Background

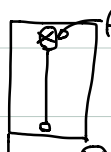
- 2012-2013 Season ~ 2nd official season, the challenge was to put rings on pegs
- April 2013, we had made it to the finalist round but failed to move to worlds
- The night before, I was taking up the autonomous program Carlton had failed to complete
- There were only light sensors, no Entach
- This wouldn't have been an issue except for the fact that there were no lines leading to the pegs
- My only option was to time the robot's movements (3sec forward, 1 sec left etc.) in order to get to the line so that the light sensors could activate
- I stayed from 10:00 or 12:00 to 8:30 pm that day but it still failed
- Why? I was relying on the assumption that motor speed stays constant
- If you look at the NXT voltage readout, it decreases a fair amount from fully charged
- So, when I told the robot to move forward 3 sec while the battery was charged, it went a very different distance from when the battery was half charged (motor speed x wait time = distance)
- I also noticed it's distance was offset by ledges, ramps, and other obstacles
- Much like how a brain was created for the purpose of controlled movement of living organisms, Entach was created for the controlled movement of a robot

The birth of Entach

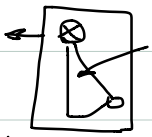
- I didn't know the engineering process and had little experience with sensors
- I did, however, just take Honors Physics and understood acceleration in terms of kinematics
- I knew in order to direct a robot anywhere, the robot needed know it's position whether in x,y or r,θ form
- The first thing that struck me was the accelerometer since it measures acceleration and tilt in all three axes
- This was a mistake and would cause Spencer and I to waste the next month on the accelerometer
- There are two problems with the accelerometer:
    1. It cannot distinguish between tilt and acceleration
    2. It's terribly noisy and imprecise

[Demo]

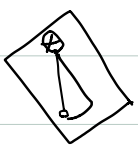
- How an accelerometer works:



- when ⊗ accelerates



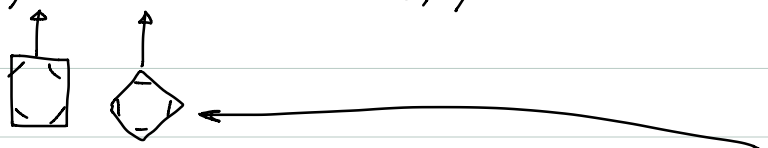
- however, it does the exact same when tilted



- Second problem is noise
  - if you look at the graph of acceleration, it looks like static
  - This is a huge problem because any noise gets magnified when converting to distance
  - Calculus fans will know why this is
- Bottom line: only use accelerometers for imprecise measurements like whether you're moving or not

- During the time that we thought the accelerometer was good, we realized there are two (actually three ways) of doing this:
  1. crab legging - only travelling vertically or horizontally
  2. Travel based on x, y input
  3. ⊢——⊣ r, θ input

- All three of these require three variables for the robot to keep in mind:
  - current x, y, & θ
  - If you don't have current θ, your robot is forced to face 1 direction



  - Calculations for driving a certain direction are different for different angles

- We realized we needed a way of calculating θ which the accelerometer cannot do
- We looked to the compass sensor which senses angle relative to Earth's magnetic field
- This would've worked just fine accept we started to notice something strange

[Demo]

- If we measure the compass at specific angles with stationary motors around it works fine because it's measuring angles relative to the motor's magnetic field
- However if we move the motors with compass sensor, all of it's readings become skewed
- You may notice, however, that the skewed nature is a predictable curve
- In theory we can get the equation of this curve and use it to correct for the error
- I tried this and it worked! once... before I accidently nudged the compass sensor
- The equation is very specific about sensor placement, so when nudged the sensor, the equation I made was no longer valid

The age of encoders

- Around the time I made this disappointing discovery, I had abandoned the accelerometer and replaced it with an encoder on a motor
- I had finally struck gold
- Unlike accelerometers, encoders have no noise and (when the disks are clean and assembled correctly) have no error
- Spencer and I, at least, agreed that the crab leg idea was pretty bad and decided the robot should turn to face it's destination then drive towards it
- This was all great except we no longer had the means to determine our angle
- At this point two important insights were made:
    - The encoder should be detached from the motor
    - Arc/radius = angle
- If you remember, I made an observation that whenever the robot hit any form of obstacle, all "distance" calculations afterwards would be skewed
- In tandum with the fact that we may use an omibase next season I had decided the best solution was detach the encoder from the motor and create a "dead wheel"
    - Thus the name "Entach" was born: **En**coder Det**ach**ed (pronounced En·tak)
- As for Arc/Radius = Angle, this was a brilliant suggesion by my dad
- If you think of radius as a scale factor of a circle: as radius increases arc length must also increase, so the net result is that angle remains the same which should be true
- As soon as discovered this, I abandoned the compass and all calculations were now being made using just one encoder

