ZERO ROBOTICS

SPHERES ISS CHALLENGE

Velocity

DARPA NASA CASIS [TOPCODER] MAP Aurora FLIGHT SCIENCES MIT SSL MIT

setVelocityTarget

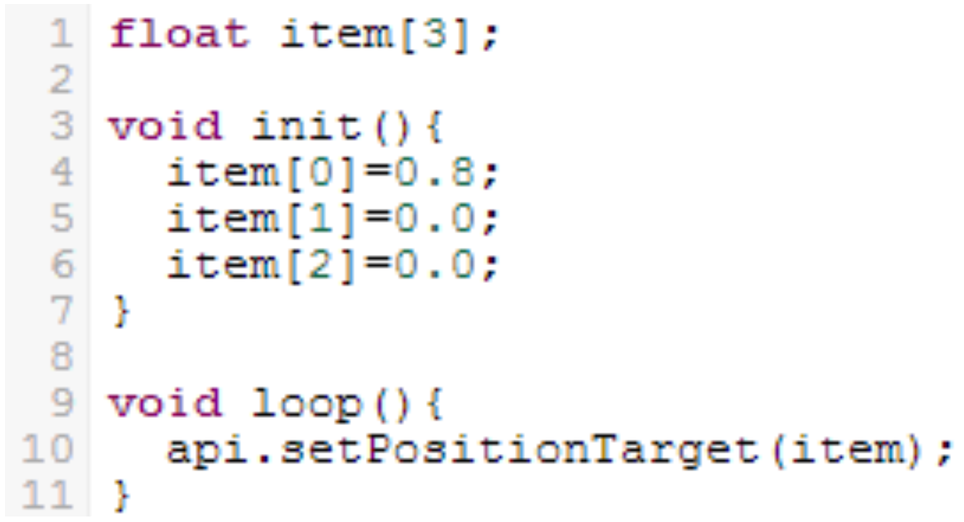
- This tutorial will teach you how to control the velocity of your SPHERES by using the function **setVelocityTarget**.
- Using setVelocityTarget is not easy. If you feel confused during this tutorial, review previous tutorials, experiment in the IDE, or ask your team members for help. Also, never hesitate to email the ZR team.

The Scenario

- In this tutorial, our goal is to move to an imaginary item at the point (0.8, 0.0, 0.0) as quickly as possible.
- We will walk you through three techniques for reaching the item. The third will be the most efficient, but you will have to go through the first two in order to understand it.
- Create a new project called Project14a. We want to start from scratch for this example.

Technique 1

- You can think of our first technique as the control group.
- Simply create a new array of floats called **item** and initialize it with the coordinates on the previous slide.
- In loop(), set the position target to item. Compile and run.



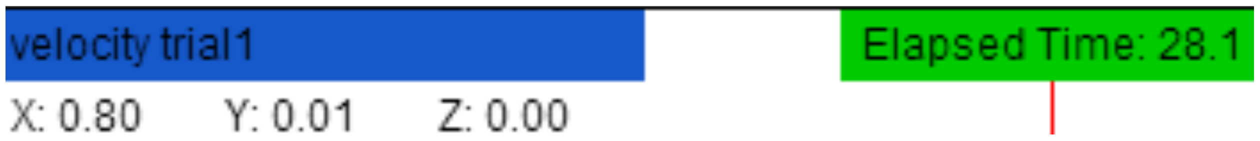
```
float item[3];

void init(){
  item[0]=0.8;
  item[1]=0.0;
  item[2]=0.0;
}

void loop(){
  api.setPositionTarget(item);
}
```

Results

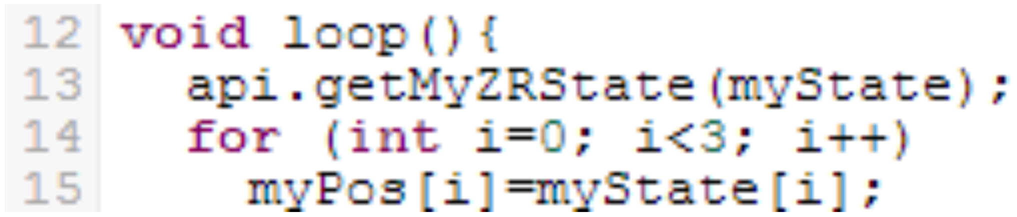
- It takes the satellite about 28 seconds to reach the item. As you already know, time is extremely valuable in Zero Robotics. Every second counts.
- If you can reach the item in less than 28 seconds, you will have an immediate advantage over teams that just use setPositionTarget to move.



```
velocity trial1                          Elapsed Time: 28.1
X: 0.80     Y: 0.01     Z: 0.00
```

Technique 2

- Now, we'll use setVelocityTarget to reach the item. setVelocityTarget is tricky because velocity has magnitude and direction.
- Direction is specified by the components of velocity (x, y, and z). We'll focus on direction first.
- Save your code as Project14b and create three new arrays of floats: **myState**[12], **myPos**[3], and **vectorBetween**[3].
- Retrieve myState and write the first three elements to myPos.

