ZERO ROBOTICS

SPHERES ISS CHALLENGE

Spinning: Partial Turns

Partial Turns

- Spinning doesn't always require making full revolutions. For instance, some ZR competitions will ask you to rotate 90° to pick up an item.
- In this tutorial, you will:
  - practice using dot product in calculations
  - implement the function mathVecInner
  - use trigonometric functions in code
  - monitor changes in angular displacement

Dot Product

- To measure your satellite's angular displacement (θ), you will need to use the **dot product**. Dot product is a scalar quantity that can be used to find the angle between two vectors.
- To find the dot product, also known as **inner product**, multiply the corresponding components of the two vectors and add the sums.
  - ex: [3, 2, -5] • [1, 4, 6] = (3*1) + (2*4) + (-5*6)

    = 3 + 8 – 30 = -19
- For partial turns, the two vectors will be your initial attitude and current attitude unit vectors.

Dot Product

- As you can see, the dot product is a scalar quantity, not a vector. We can use the dot product to find the angular displacement using this formula:

$$\mathbf{a} \bullet \mathbf{b} = (\cos\theta)(|\mathbf{a}|)(|\mathbf{b}|)$$

- **a** and **b** are vectors. Treat **a** as your initial attitude vector and **b** as your current attitude vector.
- The equation is more useful if we isolate θ.

$$\theta = \cos^{-1}\left(\frac{\mathbf{a} \bullet \mathbf{b}}{(|\mathbf{a}|)(|\mathbf{b}|)}\right)$$

Dot Product

- Let's calculate the angle between the vectors in the previous example.

$$|\mathbf{a}| = \sqrt{9+4+25} = 6.16$$

$$|\mathbf{b}| = \sqrt{1+16+36} = 7.28$$

$$-19 = (\cos\theta)(6.16)(7.28)$$

$$-0.424 = \cos\theta$$

$$\theta = \cos^{-1}(-0.424)$$

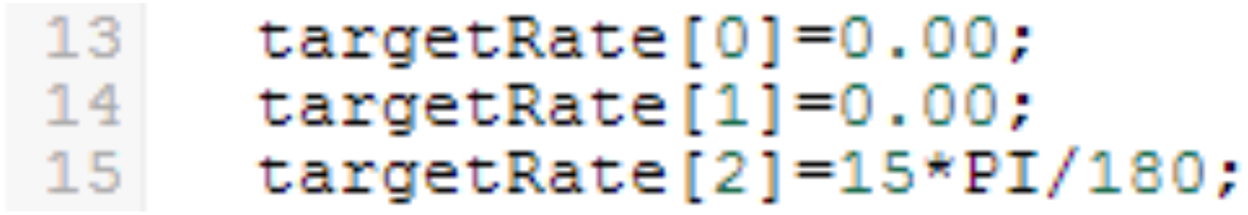
$$\theta = 115.09^{o}$$

ZERO ROBOTICS
SPHERES ISS CHALLENGE

Try It

- Let’s try the scenario in which we have to pick up an item by rotating:
  - 90° clockwise
  - 15 deg/sec on the body z-axis.
- Open Project17 from the Spinning: Angular Velocity tutorial and save it as Project18. Alter the initial values of targetRate to match the above specifications.
- Revisit previous tutorials on spinning if you don’t understand how to determine the values for the targetRate array. Try to figure it out before you see the solution on the next slide.

Solution

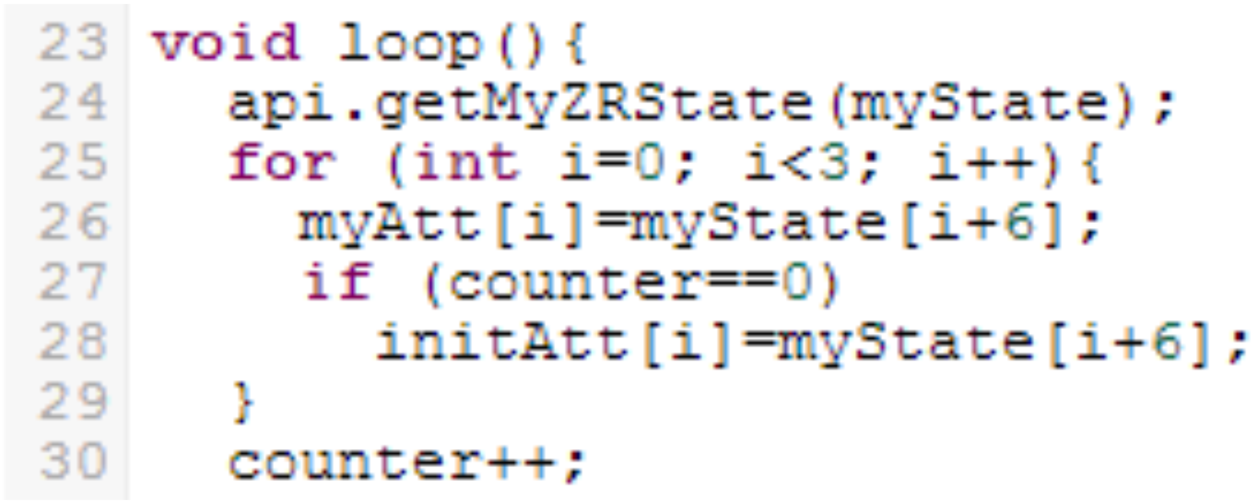


```
targetRate[0]=0.00;
targetRate[1]=0.00;
targetRate[2]=15*PI/180;
```

Prep

- We need to create a few variables. Create the arrays **myState**[12], **initAtt**[3], and **myAtt**[3], and the floats **dot** and **angle**. None of them need to be initialized.
- In loop(), start by retrieving myState and writing your attitude vector to myAtt.
- Also, if it is the first iteration (counter equals 0), store your attitude in initAtt. Remember that attitude is stored in the sixth, seventh, and eighth elements of the state array.
- This for loop contains multiple lines of code. Can you recall the rules of bracket syntax?

