
3   host     : "Indirizzo IP del database",
4   user     : "Username",
5   password : "Password per accedere al database",
6   database : "Nome del database",
7   port     : numero della porta di accesso al database
8   });
```

3.1.3 Script di richiesta dei quaternioni e TLE

All'interno del server, sono state implementate due funzioni, necessarie a scaricare i dati aggiornati per i quaternioni dal database e i TLE da CelesTrack.

Per i primi è stato necessario stabilire una connessione tra il server e il database, descritta nel paragrafo precedente. Dopodiché con una query in linguaggio SQL vengono scaricati i valori degli ultimi 5 dati ricevuti. Ad ogni passaggio, infatti, il satellite invia alla GS il suo assetto e questo viene salvato nel database MySQL. Tutti i dati del database possono essere richiamati con una query. In questo caso la richiesta è la seguente:

```
'SELECT T as time,ACS_ATTITUDE_Q1 as q1,ACS_ATTITUDE_Q2 as q2,ACS_ATTITUDE_Q3 as q3,ACS_ATTITUDE_
                    Q4 as q4 FROM beacon4 ORDER BY T DESC LIMIT 5'
```

Infine, con la funzione *console_quat* inserita di seguito, i quaternioni vengono salvati in una variabile *out* e associati ciascuno al proprio tempo, in modo da consentire la visualizzazione nel tempo degli angoli d'assetto. La variabile *out* poi viene inviata con un'architettura REST ad uno script a parte tramite il metodo *get*.

Con il termine REST (*REpresentational State Transfer*) si intende un approccio architetturale per la creazione di Web API (*Application Programming Interface*) basate sul protocollo http (*HyperText Transfer Protocol*) con le seguenti caratteristiche: i dati devono essere accessibili da un URL, avere un formato JSON o XML, essere "stateless" come http ed impiegarne i principali metodi (post, get, put, delete…). In particolare il metodo *get* è quello utilizzato in questo caso

e serve per recuperare una risorsa o un dato. Il protocollo http è il principale sistema utilizzato per la trasmissione di dati sul web; nel nostro caso viene utilizzato per inviare i dati relativi ai quaternioni e TLE dal server allo script descritto nel paragrafo 3.2.4 Caricamento TLE e quaternioni aggiornati.

Questo script contiene la funzione *getQUAT* che riamane "in ascolto", riceve i dati dal server e potrà essere poi richiamata a sua volta nello script di Cesium (vedi 3.2.4 Caricamento TLE e quaternioni aggiornati).

Notiamo nella sezione di codice sottostante che il metodo *get* assegna un certo url (`"/requesting"`) al pacchetto di dati, e questo sarà utilizzato per fornire le informazioni a Cesium.

```
function console_quat(res) {
    estrai(function (err,quat){
        console.log('Numero dati: ',quat.length,'\n')
        out = JSON.stringify({quaternioni: [{q1: quat[4].q1,q2 : quat[4].q2,q3 : quat[4].q3,q4 : quat[4].q4,time: quat[4].time},{q1: quat[3].q1,q2 : quat[3].q2,q3 : quat[3].q3,q4 : quat[3].q4,time: quat[3].time},{q1: quat[2].q1,q2 : quat[2].q2,q3 : quat[2].q3,q4 : quat[2].q4,time: quat[2].time},{q1: quat[1].q1,q2 : quat[1].q2,q3 : quat[1].q3,q4 : quat[1].q4,time: quat[1].time},{q1: quat[0].q1,q2 : quat[0].q2,q3 : quat[0].q3,q4 : quat[0].q4,time: quat[0].time}]})
        res.json(out);

    });
}

app.get("/requesting", (req, res, next) => {
    console_quat(res);
});
```

Nel visualizzatore sviluppato al capitolo 3.2 Cesium, l'assetto del satellite sarà quello ricevuto all'ultimo passaggio, ma è possibile modificare il tempo della simulazione per vedere come si sono evoluti gli angoli di assetto ad ogni periodo, da 5 passaggi fa, fino ad ora.

Per quanto riguarda i TLE invece, il procedimento per scaricarli è differente, in quanto essi vengono prelevati dall'html della pagina di CelesTrack che li genera. Si assegna come costante l'indirizzo web della pagina di CelesTrack che contiene le due righe di TLE e successivamente si utilizza la funzione *rp* del modulo *request promise* per ottenere il testo html contenuto nella pagina. Dopodichè, utilizzando la funzione *indexOf* si crea una variabile *start* che indica la posizione dei caratteri di inizio linea. Partendo da *start,* con un ciclo for si scorrono tutti i caratteri della linea di TLE (69) e si salvano nella variabile *tle,* rispettivamente nella proprietà line1 e line2.

Infine con lo stesso procedimento descritto per i quaternioni, il dato viene inviato ad uno script tramite il metodo *get* e da lì caricato su Cesium.

Di seguito il codice per estrarre i TLE da CelesTrack.

```
const indirizzo = 'https://celestrak.com/satcat/tle.php?CATNR=43792';

function estr_tle(res) {

    rp(indirizzo)

    .then(function(html){

        var tle = {
            line1: '',
            line2: ''
```

```
        };

        var start = html.indexOf('1 43792U');
        for (var i = start; i < start + 69; i++) {
            tle.line1 = tle.line1 + html[i];
        }
        start = html.indexOf('2 43792 ');
        for (var i = start; i < start + 69; i++) {
            tle.line2 = tle.line2 + html[i];
        }
        res.json(JSON.stringify({tle: {line1 : tle.line1,line2 : tle.line2}}));
  })
}

