ALMA MATER STUDIORUM
UNIVERSITA' DI BOLOGNA

SCUOLA DI INGEGNERIA Sede di Forlì

Corso di Laurea in
INGEGNERIA AEROSPAZIALE
Classe L-9

ELABORATO FINALE DI LAUREA

In Satelliti e Missioni Spaziali

Implementazione di tool grafico per la visualizzazione dei dati di telemetria della missione ESEO

CANDIDATO
Tommaso Fadda

RELATORE
Alfredo Locarini

Anno Accademico 2018/2019

Indice

Indice delle tabelle

Indice delle figure

Introduzione

Nell'ottica dell'analisi e controllo di una missione spaziale, assume un ruolo fondamentale il ground segment, ovvero la parte della missione che comprende la stazione di terra e i sistemi di ricezione dei segnali emessi dal satellite verso la terra. Il ground segment si occupa di ricevere questi dati e rielaborarli, in modo da consentire al personale del controllo di missione o al satellite stesso di attuare le procedure necessarie in ogni fase della missione. Risulta molto importante a tale scopo l'elaborazione e l'archiviazione dei dati di telemetria, nonché una loro rielaborazione *human readable*, che possa aiutare ad accedere rapidamente a tutti i dati disponibili inviati dal satellite e a trarne grafici o altre rappresentazioni dell'informazione utili a comprendere il comportamento attuale e storico dello spacecraft.

Lo scopo di questo progetto di tesi è di implementare un'interfaccia grafica utile a visualizzare i dati di telemetria della missione ESEO dell'ESA, la cui ground station è sita presso il tecnopolo della sede di Forlì dell'università degli studi di Bologna. L'applicazione scelta per la visualizzazione dei data è Grafana, una piattaforma che consente di creare pagine web utilizzando vari tool per creare grafici, tabelle e diagrammi con dei dati provenienti da un database.

Il lavoro di tesi è quindi stato diviso in due parti. Nella prima sono state implementate tramite Grafana tutti i dati di telemetria del satellite, che sono suddivisi a seconda del sottosistema o del payload a cui fanno riferimento. Tali dati sono visualizzati tramite grafici che ne rappresentano l'andamento dei valori nel tempo o tramite tabelle che riportano lo stato attuale dei sistemi e gli stati precedenti. Nella seconda parte del lavoro di tesi si è invece realizzata un'applicazione per la visualizzazione real time della posizione del satellite, utilizzando un globo 3D tramite la libreria di Javascript Cesium. Anche in questo caso il lavoro è stato caricato sulla pagina web basata su Grafana, insieme ai dati di telemetria del satellite.

Nella stesura di quest'elaborato si è proceduto inizialmente nel primo capitolo a descrivere la missione ESEO, dagli obiettivi di missione alla struttura dello space segment e del ground segment. Quindi ci si è soffermati sulle caratteristiche dell'orbita e sulle modalità di visualizzazione della stessa, tramite il modello di propagazione SGP4 e i dati orbitali in formato TLE. La descrizione di questi due aspetti risulta necessaria per comprendere il modo in cui si è lavorato alla visualizzazione 3D dell'orbita del satellite, di cui si parlerà nei capitoli successivi. Alla fine del primo capitolo si descrivono quindi i protocolli di telemetria utilizzati nella missione, e la struttura dell'apparato di archiviazione dei dati tramite database MySQL da cui Grafana attinge, ponendo in particolare l'attenzione sui campi disponibili per ogni dato. Il caricamento di questi dati è descritto nel secondo capitolo, nel quale si trattano le principali tipologie di visualizzazione dei dati e la logica secondo la quale questi sono stati organizzati e visualizzati. Nel terzo capitolo infine viene descritta la realizzazione di un visualizzatore 3D per mostrare la posizione orbitale del satellite in tempo reale. Per spiegare il funzionamento di tale applicazione saranno descritte le basi di Node.js, una runtime per javascript che offre molte possibilità per lo sviluppo delle applicazioni web, e sarà introdotta la libreria Cesium.

1. ESEO

1.1 Missione

ESEO (European Student Earth Orbiter) è un satellite a scopo educativo sviluppato dall'ESA Academy come parte di un progetto dell'Educational Office dell'ESA in collaborazione con diverse università europee, i cui studenti hanno partecipato attivamente alla progettazione e realizzazione del satellite e della missione. Il satellite infatti rappresenta la terza missione del programma educativo dell'ESA, dopo la missione SSETI Express (lanciata nel 2005), e la missione YES2 (lanciata nel 2007).

L'obiettivo della missione ESEO è di fornire agli studenti delle università partecipanti esperienza nella realizzazione di una vera missione spaziale, per formare dei profili altamente specializzati e qualificati che possano inserirsi nei prossimi anni nel contesto del settore spaziale europeo. Gli studenti hanno avuto la possibilità di partecipare sia alla realizzazione del payload, sia a quella dei principali sottosistemi e del segmento di terra, con il coordinamento dell'ESA e di Sitael, il partner industriale della missione, che ha prodotto la piattaforma satellite. Alla missione hanno partecipato più di 600 studenti provenienti da dieci università di otto diversi paesi europei (Estonia, Germania, Ungheria, Olanda, Italia, Spagna, Polonia, Regno Unito).

Il satellite è stato lanciato in orbita bassa (LEO) il 3 dicembre 2018, come parte del lancio SSA-O: SmallSat Express, a bordo di un lanciatore Falcon 9.

La missione è basata su un'orbita eliosincrona (SSO) con altitudine di 575 km, inclinazione di 97.5° e passaggio sull'equatore circa alle 10:30 UTC. Il periodo orbitale è di circa 94 minuti, e l'assetto nominale del satellite è Nadir-pointing. La durata nominale della missione è di 6 mesi, con la possibilità di prolungarla per ulteriori 6 mesi, fino alla durata massima di un anno. Al termine della missione il rientro del satellite sarà accelerato tramite l'utilizzo di una vela estendibile.

Il primo scopo educativo della missione è quello riguardante la realizzazione del satellite, ovvero di tutti i suoi sottosistemi e di tutti i payload, nonché il lancio del satellite e l'immissione in orbita.

L'università di Bologna ha un ruolo primario nella missione, in quanto è responsabile della ground station principale, del ricevitore GNSS e della ricezione dei dati in banda UHF. Altre due stazioni di terra sono presso il l'università di Vigo, in Spagna (come stazione ridondante rispetto a quella di Forlì) e presso Monaco di Baviera, in Germania (stazione di terra per la ricezione dei dati per il Payload). Tutte le altre Università partecipano alla missione sono per quanto riguarda lo space segment, occupandosi della progettazione e realizzazione dei vari Payload.

Gli obiettivi di missione sono quindi legati al funzionamento dei vari payload a bordo, durante la fase operativa della missione, tutti riconducibili a esperimenti scientifici o dimostrazioni tecnologiche progettati e costruiti degli studenti. Tali obiettivi sono:

- Scattare foto della Terra

  Una microcamera scatta foto della Terra nel visibile, per scopi didattici. La camera è sviluppata dall'università di Tartu, in Estonia.
- Misurare i livelli di radiazioni in orbita terrestre bassa.

  Una Langmuir probe e un dosimetro a tre assi sono utilizzati per misurare il livello di radiazione assorbita, la dose equivalente (a livello di effetti biologici), e lo spettro del trasferimento lineare di energia (LET) dei raggi cosmici. Gli strumenti sono stati sviluppati dalla University of Technology and Economics (BME) e dalla Academy of Science (MTA-EK) di Budapest, in Ungheria.
- Test di tecnologie per le future missioni educazionali ESA

  Si utilizza un sistema di comunicazione in Banda S, sviluppato dalla Wroclaw University of Technology della Polonia, per il downlink dei dati scientifici; un ricevitore GPS sviluppato dall'Università di Bologna per testare la possibilità di determinare la posizione in tempo reale in orbita LEO basandosi sulle tecnologie GNSS; un Attitude Determination Estimator (ADE), software sviluppato dalla Delft University of Technology dei Paesi Bassi, per valutare l'efficienza di quattro differenti algoritmi per la determinazione dell'attitudine e dell'orbita. Alla fine delle operazioni di volo, un meccanismo di de-

orbiting, sviluppato dalla Cranfield University, nel Regno Unito, sarà utilizzato per accelerare il tempo di rientro in atmosfera.

- Utilizzare le stazioni di terra alle bande UHF e S

  Una stazione di Terra in banda UHF sita in Forlì viene utilizzata per uplink e il downlink dei dati operativi. Una stazione di Terra di back-up è attiva presso l'università di Vigo, in Spagna. I dati degli esperimenti sono invece trasmessi alla stazione di Terra dell'università di Monaco, in Germania, che lavora in banda S.

- Consentire la connessione radio amatoriale in banda VHF al satellite

  Il payload AMSAT, fornito da AMSAT UK, in collaborazione con l'università di Surrey, permette di connettersi e scaricare i dati del satellite da centinaia di stazioni di terra appartenenti al network AMSAT. I dati scaricati saranno in seguito utilizzati per svolgere lezioni scientifiche e tecnologiche presso scuole e università.

Con la realizzazione di questi obiettivi si completa anche l'obiettivo di missione principale, ovvero quello educativo nei confronti degli studenti partecipanti al progetto.

1.2 Architettura del satellite

ESEO è un microsatellite, con un peso approssimativo di 50 kg di dimensioni 33x33x63 cm. Tuttavia, considerando anche l'antenna superiore e le appendici sulla faccia inferiore, l'altezza arriva a 107 cm. La struttura del satellite è stata prodotta da Sitael, ed è costituita da vari cassetti sovrapposti, all'interno dei quali sono collocati tutti i componenti del satellite. La struttura riprende quella di ALMASAT-1, primo satellite progettato e costruito dall'Università di Bologna attraverso la spin-off AlmaSpace, poi assorbita da Sitael. La piattaforma di ESEO è basata sulla S-50, che rappresenta la più piccola delle piattaforme satellitari realizzate da Sitael.

La struttura del satellite può essere suddivisa nel Bus Module, che contiene tutti i sottosistemi, e nel Payload Module, che contiene la maggior parte dei Payloads. Tali moduli sono separati fisicamente, in modo da rendere il sistema più flessibile e semplice da assemblare e controllare. L'utilizzo di una struttura a cassetti, realizzata

in alluminio, deriva dal know how acquisito da AlmaSpace nelle precedenti missioni.

Per una maggiore affidabilità quasi tutti i sottosistemi del satellite sono stati realizzati in ridondanza calda o fredda, in particolare l'intera piattaforma S-50, che sarà validata con la missione ESEO, è stata progettata secondo un criterio di ridondanza totale.

La configurazione finale del satellite prevede esternamente tre facce ricoperte da pannelli solari, e una costituita da un radiatore per la dispersione del calore, orientato sempre verso lo spazio profondo grazie al sottosistema di controllo di assetto.

1.2.1 Bus Module

I vari vassoi della struttura del satellite forniscono il supporto strutturale per il lancio e per l'integrazione dei componenti dei sottosistemi e del payload. La suddivisione dei componenti nei vari vassoi è la seguente:

- *Tray 1*: sistema di micropropulsione (MPS);
- *Tray 2*: ruota di momento (MW), magnetorquer (MT), (Y-axis, redundant);
- *Tray 3*: sottosistema potenza (PS), scheda di gestione della potenza, unità di distribuzione della potenza, pacchi batterie, interfaccia con pannelli laterali e celle laterali, più il set di connettori per le operazioni MAIT (manufacturing, assembly, integration and testing);
- *Tray 4*: sottosistema On-board Data Handling (OBDH), Magnetorquer (MT) per il controllo di assetto attorno all'asse X;
- *Tray 5*: magnetometri (MM), ricevitore GPS;
- *Tray 6*: sottosistema di telemetria e telecomando (TMTC);
- *Faccia superiore*: 2 sensori solari (SS), antenna UHF, antenna GPS, antenna AMSAT, connettore EGSE/umbilical (chiude il bus, fornisce la connesione con la strumentazione esterna)
- *Payload module*: earth sensor (ES), serbatoio in titanio del MPS, payloads
- *Pannelli laterali*: 3 pannelli solari, meccanismo di de orbiting (DOM) collocato sul pannello 2 (radiatore)

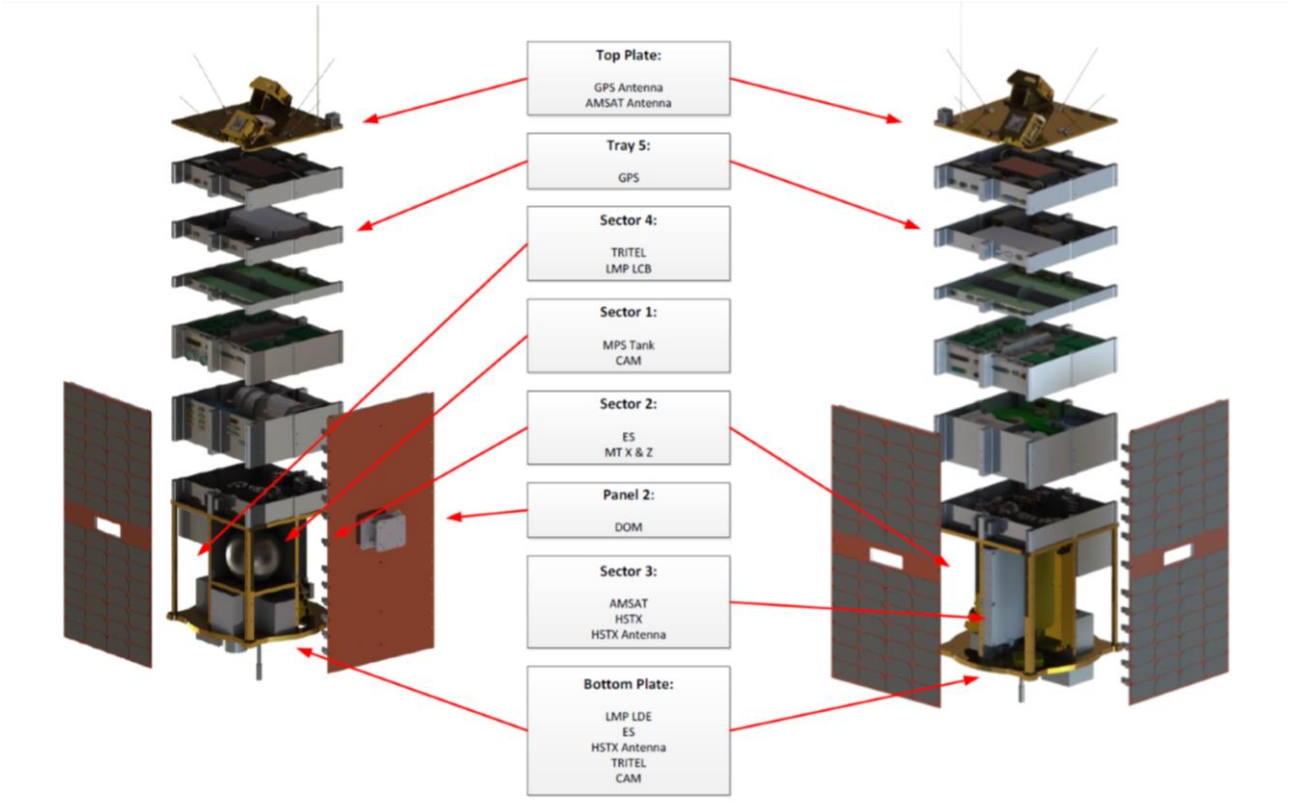

Figura 1. Esploso del satellite, con la vista di tutti i vassoi

I principali componenti del bus module sono il sottosistema di potenza, il sottosistema AOCS, il sottosistema di telemetria e telecomando, l'On-Board Data Handling e il sottosistema di controllo termico. La funzione e la composizione dei vari sottosistemi sono schematicamente elencate in seguito.

Sottosistema potenza elettrica (EPS)

È composto da pannelli solari body-mounted su tre lati del satellite (ognuno formato da 4 stringhe di celle solari), pacchi di batterie agli ioni di litio, la scheda di gestione della potenza, l'unità di distribuzione della potenza.

Il sistema è di tipo non regolato; la tensione di esercizio del sistema varia tra 18 e 25.2 V.

Sottosistema di Attitude and Orbit Control (AOCS)

È basato su tre magnetorquer ortogonali ridondanti, per le manovre di acquisizione di assetto e di puntamento, più due ruote di momento ridondante sull'asse di rollio

(asse X). È incluso anche un sistema di micropropulsione per il controllo orbitale e piccole manovre orbitali.

I sensori di ESEO includono due unità ridondanti di sensori solari, due magnetometri ridondanti AP539 a 3 assi e un earth sensor.

La comunicazione tra i sensori, gli attuatori e il sottosistema AOCS è realizzato tramite un CAN bus, che utilizza un protocollo di tipo CANOpen.

Sottosistema strutture e meccanismi (SMS)

La struttura del bus module è a cassetti, secondo un'architettura ereditata dalle missioni ALMASat-1 e ALMASat-EO.

Il bus module contiene tutti i sottosistemi principali divisi in cassetti di alluminio, e il modulo del payload realizzato con una struttura in composito a nido d'ape e alluminio. La maggior parte dei payload sono contenuti in questo modulo.

La struttura è completata da quattro pannelli laterali a nido d'ape, che costituiscono anche il supporto per i pannelli solari e il radiatore.

Sottosistema telemetria e telecomando (TMTC)

Consiste di una scheda elettronica di ricetrasmissione digitale (RTX), l'unità di distribuzione in radiofrequenze (RFDU) e l'antenna, di tipo turnstile, costituita da una serie di 4 dipoli posti sulla faccia del satellite orientata verso lo zenit.

Il sottosistema TMTC scambia la telemetria e i telecomandi con l'OBDH attraverso un'interfaccia UART inclusa nella scheda elettronica. Il sistema TMTC garantisce la ridondanza fredda del trasmettitore e la ridondanza calda del ricevitore.

On-Board data handling (OBDH)

È equipaggiata con le interfacce necessarie per connettersi all'AOCS tramite due indipendenti bus di tipo CAN, più un ulteriore bus per la connessione con il payload, un'interfaccia UART per la connessione con i sottosistemi TMTC main e redundant.

Le funzionalità dell'OBDH sono: raccogliere i dati di house keeping della piattaforma e del payload, generare report periodici da trasmettere al segmento di terra, monitorare una serie di dati di house keeping rilevanti per poter generare

errori o avvertimenti, ricevere, eseguire e indirizzare telecomandi dal segmento di terra e infine generare comandi interni per il management dei sottosistemi.

Sottosistema di controllo termico (TCS)

È completamente passivo ed è costituito unicamente dal radiatore montato su una delle facce laterali.

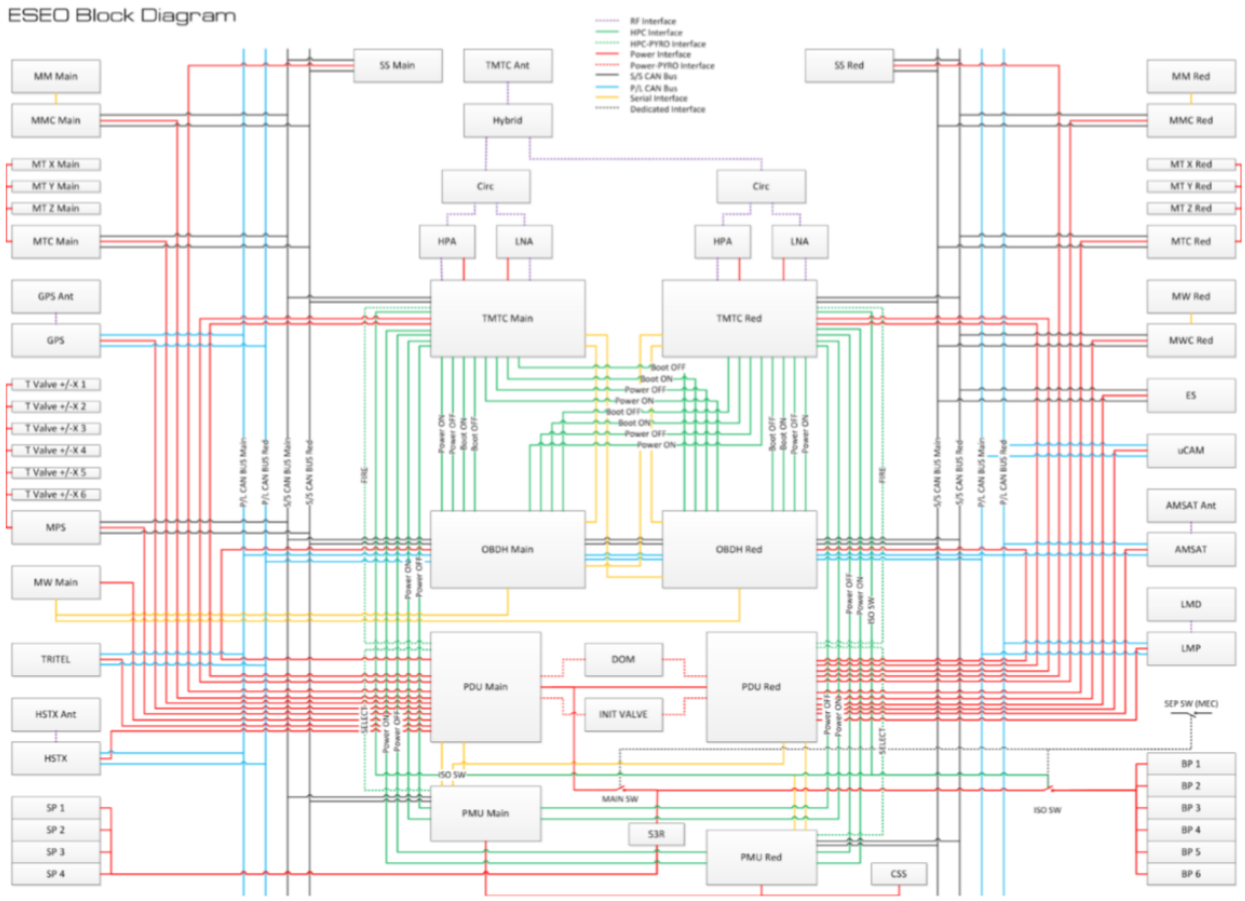

Figura 2. Schema delle connessioni tra i vari sottosistemi e il payload

1.2.2 Payload Module

Il payload rappresenta il carico utile a raccogliere i dati di interesse una volta che il satellite è in orbita in condizioni operate, in modo da adempiere agli obiettivi della missione.

