ZERO ROBOTICS

SPHERES ISS CHALLENGE

Combining Vectors

Goals

In this tutorial, you will:

- Learn the fundamentals of vectors
- Review vector notation, vector addition, vector subtraction, magnitude, and direction
- Practice using ZR vector functions to:
  - define a vector between the satellite and an object
  - monitor the magnitude of the vector as the satellite moves

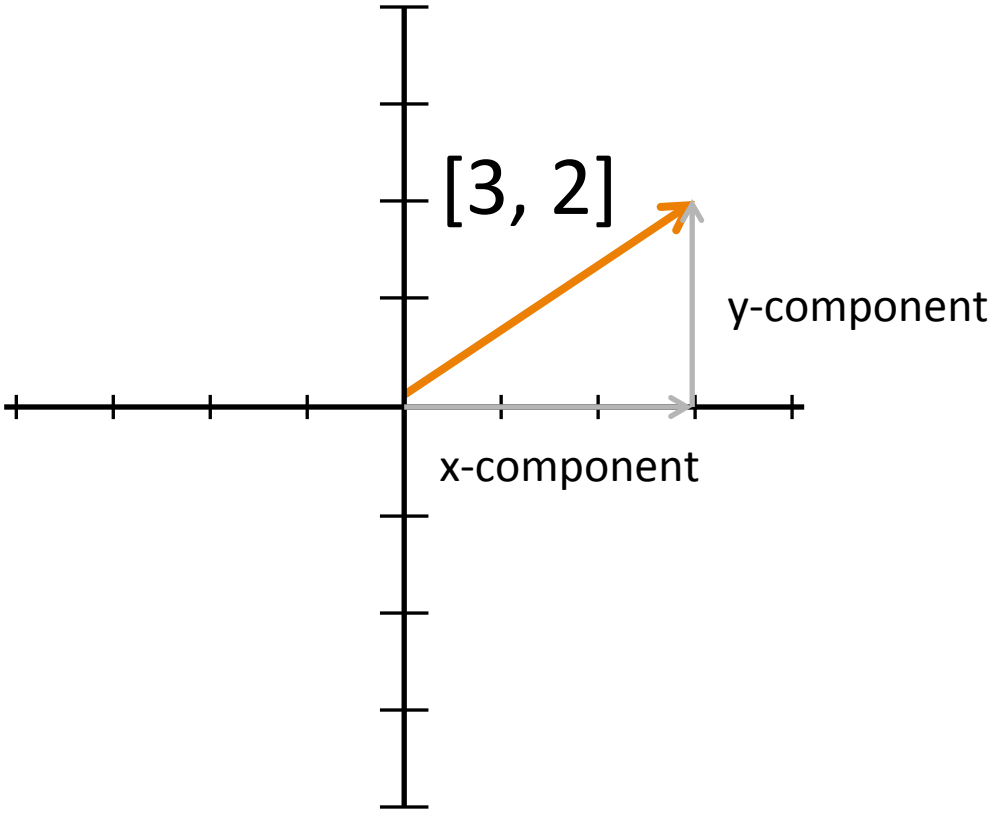
Vector vs. Scalar

- A vector is a measurement with **magnitude** and **direction**.
  - ex: 5 miles to the east
- A scalar has magnitude only.
  - ex: 5 miles
- You will need to use vectors to control the movement of SPHERES, so it is important that you understand the fundamentals. We'll review the basics first and then get into the code.

Vector Notation

- A vector is written in the notation [x, y, z], where x, y, and z are the **components** of the vector.
- The vector [x, y, z] is drawn from the origin to the point (x, y, z).
- For the 2-D phase of the competition, you can think of vectors as [x, y] or [x, y, 0].

