RoboSim Sample Robot Plant Model Calculations

Chris Gerth - FRC 1736 Robot Casserole

Fall 2015

Abstract

An analysis of a simple tank-drive robot is presented. The simple plant model is designed to be a starting point for year-to-year development of more detailed robot models. The device modeled has two sets of motors to drive motion in a tank-drive fashion, along with a normal battery and electrical system.

Contents

I Background Academics Primer **4**

1 Calculus **4**
1.1 Functions . . . 4
1.2 Continuous Functions, Derivatives and Integrals . . . 4
1.3 Discrete Functions, Derivatives and Integrals . . . 6

2 Trigonometry **7**

3 Physics **7**
3.1 Position, Linear Motion, and Calculus . . . 7
3.2 Forces and Linear motion . . . 9
3.2.1 Newton's Law . . . 9
3.2.2 Sources of Forces . . . 9
3.2.3 Frictional Forces . . . 9
3.2.4 Static Friction . . . 9
3.2.5 Kinetic Friction . . . 10
3.3 Torque and Rotation . . . 10
3.3.1 Rotational Motion Parameters . . . 11
3.3.2 Torque and Rotational motion . . . 11
3.4 Ideal Gas Law . . . 12
3.5 Electrical Laws . . . 12
3.5.1 Electrical Introduction . . . 12
3.5.2 Basic Electrical Quantities and Relationships . . . 13

II Component Models **14**

4 Electric Motor **14**
4.1 Electrical Model . . . 14
4.2 Joining of Electrical and Mechanical Models . . . 15
4.3 Integration with Larger Systems . . . 16

5 Main Battery **17**
5.1 Modeling Voltage Drop due to Instantaneous Current Draw . . . 17
5.2 Modeling Total Remaining Charge based on Aggregate Current Draw . . . 18

6 Main Circuit Breaker **18**

7 Compressor and Tank **18**
7.1 "The System" . . . 18
7.2 Compressor . . . 18

8 Traction Wheel **19**

III Integrated Model **20**

9 Electrical Subsystem **20**

10 Drivetrain Subsystem **20**
10.1 Linear Force from Drivetrain Motors . . . 21
10.2 Net Rotational Torque from Drivetrain Motors . . . 21
10.3 Wheel friction . . . 21
10.4 Induced Motion . . . 22
10.5 Motor Speed Calculation . . . 22

11 Pneumatic subsystem **23**

12 Field Collision Model **23**

IV Design Notes 24

13 Software Architecture **24**
13.1 sim_main . . . 24
13.2 sim_init . . . 24
13.3 robot_init_15 . . . 24
13.4 field_init_15 . . . 25
13.5 in_proc . . . 25
13.6 robot_15 . . . 25
13.7 field_robot_physics . . . 25
13.8 out_proc . . . 25
13.9 post_proc . . . 26

14 RoboSim Hardware **26**

Part I
Background Academics Primer

In this section, we will seek to demonstrate the "basic" math and physics required to understand RoboSim. This content is effectively a whirlwind summary of the important parts of AP Calc BC, AP Physics C, basic Chemistry, and any introductory Electronics course.

1 Calculus

Sir Isaac Newton was a smart man. He came up with calculus. Either him or Leibniz. In any event. **Calculus** is a particular field of study within mathematics. It studies functions and properties of functions.

1.1 Functions

A **function** can be thought of as a box which takes one input, and produces some outputs based on that input. For RoboSim, we will deal with many functions of time. This is to say that we will look at how **properties** change over time. The "Box" in this case is the robot itself. The most fundamental input to the robot is time. As time progresses forward, different properties of the robot change. These properties include things like location, speed, torque, rotation, electrical current draw, etc. In a mathematical sense, time is the input, and the various properties are the output. At every given time, each property has a value. Additionally, the physics of the configuration of the parts of the robot determine relationships between the quantities as time progresses forward. For example, a large voltage applied to motors will induce larger torques and speeds and current draws.

1.2 Continuous Functions, Derivatives and Integrals

Continuous calculus is what is usually studied in High School. It is assumed that time is linear and continuous. Valid times are t = 0s, t = 1.2429543987s, t = 3.25s, etc. Any value of time is a defined. An infinitely detailed timeline can be established.

Calculus looks at two primary things about functions: Derivatives and Integrals.

A **Derivative** of a function describes how quickly the function is changing as time goes on. On a curve on a Cartesian (x/y) plot, the derivative is the same as the slope of the curve at any given point, as shown in Figure 1.

$$x = y(t)$$

$$x = \frac{dy}{dt}$$

An **Integral** of a function describes how big or small the function's value has been historically. On a Cartesian plot, the integral is the same as the area between the curve and the x axis over a given period of time, as shown in Figure 2.

$$x = y(t)$$

$$X = \int y(t)dt + c$$

For the purposes of RoboSim, there is no need to discuss the calculation of integrals or derivatives in the continuous domain.

If a function is getting larger quickly, its derivative will be big at that time. If it is getting smaller in a hurry, its derivative will be negative

If a function has been big for a long time, its integral will be large. If it's been bouncing around but equally above and below zero, the integral will be very small.

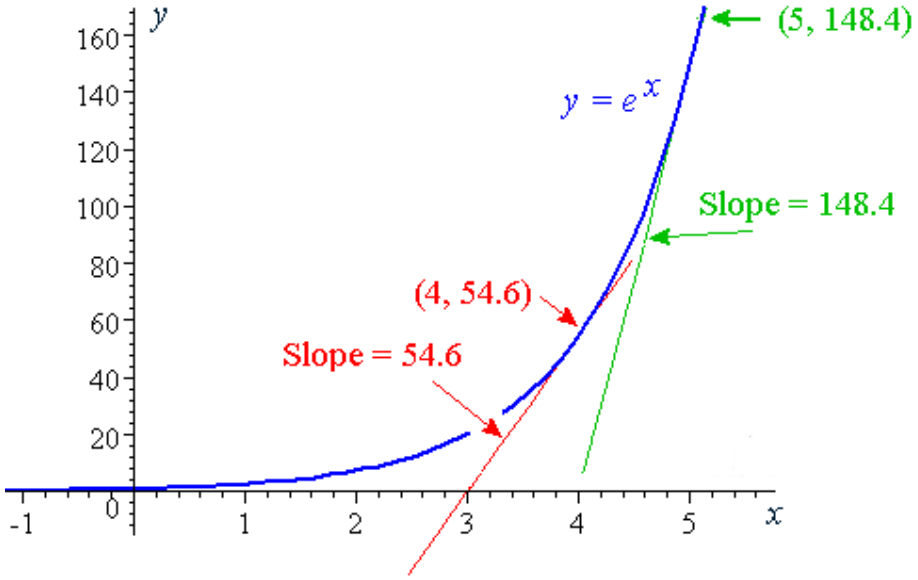

Figure 1: Illustration of a derivative as the "instantaneous slope" of a curve

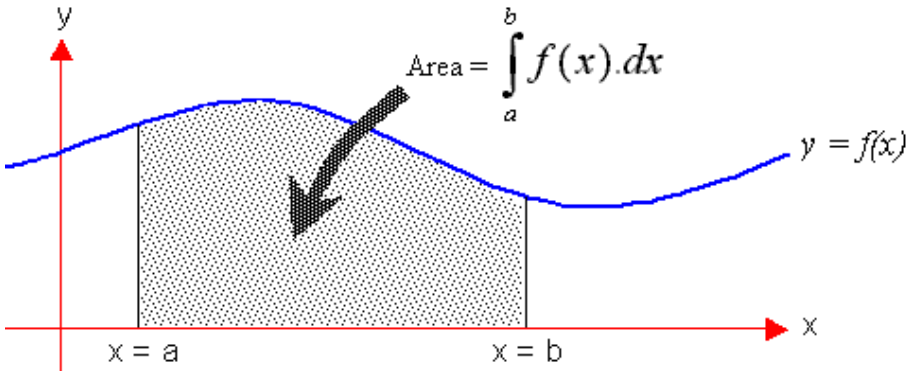

Figure 2: Illustration of a integral as the "area under a curve"

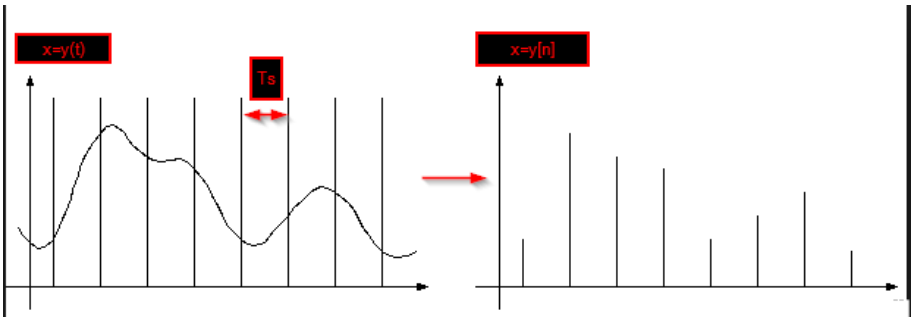

Figure 3: Sampling a continuous function x=y(t) to produce discrete function x=y[n]

1.3 Discrete Functions, Derivatives and Integrals

Discrete functions are very similar to normal functions, except they are only defined at certain, regularly-repeating time values. In this way, a discrete function could be though of as a set of $time, value$ pairs.

$$\{0.00, 3\}, \{0.01, 2\}, \{0.02, 1\}, \{0.03, 0\}, ...$$

We might turn a continuous function into a discrete one by **sampling** it at regular intervals. This is usually the way that sensors and digital systems are integrated - the sensor reads an analog value continuously, and the digital system records the sensor's value at specific points in time.

