Odometry

Alexander Koldy

Forest Hills Robotics League

alexanderkoldy.ak@gmail.com

September 18, 2024

Introduction

In this lecture, we will discuss odometry in the context of the *FIRST Tech Challenge* (FTC). The topics this lecture will cover are:

- Mecanum odometry
- Dead-wheel odometry

Mecanum odometry

Let's start with a standard odometry example where each drive motor has its encoder plugged in.

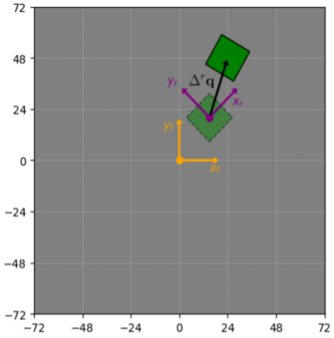
At each loop iteration k, the robot will have a small change in its x, y position and it's heading. Relative to the field, this will look like

$$\Delta^f \mathbf{q}_k = \left[(\Delta^f x)_k \quad (\Delta^f y)_k \quad (\Delta\theta)_k\right]^\top$$

However, our encoders raw measurements are defined as

$$\mathbf{y}_k = \left[\psi_{lf,k} \quad \psi_{lb,k} \quad \psi_{rb,k} \quad \psi_{rf,k}\right]^\top$$

where $\psi_{i,k}$ is raw value (in ticks) returned by a call to the `getPosition` function of the ith motor. Let's see how we can convert $\mathbf{y}$ into $\Delta^r\mathbf{q}$ with some basic geometry.



Mecanum odometry

Since the encoders do not get reset after each loop, we must subtract the raw tick values from the previous loop iteration:

$$\Delta \mathbf{y}_k = \mathbf{y}_k - \mathbf{y}_{k-1}$$

Next, we convert these raw tick values to revolutions. Most encoders will have a tick per revolution spec for us to find in their data sheet. Additionally, let us consider the gear ratio between the output shaft of the motor and the wheel shaft. For example, if the wheel spins twice for every full rotation of the motor shaft, we multiply by two. Thus, our change in rotation (in revolutions) is

$$\Delta \phi_k = \frac{\text{gear ratio}}{\text{ticks per revolution}} \Delta \mathbf{y}_k$$

Note: dividing ticks by the ticks per revolution simply yields revolutions.

To convert these revolutions to a distance, we use the circumference formula and multiply by revolutions:

$$\Delta \mathbf{w}_k = 2\pi r \Delta \phi_k$$

where r is the radius of the mecanum wheel.

Mecanum odometry

The vector $\Delta \mathbf{w}_k$ yields the change in each wheel position between the kth and $(k-1)$th loop iteration. We can simply apply our forward kinematics matrix $\mathbf{H}_{fk}$ to yield $\Delta^r \mathbf{q}_k$:

$$\Delta^r \mathbf{q}_k = \mathbf{H}_{fk} \Delta \mathbf{w}_k$$

This gives us the relative change in the robot's position in the robot frame at loop iteration $k-1$. However, we would like to know the robot's pose relative to the field frame. To do this, we simply rotate the vector $\Delta^r \mathbf{q}_k$ to the field frame and add it to our existing pose measurement, i.e.,

$$\mathbf{q}_k = \mathbf{q}_{k-1} + \mathbf{R}_{r \to f} \left(\Delta^r \mathbf{q}_k \right)$$

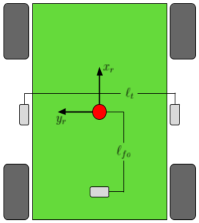
If we are keeping track of the full state x, we can simply repeat these steps with a function call to `getVelocity`. In this case, we would start with

$$\mathbf{y}_k = \begin{bmatrix} \psi_{lf,k} & \psi_{lb,k} & \psi_{rb,k} & \psi_{rf,k} & \dot{\psi}_{lf,k} & \dot{\psi}_{lb,k} & \dot{\psi}_{rb,k} & \dot{\psi}_{rf,k} \end{bmatrix}^\top$$

where $\dot{\psi}$ is the raw velocity in ticks per second. In this case, the subtraction between $\mathbf{y}_k$ and $\mathbf{y}_{k-1}$ would only happen for the first four elements of $\mathbf{y}$.

Dead-wheel odometry

One of the main issues with mecanum-wheeled odometry is wheel slip. This introduces a high amount of variability and noise which is propogated through the mathematical model and yields poor localization results. This has led teams to adopt **deal-wheel odometry**, which uses sprung pods containing encoders attached to omni wheels.



We will use three dead wheels total, two parallel and one perpendicular pod (though some teams simply run one of each and use a separate sensor to determine heading). The distance between the two parallel wheels is our dead wheel trackwidth (ℓ_t) and the distance between our center of rotation and our perpendicular wheel is our forward offset (ℓ_{fo}).

Dead-wheel odometry - set up

Before we continue with the mathematics, it is important we set up our encoders correctly. The REV Control Hub, has four encoder ports (these are the same ports your drive motors would be plugged into). Ports 2 and 3 are “software” ports and will skip ticks with certain encoders. Typically, we will reserve one of these ports for the perpendicular dead wheel as it is less important to localization accuracy. Thus, the parallel encoders should be plugged into ports 1 and 3, i.e., the “hardware ports”.