app.get("/tle", (req,res,next) => {
    estr_tle(res);
});
```

3.2 Cesium

3.2.1 Libreria Cesium JS

Cesium è una libreria JavaScript di tipo open-source ampiamente utilizzata per la creazione di scenari geospaziali 3D attorno al globo. Le funzionalità di questa libreria sono svariate: è possibile creare mappe sulla superficie terrestre, importare modelli di oggetti o composizioni 3D (Figure 16**: Mappa Cesium**), simulare lo spostamento di tali entità secondo determinate leggi ecc.

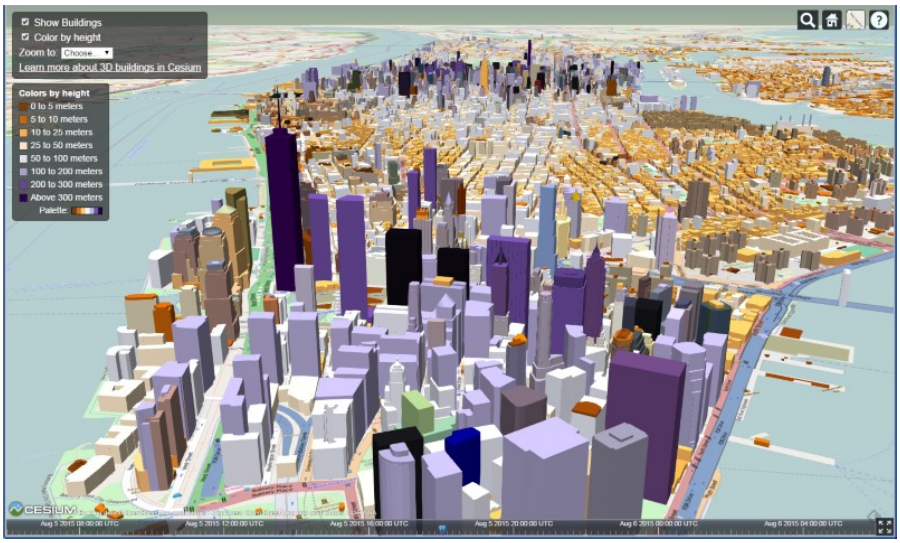

Figure 16: Mappa Cesium

Cesium JS è utilizzato anche per la visualizzazione di dati orbitali e simulazione di missioni spaziali, orbite e oggetti in movimento attorno alla terra. Ciò che è stato fatto in questa tesi è stato appunto sfruttare questa libreria per creare un modello 3D dell'orbita di ESEO che si aggiorni in diretta con i dati inviati dal satellite al database MySQL per quanto riguarda l'assetto e i TLE.

Tra gli strumenti forniti da Cesium ve ne sono due molto utili: Cesium ION e Cesium SandCastle (Figure 17: Cesium Sandcastle). Il primo non è altro che una piattaforma virtuale sulla quale è possibile caricare i modelli 3D che verranno poi utilizzati nel programma JS. Inoltre consente di avere accesso a modelli di default di oggetti, mappe e globi. Il secondo invece è un editor per programmi JavaScript che consente di vedere in tempo reale gli sviluppi del programma e ed eventuali errori.

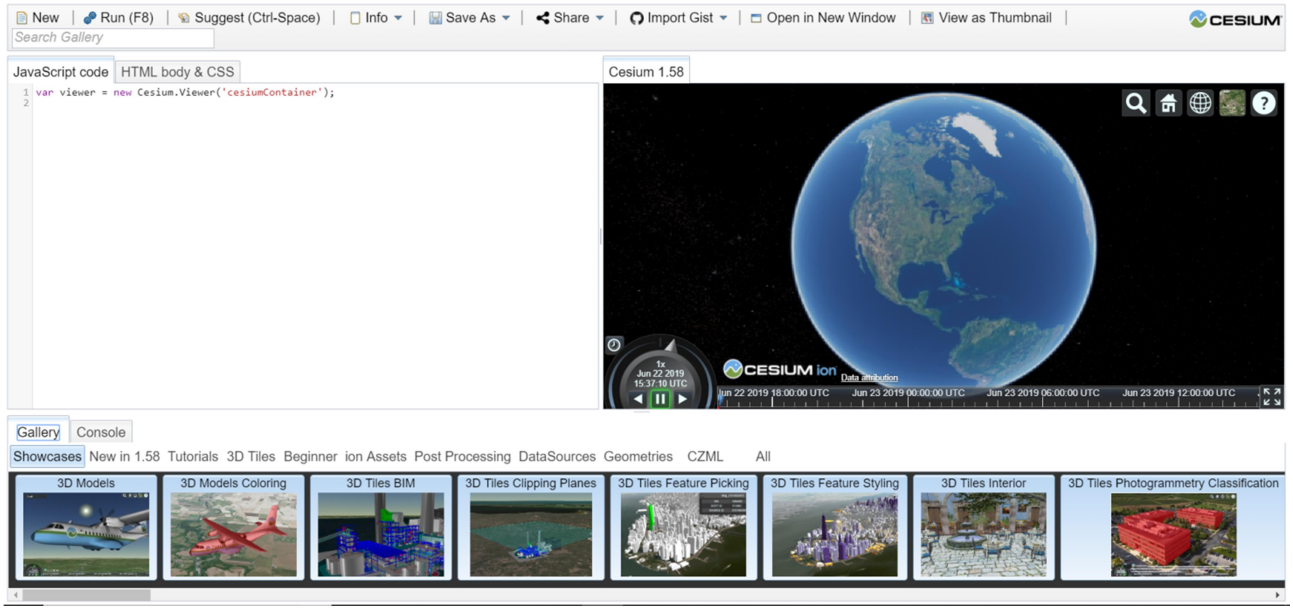

Figure 17: Cesium Sandcastle

3.2.2 Implementazione di SGP4

La prima parte del lavoro consiste nell'implementazione di un programma che calcoli l'orbita del satellite e i suoi parametri orbitali secondo il modello di perturbazione SGP4.

In questo caso il programma è stato scaricato come modulo di Node.Js al link in bibliografia (Download modulo SGP4) e successivamente è stato modificato e adattato per il nostro scopo. Le principali modifiche riguardano il tempo, che viene sincronizzato con quello di Cesium, descritto nel paragrafo successivo 3.2.3 Settaggio orologio del visualizzatore.

SPG4 calcola contemporaneamente orientamento, posizione e velocità del satellite per un dato tempo a partire dalle due linee di TLE. Dopodiché fornisce le componenti di tali vettori in più sistemi di riferimento.

Calcolo delle componenti del vettore orientamento:

```
1.              //  -------------------- orientation vectors -------------------
2.               sinsu =  Math.sin(su);
3.               cossu =  Math.cos(su);
4.               snod  =  Math.sin(xnode);
5.               cnod  =  Math.cos(xnode);
6.               sini  =  Math.sin(xinc);
7.               cosi  =  Math.cos(xinc);
8.               xmx   = -snod * cosi;
9.               xmy   =  cnod * cosi;
10.              ux    =  xmx * sinsu + cnod * cossu;
11.              uy    =  xmy * sinsu + snod * cossu;
12.              uz    =  sini * sinsu;
13.              vx    =  xmx * cossu - cnod * sinsu;
14.              vy    =  xmy * cossu - snod * sinsu;
15.              vz    =  sini * cossu;
```

Calcolo delle componenti di posizione e velocità:

```
            // --------- position and velocity (in km and km/sec) ----------
              var _mr = mrt * radiusearthkm;
              r = {x: 0.0, y: 0.0, z: 0.0};
              r["x"] = _mr * ux;
              r["y"] = _mr * uy;
              r["z"] = _mr * uz;
              v = {x: 0.0, y: 0.0, z: 0.0};
              v["x"] = (mvt * ux + rvdot * vx) * vkmpersec;
              v["y"] = (mvt * uy + rvdot * vy) * vkmpersec;
              v["z"] = (mvt * uz + rvdot * vz) * vkmpersec;
          }

          if (mrt < 1.0) {
              satrec.error_message = 'mrt ' + mrt + ' is less than 1.0 indicating the satellite has dec
    ayed';
              satrec.error = 6;
              return [false, false];
          }

          return {
              position: r,
              velocity: v
          };
```

Lo script riporta un errore anche nel caso in cui la variabile *mrt* (una sorta di posizione normalizzata con il raggio terrestre) sia minore di 1, il che sta a significare che il satellite è probabilmente rientrato.

Successivamente lo script calcola le coordinate del satellite, a partire dalla sua posizione, in diversi sistemi di riferimento. Sotto viene inserita la funzione per i due sistemi utilizzati: cartesiano e geodetico (latitudine e longitudine di un punto sulla superficie terrestre determinate dalla verticale geodetica, che è ortogonale allo sferoide specificato).

Vengono restituite latitudine, longitudine e altezza, poi assegnate come proprietà alla variabile *geodetic_coords*, e tre coordinate cartesiane X, Y, Z (*eci_coords* sono le coordinate in un sistema Earth Centered Inertial).