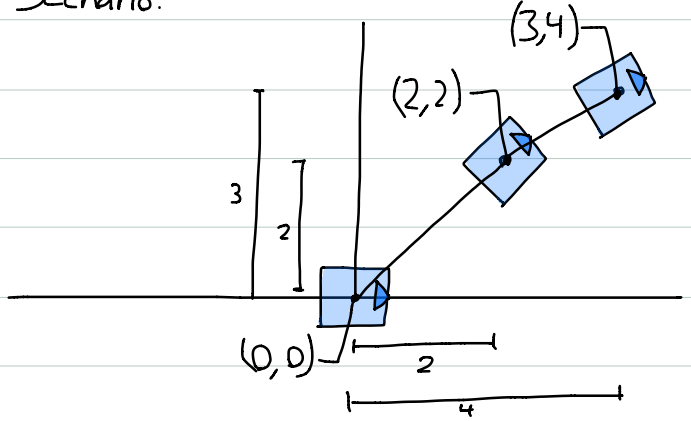
Defining the Coordinate system

- With all the hardware ready, the next phase was the software side of Entach
- School had started so I resumed work on Entach with Spencer and David
- If you recall there were three ways to write this program:
    1. crab legging - only travelling vertically or horizontally
    2. Travel based on x,y input
    3. ⊢——————⊣ r,θ input
- Spencer and I had eliminated the first option
- We thought it logical to go with option "3." since we had decided the robot would turn to face it's destination then drive towards it
- And that's what we went with until we actually had to measure out the points
- If we wanted the robot to go anywhere, we had to measure an angle and a radius
- This is why I bought the electric protractor
- To top it off, I hadn't set up an absolute coordinate system
    - This means every time the robot stops the origin is set at that point
    - You can imagine how cumbersome and ridiculous the set up looked
- We needed to re evaluate the way we handle coordinates

- At this point, I felt the RobotC jargon was getting in the way of me sorting this out
- So, I started fresh by starting a java program
- There were two issues at hand:
    1. No absolute position support
    2. x,y or r,θ input

- Based on the nightmare of measuring out θ & r, we settled on x, y input
  - But the robot would still turn and drive
  - This means we have to convert x & y to r & θ
- We tried many different algorithms including ones that involved law of sines and cosines
- Since my thought process surrounding absolute position wasn't recorded, I only have a graphical explanation
- This methodology is still being used today in goToPoint()

Scenario:



1. convert input x, y → r, θ
   - This is where the "absolute" coordinates come in
   - We realized that all the robot cares about is where to go next and current x, y, & θ should be stored separately
   - Essentially what we had turn absolute coordinates into relative coordinates
   - So instead of taking the distance formula and arctan of (3,4), we used the following:

$$\sqrt{(3-2)^2+(4-2)^2} : \text{relativeRadius}$$

$$\text{atan}((3-2)/(4-2)) = \text{goalAngle}$$

   - This gives us the radius and angle from point (2,2)
   - This (r, θ) value can now be transformed into motor instructions

- As a final step, you have to subtract currentAngle from the result of atan in order to get how far the robot should turn (turnAngle)

2. goToAngle & goToDistance
   - Now all we need to do is feed turnAngle and relativeRadius into these methods and exactly what we wanted to do in the first place will happen!!!
3. current X, Y, θ = inputX, Y, & goalAngle

- There is one little patch however. If you notice, in the code there's this:

```
if (inputX < currentX)
    goalAngle += PI;
```

  - This here for two reasons:
    1. currentAngle's range is from 0 to 2π (Tau)
    2. Issue 1. cannot be resolved by adding π to all results of atan() because that would cause atan(0) to equal π which is not true
  - To satisfy both of these conditions, we only add π if inputX < currentX
- It was at this point that I started to work on Entach in context of "Block Party"

The Battle against Drift

- As the year progressed, I grew more happy but also more upset with Entach
- It was sort of getting to right place but not quite
- I didn't realize it at the time, but it was almost entirely because of drift
- At the time, I thought it was because my "robot radius" and "encoder to cm" constants were off which was true to a certain degree
  - This compelled me to write an Entach calibrator program and toil for hours and hours fruitlessly on this problem
- With each competition / scrimmage I managed to pull a fluke that would compensate for drift, sometimes without me knowing it!
  - for the first scrimmage, I faked the robot radius
  - for the second scrimmage, some hidden factors causing drift aligned in a way that it didn't appear
  - For qualifiers, I made the robot go a little further forward with each crate
  - for states, I threw in some voltage correctors but it didn't quite fix the problem
- At this point we are in the present
- There are two paths from here:
  1. Either find out what's physically wrong with the robot
  2. Compensate for drift through software

The Present

- We are still working on drift but there still so much potential within Entach
- Some of it was realized this past week (week of 3/9/14)
- As I've explained, measuring points can be very annoying
  - because of this, I've created a java applet that allows you to create goToPoints visually and quickly
  - Also, since the origin in the applet is not where the robot starts I successful created "initializeAxes()", something I've tried in the past
- Another cool feature I've added is strafing support
  - I couldn't figure out how to implement strafing before without having a lot of user input
  - Inspired during my work on the Entach Field Mapper, I've added a third parameter "endAngle" to goToPoint
  - What this parameter says is to turn and face this angle at the end of a run
  - On it's own this useful for when you need the robot facing a particular direction at the end of a run
  - for strafing, if two endAngles equal each other, this signals the program that it should strafe to that point

What's next?

- Fix drift
- Explore the weirdness of "if (inputX < currentX)"
- Add new features
- Adapt it to the new challenge
- You decide!
  - The fate of Entach is truly in your hands as soon as I leave for Carnegie Mellon