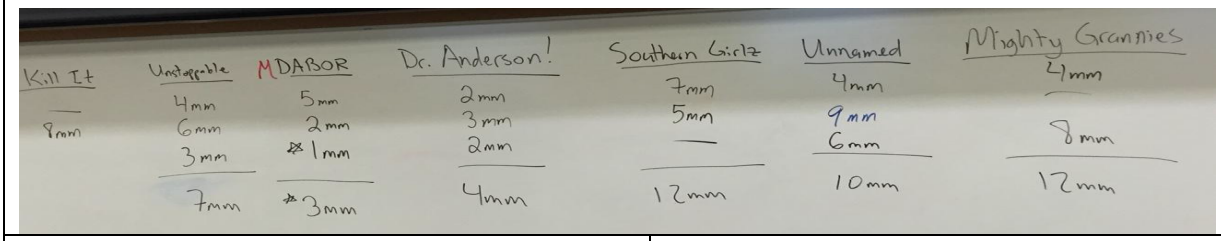
Appendix A
EV3 Robotics Challenges

Part 1 EV3 Robotics Challenges

<table>
<tr>
<td>Challenge 1: Open-Loop Movement.
With any experiment, there is an issue that all scientists and engineers run into - hysteresis. For example, a self-driving car can never assume that the same set of commands will always lead to the same results. Indeed, one of the first lessons for any team, from middle school robot teams to national science laboratory engineering teams, is to never assume the experiment will end the same way. Open loop solutions often suffer from hysteresis (an output change based on the history of the experiment) while closed loop solutions try to overcome this.</td>
<td>Rules. This challenge is to see how close a team can get to the far wall of the arena without touching the wall. The rules are as follows:

1. The robot must start touching the West wall.
2. The robot that gets closest to the East wall - without touching - is the winner. A robot that stops while touching the East wall will be penalized 8mm.
3. Each team will get three runs. The team with the smallest sum distance will be declared the winner.
4. You can use any robot design but <u>no</u> external sensors (e.g. ultrasonic or light) may be used.</td>
</tr>
<tr>
<td colspan="2">Results. The purpose of this challenge is to get a benchmark for the next challenge's results. Open loop systems will perform worse than closed loop systems and teams recorded their best performing distance to compare with the next challenge.</td>
</tr>
<tr>
<td>Challenge 2: Closed-Loop Movement.
Sensors are everywhere. And though they are a popular literary device for dystopian storylines, electronic sensors are just the method used to allow computers access to the real world. Without sensors you'd take the "smart" out of modern technology. Imagine a phone without a camera, microphone, speaker, or wi-fi (all sensors). In this challenge teams demonstrate the use of sensors as it emulates a version of some cutting-edge technology - automatic car breaking.</td>
<td>Rules. The Closed Loop challenge rules are exactly the same as the Open Loop challenge except for one important change to Rule #4:

1. The robot must start touching the West wall.
2. The robot that gets closest to the East wall - without touching - is the winner. A robot that stops while touching the East wall will be penalized 5mm.
3. Each team will get three runs. The team with the smallest sum distance, after the worst score is discarded, will be declared the winner.
4. You can use any robot design and <u>ANY</u> external sensors (e.g. ultrasonic or light).</td>
</tr>
<tr>
<td colspan="2">Results. To get teams excited and into the "spirit" of the challenge, results were displayed real-time on the board (below) with teams attempting to beat the professor and scores from Challenge 1. Note that this was successful since all teams beat their previous scores with the use of sensors (and one team was able to beat the professor).

 
</td>
</tr>
<tr>
<td>Challenge 3: The Robotic Ocean Sweep.
Robots, coding, and machine learning are used everywhere and not just in self-driving cars. This challenge emphasized that it is actually not that hard to code a robot to autonomously cover some area such as in popular vacuum robots. For context, oil, and other spills are a constant worry in oceanography. In the Gulf of Mexico oil spill of 2010 over three million barrels of oil leaked out before the well was capped. One method of cleaning up these spills is to use swimming robots. In this challenge teams design and code a robot to "clean up" the</td>
<td>Rules. After blocks are randomly scattered throughout the arena, teams must obey the following rules.
1. The robot can start anywhere in the world that you want.
2. The robot scores points by removing objects from the world's oceans. Green blocks are toxic garbage and worth 5 points while red blocks are normal trash and worth 1 point.
3. To count for your score, the block must be completely off the world map. The score for each round is the sum objects times their worth. Each round lasts at most 30 seconds or until the bot says "Stop!".
4. Since form factor is important with swarmbots, your score will also be divided by the maximum width of your bot.</td>
</tr>
</table>

world's oceans.

5. Each team will get three runs. The team with the highest total score will be declared the winner.
6. You can use any robot design but make sure you include some kind of plow!