I payload presenti su ESEO sono sette: µCAM (micro Camera), LMP (Langmuir Probe), dosimetro (un detector telescopico 3D in silicone di tipo Tritel-S), HSTX

(High speed transmitter), ricevitore GPS, DOM, AMSAT payload (transponder UHF/VHF e banda S).

µCAM

L'obiettivo di questo payload è di fornire immagini della Terra. Il payload è fornito dall'università di Tartu ed è composto da due microcamere, una primaria ad angolo ampio e una telescopica (la secondaria). La camera primaria ha un angolo di vista di 65.8° ed è a media risoluzione (640x480 pixel); fornisce il contesto per la camera secondaria, dotata di un angolo di vista molto più ristretto (8.16°) e di una risoluzione elevata (5Mpx, con un pixel di misura 20x20 m).

Langmuir Probe

Lo strumento ha lo scopo di misurare le anomalie del plasma solare e di investigare l'attività solare, nonché di misurare le anomalie nel campo geomagnetico terrestre in seguito alle eruzioni solari e alle tempeste solari. In particolare, l'obiettivo di missione è studiare l'anomalia sudatlantica e quella nordeuropea. L'orbita eliosincrona di ESEO fornisce un'ottima occasione a tale proposito. Il payload si compone della Langmuir probe e della scheda elettronica. Lo strumento è cilindrico ed è montato esternamente al satellite e lontano da esso, in modo da misurare il flusso del plasma indisturbato.

TriTel (three dimensional dosimetric telescope)

Si tratta di uno strumento in grado di misurare la dose di radiazione assorbita e la dose equivalente, al fine di valutare gli effetti che le radiazioni avrebbero sulla salute umana. Lo strumento è composto da tre rilevatori orientati lungo i tre assi, in grado di misurare in maniera precisa le radiazioni ricevute a causa dei raggi cosmici e del plasma solare.

HSTX (High Speed Transmitter)

Si tratta di un trasmettitore ad alta velocità in banda S, utilizzato per testare la trasmissione dei dati del payload ad alta velocità. È costituito dal trasmettitore, da un amplificatore e dall'antenna. Un'importante caratteristica è l'autonomina del trasmettitore rispetto all'OBDH, in quanto esso è fornito con una memoria interna per il salvataggio dei dati provenienti dai payload.

Ricevitore GPS

Il payload è composto dall'antenna, collocata sulla faccia superiore del satellite, dal processore in banda base, che genera il segnale carrier in banda base, le osservabili e gli pseudorange, e dall'unità di navigazione, che implementa l'algoritmo per la determinazione della posizione. Lo scopo del payload è quello di valutare la possibilità di realizzare il positioning di un satellite in orbita Leo tramite GPS e di testare l'utilizzo di componenti COTS per la navigazione GPS su un satellite.

DOM (De-orbit Mechanism)

Il meccanismo di de-orbiting sarà utilizzato al termine della missione, per incrementare l'area del satellite e velocizzarne quindi il tempo di rientro. Il congegno è composto da una vela arrotolata e tenuta fissata tramite dei cavi di kevlar. Il particolare packaging consente di contenere la vela all'interno di un supporto molto ridotto (100x100x60 cm), collocato sopra il radiatore del satellite. Dei bracci telescopici sono tenuti in tensione ritratti. Per il dispiegamento del dispositivo i cavi in kevlar vengono tagliati e i bracci telescopici si estendono dispiegando la vela.

AMSAT payload

L'obiettivo del payload è di fornire il downlink della telemetria di ESEO in modo accessibile a scuole e istituti di ricerca per scopi educativi. La trasmissione dei dati avviene in banda L e consente a studenti di tutto il mondo di connettersi tramite una semplice ed economica ground station, composta da un'antenna omnidirezionale con un ricevitore FUNcube USB DonglePRO+ SDR, che riceve i dati direttamente dal satellite e li rende leggibili inviandoli a un software grafico disponibile per tutti i pc Windows.

1.3 Determinazione dell'orbita

La conoscenza dei parametri orbitali del satellite risulta fondamentale per il corretto svolgimento della missione. Questi dati sono ricavabili dalla telemetria del satellite oppure dalle informazioni fornite dal NORAD, ente che persegue il tracciamento di tutti gli oggetti in orbita terrestre. Il NORAD fornisce un catalogo con i dati dei

satelliti utilizzando il formato TLE; questi dati forniscono tutti i parametri orbitali del satellite in un certo istante, a partire da quei dati è quindi possibile valutare l'evoluzione della posizione del satellite propagando l'orbita negli istanti successivi tramite un modello di propagazione. Risulta utile descrivere questi aspetti in quanto sono stati utilizzati nello svolgimento di questa tesi per realizzare il visualizzatore 3D della posizione orbitale di ESEO.

1.3.1 Modello di propagazione SGP4

Il modello SGP4 è uno dei cinque modelli di perturbazione semplificati utilizzati per determinare il vettore posizione di un satellite o di detriti spaziali in un sistema di coordinate inerziale con origine nel centro della Terra. Il modello più utilizzato è quello SGP4, utilizzato dal NORAD e dalla NASA per fornire i parametri orbitali dei satelliti utilizzando il formato dati TLE (Two Line Element Set).

Questi modelli predicono le perturbazioni che si generano su un satellite a causa della forma della terra, del drag atmosferico, delle radiazioni e degli effetti gravitazionali di altri corpi celesti come il Sole e la Luna. In generale si tratta di modelli applicabili a satelliti con periodo orbitale inferiore ai 225 minuti. Il modello SGP4, in particolare, genera un errore variabile tra 1 e 3 chilometri per giorno, e pertanto i dati relativi al satellite vengono aggiornati spesso dal NORAD.

1.3.2 Formato dati TLE

Il formato TLE è un formato di dati utilizzato dal NORAD e dalla NASA per codificare le informazioni orbitali di un satellite con riferimento a un preciso istante temporale (chiamato epoca). Il formato TLE è studiato per adattarsi bene ai modelli di propagazione semplificati, come SGP4, fornendo i parametri orbitali adatti all'implementazione di tali modelli.

Come suggerisce il nome, i TLE sono formati da due stringhe di 69 caratteri ciascuna; nella prima sono memorizzati i dati del satellite, nella seconda i suoi parametri orbitali. Ogni stringa è suddivisa in vari campi, come mostrato nelle figure e tabelle seguenti.

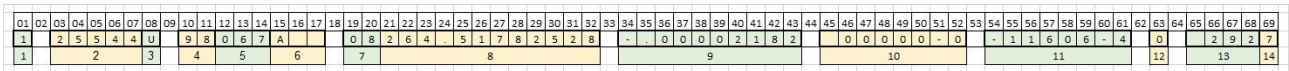

Figura 3. Line 1

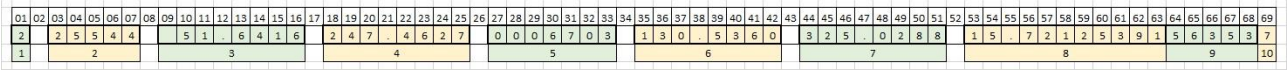

Figura 4. Line 2

Line 1

Contiene i dati di catalogazione del satellite, nonché le informazioni relative all'epoca in cui sono stati generati i TLE e al numero di set generati per quel satellite. La checksum è una somma di tutti i dati contenuti nel TLE, modulo 10, ed è utilizzata per verificare la presenza di errori nei dati.

Campo	Colonna	Contenuto	Esempio
1	01-01	Numero di linea	1
2	03-07	Numero del satellite	43792
3	08-08	Classificazione (U = non classificato)	U
4	10-11	Designatore internazionale (anno di lancio)	18
5	12-14	Designatore int. (Numero di lancio nell'anno)	099
6	15-17	Designatore internazionale (pezzo del lancio)	AL
7	19-20	Anno dell'epoca	19
8	21-32	Epoca (giorno dell'anno e frazione di giorno)	185.18976724
9	34-43	Prima derivata temporale del moto divisa per due	-.00000171
10	45-52	Seconda derivata del moto divisa per sei	00000-0
11	54-61	Termine di drag BSTAR	-10207-4
12	63-63	Numero 0	0
13	65-68	Numero progressivo del set generato	999
14	69-69	Checksum modulo 10	5

Tabella 1. Line 1

Line 2

Contiene tutti i parametri orbitali del satellite necessari per implementare il modello SGP4, ovvero inclinazione, eccentricità, nodo ascendente, argomento del perigeo, anomalia media e moto medio.

Campo	Colonna	Contenuto	Esempio
1	01-01	Numero di linea	2
2	03-07	Numero del satellite	43792
3	09-16	Inclinazione (gradi)	97.7430
4	18-25	Ascendenza retta del nodo ascendente (gradi)	257.0806
5	27-33	Eccentricità (solo parte decimale)	0011928
6	35-42	Argomento del perigeo (gradi)	279.5657
7	44-51	Anomalia media (gradi)	80.4216
8	53-63	Moto medio (rivoluzioni al giorno)	14.95507351
9	64-68	Numero di rivoluzioni effettuate fino all'epoca di riferimento	3150
10	69-69	Checksum (modulo 10)	7

Tabella 2. Line 2

Nella colonna di esempi delle tabelle soprastanti sono riportati i TLE di ESEO, con gli ultimi dati disponibili al momento della stesura di questo documento.

1.4 Ground segment

1.4.1 Stazione di terra di Forlì

Le funzioni di base necessarie al segmento di terra per stabilire un link radio per l'uplink e il downlink della telemetria sono:

- Supporto al segmento spaziale per il comando e il controllo del satellite, monitorando i dati di telemetria e gestendo il link di comunicazione. Questo è fatto tramite un link bidirezionale in banda UHF;
- Gestione della missione, in termini di pianificazione delle attività, risoluzione dei problemi e dei conflitti, ottimizzazione e gestione delle priorità;
- Gestione delle richieste dell'utente, acquisizione dei dati, archiviazione e distribuzione dei dati ricevuti da ESEO con varie stazioni di acquisizione;

L'infrastruttura del segmento di terra comprende le seguenti aree funzionali: area di sfruttamento della missione e area di programmazione e controllo della missione. Ogni area funzionale deve massimizzare lo sfruttamento della missione ESEO dal lato del segmento di terra.

L’area di programmazione e controllo della missione comprende il centro di controllo missione (MCC), il sistema di pianificazione della missione (MPS) e le varie stazioni di Terra, ovvero la GSF (Stazione di terra di Forlì), la GSV (stazione di terra di Vigo) e la GSM (stazione di terra di Monaco). La stazione di terra di Monaco si occupa per lo più del downlink dei dati dei payload, mentre la stazione di terra di Forlì ricopre il ruolo di stazione principale per la gestione della telemetria e il controllo del satellite. La stazione di Forlì lavora in UHF, ed è fornita di due differenti circuiti per l’uplink e il downlink di dati e comandi.

Il circuito primario lavora in banda UHF (430-440 MHz) usando un’antenna Yagi con 2x19 elementi. La stazione di terra secondaria invece serve solo per il downlink dei dati, e opera nella banda S (circa 2.4 GHz), usando un’antenna parabolica di 3 metri di diametro. Entrambe le stazioni si basano su una piattaforma Software Defined Radio (SDR), che consente grande flessibilità nella scelta dello schema di modulazione, nel protocollo di comunicazione e nel processamento dei segnali digitali. Entrambe le antenne tracciano automaticamente il satellite usando l’azimuth e l’elevazione calcolati dai TLE attraverso un propagatore.

1.4.2 Protocolli di telemetria e telecomando

Il capitolo seguente sarà interamente dedicato alla visualizzazione di tutti i dati di telemetria di ESEO tramite l’applicazione Grafana. Risulta tuttavia utile definire anticipatamente i protocolli di trasmissione, ricezione e archiviazione della telemetria e dei telecomandi di ESEO, in modo da dare una visione generale di tutti i dati disponibili e del modo nel quale sono gestiti e raggruppati, così da giustificare il modo in cui i dati sono stati successivamente implementati su Grafana, al fine di ricavarne informazioni utili e consentire un’adeguata gestione del satellite.

Il protocollo di comunicazione tra i sottosistemi TMTC di ESEO e della stazione di terra è una derivazione del protocollo AX.25 ad accesso limitato, utilizzato per la trasmissione radio amatoriale. Il protocollo è utilizzato dal satellite e dalla stazione di terra per inviare e ricevere i telecomandi e la telemetria, nonché i messaggi di acknowledge e rejection.

Le informazioni sono inviate per piccoli pacchetti di dati chiamati frames, divisi in tre tipologie: gli information frames (I), i supervision frames (S) e gli unnumbered frames (U). Ognuno di questi pacchetti è a sua volta suddiviso in sottogruppi chiamati fields (campi). I frames U e S sono composti di 5 campi rispettivamente lunghi 21 e 22 bytes, mentre il frame I ha lunghezza variabile, in quando dipende dalla lunghezza dell'info field e varia tra i 126 e i 159 bytes.

Per la rilevazione e correzione degli errori si usa un codice di Reed-Solomon, utilizzando un campionamento sovradimensionato dei dati.

La struttura delle tre tipologie di frames è riportata in Tabella 3, Tabella 4 e Tabella 5.

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

Tabella 3. U Frame structure

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

Tabella 4. S Frame structure

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

Tabella 5. I frame structure

Tali frame sono suddivisi nei seguenti campi:

- *Flag*: è composto da 2 bytes e delimita il frame. È formato dalla sequenza 0111111001111110;
- *Address*: è composto da 7+7 bytes. Contiene la provenienza e la destinazione del frame;

- *Control*: è formato da due bytes e definisce la tipologia del frame, ovvero S oppure I;
- *FCS (frame clock sequence)*: è una sequenza di 16 bit calcolati sia dal ricevente che dal mittente. In tal modo si assicura che il frame non sia stato corrotto;
- *PID (protocol identifier)*: indica quale tipologia di protocollo del layer 3 è stata implementata. Dal momento che per ESEO non è stato implementato nessun protocollo del layer tre, il byte sarà 11110000;
- *Info*: contiene i dati inviati (dati TC o HK), ed è composto da N bytes.

All'interno del satellite il protocollo tra i sottosistemi TMTC e OBDH (entrambi composti da unità main e redundant) prevede che si utilizzi normalmente l'unità main. Si utilizza generalmente una cross-strapped user interface, basata sullo standard RS-422. Il protocollo tra la stazione di terra e il sottosistema TMTC di ESEO lavora in maniera analoga, con la differenza che i dati vengono trasmessi via radio. In particolare, l'interfaccia seriale è utilizzata dall'OBDH per esplicare varie funzioni, tra cui:

- ricevere le TC standard inviate dalla stazione di terra e ricevute dal TMTC;
- trasmettere i risultati degli standard TC (telecomandi) tramite un messaggio;
- ricevere time-tagged TC;
- trasmettere i dati dei beacon (da TMTC a GS);
- rispondere alla richiesta di telecomandi dedicati, come la trasmissione di pagine di house keeping complete, la trasmissione di PS high rate history segments, la trasmissione delle HK history di certi dati, la trasmissione dei report sullo stato degli equipaggiamenti e della strumentazione, la restituzione di immagini dell'earth sensor o di dati del sottosistema potenza;

Un pacchetto dati standard trasmessi dal satellite a terra è composto dai seguenti campi:

- Un *flag* che segnala l'inizio del pacchetto;
- Un numero (*number*) di sequenza che identifica il pacchetto ed è utilizzato per collegare il messaggio di acknowledge/ rejection trasmesso a terra al comando associato;
- Una classe (c*lass*), definita in seguito nella tabella delle classi;

- Un campo *lenght*, che indica la lunghezza del campo *Data*;
- Il campo *Data*, che cambia in base alla classe del pacchetto e contiene il dato che viene scambiato;
- Il campo *CRC* che controlla l'affidabilità del pacchetto dopo la ricezione e la trasmissione;

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

Tabella 6. Pacchetto standard scambiato tra OBDH e TMTC

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

Tabella 7. Classi di pacchetti

Le varie classi di pacchetti implicano una diversa struttura dei dati trasmessi. In seguito, sono definite le principali tipologie di pacchetti trasmessi.

Standard TC

Rappresenta la tipologia standard di telecomandi inviati al satellite. Il pacchetto è formato dalle seguenti informazioni:

- *address*;
- *sub-address*, se il dato appartiene a una sottotipologia;
- *type*: specifica se il comando è di tipo set o get;
- *register value*: contiene il dato, salvato con little endian representation (ovvero rappresentazione nell'ordine dei bytes).

Address	Sub-address	Type	Register value
1 byte	1 byte	1 byte	4 bytes

Tabella 8. Standard TC Data Field

Time-tagged TC

Possiede gli stessi campi dello standard TC, con l'aggiunta di un'informazione sul tempo nel quale deve essere eseguito il comando. L'informazione è fornita utilizzato tre bytes per indicare i giorni passati da j2000 e altri tre bytes per indicare i secondi del giorno (a partire dalle 12:00 UTC).

Address	Sub-address	Type	Register value	Day	Sec
1 byte	1 byte	1 byte	4 bytes	3 bytes	3 bytes

Tabella 9.Time tagged TC data field

TC acknowledge/ rejection message

Definisce il tipo di dato indicandone l'indirizzo e il sub-indirizzo per l'acknowledgement, o il messaggio di errore nel caso in cui il comando venga respinto.

Type	Address	Sub-address	Register value
0x01	1 byte	1 byte	4 byte

Tabella 10. Acknowledge TC data field

Type	Error message
0x10	2 bytes

Tabella 11. Rejection TC data field

I possibili messaggi di errore sono i seguenti:

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

Tabella 12. Codici di errore

Beacon

I beacon contengono le informazioni principali di tutti i sottosistemi e payload. Sono formati da una stringa di 122 bytes. La suddivisione dei dati contenuti in ogni beacon è la seguente:

- *Beacon 1*: contiene le informazioni generali del satellite (General);
- *Beacon 2*: contiene i dati del sottosistema potenza (Power system);
- *Beacon 3*: contiene i dati relative all'OBDH;
- *Beacon 4*: Contiene i dati relativi a sensori e attuatori dell'AOCS;
- *Beacon 5*: Contiene i dati relativi a FDIR e al sottosistema TMTC;
- *Beacon 6*: contiene i dati relativi al Payload (dati di telemetria e HK).

HK data pages

Le pages rispetto ai beacon vengono trasmesse solo su richiesta con un esplicito telecomando, e contengono tutti i dati di telemetria disponibili riguardo a un determinato sottosistema o payload. Le pagine sono numerate da 1 a 28 e le informazioni specifiche sui dati contenuti sono reperibili in bibliografia. Il primo byte è utilizzato in ogni caso per l'identificazione della pagina.

1.4.3 Organizzazione dei dati ricevuti all'interno del database

L'archiviazione dei dati di telemetria avviene tramite un database MySQL, che appare molto pratico essendo organizzato secondo una struttura semplice e modulare e risultando altamente compatibile con una grande varietà di linguaggi di programmazione e applicazioni (come python, matlab, javascript, excel), ed in particolare molto performante per quanto riguarda l'accessibilità dei dati via web.

Un database MySQL viene generalmente organizzato in tabelle. Nel caso di ESEO le tabelle generate sono: *TC list*, *TM history*, *TM parameters*, *TC history*, *HK pages*, *Beacon*, *TTC schedule*, *TC queue* e *ACK*. Ogni tabella ha determinati campi per ogni campo contenuto al suo interno.

TC LIST

È una lista di tutti i possibili telecomandi che possono essere utilizzati nel mission control center (MCS). Ogni telecomando ha un suo prefisso che indica il sottosistema selezionato.

L'address identifica in modo univoco il telecomando da inviare mentre il type indica il tipo di comando inviato, ovvero se SET, GET o entrambi. Il comando SET assume il valore 0x01, mentre il comando GET assume il valore 0x10.

Field	prefix	address	type	description
Type	Char (3)	Var binary (2)	Binary (1)	Var char (100)

Tabella 13. Descrizione della TC list

Le tabelle nel MCS sono utilizzate in questo caso per visualizzare tutti i possibili comandi in ordine alfabetico, divisi per sottosistema o payload e per tipo. La query MySQL per estrarre le informazioni è indicata sotto come Query 1. La Query 2 mostra invece come ottenere il nome del particolare telecomando a partire dal suo indirizzo.

Query 1: *SELECT hex(address), description FROM tc_list WHERE description IS NOT NULL AND prefix='input' AND type=unhex('input') ORDER BY address*

Query 2: *SELECT prefix, description FROM tc_list WHERE address=unhex('input')*

TM HISTORY

La tabella *TM History* serve per registrare tutti i pacchetti di telemetria ricevuti dal satellite, per poterli visualizzare nel MCS. I pacchetti possono essere filtrati in base al loro sottosistema di riferimento o alla data di ricezione. La struttura dei vari pacchetti è indicata nella seguente tabella.

