Mobile Robots Lab Report

Noah Harvey

December 1, 2014

Abstract

The robot localization problem is a key problem faced in modern day mobile robots. This problem involves obtaining "clean" data about the robot's environment, determining the current location of the robot, and deciding on an optimal travel path towards a desired location. The robot receives information from its environment via robotic sensors (wheel encoders and sonar range finders). The robot then uses the sensor data and information about its previous location to determine its current location.

Mobile robot simulation software is a useful tool for designing and testing localization algorithms. Aria is a C++ SDK used for developing and testing mobile robot software. MobileSim is a simulation software for testing mobile robot software which uses Aria. Both Aria and MobileSim were used to develop and test mobile robot software.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Dead Reckoning

Robot localization is crucial to effective robot locomotion. Various methods are used to determine the current and future locations of robot; each unique to the available sensors, robot structure and environment. Dead reckoning is one of the simpler methods used. It requires as few sensors as possible and relies mostly on known motion models and states. Dead reckoning is a method of finding the location of an object based on a known motion model and current state information. Once the current state of an object is known then its motion model can be used to predict future motion states. For example, one can predict the future position of a robot if it is known that that the robot is moving in a straight line and at a constant speed and its current position.

A two wheeled robot as shown in Figure 1 will be used for our analysis of dead reckoning. Two wheeled odometry is used to create a motion model of the robot which is then used to predict future locations of the robot. In this study only the x and y coordinates are of interest although velocity can easily derived.

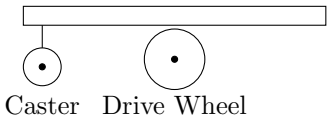

Figure 1: 2 Wheeled Robot (Side view)

We will begin our analysis with the odometry model of the two wheeled robot. The odometry model is as follows (its derivation is beyond the scope of this report):

$$P(t) = \begin{bmatrix} S \\ \theta \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{\mathrm{w}} & -\frac{1}{\mathrm{w}} \end{bmatrix} \begin{bmatrix} S_R(t) \\ S_L(t) \end{bmatrix} \tag{1}$$

Where S and θ respectively are the robot distance travelled and heading of the robot. S_L and S_R are the distances traveled by each wheel (left and right respectively). w is the distance between the contact points of the wheels of the robot.

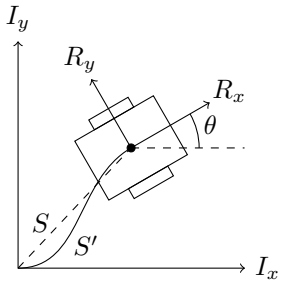

Figure 2: Robot Motion Model from $t = 0-$ Δt

From Equation 1 we can create a motion model relative to the inertial frame of reference I (as shown in Figure 2). To simplify our analysis the following is assumed:

- The robot's motion model is measured on a time differential Δt that is small with respect to the total time (T) the robot performs its motion.
- The robot's wheel encoders are error-less and no wheel slipping occurs.
- Δt is chosen to be small enough so that $S' - S \approx 0$ and thus $S' = S$.

From these assumptions we have a motion model for the robot:

$$\xi_i = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} P(t_i + \Delta\mathrm{t}) - P(t_i) \tag{2}$$

ξ_i is the position of the robot after some time lapse Δt . Thus to get the current position of the robot after some time T simply sum the interval position changes[1]:

$$\xi(T) = \xi_0 + \sum_{i=0}^{n} \xi_i : n = \frac{T}{\Delta\mathrm{t}} \tag{3}$$

Equation 3 simply means that given the robots initial position (ξ_0) and a sequence of wheel distance measurements for $t = \{\Delta\mathrm{t}, 2\Delta\mathrm{t}, \cdots, T\}$ the position of the robot at T can be predicted. This of course is only valid for the above assumptions.

Dead reckoning is not practical when used alone. Sensor noise, wheel slippage, and distance estimation error ($S - S'$) all affect the outcome of Equation 3. However dead reckoning is not completely useless. In fact it forms the base motion model for other filter and motion estimation algorithms (such as the Kalman filter[2]).

