Calibrating your odometry

Why calibrate your odometry?

The odometry of a robot is the measure of the robot's position and orientation (pose) in space. It is usually calculated by integrating the robot's velocity over time. This happens by sampling the rotation of the wheels as sensed by the wheel encoders. Knowing the rotation of each wheel in radians, the wheel radius, and the time between samples, you can calculate the distance traveled and the velocity of each wheel during that sample interval. By knowing the distance between the wheels, you can calculate the arc or line the robot traveled during the sample interval. By integrating these arcs or lines, you can calculate the robot's pose over time relative to where the robot was when it started up.

Note that correct integration relies on sampling happening over short distances. If the robot moves too far between samples along curved lines, the integration can be wildly off. This means that the odometry calculation should occur often, on the order of 20 to hundreds of times per second for a robot moving at a walking pace. More samples per second is better as a rule of thumb.

But wheel odometry is not perfect. There are errors in the measurements of the wheel encoders, such as noisy signals which may undercount or overcount the 'ticks' from the wheel encoders. The sampling interval is not precisely known. There are errors in the math used for the computation of the integration of the velocity. These errors can accumulate over time and cause the odometry to drift away from its actual pose. Also, odometry is an estimate of the robot's actual movement. The robot usually does not move exactly as predicted by the wheel encoders.

If the errors are too large or the odometry reports don't come in fast enough, the ability to create a map of the environment or to navigate through it can be compromised.

Odometry as described relies especially on two fundamental physical measurements of the robot: the wheel radius and the distance between the wheels. The wheel radius is the easier to estimate with fair precision using, say, a caliper, but that won't be enough. The effective wheel radius may change as the tire wears, or as the wheel tilts, or depend on the kind of surface the robot is moving on. Even knowing the wheel radius, the odometry may still be off if the wheel radius is not the same for both wheels. Robot wheels often slip on the ground as they turn, and usually in an unpredictable way.

The distance between the wheels is harder to estimate, and it is usually the source of the largest errors in odometry. The distance between the wheels can be estimated by measuring the distance between the wheel centers, but this is not always accurate and is affected by several factors. Tires are usually not , and the point of contact with the ground may change for a number of reasons, so it is hard to know where the contact distance between wheels is. The distance may be different when the robot is moving in a straight line versus when making a left turn versus when making a right turn. The wheels may be tilted, and each wheel can be tilted differently. The wheels might wobble, making the distance between them change as the wheels turn.

This is one of the reasons that I often say that everything about robots is hard. My example when I give talks about robots is that just because the robot's computer tells the motor controller to turn the wheels doesn't mean that the controller got that command. If the motor controller got a command to move the wheels and sent a signal for the motors to turn, it doesn't mean that the motors actually turned or turned as

commanded. If the motors turned, it doesn't mean that the wheels turned or turned as much as the motor shaft turned. If the wheels turned, it doesn't mean that the robot body moved or moved as much as the wheels turned. If the robot body moved, I can usually say that the robot did not go exactly where it was commanded to go.

It's usually true that sensors lie all the time, and it's the job of the software to figure out how much they are lying and correct for it. Fortunately, figuring out how much the odometry is lying is somewhat tractable as localization is usually a part of a closed-loop system. The software never just assumes that the robot moved as commanded; it also checks sensors like LIDARs, cameras, time-of-flight sensors, SONAR, IMUs, GPS, and so on to try to correct the reported odometry to agree with other sensors. The software tries to find the best truth from all of the lies.

Still, the more accurate the odometry, the better the robot can navigate and map. Conversely, if the odometry is very far off, the robot may not be able to navigate or map at all.

Calibrating the odometry, for the purposes of this chapter, is the process of trying to improve the measurement of the wheel diameter and the distance between the wheels. The process is usually done by driving the robot in a controlled way and comparing the reported odometry to the actual pose of the robot.

The following methodology assumes you are calibrating a two-wheel, differential drive robot. You should be able to generalize this process for a different kind of robot.

How to calibrate your odometry

I usually separate the calibration of the two measurements. The process is fundamentally the same for correcting the wheel radius and the distance between the wheels, but correcting the distance between the wheels can involve different equipment (well, a compass versus a scale) and a lot more iterations of the process.

The steps involved in both calibrations are, basically:

- Place the robot in a known pose.
- Read the odometry at the beginning of the iteration.
- Move the robot in a known way that should be affected primarily by just one of the wheel radii or the distance between the wheels.
- Read the odometry at the end of the iteration.
- Compare the actual pose of the robot to the reported odometry.
- Adjust the software values for the wheel radius or the distance between the wheels.
- Repeat until the odometry is as close as possible to the actual pose.

The assumption here is that you have the ability to alter the input to the process that is producing the odometry. Odometry readings come in two forms, as a topic and as a transform. If some manufacturer's software driver is producing the odometry, you should be able to alter the parameters of the driver to get the odometry to be more accurate. Often this comes in the form of a configuration file that you can edit, or is supplied as a parameter in a launch file. If you are writing your own odometry, as in writing your own motor driver software or writing, say, a class that implements the ros2_control API, you should be able to alter the code to use the calibrated wheel radius and distance between wheels.