We usually notate a discrete time function as such

$$x = y[n]$$

For a function y that produces a value x at time index n. For all discrete functions, we must always specify a "**Sample Time**" (notated Ts) to indicate the spacing between samples. For RoboSim, Ts is hard-coded to 50ms.

Note the square brackets which indicate discrete time. Here n is an index, not a measure in seconds. Multiplying the n index by the sample time Ts yields the actual (real-world) time in seconds.

In Discrete time calculus, we define derivatives as follows:

$$x = y[n]$$

$$\frac{dx}{dt} \cong \frac{y[n] - y[n-1]}{Ts}$$

This formula says "A function's derivative is equal to the current value minus the previous value, scaled by the sample time." You may recognize this form from part of the "Limit" definition of a derivative. Note here that if we force Ts to zero, we have a continuous derivative. However, because computer systems cannot sample infinitely fast, we accept a small but finite Ts as a very close approximation. The conditions under which this approximation is acceptable is the subject of Nyquist Frequency theory, but is a topic out of scope for this discussion.

In Discrete time calculus, we define Integrals as follows:

$$x = y[n]$$

$$\int x dt \cong \sum_{i=0}^{n} y[i] * Ts$$

This equation says "A function's integral is equal to the sum of all previous values, scaled by the sample time".You may recognize this from the Riemann sum approximation of an integral. If we force Ts to zero, we end up with a continuous integral. But again, since computer systems cannot sample infinitely fast, we accept a small but finite Ts and move on with our lives.

2 Trigonometry

You are likely already aware of most trig concepts. $sin(), cos(), tan()$are among some of the functions you should be familiar with. For the purposes of calculations of robosim, we will do all angle calculations in radians. Additionally, we will use to $atan2()$ function, which is very similar to the inverse tangent ($atan()$). However, it has some extra definitions to give it a full output range of $[0, 2\pi]$for any given X/Y inputs. You can look up more details online, but the formal definition is as follows:

$$atan2(X,Y) = \begin{cases} atan\left(\frac{-Y}{X}\right) & X > 0 \\ 180^\circ + atan\left(\frac{Y}{X}\right) & Y \geq 0, X < 0 \\ -180^\circ + atan\left(\frac{Y}{X}\right) & Y < 0, X < 0 \\ 90^\circ & Y > 0, X = 0 \\ -90^\circ & Y < 0, X = 0 \\ undefined & Y = 0, X = 0 \end{cases}$$

3 Physics

Physics is also a thing that Newton did. Told you he was smart. In this section, we will begin to analyze the constraints imposed by mother nature between different quantities on our robot.

3.1 Position, Linear Motion, and Calculus

For physics to work, we must have objects in places. The place the object is at is called its **Position**. Let us define the following function which describes an object's location along one axis at any time. Specifically, it says how far the object is from an arbitrary zero point at any time:

$$Distance = x[n]$$

Velocity is how fast an object is moving. Velocity can be positive or negative. Positive velocity means the distance gets bigger, and negative velocity means the distance gets smaller. Because of physics, velocity is therefore the derivative of distance.

$$Velocity = v[n]$$

$$v[n] = \frac{dx}{dt} = \frac{x[n] - x[n-1]}{Ts}$$

Acceleration is how fast an object is changing in velocity. Acceleration can be positive or negative. A positive acceleration means the object is going faster and faster and faster. A negative acceleration means the object is slowing down. Because physics is still physics, acceleration is the derivative of velocity.

$$Acceleration = a[n]$$

$$a[n] = \frac{dv}{dt} = \frac{v[n] - v[n-1]}{Ts}$$

Note that integration allows you to go the other way. That is to say, velocity is the integral of acceleration, and distance is the integral of velocity.

$$v[n] = \sum_{i=0}^{n} a[i] * Ts$$

$$x[n] = \sum_{i=0}^{n} v[i] * Ts$$

Plots of these relationships are shown in Figure 4.

For units, we will always use **metric**. Therefore, distance is in meters (m), velocity is in meters per second (m/s), and acceleration is in meters per second, per second (m/s^2)

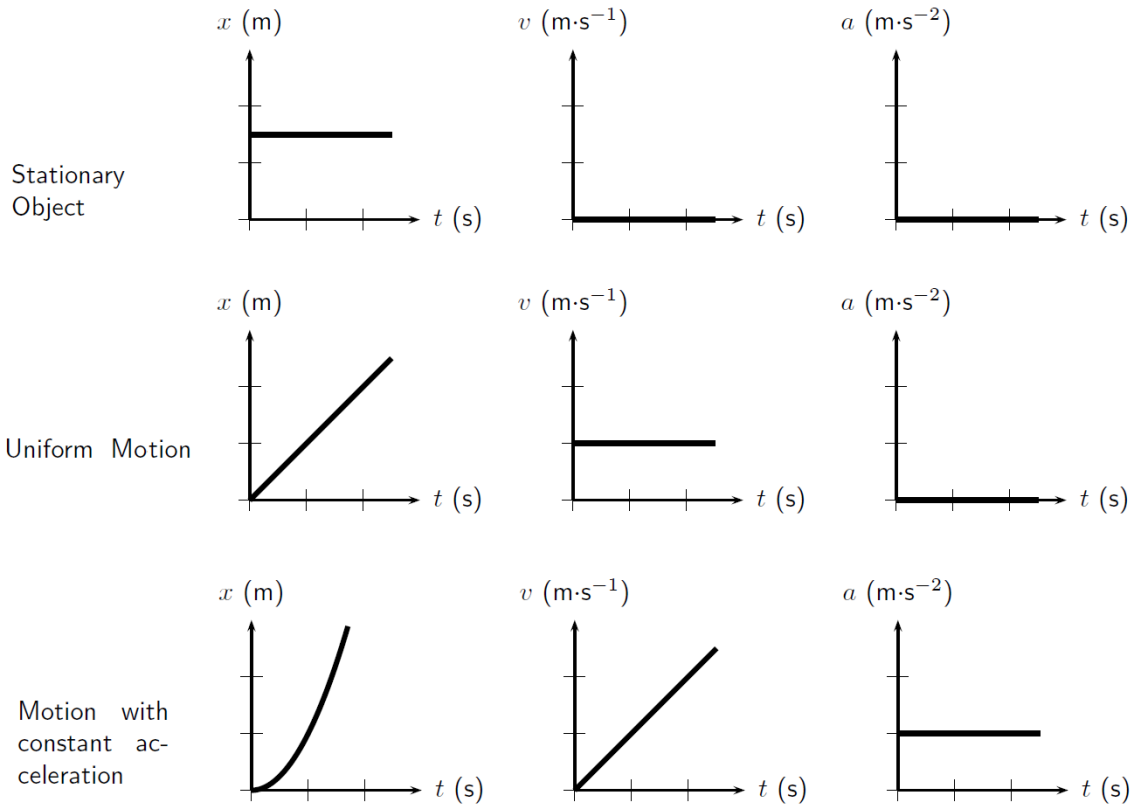

Figure 4: Examples of motion, relating acceleration, velocity, and position

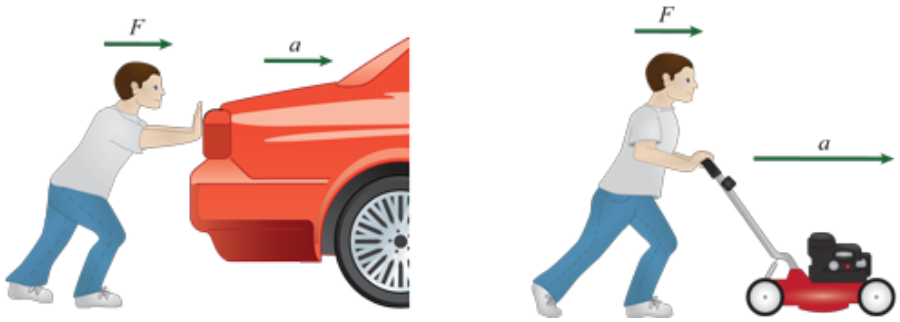

Figure 5: Newton's Second Law

3.2 Forces and Linear motion

3.2.1 Newton's Law

Force is something pushing on another thing. The more you push, the faster and faster you will go. The larger the object, the harder it is to make it go faster.