Field	Class	Info String	received at
Type	Binary (1)	Var binary (128)	timestamp

Tabella 14. Descrizione della TM history

Il primo campo è il campo Class, che indica il tipo di pacchetto ricevuto, contenuto nella Tabella 15. Il campo Info String contiene il campo Info estratto dal frame inviato. Il campo Received at contiene la data e il tempo nel quale il frame è stato ricevuto e salvato nel database, utilizzando il formato data UTC 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

Tabella 15. Campo *class* dei pacchetti

Questo tipo di tabelle è utilizzato nel MCS per immagazzinare e analizzare i pacchetti di telemetria. I dati possono essere filtrati ordinandoli in base al tempo, alla classe o all'ultima risposta ricevuta.

La data, la classe e l'info string sono visualizzati nel MCS UI (user interface), a partire dall'ultimo pacchetto ricevuto. I dati possono essere richiesti al database usando la query seguente:

Query 3: *SELECT received_at, hex(class), hex(info_string) FROM tm_histo WHERE class=unhex('input') AND received_at BETWEEN 'start time(YYYY-MM-DD HH:MM:SS)' AND now() ORDER BY received_at DESC LIMIT input*

La Query 4 consente invece di visualizzare il numero di pacchetti ricevuti.

Query 4: *SELECT count(*) FROM tm_histo*

TM PARAMETERS

È una tabella che contiene la definizione di tutti i dati di house keeping. È utilizzata nel MRT (mission report tool) per visualizzare i parametri di HK desiderati. I parametri che possono essere visualizzati sono riportati nella Tabella 16.

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

Tabella 16. Descrizione dei TM parameters

In particolare, tali parametri sono:

- *prefix*: come prima indica il prefisso del parametro per definire il payload o il sottosistema a cui si riferisce;
- *label*: breve nome del parametro di house keeping visualizzato nel MRT;
- *datatype*: tipo di dato del parametro (mostrato nella Tabella 17, indica se è integer, float etc. e la lunghezza);
- *description*: breve descrizione del parametro di house keeping;
- *default value*: indica il valore di default al quale è settato il parametro;
- *min_value/ max_value*: indica il minimo e massimo valore accettabili per quel parametro, e quindi il monitor range di visualizzazione;
- *scaling_divide* e *scaling_add*: indica il valore da aggiungere al parametro o il valore per il quale il parametro deve essere diviso per ottenere il valore corretto e calibrato dello stesso;
- *units*: unità di misura del parametro (ad esempio gradi, volt, ampere).

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

Tabella 17. HK Data type

page	label	data_type	description	default_value	min_value
2	OBD_TEMP1	I16	Temperature sensor 1	20	-10
max_value		**scaling_divide**	**scaling_add**	**units**	
70		10	NULL	degC	

Tabella 18. Esempio di TM parameter

Nella Tabella 18 si mostra un esempio dei valori della temperatura *dell'OBDH Main*.

La Query 5 mostra come possono essere selezionati questi dati.

Query 5: *SELECT page, label FROM tm_parameters WHERE page='X'*

TC HISTORY

Contiene la cronologia di tutti i telecomandi trasmessi al satellite. La lista viene visualizzata nel MCS UI ed è possibile filtrarla in base alla data o al sottosistema ai quali ci si riferisce. Nella tabella sottostante sono descritti i vari campi disponibili per ogni parametro.

Field	address	prefix	description	info string	sent at	status
Type	Var binary (2)	Char (3)	Var char (100)	Var binary (15)	timestamp	enum

Tabella 19. Descrizione della TC history

- Il campo *address* identifica in maniera univoca il telecomando da inviare;
- il campo *prefisso* contiene informazioni riguardo il sottosistema selezionato;
- il campo *description* contiene una breve descrizione del TC da inviare;

- il campo *info string* contiene le informazioni relative al TC, in base esadecimale;
- il campo *sent at* contiene la data alla quale il TC è stato inviato;
- il campo *status* può contenere uno dei valori riportati in Tabella 20;

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

Tabella 20. Descrizione del TC status

Un esempio delle variabili memorizzate è riportato in Tabella 21.

Prefix	Description	Hex(Info string)	Sent at	Status
OBD	HK AND TC MANAGEMENT	0007000F0181000000	2014-10-07 15:50:38	NAK

Tabella 21. Esempio di parametro TC history

Le Query 6 e 7 mostrano rispettivamente come selezionare i parametri con vari campi ordinandoli in base alla data di invio decrescente, e come contare il totale dei telecomandi inviati.

Query 6: *SELECT sent_at, address, description, hex(info_string), status FROM tc_histo WHERE prefix='input' AND sent_at BETWEEN 'start time(YYYY-MM-DD HH:MM:SS)' AND now() ORDER BY sent_at DESC*

Query 7: *SELECT count(*) FROM tc_histo*

HK PAGES

I valori di telemetria delle classi HK pages, HK history, PS high rate history e beacon sono immagazzinati insieme in tabelle suddivise in base al sottosistema o payload al quale appartengono. Si è preferito un approccio di memorizzazione tradizionale, rispetto a quello per parole chiave. Pertanto, tutti i caratteri sono lowercase, e agli spazi sono stati sostituiti degli underscore. Le ultime due colonne di ogni tabella devono contenere il timestamp e l'origine. La prima è uguale al tempo di ricezione per le pagine di house keeping e per i beacons. La seconda invece indica se il dato corrispondente proviene dall'house keeping history, dalla pagina dei dati o dai beacons.

Ogni parametro è salvato come variabile grezza senza calibrazione, in quanto questo è un processo estremamente veloce che viene realizzato nel momento in cui l'utente richieda il dato.

Alcune tabelle non riescono a contenere tutti i dati relativi a un sottosistema. In tal caso i dati di telemetria vengono suddivisi sue due pagine consecutive. Tutti i sottosistemi, compresi quelli divisi su più pagine, sono indicati in Tabella 22.

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

Tabella 22. Corrispondenza tra la strumentazione e le pages

Questi dati sono mostrati nel MRT tramite una query analoga alla seguente:

Query 8: *SELECT parameter, t FROM obd WHERE t BETWEEN 'YYYY-mm-dd HH:MM:SS' AND 'YYYY-mm-dd HH:MM:SS' ORDER BY t DESC*

Beacons

I beacons sono strutture di dati molto simili a quelle definite per le pages dei dati di house keeping. Esistono sei classi di beacons, che sono già state descritte precedentemente.

Time-tagged schedule

Una copia della programmazione a bordo del satellite è mantenuta a terra e visualizzata nel MCS UI. I messaggi di acknowledge del TTC sono salvati in una tabella chiamata *ttc_schedule*, e sono descritti nella tabella sottostante.

Field	number	execution_time	address	value	status
Type	Tiny int unsigned	timestamp	Binary (2)	Binary (4)	Enum

Tabella 23. Descrizione delle time-tagged schedule

I vari campi relative a questa tipologia di dati sono:

- *number*: identifica il numero di colonna nel *ttc_schedule*;
- *execution_time* identifica il momento in cui il comando è eseguito e viene ripulito se l'esecuzione del comando non è più pendente;
- *address* identifica il telecomando che viene inviato;
- *value* contiene il valore da settare nel controllore, in formato esadecimale;
- *status* indica lo stato del comando, e può essere pending, disabled o cleared.

Un esempio di colonna del *ttc_schedule* è riportata in Tabella 24.

number	execution_time	hex(address)	hex(value)	status
2	2015-01-17 12:32:23	012A	43193A41	pending

Tabella 24. Esempio di time-tagged schedule

Per visualizzare a terra la copia del comando programmato a bordo si può usare la Query 9.

Query 9*: SELECT number, execution_time, hex(address), hex(value), status FROM ttc_schedule WHERE status='pending' OR status='disabled' ORDER BY execution_time*

Si può lavorare sui telecomandi programmati nell'OBDH con i seguenti comandi:

- ADD: *UPDATE ttc_schedule SET execution_time = 'input (YYY-mm-dd HH:MM:SS)', address = unhex('input'), value = unhex('input'), status = 'pending' WHERE status = 'cleared' LIMIT 1*

- DISABLE: *UPDATE ttc_schedule SET status='disabled' WHERE number = input*
- CLEAR ALL: *UPDATE ttc_schedule SET status = 'cleared'*

TC QUEUE

Durante le operazioni offline, i telecomandi possono essere salvati in una coda (queue) che viene caricata solo quando la connessione è nuovamente disponibile, in modo da velocizzare le operazioni. Questa coda è chiamata *tc_queue*, e i suoi parametri sono descritti nella Tabella 25.

Field	seq	address	type	description	info	status
Type	Medium int	Var binary (2)	Binary (2)	Var char (100)	Var binary (15)	enum

Tabella 25. Descrizione della tabella TC queue

- Il campo *seq* indica la posizione del TC nella coda;
- Il campo *type* può essere SET, GET o entrambi;
- Il campo *status* può assumere i valori indicati nella tabella 14, più il valore RDY.

Acknowledge messages

Sia GET che SET sono seguiti da un messaggio di acknowledge da parte dello spacecraft. Il valore registrato è salvato dentro la tabella *ack* insieme all'address del TC inviato. La descrizione di questa tabella è riportata nella table 23.

Field	register	address	updated_at
Type	Binary (4)	Binary (2)	Timestamp

Tabella 26. Descrizione della tabella dei messaggi di Acknowledge

- La Colonna *register* contiene il valore corrente del messaggio;
- La colonna *updated_at* è riempita con l'istante di ricezione del messaggio.

Questa tabella è utilizzata dal software che gestisce le operazioni automatiche di volo per velocizzare le operazioni.

2. Implementazione delle pagine di OBDH HK e TC su Grafana

2.1 L’ambiente Grafana

Grafana è una piattaforma web che consente la visualizzazione di dati provenienti da un database. È compatibile con moltissimi software e database, come ad esempio il database MySQL su cui sono salvati i dati di telemetria di ESEO. L’utilità di Grafana sta nella possibilità di organizzare i dati provenienti dal database tramite un’implementazione grafica molto efficace e comprensibile, al fine di consentire l’effettuazione di analisi e valutazioni sui dati visualizzati. Inoltre, la piattaforma Grafana è pensata per l’utilizzo su server e pagine web, consentendo quindi l’accesso ai dati da qualsiasi luogo e in qualsiasi momento.

La struttura di Grafana è basata su diverse tipologie di pannelli (panel), che consentono la visualizzazione dei dati organizzandoli in maniere differenti, ad esempio tramite grafici o tabelle. Tali pannelli sono basati sul linguaggio web Javascript e assicurano un’elevata compatibilità degli stessi con tutti i principali browser web grazie a una struttura basata sul salvataggio dei parametri dei pannelli in formato json. Oltre ai pannelli per la visualizzazione dei dati da database, sono disponibili pannelli per implementare ad esempio degli *iframe* da siti esterni.

L’interfaccia di lavoro è variabile per ogni tipo di pannello disponibile su Grafana; inoltre essa risulta programmabile per adattarla alle esigenze dell’utilizzatore, consentendo di apportare modifiche per il trattamento dei dati, ad esempio inserendo le unità di misura, scalando e calibrando i dati o ordinandoli in maniera differente. Infine, la dashboard di Grafana fornisce una serie di opzioni grafiche per personalizzare e adattare la resa grafica dei pannelli.

Una volta composto un pannello, esso è facilmente inseribile all’interno della pagina trascinandolo con il cursore del mouse per sceglierne la posizione. Allo stesso modo, con il solo utilizzo del mouse è possibile modificare le dimensioni dei pannelli, che sono rettangolari, in modo da adattare i contenuti alla pagina.

2.2 Creazione dei pannelli

Il lavoro di tesi su Grafana è consistito nell'implementazione dei pannelli per la visualizzazione di tutte le pages di house keeping di ESEO, salvate nel database MySQL secondo la struttura descritta nel capitolo precedente. Il sito per la visualizzazione dei dati del satellite era già stato implementato precedentemente, all'indirizzo *eseo.ddns.net* e vi erano stati caricati sopra tutti i parametri relativi ai beacons, pertanto il lavoro svolto per questa tesi si è concentrato sulla pubblicazione delle sole pages, contenenti tutti i parametri disponibili sul satellite. La suddivisione dei sottosistemi e dei payload nelle varie pagine è già stata descritta in Tabella 22.

Per creare i panel, occorre innanzitutto creare una dashboard, ovvero una pagina, all'interno della quale inserire i pannelli desiderati. I pannelli vengono quindi creati cliccando il pulsante *Add panel* e scegliendo la tipologia di pannello desiderata. Una volta creata il pannello bisogna scegliere tra le opzioni disponibili *Edit*, che apre un riquadro con tutte le opzioni disponibili divise in sezioni, tra cui quelle per l'estrazione della query e quelle per la resa grafica.

Tutti i pannelli hanno nella sezione *general* un'opzione per definire il titolo e la descrizione, nonché un'opzione grafica per rendere lo sfondo trasparente o bianco, come mostrato in Figura 5.

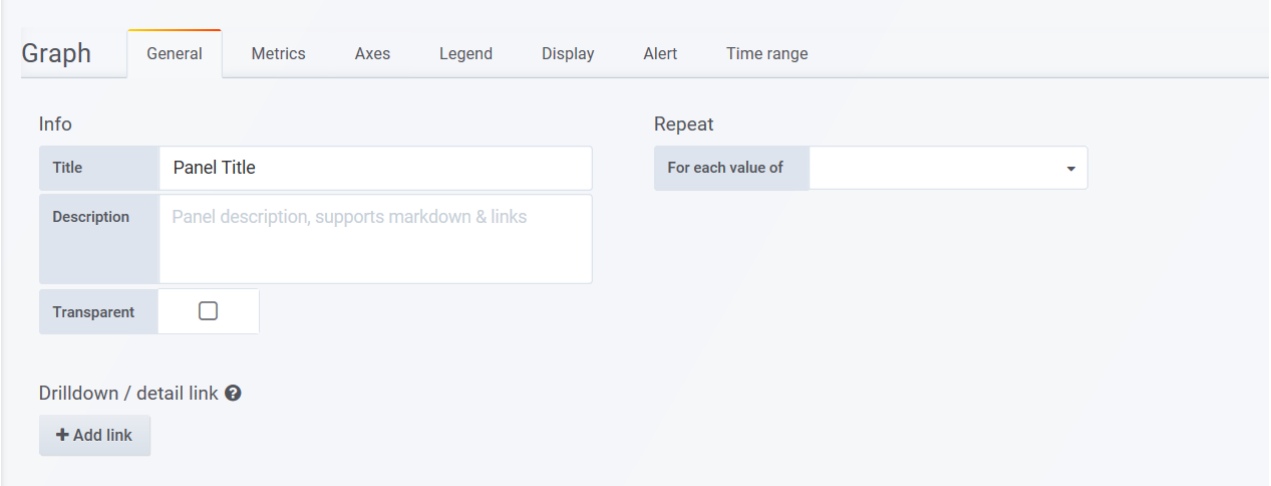

Figura 5. Finestra general dei pannelli

Per quanto riguarda l'accesso e l'estrazione dei dati dal database MySQL, questo avviene tramite una query, ovvero una chiamata al database, al quale si richiedono i

dati. Esempi di query sono riportati nel paragrafo 1.4. La query viene impostata interamente nella sezione *Metrics* del riquadro di impostazione del pannello, come mostrato in Figura 6. La struttura della query è analoga per qualunque utilizzo, in quanto essa rappresenta la stringa di cui il database necessita per sapere quali dati fornire, mentre cambia a seconda del linguaggio utilizzato il modo in cui si effettua la connessione al database e si invia la query allo stesso.

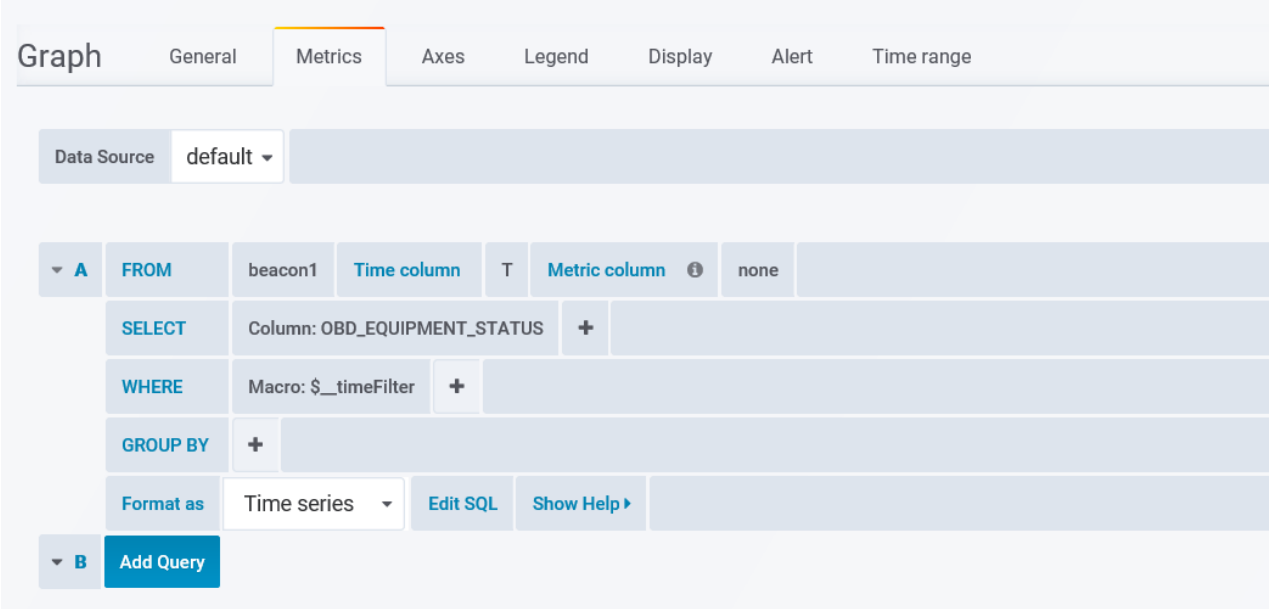

Figura 6. Finestra Metrics del pannello

L'interfaccia di Grafana è programmata per consentire un accesso facilitato ai dati, nel caso si voglia solamente estrarli e riportarli così come sono, fornendo già una struttura con menu a tendina dalla quale è possibile assemblare la query selezionando il database di riferimento, la tabella del database del quale si vogliono estrarre i dati e la colonna corrispondente al parametro che si vuole visualizzare. Vi è poi un'opzione per organizzare i dati in ordine cronologico, utilizzata per la visualizzazione di tutti i dati di telemetria di ESEO. Occorre infine sottolineare come sia possibile inserire la visualizzazione di più parametri all'interno di un unico pannello, ad esempio per confrontare diverse temperature afferenti allo stesso sottosistema o per visualizzare lo stato di tutti i componenti di un payload.

Nel caso si vogliano invece apportare modifiche ai dati, ad esempio rinominando un parametro, estraendo i singoli bit di una stringa di errore o scalando e calibrando una variabile, occorre scrivere direttamente la query indicando esattamente al database

quale dato si richiede. Occorre sottolineare che tutte queste operazioni vengono richieste e quindi sono direttamente eseguite dal software del database e non da Grafana, che si occupa solamente della resa grafica dei dati ricevuti. La scrittura personalizzata della query è resa disponibile tramite il pulsante *Edit SQL*, che apre un editor nel quale può essere scritta la query che sarà poi inviata al database per la richiesta dei dati.

I pannelli utilizzati principalmente in questo lavoro sono stati il pannello *Graph*, il pannello *Single Stat* e il pannello *Discrete*.

Graph

Il pannello *graph* consente la visualizzazione di grafici, per un singolo parametro o per più parametri sovrapposti. Il grafico mostra l'andamento storico dei parametri, con una funzione discreta per punti, ed è pertanto utilizzato per tutti quei parametri, come le tensioni e le temperature, che assumono un ampio range di valori, e di cui è rilevante sapere l'andamento in base ai cicli di funzionamento e alle condizioni esterne. La realizzazione dei grafici richiede spesso l'utilizzo dell'editor MySQL, in quanto i dati necessitano di calibrazione, che deve essere effettuata direttamente nella query a MySQL. Inoltre, per i parametri da inserire nei grafici risulta spesso utile salvarli con un nome differente rispetto a quello codificato nel database, in quanto questo nome è lo stesso che viene utilizzato per creare la leggenda del grafico. In Figura 7 si mostra come sia possibile estrarre una variabile scalandola e modificandone il nome. In questo caso si visualizza la creazione del grafico per il parametro ESE_TAU_TEMP, corrispondente alla temperatura interna alla camera dell'Earth Sensor. Il valore viene scalato a un decimo di quello salvato nel database per ottenere la temperatura vera; inoltre, la variabile è rinominata per rendere più immediata la comprensione.

Nella sezione *Axes* si possono invece impostare gli assi del grafico, e in particolare settare l'asse delle ordinate nell'unità di misura desiderata (ad esempio °C, V, A). L'asse x è sempre settato di default con l'asse dei tempi. La finestra consente anche di impostare un asse y a destra, oltre a quello a sinistra, tuttavia in questo lavoro tale asse non è mai stato utilizzato. Nella Figura 8 è mostrata la finestra di impostazione degli assi.

