Revolving: Tilted Axis
• The polar coordinate method only works in 2-D. When the axis of revolution is tilted, the satellite needs to revolve in 3-D. To accomplish this, we will use the same method we used for spinning on a tilted axis: cross product.

• This tutorial will deal with a tilted axis at a random point that is not the origin. This is one of the most advanced tutorials, and we won’t explain everything. We want you to come up with a creative solution on your own.
The Method

• We will walk you through the technique before we get into the code. Imagine this is our scenario:
Cross Product

• Taking the cross product of the satellite’s relative position and the axis will give us a target position on the plane of rotation.
The target position is also relative to the center. We need to make it relative to the origin. To do this, we can use vector addition.
Try It

• Let’s code! Create a new project called Project21 and create the following arrays:
  – center
  – axis
  – myState
  – myPos
  – relPos
  – relTargetPos
  – targetPos
  – targetAtt

• Initialize center and axis with random values. We chose (0.2, 0.8, 0.5) for center and [0.5, 0.3, 1.0] for axis.

• In loop(), normalize axis.
SPHERES Position

- Get myState and write the position array to myPos.
- We need to find the satellite’s relative position. Make relPos a vector that points from center to myPos. You can accomplish this using vector subtraction.

```java
api.getZRState(myState);
for (int i=0; i<3; i++)
    myPos[i]=myState[i];
mathVecSubtract(relPos,myPos,center,3);
```
Target Position

- Now, we can find the cross product. Crossing relPos and axis will produce a target position relative to the center. Store this in relTargetPos.

- To set a position target, we need the position vector relative to the origin. Add the vectors center and relTargetPos to find targetPos.

- Set the position target.

```javascript
29 30 31
mathVecCross(relTargetPos, relPos, axis);
mathVecAdd(targetPos, center, relTargetPos, 3);
api.setPositionTarget(targetPos);
```
Face the Center

• All that’s left is to face the center. Use vector subtraction to find the vector pointing from myPos to center. Store this vector in targetAtt.
  – An alternative method is to negate the components of relPos, as relPos points from center to myPos.
• Normalize targetAtt and set the attitude target.
• Compile and run.

```c
33    mathVecSubtract(targetAtt, center, myPos, 3);
34    mathVecNormalize(targetAtt, 3);
35    api.setAttitudeTarget(targetAtt);
36    }
```
• The radius of the orbit approaches 0 so that the satellite ends up spinning at the center.
• One reason for this is that we don’t give the satellite enough time to reach the target position.
• If you experiment with a counter, you’ll find another reason. The cross product is dependent on the satellite’s relative position, so any fluctuations in position are amplified.
  – If the satellite is too close to the center, it will approach the center. Too far from the center, and it will stray farther and farther away.
Modifications

• There are plenty of ways to optimize revolution.
• Remember uniform circular motion? Try to keep a constant radius and angular speed as you revolve. Aim to create a solution that gives you as much control over the satellite as possible.
• Like spinning, revolving is an integral part of many Zero Robotics games. Try to come up with a versatile algorithm that your team can build off of from year to year.
• As always, work as a team to find a solution. Good luck!