CHEAP OUTDOORS AUTONOMOUS NAVIGATION WITH ROS

by

Noah Johnson

A Thesis

Submitted to the Division of Natural Sciences New College of Florida in partial fulfillment of the requirements for the degree Bachelor of Arts under the sponsorship of Professor Gary Kalmanovich

Sarasota, Florida May 2017

Acknowledgments

This is the acknowledgements section. You should replace this with your own acknowledgements.

Contents

Cheap Outdoors Autonomous Navigation with ROS

by

Noah Johnson

Submitted to the Division of Natural Sciences
on May 23, 2017, in partial fulfillment of the
requirements for the degrees of
Bachelor of Arts in Computer Science
and
Bachelor of Arts in Applied Mathematics

Abstract

In this thesis, I designed and implemented a compiler which performs optimizations that reduce the number of low-level floating point operations necessary for a specific task; this involves the optimization of chains of floating point operations as well as the implementation of a "fixed" point data type that allows some floating point operations to simulated with integer arithmetic. The source language of the compiler is a subset of C, and the destination language is assembly language for a micro-floating point CPU. An instruction-level simulator of the CPU was written to allow testing of the code. A series of test pieces of codes was compiled, both with and without optimization, to determine how effective these optimizations were.

Professor Gary Kalmanovich .....

May 23, 2017

Chapter 1

Introduction

1.1 Goal

Create a cheap outdoors autonomous robot. Given a 'map' of the New College campus, this robot should be able to navigate from one outdoors location to another using footpaths. In doing so, it should dynamically avoid obstacles such as people, and recalculate alternate routes when a route is unexpectedly blocked.

1.2 Navigation

intro to the concept of navigation

Since a map with GPS coordinates is provided, this is not Simultaneous Localization and Mapping (SLAM), but a simplified navigation problem.

1.3 Choice of Sensors

IR, Microsoft Kinect can't use because the rover will be outdoors during the day and the sun gives off ambient IR radiation.

LIDAR - state of the art

And LIDAR sensors would be too costly. constrained by cheap hardware, and the inability to use IR or LIDAR sensors.

Chapter 2

Theory

2.1 Probability Background

Discrete random variables have a finite output space of possible values that may be observed. Let X be a random variable, then we define the probability that we observe value x from X as $p(X = x) \equiv p(x)$. Since x is arbitrary, this defines a probability distribution. For every random variable X, we have

$$\sum_{x \in X} p(x) = 1$$

Given two more random variables Y and Z, we'll define the joint distribution $p(X = x \text{ and } Y = y \text{ and } Z = z) \equiv p(x, y, z)$, and the conditional probability $p(X = x \text{ given that } Y = y \text{ and } Z = z) \equiv p(x \mid y, z)$. The conditional probability is defined to be

$$p(x \mid y, z) = \frac{p(x, y, z)}{p(y, z)} \tag{2.1}$$

The *Law of Total Probability* states that $p(x) = \sum_{y \in Y} p(x, y)$. Extending this law to use a third random variable Z, and incorporating the definition of conditional

probability, we end up with the following equation:

$$p(x \mid z) = \sum_{y \in Y} p(x, y, z) = \sum_{y \in Y} p(y, z) p(x \mid y, z) \tag{2.2}$$

Lastly, we can use equation 2.1 to derive a version of Bayes' Theorem.

$$p(x \mid y, z) = \frac{p(x, y, z)}{p(y, z)} = \frac{p(y, x, z)}{p(x, z)} * \frac{p(x, z)}{p(y, z)} = \frac{p(y \mid x, z) p(x, z)}{p(y \mid z)} \tag{2.3}$$

In the future this will prove to be a useful tool to compute a posterior probability distribution $p(x \mid y)$ from the inverse conditional probability $p(y \mid x)$ and the prior probability distribution $p(x)$.