Newton's laws of motion allow us to define the following: At any given time, Acceleration is proportional to Force, scaled by an objects mass:

$$Force = mass * acceleration$$

$$F[n] = m * a[n]$$

Note that Force is measured in Newtons (N), mass is measured in kilograms (kg), and acceleration is in meters per second per second (m/sˆ2).

Note also that force changes over time, and acceleration changes over time, but we assume mass is constant.

3.2.2 Sources of Forces

Forces come from many places. A student pushing on a robot frame is one possible, simple source. A turning wheel in contact with the floor is another. Meshed gear teeth pushing on one another is yet another transfer of force.

Of the list mentioned so far, friction is very important and has some special calculations associated with it.

3.2.3 Frictional Forces

Friction is a force which opposes two contacting surfaces from sliding relative to each other. It comes from microscopic interactions of matter in contact with each other. It can get very complex and is not fully understood, but can be well modeled at a macro level. We will discuss this model, as it is the most useful for FRC robotics applications. There are two main types of friction: Static and Kinetic.

3.2.4 Static Friction

Static friction deals with the case where two surfaces are not moving relative to one another. If you were to pin a book in place on a wall by only pressing on the book toward the wall, you would be using the static friction between the book and the wall to oppose the force of gravity, keeping the book from falling. This is illustrated in Figure 6.

Static frictional force will take on whatever value it needs to to counter out other forces, up to a certain point. This certain point is called the **maximum static frictional force**. It is calculated with the following formula:

$$F_{Smax}[n] = \mu_S * F_N[n]$$

Here, F_{Smax} is the maximum static frictional force. It always points in the opposite direction as any force acting along the same axis. F_N is the "normal" force, or the force pinning the two surfaces together. μ_S is the Coefficient of Static Friction, which is derived from how effective the two surfaces are at sticking to each-other. Two smooth,

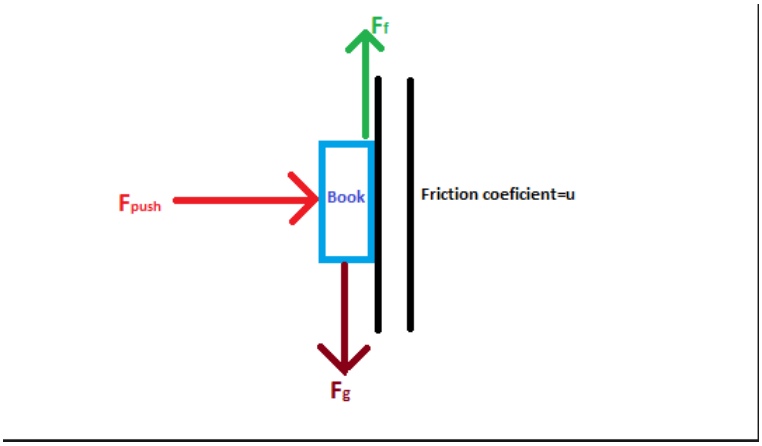

Figure 6: Static friction pinning a book to a wall

slick surfaces (like two pieces of smooth hard plastic) will have a small μ_S, while two rough surfaces (like concrete) will have a large μ_S. Usually, μ_S is just looked up in a table.

Static friction is often informally called "**Stiction**" in the engineering world, since it causes things to stick together. Also, "Stiction" is a portmanteau of "Static Friction". Engineers are a clever bunch.

3.2.5 Kinetic Friction

Kinetic friction deals with the case where two surfaces are sliding relative to each other. If you were to slide a book along the ground, the force that opposes your efforts to push the book is the kinetic frictional force of the book in contact with the ground. It is calculated by this formula:

$$F_K[n] = \mu_K * F_N[n]$$

Again, $F_K$is the kinetic frictional force. It always points in the opposite direction of motion of the interface (opposes the motion occurring). F_N is still the "normal" force, or the force pinning the two surfaces together. μ_K is the coefficient of kinetic friction, which again determines how effective the materials can interface with each other to produce an opposing force.

For most materials, μ_K is slightly than μ_S.

Note that as a wheel is rotating with the ground, there is no relative motion between the ground and the point of contact with the wheel. Indeed, the motion of the robot is due to the rotation of the wheel, and not due to the wheel slipping over the ground. A non-slipping robot's wheel-ground interface lies in the Static Friction domain. When wheel-ground interfaces start to enter the Kinetic friction domain, the effect is what many young people refer to as "drifting".

3.3 Torque and Rotation

Because symmetry is a thing of life, Newton's laws of motion also have analogs for **rotational motion**.

Instead of force, we now have **Torque**. Torque is force that goes in a circle. It is measured in Newton-meters.

All torque acts about a central pivot point. This may be the center of mass of the robot, the center of a gear, the center of a shaft, the center of a wheel, etc.

As indicated by the unit, a torque always has two important things to consider - How much force is pushing in a circle, and how far from the center of the circle is that force acting?

The distance from the central pivot point of the rotation to the point of application of force is called the "**Lever Arm**" of the torque. Bigger lever arms mean more torque for the same force.

This is why wrenches have long handles. They are designed to turn things in a circle with some amount of torque. Assuming a human can only exert a maximum amount of force, a longer handle makes for a longer lever arm, which in turn means a higher max torque applied. If a bolt is stuck, one option is a bigger wrench with a longer handle.

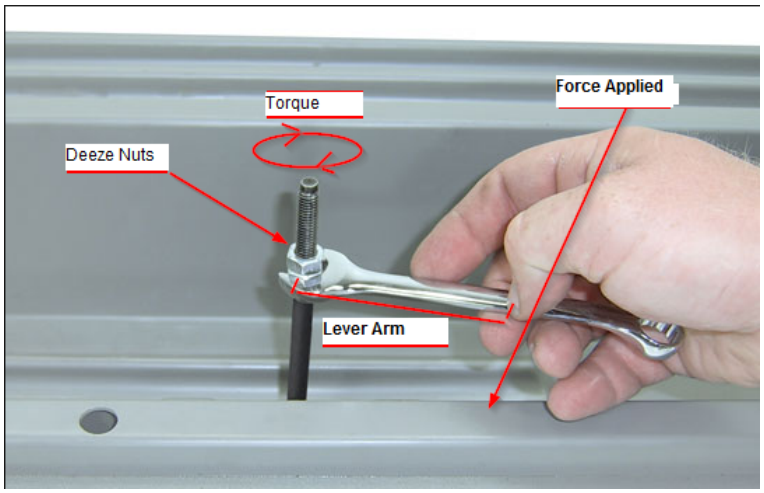

Figure 7: Torque exerted on a nut

3.3.1 Rotational Motion Parameters

There are rotational measurements which are analogous to the linear motion parameters discussed above. Lowercase Greek letters are used to describe these rotational parameters

The current **angle** a rotating object is pointed in is referred to as theta:

$$Angle = \theta[n]$$

The rate at which that angle is changing is **rotational velocity**, or omega:

$$RotationalVelocity = \omega[n]$$

The rate of change of the rotational velocity is the **rotational acceleration**, or alpha:

$$RotationalAcceleration = \alpha[n]$$

As with linear motion, velocity is the derivative of position, and acceleration is the derivative of velocity.

$$\omega[n] = \frac{d\theta}{dt} = \frac{\theta[n] - \theta[n-1]}{Ts}$$

$$\alpha[n] = \frac{d\omega}{dt} = \frac{\omega[n] - \omega[n-1]}{Ts}$$

Also as with linear motion, velocity is the integral of acceleration, and position is the integral of velocity.

$$\omega[n] = \sum_{i=0}^{n} \alpha[i] * Ts$$

$$\theta[n] = \sum_{i=0}^{n} \omega[i] * Ts$$

3.3.2 Torque and Rotational motion

The relationship between torque and rotational motion is highly analogous to force and linear motion:

At any given time, an object's rotational acceleration is proportional to the torque applied to it, scaled by the object's "moment of inertia".

$$torque = moment\ of\ inertia * rotational\ acceleration$$

$$T[n] = I * \alpha[n]$$

An object's "**moment of inertia**" is like its total mass, but also accounts for how far that mass is from the center of rotation. The more mass an object has, and the further that mass is from the center of rotation, the harder it is to get the object rotating at a different speed. The actual value of this is in units of kilogram meters-squared (kg m^2), and is most easily calculated with a table of shapes and corresponding formulas.