Results. Since teams were given total leeway on how to solve the problem of block (trash) collection, the solutions were varied in their presentations. Some teams used what they learned in the open and closed loop challenges to do an exhaustive sweep of the arena in order to get all blocks. Other teams were more hesitant and simply did a single-shot pass at cleaning. Finally, one team did minimal coding but invested their time into a mechanical solution with a large plow used to gather as many blocks as possible. Sample Python coding used to solve the challenge included below. Note the clever use of different motor powers to help blocks stay in the plow as they're pushed off the map.

```
from ev3 import *
from time import sleep

#connect the motors
motorLeft = LargeMotor('outB')
motorRight = LargeMotor('outC')

#set the run time for each motor to 5 sec
motorLeft.time_sp = 5000
motorRight.time_sp = 5000

#set the power level for each motor
#Note Right has more power to the robot will turn left
motorLeft.duty_cycle_sp = 75
motorRight.duty_cycle_sp = 100

#run the motors
motorLeft.run_timed()
motorRight.run_timed()

#wait for the robot to stop
sleep(5)
```

Challenge 4: Line Follower. A line follower robot is able to follow some arbitrary track without any physical connection with the track except for the wheels. This method has great advantages over something like tracks or rails since the cost of laying down a new course is just the cost of paint. Line follower robots have many commercial applications such as autonomous cars on roads or forklifts in large factories. One of the main math concepts behind a popular line following algorithm is something called a PID controller. Though this controller can sound complex, the "I" stands for integral and the "D" stands derivative, we simplified the challenge to making a P controller ("P" stands for proportional).

Rules. This challenge was a double elimination tournament. Robots competed head-to-head on equal length tracks. The first robot to cross the finish line was declared the winner for that round. A couple rules for this challenge:

1. Any robot design can be used; however, you are only allowed to use one light sensor.
2. If your robot leaves the track, the judge will signal the team and they are allowed to place the robot back on the track from the point of its departure.

Results. The teachers learned that in order to keep the light sensor centered on the line, the sensor must read 50% at all times. If the sensor read greater or less than this value, the motors' powers were corrected to force the robot back onto the line

$p_r = m_r s + b_r$

$p_l = m_l s + b_l$

where the teams were required to find correct constants so the robot functioned correctly. Teams quickly learned that incorrect values would cause the robot to oscillate wildly as it attempted to follow the line. Example code generated by one of the teams is included. Note how the team heuristically determined proportionality constants for the motor power functions so the resulting behavior would be more stable.

```
from ev3 import *
from nav import *

left = LargeMotor('outB')
right = LargeMotor('outC')

lsensor = ColorSensor('in3')
lsensor.mode = 'COL-REFLECT'

def rmotor(x):
    return x*0.58824 +20.09804

def lmotor(x):
    return x* (-0.58824) + 66.569

left.run_direct()
right.run_direct()

while True:
    print(lsensor.value())

    right.duty_cycle_sp= rmotor(lsensor.value())
    left.duty_cycle_sp= lmotor(lsensor.value())
```

Part 2

Robotic Sweep Python Code

```
from ev3 import *
from time import sleep

#connect the motors
motorLeft = LargeMotor('outB')
motorRight = LargeMotor('outC')

#set the run time for each motor to 5 sec
motorLeft.time_sp = 5000
motorRight.time_sp = 5000

#set the power level for each motor
#Note Right has more power to the robot will turn left
motorLeft.duty_cycle_sp = 75
motorRight.duty_cycle_sp = 100

#run the motors
motorLeft.run_timed()
motorRight.run_timed()

#wait for the robot to stop
sleep(5)
```

Part 3

Robotic Sweep READ Me Information

This program is designed for a wheeled robot that has two wheels, one connected to output ‘outB’ and the other connected to output ‘outC’. Both motors are set to run for the same amount of time. However the amount of power supplied to the right wheel is greater, resulting in a left-hand turn. The robot is allowed to roll to a stop.

Part 4

Line Follower Python Program Code

```
from ev3 import *
from nav import *

left = LargeMotor('outB')
right = LargeMotor('outC')

lsensor = ColorSensor('in3')
lsensor.mode = 'COL-REFLECT'

def rmotor(x):
   return x*0.58824 +20.09804

def lmotor(x):
   return x* (-0.58824) + 66.569

blue = 14
white = 65
midpoint = (blue + white)/2.0

left.run_direct()
right.run_direct()

while True:
   print(lsensor.value())

   right.duty_cycle_sp= rmotor(lsensor.value())
   left.duty_cycle_sp= lmotor(lsensor.value())
```

Part 5

Line Follower program for EV3 robot READ Me Information

This program is designed for a wheeled robot with motors connected to outputs ‘outB’ and ‘outC’. There is also a color sensor connected to input ‘in3’. The program implements a very basic line follower. The robot drives forward using the color sensor to detect the left edge of blue painter’s tape on a white floor. The basic rule is that if the sensor is detecting more blue, then the robot should steer to the left. If the sensor is detecting more white, then the robot should steer to the right. The parameters in the code would need to be adjusted depending on the colors of the floor and tape.

The code is included here with explanatory comments added.

```
#A basic line follower program

#import libraries
from ev3 import *
from nav import *

#connect motors
left = LargeMotor('outB')
right = LargeMotor('outC')

#connect color sensor
lsensor = ColorSensor('in3')
lsensor.mode = 'COL-REFLECT'

#Determine the power for the right wheel as a function of the color sensed
def rmotor(x):
   return x*0.58824 +20.09804

#Determine the power for the left wheel as a function of the color sensed
def lmotor(x):
   return x* (-0.58824) + 66.569

#initialize the colors for the floor and tape.
blue = 14
white = 65
midpoint = (blue + white)/2.0

#Turn on the motors
left.run_direct()
right.run_direct()
```

```
#Run forever
while True:
   print(lsensor.value())

   #Balance the power to each wheel to stay centered on the line.
   right.duty_cycle_sp= rmotor(lsensor.value())
   left.duty_cycle_sp= lmotor(lsensor.value())
```