```cpp
/*********************************************************************
*
* Software License Agreement (BSD License)
*
*  Copyright (c) 2009, Willow Garage, Inc.
*  All rights reserved.
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*   * Redistributions of source code must retain the above copyright
*     notice, this list of conditions and the following disclaimer.
*   * Redistributions in binary form must reproduce the above
*     copyright notice, this list of conditions and the following
*     disclaimer in the documentation and/or other materials provided
*     with the distribution.
*   * Neither the name of Willow Garage, Inc. nor the names of its
*     contributors may be used to endorse or promote products derived
*     from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
*  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
*  COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
*  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
*  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
*  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
*  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
*  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
*  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
*  POSSIBILITY OF SUCH DAMAGE.
*
* Author: Eitan Marder-Eppstein
*********************************************************************/
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <nav_msgs/Odometry.h>
#include "std_msgs/Int32.h"
#include "deadreckon.h"
#define ticksPerRev 210.461538 //ticks per rev is a double to the possibility of non whole number gear ratios due to how the encdoers are coupled
#define pi 3.141592
#define tau 2*pi
#define bad -2147483648
double baseWidth;
double wheelRadius;
int rightCount=bad,leftCount=bad;
void rin(const std_msgs::Int32ConstPtr &msg){
        rightCount = msg->data;
}

void lin(const std_msgs::Int32ConstPtr &msg){
        leftCount = msg->data;
}

int main(int argc, char** argv){
  ros::init(argc, argv, "odometry_publisher");
  ros::NodeHandle n;
  n.getParam("baseWidth", baseWidth);
  n.getParam("wheelRadius", wheelRadius);
  ros::Publisher odom_pub = n.advertise<nav_msgs::Odometry>("fuck/odom", 1);
  ros::Subscriber lsub = n.subscribe<std_msgs::Int32>("lwheel",1,lin);
  ros::Subscriber rsub = n.subscribe<std_msgs::Int32>("rwheel",1,rin);
//  tf::TransformBroadcaster odom_broadcaster;

  odomIntegral odomI(baseWidth,wheelRadius);//base and radius in meters. TODO get accurate values here


  ros::Time current_time, last_time;
  while(rightCount == bad || leftCount == bad){
  ros::spinOnce();
  }
  int lastValueR=rightCount,lastValueL=leftCount,curValueR,curValueL;
  ros::Rate r(120);
  last_time = ros::Time::now();
  while(n.ok()){
    ros::spinOnce();
    current_time = ros::Time::now();
    //compute odometry in a typical way given the velocities of the robot
    double dt = (current_time - last_time).toSec();
    if(dt != 0){
      curValueR = rightCount;
      curValueL = leftCount;
//    std::cout << curValueR << " " << curValueL << std::endl;
      double diffR = curValueR-lastValueR;
      double diffL = curValueL-lastValueL;
      double wR = (tau*diffR)/(ticksPerRev*dt); //revolution speed of the right wheel in radians per second. This is computed as an instantneous measurement since the last time we updated
      double wL = (tau*diffL)/(ticksPerRev*dt);
      odomI.proc(wL,wR,dt);  //this performs all the integration
      lastValueR = curValueR;
      lastValueL = curValueL;
    //since all odometry is 6DOF we'll need a quaternion created from yaw
      geometry_msgs::Quaternion odom_quat = tf::createQuaternionMsgFromYaw(odomI.theta);

    //first, we'll publish the transform over tf
    /*geometry_msgs::TransformStamped odom_trans;
    odom_trans.header.stamp = current_time;
    odom_trans.header.frame_id = "fuck/odom";
    odom_trans.child_frame_id = "base_footprint";

    odom_trans.transform.translation.x = odomI.x;
    odom_trans.transform.translation.y = odomI.y;
    odom_trans.transform.translation.z = 0.0;
    odom_trans.transform.rotation = odom_quat;

    //send the transform
    odom_broadcaster.sendTransform(odom_trans);
  */
    //next, we'll publish the odometry message over ROS
    nav_msgs::Odometry odom;
    odom.header.stamp = current_time;
    odom.header.frame_id = "odom";
    odom.child_frame_id = "odom_footprint";

    //set the position
    odom.pose.pose.position.x = odomI.x;
    odom.pose.pose.position.y = odomI.y;
    odom.pose.pose.position.z = 0.0;
    odom.pose.pose.orientation = odom_quat;
    odom.pose.covariance[0] = 1e-3;
    odom.pose.covariance[7] = 1e-3;
    odom.pose.covariance[35] = 1e-3;

    //set the velocity
    odom.twist.twist.linear.x = odomI.xdot;
    odom.twist.twist.linear.y = odomI.ydot;
    odom.twist.twist.angular.z = odomI.thetadot;
    odom.twist.covariance[0] = 1e-3;
    odom.twist.covariance[7] = 1e-3;
    odom.twist.covariance[35] = 1e-3;

    //publish the message
    odom_pub.publish(odom);
    }
    last_time = current_time;
//    std::cout << "x:" << odomI.x << " y:" << odomI.y << " t:" << odomI.theta << std::endl;
    r.sleep();
  }
}

```