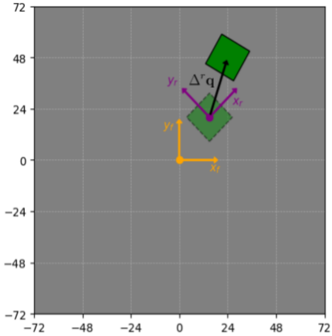
Dead-wheel odometry

Like with mecanum odometry, our first goal is to find the relative change in position and heading, i.e., $(\Delta^r \mathbf{q})_k$ at the current loop iteration.

Let's define our raw encoder measurements as

$$\mathbf{y}_k = \begin{bmatrix} \psi_{l,k} & \psi_{r,k} & \psi_{p,k} \end{bmatrix}^\top$$

where l, r and p are left, right and perpendicular, respectively.



We then follow the same process as Slide 4, i.e.

$$\Delta \mathbf{y}_k = \mathbf{y}_k - \mathbf{y}_{k-1}$$

$$\Delta \phi_k = \frac{\text{gear ratio}}{\text{ticks per revolution}} \Delta \mathbf{y}_k$$

$$\Delta \mathbf{w}_k = 2\pi r \Delta \phi_k$$

where r is the radius of the dead-wheel. The gear ratio is likely 1, since most pods do not have a gear train between the encoder and the wheel.

Dead-wheel odometry

We have that

$$\Delta \mathbf{w}_k = \begin{bmatrix} \Delta d_{l,k} & \Delta d_{r,k} & \Delta d_{p,k} \end{bmatrix}^\top$$

where Δd is the change in distance traveled for each wheel between iteration k and $k-1$. We first figure that the relative change in the x position of the robot should be the average of the two parallel values, i.e.,

$$\Delta^r x = \frac{\Delta d_l + \Delta d_r}{2}$$

Next, we find the change in heading

$$\Delta \theta = \frac{\Delta d_r - \Delta d_l}{\ell_t}$$

Finally, we can find our change in the y position with

$$\Delta^r y = \Delta d_p - \ell_{fo} \Delta \theta$$

We now have all elements of $(\Delta^r \mathbf{q})_k$, and can therefore rotate it to the field frame and add it to the robot's pose from the previous loop iteration:

$$\mathbf{q}_k = \mathbf{q}_{k-1} + \mathbf{R}_{r \to f} \left(\Delta^r \mathbf{q}_k \right)$$

Note: $\mathbf{R}$ *is calculated using* θ_{k-1}

Dead-wheel odometry - pose exponentials

The previous implementation assumes the robot moves in a straight line between loop iterations. This is not always a great assumption, so we can improve our estimate by assuming the robot travels around the arc of a circle with constant velocity. You can learn more about this in this video and its sequel. For now, we will call matrix **E** the pose exponential matrix (i.e., the matrix containing the this constant velocity assumption math):

$$\mathbf{E}_k = \begin{bmatrix} \frac{\sin(\Delta\theta_k)}{\Delta\theta_k} & \frac{\cos(\Delta\theta_k)-1}{\Delta\theta_k} & 0 \\ \frac{1-\cos(\Delta\theta_k)}{\Delta\theta_k} & \frac{\sin(\Delta\theta_k)}{\Delta\theta_k} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We simply matrix multiply $\mathbf{E}_k$ by $(\Delta^r\mathbf{q})_k$ before rotating to the field frame as follows:

$$\mathbf{q}_k = \mathbf{q}_{k-1} + \mathbf{R}_{r\to f}\mathbf{E}_k\left(\Delta^r\mathbf{q}_k\right)$$

Note: ensure the edge case where $\Delta\theta = 0$ is handled!

Dead-wheel odometry - verifying

To verify that our odometry is working correctly, we should write a `TuneLocalizer.java` file. This should be an op-mode that prints at least the following information.

- Loop time in milliseconds
- Raw left, right and horizontal encoder reading in ticks
- True x and y position in inches as well as the true heading in degrees (since units of degrees are easier to analyze over units of radians).

When we run the op-mode, we should first check that all three encoder readings are changing when we push the robot randomly. If one or more of the readings stays at 0, then either the encoder is unplugged, or the encoder is tied to the wrong motor in code.

Next, we should verify that encoders are set up in the correct direction. The left and right encoders should increase (i.e., become more positive) when pushing the robot forward and the horizontal encoder should increase when pushing the robot to the left. This lines up with our robot frame. If any encoder does not follow this convention, we simply negate its reading in code.

Dead-wheel odometry - tuning wheel radius

Let's say you manually push your robot in a straight line along its x axis 100 inches. In practice, the result of the dead-wheel calculation may yield something like 99 or 101 inches. To counteract this inconsistency, we can introduce two multipliers to our raw dead-wheel readings: λ_x and λ_y. The readings of the left and right encoders are multiplied by λ_x and the reading of the horizontal encoder is multiplied by λ_y. To tune these constants, we can go through the following process:

1. Set up a long stretch of tiles (at least 96 inches, i.e., four tiles, worth). Ensure you have a tape measure lined up to the edge of the tiles to confirm the true measurement.
2. Line up the front of the robot with the 0 inch mark.
3. Run a localization tuner/tester op-mode.
4. Push the robot until the front is lined up with the [for example] 96 inch mark. Try to keep the robot as straight as possible.
5. Calculate $\lambda_x = \frac{\text{true distance}}{\text{odometry distance}}$
6. Repeat the process 3 to 5 times and take the average multiplier.

This process is repeated for λ_y but instead the robot should be pushed sideways.