2.2 Bayes Filter

2.2.1 Scenario

Consider the general case of a robot which uses sensors to gather information about its environment. These sensors provide readings at discrete time steps $t = 0, 1, 2, ...$ Some amount of noise is associated with each of these readings. At each time step t, the robot may execute commands to affect its environment, and wishes to know its current state.

Let's encode the robot's current state at time t in the vector x_t. Similarly, z_t will represent a sensor measurement at time t, and u_t will represent the commands issued by the robot at time t. For each of these vectors we will use the notation $z_{1:t} = z_1, z_2, ..., z_t$.

The robot only has access to data in the form of z_t and u_t. Thus it cannot ever have perfect knowledge of its state x_t. It will have to make do by storing a probability distribution assigning a probability to every possible realization of x_t. This posterior probability distribution will represent the robot's belief in its current state, and should be conditioned on all available data. Thus we'll define the robot's belief distribution

to be:

$$bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t}) \tag{2.4}$$

2.2.2 Derivation

We can use equation 2.3 to rewrite $bel(x_t)$:

$$bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t}) = \frac{p(z_t \mid x_t, z_{1:t-1}, u_{1:t}) p(x_t \mid z_{1:t-1}, u_{1:t})}{p(z_t \mid z_{1:t-1}, u_{1:t})}$$

In order to simplify $p(z_t \mid x_t, z_{1:t-1}, u_{1:t})$, we'll have to make an important assumption. We'll assume that the state x_t satisfies the Markov property, that is, x_t perfectly encapsulates all prior information. Thus if x_t is known, then $z_{1:t}$ and $u_{1:t}$ are redundant. This assumption lets us remove consideration of past sensor measurements and commands, and to rewrite the belief distribution as:

$$bel(x_t) = \frac{p(z_t \mid x_t) p(x_t \mid z_{1:t-1}, u_{1:t})}{p(z_t \mid z_{1:t-1}, u_{1:t})}$$

Notice that $p(z_t \mid z_{1:t-1}, u_{1:t})$ is a constant with respect to x_t. Thus it makes sense to let $\eta = (p(z_t \mid z_{1:t-1}, u_{1:t}))^{-1}$ and rewrite the belief distribution as:

$$bel(x_t) = \eta p(z_t \mid x_t) p(x_t \mid z_{1:t-1}, u_{1:t})$$

Now we are left with two distributions of interest. Looking closely one may notice that $p(x_t \mid z_{1:t-1}, u_{1:t})$ is simply our original belief distribution, equation 2.4, but not conditioned on the most recent sensor measurement, z_t. Let us refer to this distribution as $\overline{bel}(x_t)$, and break it down further using equation 2.2 and our Markov

assumption:

$$\begin{aligned}
&\overline{bel}(x_t) = p(x_t \mid z_{1:t-1}, u_{1:t}) \\
&= \sum_{x_{t-1}} p(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t}) p(x_{t-1} \mid z_{1:t-1}, u_{1:t}) \\
&= \sum_{x_{t-1}} p(x_t \mid x_{t-1}, u_t) p(x_{t-1} \mid z_{1:t-1}, u_{1:t}) \\
&= \sum_{x_{t-1}} p(x_t \mid x_{t-1}, u_t) bel(x_{t-1})
\end{aligned}$$

We have arrived at a recursive definition of $bel(x_t)$ with respect to $bel(x_{t-1})$! As long as $p(x_t \mid x_{t-1}, u_t)$ and $p(z_t \mid x_t)$ are known, we can recursively calculate $bel(x_t)$.

