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 de-

fined 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 redun-

dant. 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

1: **function** BAYESFILTERITERATE($bel(x_{t-1})$, u_t, z_t)
2: **for** each possible state $x_t^* \in x_t$ **do**
3: $\overline{bel}(x_t^*) = \sum_{x_{t-1}^* \in x_{t-1}} p(x_t^* \mid x_{t-1}^*, u_t) bel(x_{t-1}^*)$
4: $bel(x_t^*) = \eta p(z_t \mid x_t^*) \overline{bel}(x_t^*)$
5: **end for**
6: Set $\sum_{x_t^* \in x_t} bel(x_t^*) = 1$, and solve for η
7: Use η to compute $bel(x_t)$
8: **return** $bel(x_t)$
9: **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 specific hardware used in this project was chosen to minimize cost while still producing a vehicle capable of navigating rough, uneven outdoors terrain. Parts that were already on hand, and that most college students would reasonably have access to, such as a personal laptop and an Android smartphone, were used over superior alternatives. In total these parts were purchased for less than $500.

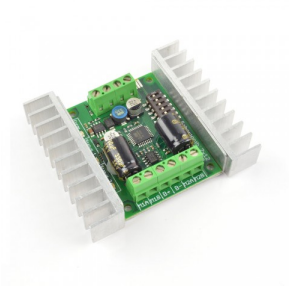
The mobile base used is the Lynxmotion A4WD1 Rover, see Figure 3-1. This kit comes with four 200 rpm DC gear motors, and four motor encoders. The chassis consists of four aluminum side brackets, and two polycarbonate panels on the top and bottom.

Figure 3-1: Lynxmotion 4WD Rover [5]

The motors are controlled by a Sabertooth dual-channel

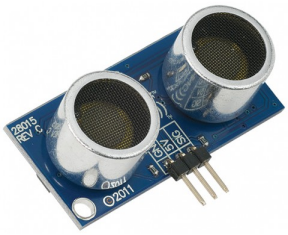
12A 6V-24V regenerative motor driver, see Figure 3-2. This motor driver is powered by two LG 18650 HE2 rechargable lithium ion cells, which sit in an 18650 battery case which has been soldered to act as a battery pack with two 18650 cells in series. The battery cells are individually charged before use with a NiteCore-i2-V2014 li-ion charger.

Figure 3-2: Saber-tooth 2x12 [2]



On top of the rover is the PING))) ultrasonic distance sensor (see Figure 3-3), which is attached to a standard Parallax servo which pans back and forth 180 degrees.

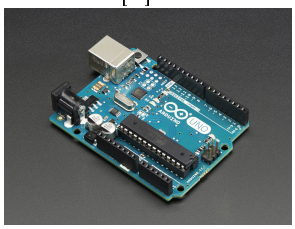
Figure 3-3: PING))) Ultrasonic Sensor [7]



At the center of the rover is an Arduino Uno R3, see Figure 3-4. 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. An Arduino prototyping shield is stacked on top to allow re-usability of the board.

Figure 3-4: Arduino Uno R3 [4]



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. Any personal laptop running Ubuntu or Debian could be used here, and additional computational resources would be beneficial. However, this laptop was a personal work machine and already available to use at no additional cost. The laptop 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 was also a personal device already available, and acts as a cheap Inertial Measurement Unit (IMU) and GPS receiver for the robot.

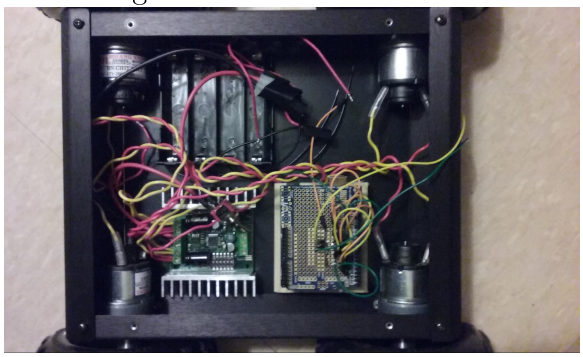
3.2 Construction

Figure 3-5 shows the base just after assembly. The aluminum side brackets' mounting holes did not line up properly with the motors, so a Dremel drill was used to widen them.