Although we have assumed the moment of inertia to be constant, that is not always the case. Although it is rare for a robot to add or loose mass during a match, it is more common for the location of that mass to change on the robot. For example, if an arm extends out by a foot from the robot, the location of the mass of that arm is now further from the center. This increases the robot's moment of inertia, which makes it harder for the drive-train to start or stop the robot from spinning.

3.4 Ideal Gas Law

Cylinders, tanks, compressors, and valves are all pneumatic components used on FRC robots. **Pneumatic components** on a robot all deal with the movement of compressed air, and the transfer of energy involved therein.

Air is the gas used within these systems. Under the normal temperatures experienced in the robot, and the 125psi pressure restriction of FRC rules, air acts much like an **ideal gas**. Therefore, it is appropriate to model it with the ideal gas law:

$$PV = nrT$$

Here, P refers to the pressure of the gas (in kilo-pascals, or kPa), V refers to the volume of gas considered (in Liters, or L), T refers to the temperature of the gas (in kelvin, or K), and n refers to the number of moles of gas molecules being considered. The remaining symbol r is the "**Gas Constant**", a constant with units L*kPa/(K*mol), and is numerically equal to 8.3144621. It is determined by the physics of interacting gas particles. Its derivation will not be described in this paper

Note also that n is in moles. A "**mole**" is a unit-less multiplier, like "dozen". For example, if I have a dozen donuts, it means I have twelve donuts. Similarly, if I have a mole of donuts, I have $6.022 * 10^{23}$donuts. [1] The reason for its use here is that gas molecules are tiny, and usually we have a lot of them. It's a lot simpler to do math talking about "1 mole of molecules" rather than absurdly large numbers.

What the ideal gas law describes is the relationship between pressure, temperature, amount of gas, the volume the gas occupies. For example, when a compressor kicks on, it forces more moles of air into the same volume. We also assume that temperature does not change. Therefore, based on the above equation, if T, r, and V are unchanged, and n increases, then P must also increase to keep the equation balanced. This makes since, since anyone who was observed a properly-set-up pneumatic system knows turning on the compressor makes system pressure go up.

3.5 Electrical Laws

Motors, motor controllers, batteries, wires, and circuit breakers are all part of the **electrical system** on a robot. These components obey many laws related to electricity.

3.5.1 Electrical Introduction

For FRC robotics purposes, **electricity** is the movement of electrons through "conductors". A **conductor** is any substance which has freely-moving charged particles, most commonly electrons. Usually these movements are random, but when they start to go in the same direction, we say there is an "electric current" present. Metals are the most common conductors.

The opposite of a conductor is an insulator. Common insulators include glass, wood, and plastic. These substances to not have freely-moving charged particles, so they cannot sustain an electric current. By wrapping a

[1] Interestingly enough, the average jelly donuts contains 10^6 Joules of energy. This means that one mole of jelly donuts would have about $6 * 10^{29}$Joules of energy. The rotational energy stored in Earth's rotation is about $2 * 10^{29}$Joules. This means that if you had a 100% efficient rocket powered by jelly donuts, and attached it to the earth's equator, you could stop the earth from spinning, and still have 2/3rds of your donuts left over for eating! The more you know...

conductor inside an insulator, you create an isolated conduit to move electrical current in an organized fashion (eg, a wire).

3.5.2 Basic Electrical Quantities and Relationships

When talking about the flow of electrons, there are two important quantities.

Current is the number of electrons flowing past a given point over a given time. It is usually indicated by the lower-case letter i (blame the French). Units are almost always in "Amperes" or "Amps" for short. The Amp unit is denoted "A". One amp of current is equal to $6.24 * 10^{18}$electrons passing a given point on a wire in one second.

$$Current = i[n]$$

Voltage is the "force" which causes the motion of electrons in a conductor. The more voltage, the more of a punch the electrons will have. It is measured in a unit called "Volts" and denoted with the letter "V".

$$Voltage = v[n]$$

Note that the word "force" is used very loosely here, as the energy associated with electricity has much more to do with electric and magnetic fields, and not so much the physical impact of tiny electrons on a mass. However, the details of this theory is beyond the scope of this paper.

Resistance is how easily a substance allows electricity to pass through. Conductors have a very low resistance, while insulators have a very high resistance. Resistance to electrical flow can be affected by many factors, including the temperature and physical shape of the device. Resistance is measured in a unit called an "Ohm", and is denoted with a capital Greek letter Omega (Ω).

The resistance of a device determines how much voltage it pushes back with (ie, the "voltage drop") when a current runs through it. This relationship is linear, and is expressed with Ohm's Law:

$$v[n] = R * i[n]$$

This equation states that for a given amount of current running through a device, that device will push back on that flow with a proportional amount of voltage, scaled by the device's resistance.

For example, Ohm's law explains why system voltage drops when you drive a robot very hard. All batteries have some small amount of internal resistance (something like 0.012Ω). They usually supply around 12V of voltage. However, as motors and solenoids and compressors draw large amounts of current, the battery internally starts to fight itself due to its internal resistance. The voltage actually supplied by the battery is then some quantity less than 12V, and the amount of drop is roughly proportional to the amount of current pulled from the battery.

Inductance and **Capacitance** are properties of devices, describing their ability to store electrical energy and release it over time. The math associated with these is beyond the scope of this paper.

Power is not specifically an electrical topic, but relates strongly to electrical quantities. Electrical Power refers to the total energy absorbed or dissipated by an electrical component over time. Positive power in an electrical component implies energy is leaving the system (via motion, heat, sound, etc.), while negative power means energy is entering the system (Battery, power supply, hand-cranked generator, etc.)

Power is calculated in many ways. In general, it can be found my multiplying voltage and current:

$$P[n] = v[n] * i[n]$$

Power will be measured in units of Watts (denoted "W").

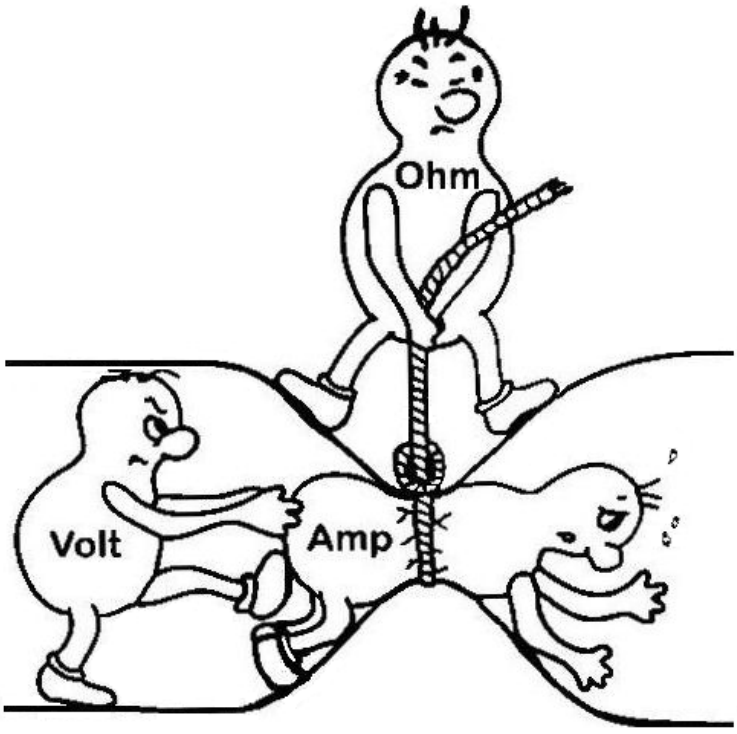

Figure 8: Ohm's Law, with various electrical quantities models as anthropomorphic peanuts.