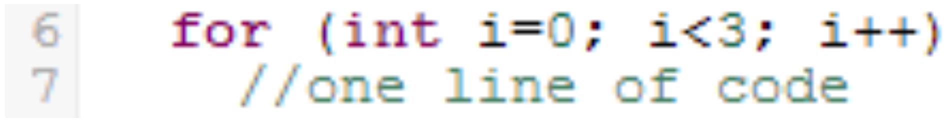


```
void loop(){
  api.getMyZRState(myState);
  for (int i=0; i<3; i++)
    myPos[i]=myState[i];
```

ZERO ROBOTICS SPHERES ISS CHALLENGE

Syntax

- Before we continue, we must address syntax again. It may seem redundant, but it's important to cover syntax so you can avoid compiler errors and code deficiencies.
- You may have noticed that the for loop in the previous slide was written without brackets. Brackets may be omitted in loops and conditionals that contain only one line of code. For example:

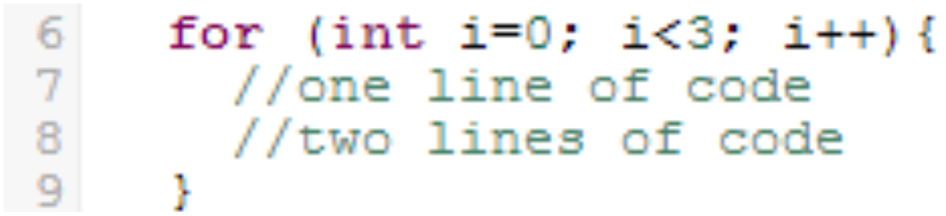


```
for (int i=0; i<3; i++)
  //one line of code
```

- From this point on, we will omit brackets for all conditionals and loops with one line of code.

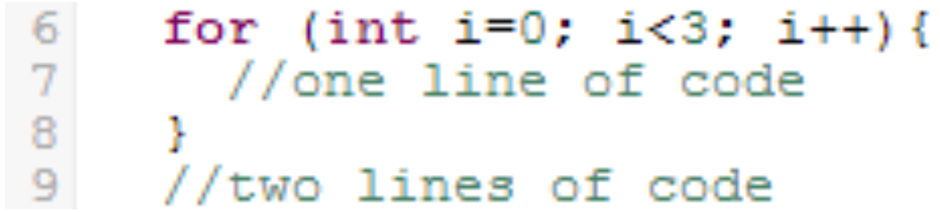
Syntax

- If we were to add another line of code to the for loop, then brackets would be required. For example:



```
for (int i=0; i<3; i++){
  //one line of code
  //two lines of code
}
```

- If you don't use brackets, only the first line will be interpreted as part of the for loop. In other words, the code will be executed like this:

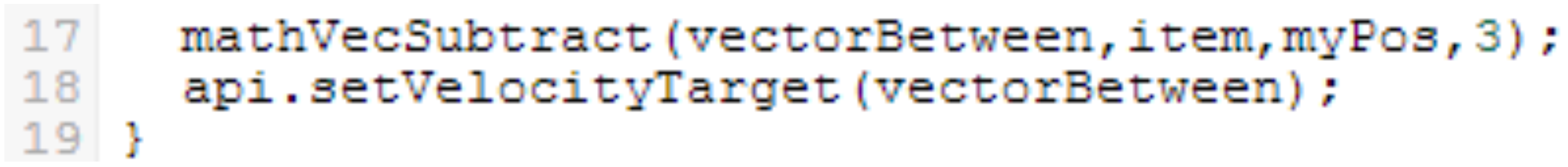


```
for (int i=0; i<3; i++){
  //one line of code
}
//two lines of code
```

- If you want to use brackets in all conditionals and loops out of habit, feel free to do so.

vectorBetween

- Let’s get back to the algorithm. At this point, you’ll want to find the vector pointing from your satellite to the item. Do you remember how to use mathVecSubtract? Store the vector in vectorBetween.
- Since we aren’t setting an attitude target, we don’t have to normalize vectorBetween. Also, we aren’t concerned with magnitude yet, so we can actually use vectorBetween as the target velocity vector. It points in the right direction, and that’s all we need right now.
- Compile and run.



```
    mathVecSubtract(vectorBetween,item,myPos,3);
    api.setVelocityTarget(vectorBetween);
}
```

Results

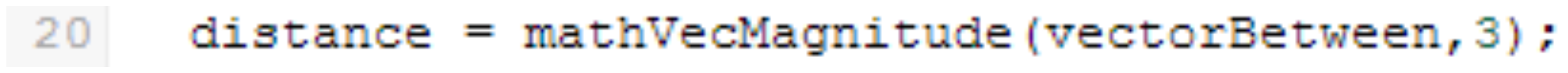
- It takes the satellite about 94 seconds to settle on the target position.
- Is this good? Yes! Granted, 94 seconds is much worse than 28 seconds, but we reached very high speeds in this trial. All we have to do now is find a way to control speed and use it to our advantage.
- The satellite passed the item many times until it finally stopped moving. It did this because of linear momentum.