Figure 3-5: Constructed Chassis

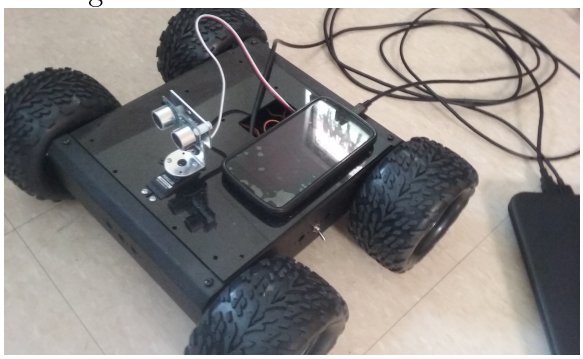
The Arduino Uno was screwed to a 2.5" x 3" x 0.5" wooden poplar block, with non-conductive nylon washers placed between the screw head and the Uno, and between the Uno and the wooden block. The wooden Arduino mounting board and the Sabertooth were both attached via double-sided foam mounting tape to the bottom panel of the rover. The battery holder was attached with glue dots to make removal easier.

Figure 3-6: Pieces Mounted



The servo fits conveniently into a precut opening in the top chassis panel, and is held in place with four 3mm x 6mm screws and corresponding washers. A mounting bracket is attached to the servo, and the PING))) sensor is screwed to that mounting bracket, using non-conductive washers and screws to separate the circuit board and the metal mounting bracket.

Figure 3-7: Construction Finished



Another opening in the top panel allows the PING))) sensor to connect to the Arduino inside the body of the rover. This opening also allows the type A/B USB cable connected to the Arduino to extend out and reach the laptop. The smartphone sits on the top panel just to the left of this opening, secured in place by removable glue adhesive dots. It is also connected to the laptop via a micro-USB to USB cable.

Both USB cables are long, stretching to just under 10 feet. The USB 2.0 specification limits the length of cable between two 2.0 USB devices to less than five meters, or about 16 feet [8]. Thus there should be no problem with the current length, but extensions in the future could not go much further.

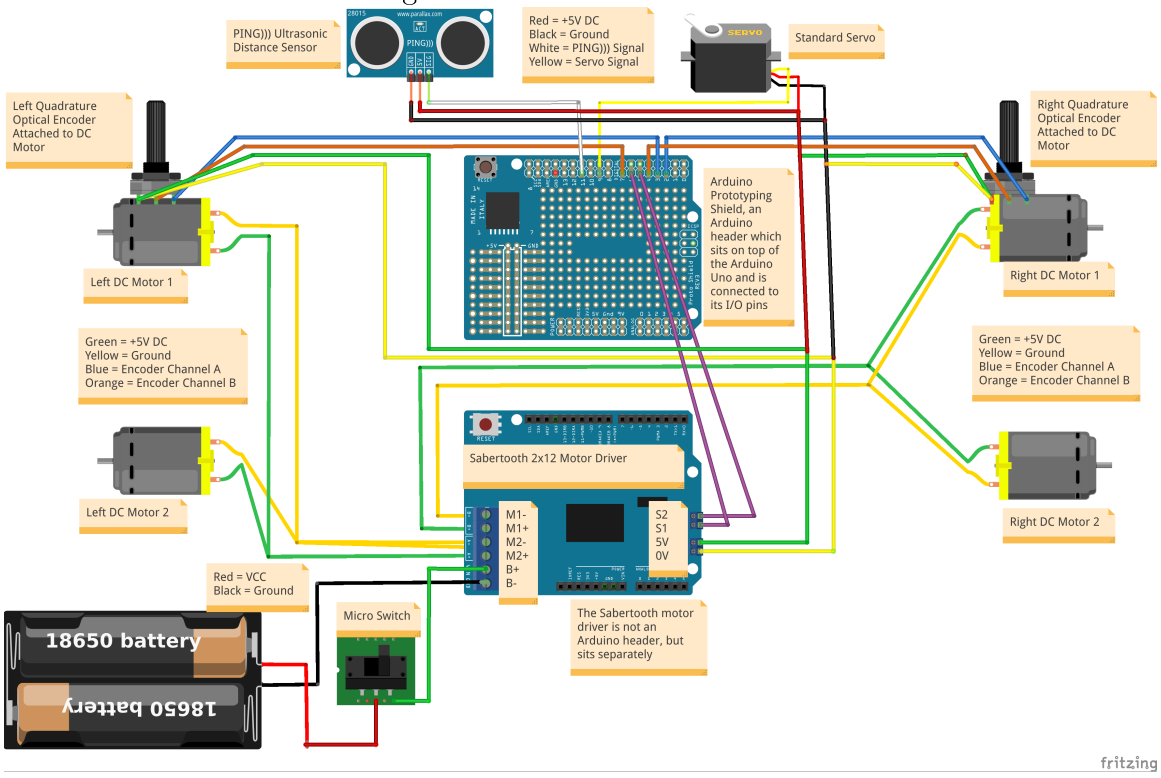
Connecting electronics on the rover to the laptop via USB means the processing laptop must manually be kept within 10 feet of the rover as it navigates. This design could easily be extended to include wireless or radio communication with a server, and a larger chassis would simply be able to carry the laptop on it. However, USB cables are cheap and still function as a proof of concept for an autonomous design.

