

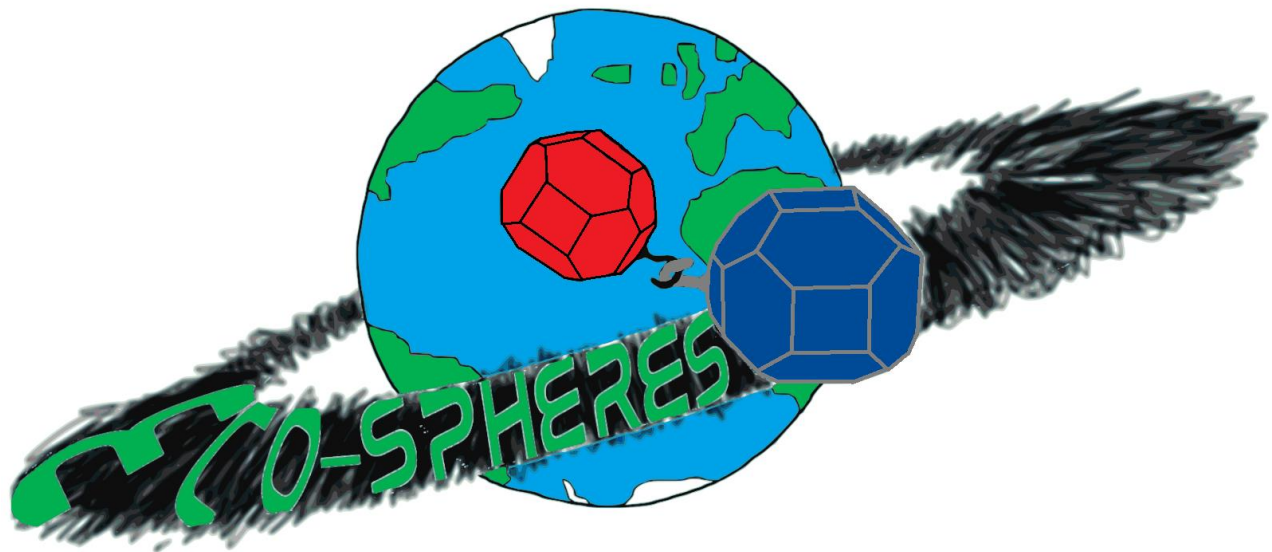
# ZERO ROBOTICS

---

## HIGH SCHOOL 2018

# GAME MANUAL

Version: AL-ISS 1.1.0 - Italian



**ECO-SPHERES**

*Evade, Capture, deOrbit space debris Spheres Program*

TO: ZERO ROBOTICS HEADQUARTERS



INCOMING

SOS

The following transmission was received just moments ago:

<i>SOS</i>					
<i>RED</i>					
<i>SPHERES_15569824</i>					
<i>SOS</i>					
<i>Thruster_09</i>	....	<i>UNRESPONSIVE</i>			
<i>Thruster_12</i>	...	<i>UNRESPONSIVE</i>			
<i>Location_Data</i>	...	<i>UNKNOWN</i>			
<i>Radar</i>	...	<i>NULL</i>			
<i>Target_Location</i>	...	<i>LEO</i>	<i>SATELLITE</i>	<i>92F003</i>	<i>RETRIEVAL</i>
<i>Last_Target</i>	...	<i>GEO</i>	<i>SATELLITE</i>	<i>35B117</i>	<i>RETRIEVAL</i>
<i>SOS</i>					
<i>RED</i>					
<i>SPHERES_15569824</i>					
<i>SOS</i>					

The Red SPHERES Satellite is in trouble.

Increasing numbers of satellites are being deployed to Low Earth Orbit (LEO) to study Earth's atmosphere, climate, land, oceans, and weather. To protect the success of this important research the SPHERES program is working with space agencies internationally to identify and remove space debris from LEO. Recently the SPHERES program deployed ECO-SPHERES as a part of its Evade, Capture, de-Orbit (ECO) initiative to remove debris. The ECO-SPHERES design includes a hook for the SPHERES to use to tow any type of cargo.

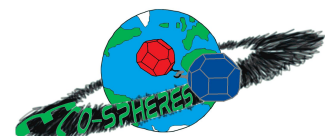
By its very nature, capturing and removing junk puts expensive debris removal satellites in harm's way. Any clean-up method is prone to damage. As you heard in the transmission above, one of our SPHERES Satellites has been damaged by debris and we need to retrieve it. The mission ahead of us, however, is dangerous. We must deploy another ECO-SPHERES to traverse the crowded LEO and, instead of moving debris, it needs to hook onto the damaged red SPHERES Satellite and bring it back to safety.

Program your satellite to avoid debris while trying to locate the red SPHERES. Use expert geometry to latch onto the damaged satellite, and bring it back to safety, taking into account the momentum of your precious cargo as to not further damage it.

Good luck to all participating space engineers.

Alvar  
SPHERES

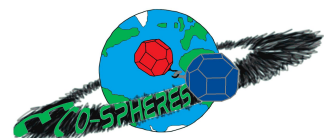
Saenz-Otero  
Lead





# Table of Contents

- 1 Game Overview
- 2 Game Rules/Phases of Game Play
  - 2.1 Playing Field
    - 2.1.1 Initial Position
    - 2.1.2 SPHERES “Pointing Face” for ECO-SPHERES
  - 2.2 Debris Field Navigation
    - 2.2.1 Bounds of Debris Field
    - 2.2.2 Size and Position of Space Debris
    - 2.2.3 Collision with Space Debris
    - 2.2.4 Cancel Debris Collision Damage tool (for test only)
  - 2.3 Rendezvous
    - 2.3.1 Motion of Target SPHERES
    - 2.3.2 Criteria for Successful Rendezvous
  - 2.4 Hooking
    - 2.4.1 Position and Motion of Target SPHERES during Hooking
    - 2.4.2. Hooking Criteria
    - 2.4.3 Unhooking
  - 2.5 Towing/Return of Target SPHERES to “Safe Zone”
  - 2.6 Scoring
    - 2.6.1 Time Dependent Scoring
    - 2.6.2 Bonus Points
      - 2.6.2.1 Rendezvous Pointing Bonus Points
      - 2.6.2.2 Target Distance Bonus Points
    - 2.6.3 Scoring Equation
  - 2.7 End of game
  - 2.8 SPHERES Satellites





2.8.1 Fuel

2.8.2 Collision Avoidance

2.8.3 Code Size

2.8.4 Noise

3 Game API

4 The Leaderboard for ECO-SPHERES

4.1 Introduction

4.2 Daily Rankings

4.3 Elimination Rankings

4.4 Tips

5 Tournament Structure

5.1 Semifinal Simulation Competition

5.2 Final Competition

5.5.1 Overview and Objectives

6 Season Rules

6.1 Tournament Rules

6.2 Ethics Code

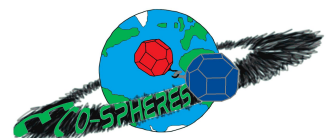
7 Appendix: Euler Angles Explanation

Why do you need Euler Angles for Eco SPHERES?

How to think about Euler Angles

Potential issues with Euler Angles (aka, why you may want to use Quaternions)

8 Revision History





# 1 Game Overview

Each team will compete individually by programming one SPHERES ("player SPHERES") to interact with a pre-programmed SPHERES ("target SPHERES"). The game consists of four phases: Debris Field Navigation, Rendezvous, Hooking, and Towing/Return Target.

## **Debris Field Navigation**

The player SPHERES must navigate through a field of debris, thruster damage will be incurred for collisions with debris.

## **Rendezvous**

The player SPHERES must approach the target SPHERES and meet the "rendezvous conditions." Once "rendezvous conditions" are met, all Space Debris will cease to exist and the target SPHERES will stop moving and actively hold position with its thrusters.

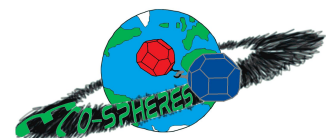
## **Hooking**

The player SPHERES will then attempt to capture the target SPHERES with its hook.

## **Towing/Return Target**

The player SPHERES must then get the target SPHERES back to the "safe zone".

Each team will compete to have the most points when the game time is up. Each game lasts 210 seconds for all phases of the competition.





## 2 Game Rules/Phases of Game Play

In order to be victorious over other participating teams, each player SPHERES should carefully navigate the debris field to rendezvous and hook the target SPHERES and return it to safety all while managing fuel, time, and their location on the gameplay area.

***Note: Satellite position relative to all game features is determined by the location of the center of the satellite as stored in the state variables (ZR state or SPHERES state) unless indicated otherwise.***

### 2.1 Playing Field

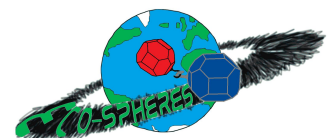
The playing field is determined by the size of the area available for the SPHERES satellites to move inside the International Space Station. The simulation creates an *interaction zone* of the same size. If players leave the Interaction Zone, they are considered out of bounds.

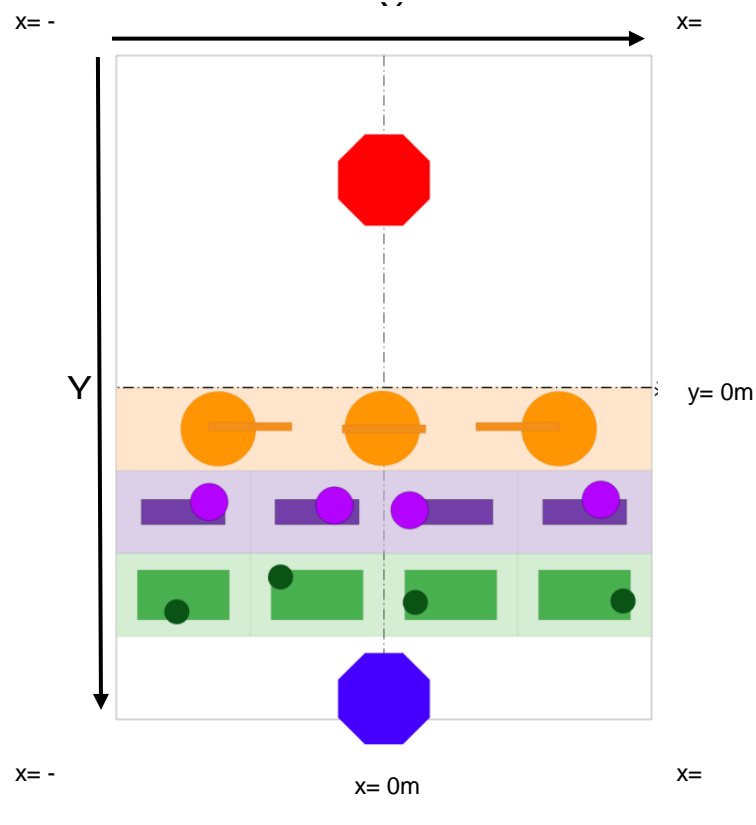
The Interaction Zone for the game has the following dimensions:

Interaction Zone Dimensions

3D	
$X [m]$	$[-0.64 : +0.64]$
$Y [m]$	$[-0.80 : +0.80]$
$Z [m]$	$[-0.64 : +0.64]$

A player will be penalized 1% of its fuel allocation for every second it remains out of these bounds.