1.2 Wheel Encoders

As seen in Equation 1 the robot's motion model is dependent on wheel distance travelled. Sensors that can translate rotational wheel motion to linear distance traveled are called wheel encoders. Different types of encoders are available each with their advantages and disadvantages. In this study we will examine a quadrature wheel encoder.

A quadrature encoder uses a disk (as shown in Figure 3) and two optical sensors to track the position of a wheel. The disk is patterned with alternating opaque and transparent wedges. As the disk spins the optical sensors produce a logic signal (0 or 1) respective to the wedge type it is over. The two sensors are placed in such a manner that their outputs are 90° out of phase (see Figure 4). The encoder is coupled with the wheel in such a manner that the disk spins with an angular velocity that is proportional to that of the wheel.

The output from the encoder can be viewed as a 4 phase signal (see Table 1). Each phase change is called a "pulse" and is detected by an observing hardware (typically a MCU). Software can then be used to determine the rotation state of the wheel by analyzing the encoder phase after every pulse it produces.

The direction of the encoder can be determined by two successive phase changes. For example if the current phase is 2 and on the next encoder reading the phase is now 1 the encoder is rotating

[1] A continuous form of Equation 3 is beyond ths scope of this lab and is not needed computationally.

[2] `https://en.wikipedia.org/wiki/Kalman_filter`

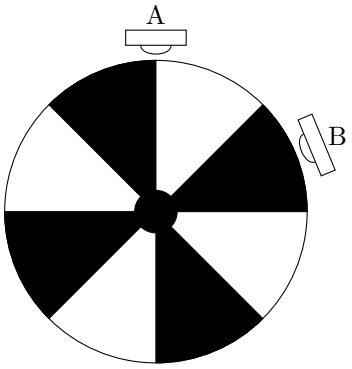

Figure 3: Quadrature Encoder

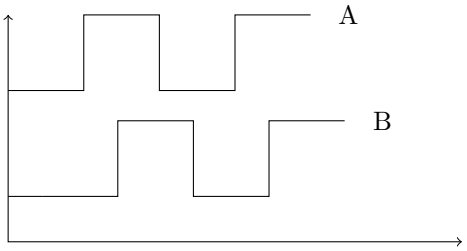

Figure 4: Quadrature Encoder Signals (90° phase shift)

clockwise. If determining the encoder phase is not performed within a certain amount of time then the software will miss some encoder ticks resulting in erroneous calculations.

Because the encoder uses wedges to indicate rotation the encoder's resolution is restricted to a certain number of pulses per rotation angle. This means that their is a minimum distance the wheel must travel in order for the encoders to detect motion.

The lateral distance that each wheel travels is:

$$S_{L,R} = R\mu \sum_{i=0}^{n} (-1)^{f(\rho_i, \rho_{i-1})} \tag{4}$$

Where $S_{L,R}$ is the lateral distance of the wheel traveled, μ is the encoder resolution in units of $[\frac{radians}{pulse}]$, n is the number of pulses, and R is the radius of the wheel the encoder is coupled with. $f(\rho_i, \rho_{i-1})$ is a function that takes the "current" phase and the previous phase and returns 2 or 1 if the direction between the two phases is forward or reverse respectively.

The translational velocity of the wheels can be found from Equation 4:

$$\dot{S}_{L,R} = \frac{S_{L,R}}{\Delta \mathrm{t}} \tag{5}$$

Phase	A Signal	B signal
0	0	0
1	1	0
2	1	1
3	0	1

Table 1: Phases of Quadrature Encoder

Note that Δt is the same as that defined in section §1.1. This allows synchronization between the movement of the robot to the quantifying of the robot motion state.

2 Methods

2.1 Square Path Robot

2.2 Sonar Measure Robot

2.3 S-Curve Robot

3 Results

4 Conclusion