```
    eciToGeodetic: function(eci_coords, gmst) {
          'use strict';
          var a    = 6904.82699;
          var b    = 5830.0623;
          var R    = Math.sqrt( (eci_coords["x"]*eci_coords["x"]) + (eci_coords["y"]*eci_coords["y"]) );

          var longitude = Math.atan2(eci_coords["y"], eci_coords["x"]) - gmst;
          var kmax = 20;
          var k = 0;
          var latitude = Math.atan2(eci_coords["z"],
                          Math.sqrt(eci_coords["x"]*eci_coords["x"] +
                                    eci_coords["y"]*eci_coords["y"]));

    (. . .)

var height = (R/Math.cos(latitude)) - (a*C);

          var velocity = Math.sqrt(398600.8 / (height + 6378.135)); // Velocity in kilemeters per secon
    d. Multiply by 3600 for kilometers per hour.

          return { longitude : longitude, latitude : latitude, height : height, velocity : velocity };

      },

    (. . .)
```

```
23.
24.  eciToEcf: function(eci_coords, gmst){
25.          'use strict';
26.          var X = (eci_coords["x"] * Math.cos(gmst))    + (eci_coords["y"] * Math.sin(gmst));
27.          var Y = (eci_coords["x"] * (-Math.sin(gmst))) + (eci_coords["y"] * Math.cos(gmst));
28.          var Z =  eci_coords["z"];
29.          return { x : X, y : Y, z : Z };
30.      },
```

Per approfondire il calcolo di questo tipo di coordinate link in bibliografia (Kelso, Orbital Coordinate Systems, Part I), (Kelso, Orbital Coordinate Systems, Part II), (Kelso, Orbital Coordinate Systems, Part III).

Per ultima figura la sezione di codice in cui vengono calcolati gli angoli del *Sistema di riferimento orizzontale,* altresì detto "*topocentrico*" (che sarà necessaria poi per la visualizzazione degli accessi di visibilità del satellite sulla ground station). Quest'ultimo è un sistema di coordinate celesti che utilizza il piano dell'orizzonte dell'osservatore come piano fondamentale. Le sue coordinate sono espresse in termini di **elevazione** (spesso chiamata **altezza**) e **azimut**.

Viene diviso il cielo in due emisferi: quello superiore, nel quale gli oggetti sono visibili, e quello inferiore, nel quale gli oggetti non possono essere visti.

- l'elevazione è l'angolo tra l'oggetto e l'orizzonte locale dell'osservatore. Se l'oggetto è nell'emisfero superiore, quest'angolo è compreso tra 0 e 90°;
- l'azimut rappresenta l'angolo tra la proiezione dell'oggetto sull'orizzonte locale e la direzione del nord, misurato da nord verso est.

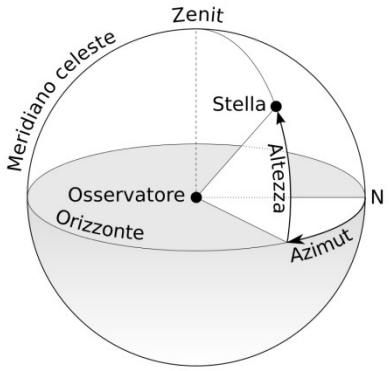

Figure 18: Sistema di riferimento topocentrico

Di seguito lo script:

```
1.      topocentricToLookAngles: function(topocentric){
2.          'use strict';
3.          var top_s = topocentric["top_s"];
4.          var top_e = topocentric["top_e"];
5.          var top_z = topocentric["top_z"];
6.          var range_sat    = Math.sqrt((top_s*top_s) + (top_e*top_e) + (top_z*top_z));
7.          var El      = Math.asin (top_z/range_sat);
8.          var Az      = Math.atan2 (-top_e, top_s) + Math.PI;
9.          return { azimuth : Az, elevation : El, range_sat : range_sat };
10.     },
```

Come posizione dell'osservatore è stata in precedenza fissata la posizione della Ground Station di Forlì, le cui coordinate sono, in gradi decimali:

Latitudine	Longitudine
44.20057	12.0671

Table 27: Coordinate osservatore

3.2.3 Settaggio orologio del visualizzatore

SGP4 utilizza come variabile tempo un numero intero dato dal numero di giorni a partire dal 1° gennaio 1970 ad oggi.

```
            gsto = SGP4.gstime(epoch + 2433281.5);
```

Per poter sincronizzare il tempo misurato da SGP4, che poi determina la successione dei dati di posizione, velocità ed orientamento del satellite, con quello di Cesium, è stato necessario settare alcune proprietà dell'orologio del visualizzatore.

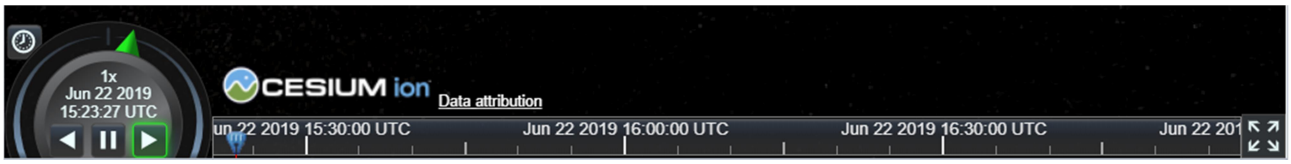

Figure 19: Orologio Cesium

La variabile *Viewer* contiene tutto il visualizzatore e viene richiamata inizialmente per importare il modello 3D del globo terrestre dai database di default di Cesium, utilizzando la seguente riga di codice:

```
var viewer = new Cesium.Viewer ('cesiumContainer');
```

Il "*clock*" è l'orologio interno del programma. Inizialmente vengono create le due variabili *start* e *stop* per l'inizio e la fine della simulazione. Entrambe vengono inizializzate come data giuliana all'istante "now", ovvero quello attuale.

```
1.  var start = new Cesium.JulianDate.now();
2.  var stop = new Cesium.JulianDate.now();
```

Il **giorno giuliano** (Julian Day, JD) è il numero di giorni passati dal mezzogiorno del lunedì 1º gennaio 4713 a.C. Questo tipo di rappresentazione è utilizzata da astronomi ed ingegneri, in quanto consente di esprimere una qualsiasi data come un numero intero e monotono cresecnte. La **data giuliana** (*JulianDate*), invece, è il giorno giuliano combinato con la frazione di giorno che sta trascorrendo, rapportato alle 24h. Se si vuole, ad esempio, esprimere un anno solare (365d e 4h circa) in data giuliana, si scriverà 365,25.

Una volta calcolata la data giuliana, la variabile *stop* viene aggiornata (riga 10) con la funzione *addSeconds* aggiungendo a *start* il tempo necessario a compiere un'orbita intera (circa 1h e 40 min).

La variabile *tempo* contiene il **Periodo orbitale del satellite** e viene calcolato con la seguente formula:

$$T = 2\pi r \sqrt{\frac{r}{\mu}}$$

dove $\boldsymbol{r}$ è il modulo del vettore posizione del satellite sulla sua orbita, approssimabile alla sua distanza dal centro della terra, e quindi calcolato come: "`geodeticCoordinates.height`" (altezza del satellite sulla superficie) + raggio terrestre (6378 km circa). Tale formula è utilizzabile al posto di quella con il semiasse maggiore, grazie alla bassissima

eccentricità dell'orbita, che può essere quindi considerata circolare. La **Costante Gravitazionale Planetaria** per la terra è $\mu = GM = 398600.8$ [m^3/sec^2], con:

G = 6,67*10−11 [m³ /kg*sec²] (**Costante di Gravitazione Universale**);

M = 5,972 × 10^24 [kg] (**Massa della terra**).

```
1.  //Based on one orbit
2.  var julianday = Cesium.JulianDate.totalDays(start);
3.
4.  (. . .)
5.
6.  var gmst = SGP4.gstimeFromDate(julianday);
7.
8.
9.  var  tempo = ((2 * Math.PI) * (geodeticCoordinates.height + 6378.135)) * (Math.sqrt((geodeticCoordina
    tes.height + 6378.135)/398600.8));
10. stop = Cesium.JulianDate.addSeconds(start, tempo, new Cesium.JulianDate());
11.
12.
13. //Make sure viewer is at the desired time.
14. viewer.clock.startTime = start.clone();
15. viewer.clock.stopTime = stop.clone();
16. viewer.clock.currentTime = start.clone();
17. viewer.clock.clockRange = Cesium.ClockRange.LOOP_STOP; //Loop at the end
18. viewer.clock.multiplier = 1;
19.
```

Viene quindi settato (righe 14-18) il tempo di inizio della simulazione al tempo *start* e il tempo di fine al tempo *stop*.

CurrentTime posiziona l'orologio all'ora attuale, mentre *clockRange* definisce come il programma si deve comportare quando si arriva a fine simulazione: in questo caso la simulazione ricomincia spostando l'inizio al *currentTime* (in pratica come se riaggiornassimo la pagina). Il *multiplier* serve per stabilire la velocità dello scorrere del tempo: in questo caso è settato a 1, quindi ad 1 secondo di simulazione corrisponde 1 secondo reale.

Nel secondo pannello che è stato sviluppato viene aggiunta una riga in cui la proprietà *shouldAnimate* del *clock* viene settata a `false`, il che significa che la simulazione partirà a tempo fermo.