Vector Notation



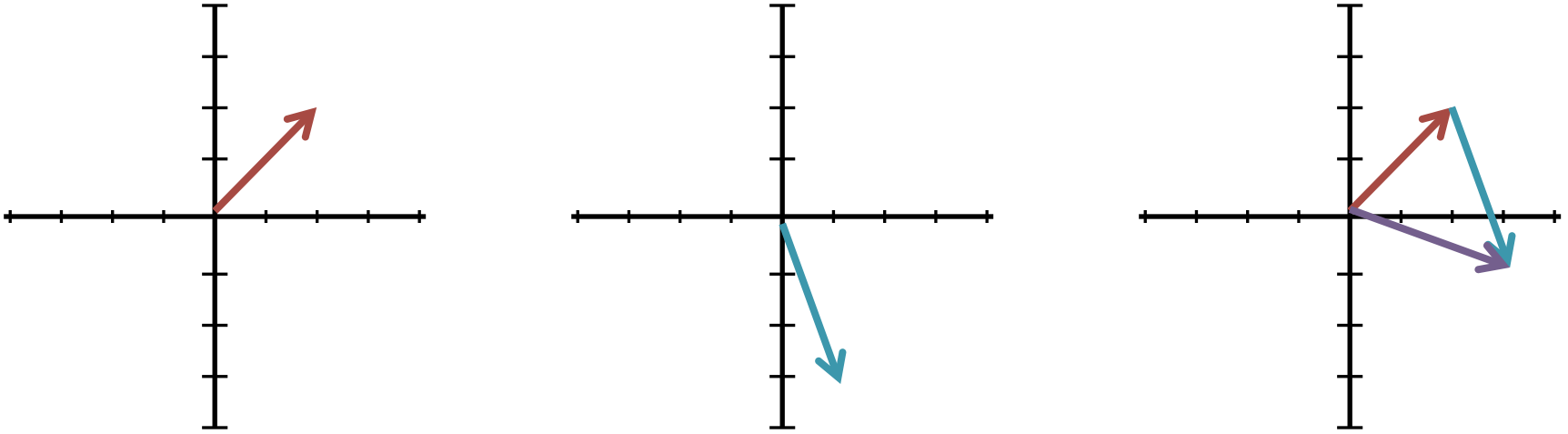
Combining Vectors

- When vectors are added or subtracted, a **resultant vector** is formed that connects the starting point (the origin) to the final destination.
- To add vectors by hand, simply add their components.
  - ex: [4, 2] + [5, 3] = [(4+5), (2+3)] = [9, 5]
- The same rule is used for subtraction.
  - ex: [4, 2] - [5, 3] = [(4-5), (2-3)] = [-1, -1]
- The ZR API has premade functions to combine vectors, but it's important that you know how to work out the calculations by hand before we teach you how to use those functions.

Adding Vectors

- Graphically, vectors are added **tip to tail**. This means the tip (arrow) of the first vector is attached to the tail of the second vector.

[2, 2] + [1, -3] = [3, -1]

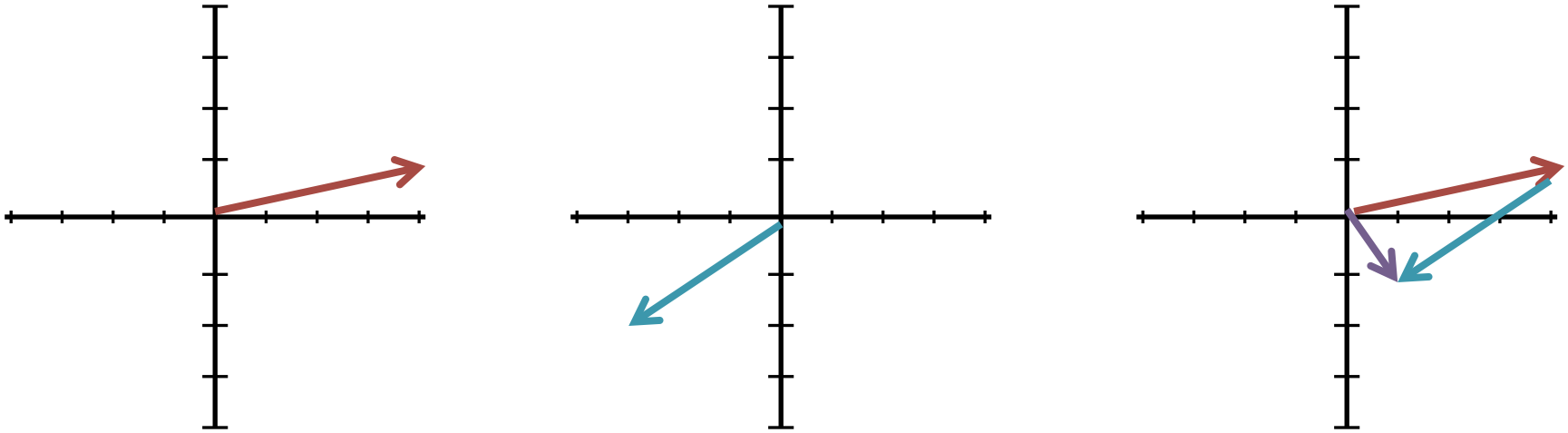


Subtracting Vectors

- Subtracting vectors is equivalent to adding the opposite of the second vector. This allows you to use the tip to tail method.

