ZERO ROBOTICS

SPHERES ISS CHALLENGE

setAttitudeTarget, Revisited

Goals

- In this tutorial you will:
  - Identify the vector between your SPHERE and a point.
  - Normalize the vector to a unit vector so it specifies only direction.
  - Use the normal vector with setAttitudeTarget to rotate to face the point.
- Now that we've covered the basics and how to use vectors to control position, we'll go over how to use vectors to control attitude.
- Rotation is no less important than translation, and learning to rotate will help you write better code.
- As before, we'll go over the fundamentals first, then the code.

Direction

- In two dimensions, direction can be measured by the angle (θ) between the vector and the x-axis.
- However, the direction of a vector is more useful in the form of a **unit vector**, a vector with a magnitude of 1.
- A unit vector has the same direction as the original vector.
- There is exactly one unit vector for every direction, so finding the unit vector is equivalent to finding the direction.

Finding the Unit Vector

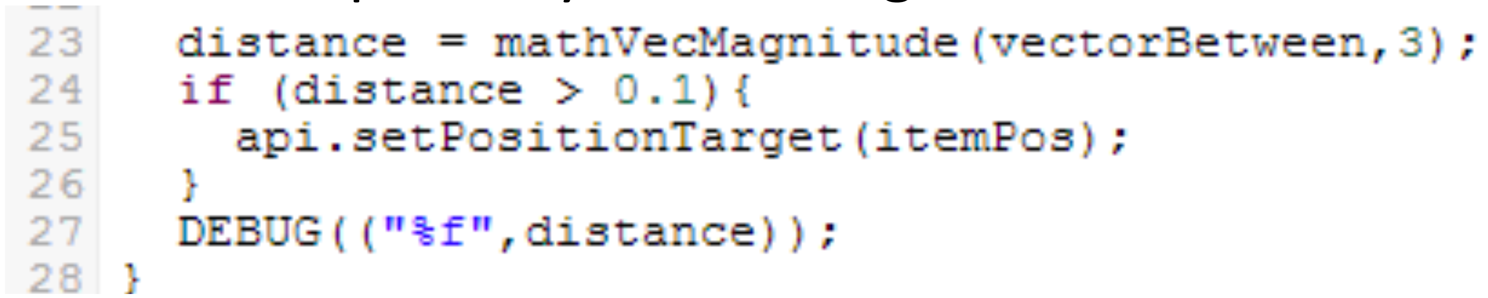
- To find the unit vector **â** by hand, first find the magnitude of the original vector **a**.
- Then, divide each component of the original vector by the magnitude. This is called **normalizing** the vector.
- ex: **a** = [2, 2, 1]

$$|\mathbf{a}| = \sqrt{2^2 + 2^2 + 1^2} = \sqrt{9} = 3$$

$$\hat{a} = [\frac{2}{3}, \frac{2}{3}, \frac{1}{3}]$$

setAttitudeTarget

- If you haven't already, go through the tutorial on **setAttitudeTarget** before you continue.
- The **setAttitudeTarget** function requires a unit vector as the argument.
- In this example, our goal is to have the satellite face the origin at all times as the satellite moves around.
- Open Project12 from the last tutorial and save it as Project13.
- Delete this last part of your existing code:



```
    distance = mathVecMagnitude(vectorBetween,3);
    if (distance > 0.1){
      api.setPositionTarget(itemPos);
    }
    DEBUG(("%f",distance));
}
```

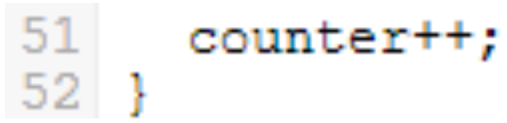
Create Imaginary Items

- We want to create several imaginary items all over the playing space. We also want an array for the origin.
- Create a new array for each item and initialize them in init() with the coordinates below.

Name	Position
item1	(0.4, 0.4, 0.6)
item2	(-0.4, -0.4, -0.6)
item3	(0.1, -0.2, 0.3)
item4	(0.5, 0.2, -0.5)
item5	(-0.2, 0.4, 0.1)
origin	(0.0, 0.0, 0.0)

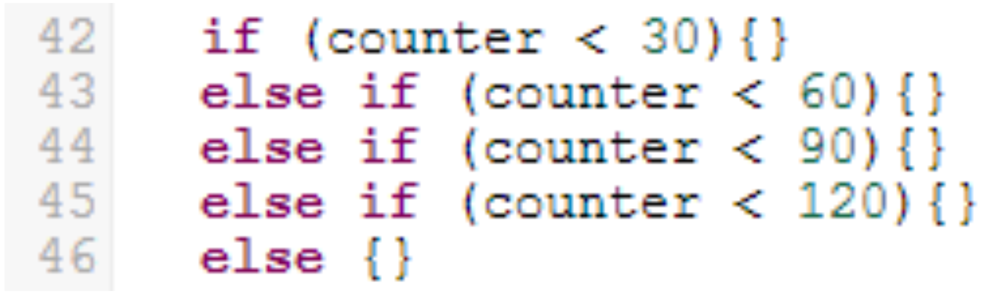
Create a Step Counter

- Declare **int counter** and initialize it to 0.
- Before the closing bracket of loop(), increment counter.



```
    counter++;
}
```

