Tips and Tricks

We want this to be a place where we can share, together as a class, useful tricks we've found for programming robots (or anything else relevant to this class).

How to contribute to this document

Please add a new heading for whatever content you add. By using the heading functionality, the document's table of contents will automatically be updated. Additionally, the content will be navigable by software such as screen readers.

If you are modifying someone else's content, please turn on the "suggesting" mode before typing your changes. The exception to this rule would be if there is something very obvious that you are fixing (e.g., a typo).

Extracting x, y, yaw from a Pose

If you are working with Odometry, you've probably seen that the orientation of the robot is specified as a quaternion. These are somewhat convenient from a numerical stability standpoint, but inconvenient for our purposes in this class. Here is a helper function that converts from the Pose representation to a tuple of x, y, theta.

```
from tf.transformations import euler_from_quaternion

def convert_pose_to_xy_and_theta(pose):
   """ Convert pose (geometry_msgs.Pose) to a (x,y,yaw) tuple """
   orientation_tuple = (pose.orientation.x,
                        pose.orientation.y,
                        pose.orientation.z,
                        pose.orientation.w)
   angles = euler_from_quaternion(orientation_tuple)
    return (pose.position.x, pose.position.y, angles[2])
```

Computing the Difference Between Two Angles

If you are doing something like proportional control on the difference between your robot's heading and a desired heading, then you may have run into issues with the angle wrapping around. For instance, the angle .1 radians and 5.9 radians are actually quite similar, but if you just subtract them naively you wind up with a large difference.

There are many ways to handle this, and I encourage you to write your own code to handle this. If, however, you'd like to spend your time working on other problems, I have included some code to make this easier.

```
def angle_normalize(z):
   """ convenience function to map an angle to the range [-pi,pi] """
   return math.atan2(math.sin(z), math.cos(z))

def angle_diff(a, b):
   """ Calculates the difference between angle a and angle b (both should
be in radians)
        the difference is always based on the closest rotation from angle a
to angle b

       examples:
           angle_diff(.1,.2) -> -.1
           angle_diff(.1, 2*math.pi - .1) -> .2
           angle_diff(.1, .2+2*math.pi) -> -.1
   """
   a = angle_normalize(a)
   b = angle_normalize(b)

   d1 = a-b
   d2 = 2*math.pi - math.fabs(d1)
   if d1 > 0:
       d2 *= -1.0
   if math.fabs(d1) < math.fabs(d2):
       return d1
   else:
        return d2
```

Projecting Laser Scan into Odometry Frame

In some cases you may want to look at the laser scan data projected into the odom coordinate frame. You may already be doing this to some extent with some basic trig. The situation gets more complicated when you need to take into account the robot's heading (which would be necessary to project into the odom coordinate frame). For your convenience, we have a topic called /projected_stable_scan that has all of the laser scan points from stable_scan projected into the odom coordinate frame. If you are interested in using this, we can give you a quick overview.