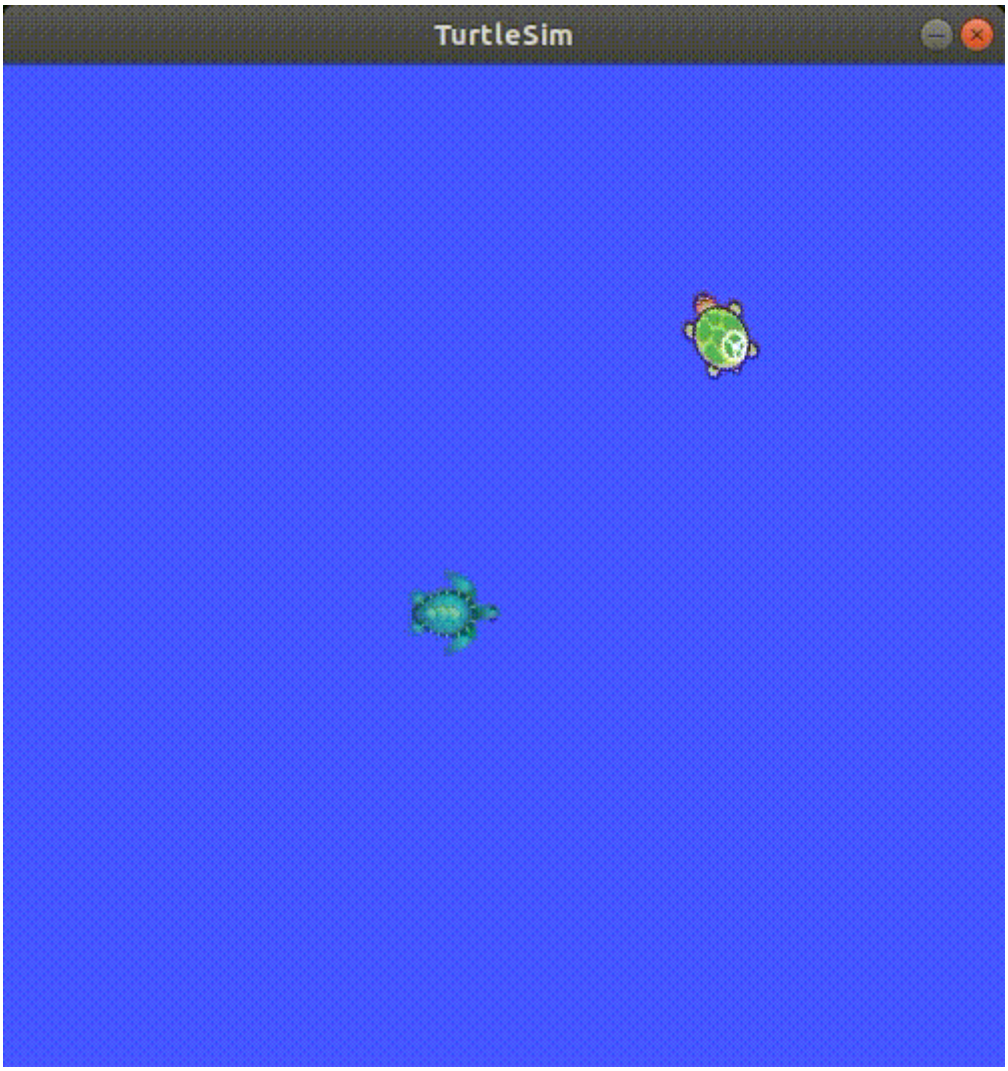