Part II
Component Models

4 Electric Motor

Simply put, an **electric motor** is a device which converts electrical energy into rotational mechanical energy. Current flowing through the motor causes a force on the motor. However, the interactions of the motor's electrical and mechanical parts also drives a variety of **equilibrium** points. Due to the level of interaction, an electric motor is one of the more complex devices to model. It helps to split the motor into two portions, the electrical and the mechanical parts.

4.1 Electrical Model

A motor, electrically speaking, is a two-terminal device. For a DC motor,its interior windings and communicator will be modeled as a resistor, inductor, and voltage source in series. The resistance and inductance model the physical configuration of the windings inside the motor. The wire itself is always non-deal, having some resistance. Additionally, since the wire is wrapped in tight coils, there is some non-negligible inductance.

The voltage source models what is referred to as "**Back EMF**". Remember that a changing magnetic field inside coils of wire will induce a voltage in those wires (this is how electrical generators work). Although at first it is the energy input to the motor that causes the shaft to start to spin, as the shaft spins it generates its own set of magnetic fields. This secondary field begins to induce a new voltage in the coils called Back EMF. This voltage's value is linearly proportional to the speed the motor is turning at. This is why a free-wheeling motor has a maximum speed.

All these parts of the electrical model create voltages to oppose or aid the flow of current through the motor. At the end of the day, the torque produced by the motor is **linearly proportional** to the current flowing through

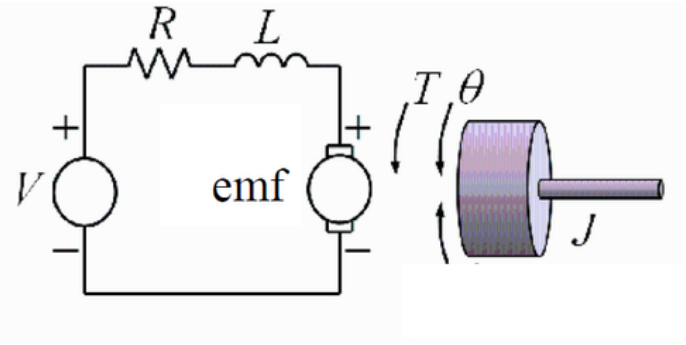

Figure 9: A combined motor electrical and mechanical model

it.

To illustrate, follow this example concerning a free-wheeling motor (no load torque): The motor is stopped, and a voltage is suddenly applied to it. Little current flows at first due to the inductance. However, as time goes on, the magnetic field builds up and more and more current flows. As current begins to flow in ever increasing quantities, more and more torque is applied to the output shaft. The output shaft begins to accelerate due to the torque. As the output shaft picks up speed, the back EMF voltage also increases. Soon, it is no longer the inductance that is limiting current flow, but instead the back EMF voltage source. Eventually, an equilibrium is hit where there is (theoretically) no current flow through the motor anymore, as the back EMF voltage completely counters out the supply voltage.

In reality, there is always a small current draw due to the load torques from friction in the internal bearings (and various other reasons).

Given a supply voltage and a back EMF, applying KVL around the motor circuit in Figure 9, we deduce the following continuous-time equation for the motor current:

$$V_{in}(t) = I_m(t)R_m + L_m\frac{dI_m}{dt} + V_{emf}(t)$$

Here, $V_{in}$is the supply voltage to the motor, $R_m$and L_m are the motor's winding's resistance and inductance (in Ohms and Henries), and V_{emf} is the Back EMF voltage from any rotation of the output shaft. The $L_m\frac{dI_m}{dt}$term comes from the voltage-current relationship in an inductor. Simply put, an inductor produces a large voltage in response to large changes in current (hence, the derivative term). The derivation of this relationship is beyond the scope of this document.

Moving to the discrete domain, we find the following equation:

$$V_{in}[n] = I_m[n]R_m + L_m\left(\frac{I_m[n] - I_m[n-1]}{Ts}\right) + V_{emf}[n]$$

We now solve for $I_m[n]$, the motor current at the present time-step:

$$I_m[n] = \frac{1}{\left(R_m + \frac{L_m}{Ts}\right)}\left(\frac{L_m I_m[n-1]}{Ts} - V_{emf}[n] + V_{in}[n]\right)$$

4.2 Joining of Electrical and Mechanical Models

Now, we shall introduce some constants to link the mechanical and electrical models together.[2]

These can be derived from analyzing the physical construction of the motor, but are usually empirically measured.

The **Torque Constant** K_t determines the relationship between Torque and Current. We define the following linking equation:

[2]Data for this section can be found in the CIM motor curves available from Andymark: http://files.andymark.com/CIM-motor-curve-am-0255.pdf

$$T_m[n] = K_t * I_m[n]$$

$T_m$is the torque produced on the output shaft of the motor.K_t is measured in units of Torque per Current, or Nm/A

It is known that at zero current, zero torque is present (hence, no need for any constant offset in the above equation). K_t is most easily calculated from a motor's Stall Torque and Stall Current, two figures readily available for most motors.

$$K_t = \frac{T_{stall}}{I_{stall}}$$

In the case of a CIM motor, the stall torque is 2.429 Nm, and the stall current is 131.227 A.

The **Speed Constant** $K_i$determines the relationship between the magnitude of the Back EMF voltage, and the rotational speed of the output shaft. We define the following linking equation:

$$V_{emf}[n] = K_i * \omega_m[n]$$

$\omega_m$is the rotational speed of the output shaft of the motor. $K_i$is measured in units of Voltage per Rotational Velocity, or V/(rad/s).

It is again known that at zero speed, there is zero back EMF. $K_i$is most easily calculated by analyzing the motor in its "free-wheeling" speed, where a positive voltage V_{supply} has been applied to the motor, and a maximum speed has been observed sustained at the output shaft (ω_{FW}). Assuming the input current in this state (I_{FW}) is also known, the following relationship is true:

$$K_i = \frac{V_{supply} - I_{FW} * R_m}{\omega_{FW}}$$

Be sure to add scaling constants for units as needed - Current is often reported in mA (not A), and free-wheel speeds in RPM (not rad/s).

Substituting these Electrical/Mechanical constant relationships in and re-arranging terms, we arrive at the final equation used in the motor model:

$$T_m[n] = T_m[n-1] + T_s * \left(\frac{K_t}{L_m} \left(V_{in}[n] - \frac{T_m[n-1]R_m}{K_t} - K_i * \omega_m[n-1] \right) \right)$$

$$I_m[n] = \frac{T_m[n]}{K_t}$$

We have now reduced the motor to a model which takes a voltage, speed, and previous torque as input, and determines what the new torque value should be. Additionally, via the second equation, we can determine the current draw of the given motor.

4.3 Integration with Larger Systems

The one remaining relationship unexplored is the speed-torque relationship in the motor. Thus far we have defined the motor which produces some torque, dependent upon the voltage applied and the speed the motor runs at. However, once this torque is applied, how quickly does the speed change? This question can only be answered by analyzing what the motor is hooked up to. In the case of a free-wheeling motor, not attached to anything, the answer lies in the moment of inertia of the shaft itself, and the rotational friction in the bearings. When driving an actual robot, the torque of the motor is transferred through the drive-train to the ground. The wheels convert the torque to a linear force, and start rotating in response to the net acceleration of the robot. This rotational speed is then transferred back up through the drive-train to the motors. In all cases, friction provides a constant drag on motion. The exact math in determining the torque-speed relationship must be derived from the device attached to the motor, not the motor itself.

