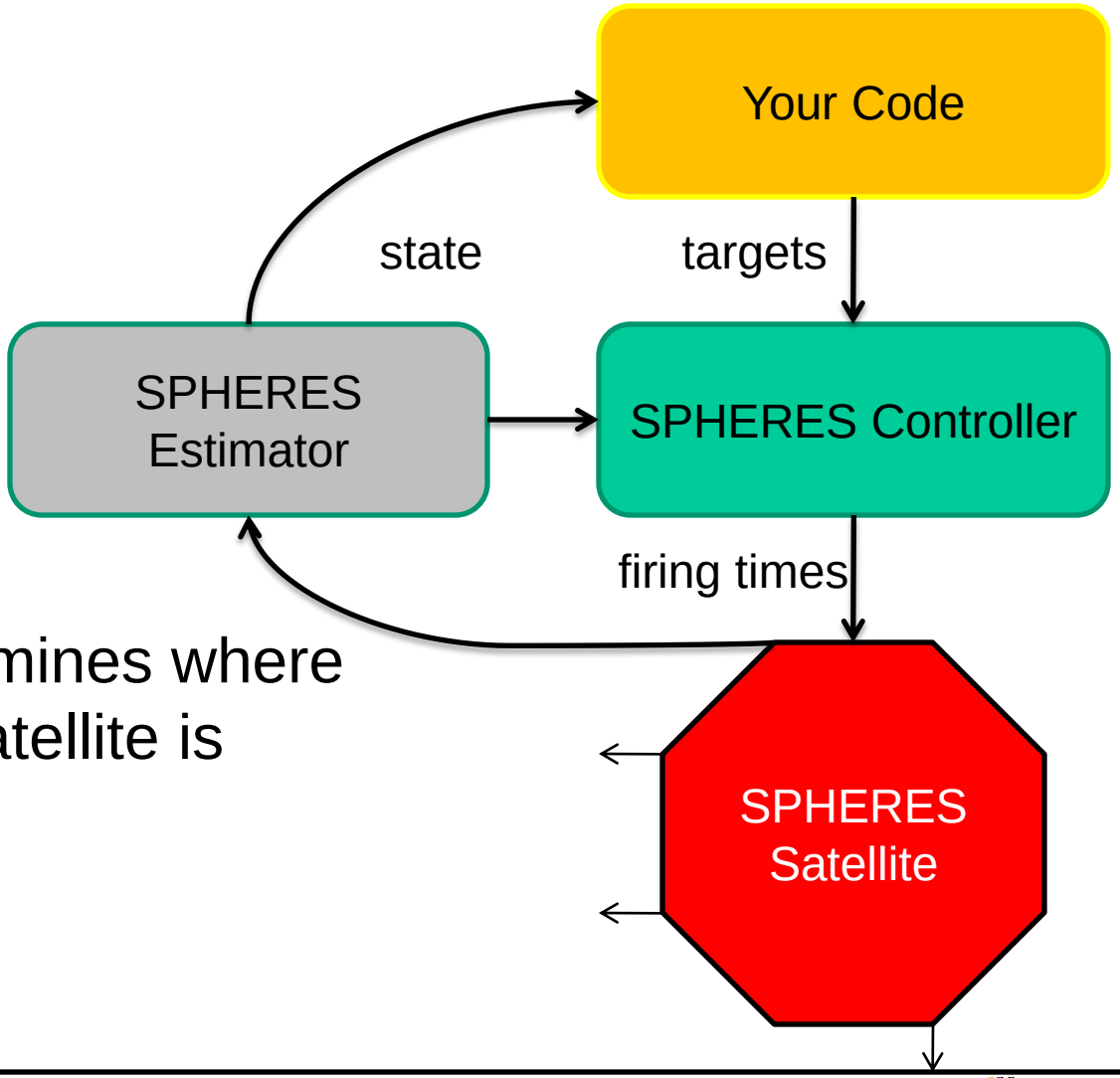

# ZR API Overview

# API Functions


This presentation will explain the basic movement functions of the Zero Robotics Application Programming Interface. The following is an abbreviated list of functions that are automatically included by the simulation, and can be called by any user function.  A full list of functions is available in the ZR API documentation.