```
viewer.clock.shouldAnimate = false;
```

3.2.4 Caricamento TLE e quaternioni aggiornati

Una delle parti fondamentali dello script è il caricamento dei Two Line Elements e degli angoli di assetto aggiornati all'ultimo passaggio. È già stato descritto nel paragrafo 3.1.3 Script di richiesta dei quaternioni e TLEcome questi dati vengano estratti direttamente dal database MySQL e da CelesTrack, ed inviati tramite un'architettura REST allo script che verrà descritto a breve. Successivamente sarà necessario caricare questi dati su Cesium e assegnarli ad un'entità (cioè il tipo di variabile che in Cesium descrive un qualsiasi elemento da visualizzare, ESEO in questo caso).

Prima di tutto vengono create due funzioni, chiamate *getTLE* e *getQUAT* le quali servono per eseguire la richiesta al server tramite il metodo *get.* È necessario che la funzione prenda in input l'URL generato dal server per quel pacchetto di dati che si sta scambiando. Inoltre viene convertito il pacchetto di dati in formato XML per renderlo leggibile da Cesium.

Sotto l'esempio dello script per la getTLE:

```
var getTLE = function(url, successHandler, errorHandler) {
    var xhr = typeof XMLHttpRequest != 'undefined'
        ? new XMLHttpRequest()
        : new ActiveXObject('Microsoft.XMLHTTP');
    xhr.open('get', url, true);
    xhr.responseType = 'json';
    xhr.onreadystatechange = function() {
        var status;
        var data;
        // http://localhost:3000/requesting
        if (xhr.readyState == 4) { // `DONE`
            status = xhr.status;
            if (status == 200) {
                successHandler && successHandler(xhr.response);
            } else {
                errorHandler && errorHandler(status);
            }
        }
    };
    xhr.send();
};
```

Fatto ciò, si richiamano le due funzioni su Cesium in questo modo:

```
getTLE('http://localhost:3000/tle', function(data) {
    risposta=JSON.parse(data);
    eseoLine1 = risposta.tle.line1;
    eseoLine2 = risposta.tle.line2;
getQUAT('http://localhost:3000/requesting', function(data) {
        quat=JSON.parse(data);
        ori = new Cesium.Quaternion(quat.quaternioni[4].q1,quat.quaternioni[4].q2,quat.quaternioni[4]
.q3,quat.quaternioni[4].q4);
```

I due differenti URL per i TLE e i quaternioni sono quelli a cui il server è programmato per rispondere.

I vari dati vengono salvati su tre variabili chiamate *eseoLine1, eseoLine2* e *quat*. Dopodichè si crea un nuovo set di quaternioni, *ori*, con l'apposita funzione di Cesium.

Infine, in caso dovesse succedere che, per qualche errore, non si riesca a scaricare i dati aggiornati su quaternioni e TLE, è stata implementata una funzione che utilizza dei dati precedenti. In questo modo l'applicazione continuerà a

funzionare senza dare errore, sebbene con dati non del tutto aggiornati. Per quanto riguarda i quaternioni, nel caso in cui il programma non riuscisse a scaricare gli ultimi valori inviati dal satellite, in automatico considera gli ultimi ricevuti (la variabile *ori* semplicemente non si aggiorna).

Per i TLE invece è stato inserito un set di due righe che può essere utilizzato in sostituzione di quello nuovo.

```
}, function(status) {
    eseoLine1 = '1 43792U 18099AL  19147.18575516 -.00000298  00000-0 -21649-4 0  9999';
    eseoLine2 = '2 43792  97.7444 219.5245 0014389  49.7225 310.5251 14.95498193 25821';
    createScenario();
});
```

3.2.5 Creazione dello scenario e delle entità

Il Viewer di Cesium è un pacchetto di widget che consentono di aggiungere svariate funzionalità allo scenario che si sta creando. È stato necessario per lo sviluppo di questo progetto, settare adeguatamente il visualizzatore e utilizzare uno dei vari modelli del globo 3D forniti di default.

```
var viewer = new Cesium.Viewer ('cesiumContainer', {
   scene3DOnly: false,
   selectionIndication: true,
   baseLayerPicker: true,
   shouldAnimate: true,
   shadows: true
 });

viewer.scene.globe.enableLighting = true;
```

Innanzitutto è stato creato un nuovo scenario, il Viewer, e si è importato il globo, che in questo caso si chiama `'cesiumContainer'`. Successivamente vengono impostate le seguenti proprietà:

- `scene3DOnly`: quando è `true`, ogni geometria dello scenario viene renderizzata solamente in 3D, aumentando notevolmente la memoria occupata dal programma. Per questo motivo viene impostato su `false`;
- `selectionIndication`: attiva il widget di selezione. Quando si fa click su un'entità, quest'ultima viene selezionata;
- `baseLayerPicker`: se si fa click su un'entità selezionata, il programma fa lo zoom direttamente su di essa;
- `shouldAnimate`: il tempo scorre automaticamente quando si ricarica la pagina. Se fosse `false`, la simulazione partirebbe a tempo fermo;
- `shadows`: attiva l'ombreggiatura sugli oggetti;

`EnableLighting` è una proprietà del globo terrestre e, se `true`, visualizza il sole e tiene conto della sua illuminazione.

Fatto ciò sono state create le entità all'interno dello scenario. Le **entità** di Cesium sono delle strutture di dati che contengono informazioni sulla loro visualizzazione e le loro proprietà. Le entità base di Cesium sono principalmente strutture geometriche come punti, linee, polilinee, poliedri, sfere ecc.

Le entità che è stato necessario creare per questa tesi sono le seguenti: le due Ground Station (Tartu e Forlì), il satellite ESEO, la traccia a terra e le due linee degli accessi alle due GS. Tutte quante sono state racchiuse all'interno di una collezione di entità denominata 'collection'.

Ogni entità è caratterizzata da una vasta serie di proprietà, riportate nella tabella Table 28: Entità.

Name	Type	Description
options	Object	optional Object with the following properties:

Name	Type	Description
id	String	optional A unique identifier for this object. If none is provided, a GUID is generated.
name	String	optional A human readable name to display to users. It does not have to be unique.
availability	TimeIntervalCollection	optional The availability, if any, associated with this object.
show	Boolean	optional A boolean value indicating if the entity and its children are displayed.
description	Property	optional A string Property specifying an HTML description for this entity.
position	PositionProperty	optional A Property specifying the entity position.
orientation	Property	optional A Property specifying the entity orientation.
viewFrom	Property	optional A suggested initial offset for viewing this object.
parent	Entity	optional A parent entity to associate with this entity.
billboard	BillboardGraphics	optional A billboard to associate with this entity.
box	BoxGraphics	optional A box to associate with this entity.
corridor	CorridorGraphics	optional A corridor to associate with this entity.
cylinder	CylinderGraphics	optional A cylinder to associate with this entity.
ellipse	EllipseGraphics	optional A ellipse to associate with this entity.
ellipsoid	EllipsoidGraphics	optional A ellipsoid to associate with this entity.
label	LabelGraphics	optional A options.label to associate with this entity.
model	ModelGraphics	optional A model to associate with this entity.
path	PathGraphics	optional A path to associate with this entity.
plane	PlaneGraphics	optional A plane to associate with this entity.
point	PointGraphics	optional A point to associate with this entity.
polygon	PolygonGraphics	optional A polygon to associate with this entity.
polyline	PolylineGraphics	optional A polyline to associate with this entity.
properties	PropertyBag	optional Arbitrary properties to associate with this entity.
polylineVolume	PolylineVolumeGraphics	optional A polylineVolume to associate with this entity.
rectangle	RectangleGraphics	optional A rectangle to associate with this entity.
wall	WallGraphics	optional A wall to associate with this entity.

Table 28: Entità (API Reference)

Di seguito è riportato il codice di creazione dell'entità ESEO.