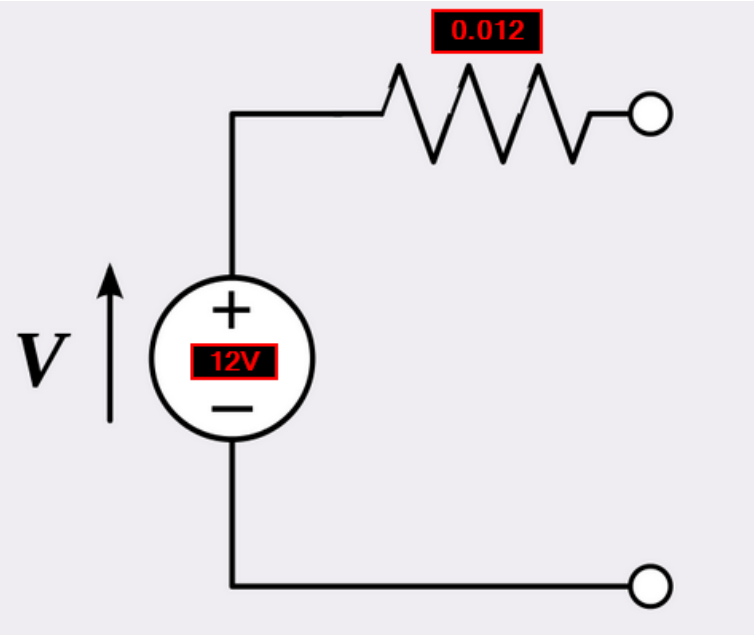

Figure 10: A non-idea voltage source, modeling a battery

5 Main Battery

There are many sources of power for electrical circuits. While analyzing circuits, the most common source used is what is referred to as a "Ideal Voltage Source". This is a device which produces a voltage across its terminals, and that voltage does not change with the amount of current flowing through the device. In theory, such a device could provide an infinite amount of power (keep voltage the same as current increases indefinitely). This assumption makes analysis of circuits easier, but fails to provide the whole picture.

5.1 Modeling Voltage Drop due to Instantaneous Current Draw

A battery is what is referred to as a "Non-ideal Voltage Source". It can be modeled with an ideal voltage source in series with a resistor. The voltage source models the current-pushing potential of the chemical reaction in the battery, and the resistor models the internal parasitic resistance which causes voltage to drop as more current is drawn. The MK ES17-12 batteries standard for FRC[3] have an average internal resistance of 0.012Ω. The nominal (no-current-draw) voltage of the battery can be anywhere from 13V down to 8V or even lower, depending on how much the battery is charged.

For the purposes of RoboSim, we have not yet modeled the decaying nominal voltage of the battery as it discharges, since the simulation is not designed to be accurate for long-term endurance tests. However, we have modeled the drop in supply voltage with elevated current draw. See Figure 10 for the circuit model employed.

In general, the output of the battery model is the system voltage, and the input is the total current draw from the battery. The model is governed by the following equation:

$$V_{sys}[n] = V_{nom}[n] - R_{bat} * I_{draw}[n]$$

$V_{sys}$is the robot system voltage at a given sample time n. $V_{nom}[n]$is the nominal (no-current-draw) voltage of the battery. For simplicity it may be assumed to be 12V, but could also be calculated based on the remaining charge in the battery. $R_{bat}$is the parasitic internal resistance of the battery, assumed equals to 0.012Ω for the ES17-12. Finally, $I_{draw}$is the total current demand of the robot at sample time n. Within the plant model, the total current draw is calculated by summing the current draw from each motor, solenoid, compressor, etc., along with nominal background current draw (such as electronics, lights, or fans).

[3]Constants derived from data-sheet at http://www.mkbattery.com/images/ES17-12.pdf

5.2 Modeling Total Remaining Charge based on Aggregate Current Draw

Additionally, the battery model uses$I_{draw}$to calculate the total charge remaining in the battery. This charge-remaining metric could be used to drop nominal battery voltage over time, but will also give developers indication of the amount of battery life drained by performing certain maneuvers with their robot. The battery is designed to have 17 Amp-Hours of charge in it. This means that you can pull 17 amps for one hour, one amp for 17 hours, or any other combination of amps and hours which, when multiplied, gets you to 17. We model battery charge by calculating the number of Amp-Hours drained from the battery by the current draw every plant loop. This behavior is defined by the equation:

$$Q[n] = Q[n-1] - \frac{I_{draw}[n] * Ts}{3600}$$

$Q[n]$ is the present charge (in amp-hours) in the battery, and $Q[n-1]$ is what the charge was last loop. Here, we always calculate the present charge by subtracting the current draw (in amps), multiplied by the sample time (with a 3600 factor to convert Ts from seconds to hours). This charge metric can be used to analyze expected battery life of a given robot design.

6 Main Circuit Breaker

7 Compressor and Tank

The compressor and tank library model reporesent the source of pneumatic energy for the robot. The compressor modeled is the Viair 90C, standard for FRC robots. The air tanks have an adjustable volume (set by defining the number of half-liter tanks).

7.1 "The System"

For all pneumatic calculations, the volume inside the tanks, tubing, valves, etc. is modeled collectively as the "system". In principle, there are a certain number of air molecules inside of this system at all time. The model track the number of moles of air inside the system, and then can calculate the pressure using the total volume and an assumed temperature. The compressor as a device adds moles of air to this system, while any leaks or cylinder actuations reduce the number of moles in the system.

The system requires a "reset" trigger at the start of the simulation. When reset, it triggers calculations to determine nominal (ie, one-atmosphere) quantities of air molecules in the system. This initalizes the state variable to the right number of moles of air for an assumed temperature.

The temperature of air in the system is assumed to always be at 70°F. This is based on the idea that the compressor does not induce much extra heat into the air it compresses, and any extra heat is dissapated to the atmosphere through the compressor metal body, tank walls, and tubing. Making this assumption allows us to not have to calculate the thermal equations relating heat loss through a complex system of many different materials of different shapes.

7.2 Compressor

Viair provides performance data for its compressors. For the 90C model, there is a non-linear relationship between flow rate, pressure, and drawn current. At any given time, the compressor must fight the system pressure to put more air into the system. Additionally, the faster the flow rate and the higher the system pressure, the more current is needed to keep pushing moles of air into the system. The numerical values can be found in the compressor's datasheet, and are plotted with interpolation in Figure 11.

Note that the performance data is listed in imperial units, so some conversion factors are applied in software to get to the metric equivilants.

Every simulation loop, a total number of new moles of air to be added to the system is calculated based on this performance data, and whether the compressor is running or not. By using the ideal gas law, and assuming the system pressure is known, the number of new moles of air can be calculated:

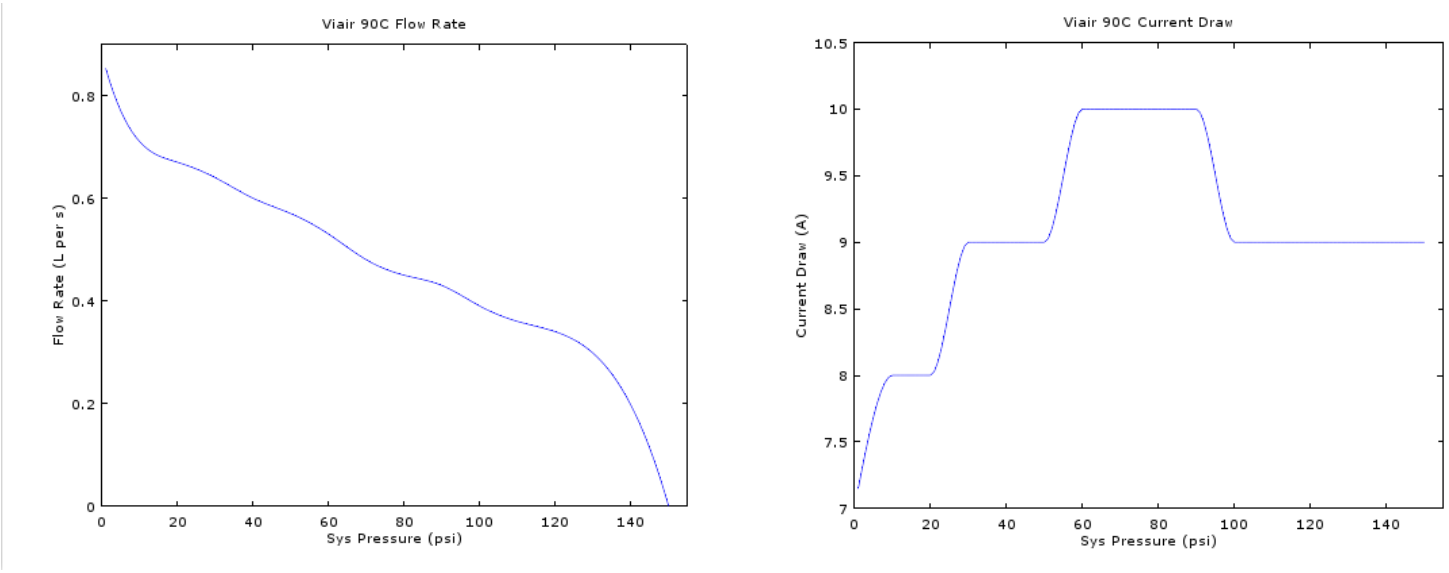

Figure 11: Viair 90C Performance Data

$$\triangle num\_moles\_comp[n] = \frac{P[n-1]F[n] * Ts}{rT}$$

For P being the previous system pressure, F being the volumetric flow rate (calculated from performance data if compressor is running, or 0 if not running),Ts being the sample time, T being the system air temperature (assumed equal to70°F), and $r = 8.31$ being the gas constant in units of $\frac{L*kPa}{K*mol}$.