$p(x_t \mid x_{t-1}, u_t)$ defines a stochastic model for the robot's state, defining how the robot's state will evolve over time based upon what commands it issues. This probability distribution is known as the *state transition probability*. []

$p(z_t \mid x_t)$ also defines a stochastic model, modeling the sensor measurements z_t as noisy projections of the robot's environment. This distribution will be referred to as the *measurement probability*. []

Once we have models for both the *state transition probability* and *measurement probability*, we can finally construct the algorithm known as Bayes' Filter:

Algorithm 1 Bayes Filter

function BAYESFILTERITERATE($bel(x_{t-1})$, u_t, z_t)
  for each possible state $x_t^* \in x_t$ **do**
    $\overline{bel}(x_t^*) = \sum_{x_{t-1}^* \in x_{t-1}} p(x_t^* \mid x_{t-1}^*, u_t) bel(x_{t-1}^*)$
    $bel(x_t^*) = \eta p(z_t \mid x_t^*) \overline{bel}(x_t^*)$
  end for
  Set $\sum_{x_t^* \in x_t} bel(x_t^*) = 1$, and solve for η
  Use η to compute $bel(x_t)$
  return $bel(x_t)$
end function

2.2.3 Example

2.3 Kalman Filter

2.3.1 Extended Kalman Filter

Chapter 3

Hardware

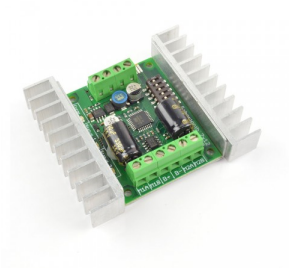
3.1 Specific Hardware Used

The mobile base used is the Lynxmotion A4WD1 Rover, shown on the right. This kit comes with four 200 rpm dc gear motors, and four optical quadrature motor encoders.

Figure 3-1: Lynxmotion 4WD Rover [**fig_lynxmotion_rover**]

The motors are controlled by a Sabertooth dual-channel 12A 6V-24V regenerative motor driver, shown on the left. This motor driver is powered by two LG 18650 HE2 rechargable lithium ion cells, which sit in an 18650 battery case which has been soldered to act as a battery pack with two 18650 cells in series. The battery cells are individually charged before use with a NiteCore-i2-V2014 li-ion charger.

Figure 3-2: Sabertooth 2x12 [**fig_sabertooth**]

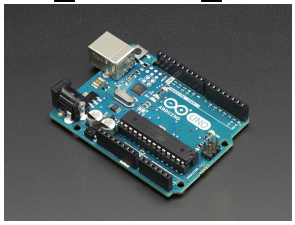


On top of the rover is the PING))) ultrasonic distance sensor, which is attached to a standard Parallax servo which pans back and forth 180 degrees.

Figure 3-3: PING))) Ultrasonic Sensor [**fig_ping**]

At the center of the rover is an Arduino Uno R3. This microcontroller handles several important tasks. It tells the motor driver what speed to set its two output channels to, and directly controls the panning motion of the standard Parallax servo. It also acts as a go-between for the digital output of the sensors on the rover and a laptop. It's connected to this laptop via a USB cable, which powers the board and allows communication over a serial port. Motor encoder values and ultrasonic range data are transmitted to the laptop, and motor power commands are received.

Figure 3-4: Arduino Uno R3 [**fig_arduino_uno**]