Also note that these two measurements are likely to not be exactly the same as that used to create the URDF for the robot. The URDF is mostly used for visualization and collision detection, and the measurements

in the URDF are for visual purposes. You can use the actual, calibrated measurements in the URDF, but you can be less accurate in the URDF, though it's possible that you pass the calibration values via the URDF to the odometry code. I do not pass calibration data via the URDF. I have a separate configuration file for the calibrated measurements because they will change over time, especially as the robot is used and parts wear.

Now, pay attemtion to this. Remember all of those things mentioned above that can affect the odometry. The sampling interval isn't precise, wheels slip, and so on. Especially if you are computing odometry using a Linux process, you will be hit with process preemption and similar Linux issues which are going to make it hard to get good odometry. If at all possible, never compute odometry using a computer that is going to result in the odometry code not being called often or not being called at a regular interval. Even if you are using, say, an Arduino processor, make sure you understand now to make your loops run at a regular interval and that any interrupt handlers don't affect system timing.

Calibrating the wheel radius

Repeat the following steps until odometry measured from driving the robot in a straight line is pretty close between the observed physical position of the robot and the reported odometry. You need to do this measurement repeatedly. Don't assume that when it works once that you're done. Try it several times, maybe a dozen times. You may even find that odometry is accurate when, say, driving forward but is off when driving backwards. If that's your case, you will need to be creative in how you fix the robot to get the odometry to be accurate since this problem won't just affect driving in a straight line. If at all possible, fix your robot so that odometry is accurate whether driving forward or backward.

1. Place the robot in a known pose. Before you move the robot, mark its current position. The easy way to do this is to take a short length of blue, painter's tape and put it on the floor at the very bottom of one of the wheels, where it touches the ground. It doesn't have to be under the wheel, it could be beside the wheel. After the robot moves, you will put another piece of tape at the bottom of the same wheel and measure the distance, in meters, between the two pieces of tape.

2. Read the current odometry value via the following command. Replace the ***/odom*** topic with the topic that your odometry is published on.

   ```
   ros2 topic echo --once --flow-style /odom
   ```

   You will see output something like:

   ```
   header:
     stamp:
       sec: 16
       nanosec: 320000000
     frame_id: odom
   child_frame_id: base_link
   pose:
     pose:
       position:
         x: -3.40717210659393e-16
         y: -8.0459616687065e-36
   ```

```
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 5.0907089586332814e-20
      w: 1.0
  covariance:
```

With more lines after the ***covariance*** line. There will be other differences as well. The information you want to capture is the ***x*** value under ***position***. This is the current position of the robot. You will next drive the robot forward a bit and, ideally, only the ***x*** value will change noticeably, becoming more positive.

3. Move the robot forward a known distance. The way I do this is to use the ***teleop_keyboard_twist*** package, like so:

   ```
   ros2 run teleop_twist_keyboard teleop_twist_keyboard
   ```

   You will see a prompt in the terminal window. Press the ***i*** key to move the robot forward and then after the robot has moved forward a bit, ideally at least a meter, press the ***k*** key to stop the robot.

4. As above, read the current odometry value again and capture the ***x*** value under ***position***. The new ***x*** value minus the previous value is the distance the software thought the robot moved, mostly based on the wheel encoders. You should have found that the other values under the ***position*** and even under ***orientation*** have not changed much, if at all. The ***x*** value now should be more positive than the ***x*** value you captured before moving the robot.

5. Measure the distance between the two pieces of tape. This is the actual distance the robot moved. If the robot moved forward more than a meter, and the odometry distance and the actual distance are nearly identical, say within one or two millimeters, you are lucky, indeed. The goal is to get the odometry distances and the actual distances to be as close as possible, repeatedly. If this is so after, say, a dozen movements, forward and backward, you are done.

6. If the odometry distance and the actual distance are not close, you will need to adjust the wheel radius as known to your motor driver software. If the actual distance is greater than the odometry distance, you will need to increase the configured wheel radius. Likewise, if the actual distance is less than the odometry distance, you will need to decrease the wheel radius. You could calculate the new wheel radius by multiplying the current wheel radius by the ratio of the actual distance to the reported odometry distance. If that is too much math, you could just add or subtract a small amount to the current wheel radius value.

7. Repeat the process until the odometry distance and the actual distance are as close as possible. What you are likely to find is that as you change the wheel radius value, you will end up with finding that the actual distance is sometimes a little bit bigger and sometimes a little bit smaller than the odometry distance as you repeat the test. When you change the wheel radius to a larger or smaller value and you find that the error between the two distances will tend to be larger than with the old wheel radius value, the old radius value will likely be the best you can do for calibration.