Playing Field Area and Interaction Zone Dimensions

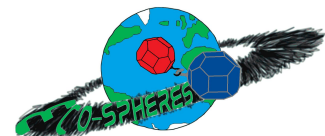
The Playing Field Area and Interaction Zone Dimensions are shown in the Figure above. The Debris Field is represented by the green, purple and orange circles. At the start of the ECO-SPHERES game, the Player SPHERES ( Blue SPHERES) is on one side of the Debris Field and the Target SPHERES (Red Spheres) is on the other side of the Debris Field.

### 2.1.1 Initial Position

The Blue and Red SPHERES satellites are deployed to the same initial position every game as follows:

Initial Positions

	3D
<i>Player SPHERE (Blue)</i>	
<i>X [m]</i>	<i>0.0</i>
<i>Y [m]</i>	<i>0.75</i>
<i>Z [m]</i>	<i>0.0</i>





<i>Target SPHERE (Red)</i>	
<i>X [m]</i>	<i>0.0</i>
<i>Y [m]</i>	<i>-0.5</i>
<i>Z [m]</i>	<i>-0.2</i>

The satellite radius is 0.11m.

### 2.1.2 SPHERES “Pointing Face” for ECO-SPHERES

Since the hooks used in the ECO-SPHERES game mount on the +X face (docking face) of the SPHERES both satellites are configured with the +X face as the pointing face for the ECO-SPHERES game. This means that the `setAttitudeTarget()` function will determine the pointing direction of the hook of the satellite on the +X face.

(This is as compared to Free Mode which assigns the -X face of the SPHERES as the pointing face when using `setAttitudeTarget()`)

## 2.2 Debris Field Navigation

During the first phase of ECO-SPHERES, the player SPHERES must carefully navigate through a Debris Field in order to reach the Target SPHERES. Collisions with debris will result in thruster damage.

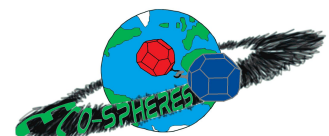
### 2.2.1 Bounds of Debris Field

Space Debris obstacles are located in range  $0m < y < 0.7m$

Parameter	Description	Value (m)
<i>y<sub>debris,start</sub></i>	<i>y value of start of debris field</i>	<i>0.7</i>
<i>y<sub>debris,end</sub></i>	<i>y value of end of debris field</i>	<i>0.0</i>

### 2.2.2 Size and Position of Space Debris

In the 3D game there are 41 Debris obstacles located within the Debris Field. Obstacles occur in the three sizes shown in the table below. Use the Function: `game.getDebris` to retrieve information about debris position and size.





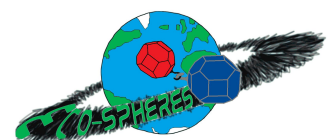


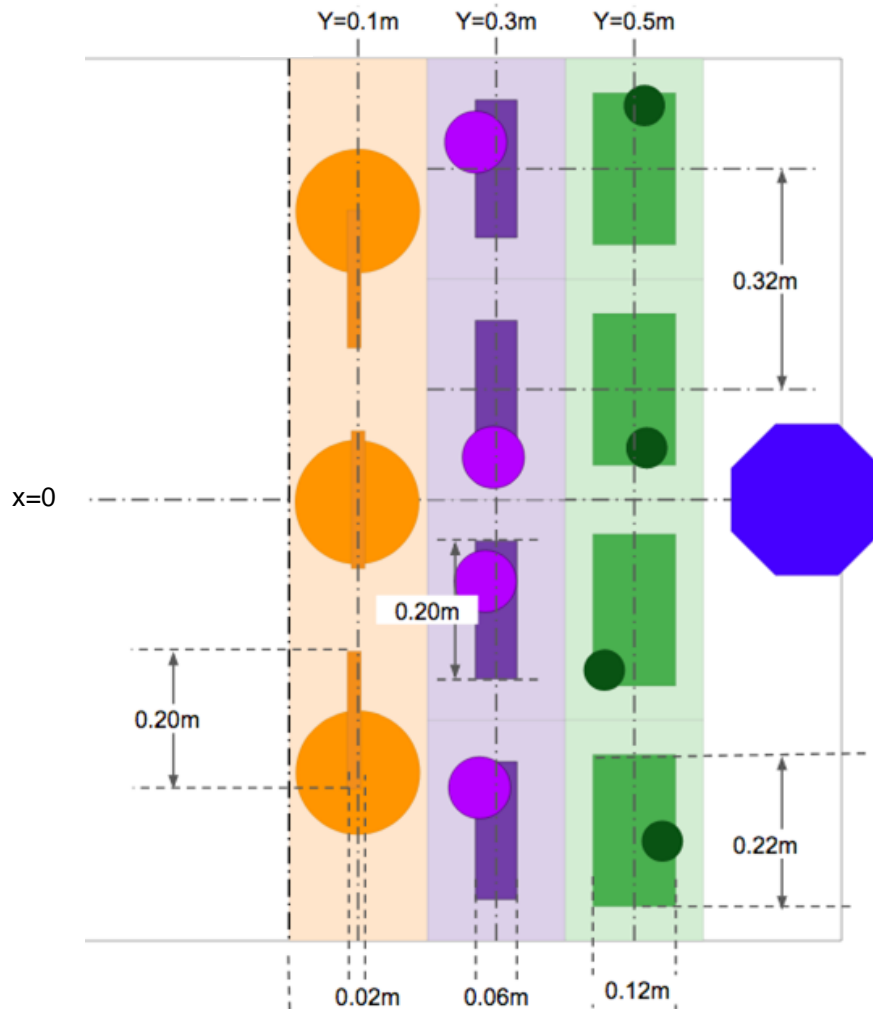
Debris Type	radius(m)
<i>Small</i>	<i>0.03</i>
<i>Medium</i>	<i>.045</i>
<i>Large</i>	<i>.09</i>

Space Debris Sizes

Game	Number
<i>3D</i>	<i>41</i>

Number of Debris

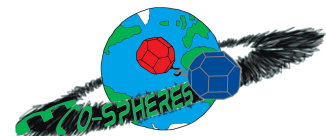




Possible locations of Space Debris within Debris Field  
(represented by orange, purple, green rectangles)

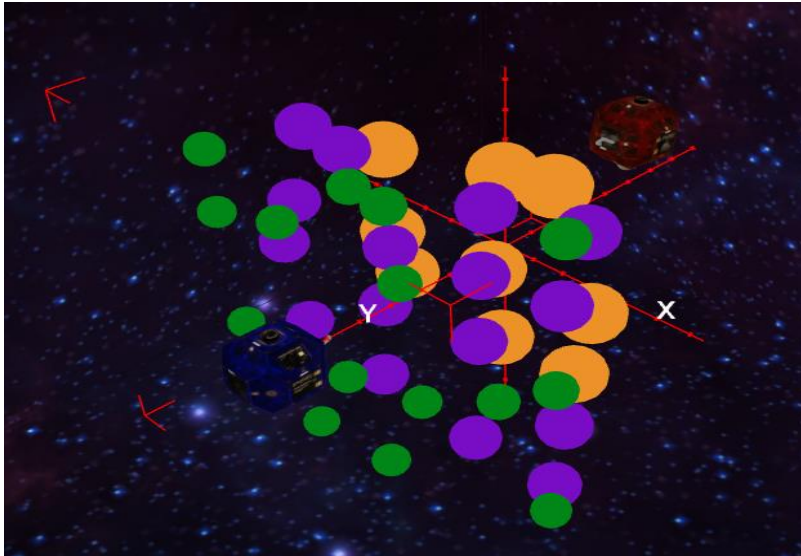
The three different sizes of debris occur in 3 zones centered on  $y=0.1\text{m}$  (large debris),  $y=0.3\text{m}$  (medium debris) and  $y=0.5\text{m}$  (small debris) as shown in the figure above. The orange, purple and green rectangles (with specified width and length) in the figure above reflect the possible locations for debris within each zone. The debris position is specified by the location of the debris center.

The same geometry is repeated about the Z axis, such that there are 4x4 small debris, 4x4 medium debris, and 3x3 large debris (41 total debris parts). The exact location is also random about the Z axis in the same way as in the X axis. An example of a 3D Debris Field is shown in





the figure below.

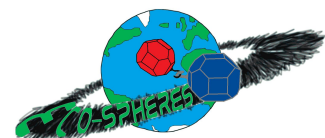


Example of 3D Debris Field

### 2.2.3 Collision with Space Debris

Collision with debris results in thruster damage, which reduces the maximum translational and rotational velocity of the SPHERES. The damage incurred depends on the size of the debris as shown in the table below. If the player SPHERE is out of bounds and it crosses into any of the debris zone areas (small:  $0.60\text{m} < Y < 0.40\text{m}$ ; medium  $0.40\text{m} < Y < 0.20\text{m}$ ; large  $0.20\text{m} < Y < 0.00\text{m}$ ) this will incur thruster damage equivalent to collision with one of the Debris located in that region.