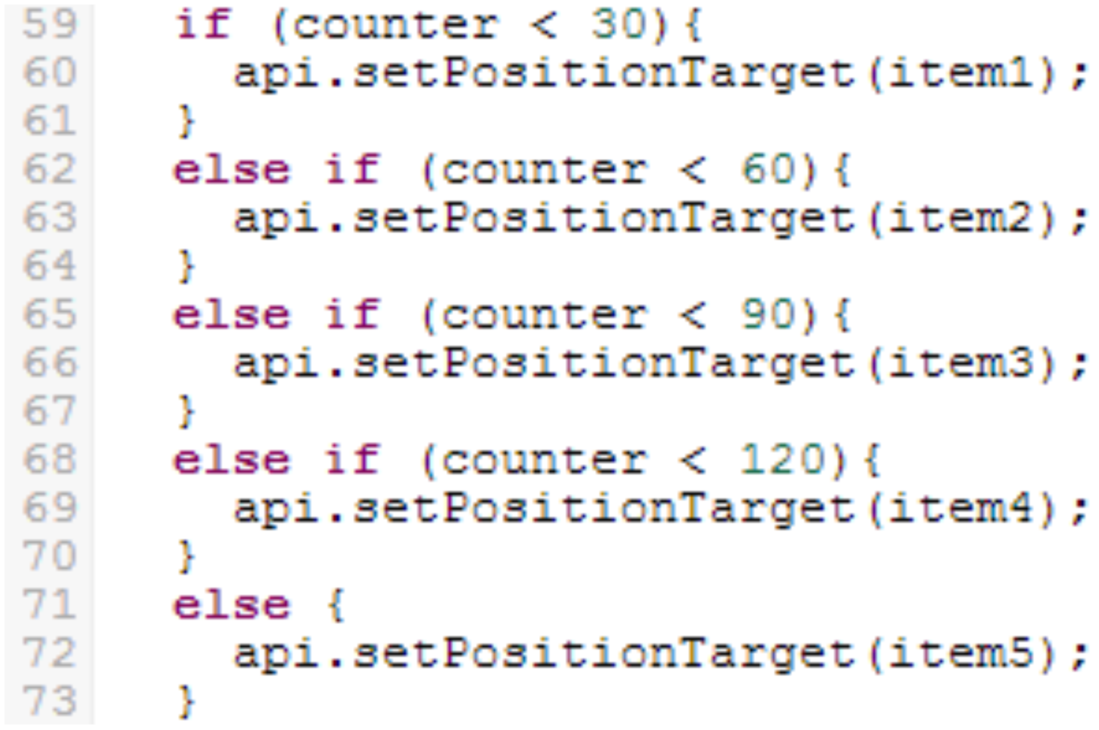
- Since loop() is called once per second, counter keeps time. For the sake of simplicity, we will give the satellite 30 seconds to navigate to each item.
- Create an “if, else if, else” framework dependent on time.



```
if (counter < 30){}
else if (counter < 60){}
else if (counter < 90){}
else if (counter < 120){}
else {}
```

Set the Position Target

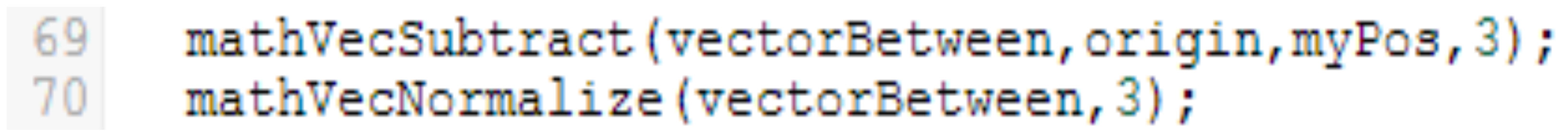
- For each conditional statement, set the position target to the corresponding item.



```
if (counter < 30){
  api.setPositionTarget(item1);
}
else if (counter < 60){
  api.setPositionTarget(item2);
}
else if (counter < 90){
  api.setPositionTarget(item3);
}
else if (counter < 120){
  api.setPositionTarget(item4);
}
else {
  api.setPositionTarget(item5);
}
```

