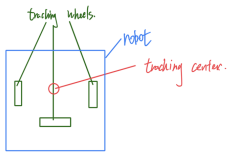
Project Title: Wheel Odometry
Group Member: Dheer Baldua, Yash Mathur, Steven Song
Type of Project: Combination of Hardware and Software.
Summary: In our final project, we aim to create a wheel odometer. We will compare the position and orientation data we collect to the respective true values which will be obtained through physical measurement.
Level: Undergraduate

Introduction: Our project is to build a robot that uses wheel odometry. Wheel Odometry will use trigonometry at each timestep along with the data from the sensors(wheel encoders) to estimate where the robot is and its orientation. We are going to be programming the wheel odometer with python coding.

Using a Wheel Odometry Kit, bought from Amazon, a robot will be built. The robot will consist of wheel encoder sensors, three dead wheels, and an arduino which will help us read data. The written code will help us calculate and eventually track the robot. The aim of this project is to create a wheel odometer and test its accuracy.

Background: Odometry, by definition, is the process of using different motion sensors to detect changes in the position of a robot. This process can be used in different robotics projects including VEX and FRC robotics competitions as well as in some autonomous vehicles and drones. There are different types of odometry depending on which kind of sensor the system is used, the most common of which are encoders, visuals, and distance sensors. In this project, we are focusing on wheel odometry which mainly uses encoders and the encoders will be attached to three non-driven wheels in order to prevent wheel slippage that will affect the input data. There are certain rules about positioning these three wheels on the robot. Firstly, there should be a tracking center on the robot and all three tracking wheels should be perpendicular to the tracking center. Secondly, two of the wheels should be placed on the left and right sides of the robot and they must be parallel to each other. Thirdly, another tracking wheel should be placed on the back of the robot, perpendicular to the other two tracking wheels. As long as the three wheels fulfill these three rules, the robot designer can place them wherever they want.



Another part of the background information we might need is the mathematical formulas for calculating the orientation of the robot and the position of the robot.
Orientation Formula: θ is the angle the robot changed, ΔL and ΔR is the two data from the tracking wheels, and T_L and T_R are the distance between the wheel and the tracking center.

$$\theta = \frac{\Delta L - \Delta R}{T_l + T_R}$$

Position Formula: d is the vector of the position (x and y), ΔS is the data from the back-tracking wheel, and T_s is the distance between the tracking wheel and the tracking center.

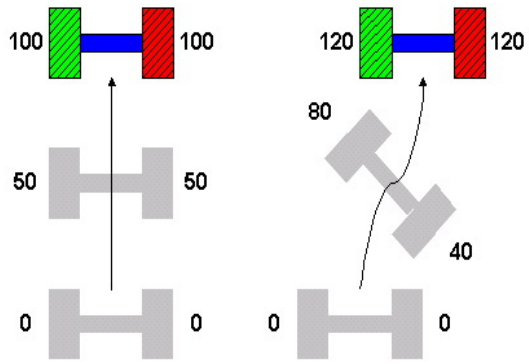
$$d = 2\sin(\frac{2}{\theta})\begin{bmatrix}\frac{\Delta S}{\theta} - T_s \\ \frac{\Delta R}{\theta} - T_R\end{bmatrix}$$

Literature Review (5 sources): *find five sources on your topic and summarize them (a paragraph each) (provide embedded links like this) (make them a list)*

1. SLAMcore Blog

Wheel odometry is a method of determining the location of a robot. Wheel odometers can provide data from the robot itself, rather than the surrounding environment. Using the wheel's rotation, direction and magnitude of movement can be obtained. However, wheel odometers have several limitations, such as the output of noisy data and other factors, including wear and tear, uneven terrain, and changes in weight of the robot, can also influence the accuracy of wheel odometers. Hence, wheel odometers are often used in conjunction with other sensors, such as LiDAR and cameras.

2. Slow Processing: Odometry



Another important aspect to consider when creating a wheel odometer is the processing speed. Slow encoder processing can cause inaccurate results as key movements that occur between time steps may pass unnoticed. For example, the image to the right shows two different paths and two different resulting locations. However, the data gathered by the for both paths has the equal left and right wheel positions. Hence, according to the data, both robots should have traversed a straight path, even though this is not the case. Slow processing errors will be taken into account in our final project.