Debris Type	3D
<i>Effect</i>	<i>Decrease in total thrust power (position and attitude) as a % of original thrust.</i>
<i>Small</i>	<i>10%</i>
<i>Medium</i>	<i>25%</i>
<i>Large</i>	<i>50%</i>





*Note: collisions are determined numerically solely by the distance between the center of the satellite and the center of the debris assuming perfect spheres. The hook is not considered for debris collision purposes.*

## 2.2.4 Cancel Debris Collision Damage tool (for test only)

In order to facilitate concurrent development of code for all phases of the ECO-SPHERES game a flag has been added to the simulation window which can be set to allow the player satellite to pass through the debris field without thruster damage. Look for the “Cancel Debris” field at the bottom of the simulation window, which can be toggled between 0 and 1.

0 = incurs normal thruster damage (default)

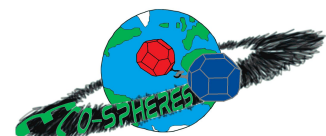
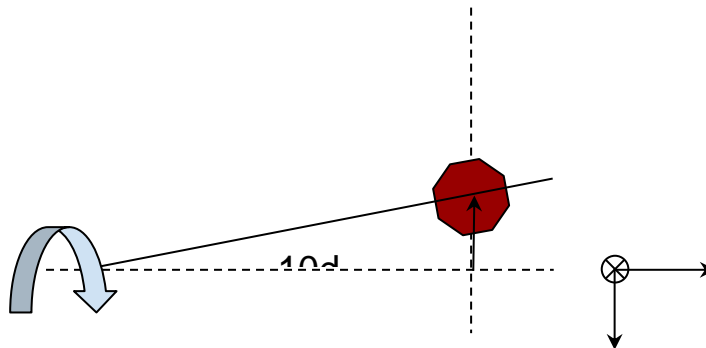
1 = cancels debris collision damage.

## 2.3 Rendezvous

During the second phase of the game the player SPHERES must rendezvous with the Target SPHERES.

### 2.3.1 Motion of Target SPHERES

In the 3D game the target SPHERES will always follow a “cone” trajectory, as if it were the end of a docking port on a spacecraft rotating off-center from the docking port axis. The angle of the cone is  $10^\circ$ . The figure below illustrates the motion of the target satellite at the start of each match.





### 2.3.2 Criteria for Successful Rendezvous

The "rendezvous position" will be defined by the following conditions being met: 1) the center to center separation of the SPHERES must be less than  $\epsilon_r$ , 2) the hook of the player satellite must be pointing at the center of the target satellite within  $\epsilon_{r,\theta}$ , 3) the player satellite must be moving with a speed (absolute value, not directional velocity) less than  $v_{max}$ , and 4) the player satellite must have a body rotation rate under  $w_{max}$ . Note that the two velocities are inertial, and not relative to the target satellite.

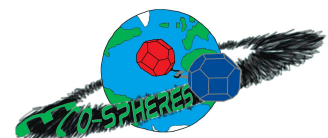
Call the function `game.completeRendezvous()` when conditions are met to complete this phase.

Parameter	Description	Value	Units
$\epsilon_r$	<i>max "center to center" distance for rendezvous</i>	0.40	m
$\epsilon_{r,\theta}$	<i>max angular tolerance for rendezvous</i>	0.052 3	rad deg
$v_{max}$	<i>maximum local speed of the player satellite</i>	0.005	m/s
$w_{max}$	<i>maximum body rotation rate of the player satellite</i>	0.0175 1	rad/s deg/s

Note that the Rendezvous criteria only checks that the player is pointing correctly to the target. However, successful hooking will also require the target to point correctly to the player. Because MIT controls the motion of the target, there is no requirement on the pointing of the target to the player. However, there is a scoring bonus if the target hook is pointing to the player or penalty if the target hook is pointing away from the player. The scoring section details this bonus/penalty.

## 2.4 Hooking

During the third phase of the game the player SPHERES must successfully Hook the Target SPHERES.



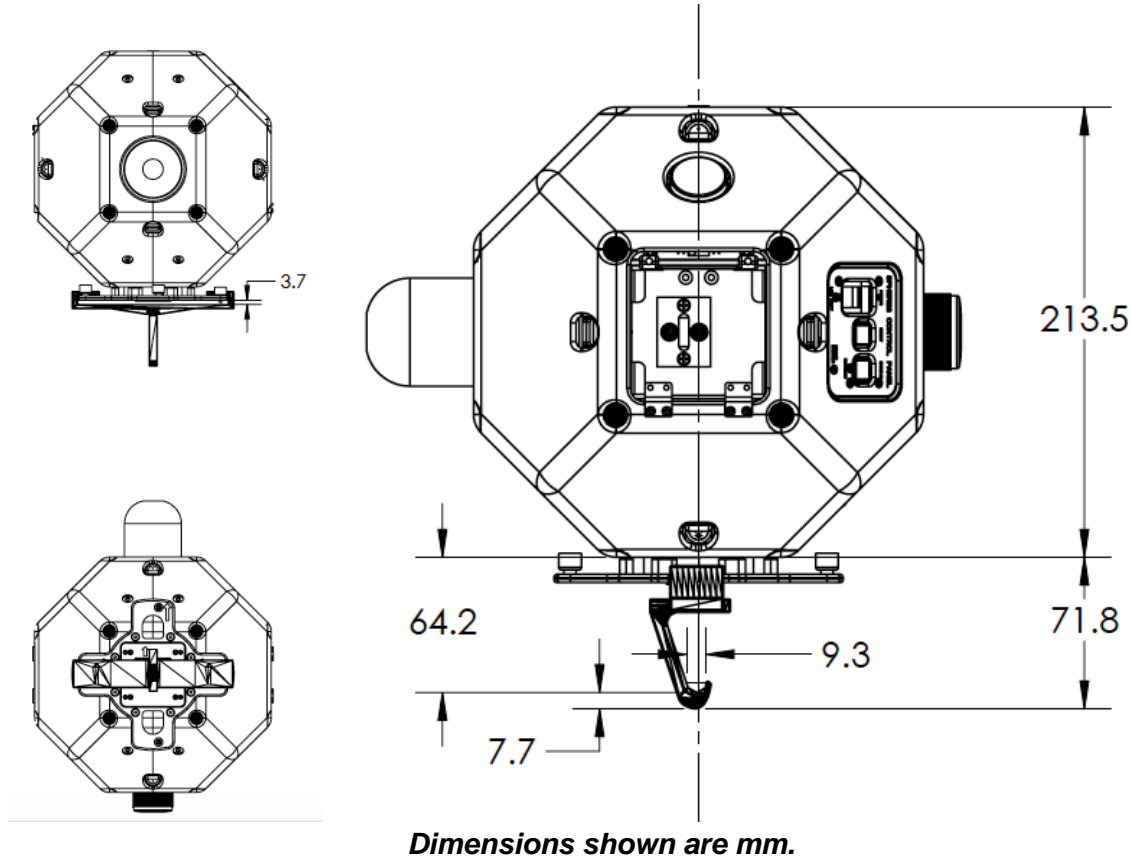


### 2.4.1 Position and Motion of Target SPHERES during Hooking

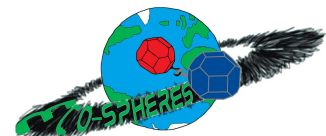
Once the rendezvous phase has been completed, all Space Debris will cease to exist and the target SPHERES will come to a stop as quickly as possible and actively hold at this “stop” position and attitude.

### 2.4.2. Hooking Criteria

The Player SPHERES must carefully position its hook into the hooking area of the Target SPHERES. The figure below shows the 3D printed hook assembled on the SPHERES. The Hook assembly attaches to the SPHERES docking face (+x face).



Key Details/Dimensions	Value
<i>Hook assembly attaches to the SPHERES docking face (+x face)</i>	N/A





<i>Distance between center of SPHERES and hooking center</i>	<i>0.17095m</i>
<i>Width of Hook</i>	<i>.008m</i>

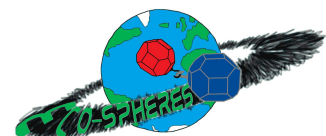
In order to simplify control of the player satellite, the two hooks are pre-oriented 90° relative to each other, such that the hooks can couple when the two satellites are oriented with the tank pointing in the same direction and the +X face pointing exactly opposite of each other.

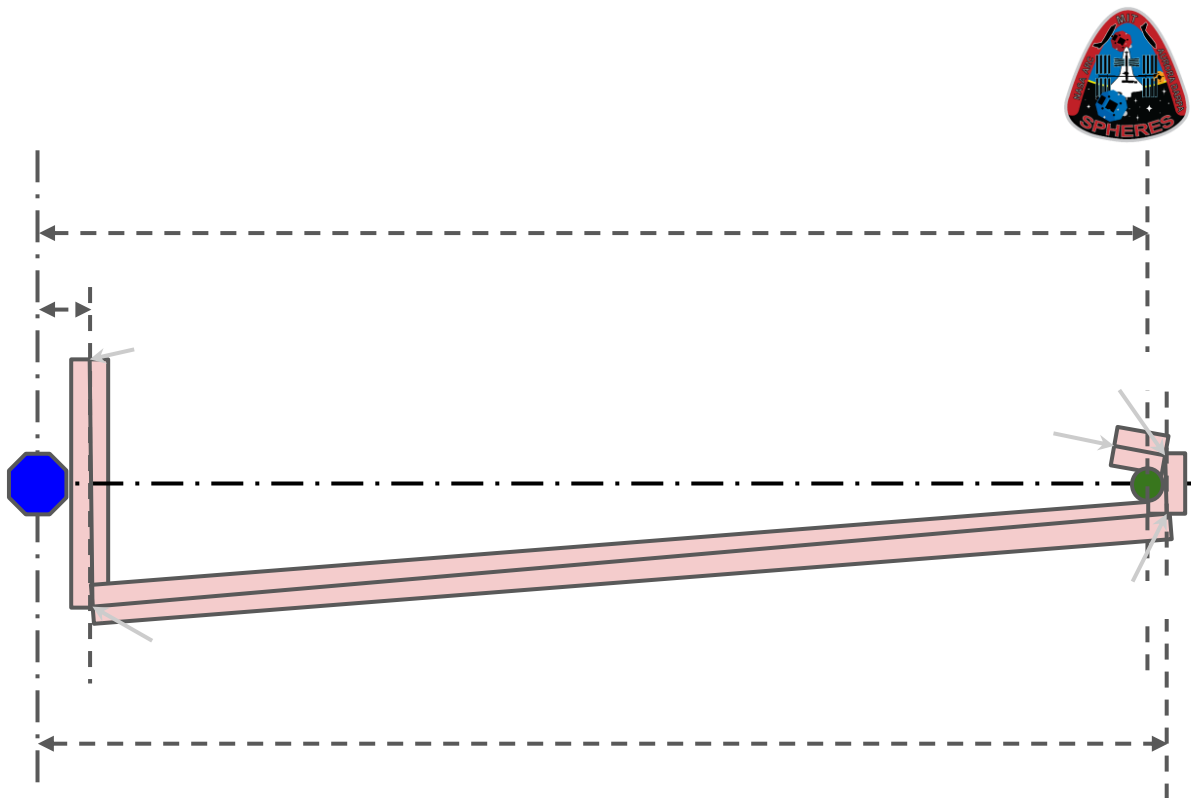
Successful hooking requires:

- The distance between the two “hooking centers” to be less than 0.005m
- The tanks (satellite Z Body Axis) to be aligned within 5°
- The other axes (satellite X & Y Body Axes) to be aligned within 120° (e.g., the hooks can hook while at 90° from each other as long as the tanks are aligned)
- The satellites must reach hooking distance without causing a collision.