- init – one-time initialization function
- loop – the main user code loop
- api.setForces – apply an impulse to the satellite
- api.setPositionTarget – set a target position for the satellite to move to
- api.setVelocityTarget – set target velocity for the satellite to move at
- api.setAttitudeTarget – set the unit vector for the satellite to point along
- api.setTorques – apply a body-frame torque impulse to the satellite
- DEBUG – print a message to the console with information for debugging

# Zero Robotics Control System




tells the satellite where to go

uses your targets to calculate thruster commands

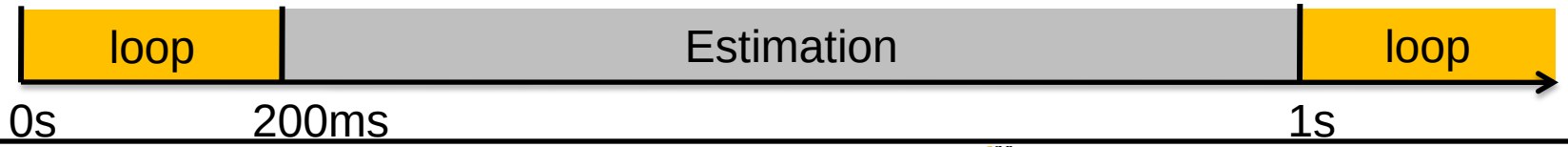
determines where the satellite is

fires thrusters at specific times

# void loop()

```
void loop();
```

- This is the starting point for user code. From loop() calls can be made to other API functions or to other user-defined procedures.
- It is called **once per second** by the SPHERES control system in a repeating loop. **You should not include any infinite loops in your loop() code.**
- After loop() runs, the SPHERES control system converts the targets specified with the api.set*() functions and converts them into thruster impulses. The thruster impulses last for 200ms, after which the SPHERES estimator starts estimating the position and orientation of the satellite.



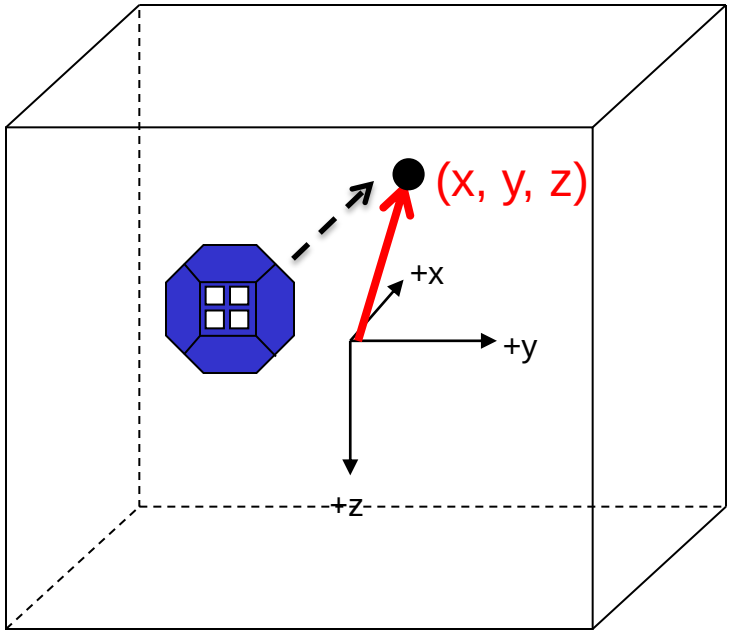
# api.get*State()


- The combination of position, velocity, orientation, and angular velocity is called the satellite's **state**.
- In ZR there are two types of state arrays defined in the API that differ in the way they represent attitudes:
  - `typedef ZRState float[12] | void api.getMyZRState(ZRState state)`
    - Position (x, y, z)
    - Velocity ($v_X$, $v_Y$, $v_Z$)
    - Attitude Vector ($n_X$, $n_Y$, $n_Z$)
    - Attitude rate ($\omega_X$, $\omega_Y$, $\omega_Z$)
  - (Advanced) `typedef state_vector float[13] | void api.getMySphState(state_vector state)`
    - Position (x, y, z)
    - Velocity ($v_X$, $v_Y$, $v_Z$)
    - Quaternion Attitude ($q_1$ ,$q_2$ ,$q_3$ ,$q_4$ )
    - Attitude rate ($\omega_X$, $\omega_Y$, $\omega_Z$)
- The state of the other satellite can be retrieved with `api.getOther*State`
- A SPHERES state can be converted to a ZR state with `api.spheresToZR`

# api.setPositionTarget

```
void api.setPositionTarget(float posTarget[3]);
```

- Input:
  - **posTarget[3]** {x, y, z} Allows you to set the x, y, and z position targets for the satellite control ssytem
  - Units are in meters
- Commanding a position target activates the SPHERES control system and will result in the satellite firing thrusters.

