

# Differential Drive

Falon Scheers  
Tristan Meleshko

# Introduction

The purpose of this lab is to design a working differential drive vehicle and a Braitenberg vehicle using the Lego Mindstorms EV3 kit. Also in the process to notice and investigate the error accumulation that is commonly found when dealing with common hardware.

## Visual error detection with drawing

We implemented code to allow the robot to drive in one of four configurations: a straight line, a rectangle, a circle, and a figure eight. For visual error detection we attached a marker to the pivot point of the robot to trace its journey as it drove.


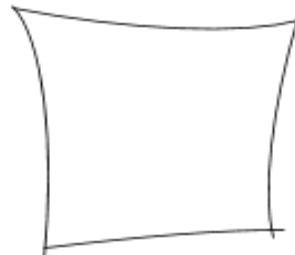
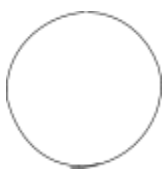
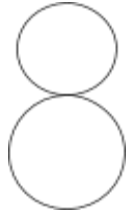
Tracing shape	Observations after 3 or more trials
Straight line 	our robot veers to the left from (0, 10) degrees after 3 wheel rotations (this is after our attempt to correct for motor power differences)
Rectangle(square in our code) 	Due to our inability to drive in a straight line, paired with a slight delay when the motors turn off after turning... since our code has the robot making 4 right turns, the veering left and turning a little more than 90 degrees each time cancel each other out and a completely connected square is drawn. However, this also means that each side edge is not straight, rather it is more concave towards the center slightly.
circle 	Does pretty well, but sometimes doesn't exactly connect the circle and is a little off to one side. Usually overlaps a little from the starting point. This makes sense because our turns are slightly overshoot each time due to delay in motors getting reduced power.

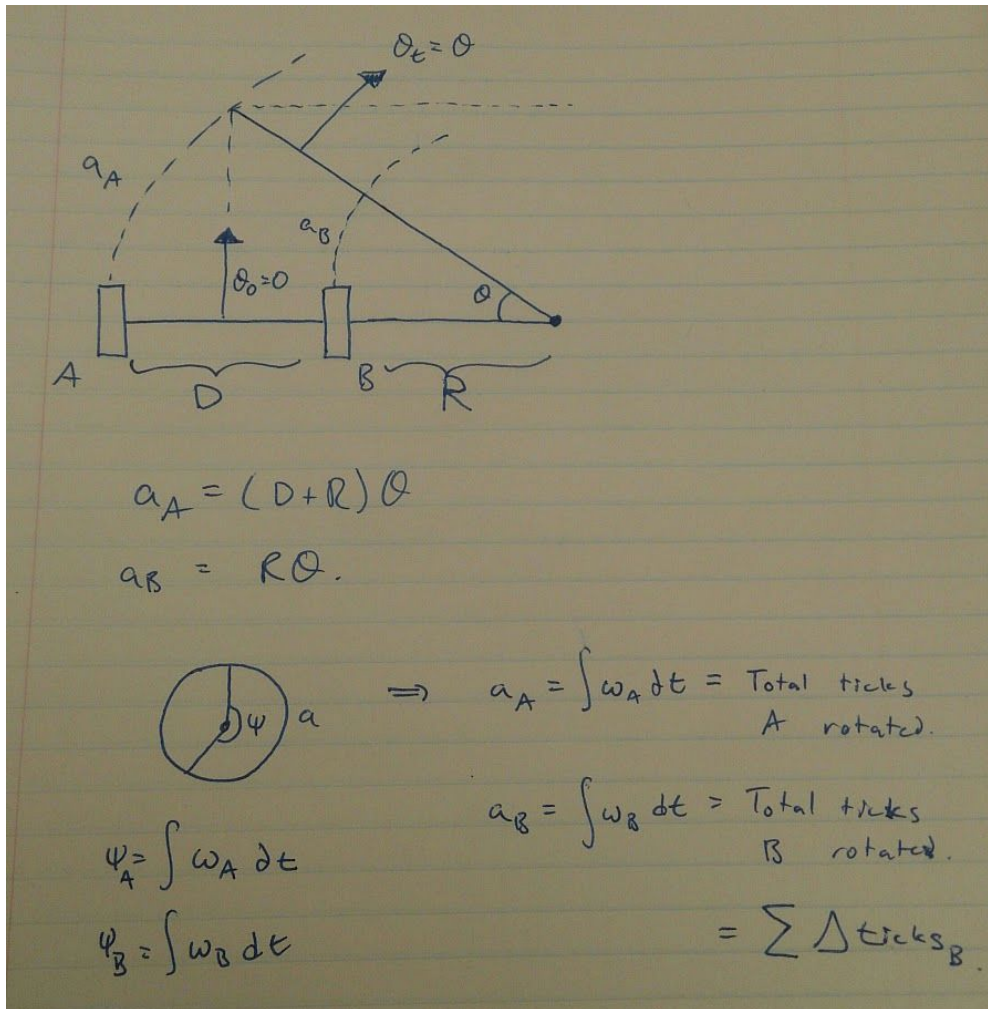
figure eight



Our first circle is larger than our second, much like an '8'. This is due to our left motor being faster than our right motor... so the turning radius for turning clockwise is larger than the radius for turning counterclockwise slightly.