During this stage the hook is modeled as rigid object, and collisions between the two hooks will result in forces and torque that override the SPHERES motion to simulate collisions in the real hardware. The override takes place for up to 5 seconds. Both satellites are affected by the motion. The target satellite will return to the “stop” position after the collision override. The player is responsible for the motion of the player satellite after collisions.

The hooks are modeled as cylinders of 0.0006m diameter as shown in the figure below. The red rectangles represent the cylinders, or the keep-out area to avoid collisions. The green dot at the tip of the hook is the “hooking center”. The satellites must make their “green hooking centers” overlap without causing overlap of the red hook cylinders.





Points in  $[X, Z]$  body coordinates

There are no specific requirements on the velocity of the satellites during hooking, but note that the velocity will account of the magnitude of collision forces in case the hooks collide instead of hooking successfully.

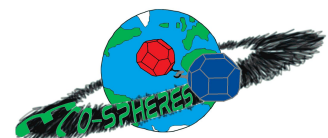
Hooking is determined automatically by the simulation; there is no need to call any functions.

Once hooking is successful the target satellite will completely stop using its thrusters and will free float for the remainder of the game. Its motion will be fully dependent on forces and torque imparted by the hook due to the motion of the player satellite.

NOTE: there is no simulation for SPHERES satellite collision detection/avoidance. This means in the simulation the satellites will be allowed to overlap in the same space, so teams are expected to not make any use of this simulation simplification towards their strategies. Any intentional use of satellite overlap in a strategy is against the spirit of the game.

### 2.4.3 Unhooking

The satellites will unhook when:







- The hooks are not under “tension” by pulling on each other (most unhooking will happen when the hooks loose tension).

**and** at least one of **three** conditions are met:

- The angles between the satellites grow beyond the hooking requirements (rotation about the X Body Axis misaligns beyond 20°, or rotation about the Y or Z Body Axes misalign beyond 120°)
- The relative velocity along the X Body Axis of the satellites (ie, the hook main axis) is greater than 0.005m/s.
- The virtual separation between the hook tips is beyond 0.04m (the separation is allowed to grow due to the lag in the simulation to create the hook forces)

Unhooking does not result in any extra motion or penalty. The target satellite will free-float with whatever motion it had at the time of un-hooking, and the player satellite can be controlled as normal.

## 2.5 Towing/Return of Target SPHERES to “Safe Zone”

The goal of the fourth and final phase of the game is for the Player SPHERES to return the Target SPHERES back to a “Safe Zone”, ie, as far as possible into the positive Y (+Y) half of the playing field.

During towing the hook dynamics create both forces (pulling along the +X, ±Y, and ±Z body axes) and torque (due to any misalignment of the main axis of the hooks). The hook does not impart forces along the -X body direction.

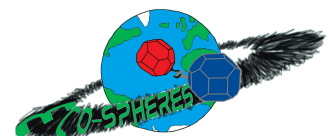
To complete the phase it is only necessary for the center of the Target satellite to go as far as possible into the +Y half of the playing field, regardless of its velocity (linear or angular) and hooking status.

## 2.6 Scoring

Scoring includes both Time Dependent Scoring and Bonus Point Scoring. The function `game.getScore` can be used to check player score.

### 2.6.1 Time Dependent Scoring

The primary way teams earn points is by completing different phases of the game: Debris Field Navigation, Rendezvous, Hooking, Return to Safe Zone.





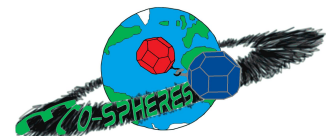
Each phase has a baseline point value earned when the phase is completed. Additionally, the faster teams complete each phase, the more points they will earn.

Phase	Time 3D	Min Score	Max Score
Debris Field	30s	2.5	5.0
Rendezvous	60s	2.5	5.0
Hooking	90s	5.0	10.0
Return Target	30s	5.0	10.0

### Time Dependent Scoring

The number of points awarded for each stage is determined by:

Symbol	Description
$p_{0,L}$	Minimum points awarded for crossing debris field
$p_{0,U}$	Points awarded for crossing debris field in 0 time
$\Delta T_{0,ref}$	Time for crossing debris field which causes minimum score
$p_{1,L}$	Minimum points awarded for achieving rendezvous
$p_{1,U}$	Points awarded for achieving rendezvous in 0 time
$\Delta T_{1,ref}$	Time for achieving rendezvous which causes minimum score
$p_{2,L}$	Minimum points awarded for achieving hooking
$p_{2,U}$	Points awarded for achieving hooking in 0 time
$\Delta T_{2,ref}$	Time for achieving hooking which causes minimum score
$p_{3,L}$	Minimum points awarded for returning target SPHERES
$p_{3,U}$	Points awarded for returning target SPHERES in 0 time
$\Delta T_{3,ref}$	Time for returning target SPHERES which causes minimum score





$$p_{n,L} + (p_{n,U} - \frac{p_{n,U} - p_{n,L}}{\Delta T_{n,ref}} \Delta T_n)(\Delta T_n < \Delta T_{n,ref})$$

where  $p_{n,L}$  is the baseline score earned when phase  $n$  is complete,  $p_{n,U}$  is the maximum possible score that could be earned (if the phase was completed in zero time),  $\Delta T_n$  is the time elapsed since the phase began, and  $\Delta T_{n,ref}$  is the time at which no more than the min score will be awarded for that phase.

## 2.6.2 Bonus Points

### 2.6.2.1 Rendezvous Pointing Bonus Points

At the end of the rendezvous phase (when the player calls `game.completeRendezvous` successfully) a bonus or penalty will be scored based on how close the hook of the **target** is pointing to the center of the **player**.

The bonus/penalty is assessed as follows:

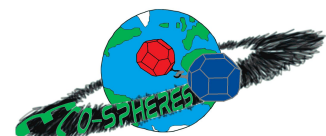
- Determine the “pointing vector” between the target to the player
- Take the *dot product* of the *normalized* “pointing vector” and the ZR attitude of the target (which is itself a normalized pointing vector)
- Square the *dot product* magnitude (which increases the effect of misalignment)
- Scale by 2.5 (so 2.5 is the maximum bonus and -2.5 the maximum penalty)

Symbol	Description	Max bonus	Max penalty
$b_r$	Rendezvous pointing bonus	2.5	-2.5

Note: The *dot product* of two *normalized* vectors returns 1 if the vectors are perfectly aligned, 0 if they are 90° offset, -1 if they are pointing away from each other, and a fraction less than 1 when they are partially aligned.

Note: the sign is maintained, only the magnitude is squared.

### 2.6.2.2 Fuel Bonus Points





### 2.6.2.2 Target Distance Bonus Points

At the end of the game a bonus point of up to 5 points will be given based on the distance that the Target has moved along the inertial Y direction towards the “safe zone” (positive Y). The bonus will be directly proportional between the total Y distance traveled up to  $Y=0.75$ :

$$b_d = (-\text{target}_{Y\_initial} + \text{target}_{Y\_final}) / 1.25 * 5 = (0.5 - \text{target}_{Y\_final}) / 1.25 * 5$$

NOTE: if the target is pushed “backwards” so that  $\text{target}_{Y\_final} < -0.5\text{m}$  a proportional **penalty** will be assessed.

### 2.6.3 Scoring Equation

Given the above information, the total score at the end of the game is calculated as follows:

$$\text{Score} = \sum_{n=1}^4 [p_{n,L} + (p_{n,U} - \frac{p_{n,U} - p_{n,L}}{\Delta T_{n,ref}} \Delta T_n)(\Delta T_n < \Delta T_{n,ref})] + b_r + b_d$$

*Note: during the Alliance and ISS phase there is a Collision **Penalty** (see section 2.8.2).*

## 2.7 End of game

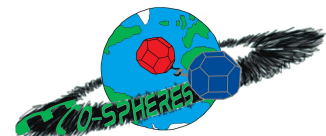
The game ends after 210 seconds. Final Score is calculated.

## 2.8 SPHERES Satellites

Each team will write the software to command a SPHERES satellite to move in order to complete the game tasks. A SPHERES satellite can move in all directions using its twelve thrusters. The actual SPHERES satellites aboard the ISS, like any other spacecraft, have limited fuel (in this case liquid carbon dioxide), power (in this case AA battery packs), and computer (a digital-signal processor). These resources are limited and must be used wisely. Therefore, the players of Zero Robotics are limited in the use of these resourced by virtual limits within the game. The use of batteries is limited by having a fixed game time of 210 seconds for all competition phases. The other limitations are detailed below.