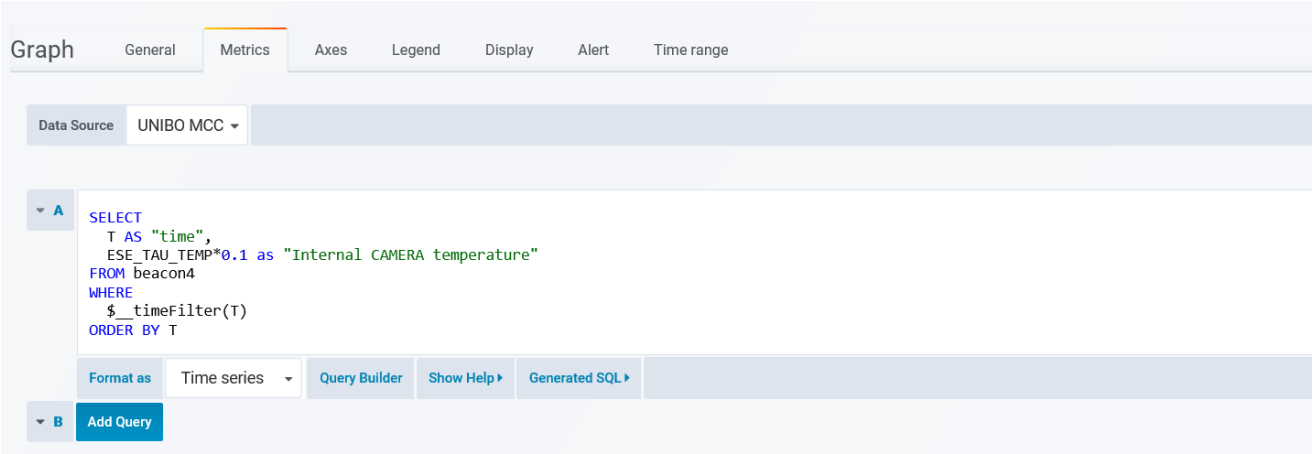

Figura 7. Rinominazione e scalatura di un parametro

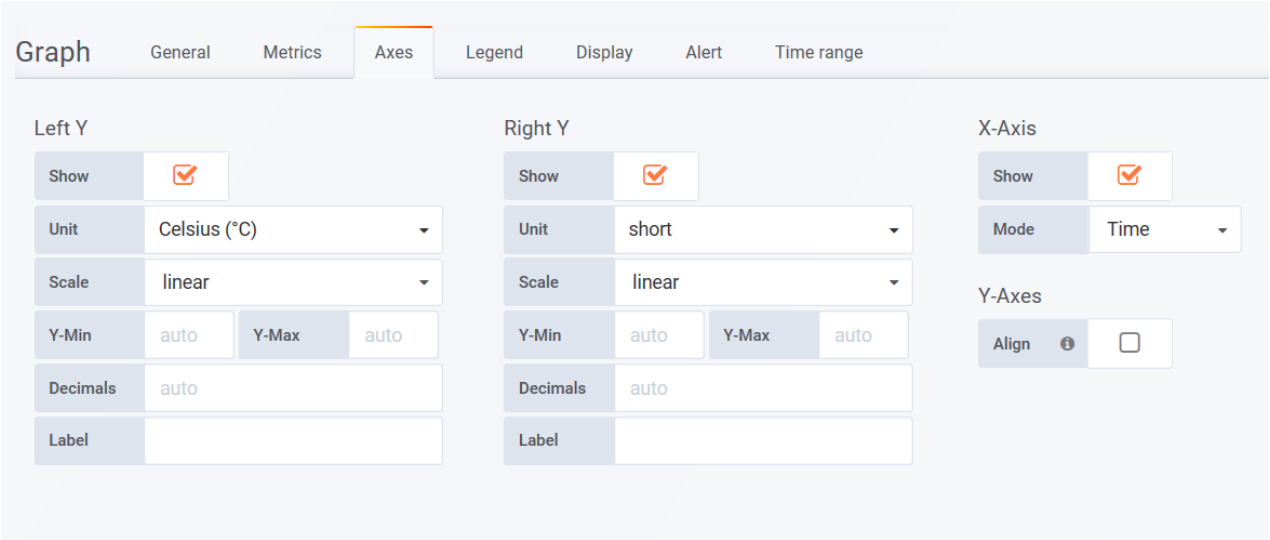

Figura 8. Finestra di impostazione degli assi

Nella sezione *Legend* si può impostare la leggenda del grafico, che associa il colore del grafico al nome delle variabili corrispondenti. Questo risulta particolarmente utile nel caso il grafico venga utilizzato per rappresentare contemporaneamente l'insieme di più parametri. La visualizzazione della leggenda può in ogni caso essere disattivata.

La sezione *display* consente invece di impostare le opzioni grafiche del pannello, ovvero la dimensione dei punti del grafico, lo spessore e il colore delle linee,

l'opacità dell'area sottesa al grafico e se visualizzare le linee o i punti del grafico. Il layout della sezione è riportato in Figura 9.

Sono presenti anche altre due finestre di impostazione, *Alert* e *Time range*, che però non sono state utilizzate per la visualizzazione della telemetria di missione.

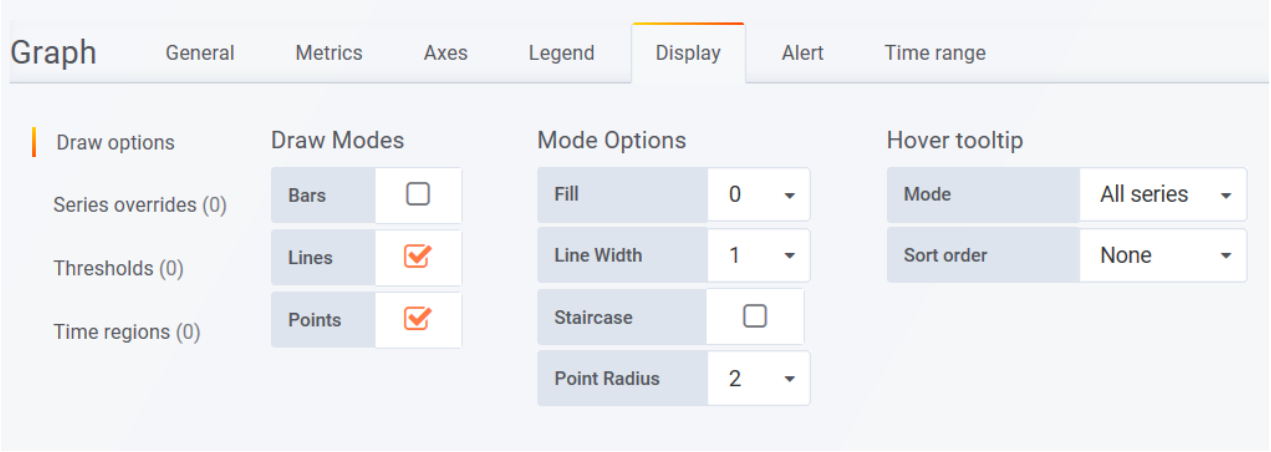

Figura 9. Finestra di personalizzazione del grafico

Single Stat

Il pannello *single stat* è utilizzato per visualizzare singoli dati numerici, riferentesi ad esempio alla data interna del sistema o al numero di reset effettuati per un certo sottosistema. Tale pannello presenta sempre le due sezioni *General* e *Metrics*, che hanno le stesse opzioni del panel *Graph*.

Peculiare del pannello *Single stat* è invece la sezione *Options*, attraverso la quale è possibile impostare l'unita di misura del parametro visualizzato (*unit*) nonché quale valore visualizzare tra quelli contenuti nel database per quel parametro (*stat*). Le opzioni si dividono in *current*, che visualizza l'ultimo valore del parametro disponibile nel database, *max* e *min*, che mostrano il valore più grande e più piccolo rispettivamente, *total*, che mostra la somma di tutti i valori, *average*, che mostra la media di tutti i valori. È inoltre disponibile un'opzione per colorare le varie parte del pannello, che però non è stata utilizzata nell'implementazione dei dati di ESEO. Nella sezione è infine disponibile un'opzione per impostare la dimensione del testo visualizzato.

L'ultima sezione di interesse è *Value Mappings*, che consente di mappare tutti o parte dei valori disponibili per quel parametro, assegnando ad un determinato valore un corrispondente valore o flag da mostrare nel pannello (ad esempio per un parametro di errore si può impostare per il valore "1" il flag *ERR* e per il valore "0" il flag "OK").

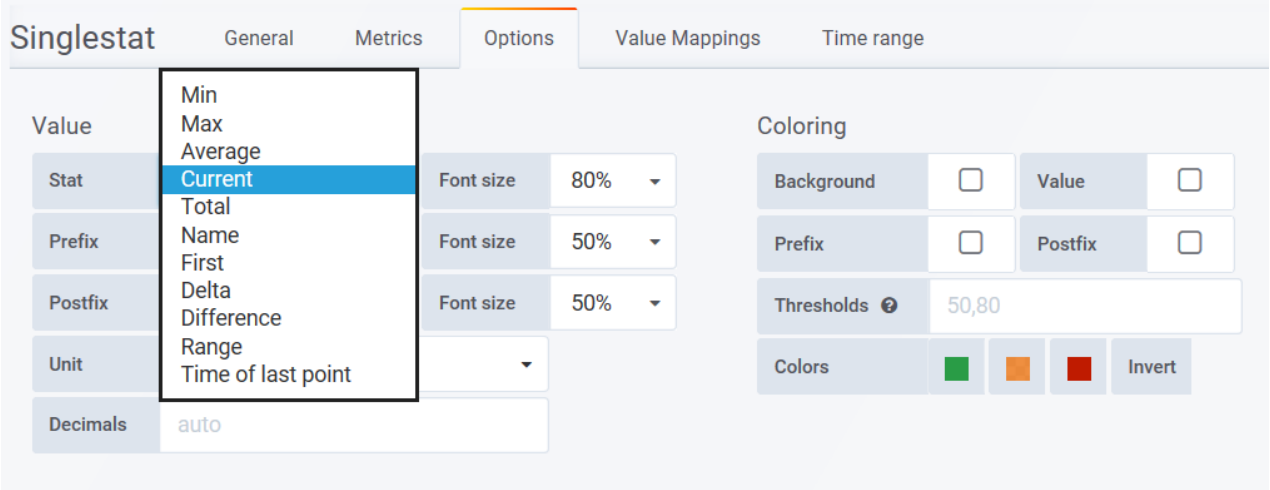

Figura 10. Finestra Options del panel Single stat

Discrete

L'ultima tipologia di panel utilizzata per la visualizzazione dei dati di telemetria è *discrete*. Questo è un pannello che consente di visualizzare i dati come righe di una tabella, mostrando per ogni riga tutti i valori assunti dal parametro nell'intervallo di tempo considerato. I panels *discrete* possono essere utilizzati o per visualizzare singoli parametri, o più spesso per visualizzare lo stato di tutti una serie di sistemi riferiti a uno stesso parametro, ad esempio una stringa di bit di errore, i cui singoli bit rappresentano lo stato di uno dei sistemi considerati o una diversa tipologia di errore che si può verificare. In tal caso occorre una query dedicata, che consenta di estrarre i singoli bit della stringa contenuta nel database. La query viene implementata direttamente nell'editor di Grafana, estraendo i singoli bit a partire dal primo della stringa e rinominandoli tutti in base al flag a cui corrispondono. Le stringhe come visto nel capitolo precedente possono essere di 8, 16 o 32 bit. I parametri da dividere nei singoli bit sono generalmente di tipo *unsigned*. Una tipica

stringa per estrarre un bit da un parametro è la seguente, riferita al parametro PMM_PDU_LCL_CONTROL:

CONVERT(SUBSTRING(LPAD(BIN(PMM_PDU_LCL_CONTROL),32,'0'),32,1), UNSIGNED INTEGER) as 'LCL_1-PDU_1-LCL STATE'.

Tale stringa è inserita all'interno della query al database, ed estrae il primo bit (il 32 in una numerazione decrescente da 32 a 1), da una stringa di 32 bit, in cui la lettura viene effettuata a partire da sinistra (LPAD). È possibile estrarre tutti i bit della stringa in un'unica query, semplicemente utilizzando una stringa come quella sopra per ogni bit e separando le stringhe con una virgola. Grafana è quindi in grado di riconoscere tutti i valori estratti creando una riga della tabella *discrete* per ognuno.

Alcuni parametri possono richiedere un'operazione più complessa, in quanto possono contenere dei flag associati a un sottoinsieme dei bit che compongono la stringa del parametro. In questo caso la stringa per l'estrazione richiede di estrarre i singoli bit e di sommarli moltiplicati per una potenza di 2 corrispondente alla loro posizione nella sottostringa, così da ricomporre un numero in base decimale che assume più di due valori a seconda del valore assunto dai bit che lo compongono. Una tipica stringa di questo genere può essere composta da 3 bit, fornendo 8 possibili valori, o da 8 bit, per un totale di 256 valori ammissibili. Non sempre vengono utilizzate tutte le combinazioni di bit possibili, ma le combinazioni sono ridondanti per poter verificare la presenza di errori.

Il panel *discrete* fornisce in aggiunta alcune sezioni specifiche per il settaggio delle opzioni. La sezione *Options* consente di configurare tutte le opzioni grafiche del pannello, ovvero lo spessore delle righe della tabella, la dimensione del font, se scrivere i nomi delle variabili, se mostrare l'asse del tempo o l'ultima variabile memorizzata e se mostrare il numero per ogni valore assunto.

La sezione *Legend* è analoga a quella del pannello *graph*, con la differenza che in questo caso la tabella mostra una singola voce della leggenda per ogni valore presente nella tabella e non per ogni parametro.

Come nel *single stat*, anche qui è presente la sezione *Mappings*, che consente di effettuare esattamente le stesse operazioni. In questo caso tuttavia la finestra risulta molto più importante, perché, nella visualizzazione dello stato dei parametri di un

sottosistema, la risposta in bit 0 e 1 deve sempre essere riconvertita in un messaggio leggibile per l'operatore. Nel caso in cui i diversi bit debbano dare come risposta messaggi differenti, dal momento che i valori ottenuti sono solo 0 e 1, occorre scalare in maniera differente tutti i bit ricevuti dal database, ad esempio con un offset 100, 200, 300… per ogni differente bit, in maniera tale da poter mappare ogni bit con un messaggio personalizzato.

2.3 Visualizzazione dei dati

Come già esplicato nei paragrafi precedenti, questo lavoro di tesi è stato incentrato sulla visualizzazione grafica di tutti i parametri contenuti nelle pages di telemetria del satellite, scegliendo opportunamente il migliore metodo di rappresentazione di ogni parametro e occupandosi di visualizzare nei pannelli discrete tutti i dati relativi ai singoli bit di stato dei sottosistemi.

La visualizzazione dei dati sul sito è stata eseguita cercando di rispettare quanto più possibile l'organizzazione stabilita nel manuale di descrizione dell'OBDH HK utilizzato nel centro di controllo missione a Forlì. Pertanto, i parametri sono stati suddivisi nelle 28 pages secondo le quali è organizzata la telemetria di ESEO nel protocollo di telecomunicazione del satellite, e che sono salvate con la stessa organizzazione all'interno del database. All'interno di ogni pagina i pannelli sono stati organizzati in maniera sequenziale e in fila per due, seguendo lo stesso ordine con cui i relativi parametri sono elencati nelle tabelle dei manuali. Un esempio per quanto riguarda la page1 è riportato in Figura 11.

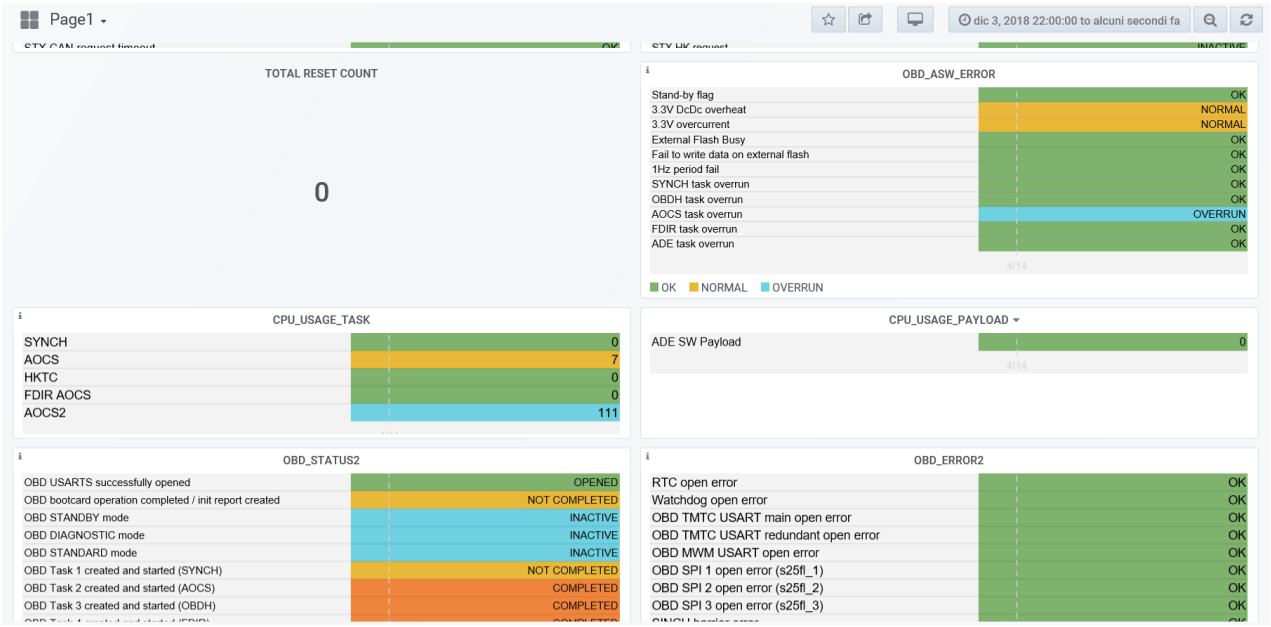

Figura 11. Spezzone della page 1

Molti sottosistemi sono completamente ridondati, e pertanto vi è una page dedicata ai parametri del sottosistema main e una dedicata ai sottosistemi redundant. In questo caso le pages sono state unificate su Grafana, disponendo i pannelli su due colonne affiancate, in maniera tale che i dati del sistema main e redundant siano sempre confrontabili immediatamente.

Per quanto riguarda il range di dati visualizzato, esso non è dato da un'impostazione specifica, ma è regolabile da qualsiasi utente che visiti il sito attraverso il banner in alto a destra, nel quale è visualizzato il range selezionato attualmente. Cliccando sulla barra si possono selezionare vari range temporali preimpostati, oppure specificare direttamente la data di inizio e di fine del range desiderato.

Per quanto riguarda la diversa tipologia di pannelli utilizzati, sono state fatte delle scelte grafiche precise per ognuno di essi, al fine di avere omogeneità tra tutti i pannelli per una migliore resa grafica. In particolare, per quanto riguarda il panel *graph* si è scelto di utilizzare linee di spessore unitario, di visualizzare punti di raggio 2 pixel e di lasciare trasparente l'area sottesa al grafico, in quanto nel caso dei dati di ESEO essa non aveva alcun significato fisico. Per quanto concerne invece i pannelli discrete si è scelto di usare righe alte 15 pixel con un font di 12 pixel. Per i grafici e le tabelle non sono stati scelti dei colori specifici, lasciando che fosse il software di Grafana ad impostare i colori.

L’aspetto assunto dai tra tipi di pannelli è riportato nelle figure seguenti.