3. Improving Odometer Accuracy for an Autonomous Electric Cart

Wheel odometers are useful sensors for position and direction estimation. However, accuracy tends to deteriorate as distance increases. Errors can build up and accumulate to provide vastly inaccurate location measures, especially when other factors like wear and tear or skidding are involved, but not accounted for. Interestingly, neural networks can also be used to minimize odometric errors. This can be achieved in multiple ways, including using neural networks to rectify errors generated in a simulation or estimating errors and adjusting for them. According to the paper, using neural network based optimization can be an effective tool to reduce error in data points.

4. Dead Wheels

Use of Dead wheel is a very unique solution to ensure accurate tracking.These dead wheels are attached to an encoder sensor. There are two different ways of putting the dead wheels: First Is Two dead wheels and the other is Three Dead Wheels. In the two dead wheels, there is one dead wheel placed parallel and the other one placed perpendicular with respect to the drive wheel axis. The latter one has two parallel and one perpendicular placement of dead wheels. Two dead wheels measure the x and y movements, however three dead wheels measure the heading(direction/bearing) by calculating the difference in two parallel wheels.

5. Encoders in Odometry

There are different kinds of Encoders that can be used in wheel odometry. Quadrature encoders use hall-effect sensors to measure magnetic pulses as the shaft turns, therefore it will be very precise as well as being able to tell the difference between forward and reverse rotation. If we are using Arduino to read the data, quadrature encoders are the best. The second type of encoders are Single-output encoders. They are similar to quadrature encoders but they can not determine the actual direction of motion. The third type is optical encoders, which normally have a Light sensor combo with a disk that has slots cut in-between them. As the disk rotates, the light is interrupted and those pulses are counted. Optical encoders are cheap and easy to install but cannot determine the direction of rotation and it is best to use RaspberryPi's GPIO pins.

Methodology:

The robot we built consisted of the following component:

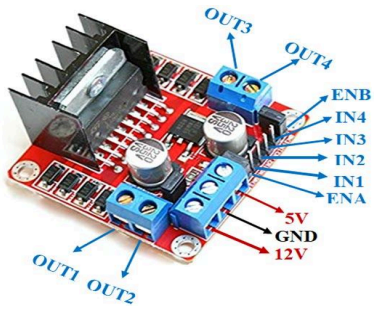
1. 4 wheels
2. 2 motors
3. L298N Motor Driver
4. Arduino Uno R3
5. Optical Encoder
6. Chassis

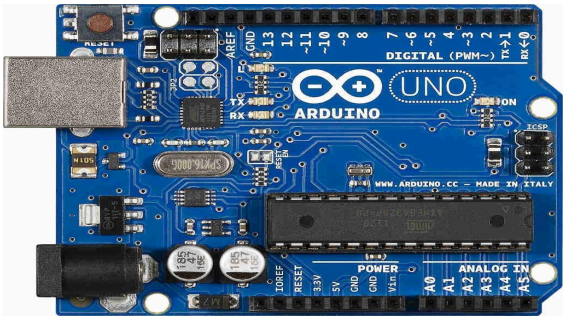
Physical Methodology:

1. First we paced the chassis, then used screws to connect the two motors(Both of Back Two wheels) and 2 encoders.

Connected the Motor Driver to Arduino

2. First we paced the chassis, then used screws to connect the two motors(Both of Back Two wheels) and 2 encoders.
3. Connected the Motor Driver to the two motors in its 4 Output terminals (2 Output terminals for 1 motor) as shown in the picture below.
    - Input 1,2,3,4 on the Motor Driver are connected to Digital PWM 2,3,4,5 on the Arduino Uno R3.