The *nominal* properties of the SPHERES satellites are summarized as follows:

Property	Value	Units
<i>Radius</i>	<i>0.11</i>	<i>m</i>
<i>Diameter</i>	<i>0.22</i>	<i>m</i>





<i>Mass</i>	<i>4.0</i>	<i>kg</i>
<i>Single Thruster Force</i>	<i>0.11</i>	<i>N</i>

These are called *nominal* properties because they are not exact. The Mass changes up to 0.2kg as fuel is consumed. The single thruster force is affected by how many thrusters are open at one time, varying up to 20% of the nominal force. In addition, while every attempt was made to align the thrusters with the satellite body axes, there are imperfections in the alignment (within 2°), therefore not all the force goes in the exact desired direction.

### 2.8.1 Fuel

Each player is assigned a virtual fuel allocation of 60 seconds of total accumulated thruster firing time. This is calculated by summing individual thruster firing during the game. Once the allocation is consumed, the satellite will not respond to the player SPHERES control commands. It will fire thrusters only to avoid leaving the Interaction Zone or colliding with the other satellite.

Any action that requires firing the thrusters (rotating, accelerating, decelerating), whether it was commanded by the player, due to activate collision avoidance or out-of-bounds breaking, or other penalties of the game play, will consume virtual fuel allocation.

The function `getFuelRemaining` can be called to obtain the fraction of fuel still available during a game. The function returns 1.00 for a full tank, and then a fractional value (e.g. 0.50 = half a tank) until reaching 0.0. Once the function returns 0.0 all user motion commands will be ignored.

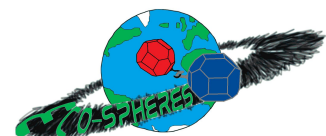
### 2.8.2 Collision Avoidance

*Collision Avoidance is not activated for this game*, since part of the goal of this year's game is to get close enough to hook. This means that in the simulation the satellite **will** be able to go across each other, and the simulation will **not** stop what would be real collisions aboard ISS. It is up to each team/alliance that advances to the finals that their code will not result in actual physical collisions (as much as possible).

During the finals the simulation will penalize collisions in any of the following cases:

- A hook tip comes within the SPHERES radius
- The two satellites come too close to each other

If any of those cases occur, there will be a 0.25 points penalty per second that the collision takes place. A total of 0.25 points will be deducted, regardless of how many of the conditions are met.





**NOTE:** the score **can** go negative!

### 2.8.3 Code Size

A SPHERES satellite can fit a limited amount of code in its memory. Each project has a specific code size allocation. When you compile your project with the “Code Size Estimate” menu option, the compiler will provide the percentage of the code size allocation that your project is using. Formal competition submissions require that your code size be 100% or less of the total allocation.

### 2.8.4 Noise

It is important to note that the SPHERES simulations create noise similar to that experienced by the satellites aboard the ISS. This noise means two main things:

- The satellites will never know exactly where they are; their “estimate” of their location reflects that the sensors are not perfect, so at all times even if the satellite is supposed to be completely still, their location will vary approximately  $\pm 0.005\text{m}$  independently on every axis.
- The satellites will not thrust perfectly. While the thrusters use identical designs, each thruster has a slight variation in thrust, and the total thrust varies by the number of thrusters used at one time. This means that the thrust of the satellites may vary in general 10% and up to 20% in some cases.

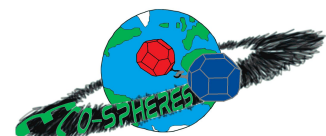
This is fully intended as part of the challenge and reflects uncertainties in real aerospace engineering systems, which have imperfect dynamic models, sensors, and actuators. The best performing solutions will be those that prove to be robust to these variations and a wide variety of different initial conditions of the gameplay features.

## 3 Game API

The following table lists the functions available in this year’s game. In order to use the game-specific functions, use the following syntax depending on the editor you choose:

C: `game.functionName(inputs)`, for example:  
`game.completeRendezvous()`

MATLAB: `game18.functionName(inputs)`, for example:  
`game18.checkRendezvous()`





For the general Zero Robotics functions use the syntax `api.functionName(inputs)`, for example:

```
api.setPositionTarget(posTarget)
```

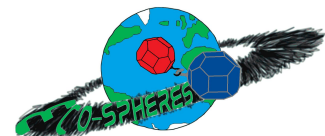
(unless they are math functions, which can be called without reference to the instance).

The generic ZR user API (in both C and MATLAB languages) is available at:

[http://static.zerorobotics.mit.edu/docs/tutorials/ZR\\_user\\_API\\_2017.pdf](http://static.zerorobotics.mit.edu/docs/tutorials/ZR_user_API_2017.pdf)

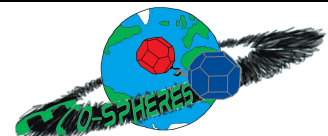
### ECO-SPHERES API Reference

Name	Description
C: void getDebris(float debris[NUM_DEBRIS][4])  MATLAB: debris = getDebris(NUM_DEBRIS)	Puts the following information into the array "debris": [debris[n][0], debris[n][1], debris[n][2]] = position of the debris Debris[n][3] = radius of the debris  The array must be of length "NUM_DEBRIS", which is: 3D Game: 41  It is the responsibility of the players to correctly size the array.
C: bool checkRendezvous()  MATLAB: result = checkRendezvous()	Returns true if the Rendezvous requirements are met. Returns false otherwise.
C: bool completeRendezvous()  MATLAB: result = completeRendezvous()	If the Rendezvous requirements are met, will complete the phase and command the target satellite to stop moving.  The first time this is called successfully is the scoring time for the Rendezvous phase; the pointing of the target satellite to the player at this instant is also used for bonus points.  Returns true if the Rendezvous completes successfully (or had already completed in the past); returns false if the requirements are not met.
C: bool getRendezvous()  MATLAB: result = getRendezvous()	Returns true if the rendezvous phase is complete.
C: bool getHooked()  MATLAB: result = getHooked()	Returns true if the satellites are currently hooked.
C: bool getHookCollision()  MATLAB: result = getHookCollision()	Returns true if the simulation is currently overriding forces/torque due to collision of the hooks.
C: int getThrusterHealth()  MATLAB: result = getThrusterHealth()	Returns the thruster health of the player satellite between 0 to 100 %.





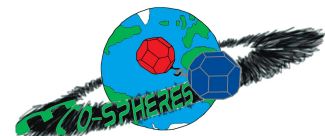
C: <code>int getGamePhase()</code>  MATLAB: <code>result = getGamePhase()</code>	Returns the phase of the player satellite. Phase changes when the player satellite meets the requirements for each phase. Phases cannot be completed out of order. <ul style="list-style-type: none"> <li>1 = Debris avoidance</li> <li>2 = Rendezvous</li> <li>3 = Hooking</li> <li>4 = Towing.</li> </ul>
C: <code>float getScore()</code>  MATLAB: <code>myScore = getScore()</code>	Returns the player's score.
C: <code>float getFuelRemaining()</code>  MATLAB: <code>fuel = getFuelRemaining()</code>	Returns remaining fuel as a fraction of total fuel allowed (1.00 = full tank; 0.50 = 50% remaining; 0.00 = empty tank).
C: <code>void getMyEulerState(float *state);</code>  MATLAB: <code>myState = getMyEulerState()</code>	Returns the state of the satellite representing the attitude with Euler angles of [roll, pitch, yaw] as follows:  <code>state[0][1][2] = pos X, pos Y, pos Z [m]</code> <code>state[3][4][5] = vel X, vel Y, vel Z [m/s]</code> <b><code>state[6][7][8] = roll, pitch, yaw [rad]</code></b> <code>state[9][10][11] = rate X, rate Y, rate Z [rad/s]</code>  The output variable must be a float array of length 12.  The rotations are defined as follows: <ul style="list-style-type: none"> <li>roll = rotation about the X axis in Radians</li> <li>pitch = rotation about the Y axis in Radians</li> <li>yaw = rotation about the Z axis in Radians</li> </ul>
C: <code>void getOtherEulerState(float *state);</code>  MATLAB: <code>myState = getOtherEulerState()</code>	Returns the state of the other satellite (the "target") representing the attitude with Euler angles of [roll, pitch, yaw] as follows:  <code>state[0][1][2] = pos X, pos Y, pos Z [m]</code> <code>state[3][4][5] = vel X, vel Y, vel Z [m/s]</code> <b><code>state[6][7][8] = roll, pitch, yaw [rad]</code></b> <code>state[9][10][11] = rate X, rate Y, rate Z [rad/s]</code>  The output variable must be a float array of length 12.
C: <code>void eulerToQuaternion(float* euler, float* quaternion);</code>  MATLAB: <code>quaternion = eulerToQuaternion(euler)</code>	Converts the given Euler angle array of [roll, pitch, yaw] into a quaternion.  <code>euler = [roll, pitch, yaw] (floats) [rad]</code> <code>quaternion = [q.x, q.y, q.z, w] (floats)</code>
C: <code>void quaternionToEuler(float* quaternion, float* euler);</code>  MATLAB: <code>euler = quaternionToEuler(quaternion)</code>	Converts the given quaternion [q.x, q.y, q.z, w] into an Euler angles rotation of [roll, pitch, yaw].  <code>quaternion = [q.x, q.y, q.z, w] (floats)</code> <code>euler = [roll, pitch, yaw] (floats) [rad]</code>
C: <code>void setEulerTarget(float* euler);</code>  MATLAB: <code>setEulerTarget(euler)</code>	Sets the target attitude of the satellite to point towards the [roll, pitch, yaw] angles commanded in the input Euler angles vector.