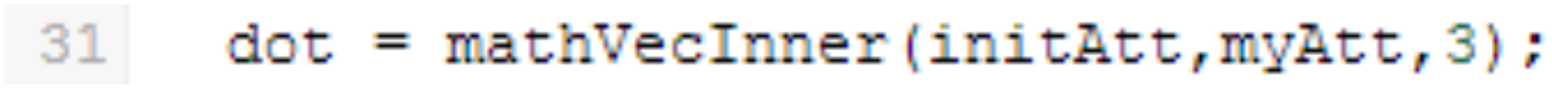
Code



```
void loop(){
  api.getMyZRState(myState);
  for (int i=0; i<3; i++){
    myAtt[i]=myState[i+6];
    if (counter==0)
      initAtt[i]=myState[i+6];
  }
  counter++;
```

mathVecInner

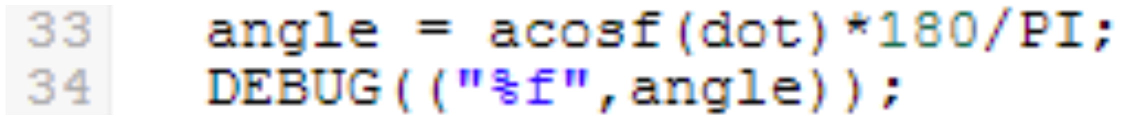
- The first step in finding angular displacement is to calculate the dot product between the initial and current attitude vectors.
- To calculate the dot product, use the function **mathVecInner**. It takes the two vectors and the length of the vectors as arguments and returns the dot product.
- Call mathVecInner to find the dot product of initAtt and myAtt and store the returned float in dot.



```
dot = mathVecInner(initAtt,myAtt,3);
```

Find the Angle

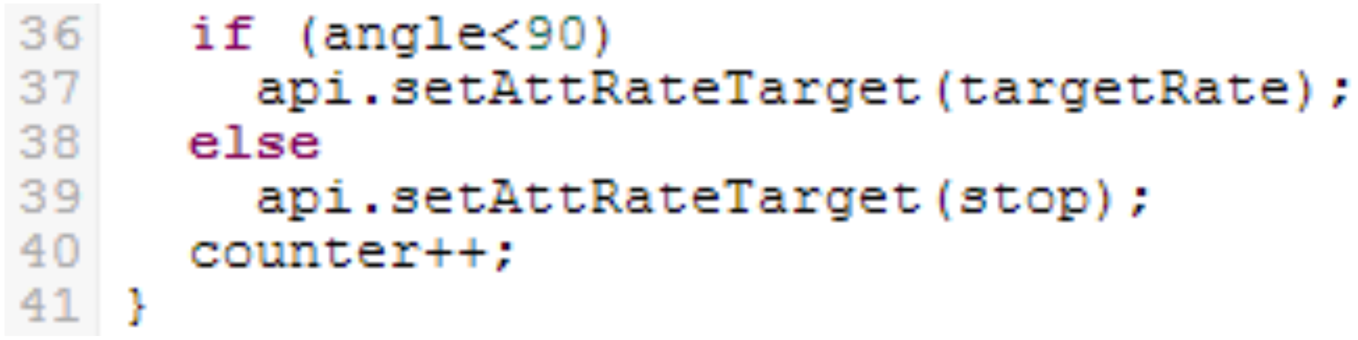
- Now we need to use our formula to find the angle between initAtt and myAtt. Use the function **acosf** to find the arccosine of dot.
- The formula requires you to divide the arccosine by the magnitudes of the two vectors. We can skip this step because both are attitude vectors and therefore unit vectors, so the magnitudes equal 1.
- acosf, like all other trig functions in math.h, returns a float in radians. The angle is easier to comprehend in degrees, so multiply it by 180/π.
- We want to monitor angular displacement, so debug angle.



```
angle = acosf(dot)*180/PI;
DEBUG(("%f",angle));
```

Conditional

- The only thing left to do is to change our “if” statement so it is conditional on angle, not counter. Set the attitude rate target to targetRate if the angular displacement is less than 90°.



```
  if (angle<90)
    api.setAttRateTarget(targetRate);
  else
    api.setAttRateTarget(stop);
  counter++;
}
```

- Compile and run. Make sure the console is open so you can monitor angle.

Results

- We didn't take into account the time it takes to stop moving, so we overshot the target angle. As an improvement, fine tune the conditional angle so you stop at 90°.
- But remember, versatility is everything. Your target angle won't always be 90° and you won't always be spinning at 15 deg/sec. Use your math and physics skills to create a solution that works in every situation.
- You may find the equations on the next slides helpful. Good luck!

Motion Equations Revisited

- The five essential motion equations of translational kinematics can be modified for their rotational counterparts.
- Keep in mind that these motion equations are only valid for constant angular acceleration (α).
- Remember that the subscript 0 indicates an initial value, and the lack of a subscript indicates a final value.

Motion Equations Revisited

$$\alpha = \frac{\omega - \omega_0}{t}$$

$$\theta = \frac{\omega + \omega_0}{2} t$$

$$\theta = \omega_0 t + \frac{1}{2} \alpha t^2$$

$$\theta = \omega t - \frac{1}{2} \alpha t^2$$

$$\omega^2 - \omega_0^2 = 2\alpha\theta$$