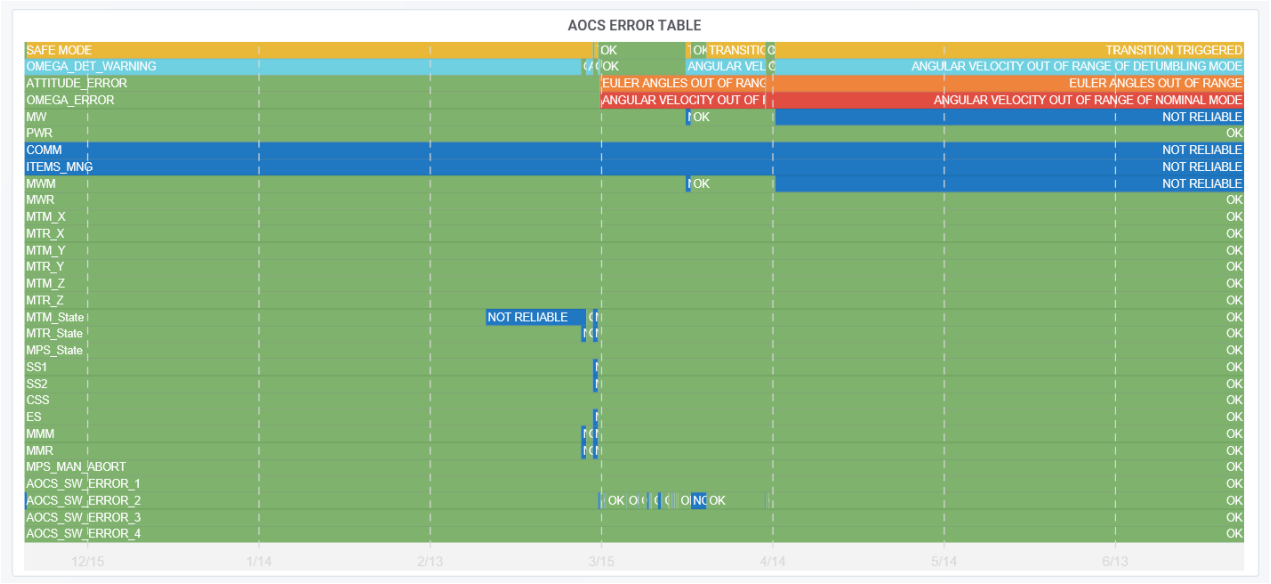

Figura 12. Tabella di errore di AOCS

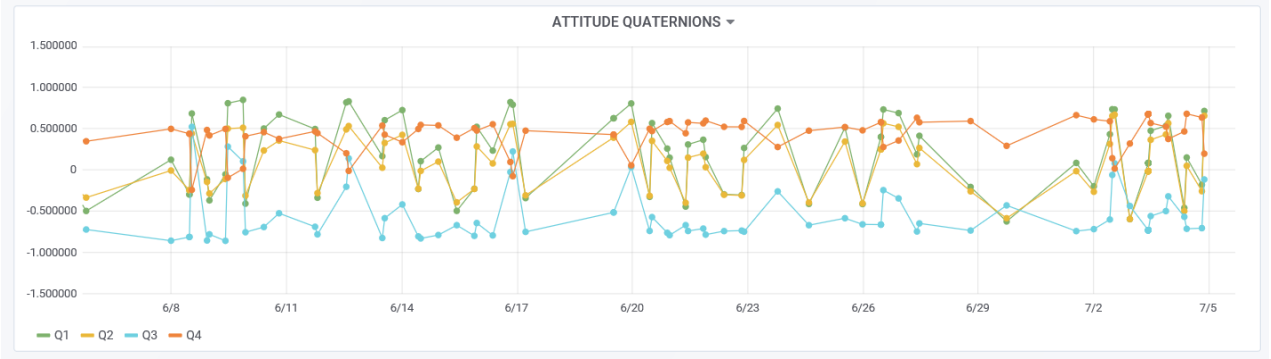

Figura 13. Grafico con la visualizzazione dei quaternioni

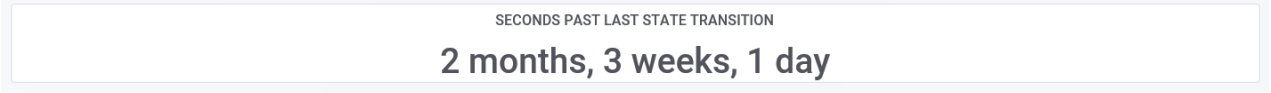

Figura 14. Single stat dell'ultimo cambio di stato dell'OBDH

Si osserva dalle figure come Grafana scelga automaticamente per le tabelle un colore per ogni stato disponibile, mostrando quindi anche lo storico nel range temporale selezionato di tutti gli stati assunti dal parametro. Sotto la tabella è presente una barra temporale per riferire ogni stato a un’epoca. Inoltre, scorrendo col cursore sopra la tabella, viene visualizzato un banner che mostra lo stato e il range temporale dello spezzone sopra al quale è posizionato il cursore. In tal modo risulta immediata la visualizzazione dei cambi di stato di un sottosistema, di un

sensore, di un parametro, nonché del momento in cui ad esempio si è riscontrato un errore all'interno di un dispositivo. Queste tabelle sono state utilizzate per visualizzare molti tipi differenti di parametri, ad esempio la modalità operativa nella quale si trova l'OBDH, se il satellite si trova in eclisse o in daylight, il corretto funzionamento dei vari sottosistemi o se essi sono accesi o spenti, tutti i possibili messaggi di errore per un sottosistema, la disponibilità o meno dei canali di memorizzazione di un telecomando programmato e lo stato di ogni telecomando programmato (eseguito, pendente, nessun comando programmato).

I grafici invece presentano una minore varietà di opzioni. Essi infatti rappresentano principalmente dati di cinque tipi, ovvero tensione, intensità di corrente e temperatura, ottenuti dai rispettivi sensori presenti in tutti i sottosistemi e lungo tutte le linee di distribuzione di potenza, e velocità e posizioni angolari, misurati in gradi o in rivoluzioni per minuto. I dati sono stati raggruppati nei grafici secondo due criteri principalmente: sono stati raggruppati i parametri che riguardano lo stesso dato di sottosistemi afferenti, come ad esempio le velocità angolari del satellite calcolate con i 4 diversi algoritmi dell'ADE, oppure sono stati raggruppati differenti misurazioni dello stesso parametro all'interno di uno stesso sottosistema, come ad esempio le temperature in diversi punti del sottosistema di micropropulsione. I vari dati mostrati all'interno di uno stesso grafico sono rappresentati con colori differenti e indicati nella leggenda con il nome che gli è stato assegnato al momento della richiesta al database MySQL.

Per quanto riguarda i pannelli *single stat*, l'implementazione è risultata più semplice rispetto a quella delle altre tipologie di pannelli. Infatti, all'infuori di pochi casi, il single stat è stato utilizzato per rappresentare un singolo valore numerico. Il motivo per cui si è preferito usare questo pannello a un pannello di tipo *discrete* composto da una singola riga, è che i dati visualizzati tramite single stat sono per lo più dati progressivi, destinati a crescere nel tempo e ad essere resettati, in quanto misurano ad esempio il numero di secondi passato dall'ultimo cambio di stato. Per tali dati risulta quindi funzionale mostrare un unico valore, avendo l'accortezza di impostare il pannello affinché visualizzi unicamente l'ultimo stato disponibile, ovvero lo stato *current*, e non la somma o la media di tutti i valori del parametro contenuti nel database.

3. Sviluppo di un visualizzatore web 3D di dati orbitali

3.1 Il contesto di sviluppo del visualizzatore 3D

Nel capitolo precedente sono stati descritti i pannelli utilizzati per visualizzare tutti i dati di telemetria di ESEO. Grafana offre però molte tipologie di pannelli, tra cui un pannello per l'inserimento di *iframe* da siti esterni, che risulta molto utile per inserire sul sito app di visualizzazione di altri dati del satellite, come ad esempio la sua posizione orbitale attuale e futura. Sul sito di ESEO era già stata implementata precedentemente un'app per la visualizzazione 2D dell'orbita del satellite in tempo reale. Tuttavia, questa app risulta molto limitata da un punto di visto grafico, e essendo fornita da un sito esterno risulta scarsamente personalizzabile. Per questo motivo si è deciso di sviluppare una nuova applicazione in proprio, usando un'interfaccia grafica tridimensionale e ad alta definizione. Le caratteristiche di questa applicazione sono l'utilizzo di un globo terrestre tridimensionale, ad alta definizione, basato sulle foto satellitari delle mappe Bing; l'utilizzo di un modello 3D del satellite a colori; la possibilità di rappresentare l'assetto posseduto dal satellite in un certo istante, la possibilità di visualizzare graficamente gli accessi del satellite sulla stazione di terra. La realizzazione di tale applicazione è stata oggetto della seconda parte di questo lavoro di tesi, rappresentando la parte principale del lavoro stesso.

L'applicazione per la visualizzazione del globo si basa sull'applicazione CesiumJS, un pacchetto realizzato dall'azienda AGI (Analiyical Graphics, Inc.), fornitrice tra l'altro del software STK, costituito da una libreria Javascript e da varie altre infrastrutture di supporto, tra cui la più importante risulta essere un server basato sull'applicazione Node.js. Attraverso Cesium è stato possibile implementare il visualizzatore, sfruttando tra l'altro le possibilità offerte dalla piattaforma di runtime Node.js, che sarà descritta nel paragrafo successivo, e implementando delle procedure per ricavare dati quali i quaternioni e i TLE rispettivamente dal database con la telemetria di ESEO e dal sito celestrak.com, nel quale sono salvati tutti i dati forniti dal NORAD riguardo alla posizione orbitale del satellite.

Nei paragrafi successivi si descriverà inizialmente Node.js, soffermandosi sulle caratteristiche di un server basato su Node e mostrando come questo è stato adattato

per gli scopi di questo lavoro. Inoltre, si descriverà come è stato reso possibile utilizzare il server per ottenere i quaternioni e i TLE. Quindi si descriverà CesiumJS, che costituisce il nucleo centrale del lavoro, essendo basato su di esso il visualizzatore 3D. Si descriveranno in particolare i passaggi effettuati per la creazione del visualizzatore, che hanno seguito tale ordine: innanzitutto si è creato lo scenario, quindi si è effettuata la richiesta al server per ottenere i *quaternioni* e i TLE, che sono stati utilizzati per eseguire un propagatore sgp4, attraverso il quale è stato possibile ricreare un set di punti dell'orbita, interpolandoli per ottenere l'orbita completa; in questo caso è stato necessario impostare il visualizzatore per far sì che la posizione del satellite in funzione del tempo corrispondesse a quella reale, in modo che il visualizzatore fosse real time. Si è quindi proceduto a creare l'entità ESEO, impostata per muoversi all'interno dell'orbita con un assetto impostato attraverso i quaternioni. Si è quindi proceduto a creare una funzione per segnalare il passaggio del satellite sulle stazioni di Terra, e infine si è creato un secondo script incentrato sul satellite per evidenziarne l'assetto. Nell'ultima parte del capitolo si mostrerà in conclusione come le varie parti prodotte sono state assemblate e integrate su una pagina html, poi resa disponibile sul sito della telemetria tramite un *iframe* di html.

3.2 Node.js

3.2.1 Introduzione a Node.js

Node.js è una piattaforma molto recente utilizzata per scrivere applicazioni web, che negli ultimi anni ha raccolto molti consensi tra gli sviluppatori di tali applicazioni.

Prima di Node, per scrivere applicazioni web si utilizzava un processo standard: il server resta in ascolto su una certa porta e quando arriva una richiesta viene attivato un nuovo processo o un thread per rispondere alla query. La risposta alla query implica spesso di comunicare con servizi esterni, come database, cache di memoria o server di elaborazione. Quando il processamento finisce, il thread viene segnalato come disponibile, e si può gestire una nuova richiesta. Il processo così è lineare e semplice da implementare, tuttavia presenta due svantaggi. Il primo è che ogni processo implica un sovraccarico di memoria, soprattutto se deve utilizzare l'accoppiamento PHP + Apache. Il secondo svantaggio è rappresentato dalla

mancata ottimizzazione del tempo. Infatti, in una catena di questo tipo, la maggior parte del tempo è inutilizzata, in quanto viene spesa aspettando che i servizi esterni elaborino la risposta e restituiscano il risultato. In questi momenti il thread è effettivamente bloccato e tutti i processi e le risorse sono fermi. Pertanto, questo modello risulta inefficace anche se semplice, e si sono cercate nuove soluzioni per affrontare il problema.

Nato nel 2009, Node è nato ispirato da progetti simile realizzati su Ruby e Python, con la caratteristica fondamentale di essere realizzato intorno ad un modello non bloccante di programmazione degli eventi. In pratica, se l'applicazione web deve lanciare una query su un database o un'altra risorsa esterna, essa avvia l'esecuzione e informa Node su cosa farà quando sarà restituito il risultato. Nel frattempo, il codice è libero di iniziare l'elaborazione di altre richieste in ingresso, oppure di ogni altra operazione necessaria. In questo modo possono essere creati server che riescono a gestire centinaia di richieste simultanee anche senza grandi risorse di memoria.

Se i vantaggi della programmazione asincrona sono innumerevoli, occorre però sottolineare che Node lavora su un singolo thread, e sebbene questo sia un vantaggio nella gestione di richieste a molti server e database, può risultare problematico nel caso si debbano elaborare richieste molto lunghe e complesse. Inoltre, il modello di programmazione singola risulta piuttosto inusuale da utilizzare, e bisogna conoscerne bene le problematiche per riuscire a realizzare applicazioni robuste.

Riassumendo Node.js è quindi una piattaforma per la runtime (l'esecuzione) del codice Javascript, realizzata e ottimizzata per poter utilizzare Javascript lato server e non più solamente lato client, sfruttando le potenzialità dinamiche e correggendo le debolezze attraverso il funzionamento asincrono.

Per poter utilizzare Node, esso deve essere installato sul computer, assicurandosi che esso sia stato aggiunto alla variabile d'ambiente *path*, in modo che possa essere riconosciuto come comando interno in qualunque percorso del computer.

I file Javascript possono quindi essere eseguiti dal prompt dei comandi, digitando *node nome_programma.js* direttamente nel prompt all'indirizzo della cartella nel quale è contenuto lo script Javascript che si vuole eseguire.

I moduli in Node.js

Uno dei vantaggi di Node, è l'organizzazione delle funzionalità esterne e del codice realizzato da altri sviluppatori e reso disponibile online sotto forma di moduli. I moduli sono un sistema che raggruppa funzionalità simili in Node.js, ovvero dei pacchetti di librerie e script che assolvono ad una certa funzione. I moduli possono essere anche piuttosto complessi, in quanto composti da molti file, documentazione, cartelle e unità di test. È facile creare semplici moduli con Node, tuttavia molte delle migliori funzionalità di Node dipendono dall'utilizzo dei moduli già integrati nella piattaforma, ad esempio quelli utilizzati per creare un server web. Tali moduli possono essere installati attraverso *npm*, il Node package manager, che viene automaticamente installato insieme a Node e permette di installare in una sottocartella di lavoro tutti i moduli integrati in Node e ogni altro modulo sviluppato da altri utenti e reso disponibile sulla piattaforma online github.com, semplicemente digitando *npm install nome_modulo* nel prompt all'indirizzo desiderato. Per integrare un modulo in un file Node occorre utilizzare la funzione di Node *require("Modulo")*, assegnando il modulo a una determinata variabile.

I moduli risultano fondamentali per sfruttare al meglio Node per la realizzazione di un server, pertanto ne è stato fatto ampio uso anche nello svolgimento di questo lavoro di tesi. In particolare, si sono utilizzati i moduli *mysql*, per consentire la connessione con il database MySQL contenente i dati di telemetria di ESEO, il modulo *sgp4*, contenente un propagatore orbitale sgp4, il modulo *request-promise*, per la richiesta dei TLE al sito celestrak.com, e soprattutto il modulo *express*, per la creazione del server web, sul quale gira l'app Cesium, che rappresenta un elemento fondamentale di questo progetto, in quanto ad esso è legato il funzionamento di tutte le componenti del progetto.

3.2.2 Creazione del server web

Come appena accennato, il server web rappresenta il nucleo di qualunque applicazione web, e pertanto anche Cesium necessita dell'utilizzo di esso. Il server infatti deve ottemperare alle richieste avanzate dal client, fornendo i file e svolgendo le operazioni necessarie al client per poter visualizzare una determinata applicazione web. Tale applicazione è solitamente formata da molti file e assets, che devono

essere forniti al client perché esso possa utilizzare correttamente l'applicazione. Il server funziona quindi fornendo un url che può essere richiamato dal client tramite un browser web. L'url rimanda all'indirizzo del server e indicizza una precisa richiesta al server. Il server può quindi rispondere a tale richiesta fornendo tutti i file e gli elementi necessari, che possono essere file statici solamente da inviare o dati che devono prima essere processati o elaborati dal server.

Il server non si occupa solamente di fornire dati salvati nella macchina su cui gira, ma deve spesso essere in grado di recuperare informazioni da server e database esterni. In questo modo il client potrà accedere ai dati evitando di dover salvare tutte le informazioni a spese della propria memoria statica, mentre si dovrà fare carico dell'elaborazione dei file forniti dal server, che a tale scopo fornirà anche un file principale in grado di richiamare tutte le altre risorse fornite dal server per far funzionare l'applicazione. Il server si dovrà così occupare solamente di servire i file, che saranno poi assemblati ed eseguiti dal browser del client, e potrà gestire molte più richieste.

Per quanto riguarda i dati trasmessi, essi possono essere ad esempio script di javascript, pagine html, immagini, modelli. Nel caso invece che il server debba fornire dei semplici dati, ad esempio quelli provenienti da un database, si sceglie genericamente il formato dati *json* (javascript object notation). Tale formato è appositamente pensato per l'interscambio di dati tra applicazioni client – server ed è diventato molto popolare a causa della semplicità che lo caratterizza, che lo rende facilmente utilizzabile in molti linguaggi di programmazione. Inoltre, Javascript è in grado di interpretare direttamente i file json tramite un parser, e pertanto tali file sono diventati molto comuni in seguito alla crescente importanza di Javascript nello sviluppo delle applicazioni web.

Il server di Cesium

Dal momento che CesiumJS non è solamente una libreria Javascript, ma un'applicazione basata su tale libreria, con varie funzionalità aggiuntive e moltissimi assets aggiuntivi, accessibili in parte direttamente dalla macchina su cui opera il server, e in parte dal cloud Cesium ION, viene fornito anche lo script per un server preimpostato per le funzionalità di base, che contiene già il codice necessario per gestire tutte le risorse contenute nel pacchetto. Le richieste preimpostate a

questo server non vengono effettuate direttamente nello script del visualizzatore, ma sono effettuate tramite lo script cesium.js, fornito già nel pacchetto Cesium. Le operazioni compiute dal server e il modo in cui questo fornisce gli asset salvati in molte cartelle e sottocartelle su diversi percorsi rende molto difficoltosa una modifica radicale del server o la creazione di un server partendo da zero. Pertanto, si è scelto di utilizzare il server fornito nel pacchetto CesiumJS, integrandolo per adattarlo alle esigenze del visualizzatore di ESEO.

Le modifiche effettuate al server saranno descritte nel paragrafo successivo; di seguito ci si soffermerà invece sulle caratteristiche del server Cesium e sul modo in cui esso opera.

Il server di Cesium è basato sul modulo di Node *express*. Tale modulo è un framework che si occupa di molte delle attività base di un server o di un'API.

Con *express* è possibile ottemperare a entrambe le necessità di un server, ovvero mettersi in condizione di ascolto attraverso una porta per ricevere le richieste di un client esterno, e gestire le diverse richieste ricevute attraverso una serie di percorsi che queste richieste possono seguire. A tale scopo *express* utilizza l'architettura REST, indicizzando tutte le richieste tramite un indirizzo univoco, ovvero un *url*, che punta al dominio IP del server e quindi segue un percorso all'interno del server per recuperare un file specifico oppure segue un *url* fittizio codificato perché il server riporti una certa risposta se interrogato con quell'*url*. Il server quindi rappresenta una RESTful API, ovvero un'*api* (application program interface) che utilizza l'architettura *REST*. L'*api* è il codice che consente a due programmi di comunicare tra loro; l'architettura *REST* è invece l'architettura per la richiesta e lo scambio dei dati, basata sul protocollo *http*. Come già accennato tale protocollo prevede che le richieste siano fatte a un *url* che indicizza in maniera univoca i dati da richiedere, inoltre le richieste a un'*api* di questo genera devono essere fatte tramite i verbi specifici di *http* per il recupero di informazioni (*GET*) o per la modifica dei dati (*PUT*, *DELETE*, *POST*, *PATCH*). Nel caso del visualizzatore il server utilizza i verbi *GET* e *PUT* principalmente. La praticità di *express* è dovuta al fatto che esso non solo resta in ascolto su una porta, ma ha già implementate le funzioni e i verbi dell'architettura *REST*, che possono essere richiamate in maniera semplice e rapida.

Dal momento che il progetto è stato sviluppato interamente in localhost, ed è stato implementato su un server disponibile sul web solo una volta completato, l'indirizzo di riferimento è sempre stato *http://localhost:3000*, dove *:3000* indica che il server è in ascolto sulla porta 3000. All'interno del server la richiesta viene processata tramite la funzione *get* di *express*, alla quale viene dato in input la parte finale dell'url che deve essere processato, e in base a tale url il server sa in che modo deve processare la richiesta, ovvero quali operazioni compiere. Ad esempio se la richiesta viene effettuata all'indirizzo http://localhost:3000/requesting, nel server, in ascolto sulla porta 3000, occorrerà la stringa *app.get("/requesting", (req, res, next) => {operazioni da compiere}*, nella quale occorre inserire tra le parentesi graffe il codice relativo alle operazioni da eseguire in seguito a tale richiesta. Nella variabile *app* sono memorizzate le funzionalità del modulo *express*, in quanto essa è creata con la stringa *var app = express()*.

Solitamente, il server deve processare la richiesta e inviare una risposta, come si osserva anche dalla struttura della stringa riportata sopra, dove *req* sta per *request* e *res* per *response*. Il server viene quindi interrogato all'url *requesting*, esegue il codice relativo a tale richiesta e invia il risultato al client che effettua la richiesta, utilizzando il formato *json* nel caso la risposta sia costituita da dati. A tale scopo viene utilizzata la funzione *res.json(JSON)*, che restituisce al client un'informazione già convertita in formato json. Il client dovrà quindi parserizzare la risposta per poterla tramutare in una funzione o in un valore di determinate variabili, ovvero in un'informazione utilizzabile da Javascript. Nel prossimo paragrafo saranno descritte in particolare due modifiche effettuate al server per la gestione di due specifiche richieste, riguardanti la restituzione dei dati relativi ai quaternioni del satellite e ai TLE.

3.2.3 Richiesta dei TLE e dei quaternioni al server tramite REST

Le modifiche effettuate al server di Cesium sono state relative all'aggiunta di due call per fornire al client i quaternioni e i TLE, dati necessari per la realizzazione del visualizzatore, il ruolo dei quali sarà discusso in seguito.

I quaternioni provengono dai dati di telemetria di ESEO, e pertanto devono essere estratti dal database nel quale sono stati salvati. Per fare questo si è innanzitutto

aggiunto al server il modulo *mysql*, tramite la funzione *require*. Tale modulo contiene tutte le funzioni per potersi collegare tramite Javascript a un database Mysql, e per poter fare una query per estrarne i dati. Quindi si è creata una variabile *con* per effettuare la connessione al server, usando la funzione *mysql.createConnection({})*, che vuole tra parentesi tutti i dati relativi al database al quale si vuole connettere (nome del database, password, indirizzo IP, porta di riferimento e nome dell'utente). A questo punto si è creata una serie di funzioni annidate per estrarre i quaternioni e inviarli al client che ha effettuato la richiesta.