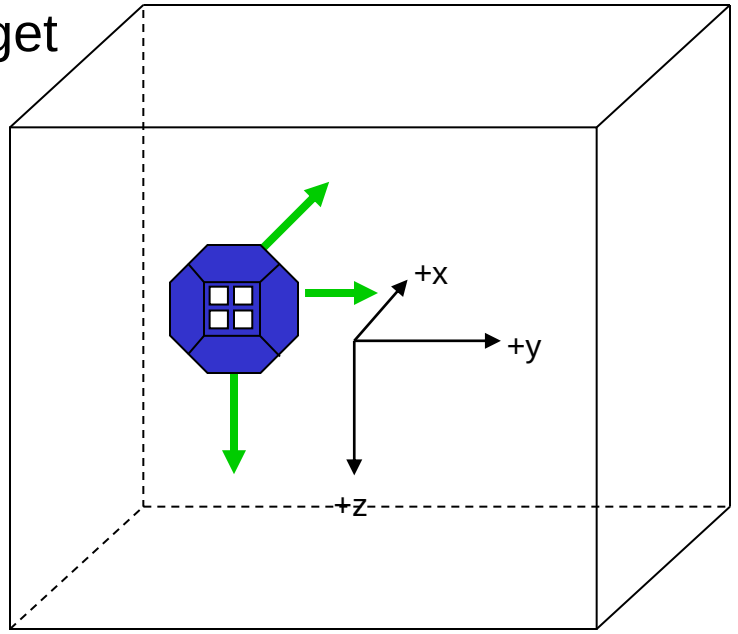


# api.setVelocityTarget

```
void api.setVelocityTarget(float velTarget[3]);
```

- Input:
  - **velTarget[3]** $\{v_X, v_Y, v_Z\}$ sets the x, y, and z target velocities
  - Units are in meters per second
- Can be combined with api.setPositionTarget to follow a trajectory specified in terms of position and velocity targets
- Commanding a velocity target activates the SPHERES control system and will result in the satellite firing thrusters.

Target velocities in $v_X$, $v_Y$, and $v_Z$



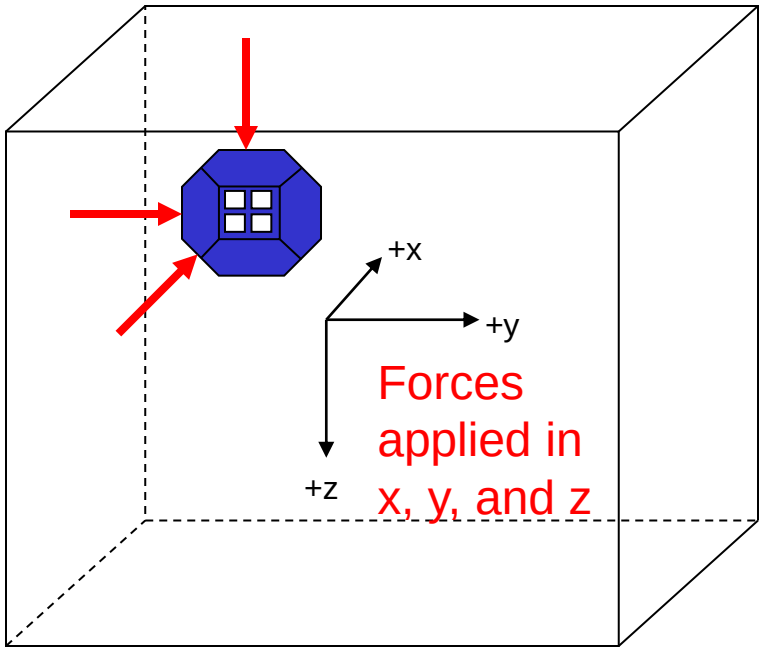
# api.setForces

```
void api.setForces(float forces[3])
```