Finally, given the current system pressure and running/not-running state of the compressor, the current draw at timestep n may also be calculated.

8 Traction Wheel

Part III
Integrated Model

There are multiple parts to the robot model. Each shall be discussed in detail here.

9 Electrical Subsystem

The electrical subsytem consists of the robot's power source (the main battery), all current drawing devices (motors, solenoids, compressors, etc.), and any safety devices (main circut breaker, etc.).

One of the key metrics to track is total current draw from the battery. As soon as we know the compressor state and motor torques, we calculate total current draw with the following equation:

$$I_{draw}[n] = (I_{MR}[n] + I_{ML}[n]) * N_M + I_{comp}[n] + I_{nom}$$

Here, $I_{ML}$and $I_{MR}$are the current draw from the left and right motor respectively. $I_{comp}$is the air compressor current, which is assumed 0 until the pneumatic system is implemented. Finally, I_{nom}represents background, nominal current draw from the roboRIO, LED's, fans, etc. We have modeled it as a constan 0.5A current draw.

System voltage is the voltage supplied by the battery to all of the robot components. Usually it is near 12V, but large current draws or weak batteries may decrease it. This $I_{draw}$is used to calculate the present system voltage via the equations laid out in section 5. Additionally, the main battery charge is tracked via the equation presesnted in that same section. $I_{draw}$is also used by the main circuit breaker model to determine if the breaker should be in a tripped or closed state. If tripped, system voltage is driven to zero. See section 6 for more details on this component.

Once the system voltage is known, it can be used to calculate adjusted motor voltages driving the motors. There is feedback here: as motors push harder, they draw more current, which drops the system voltage. In turn, this prevents the motors from outputting their maximum power. This systsem-voltage-drop relationship is maintained by treating the voltage input waveform (whether from the RoboSim hardware measurements or from test vectors in a fully-simulated setup) as "voltage commands". The actual voltage going to each motor is calculated off of this equation

$$V_M[n] = \frac{V_{sys}[n]}{12} * V_{cmd}[n]$$

Where $V_{cmd}$is the commanded voltage to the motor, $V_{sys}$is the robot system voltage, and $V_M$is the voltage actually applied to the motor.

10 Drivetrain Subsystem

The drivetriain consists of the components which exert forces on the net robot body, and move it around the field. Conceptually, it transforms torque from the motors, and calculates what forces those torques exert on the robot. Then, an acceleration, velocity, and position of the robot body can be calculated from these forces. Rotation is also accounted for. Finally, the net motion of the robot is transformed back into a speed, which is fed back up through the drivetrain to determine what rotational speed the motors are running at.

For Robot_15, a simple tank drive is modeled. The Drivetrain consists of two sets of drive wheels/motors/gearboxes - a left and a right set. Each set of drive equpment has a given gear ratio, wheel diameter, and number of drive motors. It is assumed that each drive equipment set can exert a force toward the front or the back of the robot, and that force is applied at a point offset to the leftt or right of the robot's physical center. The distance from the center to the point of force application is called the "Motor Width"

The robot additionally has a width and height (the half-width and half-height are shown, and are used in the software for calculations). The X and Y axes are also provided. Note that all linear velocity components are assumed relative to reference frame fixed on the field, not on the robot (unless otherwise indicated)

The first step in the calcualtions for the drivetrain are to calculate all the forces acting on the robot.

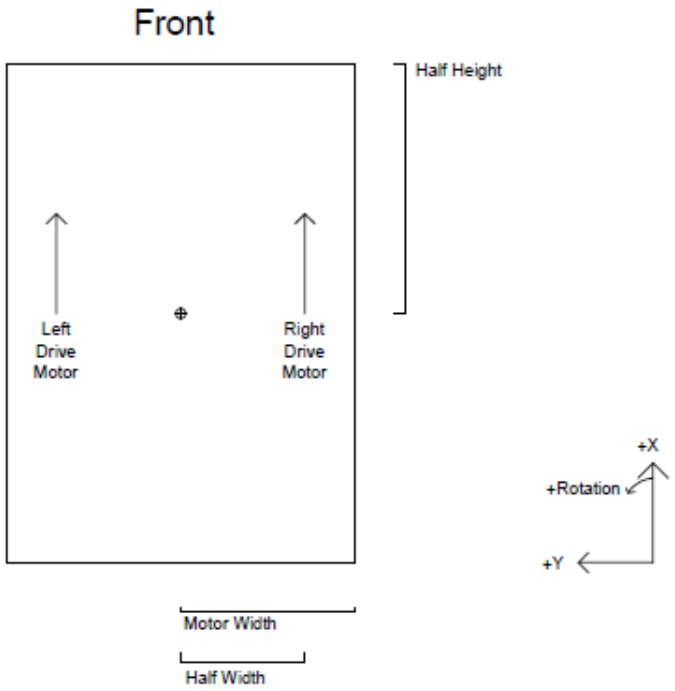

Figure 12: The model of the tank drive robot. Note that MotorWidth and HalfWidth are mixed up, this should be changed.

10.1 Linear Force from Drivetrain Motors

The torque from the motor is a function of the motor's speed and the supply voltage. See section 4 for the derivation of torque from speed and supply voltage. Given the left and right motor torques, a linear force magnititude on the robot can be calculated:

$$F_{lin}[n] = \frac{(T_L[n] + T_R[n]) * N_M}{D_W * R}$$

For $T_L$and $T_R$being the torque (in Nm) produced by the left and right motors respectively, $N_M$being the the number of motors powering each side of the drivetrain (usually 2 or 3), D_W being the wheel diameter (in m), and R being the gear ratio of the gearboxes and any sprokets used. Note that R might actually change over time if shifting gearboxes are used.

10.2 Net Rotational Torque from Drivetrain Motors

Next, because the forces from each side of the drivetrain act at a non-zero difference from the center of mass of the robot, they may exert a net torque on the robot. This is what allows the robot to rotate in place. The magnititude of this torque on the robot is calculated:

$$T_{net}[n] = \frac{(-T_L[n] + T_R[n]) * N_M * W_{mot}}{D_W * R}$$

All variables used here are the same as above, with the addition of the motor width, W_{mot}in meters.

10.3 Wheel friction

Wheel friction is currently under development. So far, we will only calculate kinetic friction which opposes the robot from sliding side-to-side (perpendicular to the direction the drive wheels point). Note this force is large for

traction wheels, but very small for omni-wheels. By adjusting the coeficent of friction in the robot configuration structure, different types of wheels can be tested.

The first step is determining what component of the robot's velocity is perpendicular to the direction the wheels are pointed

$$\Delta\theta[n] = atan2\left(\frac{V_y[n]}{V_x[n]}\right) - \theta_{robot}[n]$$

$$V_{perp}[n] = \sqrt{V_x[n]^2 + V_y[n]^2} * sin\left(\Delta\theta[n]\right)$$

For V_y and $V_x$being the y and x components of the robot's current velocity, and θ_{robot}being the current direction the robot is pointed in.

Then, using the equations for kinetic friction described in section 3.2.5, we can determine a magnititude for the frictional force from the wheel opposing side-to-side motion:

$$F_{fric,perp}[n] = -V_{perp}[n] * K_{k,wheel}$$

For $K_{k,wheel}$being the net kinetic coefficent of friction for all wheels in contact with the ground.

10.4 Induced Motion

We now have all forces requried to calculate the net motion of the robot from the drivetrain. We will use the relationship force equals mass times acceleration, solved for acceleration. We will consider the X and Y components separately. We must use trig to determine the net X/Y motion based on the direction the robot is pointed, which in turn dictates which direction the drivetrain can push or apply friction in.

The acceleration in the X and Y directions (WRT a fixed reference frame of the field) for the robot is:

$$A_x[n] = \frac{F_{lin}[n] * cos\left(\theta_{robot}[n]\right) + F_{fric,perp}[n] * -sin\left(\theta_{robot}[n]\right)}{M_{robot}}$$

$$A_y[n] = \frac{F_{lin}[n] * sin\left(\theta_{robot}[n]\right) + F_{fric,perp}[n] * cos\left(\theta_{robot}[n]\right)}{M_{robot}}$$

For M_{robot}being the robot's mass in kg.