La necessità di annidare le funzioni è legata al funzionamento asincrono di Node, che da un lato consente di rendere il server una piattaforma estremamente potente, in quanto esso può rispondere a diverse richieste che gli vengono fatte limitandosi a eseguire la funzione richiamata, invece di eseguire tutto il programma, ma il cui funzionamento asincrono non consente di eseguire le funzioni in serie, in quanto il server mentre effettua la query al database, potrebbe proseguire l'elaborazione inviando i dati al client tramite un json, che però risulterà vuoto in quanto i dati non sono ancora stati estratti. Pertanto, l'utilizzo delle funzioni annidate fa sì che una funzione venga richiamata direttamente dalla precedente solo quando quest'ultima abbia finito di eseguire le proprie task, fornendo i risultati alla funzione successiva tramite la chiamata callback. Il funzionamento della richiesta dei quaternioni è quindi il seguente: quando il client effettua una richiesta di tipo GET (per ottenere indietro dei dati) all'indirizzo *http://localhost:3000/requesting*, nel server la richiesta viene processata dalla funzione *app.get("/requesting", (req, res, next)*. Tale funzione richiama la funzione *console_quat(res)*, che a sua volta richiama la funzione *estrai()*. Nel caso la funzione *estrai()*, basata su una chiamata callback, restituisca un errore l'esecuzione si interrompe, in caso contrario i dati ricevuti, che sono i quaternioni, vengono assemblati in una stringa formato json e rinviati al client con il comando *res.json(dati)*, che restituisce la risposta *res* alla richiesta. La funzione *estrai* come accennato lavora tramite callback. Tale funzione nella pratica effettua prima la connessione al database tramite il comando *con.connect()*, dove *con* è la variabile precedentemente creata che contiene i dati del database, quindi sempre tramite la variabile *con* si utilizza la funzione *query* per effettuare la query al database, utilizzando una stringa analoga a quelle viste nel capitolo precedente. La funzione *query* prende in ingresso anche una funzione per valutare i risultati, che in

caso di errore lancia un messaggio nella console, e in caso di successo invece restituisce i risultati tramite il callback. La query è impostata per selezionare dal database gli ultimi cinque dati disponibili relativi ai quaternioni, riordinandoli dal più recente a più vecchio, e riportando anche l'epoca relativa a una data serie di quaternioni. I quaternioni sono rinominati dalla query per averne una notazione più compatta, e sono rinviati al client in ordine di data crescente.

L'annidamento delle funzioni, unito alla funzione di callback, che restituisce il risultato solo quando l'estrazione dal database è completa, consente di eseguir le operazioni nel giusto ordine, assicurando che la risposta venga inviata al client solo dopo che il callback si è completato e che quindi i quaternioni sono stati estratti. Un'altra possibilità per l'esecuzione sincrona delle operazioni sarebbe stata l'utilizzo della funzione *waterfall*, contenuta nel modulo *sync*, che ha lo specifico ruolo di eseguire le funzioni a cascata tramite chiamate callback. L'utilizzo di tale funzione tuttavia sarebbe risultato complicato e non necessario per una sequenza relativamente semplice, come quella utilizzata per i quaternioni. Inoltre, in fase di implementazione tale funzione ha dato diversi problemi con l'utilizzo delle chiamate *promise* per estrapolare i TLE, pertanto si è preferito utilizzare una serie di funzioni annidate per entrambe le operazioni.

La funzione per l'estrazione dei *TLE* è stata la seconda modifica effettuata al server. In tal caso sono stati necessari diversi moduli non compresi tra quelli integrati in Node, che sono pertanto stati installati tramite npm. Tali moduli sono *request*, *request-promise* e *cheerio*. L'unico modulo che deve essere integrato nel server è *request-promise*, in quanto gli altri due vengono richiamati poi da quest'ultimo. La peculiarità di questi moduli è che implementano una funzione in grado di estrapolare il codice html da un sito esterno, come *celestrak.com*. La richiesta del client per i TLE viene indicizzata come quella per i quaternioni con riferimento però questa volta all'url *http://localhost:3000/tle*. La funzione *app.get* questa volta richiama la funzione *estr_tle(res)*. Tale funzione si limita a eseguire il modulo request-promise sull'indirizzo che gli è indicato, che in questo caso è *https://celestrak.com/satcat/tle.php?CATNR=43792*, ovvero la pagina di celestrak nella quale sono salvati i tle di ESEO. Si usa quindi la funzione *.then(html)*, che viene eseguita solo quando l'estrazione del codice html è completa. All'interno di questa funzione viene eseguito il codice per parserizzare l'html, trascurando tutto il

codice html e tenendo solo le due stringhe che contengono i TLE, individuate a partire dalle sottostringhe “1 43792U” e “2 43792” che caratterizzano rispettivamente l’inizio della prima e della seconda linea dei TLE. Le due stringhe che vengono create sono quindi salvate come due attributi della variabile TLE, trasformati in json e reinviati al client. Anche in questo caso l’utilizzo di una funzione annidata assicura che tutte le operazioni vengano eseguite in ordine corretto, evitando di rinviare al client una risposta vuota.

Gli script per estrarre i quaternioni e i TLE da celestrak.com sono riportati in Appendice A.

3.3 Cesium

La piattaforma Cesium rappresenta una parte fondamentale del progetto, in quanto essa contiene tutti gli attributi e le funzioni necessarie alla costruzione del visualizzatore, oltre a vari asset per modelli 3D, terreni 3D, immagini, loghi, esempi di applicazioni già create. La piattaforma presenta vari supporti online, con una lunga lista di tutorial, di esempi già implementati e le *api* riferite a tutte le funzioni fornite dalla libreria CesiumJS. Il pacchetto Cesium si divide principalmente in CesiumJS, la libreria Javascript per lo sviluppo delle applicazioni web, completata da una serie di applicazioni di supporto come il server, e il cloud CesiumION, sul quale è possibile salvare asset che saranno poi resi disponibili a tutti i client che utilizzano l’applicazione, evitando di dover occupare spazio di memoria sulla macchina in cui gira il server.

3.3.1 Libreria CesiumJS

La libreria CesiumJS è stata utilizzata per la realizzazione di questo lavoro di tesi, in quanto è essa a fornire le funzioni per creare un globo 3D, per creare tutte le polilinee e gli oggetti posizionati all’interno del globo e per gestire il tempo all’interno di un’applicazione. La più semplice applicazione che può essere creata con CesiumJS è un semplice globo, inserito all’interno dell’interfaccia di Cesium. Tale globo è creato con la variabile *viewer* alla quale viene attribuito il tipo *Cesium.viewer*. Il codice è: *var viewer = new Cesium.Viewer ('cesiumContainer',*

{}), che crea tutto l'ambiente del visualizzatore, implementando quindi lo sfondo dello spazio profondo con le stelle, il globo con le mappe bing al centro, e in basso una barra per regolare lo scorrimento del tempo, fermandolo o modificandone la velocità e mostrando l'orario UTC di riferimento, e una barra che mostra in quale momento della simulazione ci si trova, riportando l'intera estensione temporale della funzione, che di default è settata a 24 ore. Il globo così creato può essere ruotato e zoomato fino a una risoluzione di circa un metro. Nella figura Figura 15 è riportato l'ambiente creato in questo modo, che risulta il più semplice possibile.

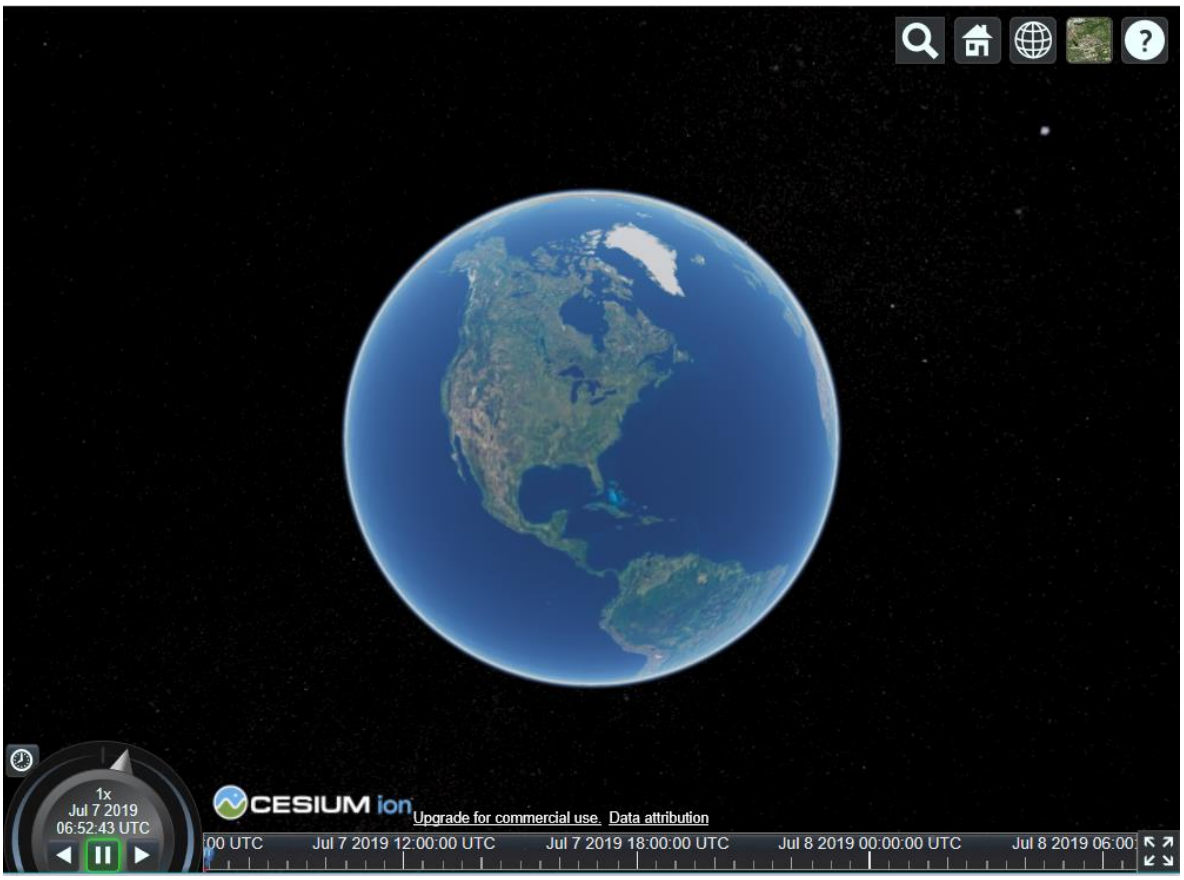

Figura 15. Ambiente Cesium, con la sola visione del globo 3D

3.3.2 Caricamento dei tle e dei quaternioni tramite richiesta al server

Una volta creato il globo 3D, il primo passaggio è stato la richiesta dei dati al server. Il caricamento dei TLE è stato effettuato immediatamente, in quanto essi sono richiesti per il funzionamento della restante parte dell'applicazione. Dal momento che la richiesta al server è un passaggio delicato, si è scelto di caricare immediatamente anche i quaternioni. Inoltre, dal momento che la presenza dei TLE

è fondamentale per la corretta esecuzione dell'applicazione, e che la loro richiesta tramite il server, che deve effettuare un'ulteriore richiesta al sito *celestrak.com*, può risultare piuttosto lunga, anche in questo caso come nel server si è proceduto realizzando l'intero script principale come una serie di funzioni annidate, che si richiamano solamente dopo che si è completata l'operazione precedente. Si è comunque fatto in modo che tutte le operazioni che non necessitano di altri processi siano libere di essere eseguite in modo asincrono, per rendere l'esecuzione dell'applicazione la più rapida possibile.

La richiesta al server viene effettuata tramite la funzione *getTLE(url, successHandler, errorHandler)*, che possiede come parametri l'*url* al quale effettuare la richiesta, e due funzioni *successHandler* e *errorHandler* che vengono eseguite in base alla risposta ricevuta dal server. La funzione agisce eseguendo a sua volta la funzione *XMLHttpRequest()*. Tale funzione si connette al server, selezionando il tipo di richiesta (*GET* in questo caso), il tipo di file che vuole venga restituito (json), e l'*url* al quale vuole effettuare la richiesta, in questo caso *http://localhost:3000:requesting*. A questo punto la richiesta viene spedita al server e si attende una risposta dal server. All'interno della funzione vi è una serie di *if* annidati: il primo rileva quando l'operazione è completata ed è stata ricevuta una risposta dal server, il secondo *if* agisce in base alla risposta ricevuta, eseguendo la funzione *successHandler* in caso la richiesta sia stata processata correttamente e *errorHandler* in caso contrario.

Per utilizzare la funzione XMLHttpRequest, occorre implementare la libreria Javascript *jquery*, che consente di utilizzare il metodo di programmazione chiamato ajax e in particolare di effettuare richieste a un server tramite il protocollo http. Per far funzionare il programma non è stato necessario installare il modulo *jquery*, ma è risultato sufficiente utilizzare lo script principale della libreria *jquery.js*, richiamandolo come script a sé stante prima di effettuare la richiesta, senza dovervi effettuare modifiche.

La seconda funzione utilizzata, per l'estrazione dei quaternioni opera esattamente allo stesso modo, ed è chiamata *getQuat()*. Tale funzione si connette all'*url http://localhost:3000/requesting*, e opera richiedendo i quaternioni e processandoli con una funzione *successHandler* specifica.

Gli script contenenti le due funzioni sono tenuti separati dallo script principale, mentre in quest'ultimo viene effettuata la chiamata alle funzioni e vengono definite le funzioni *successHandler* e *errorHandler* per ciascuna. Questa separazione è effettuata per ragioni di gestione dell'applicazione, in quanto la modalità di richiesta dei dati potrebbe dover subire variazioni a seguito di modifiche al server, e queste modifiche non riguardano l'utilizzo dei parametri che viene fatto nel visualizzatore. Il modo in cui però i dati richiamati vengono utilizzati, e quindi le funzioni di handling, vengono definiti invece nel visualizzatore, perché modifiche e aggiornamenti di quest'ultimo potrebbero implicare un diverso processamento dei dati.

Nel caso in cui la richiesta al server non abbia successo, le funzioni *errorHandler* assicurano che l'esecuzione possa proseguire correttamente, anche in assenza dei dati aggiornati. Il funzionamento della sequenza è il seguente: viene eseguita la funzione getTLE; nel caso in cui tale richiesta non abbia un buon esito viene eseguita la funzione *errorHandler,* e alla variabile *tle* viene assegnato l'ultimo valore che è stato inserito manualmente nel codice, pertanto un set di TLE è sempre assegnato, anche se non sempre è aggiornato. La funzione *successHandler* assegna invece alla variabile *tle* gli attributi *line1* e *line2*, nei quali vengono salvate le due linee dei TLE. A questo punto, sia che la funzione abbia successo, sia in caso contrario le funzioni *successHandler* e *errorHandler* richiamano la funzione *getQuat()*, che lavora in maniera analoga, assicurando sempre il funzionamento dell'applicazione. Tale funzione infatti processa i quaternioni in caso di successo, mentre assegna un orientamento prefissato in caso di fallimento. Riguardo all'assegnamento dei quaternioni, vengono estratti gli ultimi cinque set disponibili dal database, tuttavia in fase di assegnamento delle variabili è stata riscontrata una certa difficoltà nel creare un'interpolazione per l'assetto assunto dal satellite, e inoltre essendo la simulazione del visualizzatore real time, essa è sempre posteriore alla data alla quale si riferisce un certo set di quaternioni. Pertanto, si è scelto di utilizzare solamente l'ultimo set di quaternioni disponibile, e di considerare l'assetto che ne deriva costante per tutto il tempo della simulazione. Tale set viene implementato assegnandolo a una variabile di tipo *Quaternion*, attraverso il costrutto *ori = new Cesium.Quaternion(q1, q2, q3, q4),* dove *Quaternion* è uno degli attributi disponibili su CesiumJS. La variabile *ori* verrà poi assegnata al

modello del satellite, nell'attributo *orientation*. In caso di insuccesso invece, per consentire il funzionamento dell'applicazione, viene assegnato un valore fittizio alla variabile ori, utilizzando un modello di orientamento allineato al vettore velocità definito da Cesium (*Cesium.VelocityOrientationProperty(posizione)*), in cui il satellite è nadir pointing. Si è fatta questa scelta in quanto questo dovrebbe essere l'assetto nominale del satellite.

Le parti di codice relativi alla richiesta dei quaternioni e dei TLE e al loro riferimento sono riportati in Appendice B.

Una volta che il processo è completato, ovvero dopo che sono state eseguite le funzioni *successHandler* o *errorHandler* di *getQuat()*, tali funzioni fanno proseguire l'elaborazione richiamando la funzione *createscenario();* Tale funzione inizia a creare lo scenario, stabilendo la durata della simulazione e impostando il tempo di inizio. Per fare questo, essendo la durata della simulazione legata alla durata dell'orbita, si fa un primo uso della variabile *sgp4*, implementata modificando l'analogo modulo.

3.3.3 Adattamento del propagatore SGP4 a Cesium

Il propagatore sgp4 è stato necessario per due processi nel visualizzatore, il primo come già sottolineato è il calcolo della durata di un'orbita, corrispondente alla durata della simulazione, il secondo processo, più importante è la propagazione dell'orbita, al fine di poterne calcolare lo sviluppo e tracciarne l'andamento insieme a quello del satellite.

Il propagatore sgp4 non è stato sviluppato da zero, ma si è fatto uso di un modulo per Node disponibile su github.com. Tale modulo non è stato utilizzato interamente, ma si è utilizzato solo lo script comprendente il propagatore, attuando unicamente alcune piccole modifiche. Il propagatore necessita di un set di TLE per caratterizzare l'orbita, e per questo motivo è stata necessaria l'estrazione degli stessi. A partire dai TLE, sgp4 esegue una lunga serie di funzioni concatenate, per definire un record delle caratteristiche dell'orbita, attraverso il comando *eseoSatRec = SGP4.twoline2rv(eseoLine1, eseoLine2,SGP4.wgs84())*. A questo punto possono essere ricavate la posizione e la velocità del satellite in un certo istante, attraverso la funzione *SGP4.propogate(eseoSatRec, julianday)*, dove *julianday* rappresenta la

data giuliana dell'istante rispetto al quale si vogliono sapere le caratteristiche vettoriali del moto del satellite. In questa funzione è stata effettuata una prima modifica, in quanto Javascript non è in grado di fornire la data in formato giuliano, mentre la libreria Cesium implementa questa possibilità. Invece di ricavare la data in formato ISO UTC, per poi usare una funzione intermediaria per la conversione, è stato quindi possibile utilizzare direttamente le funzioni di sgp4 fornendo in ingresso il giorno giuliano. Il compito del propagatore si completa con le funzioni per calcolare le coordinate del vettore posizione e velocità in un S.d.R. inerziale. Tuttavia, il propagatore fornisce una serie di funzioni utili che sono state usate nel visualizzatore per altri scopi, e che sono di seguito elencate. Le funzioni sono un convertitore da posizione e velocità a coordinate geodetiche, un convertitore da coordinate geodetiche a latitudine, longitudine e altitudine, e una serie di funzioni per ricavare l'angolo di vista e l'elevazione del satellite in un dato istante rispetto a un punto della terra, definito in coordinate cartesiane. Nessuna di queste funzioni ha dovuto subire modifiche. Nell'implementazione di sgp4 nessuna delle funzioni fornite nel modulo è stata rimossa, anche se inutilizzata, in quanto tali funzioni potrebbero rivelarsi utili in sviluppi futuri.

3.3.4 Settaggio real-time del visualizzatore di Cesium

Come precedentemente accennato, qualsiasi applicazione in Cesium è impostata come una simulazione dinamica di una certa durata. Dal momento che il visualizzatore attua una simulazione in tempo reale, esso deve essere impostato per lavorare in un certo range temporale partendo dall'istante in cui si accede allo stesso e eseguendo la simulazione a velocità normale. Nel caso di questo visualizzatore si è scelto di attuare una simulazione della durata di un periodo orbitale. Poiché tale periodo è variabile, occorre calcolare la durata dell'orbita sulla base della posizione attuale del satellite.

Il settaggio del visualizzatore inizia salvando due variabili *start* e *stop* nel formato giorno giuliano di Cesium, impostandole entrambe al valore di tempo corrente, con l'attribuzione *var start = new Cesium.JulianDate.now()*. Si procede quindi a ricavare la posizione e la velocità del satellite nell'istante start con la funzione *SGP4.propogate(eseoSatRec,julianday)*, e quindi le coordinate geodetiche e poi

latitudine, altitudine e longitudine. A questo punto il tempo orbitale può facilmente essere calcolato come 2π moltiplicato per la radice quadrata del raggio al cubo diviso per la costante μ ($\mu = 398600.8\ m^3/s^2$ per la Terra), dove il raggio è uguale alla radice quadrata della somma dei quadrati delle tre coordinate cartesiane del vettore posizione del satellite. Il valore ottenuto viene salvato nella variabile *tempo* e quindi viene definito l'istante *stop*, sommando i valori *start* e *stop*. A questo punto può essere settata la variabile *viewer*, impostando le caratteristiche temporali della simulazione attraverso il suo attributo *clock* (quindi si lavora su *viewer.clock*), che contiene le informazioni sugli estremi della simulazione e sull'istante in cui questa deve essere fatta partire. Si imposta l'inizio della simulazione all'istante *start*, la fine all'istante *stop* e l'epoca corrente nuovamente all'istante *start*. Quindi si imposta la velocità di simulazione a 1x e si imposta la simulazione per ricominciare dall'inizio nel momento in cui termini. Infine si imposta la barra del tempo visualizzata in basso nel simulatore sugli stessi estremi della simulazione, ovvero *start* e *stop*, con il comando *viewer.timeline.zoomTo(start, stop)*. Il codice sorgente per il settaggio del tempo di simulazione è riportato in appendice C.