	euler = [roll, pitch, yaw] (floats)
<pre>C: void enableHookGT(); void disableHookGT();</pre> <p><b>MATLAB:</b></p> <pre>enableHookGT() disableHookGT()</pre>	<p>These functions enable and disable “detailed” Game Trace messages that help while the game is in phase 3 (hook) or 4 (tug) to understand how the simulation is determining the hook status and actions. The following GT messages will appear every second, in addition to the regular traces:</p> <p><i>NOTE: these tests are done 10 times per second, interpolating between the previous position and the current position, so there are up to 10 of these traces each loop cycle.</i></p> <p>- HOOK too FAR (i): actual_dist &gt; ref_dist {y_diff = [ active_y - target_y = y_diff]}</p> <p>where:</p> <ul style="list-style-type: none"> <li>- actual_dist is the 3d distance between the two hook contact points</li> <li>- ref_dist is the reference distance between hooks needed to achieve hooking</li> <li>- y_diff, active_y, target_y are only the Y distances of the difference, which should account for most of the distance when the hook is approximately aligned with the Y axis (as in the initial conditions)</li> </ul> <p>- Hook angles test TOO LARGE:: alpha = [bool] = [alpha_act &gt; alpha_ref] beta = [bool] = [beta_act &gt; beta_ref] gamma = [bool] = [gamma_act &lt; gamma_ref]</p> <p>- Hook angles test OK: alpha = [bool] = [alpha_act &gt; alpha_ref] beta = [bool] = [beta_act &gt; beta_ref] gamma = [bool] = [gamma_act &lt; gamma_ref]</p> <p>where:</p> <ul style="list-style-type: none"> <li>- [alpha, beta, gamma] are the Euler angles rotations between the satellites</li> <li>- for hooking there are three tests to determine if things <b>UNhook</b> (ie, true if they will <b>UNhook</b>, false if angles are correct for hooking; all three tests must be false for hooking to either take place or to remain hooked): <ul style="list-style-type: none"> <li>- alpha [bool] = alpha_act &gt; alpha_ref: alpha angle in radians (about the red X axis), which should be near 0</li> <li>- beta [bool] = beta_act &gt; beta_ref: beta angle in radians (about the red Y axis), which should be near 0</li> <li>- gamma [bool] = gamma_act &lt; gamma_ref: gamma angle in radians (about the red Z axis), which should be near PI</li> </ul> </li> </ul> <p>- Relative velocity 1vs2 X body axis = vel_x_rotSat2 &lt; max_vel_x</p> <p>- Relative velocity 2vs1 X body axis = vel_x_rotSat1 &lt; max_vel_x</p>





	<p>where</p> <ul style="list-style-type: none"> <li>- <code>vel_x_rotSat1</code> and <code>vel_x_rotSat2</code> are the relative X velocities, in blue and red body coordinates respectively, between the two satellites, which must remain small for the satellites to remain hooked</li> </ul> <p><i>NOTE: the following GT appears only once per loop cycle</i></p> <p>- Hook collision force override: <code>F[3]</code></p> <p>where:</p> <ul style="list-style-type: none"> <li>- <code>F[3] = [Fx, Fy, Fz]</code> is the new force commands in the inertial reference frame sent to the satellite when hook collisions cause the game to override any user commands (in [N])</li> </ul>
<pre>C: void enableTugGT(); void disableTugGT();  MATLAB: enableTugGT() disableTugGT()</pre>	<p>- Hook Dynamics: <code>forces = [F<sub>req</sub> + F<sub>aug</sub> = F<sub>out</sub>]</code>, <code>torque = [T<sub>req</sub> - T<sub>aug</sub> = T<sub>out</sub>]</code></p> <p>where:</p> <ul style="list-style-type: none"> <li>- <code>F<sub>req</sub></code>, <code>T<sub>req</sub></code> are the requested 3D forces/torque by the user</li> <li>- <code>F<sub>aug</sub></code>, <code>T<sub>aug</sub></code> are the augmented 3D forces/torque calculated by the hook dynamics</li> <li>- <code>F<sub>out</sub></code>, <code>T<sub>out</sub></code> are the final 3D forces/torque requested to the satellite control system</li> <li>- The individual components of the forces/torque are shown as [X, Y, Z]</li> <li>- Forces are in [N] and torque are in [Nm]</li> </ul>

## 4 The Leaderboard for ECO-SPHERES

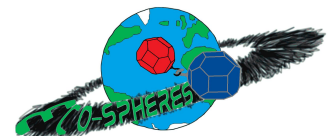
### 4.1 Introduction

This year's leaderboard has been modified to support single player games.

The Leaderboard calculates rankings daily from the beginning of a competition until the submission deadline. Results from prior days have no impact on current days rankings. Only the final standings on the Leaderboard at the end of each competition phase will determine which teams advance to the next phase.

### 4.2 Daily Rankings

Each day, at 21:59:59 UTC, (except on competition deadlines where posted times apply) your most recently submitted code is collected by an automated system and played as the Blue





SPHERES. Each team's code is played a total of ten (10) times. Scores from the 10 games are averaged to determine rankings. Results are posted on the website after rankings are complete. While the daily competition is ongoing the leaderboard will not be visible and clicking on the Leaderboard will return the message: "Results will be posted once the daily leaderboard run is complete". Multiple games are completed to provide teams with more game data given the random elements in the game. Ranking data is refreshed daily.

## 4.3 Elimination Rankings

On the last day of each competition period, each team's code will be run  $(10 + n)$  times until the ranking stops changing or  $n=15$  (every team has played 25 games), whichever comes first. Each iteration every teams' top 10 scores (only) will be averaged to determine rank.

## 4.4 Tips

The team with the highest average score will be rank 1 on the leaderboard. The best way to improve your rank is the most logical way: keep working at your algorithms. There are no surefire alternatives. Also, don't let fear of a bad game keep you from submitting early. Results of past competitions demonstrate that teams that make many submissions tend to perform better.

Finally, it is important to note that ranking data is refreshed and teams start from scratch daily.

# 5 Tournament Structure

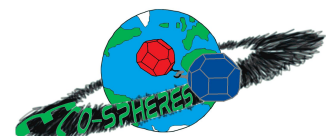
## 5.1 Semifinal Simulation Competition

All teams that complete a valid registration are eligible to participate in the semifinal phase of the Italian Championship.

## 5.2 Final Competition

The top 6 teams on the leaderboard at the end of semifinal play will advance to the Finals Competition.

The finals will take place **in simulation** during an event which will be held in April in Sicilia. All finalists will be invited to participate at the final event.





### 5.5.1 Overview and Objectives

## 6 Season Rules

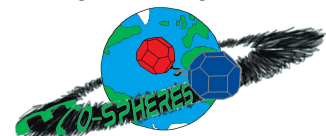
### 6.1 Tournament Rules

All participants in the Zero Robotics Italian Tournament 2018 must abide by these tournament rules:

- The Zero Robotics team (MIT / Aurora/ ILC/ Italian Guide Committee) can use/reproduce/publish any submitted code.
- In the event of a contradiction between the intent of the game and the behavior of the game, MIT or Italian Guide Committee will clarify the rule and change the manual or code accordingly to keep the intent.
- Teams are expected to report all bugs as soon as they are found.
- A “bug” is defined as a contradiction between the intent of the game and behavior of the game.
  - The intent of the game shall override the behavior of any bugs up to code freeze.
  - Teams should report bugs through the online support tools. ZR reserves the right to post any bug reports to the public forums (If necessary, ZR will work with the submitting team to ensure that no team strategies are revealed).
- Code and manual freeze will be in effect 3 days before the submission deadline of a competition.
- Within the code freeze period the code shall override all other materials, including the manual and intent.
- There will be no bug fixes during the code freeze period. All bug fixes must take place before the code freeze or after the competition.
- Game challenge additions and announcement of TBA values in the game manual may be based on lessons learned from earlier parts of the tournament.

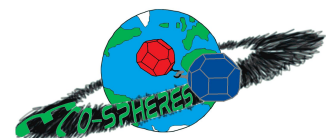
### 6.2 Ethics Code

- The ZR team will work diligently upon report of any unethical situation, on a case by case basis.
- Teams are strongly encouraged to report bugs as soon as they are found; intentional abuse of an unreported bug may be considered as unethical behavior.
- Teams shall not intentionally manipulate the scoring methods to change rankings.





- Teams shall not attempt to gain access to restricted ZR information.
- We encourage the use of public forums and allow the use of private methods for communication.
- Vulgar or offensive language, harassment of other users, and intentional annoyances are not permitted on the Zero Robotics website.
- Code submitted to a competition must be written only by students.
- Players may not access the implementation instance of the game or modify any variables of the object. In particular, the api and game objects should not be duplicated or modified in any capacity.
- Simulation requests may only be done manually via the website interface, API calls for simulation are not allowed (even if doable).
- Comments may not be used to add executable code in order to bypass the codesize limits or for any other reason not explicitly approved by the ZR team.





## 7 Appendix: Euler Angles Explanation

Eco SPHERES introduces the use of Euler Angles to control the attitude of the SPHERES. Like the `api.setAttitudeTarget` and `api.setQuatTarget` functions, the new `game.setEulerTarget` function controls the direction in which the satellite points. Only one of those functions should be called in each “loop” cycle, since they all try to control the same property (attitude) of the satellite<sup>1</sup>.

### Why do you need Euler Angles for Eco SPHERES?

The commonly used `api.setAttitudeTarget` function represents the “pointing” of the satellite using a “pointing unit vector”. It indicates where the *reference axis* of the satellite points. In the case of EcoSPHERES the *reference* is the +X body axis of the satellite, which is the same body axis as where the hook is. However, the *pointing unit vector* does not control rotation *about* the reference axis, in this case the hook. This means that the satellite can rotate *about* the body axis any amount, and still be pointing in the same valid *pointing unit vector*. The following figures illustrate this (pay attention to the pointing vectors [nx, ny, nz] in the state information box, and the resulting pointing in the visualization):

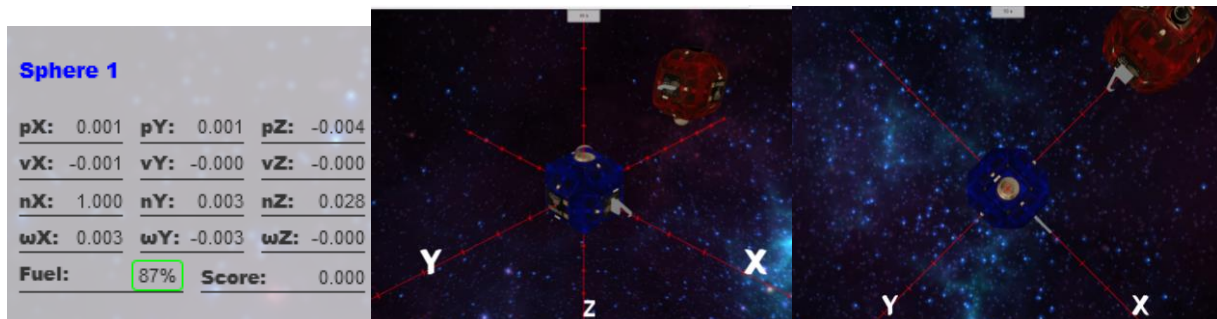
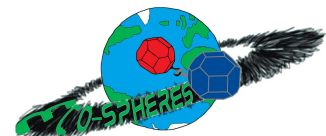


Figure 7.1 - Blue satellite pointing to “origin” pointing vector [1,0,0] (note the X,Y,Z axes orientation like in the game, with Z pointing “down”). In this “initial condition” case, the body axes of the SPHERES satellite are completely aligned with the inertial axes of the game field. This is the “origin” of all the rotations for SPHERES.