- Input:
  - **forces[3]** $\{f_x, f_y, f_z\}$ Applies global frame x, y, and z impulses to the satellite
  - Units are in Newton-seconds
- Since the thrusters only fire for at most 200ms at a constant thrust, the "forces" commanded by this function are delivered as equivalent **impulses**. The force commanded times one csecond matches the thruster force times the thruster on time.

$$F_{command} * 1s = F_{thruster} * t_{on}$$

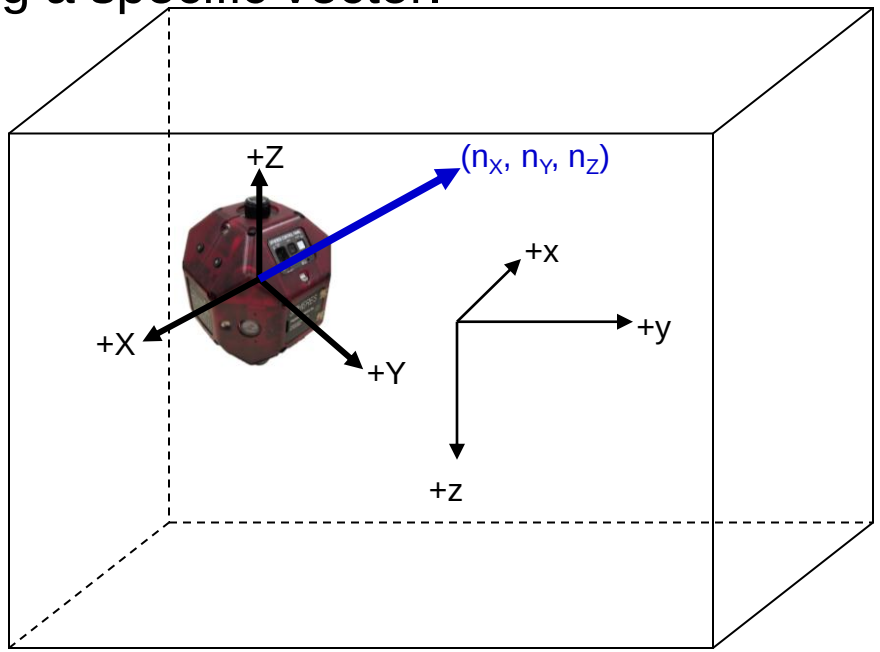
- May be combined with api.setPositionTarget and api.setVelocityTarget



# api.setAttitiudeTarget

```
void api.setAttitudeTarget(float attTarget[3]);
```

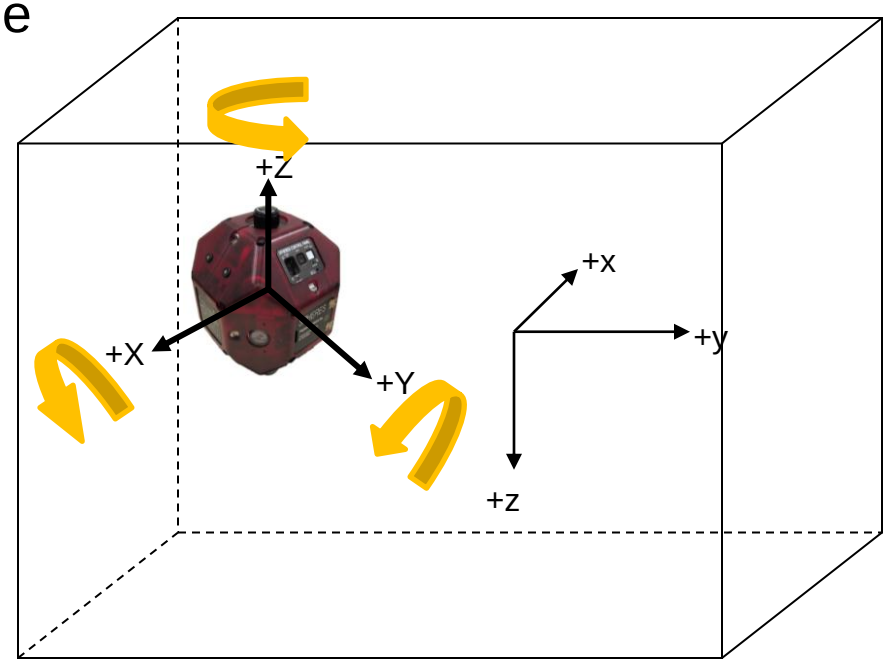
- 1 Input
  - **attTarget[3]** {$n_X$, $n_Y$, $n_Z$} sets the unit vector for the satellite to point its Velcro (-X) along
  - Units are dimensionless
- Allows you to point the satellite along a specific vector.
- This vector specifies a pointing **direction**, not a pointing **location.**
- For more information on setting Attitude, see the Rotate Tutorial.