Dopo aver settato il tempo, l'applicazione prosegue con l'interpolazione dell'orbita del satellite e quindi con la creazione di tutte le entità, come sarà descritto nei paragrafi successivi.

3.3.5 Interpolazione e tracciamento dell'orbita e della traccia a terra.

L'interpolazione dell'orbita avviene utilizzando nuovamente il propagatore sgp4. Per ricreare l'orbita si è scelto di interpolare una serie di punti calcolando le posizioni orbitali ad una distanza di 30 secondi l'una dall'altra, e quindi di interpolare i punti ottenuti per ottenere l'orbita. La scelta di un intervallo di 30 secondi è un compromesso tra il peso della computazione e la precisione dell'interpolazione ottenuta. Per calcolare i punti si è utilizzato un ciclo *for*. All'interno di tale ciclo sono stati calcolati non solo i punti dell'orbita, ma anche gli analoghi punti della traccia a terra. Inoltre, per risparmiare risorse computazionali, il ciclo è stato utilizzato anche per calcolare gli intervalli di passaggio del satellite sopra alle stazioni di terra.

La creazione dei punti dell'orbita viene effettuata dalla funzione *printPosition()*, che definisce la variabile *property*, del tipo *samplePositionProperty*, ovvero una variabile nella quale possono essere salvati gli attributi di posizione di un'entità. Quindi ha inizio il ciclo for, che si svolge per *i* che va da 0 alla durata dell'orbita con step di 30. Quindi per ogni valore di *i* si calcola la posizione del satellite tramite sgp4, definendo il valore temporale *time*, nel quale calcolare la posizione, sommando al valore *start* il valore corrente di *i* e ricavando la posizione e la velocità per quell'istante. Per una corretta visualizzazione dell'orbita la posizione del satellite deve essere calcolata in un sistema di riferimento inerziale. Quindi è sufficiente la posizione, che viene salvato in una variabile d'appoggio *position* come vettore cartesiano utilizzando il metodo *Cartesian3()*, nel quale vengono inserite le tre coordinate moltiplicate per mille, in quanto il propagatore restituisce le coordinate in chilometri, mentre Cesium lavora in metri. Occorre poi impostare la variabile *property* come una *samplePositionProperty()* con sistema di riferimento inerziale, attraverso il parametro *Cesium.referenceFrame.INERTIAL* inserito nella dichiarazione dell'attributo. Per ogni ciclo del for gli attributi di posizione e *time* vengono quindi salvati in *property* con il metodo *property.addSample (position,time)*. Nella variabile property, che diventerà un vettore di posizioni, occorre salvare anche l'istante relativo ad una certa posizione, in questo modo l'applicazione sarà in grado di interpolare non solo la posizione occupata dalla traccia del satellite, ma anche l'istante in cui il satellite si troverà in una determinata posizione, potendo quindi effettuare una simulazione in cui il satellite si sposta nel tempo.

Per quanto riguarda la traccia a terra, l'interpolazione è analoga, con la differenza che questa volta i punti devono essere calcolati in un sistema di riferimento ad assi fissi solidali con la terra, utilizzando quindi la latitudine e la longitudine e ponendo l'altezza uguale a zero. In questo caso si possono usare le opportune funzioni già implementate nel propagatore per la conversione delle coordinate, utilizzando poi il metodo *Cesium.Cartesian3.fromDegrees(coordinate_in_lat_e_lon)* per aggiungere le posizione della traccia a terra ad un vettore dedicato, chiamato in questo caso *traccia*. I due vettori *property* e *traccia* vengono quindi utilizzati per assegnare la posizione alle entità ESEO (chiamata *entity*) e traccia a terra (chiamata *tracciamento*). Al termine del ciclo *for* le variabili vengono calcolate un'ultima

volta all'istante *time = stop*, in modo che i punti siano calcolati esattamente sulla lunghezza di un'orbita.

La funzione *printPosition()* è riportata per intero in appendice D, comprendendo anche il codice per il calcolo del passaggio sulla stazione di terra, che sarà discusso in uno dei prossimi paragrafi.

3.3.6 Creazione delle entità per ESEO, Ground Station e traccia a terra.

Dopo che la funzione *printPosition()* ha calcolato i punti dall'orbita, possono essere create le entità ESEO e traccia a terra. Nella libreria Cesium le entità sono gli oggetti che vengono creati all'interno del visualizzatore, come polilinee, punti, solidi, o anche oggetti 3D. Le entità hanno una grandissima varietà di attributi disponibili, dei quali saranno descritti solamente quelli utilizzati per la realizzazione del visualizzatore.

La più importante entità creata è il satellite ESEO. Tale entità racchiude sia la rappresentazione dell'orbita, che risulta statica, sia la rappresentazione del satellite, che invece è dinamica e durante la simulazione si muove lungo la polilinea dell'orbita. All'entità ESEO, chiamata nel codice *entity*, è assegnato innanzitutto l'attributo *availability,* ovvero un intervallo di disponibilità nel quale l'entità viene visualizzata, impostato tra i tempi *start* e *stop*. Poi è assegnata all'entità la posizione calcolata con la funzione *printPosition()* e l'attributo *'id'*, usato per identificare l'entità, al quale viene dato il nome *'satellite'*. Al satellite viene poi aggiunto l'attributo *name*, al quale viene assegnato la stringa *'ESEO'* e l'attributo *orientation*, che definisce l'orientazione del satellite e al quale viene assegnata la variabile *ori* nella quale sono inseriti i quaternioni. Si procede quindi assegnando al satellite gli attributi *model* e *path*. Il primo carica il modello 3D dalla directory del server, che poi viene adeguatamente scalato impostando una dimensione minima e massima ottenibili con il comando zoom del visualizzatore, mentre il secondo crea la traccia dell'orbita. Cesium riconosce automaticamente il ruolo dell'attributo *model*, facendo sì che il modello caricato sia animato e segue la posizione istantanea del satellite nell'orbita ottenuta dall'interpolazione spazio-temporale della posizione dell'entità. Il modello utilizzato è un modello 3D in formato glb, ovvero la versione binaria dei formati glTF, ottimizzati per l'utilizzo in applicazioni web. Tale modello è stato

fornito da Sitael nel formato corretto, ma privo di colori. Esso è stato pertanto colorato utilizzando l'applicazione 3D Builder di Windows, al fine di riprodurre l'aspetto del satellite reale.

L'attributo *path* invece interpola solo spazialmente la posizione, e quindi crea una polilinea statica che rappresenta il percorso effettuato dal satellite. All'entità è infine aggiunto l'attributo *label*, che visualizza sopra al satellite una scritta con il suo nome, settando le impostazioni in modo che la scritta sia delle giuste dimensioni e non venga coperta dal satellite.

La traccia a terra è invece creata attraverso l'entità *tracciamento*, con id *'track'* e la stessa availability del satellite. All'attributo *position* è assegnato in questo caso la variabile *traccia*, precedentemente definita, mentre viene definito anche in questo caso un attributo *path*, che genera una polilinea rappresentante la traccia a terra, definita in assi fissi, e l'attributo *point*, che crea un punto rappresentante la posizione a terra del satellite. Tale punto è dunque il *sub-satellite point*, e il suo spostamento è solidale con quello rispettivo del satellite.

Nel caso di ESEO e della traccia a terra occorre aggiungere un ulteriore passaggio per la corretta visualizzazione dei path. Infatti, Cesium di default interpola i punti con una funzione di primo grado, mostrando quindi la polilinea come una serie di spezzate. Per la visualizzazione delle orbite occorre invece usare un'interpolazione di grado superiore. A tale scopo si può utilizzare il metodo *setInterpolationOption*, assegnato all'attributo *position* delle due entità, che in Javascript può essere richiamato successivamente alla creazione delle entità, impostando i suoi due attributi come di seguito: ad *interpolationDegree* viene assegnato il valore 5, corrispondente a un'interpolazione di quinto grado e ad *interpolationAlgorithm* l'algoritmo di interpolazione lagrangiana (*LagrangePolynomialApproximation*).

Le ultime due entità create sono la stazione di terra di Forlì e quella di Tartu, in Estonia. Pur non essendo una delle stazioni di terra principali di missione, l'osservatorio di Tartu ha avuto un ruolo importante nello stabilire il contatto con il satellite, mettendo a disposizione l'antenna per le comunicazioni con la stazione di terra di Forlì. Pertanto, si è deciso di visualizzare entrambe le stazioni. Per entrambe le entità si definisce un id (*GSF* e *GST* rispettivamente), e si definisce in position la posizione in coordinate cartesiane delle stazioni. Le stazioni sono indicate sul globo

caricando un logo dalla directory di Cesium con l'attributo *billboard*, e ponendolo nelle giuste coordinate. Infine, viene visualizzato il nome delle due stazioni, utilizzando ancora una volta l'attributo *label*. Il posizionamento delle due stazioni è indicato nella Figura 16 e nella Figura 17, mentre nella Figura 18 si mostra tutta la configurazione del visualizzatore a questo punto. Lo script per la creazione delle entità è riportato in appendice E.

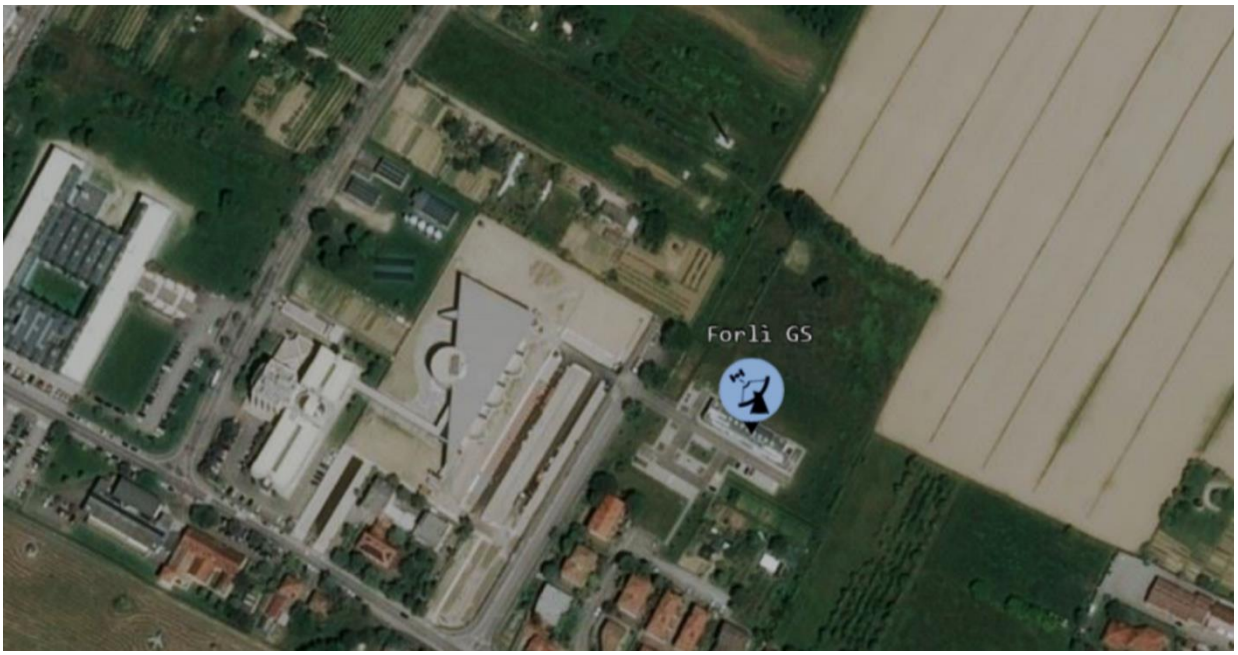

Figura 16. Stazione di terra di Forlì

Figura 17. Stazione di terra di Tartu

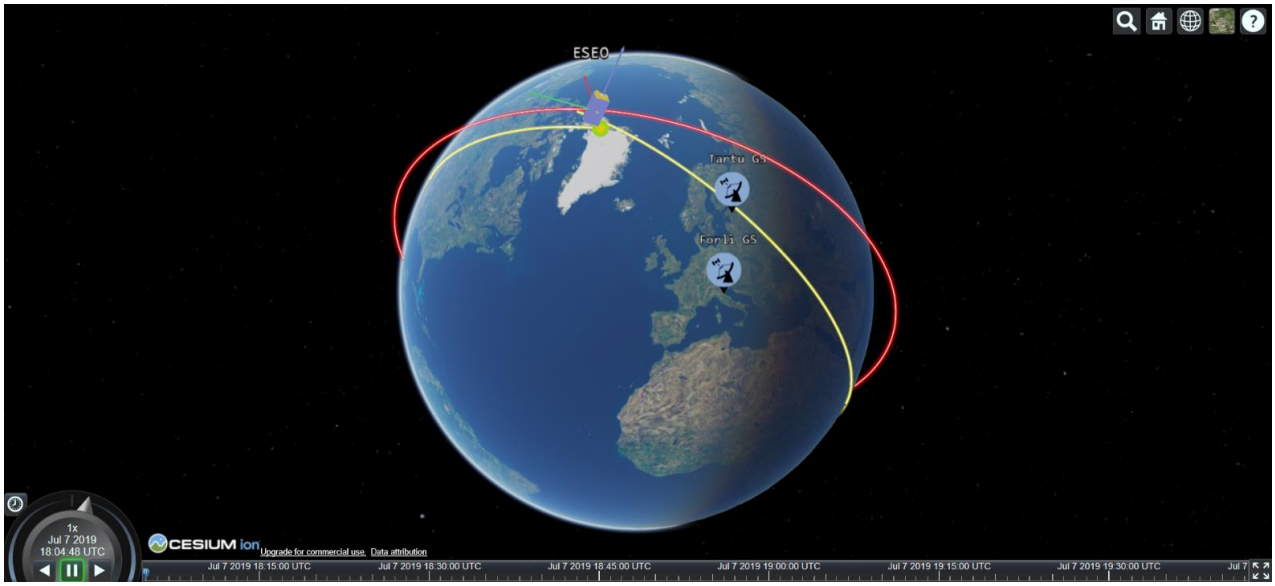

Figura 18. Visualizzatore con l'aggiunta delle entità

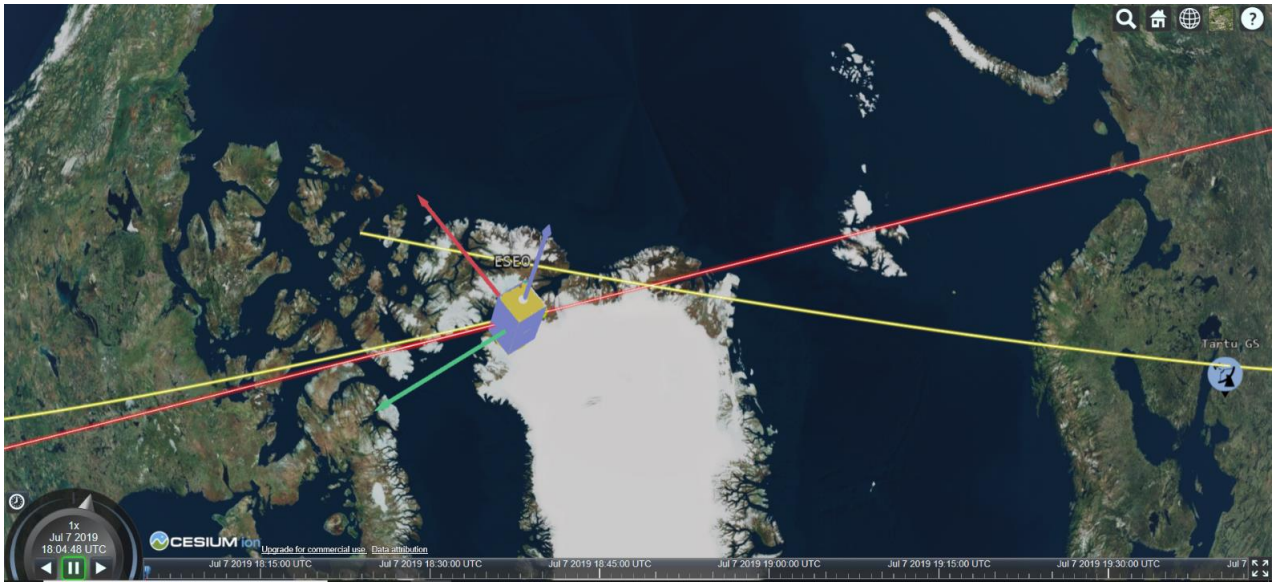

Figura 19. Zoom sul satellite

3.3.7 Visualizzazione degli accessi alle Ground Station

L'ultima funzione che è stata aggiunta al visualizzatore è quella per visualizzare l'accesso del satellite alle due stazioni di terra, che viene rappresentato tramite una polilinea che collega il satellite alla stazione di terra, quando esso si trova sopra l'orizzonte rispetto ad essa. La creazione degli accessi viene effettuata sempre nel ciclo for della funzione *printPosition()*, sfruttando le funzioni di sgp4, *eciToEcf*, *topocentricToLookAngles* e *elevation*. Con queste funzioni si riesce a calcolare

l'angolo di vista del satellite rispetto a una certa posizione (in questo caso viene utilizzata quella delle due stazioni di terra), quindi si trasforma l'angolo di vista in un'elevazione. Per visualizzare la polilinea che segnala l'accesso, occorre definire degli intervalli nei quali tale polilinea deve essere visualizzata, ovvero quando l'elevazione è maggiore di zero. Per fare ciò si imposta inizialmente l'elevazione a zero, quindi si avvia il ciclo *for*. Un *if* verifica le condizioni sull'elevazione. Se la nuova elevazione calcolata è maggiore di zero, viene salvato l'istante di tempo corrispondente come istante iniziale (start_pas). Una volta che si ha un'istante iniziale si osserva ciò che accade nei cicli successivi. Finché l'elevazione è maggiore di zero, e anche l'elevazione nel ciclo precedente lo è, non accade nulla, quando però l'elevazione diventa negativa, essendo stata positiva fino al ciclo precedente, vuol dire che il passaggio è terminato; si salva l'istante finale e i due istanti *start*_pas e *stop*_pas vengono aggiunti a un array di intervalli come attributi *start* e *stop*. In questo modo viene creata una serie di intervalli temporali dei passaggi. Nel caso della simulazione impostata sul visualizzatore, della durata di un'orbita, raramente si ha più di un passaggio nella stessa simulazione. Tuttavia, lavorare in questo modo consente di avere un'applicazione già funzionante nel caso si voglia per qualsiasi motivo estendere il tempo di simulazione. Il calcolo dei passaggi viene effettuato per entrambe le stazioni. Una volta effettuato questo, occorre creare la polilinea che segnalerà il passaggio. Tale polilinea viene creata come un'entità a sé stante, in maniera identica per le due stazioni di terra, pertanto si riporteranno solo le operazioni effettuate per visualizzare il passaggio sulla stazione di terra di Forlì. Si crea dunque l'entità pasF, impostando come id *'pasF'* e come availability quella calcolata con l'elevazione. La posizione dell'entità viene fissata calcolando la posizione del satellite nella funzione *printPosition()* utilizzando questa volta l'altitudine la latitudine e la longitudine. Quindi si definisce la polilinea che collega il satellite e la stazione con l'attributo *polyline*. Bisogna impostare un attributo della polilinea perché questa non segua la superficie terrestre, inoltre per poterla rappresentare in movimento occorre assegnare un array di posizioni alla polilinea, le cui estremità sono create con la proprietà *Cesium.ReferenceProperty*. Tale proprietà fa si che la polilinea si riferisca alla posizione attuale delle due entità indicate da *ReferenceProperty*, che però devono essere riferite attraverso il loro id, indicando quale attributo dell'entità si vuole riferire (in questo caso la posizione). Il comando completo per la creazione degli estremi della polilinea risulta quindi:

positions: new Cesium.PositionPropertyArray([new Cesium.ReferenceProperty(collection, 'GSF', ['position']), new Cesium.ReferenceProperty(collection, 'pasF', ['position']). Come si osserva bisogna riferirsi agli id *GSF* della ground station e *pasF*, dell'entità polilinea, dove la posizione di *pasF* è la stessa del satellite ma in coordinate cartografiche. Tali id devono prima essere inseriti in una collection, e questo viene effettuato tramite il comando *collection.add(id_della_entità)*, eseguito dopo la creazione di ogni entità. Le parti dello script necessarie per la visualizzazione dei passaggi sono riportate nell'appendice F, mentre nella figura sotto si mostra il risultato di quest'operazione.