[4, 1] - [3, 2] = [1, -1]

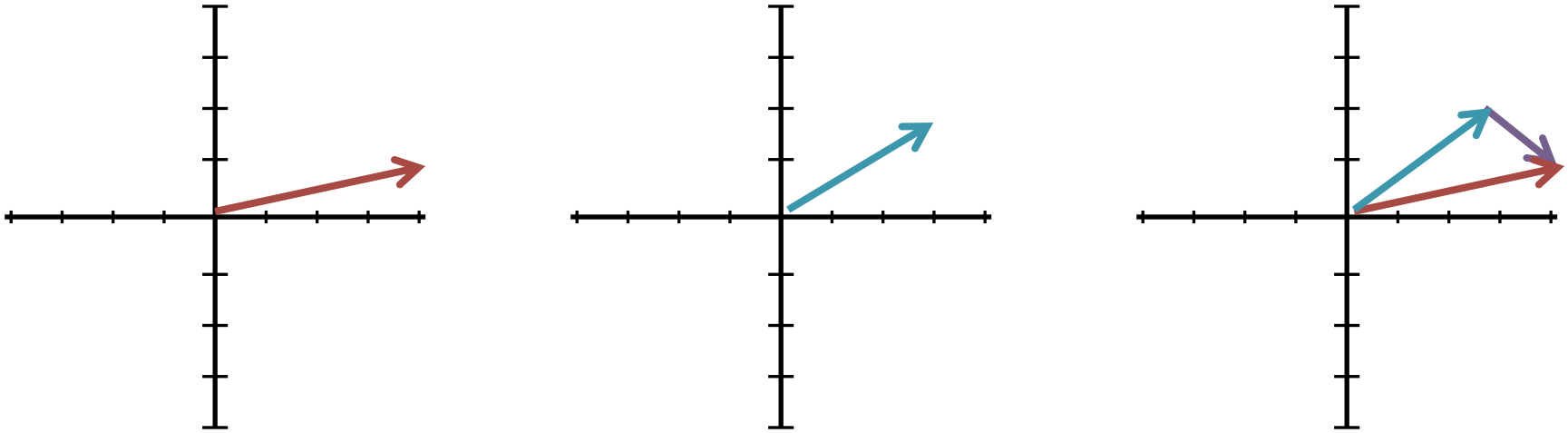
[4, 1] + [-3, -2] = [1, -1]



Subtracting Vectors

- There is an alternative method to subtract vectors that creates a resultant vector which points from the second vector to the first.

[4, 1] - [3, 2] = [1, -1]



Subtracting Vectors

- Although the two methods yield the same results, you may find the second method to be more helpful for Zero Robotics.
- If the first vector represents your satellite and the second vector represents an item you wish to pick up, the second method clearly shows the direction and distance you have to travel.

Magnitude

- Magnitude is the length of a vector. The magnitude of vector **a** is commonly written as $|\mathbf{a}|$.
- The magnitude of a vector can be calculated by hand using a variation of the Pythagorean Theorem:

$$|\mathbf{a}| = \sqrt{x^2 + y^2 + z^2}$$

- Use magnitude for distance. If you have a vector that points from your satellite to an item you want to pick up, the distance you need to travel is the magnitude of that vector.

ZERO ROBOTICS
SPHERES ISS CHALLENGE

Vectors in Code

- The math_matrix.h file in the API will be very useful. Take a moment to familiarize yourself with it. Click here.
- Example: In this scenario, we want to stop 0.1 m away from an imaginary item located at the point (0.8, 0.5, 0.4), regardless of what our starting point is.
- To do this, we will:
  - define the vector between our satellite and the object
  - monitor the magnitude of this vector as our satellite moves toward the object
  - stop moving toward the object when the magnitude of the vector is less than or equal to 0.1.