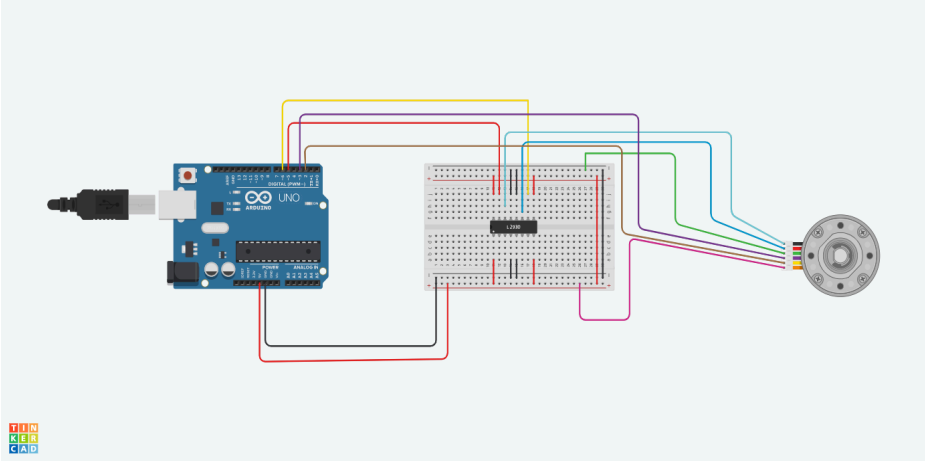
4. Now that everything is connected, a code was written on Arduino IDE software for:
    a. **Controlling the Motor**:
        - The code for this part written solely for controlling the speed of the motor and the direction
    b. **Reading the data from the encoder**:
        - Define a tick - Meaning every time the pin from the encoder goes on, that is a single tick also tells us the angle at which the encoder has turned.
        - Reading ticks
    c. **Sending to computer**:
        - First we set the Serial Port to 9600 tps.
        - For each second, the data is sent to the computer.

Simulator Methodology:

The parts we ordered did not arrive on time, hence we had to build a simulator to provide our results for the encoder. The components used on this simulator were not the same except the Arduino. In the simulator we used the following components:

1. Arduino Uno R3
2. Breadboard
3. L293D Motor Driver - The one mentioned before is L298N which is similar to this motor driver as both can handle two DC motors and consist of H-Bridge(simplest circuit for controlling low current motors)
4. DC Motor Embedded with a Magnetic Encoder. The robot uses Optical Encoder and the biggest difference between the two is that Magnetic Encoder looks at the number of ticks whereas the latter one looks at the number of captured dials. We had no other option than using the Magnetic encoder for our calculations.

On the simulator in TinkerCad, we used 1 arduino, 1 breadboard and 1 DC motor embedded with Magnetic Encoder. The picture below shows so.



1. Using the Arduino IDE software, we wrote three separate codes. One for Back Encoder, One for Right encoder and last one for Left Encoder.
2. We used one encoder code at a time to get results for each timestep. Then after running all the three codes, we put them in a table as mentioned in the results part.
3. The data from the simulator was then used to calculate the angle for Back, Left and Right Encoders. The angle for the back encoder will always be zero since the robot is moving in a straight direction.
    - The angles were calculated using this formula:
      Angle = ((Data at each time step for One Encoder)/ 4096) * 360 (degree)
        - 4096 is a discrete position for the encoder resolution of the 12-bit encoder we used.
        - Multiplying it by 360 degrees gives us the angle.

Division of Labor: *provide what role each person will do (1 paragraph for each member) (make them a list)*

Dheer

- Build the robot by connecting the arduino to the L298N motor driver (which is connected to 2 motors), wheel encoder sensors, and cells and gather the data.
- Gather and organize the data from Yash's TinkerCAD simulation by running Steven's arduino code in TinkerCAD.
- Run the appropriate calculations on the data gathered from the robot and present the data in an organized manner.

Yash

- Create a schematic of the arduino wired with the battery pack, L293D unit and wheel encoder
- Build the simulation on TinkerCAD using the arduino, magnetic encoder, and DC motor.
- Help Dheer build the robot.
- Designed how the robot should look on Solidworks and then figured out which robot kit is to be ordered.

Steven

- Write the Arduino IDE code for the wheel encoders and DC motors.

- Run part of the calculations on the data gathered from the simulation, by calculating the degrees the rotary encoder rotated per timestep.
- Help Dheer in building the robot

Results

Upon receiving delivery of the parts, we realized our encoder sensor was not functioning appropriately and the arduino stopped working when unplugged from the computer. Hence, we relied on the TinkerCAD simulator to gather the data. We researched how a magnetic encoder can be used to calculate position and orientation values.