# api.setAttRateTarget

```
void api.setAttRateTarget(float rates[3])
```

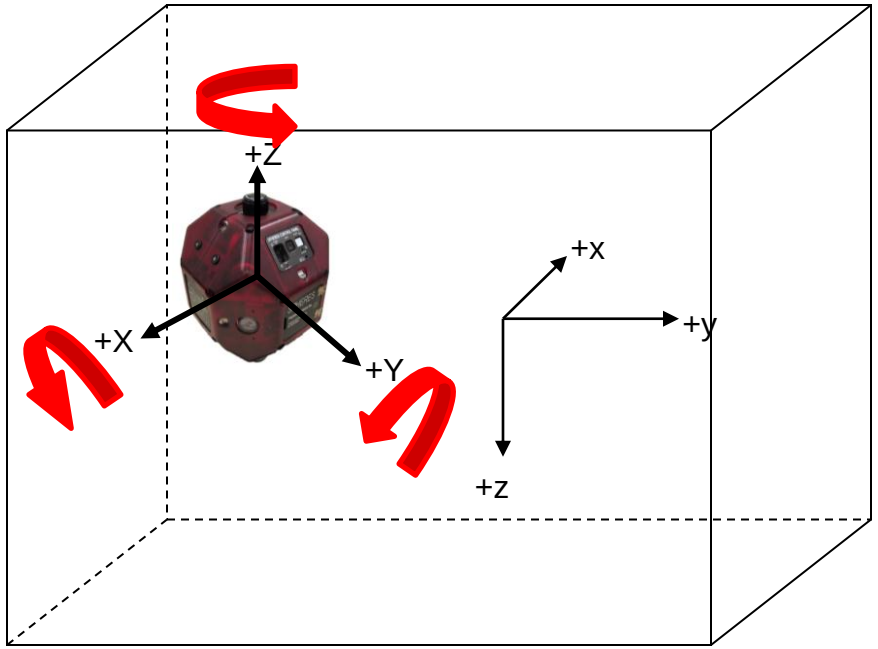
- 1 input
  - **rates[3]** $\{\omega_x, \omega_y, \omega_z\}$ body-frame x, y, and z rotational rate targets
  - Units are in rad/s
- This functions serves as the attitude equivalent of api.setVelocityTarget
- Note that the rotation rates are in the body frame, not in the global frame.
- Can be combined with api.setAttitudeTarget()



# api.setTorques

```
void api.setTorques(float torques[3])
```

- 1 input
  - **torques[3]** $\{\tau_x, \tau_y, \tau_z\}$ body-frame x, y, and z rotational impulses to be applied to the satellites
  - Units are Newton-meter-seconds
- Like the api.setForces command, this command applies equivalent rotational impulses to the supplied torque commands.



# DEBUG


```
void DEBUG((const char *format, ...))
```

- Variable number of inputs
  - **format** “Text %d %f \n” is a string containing text and special format characters to display variables. This string will be printed to the console window, with format characters replaced by the current values of corresponding variables.
  - Subsequent parameters are the variables to be printed to the console, in the order in which the format characters are displayed in **format**. These can be ints (%d), floats (%f), doubles (%lf), strings (%s), etc.

- DEBUG statements are NOT counted toward total code size.
- The format string can be formatted using the same specifiers as those in the C++ command printf().