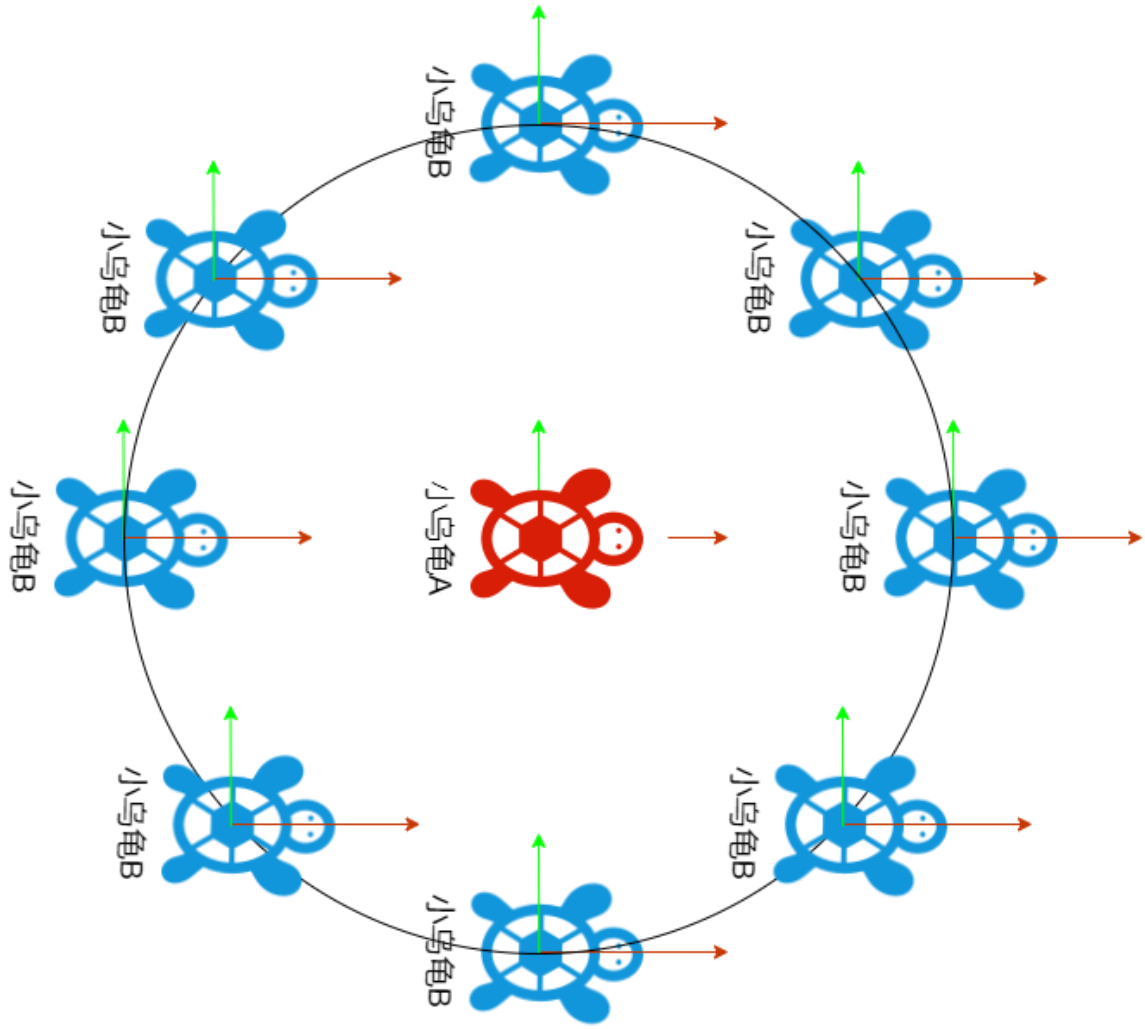
环绕效果