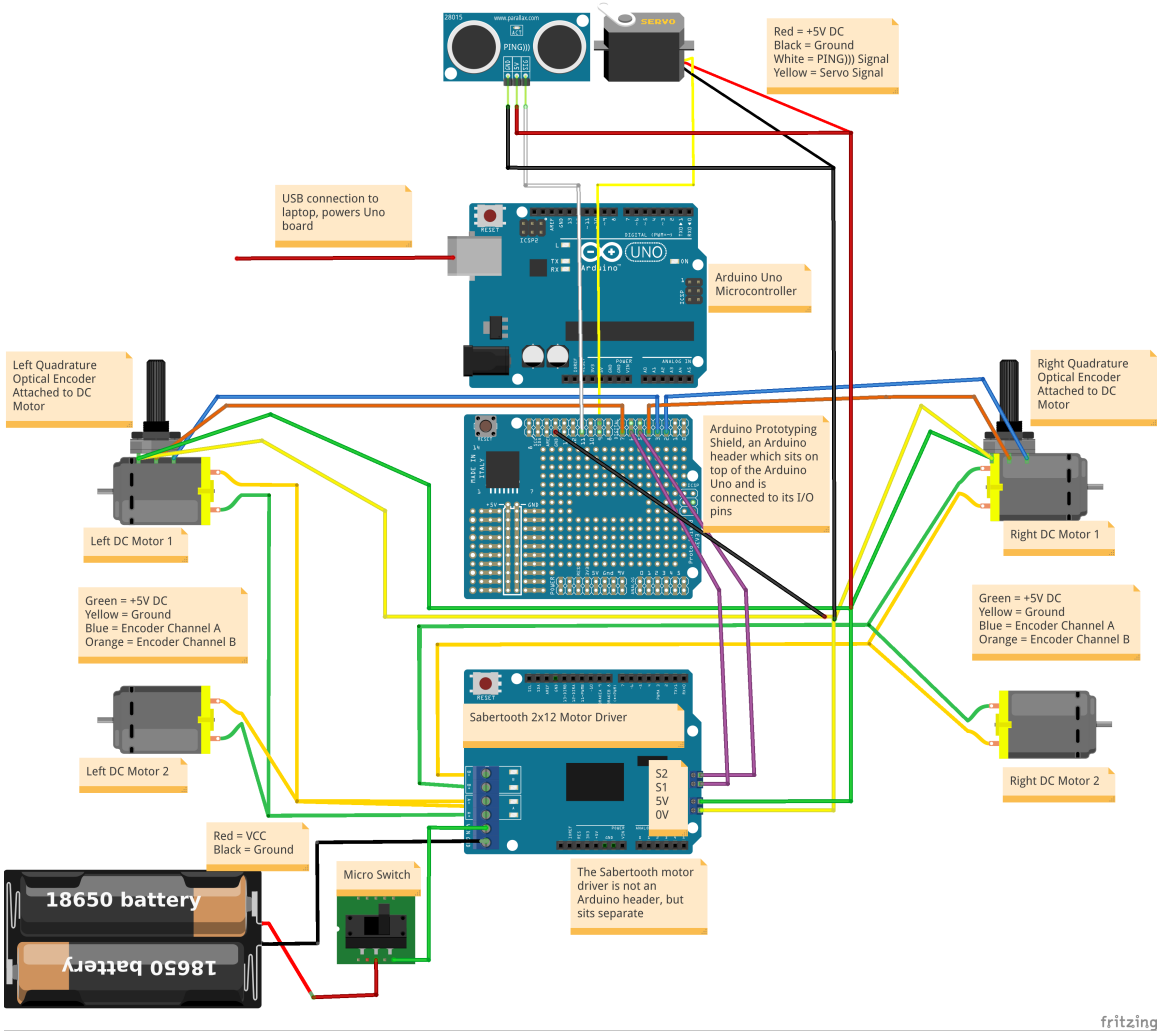
The specific laptop used in this project is the Dell Inspiron 3531, which has a quad core 2.16 GHz processor, and 4 GB of RAM. While these are relatively limited computational resources, the laptop was a personal work machine and already available to use for no additional cost. The machine is used as the main processing unit for the navigation logic.

The last component is a Nexus 4 smartphone placed on the top panel of the rover, which is also connected to the laptop by USB. Inside this phone is an MPU-6050 chip which contains a gyroscope and accelerometer. Elsewhere on the phone's logic board are

a magnetometer, otherwise known as a digital compass, and a GPS receiver. This smartphone was also already available, and acts as a cheap Inertial Measurement Unit (IMU) and GPS receiver for the robot.

3.2 Construction

Figure 3-5: Connections Made



This image was created with Fritzing.

3.2.1 Arduino Pin Connections

Most digital pin numbers used are arbitrary, and connections may be permuted without any change. The sole exception for Special care is taken in the

3.3 Arduino Sketch

A sketch is Arduino-speak for a program uploaded to the board which will run on a loop as long as the board is powered.