<sup>1</sup> For completeness, it must be understood that, only one of [`api.setAttitudeTarget`, `api.setQuatTarget`, `game.setEulerTarget`, `api.setAttRateTarget`, `api.setTorques`] should be used in each loop cycle, since they all need to use the same set of thruster commands to actually move the satellite. Any mixed use of those functions will result in unknown behavior.





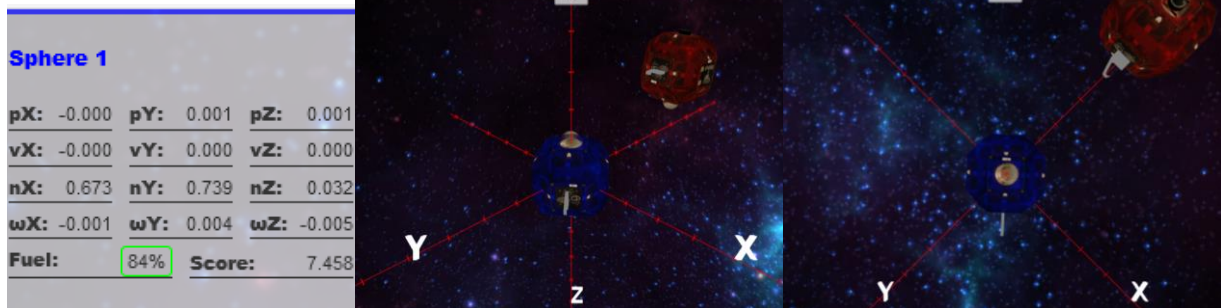


Figure 7.2 - Blue satellite pointing its +X face to  $[0.7071, 0.7071, 0]$  unit vector (in this case due to noise to  $[0.673, 0.739, 0.032]$ , but close enough to the intended angle).

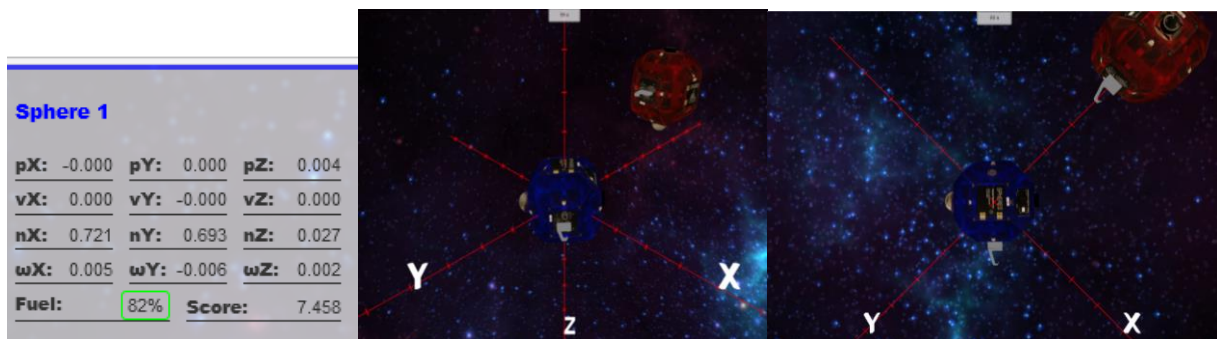


Figure 7.3 - Blue satellite still has a pointing unit vector of approximately  $[0.7071, 0.7071, 0]$ . However, its tank is now pointing “sideways” along on the XY plane, instead of “up down” on the Z axis. The same pointing unit vector allows different attitudes of the satellite, and it is not possible to control them.

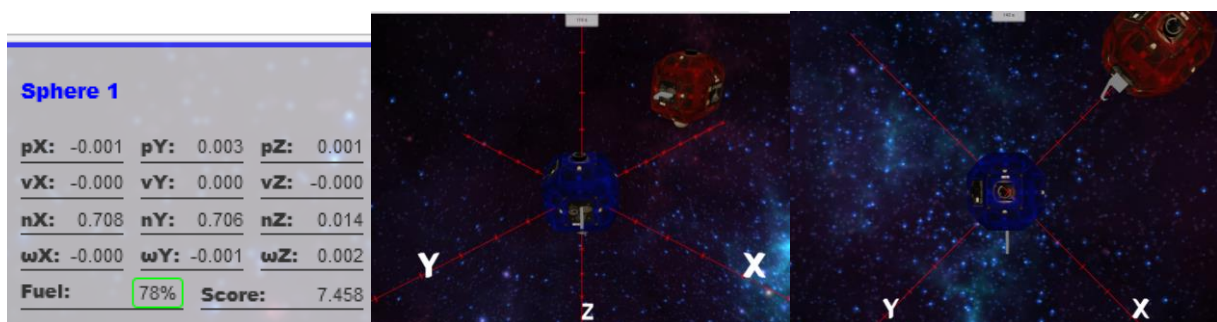
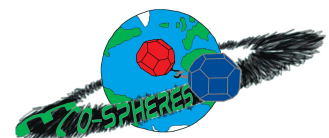


Figure 7.4 - Blue satellite still has a pointing unit vector of approximately  $[0.7071, 0.7071, 0]$ , but now its tank is pointing opposite of the original orientation.





## How to think about Euler Angles

Euler angles represent rotations about the **reference** axes in a pre-specified order, in such a way that it is possible to fully define the attitude of the system, including rotation about all three body axes. Euler angles do not need a reference vector; instead, they need a predefined reference axes with respect to which all rotations occur.

In the SPHERES implementation of Euler angles, we always do the rotation in the following order:

- |     |       |  |
|-----|-------|--|
| 1 - | Roll  | Rotation about the X <b>Reference</b> Axis |
| 2 - | Pitch | Rotation about the Y <b>Reference</b> Axis |
| 3 - | Yaw   | Rotation about the Z <b>Reference</b> Axis |

*Note: Euler angles do **not** rotate about the body axes of the vehicle.*

The following figures illustrate the examples above, but with Euler angles defining the pointing of the SPHERES satellite.

*NOTE: the following examples show progressive “roll - pitch - yaw” steps of pointing in a specific direction. However, it is not necessary to command in those steps. Once the correct full Euler angles vector of [roll, pitch, yaw] that ends in the desired attitude is obtained, it is possible to directly command to go to that attitude.*

### ***Euler angle: [0,0,0]***

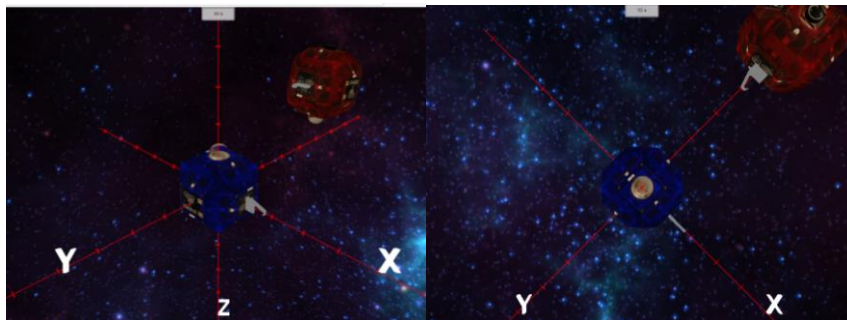
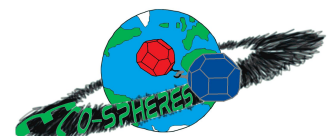


Figure 7.5 - Euler angles “origin”: the vehicle is rotated 0 radians about all the axes, so its own body axes are fully aligned with the reference frame.

### ***Euler angle: [0,0,pi/4]***





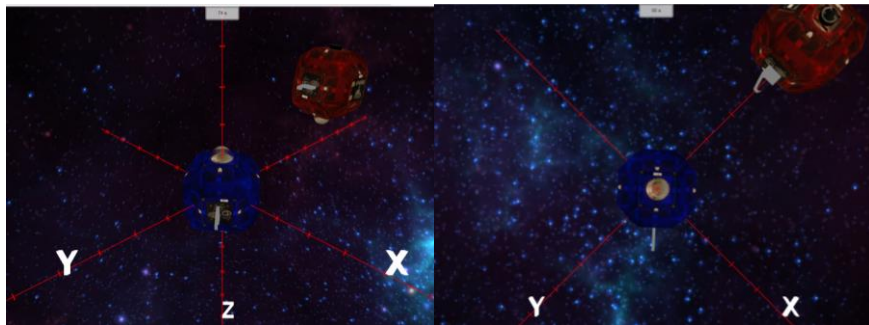


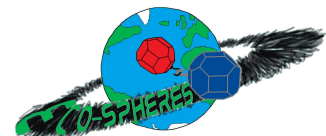
Figure 7.6 - Euler angles that results in the basic rotation to pointing vector  $[0.7071, 0.7071, 0]$ , but with all the axis being controlled. In this case Euler angles command: 0 roll (no rotation about the X reference axis), 0 pitch (no rotation about the Y reference axis, and  $\pi/4$  yaw (rotation about the Z reference axis).