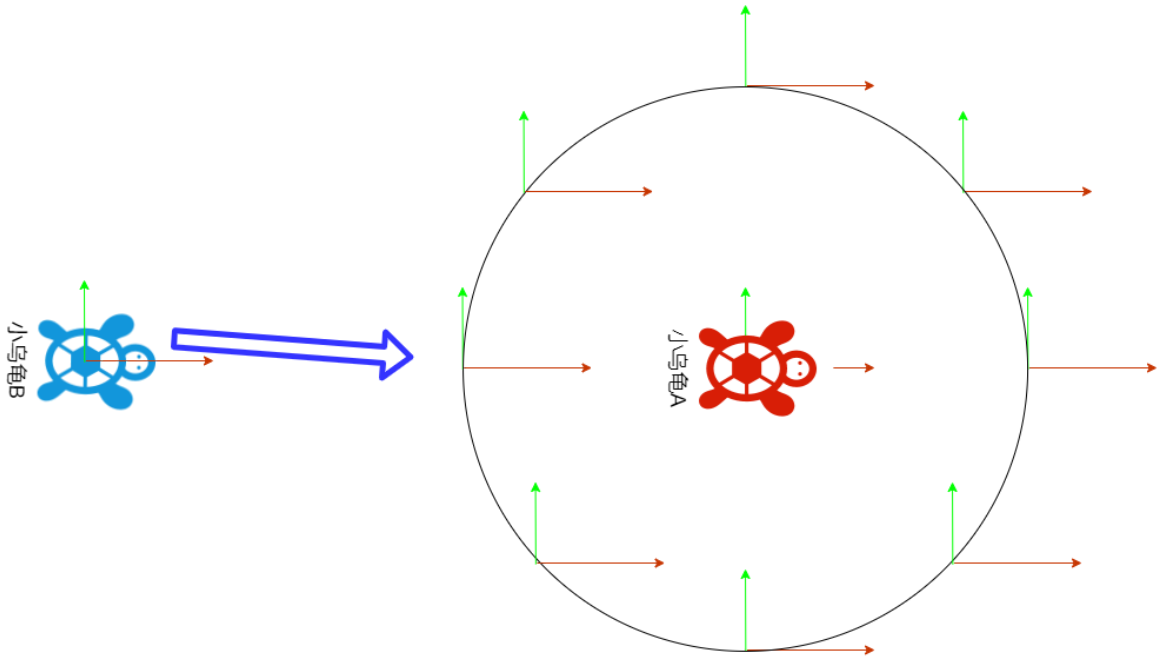
界面上有两只小乌龟：

- 小乌龟A
- 小乌龟B
- 小乌龟B绕着小乌龟A绕圈圈
- 小乌龟A通过键盘控制移动

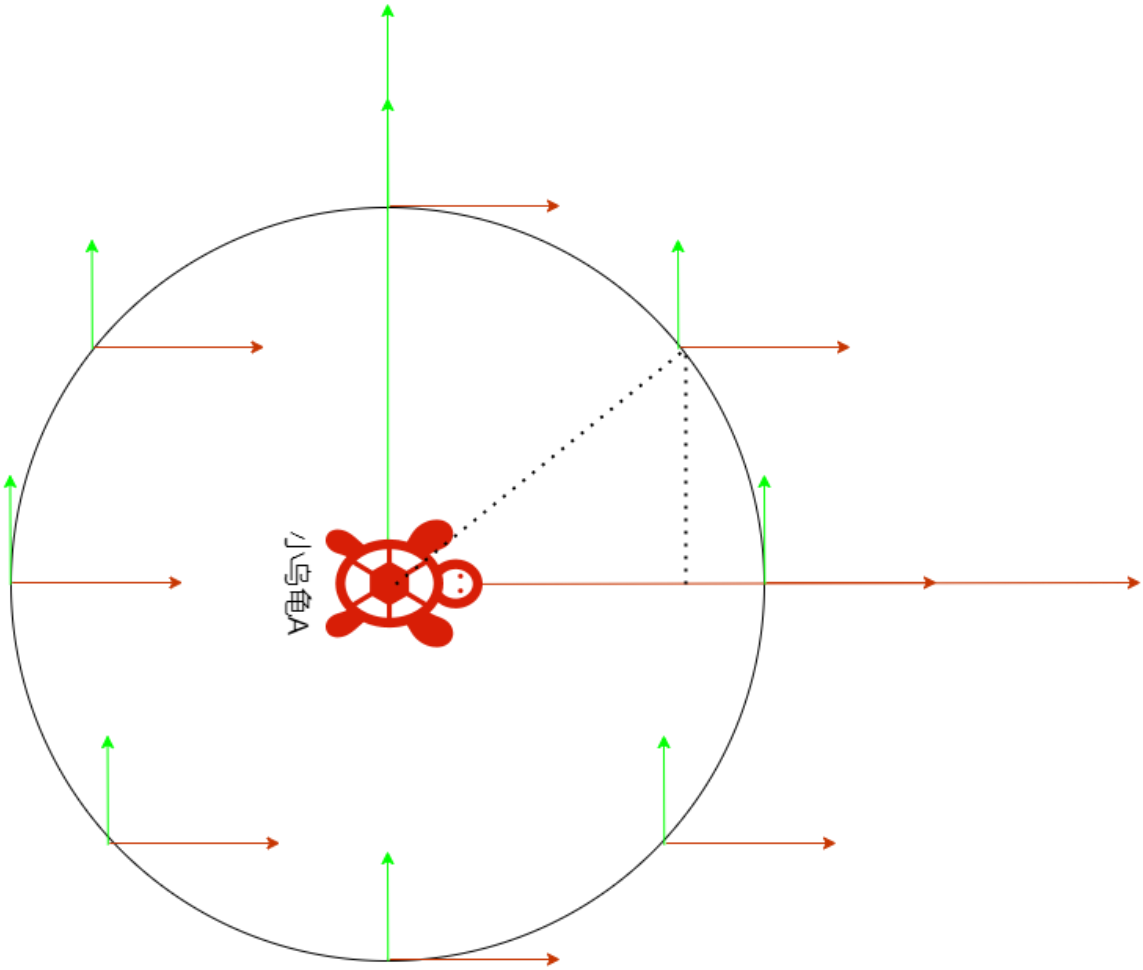
业务实现



小乌龟B期望在小乌龟A四周环绕运动



可以将环绕的点当成一个坐标系，小乌龟B想跟随这个坐标系运动



对于小乌龟A来说,环绕是一个相对的，小乌龟A可以动，但是对于环绕的坐标系，小乌龟是不动的。

对于不动的小乌龟A而言，要构建一个动态环绕的坐标系，需要通过单独的节点来发布小乌龟A和环绕坐标系的关系。

环绕坐标系可以按照一定的时间周期，做到动态坐标系变化，这里有几个关键的值：