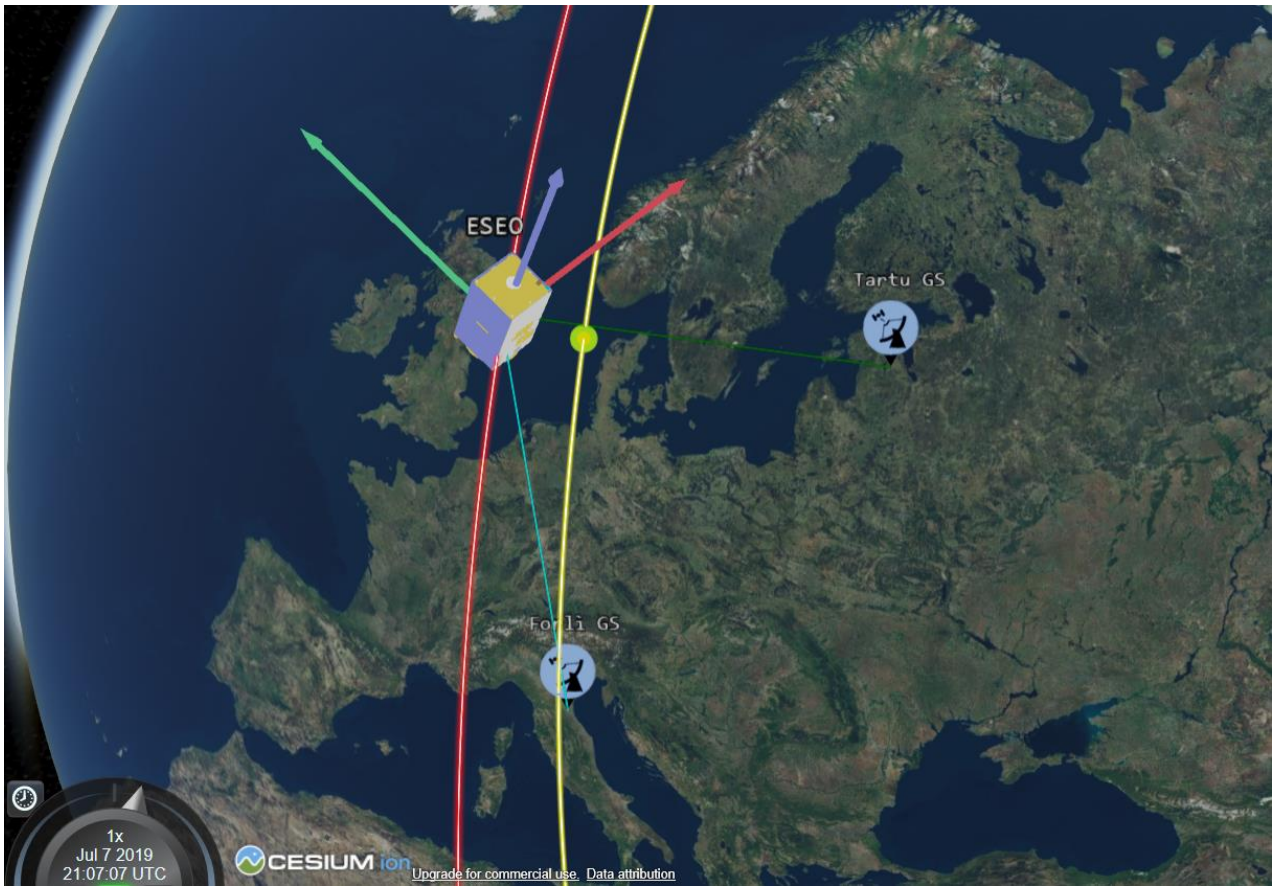

Figura 20. Passaggio di ESEO sulle stazioni di terra

3.3.8 Modifica dello script per la visualizzazione dell'assetto del satellite

Con la creazione degli accessi alle stazioni di terra la realizzazione del visualizzatore è completa. Tuttavia, come parte successiva del lavoro è stata creata un ulteriore applicazione, semplificata, per mostrare uno scenario in pausa nel quale la vista è incentrata sul satellite ed esso è rappresentato con l'ultimo assetto

disponibile. Questa applicazione è stata realizzata allo scopo di mostrare un assetto vero del satellite. Per realizzare l'applicazione si è solamente dovuto aggiungere il comando *viewer.trackedEntity = entity*, che fa in modo che la visualizzazione sia sempre incentrata sul satellite. Si è quindi proceduto a eliminare la traccia dell'orbita, i passaggi sulle stazioni di terra e a impostare il visualizzatore perché inizi la simulazione in pausa. Il risultato è mostrato in figura

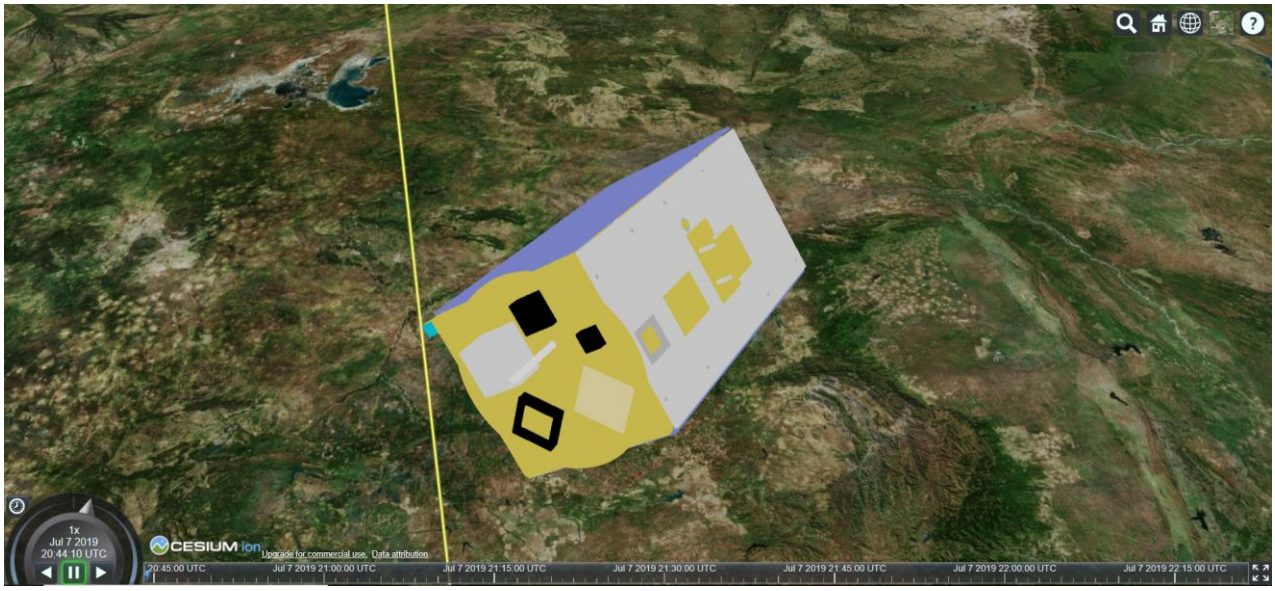

Figura 21. Visualizzatore per la visione centrata sul satellite

3.4 Pagina html di avvio

3.4.1 Unificazione degli script all'interno del codice html

Per far funzionare gli script di Javascript su un'applicazione web, essi devono essere eseguiti all'interno di una pagina html, che raggruppa le funzioni di Javascript e le impostazioni grafiche della pagina, che vengono definite attraverso il linguaggio CSS. La pagina html viene fornita al client tramite il server e può essere aperta nel browser. Attraverso questa pagina il client richiama quindi gli script con la stringa *<script src="nome_script.js"></script>* contenuta nel codice html ed effettua le richieste al server per richiamare tutte le risorse necessarie al funzionamento dell'applicazione. Per la creazione della pagina html si è usata una pagina di base fornita nel pacchetto CesiumJS, nella quale viene impostato uno stile importato da Cesium e nella quale viene caricato inizialmente lo script Cesium.js, che si occupa di molte funzioni necessarie al funzionamento della libreria. Nella parte del body di

html viene inserito inizialmente un *div* con id CesiumContainer e vengono quindi inseriti tutti gli script dell'applicazione.

Per quanto riguarda l'applicazione principale, la pagina html prende il nome di ESEO_HTML.html, mentre i file Javascript sono in ordine jquery.js, richiestaQuaternioni.js, richiestaTle.js, sgp4.js e ESEO_JS_quat.js, tra i quali sgp4 è lo script contenente il propagatore sgp4 e ESEO_JS_quat.js è lo script principale dell'applicazione, contenente tutta l'implementazione del visualizzatore. Per quanto riguarda il visualizzatore secondario, la pagina html prende il nome di ESEO_HTML_ZOOM.html, mentre lo script principale prende il nome di ESEO_JS_zoom.js.

Eseguendo il server è possibile navigare all'indirizzo delle due pagine html e aprire il visualizzatore, facendo partire la simulazione.

3.4.2 Inserimento dei visualizzatori su eseo.ddns.net

L'ultimo passaggio richiesto per il completamento del lavoro è il caricamento dei due visualizzatori sulla pagina web di visualizzazione della telemetria di ESEO. L'inserimento viene effettuato attraverso il panel text di Grafana. L'unica operazione da compiere è il settaggio del panel per l'utilizzo dell'html e l'inserimento di una stringa *iframe* con l'indirizzo della pagina html da caricare. In tal modo possono essere caricati entrambi i visualizzatori utilizzando due panel.

Dato il funzionamento iniziale del server in localhost, i visualizzatori erano inizialmente visibili solo dal computer dell'operatore che avviava il server. Cesium è quindi stato traferito da Sitael su un server aperto alla rete, nel quale tutti i dati e le pagine vengono richiesti non più all'indirizzo localhost ma all'ip del server.

Conclusioni

Questo lavoro di tesi si è concentrato sulla visualizzazione dei dati di telemetria della missione ESEO, con l'obiettivo di rendere questi dati facilmente accessibili e interpretabili, per rendere più semplice l'analisi e il controllo di missione. Nella prima parte del lavoro si sono implementate le *pages* di *housekeeping* della telemetria di ESEO, creando dei grafici e delle tabelle attraverso la piattaforma di visualizzazione di dati analitici Grafana, seguendo la suddivisione iniziale delle pages e organizzando visivamente i dati in maniera tale da riprodurre l'ordine nel quale questi sono stati riportati nei manuali di missione, per rendere più facile la loro ricerca in caso di necessità.

La seconda parte del lavoro si è invece concentrata sulla realizzazione di un visualizzatore 3D, per mostrare in tempo reale la posizione e l'assetto del satellite ESEO. Tale lavoro ha richiesto la creazione di un server web e l'utilizzo della libreria open source Cesium di Javascript. La creazione del visualizzatore va a costituire uno step ulteriore nell'implementazione della pagina web *eseo.ddns.net*, sulla quale sono caricati i dati di telemetria del satellite, andando a migliorare la precedente applicazione per la traccia del satellite, che operava utilizzando una mappa topografica 2D. Tuttavia, anche se il lavoro di creazione del visualizzatore è stato completato rispondendo a tutti gli obiettivi in essere, lo stesso modo in cui il viewer è stato progettato apre alla possibilità di nuovi aggiornamenti futuri, svolti magari attraverso nuovi lavori di tesi, con le stesse prospettive formative che costituiscono l'obiettivo primario della missione ESEO, sviluppata dall'ESA in stretta collaborazione con gli studenti di molte università europee. Questo lavoro pertanto non può essere visto semplicemente come un'attività da svolgere prima della laurea, ma piuttosto come un percorso formativo parte di un'attività più grande, alla quale molti hanno preso parte e altri prenderanno parte ancora.

Bibliografia

Elliott, Eric, *Programmare applicazioni Javascript*, 2014.

Wandschneider, Marc, *Node.js*, 2013.

Documentazione di ESEO, *TMTC_AR-0*, Capitolo 1; *OBDH_AR-04*, Capitolo 1 e dal 2.1 al 2.4; *OBDH_AR-03*

https://esamultimedia.esa.int/docs/edu/The-ESEO-layout-and-subsystems.pdf

https://www.esa.int/Education/ESEO

https://www.esa.int/Education/ESEO/The_ESEO_Mission

https://www.esa.int/Education/ESEO/The_ESEO_spacecraft

https://directory.eoportal.org/web/eoportal/satellite-missions/e/eseo

https://en.wikipedia.org/wiki/Simplified_perturbations_models

https://www.celestrak.com/NORAD/documentation/tle-fmt.php

https://Grafana.com/docs/

https://cesiumjs.org/refdoc/

Appendice

A. Modifiche al server

```
//codice per restituire i quaternioni
var con = mysql.createConnection({
  host     : "137.204.201.133",
  user     : "PublicUser",
  password : "ESEO_DB_2018",
  database : "publicmib",
  port     : 61558
});

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

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);
   });
}

function estrai(callback) {
   con.connect(function(err) {
      console.log('connected as id ' + con.threadId);
      con.query('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', function(err, result) {
   if (err) throw err;
      callback(null,result);
         });
      });
   }


//estrazione dei TLE
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);
});
```

B. Codice di richiesta dei quaternioni e dei TLE

```
//richiesta dei quaternioni
var getQUAT = 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();
};

//richiesta dei TLE
```

```
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/tle
            if (xhr.readyState == 4) { // `DONE`
               status = xhr.status;
               if (status == 200) {
                  successHandler && successHandler(xhr.response);
               } else {
                  errorHandler && errorHandler(status);
               }
            }
         };
   xhr.send();
};


//definizione annidata delle funzioni successHandler e errorHandler nello script principale
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);
         //alert(quat.quaternioni[0].q1);
         createScenario();
         }, function(status) {
            ori = 111;
            createScenario();
         });

}, 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';
   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);
         //alert(quat.quaternioni[0].q1);
         createScenario();
```

```
    }, function(status) {
        ori = Cesium.VelocityOrientationProperty(position);
        createScenario();
    });
});
```

C. Settaggio del visualizzatore real-time

```
function createScenario() {

  eseoSatRec = SGP4.twoline2rv(eseoLine1, eseoLine2,SGP4.wgs84());

  //Based on one orbit
  var julianday = Cesium.JulianDate.totalDays(start);
  var positionAndVelocity = SGP4.propogate(eseoSatRec, julianday);
  // GMST required to get Lat/Long
  var gmst = SGP4.gstimeFromDate(julianday);
  // Geodetic coordinates
  var geodeticCoordinates = SGP4.eciToGeodetic(positionAndVelocity.position, gmst);
  // Coordinates in degrees
  var longitude = SGP4.degreesLong(geodeticCoordinates.longitude);
  var latitude = SGP4.degreesLat(geodeticCoordinates.latitude);
  var height = geodeticCoordinates.height;
  tempo=((2*Math.PI)*(geodeticCoordinates.height+6378.135))*
  (Math.sqrt((geodeticCoordinates.height + 6378.135)/398600.8)) +50;
  stop = Cesium.JulianDate.addSeconds(start, tempo, new Cesium.JulianDate());

  //Make sure viewer is at the desired time.
  viewer.clock.startTime = start.clone();
  viewer.clock.stopTime = stop.clone();
  viewer.clock.currentTime = start.clone();
  viewer.clock.clockRange = Cesium.ClockRange.LOOP_STOP; //Loop at the end
  viewer.clock.multiplier = 1;

  //Set timeline to simulation bounds
  viewer.timeline.zoomTo(start, stop);
}
```

D. Funzione printPosition()

```
//creazione delle variabili per salvare i punti
var traccia = new Cesium.SampledPositionProperty();
var poli = new Cesium.SampledPositionProperty(Cesium.ReferenceFrame.FIXED);

//impostazione delle posizioni dei due osservatori
var observerPosF = {
  longitude: 12.0671 * SGP4.deg2rad,
  latitude: 44.20057 * SGP4.deg2rad,
```

```
  height: 1
};

var observerPosT = {
        longitude: 26.465951 * SGP4.deg2rad,
        latitude: 58.265651 * SGP4.deg2rad,
        height: 1
};

//creazione delle variabili per segnare il passaggio sulle stazioni di terra
var start_pasF = 0;
var stop_pasF = 0;
var start_pasT = 0;
var stop_pasT = 0;
var intervalF = new Cesium.TimeIntervalCollection();
var intervalT = new Cesium.TimeIntervalCollection();
var ElevationF=0;
var ElevationT=0;

//funzione printPosition()
function printPosition() {
  var property = new Cesium.SampledPositionProperty(
  Cesium.ReferenceFrame.INERTIAL);

  for (var i = 0; i <= tempo + 250; i += 30) {
    var time = Cesium.JulianDate.addSeconds(start, i, new Cesium.JulianDate());
      julianday = Cesium.JulianDate.totalDays(time);
      var positionAndVelocity = SGP4.propogate(eseoSatRec, julianday);
      var gmst = SGP4.gstimeFromDate(julianday);
      var geodeticCoordinates = SGP4.eciToGeodetic(positionAndVelocity.position, gmst);
      var longitude = SGP4.degreesLong(geodeticCoordinates.longitude);
      var latitude = SGP4.degreesLat(geodeticCoordinates.latitude);
      var height = geodeticCoordinates.height;
      var position = new Cesium.Cartesian3(
      positionAndVelocity.position["x"]*1000,positionAndVelocity.position["y"]*1000,
      positionAndVelocity.position["z"]*1000);
      property.addSample(time, position);
      posTraccia = new Cesium.Cartesian3.fromDegrees(longitude,latitude,0);
      traccia.addSample(time, posTraccia);
      var posPoli =  new Cesium.Cartesian3.fromDegrees(longitude,latitude,height*1000);
      poli.addSample(time, posPoli);

      var satEcf = SGP4.eciToEcf(positionAndVelocity.position, gmst);
      var lookAnglesF = SGP4.topocentricToLookAngles(SGP4.topocentric(observerPosF,
      satEcf));
      var oldElF = ElevationF;
      ElevationF = lookAnglesF.elevation * SGP4.rad2deg;
      if (oldElF<0 && ElevationF>0) {
        start_pasF=time;
      } else if (oldElF>0 && ElevationF<0) {
```

```
            stop_pasF=time;
            var interF = new Cesium.TimeInterval({
              start: start_pasF,
              stop: stop_pasF
            });
          intervalF.addInterval(interF);
          }
          var lookAnglesT = SGP4.topocentricToLookAngles(SGP4.topocentric(observerPosT,
          satEcf));
          var oldElT = ElevationT;
          ElevationT = lookAnglesT.elevation * SGP4.rad2deg;
          if (oldElT<0 && ElevationT>0) {
            start_pasT=time;
          } else if (oldElT>0 && ElevationT<0) {
            stop_pasT=time;
            var interT = new Cesium.TimeInterval({
              start: start_pasT,
              stop: stop_pasT
            });
          intervalT.addInterval(interT);
      }
    }
    return property;
}
```

E. Creazione delle entità

```
//CREA ESEO
var entity;
//CREA TRACCIA A TERRA
var tracciamento;

//ESEO
entity = viewer.entities.add({
  id: 'satellite',
  availability : new Cesium.TimeIntervalCollection([new Cesium.TimeInterval({
    start : start,
    stop : stop
  })]),
  name: 'ESEO',
  position : posizione,
  orientation : ori,
  model : {
    uri : '../Apps/SampleData/models/ESEO/ESEO.glb',
    minimumPixelSize : 500,
      maximumScale : 2000000,
      shadow: true
  },
```

```
    //Show the path as a red line sampled in 1 second increments.
    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)
    }
});
collection.add(entity);

//Creazione della traccia a terra
tracciamento = viewer.entities.add({
    id: 'track',
    availability : new Cesium.TimeIntervalCollection([new Cesium.TimeInterval({
      start : start,
      stop : stop
    })]),
    name: 'traccia a terra',
    position : traccia,
    point : {
      pixelSize : 10,
      color : Cesium.Color.YELLOW,
      outlineColor : Cesium.Color.GREENYELLOW,
      outlineWidth : 5
    },
    path : {
      resolution : 1,
      material : new Cesium.PolylineGlowMaterialProperty({
        color : Cesium.Color.YELLOW
      }),
      width : 5
    }
});

entity.position.setInterpolationOptions({
    interpolationDegree : 5,
    interpolationAlgorithm : Cesium.LagrangePolynomialApproximation,
});
tracciamento.position.setInterpolationOptions({
```

```
   interpolationDegree : 5,
   interpolationAlgorithm : Cesium.LagrangePolynomialApproximation,
});
}

//Stazione di terra FORLI'
var GroundStationF = viewer.entities.add({
   id : 'GSF',
   name : 'Ground Station Forlì',
   position : Cesium.Cartesian3.fromDegrees(12.0671, 44.20057,1),
   billboard :{
         image : '../Apps/SampleData/kml/facilities/GroundStation.png',
          verticalOrigin : Cesium.VerticalOrigin.BOTTOM,
          scale: 0.8
      },
   label : {
      text : 'Forlì GS',
      font : '12pt monospace',
      style: Cesium.LabelStyle.FILL_AND_OUTLINE,
      outlineWidth : 2,
      verticalOrigin : Cesium.VerticalOrigin.BOTTOM,
      horizontalOrigin : Cesium.HorizontalOrigin.LEFT,
      pixelOffset : new Cesium.Cartesian2(-30, -60)
   }
});
collection.add(GroundStationF);
```

F. Codice per la creazione dei passaggi

```
//Esempio per la ground station di forlì
pasF = viewer.entities.add({
   id: 'pasF',
   //Set the entity availability to the same interval as the simulation time.
   availability : intervalF,
   name: 'pasF',
   //Use our computed positions
   position : poli,
   polyline: {
      followSurface: false,
      positions: new Cesium.PositionPropertyArray([
         new Cesium.ReferenceProperty(collection,'GSF',['position' ]),
         new Cesium.ReferenceProperty(collection,'pasF',[ 'position' ])
      ]),
      width : 5,
      arcType : Cesium.ArcType.NONE,
      material : new Cesium.PolylineArrowMaterialProperty(Cesium.Color.CYAN)
   }
});
collection.add(pasF);
```