```
1.  entity = viewer.entities.add({
2.
3.      id: 'satellite',
4.
5.      availability : new Cesium.TimeIntervalCollection([new Cesium.TimeInterval({
6.          start : start,
7.          stop : stop
8.      })]),
9.      name: 'ESEO',
10.
11.     position : posizione,
12.
13.     orientation : ori,
14.
15.     model : {
16.         uri : '../Apps/SampleData/models/ESEO/ESEObis.glb',
17.         minimumPixelSize : 300,
18.         maximumScale : 2000000,
19.         shadows: true
20.     },
21.
22.     path : {
```

```
        resolution : 1,
        material : new Cesium.PolylineGlowMaterialProperty({
            glowPower : 0.1,
            color : Cesium.Color.RED
        }),
        width : 10 },

    label : {
    text : 'ESEO',
    font : '15pt monospace',
    style: Cesium.LabelStyle.FILL_AND_OUTLINE,
    outlineWidth : 4,
    verticalOrigin : Cesium.VerticalOrigin.BOTTOM,
    horizontalOrigin : Cesium.HorizontalOrigin.LEFT,
    pixelOffset : new Cesium.Cartesian2(-30, -60)
    }
});
```

Una proprietà importante dell'entità ESEO è quella del '*model'*: questa consente di caricare un modello 3D del satellite, in formato file *.glb*. Nel caso descritto si è utilizzato il disegno CAD di ESEO fornito da SITAEL, a cui sono stati aggiunti gli assi cartesiani. È necessario poi caricare il modello su Cesium ION nella sezione My Assets e richiamarlo con il suo ID.

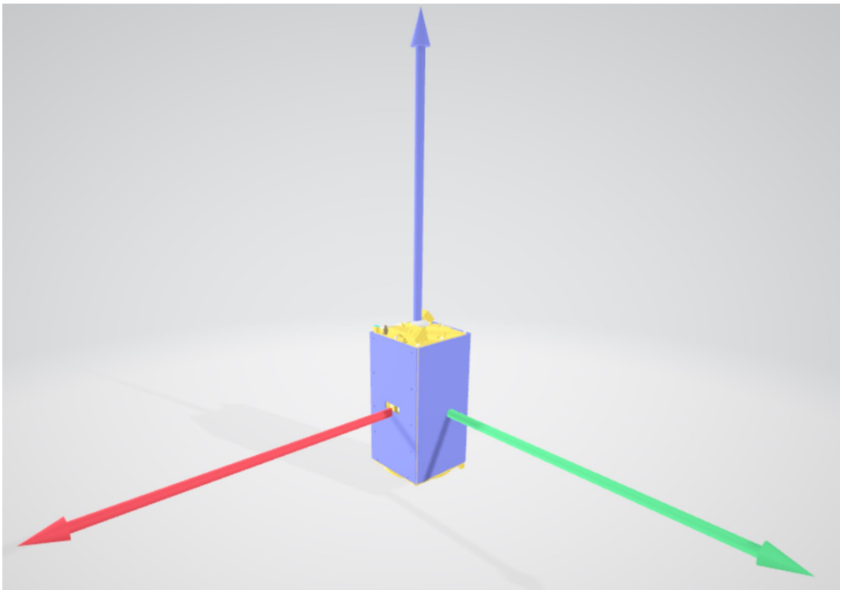

Figure 20: ESEO 3D

Altre due importanti proprietà sono *position* e *orientation,* alla quale sono stati assegnati le variabili *posizione* (vedi 3.2.6 Interpolazione dell'orbita e traccia a terra) e ori (vedi 3.2.4 Caricamento TLE e quaternioni aggiornati).

Path , invece, visualizza il percorso compiuto dall'entità: è questa proprietà che permette di visualizzare l'orbita, dato che il satellite si muove seguendo la variabile posizione, che è interpolata come SampledProperty (vedi 3.2.6 Interpolazione dell'orbita e traccia a terra).

Nel caso del secondo pannello, che richiedeva la visione diretta del satellite, è stata aggiunta la seguente riga di codice, per impostare la telecamera del visualizzatore su una visione in prima persona su ESEO. ' *TrackedEntity'* è una funzione di Cesium.

```
1.  viewer.trackedEntity = entity;
```

Per le due Ground Station è stato seguito per lo più lo stesso procedimento, ma con alcune sostanziali differenze. La posizione è fissa sulle coordinate della GS, non vi è availability in quanto le GS sono sempre disponibili presenti sul territorio, ed è stata caricata un'immagine con il simbolo di un'antenna per indicare la sua posizione.

Figure 21: Simbolo GS

	Latitudine	Longitudine
Forlì GS	44.20057	12.0671
Tartu GS	58.265651	26.465951

Table 29: Coordinate GS

Per quanto riguarda gli accessi, invece, è stato necessario creare una nuova SampledProperty per ottenere le coordinate degli estremi di una polilinea che deve collegare la GS con il satellite quando quest'ultimo risulta visibile. Il tutto è descritto nel paragrafo 3.2.7 Accessi.

Figure 22: Ground Stations

3.2.6 Interpolazione dell'orbita e traccia a terra

Per creare l'orbita del satellite è stato necessario calcolare una serie di punti e successivamente interpolarli con una polilinea. È stata creata innanzitutto una funzione *printPosition* che calcola questi punti e li salva in una *Sampled Property*. Quest'ultima è un "Property object" (un'oggetto contenente una serie di proprietà) di Cesium che consiste in un set di campioni (samples) che può essere interpolato nel tempo e secondo uno specifico algoritmo d' interpolazione.

In questo caso abbiamo utilizzato una SampledPositionProperty, chiamata semplicemente 'Property'. La SampledPositionProperty consente di creare una Sampled Property di posizioni sul globo.

I vari punti vengono quindi calcolati in un ciclo 'for' che va da 0 fino a 'tempo + 250', quindi poco di più della durata di un periodo orbitale (vedi 3.2.3 Settaggio orologio del visualizzatore), con un passo di 30 secondi.

In Figure 23**: Orbita non interpolata** è possibile vedere il risultato senza interpolazione.

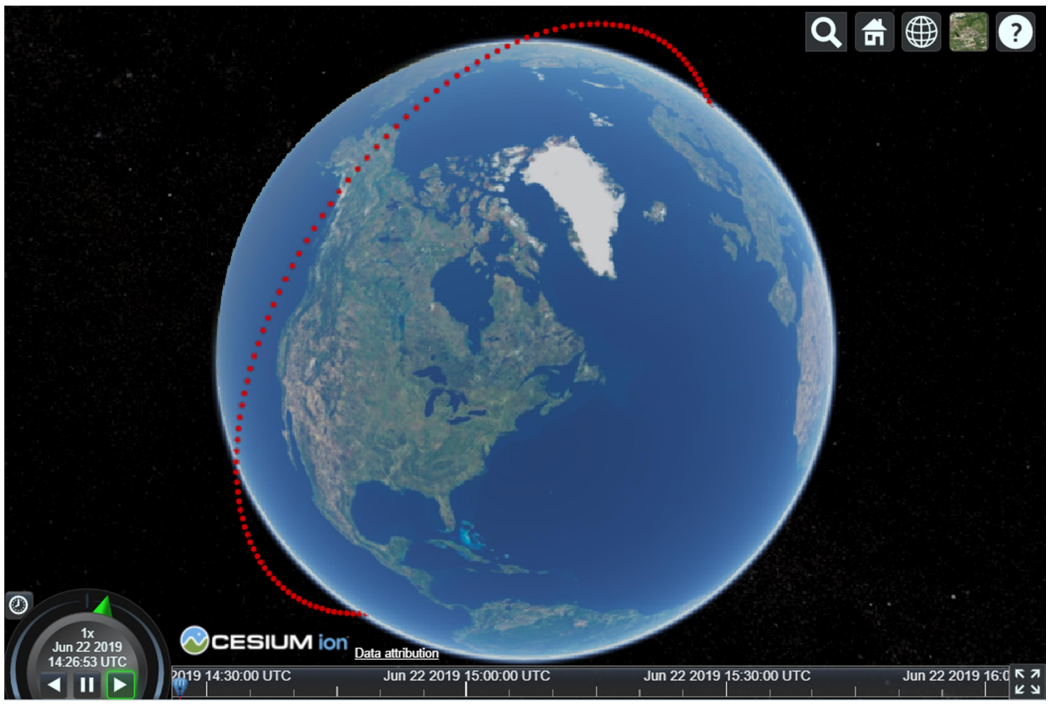

Figure 23: Orbita non interpolata

Di seguito viene fornito il codice della funzione printPosition:

```
1.  function printPosition() {
2.
3.      var property = new Cesium.SampledPositionProperty(Cesium.ReferenceFrame.INERTIAL);
4.
5.      for (var i = 0; i <= tempo + 250; i += 30) {
6.          var time = Cesium.JulianDate.addSeconds(start, i, new Cesium.JulianDate());
7.          julianday = Cesium.JulianDate.totalDays(time);
8.          var positionAndVelocity = SGP4.propogate(eseoSatRec, julianday);
9.
10.         var gmst = SGP4.gstimeFromDate(julianday);
11.
12.         var geodeticCoordinates = SGP4.eciToGeodetic(positionAndVelocity.position, gmst);
13.
14.         var longitude = SGP4.degreesLong(geodeticCoordinates.longitude);
```

```
15.         var latitude = SGP4.degreesLat(geodeticCoordinates.latitude);
16.         var height = geodeticCoordinates.height;
17.
18.         var position = new Cesium.Cartesian3(positionAndVelocity.position["x"]*1000,positionAndVeloci
    ty.position["y"]*1000,positionAndVelocity.position["z"]*1000);
19.             property.addSample(time, position);
20.
21.             posTraccia = new Cesium.Cartesian3.fromDegrees(longitude,latitude,0);
22.             traccia.addSample(time, posTraccia);
```

Tutte le funzioni calcolano la posizione del satellite per un dato tempo che ad ogni ciclo si aggiorna (vedi righe 6-7). Tali funzioni sono tutte state create all'interno di SGP4, implementato al paragrafo 3.2.2 Implementazione di SGP4. Si calcola quindi la posizione in coordinate geodetiche e cartesiane e ad ogni ciclo viene aggiunto un campione alla variabile Property creata in precedenza tramite `property.addSample(time, position);`

Per interpolare i punti è stato scelto un algoritmo di tipo lagrangiano. L'interpolazione di Lagrange è un particolare tipo di interpolazione polinomiale in cui il polinomio interpolatore è definito in questo modo:

$$P(x) = \sum_{i=0}^{n} f(a_i) \prod_{j \neq i}^{n} \frac{x - a_j}{a_i - a_j}$$

dove, nel nostro caso, f(x) è la funzione che calcola la posizione e $f(a_0)$, $f(a_1)$ … $f(a_n)$, sono i punti calcolati.

Di seguito il codice per l'interpolazione.

```
1.  entity.position.setInterpolationOptions({
2.      interpolationDegree : 5,
3.      interpolationAlgorithm : Cesium.LagrangePolynomialApproximation,
4.  });
```

In questo codice 'Entity' rappresenta il satellite (descrizione nel paragrafo 3.2.6 Interpolazione dell'orbita e traccia a terra) e position la sua posizione. Viene scelto un polinomio di grado 5 e l'algoritmo di interpolazione opportuno.

Infine viene salvata la posizione in una variabile '*posizione*' che verrà poi utilizzata da assegnare alle proprietà del satellite (vedi 3.2.5 Creazione dello scenario e delle entità).

```
1.  var posizione = printPosition();
```

Per quanto riguarda la traccia a terra il procedimento è pressoché analogo, tranne che per altezza e sistema di riferimento. L'altezza dei punti naturalmente è fissata a 0, quindi sulla superficie, mentre il sistema di riferimento è di tipo **fisso**, a differenza di quello per l'orbita che era stato definito **inerziale**.

```
1.  var traccia = new Cesium.SampledPositionProperty(Cesium.ReferenceFrame.FIXED);
2.  var property = new Cesium.SampledPositionProperty(Cesium.ReferenceFrame.INERTIAL);
```

La variabile *traccia* è la *SampledPositionProperty* per la traccia a terra, mentre *posTraccia* serve per salvare momentaneamente ad ogni ciclo il campione calcolato e aggiungerlo con `addSample`.

Come abbiamo visto, Cesium consente di definire due diversi sistemi di riferimento: quello fisso e quello inerziale. Per la traccia a terra è necessario utilizzare un sistema fisso con origine al centro della terra, perché bisogna tener conto del

fatto che, mentre il satellite orbita, la terra al di sotto sta ruotando. Questo si nota dal fatto che la traccia non si chiude perfettamente dopo un giro, ma il satellite al periodo successivo passerà sopra un punto più a ovest del precedente.

L'orbita invece utilizza un sistema inerziale.

In Figure 24: Orbita + traccia a terra sono evidenziate entrambe le linee.

Figure 24: Orbita + traccia a terra

3.2.7 Accessi

Come ultima parte del progetto, è stata implementata una funzione per visualizzare gli accessi del satellite alla Ground Station. Per accesso si intende un intervallo di tempo in cui il satellite è in visibilità con la GS e durante questo arco temporale si disegna una linea che collega le due entità per tutta la durata dell'accesso. Per fare ciò è stato necessario fare una nuova SampledProperty e calcolare, all'interno del ciclo 'for' , l'elevazione del satellite in ogni punto della sua orbita, rispetto alla posizione dell'osservatore. Dopodiché è stata creata la polilinea *pas* che viene visualizzata solamente quando l'elevazione è maggiore di 0.

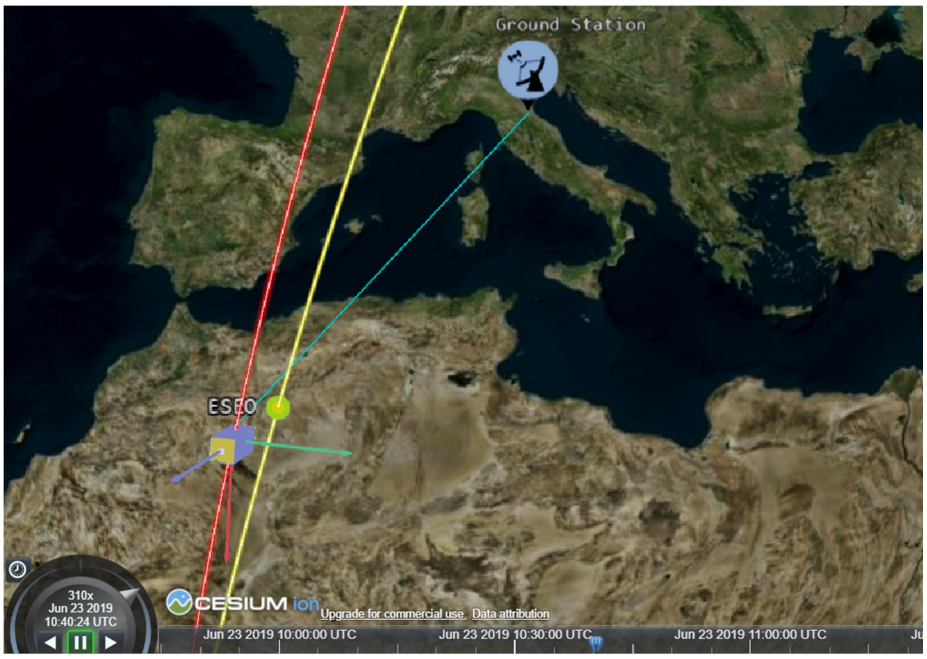

Figure 25: Accesso

Per prima cosa vengono inizializzate la posizione dell'osservatore (coordinate della GS), il tempo di inizio e di fine passaggio, l'elevazione e una collezione di intervalli di durata.