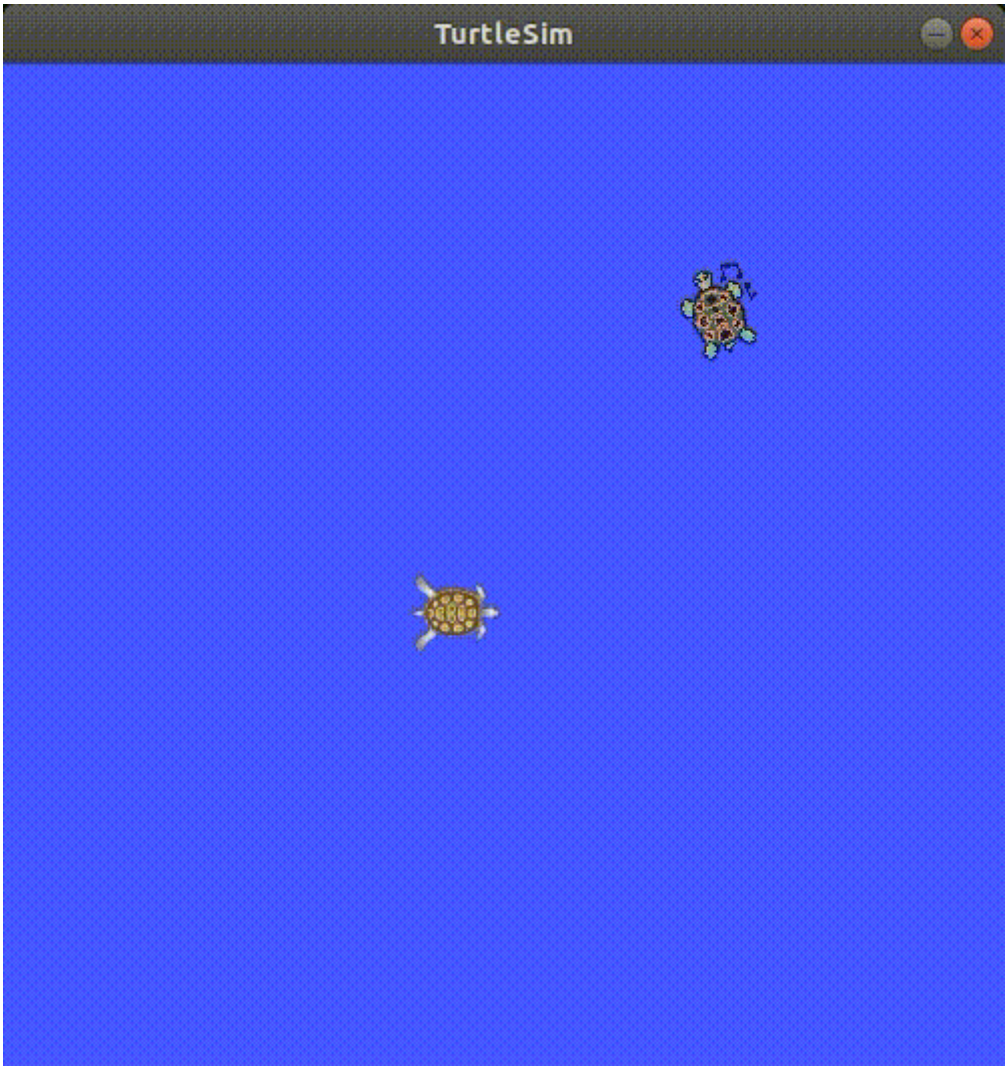
- 环的半径
- 环绕的角度

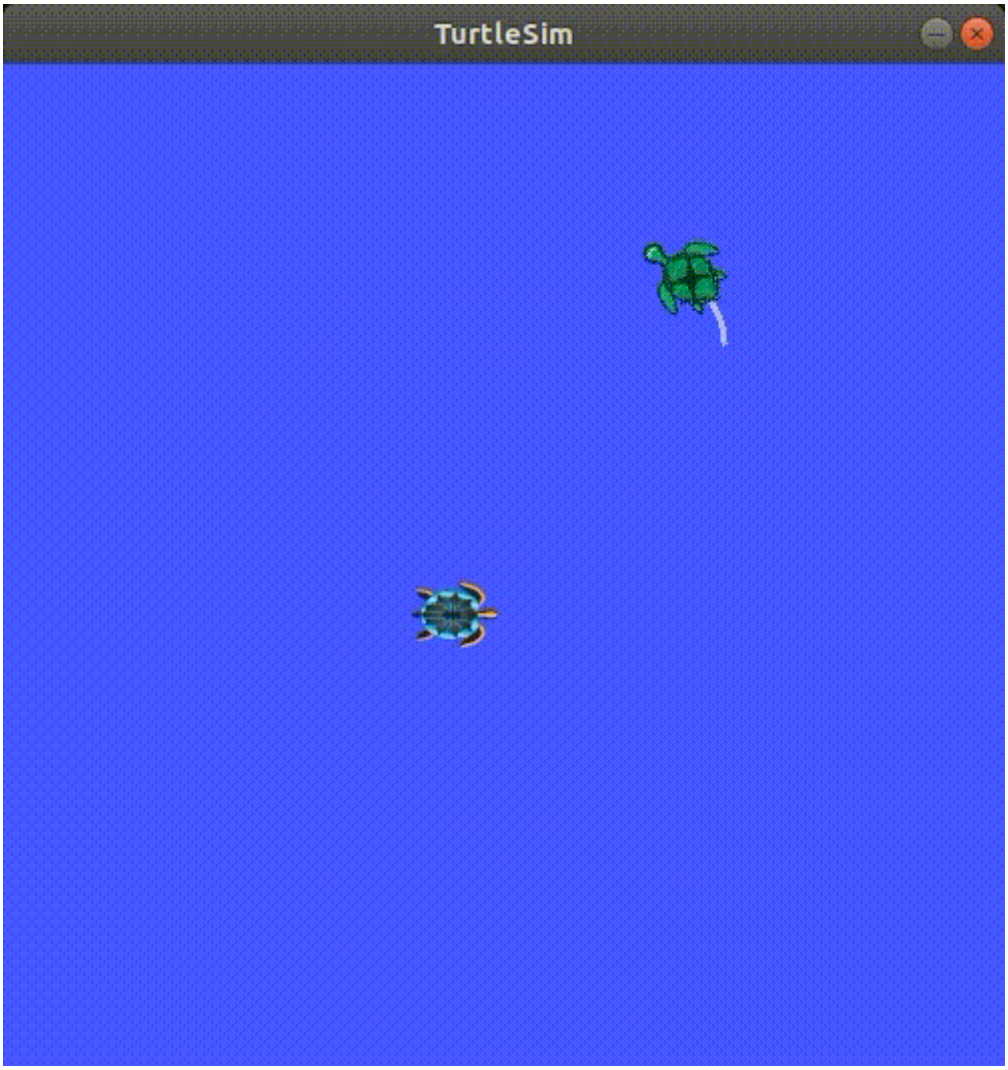
```
x = r * cos(radians(theta))
y = r * sin(radians(theta))

translation = (x, y, 0)
rotation = quaternion_from_euler(0, 0, 0)
broadcaster.sendTransform(translation, rotation, rospy.Time().now(),
'circle', "turtle_a")
```

!!!note
半径r和自增长的theta角度。

```
通过theta角度自增长来控制动态的环绕坐标系
```

笛卡尔爱心


TurtleSim



界面上有两只小乌龟：

- 小乌龟A
- 小乌龟B
- 小乌龟B绕着小乌龟A绕爱心
- 小乌龟A通过键盘控制移动

公式

笛卡尔爱心

```
x = a * (2 * sin(t) + sin(2 * t))
y = a * (2 * cos(t) + cos(2 * t))
```

!!! note
    t为自增长的弧度值

标准爱心

```
x = 16 * pow(sin(t), 3)
y = 13 * cos(t) - 5 * cos(2 * t) - 2 * cos(3 * t) - cos(4 * t)
```

!!! note
    t为自增长的弧度值