## Dead Reckoning

By measuring local changes in the robot's position, we are able to make an estimate about the final, global position of the robot. Specifically, we used the changes in the tachometer reading for each wheel to compute its final (x, y)-distance relative to its start position as well as its final heading at any point during the program with some margin of error.



$$a_A = (D+R)\theta = \int \omega_A(t)dt = \phi_A r$$

$$a_B = R\theta = \int \omega_B(t)dt = \phi_B r$$

$$R = \phi_B r / \theta$$

$$\phi_A r = (D+R)\theta = (D + \phi_B r / \theta)\theta = D\theta + \phi_B r$$

$$\theta = \phi_A r - \phi_B r / D = ((\phi_A - \phi_B) / 2) \cdot (d / D)$$

Where d is the wheel diameter, D is the axis diameter, a is the arc length, and R is the distance to the ICC. This gives the final heading assuming that it remains turning in the same direction at the same speed the whole journey. To remove that assumption, we can take the instantaneous angle change and integrate these values, giving:

$$\Delta\theta = ((\Delta\phi_A - \Delta\phi_B) / 2) \cdot (d / D)$$

Assuming the robot travels in a straight line for a short enough period, we can use these to compute the change in position:

$$\Delta x = \Delta distance \cos(\theta)$$

$$\Delta y = \Delta distance \sin(\theta)$$

$$\Delta distance = (\Delta\phi_A + \Delta\phi_B) / 2 \cdot (d \pi / 360)$$

Due to the slow processing speed, it was necessary to compute these distance as late and as often as possible, including an extra time after the loop and especially after the motors stopped.

## Error Accumulation Measure

In order to measure error accumulation with linear movements for our robot we used two computed methods to compare with the actual distance travelled by the robot in a straight line. the first method we used was to calculate the math using the tacho count for each motor the distance travelled versus the manually measured distance. This also helped us to verify the difference in motor speeds when both are supposed to be operating at the same speed. The second measure of comparison we used against the manually calculated, true distance made use of the ultrasonic sensor mounted on the front of our robot. When driving straight towards a wall the sensor takes an ultrasonic reading of the initial distance to the wall and then after the robot has stopped moving and we calculated the difference between the two readings. Please see picture 1.0 (below) for a visual for how we mounted the ultrasonic sensor. All of these tests were done with motor speed of 80%.

Trial #	Distance calculated from Tacho	<b>Actual</b> Distance	Distance calculated
---------	--------------------------------	------------------------	---------------------

	count (cm)	measured by hand(cm)	by US sensor (cm)
1	motorA: 19.74    motorB: 20.4	18	18.7
2	motorA: 54.93    motorB: 56.59	53	52.7
3	motorA: 54.73    motorB: 56.59	53	53.1

The error in the prediction using the tachometers is slightly above the ground truth (measured with a ruler) and the ultrasonic sensor data. However, the ultrasonic sensor gave the more accurate result, most likely because the error in the tachometers is repeatedly summed and accumulated, whereas the ultrasonic sensor can give an absolute value for the distance traveled. Any error in the US sensor is due to a change in heading over time, whereas the motors have multiple error sources that are repeatedly accumulated.



picture 1.0: Robot with Ultrasonic and light sensors mounted.

In order to measure error accumulation when our robot is rotating we used two computed methods to compare with the actual angle of reference the robot is after a rotation. the first method we used was to calculate the math using calculus and the tacho count for each motor the distance and angle travelled versus the manually measured angle. for this we adapted our Dead Reckoning code to take specific test cases that measured left and right turns at various speeds. The second measure of comparison we used against the manually calculated, true angle of rotation made use of the gyro sensor mounted as close to the pivot point of our robot between the wheels as possible. The gyro sensor is zeroed before any movement and then the

measure is taken again after the movement and compared to the actual hand rotation measured with a protractor. The data for an average of 3 trials of each test is shown in the table below in degrees.

test type and coords used	Actual angle avg.	Dead Reckoning calculated avg.	Gyro sensor reported avg.
fast left turn { 80, 0, 3}	-24 degrees	-14 degrees	-13 degrees
fast right turn { 80, 0, 3}	-8 degrees	-10.43 degrees	+3 degrees
slow right turn { 40, 0, 4}	-234 degrees	-221 degrees	-226 degrees
slow left turn { 0, 40, 4}	223 degrees	216 degrees	215 degrees

Additional error may have been accumulated from human error in measuring the angle with a protractor and “eye-balling” the robot’s turn. In general there is more error in turning left because motor B on the the left has already found to be slower. surprisingly, the gyro sensor doesn’t appear to be significantly more accurate than our dead reckoning calculations. This is likely due to our hardware errors and also not being able to mount the gyrosensor in the exact pivot point. There is more error when going slower as seen from the data.

## Braitenberg Vehicle

Our sensors were not able to read a value of 0 due to the ambient light. We included a calibration parameter that maps the ambient light to 0 and the brightest value to 1.0. However, due to the variances in the light level in the room, these calibrated values were then allowed to flexibly adjust to the new lowest and highest value visible in order to prevent an overflow or underflow of values. The value is then fed directly to the motor with minimal post processing as the power level. Doing this, we were able to implement the required behaviours.

Our implementation isn’t a true Braitenberg vehicle because it performs some processing on the input data. However, we found that it performed more predictably and consistently with these calibrations. Please see picture 1 (above) to visually see how the light sensors were mounted.

# Teleoperation

Another way one might want to control a robot is using a remote control. We implemented a remote control over Bluetooth. The program is operated strictly with the keyboard:

W/S = move forward/backward

A/D = turn left/right

Q/E = pivot left/right

Page Up/Page Down = Increase/Decrease motor speed

Please see code comments for a more detailed description of implementation.