```
var observerPosF = {
    longitude: 12.0671 * SGP4.deg2rad,
    latitude: 44.20057 * SGP4.deg2rad,
    height: 1
};

var start_pasF = 0;
var stop_pasF = 0;

var intervalF = new Cesium.TimeIntervalCollection();

var ElevationF=0;
```

Successivamente si crea la SampledPositionProperty per la posizione degli estremi della polilinea che collegherà le due entità. Come per la traccia si utilizza un sistema di riferimento fisso.

```
var poli = new Cesium.SampledPositionProperty(Cesium.ReferenceFrame.FIXED);
```

Dopodiché si entra nel ciclo 'for' e si comincia ad aggiornare le variabili: grazie ad un'istruzione di tipo 'if – elseif' si calcola in ogni punto l'elevazione e si assegna opportunamente il tempo a *start_pas* e *stop_pas*.

```
1.   var satEcf = SGP4.eciToEcf(positionAndVelocity.position, gmst);
2.   var lookAnglesF = SGP4.topocentricToLookAngles(SGP4.topocentric(observerPosF, satEcf));
3.          var oldElF = ElevationF;
4.          ElevationF = lookAnglesF.elevation * SGP4.rad2deg;
5.
6.  if (oldElF<0 && ElevationF>0) {
7.              start_pasF=time;
8.          } else if (oldElF>0 && ElevationF<0) {
9.              stop_pasF=time;
10.             var interF = new Cesium.TimeInterval({
11.                 start: start_pasF,
12.                 stop: stop_pasF
13.             });
14.             intervalF.addInterval(interF);
15.         }
```

Le funzioni *eciToEcf* e *topocentricToLookAngles* sono funzioni di SGP4. Se l'elevazione nel punto precedente è minore di 0, e quella del punto in questione è maggiore di 0, allora si assegna il tempo alla variabile *start_pas.* Se invece l'elevazione nel punto precedente era positiva mentre quella del punto in questione è negativa, allora si assegna il tempo alla variabile *stop_pas* e contemporaneamente si crea l'intervallo tra le due. Intervallo che viene poi aggiunto alla collezione creata in precedenza.

Sempre all'interno dello stesso ciclo 'for', che ci serve per calcolare la posizione del satellite (vedi 3.2.6 Interpolazione dell'orbita e traccia a terra) e la collezione di intervalli di passaggio, si aggiunge una variabile di supporto, *posPoli*, che servirà per aggiungere ogni 'sample' alla SamplePositionProperty *poli* che abbiamo creato.

```
1.   var posPoli =  new Cesium.Cartesian3.fromDegrees(longitude,latitude,height*1000);
2.          poli.addSample(time, posPoli);
```

Infine è stata creata un'entità polilinea di questo tipo:

```
1.  pasF = viewer.entities.add({
2.      id: 'pasF',
3.
4.      availability : intervalF,
5.      name: 'pasF',
6.
7.      position : poli,
8.
9.      polyline: {
10.         followSurface: false,
11.         positions: new Cesium.PositionPropertyArray([
12.             new Cesium.ReferenceProperty(collection,'GSF',['position' ]),
13.             new Cesium.ReferenceProperty(collection,'pasF',[ 'position' ])
14.         ]),
15.         width : 5,
16.         arcType : Cesium.ArcType.NONE,
17.         material : new Cesium.PolylineArrowMaterialProperty(Cesium.Color.CYAN)
18.
19.     }
20. });
```

La *availability* definisce l'intervallo di tempo in cui l'entità deve essere visualizzata, nel nostro caso la variabile *intervalF* (collezione di intervalli), e ciò consente al programma di visualizzare anche più di un passaggio durante la stessa simulazione.

Come ‘position property’ si utilizza la variabile *poli* e successivamente si definiscono gli estremi della polilinea. Per fare ciò è stata utilizzata una particolare funzione di Cesium, *ReferenceProperty*, che serve per far riferimento ad una determinata proprietà di un’entità.

Ad esempio scrivendo ‘ `new Cesium.ReferenceProperty(collection,'GSF',['position' ])’` si istruisce il programma a riferirsi alla proprietà `'position'`, dell’entità con id `'GSF'`, contenuta all’interno della collezione `‘collection’`. In questo modo, gli estremi della polilinea saranno la posizione della Ground Station di Forlì e la posizione stessa di pas, a cui è stata assegnata *poli*, una SampledPositionProperty.

Lo stesso procedimento è stato ripetuto per entrambe le stazioni di terra: Forlì e Tartu.

3.3 Pagina HTML di avvio

Per rendere tutto il programma più snello e facile da richiamare da un pannello di Grafana, è stato creato un file HTML di avvio.

Questo deve contenere all'interno di 'script tags' tutti i programmi che devono funzionare contemporaneamente all'interno del visualizzatore, il modello 3D del globo di Cesium e il link di connessione a Cesium ION.

Per caricare i vari programmi sulla pagina si è utilizzato l'attributo *src* (source) per indicare il file sorgente del codice. I tags sono i seguenti:

```
1.  <div id="cesiumContainer"></div>
2.  <script> Cesium.Ion.defaultAccessToken = 'link per l'accesso a Cesium ION';</script>
3.  <script src="richiestaQuaternioni.js"></script>
4.  <script src="richiestaTle.js"></script>
5.  <script src="sgp4.js"></script>
6.  <script src="ESEO_JS_quat.js"></script>
```

La pagina è stata poi formattata adeguatamente con titolo, larghezza e scala.