Find the Unit Vector

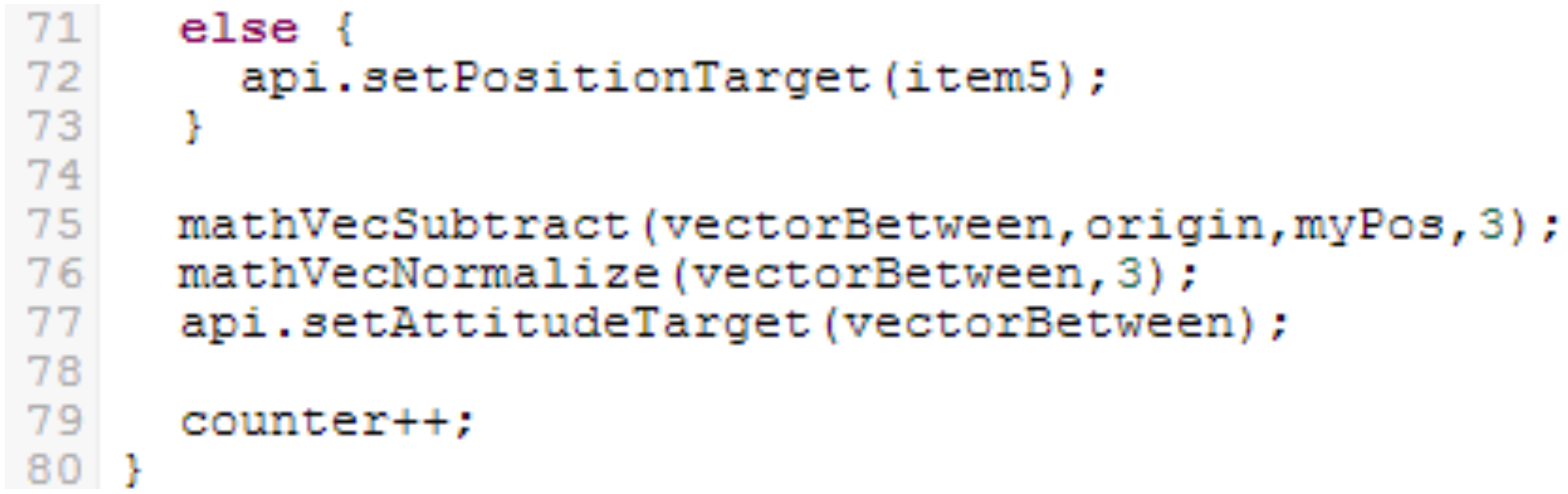
- You should still have the mathVecSubtract statement from the last tutorial in loop().
  - Copy and paste it below the **else** statement.
  - Change **itemPos** to **origin**.
- vectorBetween now references the vector between your satellite and the origin. To make it a unit vector, we have to normalize it.
- Use the function **mathVecNormalize**. Give the vector and the length as arguments.



```
mathVecSubtract(vectorBetween,origin,myPos,3);
mathVecNormalize(vectorBetween,3);
```

Rotate to Face

- All that's left is to set the attitude target to the unit vector vectorBetween. Use the function setAttitudeTarget.
- The last few lines of your code should look like this:



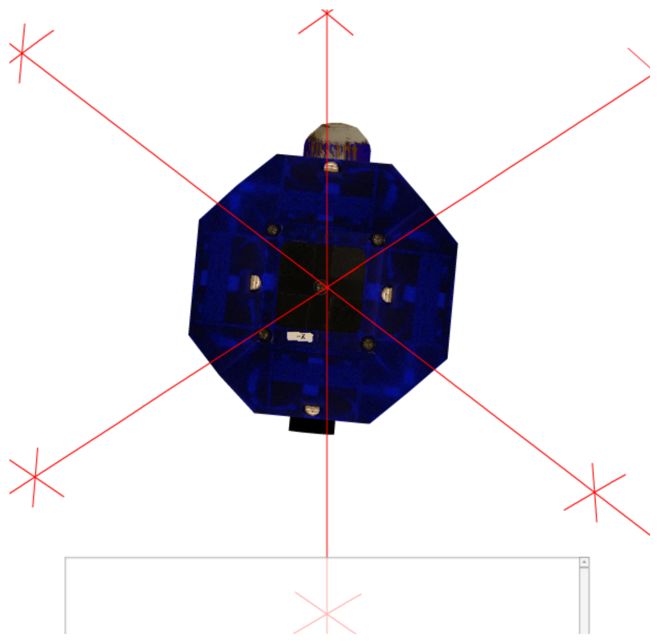
```
  else {
    api.setPositionTarget(item5);
  }

  mathVecSubtract(vectorBetween,origin,myPos,3);
  mathVecNormalize(vectorBetween,3);
  api.setAttitudeTarget(vectorBetween);

  counter++;
}
```

Testing

- Compile and run the simulation. Make sure the maximum simulation time is at least 180 seconds.
- Zoom in and rotate the coordinate plane. Can you see the Velcro face of the satellite facing the origin?



What’s Next?

- Now you know how to use vectors to move around the coordinate plane and rotate to face any direction. This will come in handy when you program your satellite to spin and revolve, as covered in later tutorials.
- So, you know the basics. But what about efficiency? Go on to the next tutorials to learn how to make your satellite fast and agile. Good luck!