Additionally, we also calculate the rotational acceleration from the induced torque:

$$\alpha[n] = \frac{T_{net}[n]}{I_{robot}}$$

For I_{robot}being the robot's net moment of inertia (like mass, but for spinning things). Not having any better ideas, we modeled the robot's mass as being uniform throughout a subsection of its base. By making this mass-distributed-in-a-rectangular-prisim assumption, we can assume the following value for the moment of inertia:

$$I_{robot} = \frac{M_{robot}}{12} * \left[\left(0.9 * L_{robot}\right)^2 + \left(2 * W_{motor}\right)^2\right]$$

For M_{robot}being the robot mass in kg, L_{robot}being the robot's total length from front to back bumper, and W_{motor}being the distance from the robot's centerpoint to the point of force application of the drivetrain(in m).

Now that we have the accelerations, we can use the physics equations determined in section 3.1 to deduce the robot's current position, velocity, rotation, and rotational velocity.

10.5 Motor Speed Calculation

Finally, we must pass back up the robot's net motion to calcualte the speed each motor is running at. This is crucial for determining accurate future torque values.

We must account for the robot's current orientation on the field, it's X and Y velocity, and its rotational velocity. For both motors, we calcualte speeds with the following equations.

$$\omega_L[n] = \frac{[V_x[n]cos\left(\theta_{robot}[n]\right) + V_y[n]sin\left(\theta_{robot}[n]\right) - \omega_{robot}[n] * W_{motor}] * 2\pi}{D_W * R}$$

$$\omega_R[n] = \frac{[V_x[n]cos\left(\theta_{robot}[n]\right) + V_y[n]sin\left(\theta_{robot}[n]\right) + \omega_{robot}[n] * W_{motor}] * 2\pi}{D_W * R}$$

The next control loop, these new speeds will be fed into the motor model to calculate the next torque.

11 Pneumatic subsystem

This has been paritally implemented, and not at all documented. To be updated....

12 Field Collision Model

The field collision model determines behavior when the robot collides with a wall of the field. The collisions are inelastic - that is to say, some of the energy is absorbed by the bumpers. This means the robot bounces back on a collision, but is traveling slower.

The field collision model is still very crude, and for that reason will not be described in depth. Suffice to say, the basic principe is that when the robot comes in contact with a wall, one part of the velocity (X direction or Y direction) is reversed and reduced to simulate a "bounce". Future work will model this bounce more accurately, and also examine the rotational effects of a corner of the robot hitting the wall before the rest of the robot does.

Part IV
Design Notes

13 Software Architecture

The sim software architecture is desiged to be flexible and modular, with large parts that can be re-used year to year. Most "components"' to the simulation will have an init (run once) and a main (run once per loop) file. We will briefly describe each file

13.1 sim_main

Sim_main.m is the main entry point for the simulation. This is the file to run in Octave when you wish to run a simulation. It calls all the initilization functions and runs the main control loop.

After clearing the Octave workspace, sim_main will load the instrument control package (needed for serial communication with the RoboSim hardware), add additional folders to its path (so it knows where to find utility and library files), start up some logging, and then call additional initilization inside sim_init.m. After that, a progress bar is displayed and a timer is started to ensure the perioidic loop is run at the correct rate.

The periodic loop will run for the number of timesteps specificed at sim initilization time.On each loop, the first thing done is to mark the time the loop started at. This will be used at the end to determine how long to wait to maintain a consistant run rate. Next, the inputs for this control loop are gathered via in_proc.m. Then the robot plant model is run, and any robot/field physics calculations are run. Then, the results of this loop are set to outputs via out_proc.m. Lastly, if it is desired to run this simulation in real-time, the simulation waits for a given amount of time to ensure a consistant loop run time (ie, sample rate).

Finally, after the simulation has completed the required number of main control loops, the post_proc tasks are run. Then the simulation exits.

13.2 sim_init

Sim_init contains some of the simulation-enviroment-specific setup and data structures. Here, you can set the sample time (reccommend 50ms. Also, any changes here must have corresponding changes in the arduino software as well). You can also specify simTime, the length of time in seconds the sim will run for. There are two other important switches: one to decide if input comes from predefined simulation testcases or from RoboSim hardware (the use_serial variable), and another to determine if the simulation should attempt to execute in real time (ie, each control loop is 50ms) or not (ie, each control loop runs as fast as possible). Running real-time is nice to get a good conceptual feel for how the robot will look as it moves around, and is required if use_serial is 1. However, if plots of test results are of primary concern, you can set enforce_realtime to 0 and get faster execution time.

After setting these variables, sim_init runs initilization for the robot and for the field, then opens up the serial connection to the RoboSim hardware if hardware is being used.

13.3 robot_init_15

Robo_init is run once before the start of the main simulation loop. Its purposes is to initalize the internal state and configuration structutes of the robot, and draw the initial robot onto the field.

There are three main structures - config, state, and calc_config. Config values will be constant throughout the simulation. They represent the physical properties of the robot (physical dimensions, weight, number of motors, number of air tanks, etc.). These physical paramaters are all used in the physics-based calculations of the robot's actions. Note that the units of each parameter should be included into the comments associated with the params (units matter! And there's little oppurtunity to figure them out from the rest of the code!). The state structure holds all the physical paramaters which change over time (acceleration, rotation, position, system voltage, etc.). Again, units should be specified. Since this is a digital system, it's expected that some of the values will have "current" and "previous" values. At the end of each main loop, the structure should be considered to have all the data associated with the robot at timestep n. Finally, there is an additional structure for the calculated config values. These should not be edited from robot to robot - the values are derived from other config values. These

are things like unit-converted values (ex: mass in kg) or the moment of inertia, which can be calculated from other known values.

After the structures for the robot have been defined, a few more are added for specific components. Currently, this is for the motors.

Finally, a few calculations are done to draw the initial robot onto the field. The initial figure is selected, and the robot square is drawn. Additionally, a small marker circle is drawn on the front of the robot to differentiate it from the back.

13.4 field_init_15

Field_init is responsible for setting up the field paramaters structure, and drawing the field. The field structure contains info about the physical dimensions of the field components, as well as various drawing paramaters. After the struct is initalized, a figure is created, the axis is set up to be used as a canvas to draw the field on, and the actual components of the field are drawn. This drawing needs only be done once, unless a field element is animated.

13.5 in_proc

In_proc is the input processing script. It is responsible for getting inputs from some source, and updating the input structure based on those inputs. The input structure is its own set of time-changing values, containing the current set of inputs (usually motor and solenoid voltages).

In general, if a serial port connection is used, the serial_read_packet function is called to get input from the arduino. On a bad packet being recieved, the old packet is maintained and a warning is printed to screen.

If no serial port is needed, the input structure is instead populated using some pre-defined time-voltage vectors. These vectors are defined within the in_proc script itself. The define a voltage to set, and the time at which to set it. Between given points, linenar interpolation is used.

This "no-serial-port" mode allows robot designers to test different driving conditions with different physical parameters, with no electrical board built.

13.6 robot_15

The Robot_15 script executes the integrated model of the robot. No data structures are defined here - the robot state and outputs structures are updated based on the inputs, physical parameters, and previous state. See Part III for more information on the content of the robot_15 script.

13.7 field_robot_physics

The field_robot_physics script re-adjusts the robot state structure based on any interaction between the robot and the field. Usually, this interaction means the robot ran into a wall or a part of the field. See section 12 for more info on how collisions are handled.

13.8 out_proc

The Output Processing script is designed to, well, process outputs. This means updating any animations based on the newly determined state of the robot, and recording values of importance. To limit memory usage, only specific signals are recorded over time - output processing is where a full time-based vector of imporptant values (ex: robot x and y position) is updated with the current value. For most values, only the current (and maybe the previous) value is kept in memory. But for signals which will be analyzed after the simulation is complete, a vector which has enough space to hold all values of the signal during the simualtion is updated.

Additonally, the verticies of the robot on the field are recalculated and re-set. A command is called to update the drawing on the screen.

13.9 post_proc

After the simulation is completed, the recorded vectors must be plotted. These plots are generated with proper scaling and labeling in the post_proc script. Additionally, if a serial port was used, it is shut down cleanly in this script.

14 RoboSim Hardware

To be completed.....