Figure 3-8 is a schematic specifying the overall design for the rover.

Connections were made with flexible stranded core, 22 AWG breadboard wires. Connections to the Arduino's digital pins were made indirectly. A prototyping shield was stacked on top of the Arduino, and connected to its digital pins via pin headers. Signal pins were then soldered to the prototyping shield. Breadboard wires needing direct connection would ideally use terminal block connectors. However, these were difficult to find at a reasonable price, and so wires were soldered together tip to tip, and then wrapped in electrical tape.

The Sabertooth motor driver controls two motor channels. It drives DC motors

Figure 3-8: Connections Made



This image was created with Fritzing

from these channels in a relatively simple way. The speed of DC motors is proportional to the voltage supplied to them, and the direction of rotation can be flipped simply by flipping the polarity of the supplied voltage. The motor driver manages the effective voltage supplied to each channel using pulse-width modulation (PWM), and handles polarity using an on-board H-bridge. [3]

3.2.1 Power

Besides the Arduino Uno, which is powered separately by a USB connection to the laptop, most of the rover's components are powered by the battery pack. This pack contains two individual 18650 lithium-ion cells placed into a battery holder and connected in series. The battery pack is then connected to the motor driver's battery terminals B+ and B-. The positive B+ output goes through a microswitch, which is attached to a side bracket in the rover, and is accessible from the outside. This acts as a kill switch for the battery pack.

Each 18650 cell holds 4.2V at full charge, and discharges down to a minimum of 2.7V. The motor driver has a lithium cutoff mode which shuts the driver down when the average voltage of cells in the battery pack reaches 3.0V. Thus the voltage supplied to the four motors through the motor driver's output channels will range from 8.4V to 6.0V, which is within their acceptable operating range.

The specific li-ion cells being used can supply up to 20A continuously, and the motor driver can handle up to 12A per channel. The motors each draw a maximum current of 1.5A, and two are used per channel, putting the total possible current draw

of 3A per channel well below the limits of the motor driver and battery pack.

The motor driver has an onboard battery eliminator circuit (BEC) which is an efficient 5V voltage regulator capable of supplying up to 1 amp of continuous current, with 1.5 Amps at peak. The ultrasonic sensor, its servo, and the two rotary encoders combined use less than 500 mA, and are all powered through this BEC. The Arduino is also connected to this BEC's ground, as the Uno and Sabertooth must share a common ground plane in order for the control signals to be read correctly. [1]

The standard servo has the potential to draw peak currents of up to 1A, if it hits a snag and is stopped from moving. Therefore the input wires to the 5V and 0V BEC terminal connectors should be capable of handling those peaks. Since we are using 22 AWG wires, we are close to the limit, but a 22 AWG wire with 43 or more internal cores is rated to handle 1A. And the expected consistent draw is much lower, less than 500mA.

Note that the sensors attached to the microcontroller should be powered off before the Arduino, else the Arduino may try to power the whole Mega chip via its input pins. The sensors are powered from the BEC on the motor driver, so they may be turned off by using the microswitch between the battery holder and the motor driver.