*Note:  $\pi/4 = 45$  degrees*

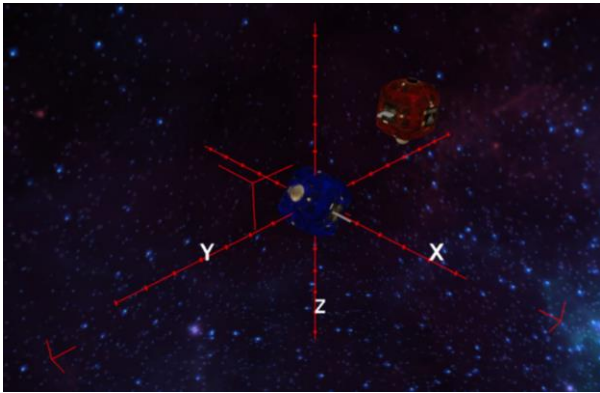
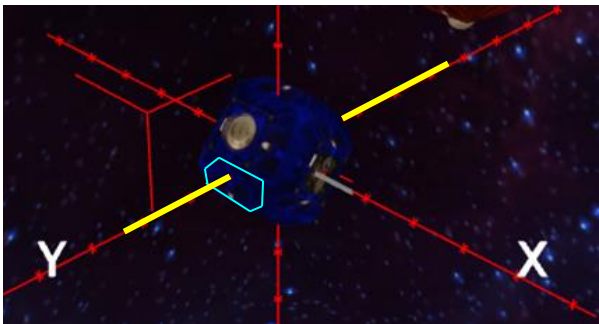
Recall that Euler angles **are applied in order about the reference frame**, therefore it is *not* possible to think about the current pointing of the satellite and think about the *body* rotation you would do in order to reach a new pointing target. The Euler angle rotation is referenced always from the origin to the final pointing.

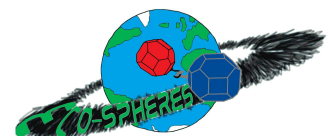
The next example shows how when followed in order, it is possible to visualize the rotation of the vehicle with respect to the **reference** axis.

<p>[0,0,0]</p>	<p>Origin - the hook is pointing on the +X reference axis direction, the tank to the -Z axis, and the control panel to the -Y axis. This are the SPHERES body axes aligned to the reference frame.</p> <p>The next step is to do a <b>roll</b> about the <b>reference</b> body axis...</p>
<p><math>[\pi/4, 0, 0]</math></p>	<p>This rotation should be simple to visualize, the hook rotated <math>\pi/4</math> about the X axis.</p>

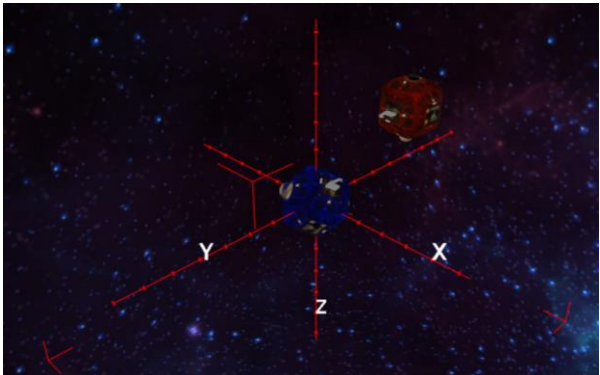
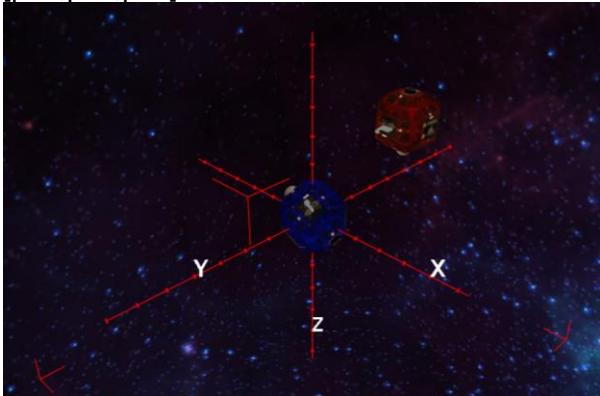




	<p>An Euler roll rotation does not mean that the X axis moves, it means the satellite rotates about the original reference X axis.</p> <p>Now a combined roll &amp; pitch attitude...</p>
	<p>To understand this rotation, one must think about the <b>reference</b> axis for pitch - in this case the Y axis in the figure.</p> <p>From the pointing it had in its original roll, the vehicle rotates about the Y <b>reference</b> axis. Note that the Y reference axis is no longer the same as the body Y axis, because of the first rotation.</p> <p>The Euler pitch rotation happens about the Y <b>reference</b> axis, so in this example imagine putting a stick between the two SPHERES flat panels that are intersecting the Y reference axis, and rotating about it:</p> 
<p>[pi/4,pi/4,0]</p>	<p>Resulting Euler angles rotation of</p> <ul style="list-style-type: none"> <li>- Roll: pi/4</li> <li>- then Pitch: pi/4</li> </ul>

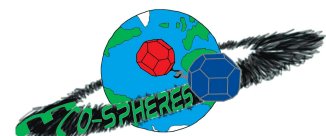




	
	<p>The next rotation will be about the Z reference axis.</p> <p>Therefore, the next rotation does not mean that the satellite will turn on its tank axis (the <i>body</i> Z axis), but that it will start with its rotation of <math>[\pi/4, \pi/4, 0]</math> and then rotate the body along the Z <b>reference</b> axis, which crosses a new set of panels on the SPHERES satellite.</p>
<p><math>[\pi/4, \pi/4, \pi/4]</math></p> 	<p>Final pointing with Euler angles of:</p> <ul style="list-style-type: none"> <li>- Roll: <math>\pi/4</math></li> <li>- Pitch: <math>\pi/4</math></li> <li>- Yaw: <math>\pi/4</math></li> </ul>

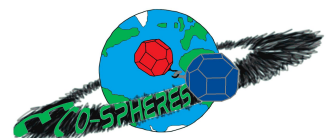
The following example shows how the order of the Euler angles does affect the pointing, and the need to think about Euler angles always from the origin of the reference frame in that order.

<p><math>[0,0,0]</math></p>	<p>Start at the Origin.</p>
-----------------------------	-----------------------------

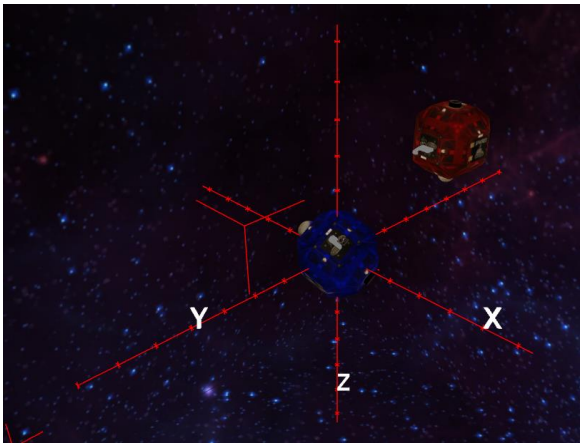
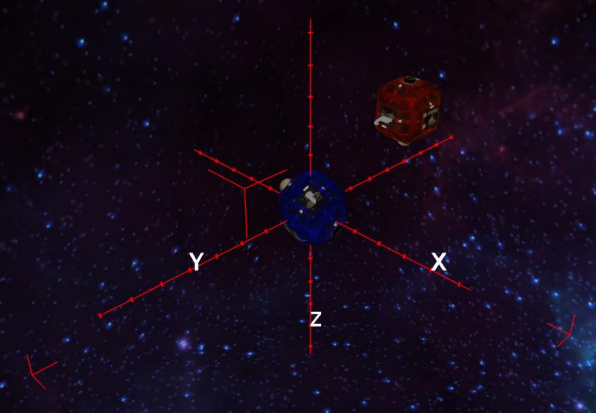




<p>[0,0,pi/4]</p>	<p>Apply in order:</p> <ul style="list-style-type: none"> <li>- Roll 0</li> <li>- Pitch 0</li> <li>- Yaw <math>\pi/4</math></li> </ul> <p>results in an expected rotation about the Z axis of both the reference frame and original body axis. This one should be simple to understand.</p>
<p>[0,pi/4,pi/4]</p>	<p>This new rotation requests:</p> <ul style="list-style-type: none"> <li>- Roll 0</li> <li>- Pitch <math>\pi/4</math></li> <li>- Yaw <math>\pi/4</math></li> </ul> <p>While it is only “adding” the Y axis pitch, it is <b>not</b> rotating the previous attitude of [0,0,pi/4] along the Y reference axis.</p> <p>Instead, it is starting from the origin and applying roll, pitch, yaw in that order.</p>
	<p><i>Note: if the order was applied as Yaw first, then Pitch, this would be the resulting attitude:</i></p>

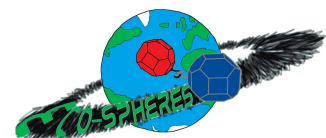




	
<p>[<math>\pi/4, \pi/4, \pi/4</math>]</p> 	<p>The final rotation is the same as in the previous example. Note that it does <b>not</b> show the <math>[0, \pi/4, \pi/4]</math> vehicle rotated about the reference X axis, because that would be the wrong order. To get here the attitude is calculated in order: roll, pitch, yaw.</p>

## Potential issues with Euler Angles (aka, why you may want to use Quaternions)

Euler angles have been used heavily in the field, including many control systems for airplanes and other vehicles. However, Euler angles have a problem with full 3D motion (not common in airplanes): when the angle points “straight up/down” (the pitch is  $\pm\pi/2$ ) there is an issue called “gimbal lock”. The simplified explanation is: when pointing straight up/down either the yaw and the roll or the yaw and the pitch cannot be distinguished from each other and control systems easily go unstable. Satellites and other vehicles in space do point straight up (or in any direction); airplanes and cars usually do not.

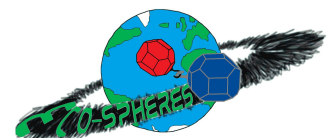






Fortunately for EcoSPHERES this is not a serious problem, because the pointing requirements will be mostly in the XY plane, with only small offsets towards the Z axis direction. Therefore Euler angles should not be a problem for EcoSPHERES.

It is of course still possible to use quaternions; obtain references for them online. Use the `api.setQuatTarget` function.





## 8 Revision History

Revision	Date	Changes Made	By
Italian 1.0	Feb 1 <sup>st</sup> , 2019	Adapted to Italian Championship	L. Reyneri
1.1	Feb 13 <sup>th</sup> , 2019	Removed section Scoring at pag. 21	L. Reyneri