```
1.    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, minimum-
    scale=1, user-scalable=no">
2.    <title>ESEO Live Zoom!</title>
```

Ultima fase del lavoro è stata caricare tutto il programma in un pannello Grafana: è stato necessario quindi creare un '*iframe*' (un elemento HTML interno alla pagina) e indicare come *src* (sorgente) la pagina HTML creata appositamente per il programma. Il tutto è stato ripetuto per il pannello di zoom sul satellite.

Il risultato è quello in Figure 26: Applicazione completa e può essere visualizzato al link in bibliografia (ESEO).

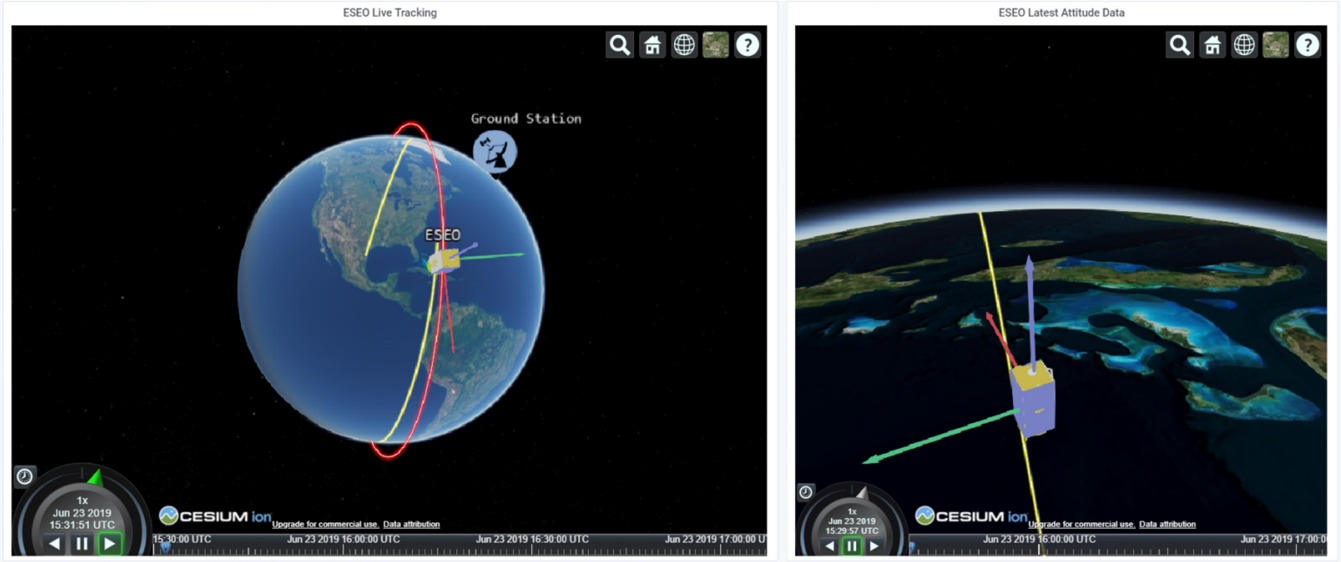

Figure 26: Applicazione completa

4 Conclusioni e lavori futuri

Per concludere, lo scopo della prima parte di questa tesi è stato quello di organizzare una serie di dati di telemetria del satellite ESEO attraverso l'utilizzo di una piattaforma chiamata Grafana. I dati sono stati ordinati in tabelle, grafici e contatori singoli e resi disponibili online. Lo scopo di base è stato rendere una serie di dati che si presentano al database come semplici sequenze di bit, 0 e 1, a prima vista senza significato, umanamente leggibili e comprensibili. Grazie a ciò è stato possibile monitorare l'andamento della telemetria di ESEO, analizzarne i comportamenti e accorgersi velocemente di cambiamenti imprevisti di funzionalità e performance. La seconda parte della tesi, invece, ha raggiunto lo scopo di sviluppare un visualizzatore 3D dello stato orbitale e d'assetto di ESEO. Attraverso l'utilizzo di Cesium e NodeJS, è stata creata un'applicazione che mostra il satellite muoversi attorno al globo terrestre, seguendo la sua orbita corretta e visualizzando la traccia a terra e gli accessi alle due Ground Station. Il modellino 3D del satellite si muove in tempo reale con le giuste coordinate e mostra il satellite orientato secondo l'assetto registrato nell'ultimo passaggio. Per fare ciò, è stato necessario creare un piccolo server sul quale far girare il programma, e implementare un'architettura di tipo REST per fornire al programma i dati reali ed aggiornati sui quaternioni e TLE. I primi prelevati direttamente dal database UNIBO MCC, che viene popolato dalla ground station, mentre i secondi estratti dal sito CelesTrack. Per completare il lavoro è stata aggiunta una seconda applicazione, analoga alla prima ma con una visuale zoomata su ESEO. Da requisito il satellite si vede fermo in questo pannello, per monitorarne l'assetto, ma è comunque possibile avviare la simulazione premendo play sull'orologio. Per ragioni grafiche, in questo pannello è stato scelto di eliminare l'orbita e gli accessi alle GS, ma lasciare soltanto il satellite, la traccia a terra e le due GS sulla superficie.

Questo lavoro di tesi mi ha permesso di sviluppare nuove conoscenze sulla gestione di database con linguaggio SQL e programmazione JavaScript, in particolare l'utilizzo della libreria CesiumJS. Ho potuto osservare da vicino il lavoro svolto in un Mission Control Centre e partecipare in maniera attiva alla gestione e organizzazione dei dati di telemetria inviati. Inoltre ho acquisito alcune nozioni di base sulla programmazione di server tramite NodeJS e sull'utilizzo delle architetture REST per interrogare un database.

Come possibili lavori futuri su questo visualizzatore si potrebbe implementare una funzione che, in tempo reale, registra e mostra quanti radioamatori si stanno connettendo in quel momento al satellite oppure, quando tutti i sistemi saranno operativi, visualizzare sull'applicazione le foto e i dati (che possono esser resi disponibili) che ESEO sta prelevando in quel momento, ad esempio i livelli di radiazione misurati sull'orbita LEO oppure le foto scattate dalla microcamera.

Ringraziamenti

Vorrei ringraziare in primis i miei genitori, Mario e Angela, i quali oltre che avermi sempre sostenuto e supportato con amore, mi hanno dato la possibilità di compiere questo percorso di laurea triennale. Grazie per avermi aiutato a superare le difficoltà, avermi reso la persona che sono adesso e per essermi sempre stati vicini in ogni scelta, la vostra presenza è stata per me essenziale.

Ringrazio anche i professori e le persone che mi hanno seguito in questo percorso di tesi, per i consigli e la disponibilità dimostratami: il prof relatore P. Tortora, prof. A. Locarini, ing. G. Mariotti e ing. A. Lucci.

Ringrazio la mia fidanzata Laura, per la pazienza e la vicinanza in questi 5 anni insieme, mio fratello Federico, i miei amici e coinquilini per il sostegno e l'aiuto.

Infine un ringraziamento speciale ai miei allenatori Marco, Giulia e Luisa che sono stati per me un esempio in tante situazioni, mi hanno insegnato a non mollare mai, non fermarmi di fronte alle difficoltà, rimanere concentrato e a dare sempre il meglio di me; valori fondamentali nello sport, ma ancor più determinanti nella vita. Il nuoto in questi anni mi ha dato tanto, e loro ne sono gli intermediari.

Indice delle figure

Indice delle tabelle

Bibliografia

API Reference. Tratto da Cesium: https://cesiumjs.org/refdoc/

Avanzi, A. (2018). OBDH HK and TC description.

CelesTrack ESEO. Tratto da CelesTrack: https://www.celestrak.com/satcat/search.php

Comini, M. (2016). Orbit determination with the Simplified Perturbation Model.

Download modulo SGP4 . Tratto da npmjs: https://www.npmjs.com/package/sgp4

ESA Education. Tratto da ESA: https://www.esa.int/Education/ESEO

ESEO . Tratto da ESEO status: http://eseo.ddns.net/

Kelso, D. T. *Orbital Coordinate Systems, Part I*. Tratto da CelesTrack: http://www.celestrak.com/columns/v02n01/

Kelso, D. T. *Orbital Coordinate Systems, Part II*. Tratto da CelesTrack: http://www.celestrak.com/columns/v02n02/

Kelso, D. T. *Orbital Coordinate Systems, Part III*. Tratto da CelesTrack: http://www.celestrak.com/columns/v02n03/

OBDH_AR-03.

OBDH_AR-04.

TMTC_AR-03.

Tortora, P. (2018). Appunti di Meccanica Orbitale. Forlì.

Wandshneider, M. (2013). *Node.js: creare applicazioni in JavaScript.* APOGEO.