- **goal_z: 0.8** 机器人和人保持0.8m
- **z_threshold: 0.025** 人点云z的距离超出或低于0.8m±0.025m会给linear.x速度。速度是linear_speed = (z - self.goal_z) * self.z_scale即0.8m++会向前给正速度，0.8m--给负速度
- **slow_down_factor: 0.8** 如果追上了或者远离了人到了0.8m±0.025m会用这个参数给当前线速度和角速度减速
- **x_threshold: 0.025** 人点云x即左右移动超出或低于0.8m±0.025m给angular.z速度。（这里不能给全向轮y速度，因为如果人是转弯了，那y速度就不能跟着人转）。角速度是-x * self.x_scale。x是相对于相机坐标系的，所有x>0的话就在相机的右边，-x*self.x_scale的原因是机器人顺时针就是向右转的角速度是负的。
- **z_scale: 1.0** 线速度放大倍数
- **x_scale: 2.0** 角速度放大倍数
- **max_angular_speed: 5.0** 最大角速度
- **min_angular_speed: 0.1** 最小角速度
- **max_linear_speed: 0.4** 最大线速度
- **min_linear_speed: 0.05** 最小线速度

程序也是用一个Follower类的写法，可以一个类初始化把变量和处理用的回调函数都包含进去

类的初始化用了rospy.getparam获取上述参数，在.launch中可以用<rosparam file="${filename}.yaml" command="load" />导入参数。

depth_subscriber订阅了'point_cloud'，跳入回调函数set_cmd_vel，回调函数中，把点云读进来（*for point in point_cloud2.read_points(msg, skip_nans=True)*）skip_nans是排除坏点，然后用pt_x，pt_y，pt_z读取所有点的xyz坐标，后都加进x y z三个变量中，n变量用来计算多少个点云。

然后把所有点的x y z的总和除以n，得到所有点云的平均xyz，人在移动时，会改变局部的点云xyz，这样整体的平均xyz也会发生变化，就是用这个加上上面的linear和angular的调速改变机器人位置的。

学习一下py写Class的ROS系统：

- 创建个类 def __init__(self): 用来初始化里面的变量和publisher subscriber等。可以用self.变量名=rospy.get_param("变量名字", 预设值)，从外部可以用rosparam导入yaml。很方便的是回调函数中可以直接用self.变量名改这个类中的变量，publisher也可以在回调函数中把类中的变量pub出去，很方便
- 几个细节：self.depth_subscriber = rospy.Subscriber('point_cloud', PointCloud2, self.set_cmd_vel, queue_size=1）这里self.set_cmd_vel是回调函数。**可以注意到python里类中要写变量还是函数都需要加self.**，这个self就很像c++里的this指针。这里的queue_size=1等于一直处理当前的点云数据
- self.cmd_vel_pub.linear.x 和 self.cmd_vel_pub.angular.z是这个程序里发布的速度数据，在回调函数中依照上面的解释通过点云（环境改变小，人运动后改变x z大）修改了质心坐标，进而修改速度值，直接在回调函数里发布出去self.cmd_vel_pub.publish(self.move_cmd)
- 这样在if __name__ == "__main__": 的主函数中只需要建立这个类即可通过初始化（构造函数）建立整个pub sub系统