3.3 Arduino

TODO: Explain what digital pins are ==========

As one can see in Figure 3-8, only two optical quadrature encoders are used, with one placed on the front motor for both the left and right side of the rover. This is

due to a hardware limitation of the Arduino Uno. The Uno only has two hardware interrupt pins, digital pins 2 and 3. A hardware interrupt is necessary to keep up with the fast rate of pin voltage changes that rotary encoders output. Because there are only two hardware interrupt pins, only two of the four quadrature motor encoders are used. The two encoders chosen in this design are attached to the front motors of the rover.

If a different Arduino board such as the Mega were used, there would be sufficient hardware interrupt pins for all four encoders. Using a board with plentiful interrupts, one could even attach both channel outputs of the encoders to interrupt pins, rather than only one. This would double the encoders' resolution **??**.

3.3.1 Arduino Pin Connections

Each encoder has two output channels, channel A and channel B. Both encoders attach one of their output channels, channel A, to a hardware interrupt pin. The right motor's encoder connects channel A to pin 2, and channel B to pin 4. The left motor's encoder connects its channel A output to pin 3, and its channel B output to pin 7.

The S1 and S2 signal input terminals on the Sabertooth motor driver are connected to digital pins 5 and 6. The signal input for the hobby servo is connected to digital pin 9. The signal input for the ultrasonic sensor is connected to digital pin 11. The Arduino's ground pin is connected to the motor driver's BEC's ground, to ensure a common ground plane.

Most digital pin numbers used are arbitrary, and connections may be permuted without any change. The exceptions are pins 0-3, which must be left unchanged. Pins 0 and 1 must be left unattached for serial data transfer to work properly over USB. And pins 2 and 3 are hardware interrupt pins which must be used to handle the quadrature encoders' output.

3.3.2 Motor Driver Configuration

The Sabertooth has two signal input terminals, S1 and S2, which allow the Arduino to issue instructions for the motors. Configurations and settings of the driver are specified by flipping six on-board DIP switches between their down and up positions. Setting switch 1 down and switch 2 up places the driver into R/C input mode, which configures S1 and S2 to expect pulse-position modulation (PPM) signals, Ãă la R/C controllers. The Arduino controls servos via the Servo library using PPM signals. This allows the Arduino code to treat S1 and S2 as ordinary servos. [1]

Turning switch 3 down selects the lithium cutoff mode, which detects the number of lithium cells in series powering the driver, and shuts off when the battery pack's voltage drops below 3.0V per cell, or 6.0V in the current configuration of two cells. This is used to prevent accidental damage to the 18650 cells by over-discharging them in the field.

Flipping switch 4 down selects independent (differential) drive, which allows S1 and S2 to each independently control the speed of one of the motor channels. Thus turning of the vehicle is achieved by lowering the relative speed of the motors on the

side which the rover is turning to.

Switch 5 is kept up to ensure a linear rather than exponential response of the motors to the microcontroller's input signal. Switch 6 is set down to select the microcontroller mode, which turns off auto-calibration of the zero-movement point, and turns off an automatic timeout. Thus if the signal connection is somehow lost the motor driver will continue driving the motors according to the last signal received. This is necessary for smooth performance of the motors since the Arduino may slightly delay control pulses due to hardware interrupts. Though this introduces a risk of loss of control, it is a small one as the only way for the connection between the microcontroller and motor driver to be lost would be for the signal breadboard wires to come undone during use.

3.3.3 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. Though this is not hardware, the sketch interacts with the components on a low level, and so it seems appropriate to cover here.

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

[6]