Chapter 4

ROS

4.1 ROS

The Robot Operating System (ROS)

4.1.1 Nodes

4.1.2 Frames

4.1.3 Topics

publishing and subscribing messages

4.2 Overview

4.3 Ros Sensors App

Android app to publish IMU and GPS data from a smartphone.

4.3.1 How to use

4.4 robot_localization package

odometry estimate from wheel encoders

[**robot_localization_paper**]

Chapter 5

ROS Navigation

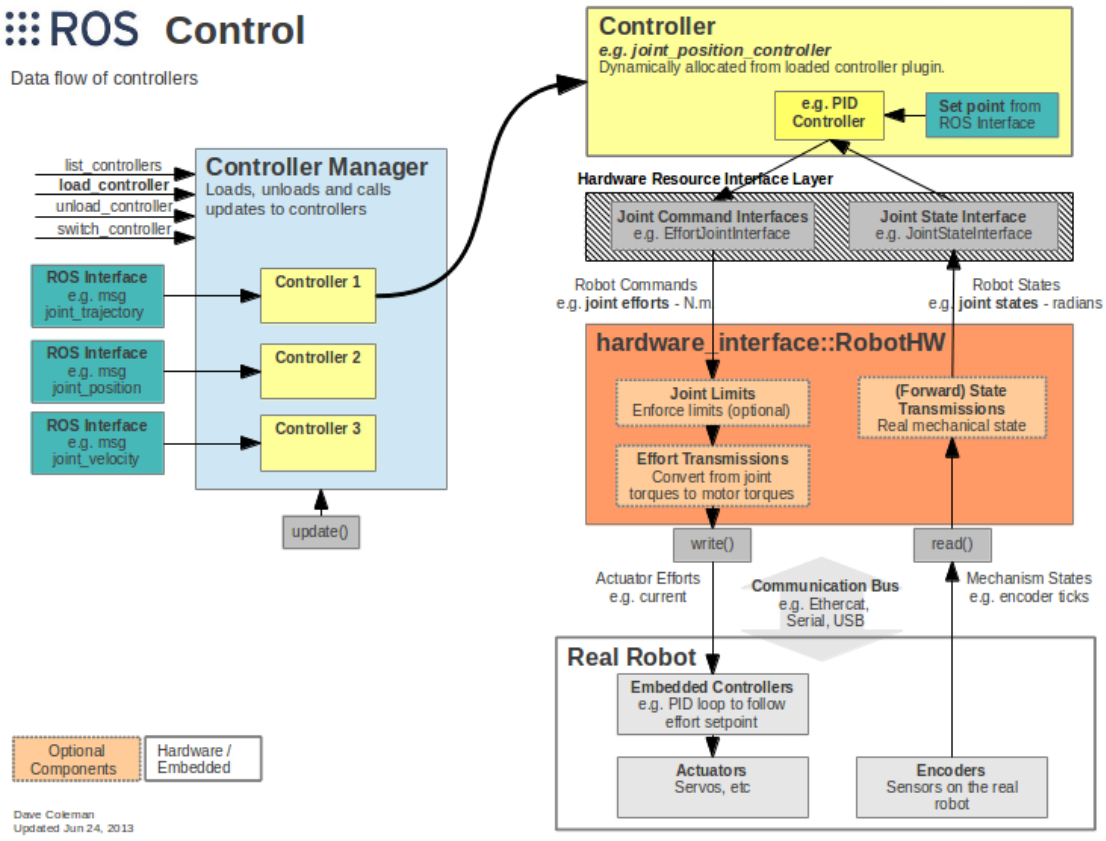
5.1 Navigation stack

5.2 ros_controller

Skid steering

5.2.1 Arduino Repeater Node

Figure 5-1: ROS control overview



Chapter 6

6.1 Field Test scenario

6.1.1 GPS Waypoints

GPS markers form a linearized path travel in a straight line from one GPS node to another

6.2 Results

6.3 Future Steps

how this project is limited

6.4 Conclusion