Linear Momentum

- The magnitude of the target velocity vector (vectorBetween) is equal to the distance between the satellite and the item. So, when the satellite reached the item and the target velocity was 0, why didn't it stop?
- The satellite was still moving quite fast at this point. Even as it passed the item and the thrusters fired in the opposite direction (because the target velocity vector always points toward the item), momentum carried the satellite away from the item.
- With each pass, the satellite reached a lower speed and had less momentum until it finally came to rest on the item.

Technique 3

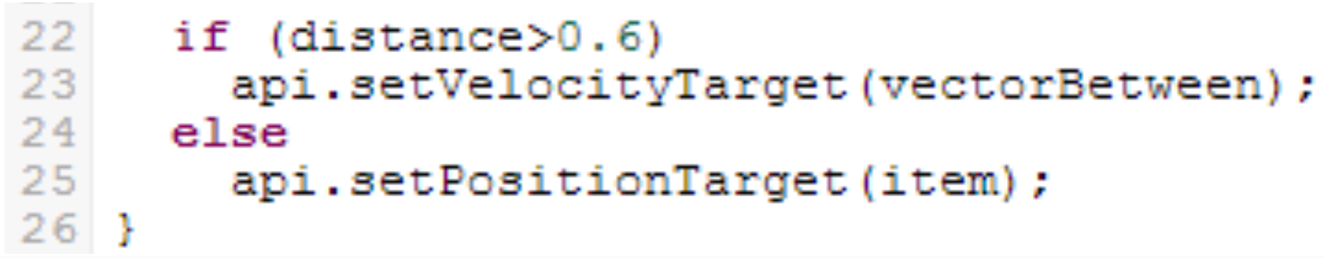
- In order to reach the item without having to pass it several times, we need to account for linear momentum.
- The best way to do this is to use setVelocityTarget in conjunction with setPositionTarget.
- Save your code as Project14c.
- Cut the setVelocityTarget statement from loop(). We'll paste it back in later.
- We know that the magnitude of vectorBetween equals distance. Let's create a float **distance** to store this value.



```
distance = mathVecMagnitude(vectorBetween,3);
```

Distance

- Distance is key. We'll use a set of conditionals dependent on distance to tell the satellite when to aim for a target velocity, and when to just use a target position.
- We know that because of momentum, the satellite needs time (and space) to come to rest.
- For this example, we found by trial and error that 0.6 m is about enough distance for the satellite to slow to a stop.
- Apply the target velocity when distance > 0.6, and just set a target position when the satellite is within 0.6 m of the item.



```
  if (distance>0.6)
    api.setVelocityTarget(vectorBetween);
  else
    api.setPositionTarget(item);
}
```

ZERO ROBOTICS
SPHERES ISS CHALLENGE

Results

- The satellite reaches the item in about 25 seconds. That's better than 28! Like we said, every second counts. Being able to reach a position faster than your opponent might mean the difference between winning and losing that match.
- Can it be done faster? Yes! This code is better than the first two algorithms, but it's still not optimal. The distance cutoff of 0.6 m was an estimation. You may be able to apply setVelocityTarget for longer – it's up to you to do the math.
- After 25 seconds, the satellite continues to adjust its position within a range of about 0.02 m. This is a result of drifting and momentum. Linear momentum is a pain! Learn to handle momentum and you will do well in this competition.

Additional Notes

- vectorBetween is really a position vector; the values are arbitrary when used for velocity. We used it because it pointed in the right direction.
- The magnitude of vectorBetween is the SPHERE's speed. We won't walk you through the process of controlling speed. That's up to you. You have the skill set – now put it to use!
- In the last example, the target velocity changed as the satellite's position changed. You may choose this approach, or you may implement a constant target velocity.
- Keep in mind that SPHERES cannot move infinitely fast. Keep target speeds well under 0.1 m/s.

Unique Code

- At this stage in the tutorial set, you'll find that we won't walk you through everything. We want you to write creative and innovative algorithms that are unique compared to your opponents' algorithms.
- You will also need to write versatile code. In this tutorial, we were able to estimate a distance cutoff of 0.6 m because we knew how far we were from the object. To be able to move everywhere as efficiently as possible, make the distance cutoff a variable dependent on distance, current velocity, and/or other factors.
- Work together as a team to find the most efficient approach and implement it in your code. Good luck!