Calibrating the distance between the wheels

This process is similar to that of calibrating the wheel radius. Repeat the following steps until the odometry measured from rotating the robot in place is pretty close between the observed physical heading of the robot and the heading from the odometry topic. You need to do this measurement repeatedly. Don't assume that when it works once that you're done. Try it several times, maybe a dozen times. You may even find that odometry is accurate when, say, turning left but is off when turning right.

To make this process easier, I've provided a LibreOffice Calc spreadsheet that will help you calculate the error in rotation of the robot. It can be found in the workspace at src/Robotics_Book/book/media/CalibratingRotation.ods.

Each time you are going to do a rotation, put the current distance between the wheels into the spreadsheet in the ***Wheel Spacing*** column. This is just for documentation purposes. To correct the distance between the wheels, you are going to observe the difference between the actual rotation of the robot and the rotation reported by the odometry, make a change to the configuration value of the distance between the wheels, do a 360-degree rotation again, and then observe the difference again. By keeping track of the distance between the wheels for each rotation test, you can see if you should be making the configuration distance between the wheels larger or smaller to get a smaller error in the actual versus the expected rotation.

Here are the steps:

1. Place the robot in a known pose. Before you move the robot, mark its current position. The easy way to do this is to use your phone with a built-in compass app and place the phone on the surface of your robot and don't move it until you are done calibrating, or to mark the robot's current position with a piece of tape by putting it on the ground beside one of the wheels at the bottom of the wheel where it touches the ground. You are going to rotate the robot in place and you want to try to rotate the robot 360 degrees in place, getting the marked wheel back to the original position.

2. Read the current odometry value via the following command. Replace the ***/odom*** topic with the topic that your odometry is published on.

   ```
   ros2 topic echo --once --flow-style /odom
   ```

   You will see output something like:

   ```
   header:
   stamp:
    sec: 16
    nanosec: 320000000
   frame_id: odom
   child_frame_id: base_link
   pose:
   pose:
    position:
      x: -3.40717210659393e-16
      y: -8.0459616687065e-36
      z: 0.0
   ```

```
  orientation:
    x: 0.0
    y: 0.0
    z: 5.0907089586332814e-20
    w: 1.0
  covariance:
```

With more lines after the ***covariance*** line. There will be other differences as well. The information you want to capture is the ***w*** and ***z*** values under ***orientation***. This is the current heading of the robot.

Place the ***w*** and ***z*** values into the spreadsheet in the ***Prev Odom W*** and ***Prev Odom Z*** cells. As mentioned above, you should always note the current configured distance between the wheels in the ***Wheel Spacing*** column. You will next rotate the robot in place and, ideally, only the ***w*** and ***z*** values will change noticeably.

If you are using a compass to track rotation, you can use the ***Compass Heading*** column to note the compass heading before you do the rotation.

3. Rotate the robot in place 360 degrees. The way I do this is to use the *__teleop_keyboard_twist__ package, like so:

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

You will see a prompt in the terminal window. Press the ***l*** key to rotate the robot clockwise in place or the ***j*** key to rotate anticlockwise. Then after the robot has rotated 360 degrees, press the ***k*** key to stop the robot. I found that I needed to jog the robot a bit to get to an accurate 360-degree rotation. This I did by tapping either the ***j*** or ***l*** key as needed for a short time and then pressing the ***k*** key to stop the robot. Remember, you are trying to get the robot back to the point where it was when you started the 360-degree rotation.

4. As above, read the current odometry value again and capture the ***w*** and ***z*** values under ***orientation***. If your robot is a well-designed, 2-wheel differential drive robot, the ***x*** and ***y*** values under ***orientation*** should also have not changed much from the values captured in step 2, nor should the ***x*** and ***y*** values under ***position***.

5. Using the LibreOffice Calc spreadsheet that I provided, put the new values of ***w*** and ***z*** from step 4 into the spreadsheet into the ***Odom W*** and ***Odom Z*** cells. The ***Heading Radians*** shows the reported current heading of the robot from the ***odom*** topic. The ***Rotated Radians*** shows the reported rotation in radians of the robot from the ***odom*** topic. The ***Rotated Degrees*** shows reported rotation in degrees—this value should be close to 360 or 0.

6. If ***Rotated Degrees*** value is more than 360 or 0 degrees, you will need to increase the configured distance between the wheels. If ***Rotated Degrees*** is less than 360 or 0 degrees, you will need to decrease the configured distance between the wheels. You could calculate the new distance between the wheels by multiplying the current distance between the wheels by the ratio of the actual rotation to the odometry rotation. If that is too much math, you could just add or subtract a small amount to the current distance between the wheels value.

7. Repeat the process until the odometry rotation and the actual rotation are as close as possible. What you are likely to find is that the ***Rotated Degrees*** gets closer and closer to the ideal 360 or 0 degrees as you change the distance between the wheels value, and then as you make further changes in the same direction (smaller or larger between iterations), the ***Rotated Degrees*** will start to get further away from the ideal 360 or 0 degrees. Go back to the settings that gave you the smallest error.