Step 1 - Raw Data:

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Right	785	1200	1617	2033	2448	2863	3279	3693	4105	4516	4925	5333	5740	6143	6542
Left	784	1200	1618	2035	2451	2866	3281	3694	4108	4520	4930	5337	5741	6142	6537
Back	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

16	17	18	19	20	21	22	23	24	25	26	27	28	29
6935	7317	7700	8107	8549	8993	9435	9877	10318	10759	11200	11640	12079	12516
6923	7229	7623	8043	8486	8928	9369	9811	10252	10693	11133	11573	12011	12428
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Using the formula, Angle = ((Data at each time step from an Encoder)/ 4096) * 360 (degree), we were able to convert this data into the number of degrees the encoder rotated per timestep.

Step 2 - Degrees Rotated:

Time Step	1	2	3	4	5	6	7	8
Right	68.9	105.5	142.1	178.7	215.2	251.6	288.2	324.6
Left	68.9	105.5	142.1	178.8	215.4	251.8	288.3	324.6
Back	0	0	0	0	0	0	0	0
Time Step	9	10	11	12	13	14	15	
Right	360.7	396.9	432.9	468.7	504.5	540.3	575	
Left	361.1	397.2	433.3	469.1	504.6	540.3	574.5	
Back	0	0	0	0	0	0	0	
Time Step	16	17	18	19	20	21	22	
Right	609.5	643.1	676.8	712.5	751.4	790.4	829.2	
Left	609.3	635.4	670	707	745.8	784.7	823.4	
Back	0	0	0	0	0	0	0	
Time Step	23	24	25	26	27	28	29	
Right	868.1	906.9	945.6	984.4	1023	1061.6	1100.3	
Left	862.3	901.1	939.8	978.5	1017.2	1055.7	1092.2	
Back	0	0	0	0	0	0	0	

Using the above table, we were able to find the change in the degrees between two time steps.

Step 3 - ΔDegrees:

Time Step	1	2	3	4	5	6	7	8
Δright	68.9	36.6	36.6	36.6	36.5	36.4	36.6	36.4
Δleft	68.9	36.6	36.6	36.7	36.6	36.4	36.5	36.3
Δback	0	0	0	0	0	0	0	0
Time Step	9	10	11	12	13	14	15	
Δright	36.1	36.2	36	35.8	35.8	35.8	34.7	
Δleft	36.5	36.1	36.1	35.8	35.5	35.7	34.2	
Δback	0	0	0	0	0	0	0	
Time Step	16	17	18	19	20	21	22	
Δright	34.5	33.6	33.7	35.7	38.9	39	38.8	
Δleft	34.8	26.1	34.6	37	38.8	38.9	38.7	
Δback	0	0	0	0	0	0	0	
Time Step	23	24	25	26	27	28	29	
Δright	38.9	38.8	38.7	38.8	38.6	38.6	38.7	
Δleft	38.9	38.8	38.7	38.7	38.7	38.5	36.5	
Δback	0	0	0	0	0	0	0	

Step 4 - ΔR, ΔL, ΔB:

We calculated these values by dividing the values at each timestep by 360 and multiplying it by 1^2 = 1. We estimated the simulator's encoder's radius to be 1 cm as most encoders available online have a radius of 1 cm.

Time Step	1	2	3	4	5	6	7	8
ΔR	0.60126593	0.31939525	0.31939525	0.31939525	0.31852259	0.31764992	0.31939525	0.31764992
ΔL	0.60126593	0.31939525	0.31939525	0.32026792	0.31939525	0.31764992	0.31852259	0.31677726
ΔB	0	0	0	0	0	0	0	0
Time Step	9	10	11	12	13	14	15	
ΔR	0.31503193	0.31590459	0.31415927	0.31241394	0.31241394	0.31241394	0.30281463	
ΔL	0.31852259	0.31503193	0.31503193	0.31241394	0.30979594	0.31154127	0.2984513	
ΔB	0	0	0	0	0	0	0	
Time Step	16	17	18	19	20	21	22	
ΔR	0.3010693	0.29321531	0.29408798	0.31154127	0.33946654	0.3403392	0.33859387	
ΔL	0.30368729	0.22776547	0.30194196	0.32288591	0.33859387	0.33946654	0.33772121	
ΔB	0	0	0	0	0	0	0	
Time Step	23	24	25	26	27	28	29	
ΔR	0.33946654	0.33859387	0.33772121	0.33859387	0.33684855	0.33684855	0.33772121	
ΔL	0.33946654	0.33859387	0.33772121	0.33772121	0.33772121	0.33597588	0.31852259	
ΔB	0	0	0	0	0	0	0	