Chapter 5

ROS Navigation

5.1 Navigation stack

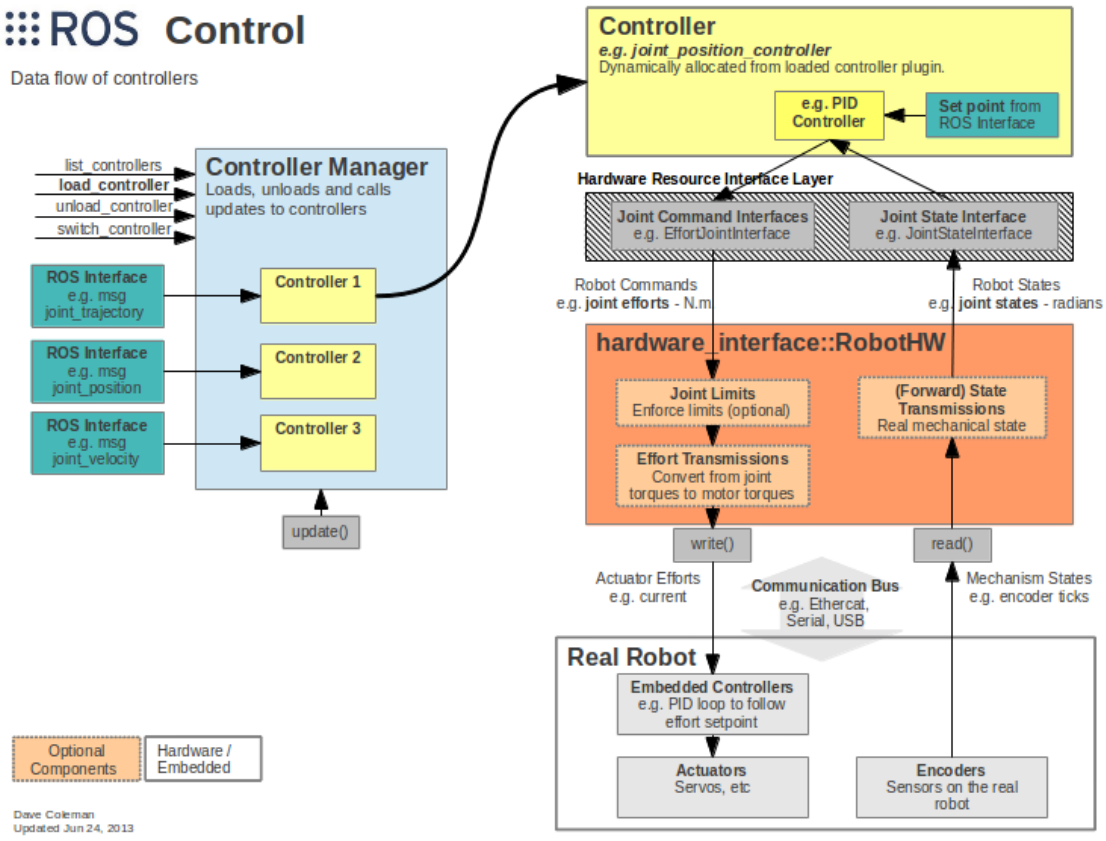
5.2 ros_controller

Skid steering

5.2.1 Arduino Repeater Node

Adjust code for conversion of ultrasonic sensor ping times to distance, to take into account the ambient air temperature

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

Bibliography

[1] Dimension Engineering. *Sabertooth 2x12 User's Guide*. English. 21 pp.

[2] Dimension Engineering. *Sabertooth Dual 12A 6V-24V Regenerative Motor Driver*. [Online: accessed April 11, 2017]. URL: https://web.archive.org/web/20160402162832/http://www.robotshop.com/en/sabertooth-dual-regenerative-motor-driver.html.

[3] Tom Igoe. *Controlling DC Motors*. http://web.archive.org/web/20160915053759/http://www.tigoe dc-motors/. Blog. 2017.

[4] Adafruit Industries. *Arduino Uno R3 (Atmega328 - assembled)*. [Online: accessed April 13, 2017]. 2017. URL: http://web.archive.org/web/20170321022401/https://www.adafruit.com/product/50.

[5] Lynxmotion. *Lynxmotion Aluminum A4WD1 Rover Kit (w/ Encoders)*. [Online: accessed April 11, 2017]. URL: https://web.archive.org/web/20151208053255/http://www.robotshop.com/en/lynxmotion-aluminum-a4wd1-rover-kit-w-encoders.html.

[6] T. Moore and D. Stouch. "A Generalized Extended Kalman Filter Implementation for the Robot Operating System". In: *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

[7] Parallax. *Parallax PING Ultrasonic Sensor*. [Online: accessed April 13, 2017]. 2017. URL: `http://web.archive.org/web/20161208012823/http://www.robotshop.com/en/parallax-ping-ultrasonic-sensor.html`.

[8] *USB 2.0 Frequently Asked Questions*. http://web.archive.org/web/20170225172623/http://www.us Online. 2017.