Create and Initialize Arrays

- Create a new project called Project12 in FreeMode.
- Create an array for your position and an array for the item's position. Call them **myPos** and **itemPos**.
- Also, it's always useful to have a **myState** array.

```
1  float myState[12];
2  float myPos[3];
3  float itemPos[3];
```

- Initialize itemPos with the coordinates on the previous slide.

```
 9  void init(){
10    itemPos[0]=0.8;
11    itemPos[1]=0.5;
12    itemPos[2]=0.4;
13  }
```

Create and Initialize Arrays

- Write the first three elements of myState to myPos, but be sure to do this in loop(), not init(), so that myPos is updated every second with your current position.
- You can do this by writing one line of code for each element, or more efficiently by using a for loop.

```
15 void loop(){
16   api.getMyZRState(myState);
17   for (int i=0; i<3; i++){
18     myPos[i]=myState[i];
19   }
```

Syntax Reminders

- A note on syntax: remember that if a conditional only has one line of code, you don't need curly brackets. Brackets can't hurt, though. We used brackets for the for loop on the last slide even though we didn't need them.
- In the for loop in the last slide, we defined the integer **i** in the declaration of the loop itself. This is legal, and a good coding practice, but has one catch: **int i** has *block scope.*
- Block scope means that the variable only exists inside the loop. If we were to reference **i** outside the brackets, we would get a compiler error.

Create a Vector

- Create a new array called **vectorBetween**. This will be the vector that points from your satellite to the item. Since a vector has 3 components (x, y, and z), make sure vectorBetween has a length of 3.

```
5 float vectorBetween[3];
```

- Although the arrays myPos and itemPos are simply coordinates, they act as vectors with tails at the origin.
- So, we need to subtract myPos from itemPos to get vectorBetween. We will use the built-in function called **mathVecSubtract**.

mathVecSubtract

- mathVecSubtract has four parameters:
  - vector output (vectorBetween)
  - vector input 1 (itemPos)
  - vector input 2 (myPos)
  - length (number of components) of each vector (3)
    - For functions like mathVecSubtract, don't confuse length (size of the arrays) with magnitude. In 3D, length is always 3.

```
mathVecSubtract(vectorBetween,itemPos,myPos,3);
```

- Now, vectorBetween references the vector between your satellite and the item.

Distance

- We want to find the distance between your satellite and the item. This distance is the magnitude of vectorBetween, so we'll use the function **mathVecMagnitude.**
- mathVecMagnitude takes the vector and the size of the vector as parameters, and returns the magnitude as a float. Use this function to find the magnitude of vectorBetween.
- Create a float **distance** to store the magnitude.
  - Note that distance is a single value, not an array.

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

Stop

- All we have to do now is write a conditional to tell the satellite to stop 0.1 m away.
- We'll do this by setting the position target to itemPos while distance > 0.1. We don't need an else statement because the satellite will stop moving if it doesn't have a position target.
- Also, add a debug statement so you can numerically track distance.

```
25    if (distance > 0.1){
26      api.setPositionTarget(itemPos);
27    }
28    DEBUG(("%f",distance));
```

ZERO ROBOTICS
SPHERES ISS CHALLENGE

The Whole Thing

```
1  float myState[12];
2  float myPos[3];
3  float itemPos[3];
4  float vectorBetween[3];
5  float distance;
6
7  void init(){
8    itemPos[0]=0.8;
9    itemPos[1]=0.5;
10   itemPos[2]=0.4;
11 }
12
13 void loop(){
14   api.getMyZRState(myState);
15   for (int i=0; i<3; i++){
16     myPos[i]=myState[i];
17   }
18   mathVecSubtract(vectorBetween,itemPos,myPos,3);
19   distance = mathVecMagnitude(vectorBetween,3);
20   if (distance > 0.1){
21     api.setPositionTarget(itemPos);
22   }
23   DEBUG(("%f",distance));
24 }
```

Testing

- Compile and run your code. Keep an eye on the console as your distance approaches 0.1. It works!
- Change your initial position from the simulation menu and watch your satellite approach the imaginary item from another direction, but still stop 0.1 m away.

More to come…

- Congratulations on making it this far! Vectors are hugely important to Zero Robotics, and knowing how to use them is crucial.
- You now know about magnitude and combining vectors.