Step 5 - Position and Orientation Values:

```
array([[ 0.01      ,  0.02      ,  0.03      ,  0.04      ,  0.05      ,
         0.06      ,  0.07      ,  0.08      ,  0.09      ,  0.1       ,
         0.11      ,  0.12      ,  0.13      ,  0.14      ,  0.15      ,
         0.16      ,  0.17      ,  0.18      ,  0.19      ,  0.2       ,
         0.21      ,  0.22      ,  0.23      ,  0.24      ,  0.25      ,
         0.26      ,  0.27      ],
       [ 0.        ,  0.        ,  0.        ,  0.00043633,  0.00087266,
         0.00087266,  0.00043633,  0.        ,  0.00174533,  0.001309  ,
         0.00174533,  0.00174533,  0.00043633,  0.00479965,  0.00349066,
         0.00872664, -0.0244331 , -0.02923275, -0.03752304, -0.03839571,
        -0.03795937, -0.03883204, -0.03795937, -0.03795937, -0.03752304,
        -0.03708671, -0.03708671],
       [ 0.        ,  0.        ,  0.        ,  0.01437258,  0.02870708,
         0.02870708,  0.0143345 ,  0.00003808,  0.05676684,  0.04254657,
         0.05669069,  0.05669069,  0.01448681,  0.15097783,  0.11025895,
         0.26902194, -0.73932578, -0.89365436, -1.18337227, -1.21394513,
        -1.19873485, -1.22923156, -1.19888716, -1.19888716, -1.18375304,
        -1.16861892, -1.16861892]])
```

Using the same code we developed in Problem 5 of Problem Set 2, we calculated the position and orientation values. We faced one issue - as our robot followed a straight line path, the orientation angle values were always zero. Hence, we couldn't use the formulae to calculate the position values initially. However, after adding a slight noise of 0.01 to the orientation angles, we were able to calculate the position values too. In the image above, the first row of the matrix represents the orientation angle values, the second row represents the x values and the third row represents y values.

Conclusion:

This project had its ups and downs but we still managed to give our best and have given the results and calculations. The library of components on Tinkercad only had a magnetic encoder , so the crucial part, which were the calculations, could not be completed. The test track could only be a straight line since the motor was not moving after uploading the code on arduino. We were sure of the right code written but it's always expected, especially in hardwares, for these types of problems to come. Since the robot moved in a straight line, the left and right encoder had similar values and the back encoder always had a value of zero.

We could not compare the attained position and orientation of the robot against the true values due to the parts arriving late and one of the sensors being faulty, preventing us from building the robot, testing it on the track, and rectifying problems with components. Our only option left was the simulator to work and it eventually worked where we gathered data for back, left and right encoders and performed the calculations for Δ R, L and B.

Next Steps:

There are several places that we can improve on in the future for this project, which are based on the limitations that we faced this time. Firstly, the greatest wish and the improvement we can make is to use functioning sensors to gather data using the robot rather than the simulation. Since the three of us are in different places, and this is a hardware project, most of our groupmates didn't get a chance to build a physical robot. So, in the future, we might use the odometry system on different vehicles and larger scales like driverless cars, autonomous delivery robots, etc. Also, this odometry system can be used for different robotics competitions like FRC and VEX. Another potential "next step" for us is to improve the codes. For now, our codes are still facing several errors and areas for improvement. For example, the raw data we get from the Arduino is in "ticks" which is not a usable unit for the calculations, leading to further processing and more steps being added to the process, so maybe in the future, we will be able to write the code that directly converts the raw data to the numbers that we can use for our calculation.