An Introduction to Simultaneous Localization and Mapping for Mobile Robots

# Introduction

Simultaneous Localization and Mapping (SLAM) is the problem of tracking one’s position (localization) within a map while still being in the process of building said map. This is a type of “chicken and egg” problem, because your position is described with reference to a map of some type, but to build a map you must know where you are. This problem deals largely with how we handle “dual uncertainties”, which we will explore later.

To start, let’s focus on strictly the localization aspect. Localizing a robot can be a challenging problem on its own. Mobile robots are often equipped with inertial measurement units (IMUs) and wheel encoders, which can provide odometry information (linear and rotational velocities).

<insert diagram indicating odometry sensor information>

If we know the robot’s velocity at any given moment, we can take samples at specific time intervals, multiply that time interval by the velocity, and get some trajectory for that time interval that the robot traveled. We can add these trajectories together to get the robot’s overall position within a coordinate frame! This process is known as “dead reckoning”.

Diagram

Description automatically generated

While this may seem like a solution to the localization problem, dead reckoning is subject to very serious “drift” in the estimate due to accrued errors in the odometry reading. For instance, if a wheel slips, the robot has no way of knowing that happened, and those wheel rotations will be incorporated into its estimate.

Consider this example: you are walking down a hallway to get into your bedroom. The hallway has a bend in it, and your bedroom is on the right. If you start at the beginning of the hallway, close your eyes, and walk down the hallway and attempt to go into the bedroom, could you do it? Perhaps, but you probably would feel anxious doing so and likely bump into a doorframe or wall. This is because you have some idea how long each step you take is, but there is some error in that estimate. You usually get where your going accurately by opening your eyes and using visual information to correct your motion.

This is the same for mobile robots. Mobile robots cannot rely on odometry sensors alone and must make observations in their environment to correct their erroneous motions. Robots will typically utilize features with known positions within its environment for motion correction.

# Modeling Robot Motion

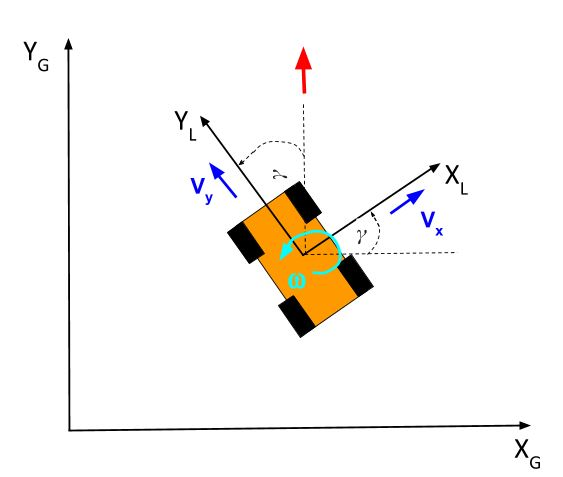
To begin, we’ll develop a set of equation to describe a robot’s pose in 2D space. The term pose refers to the robot’s position (x,y) and orientation (theta). We’ll assume through some kind of odometric sensor the robot is able to read its instantaneous velocities in the form of linear velocity and rotational velocity. Linear velocity is how quickly the robot is moving “forward” and rotational is how quickly it is “turning”.

To understand these items, we need to define a couple of reference frames. First, we have the global / world frame. This is the coordinate frame that represents the environment the robot operates within. The robot’s position in x,y and orientation, theta, is described within these axes. We also have the robot’s local frame / base frame. This frame is used to describe positions and orientations of objects from the robot’s perspective.

Chart

Description automatically generated

Typically, when modelling a robot, we will describe the robot’s position as a single point at the center of the robot. The x axis of the robot’s local frame is typically oriented towards the “front” of the robot, or the direction the robot would move in if it where to drive “forward”. The odometric information, the linear and rotational velocities, are usually reported from the robot’s local frame. This means the robot’s linear velocity is along the x axis, and rotational velocity is about the z axis (which protrudes out of the page. This is often referred to as “yaw” in 3D space).



If we are reading linear and rotational velocities in the robot’s local frame, and we want to track the robot’s position within a global frame, we need to transform these velocities into the x, y and omega velocities within the global frame.

We transform the linear velocity of the robot into x and y velocities in the global frame by multiplying it by the sine and cosine of the velocity using the robot’s orientation:

Where *vxg* and *vyg* is the robot’s velocities along the *x* and *y* axes, respectively, *vxl* is the robot’s linear velocity along the x axis of its local frame, and theta g is the robot’s orientation in the global frame. We then update the robot’s global pose by multiplying these velocities by the period of time over which they were measured.

Where xg and yg is the robot’s x and y position in the global frame, respectively, delta t is the period of time over which the velocities where measured, and omega is the robot’s rotational velocity.

This method of updating the robot’s position models the robot’s motion as a combination of moving forward and turning. There are kinematic models that better describe the robot’s motion, but so long as delta t is very small, this method works well enough.

## Example:

Let’s consider that we have a robot with the pose (2,3) at 45 degrees. We are taking odometry measurements at every 0.5 seconds. If we measure 0.7 meters per second and 3 degrees per second, followed by another measurement of 0.3 meters per second and 20 degrees per second. What is the robot’s pose after these two measurements?

We’ll denote the robot’s position at specific time intervals with the subscript k. At interval k, the robot’s pose in the global can be described as:

We can compute the next x,y position using:

And the new orientation:

Solving for time step k+1, we get:

Then, using the new values for *k+1*, we solve for iteration *k+2*:

Therefore, the robot’s pose within the global frame after two iterations can be described as:

We can plot the robot’s position at each time step:

A picture containing chart

Description automatically generated

# Dead Reckoning as a Non-Linear System of Equations

Formally, we can describe the robot’s pose as the state of its system. We will use the variable, x, to represent the system’s state (the vector used to describe the robot’s pose).

Using the state vector, we can describe the dead reckoning methodology as a vector-valued function:

Where *x0* indicates the first element of the state vector, *x1* is the second, and so on. This system of equations is non-linear due to the x2 / theta component that exists as a parameter of the sine and cosine functions.

Velocities as Control Inputs