ME461 Laboratory #6 (Two Week Lab)
DC Motor Speed Control and Steering a Three Wheel Robot Car

Goals:

1. Understand the use of the eQEP peripheral of the TMS320F28379D processor to use the DC motor's magnetic encoder to sense the angle of the DC motor.
2. Calculate the velocity of your robot car using the eQEP's angle measurements.
3. Implement decoupled PI speed controllers to control the speed of each DC motor.
4. Implement a coupled PI speed controller that allows for steering and forward/backward driving.
5. As the robot is driven, calculate and keep track of the Pose (x,y,theta) of the robot car. This assumes no wheel slippage. Using the fact that the floor tiles are 1 foot square, make a judgement of how accurate this Dead Reckoning method is for knowing the robot's Pose. "Tune" the radius of the wheel and distance between the wheels to make your measurement more accurate if possible.

Exercise 1:

For this lab, I recommend you start with your Lab 5 code that reads the MPU-9250 every 1 millisecond. The MPU-9250 sensor readings will not be needed in this lab but keeping them in your code will help you in Lab 7 when you are asked to use the IMU feedback to balance the Segbot. So I would go ahead and create a new "labstarter" project and call it "lab6" or whatever you like. Then simply copy all the text in your Lab 5 code file and paste it over all the code in your lab6 source file. As a sanity check, I would compile and run this project and make sure your IMU values are still printed correctly to Tera Term.

The eQEP peripheral, right out of a "power on reset" of the F28379D processor, is pretty much ready to count the A and B channels of an encoder angle sensor. For that reason I am giving you the code for initializing and reading the angle values. There are many advanced features of the eQEP module and a number I have not played with yet. If this sounds interesting to you, you could turn playing with the advanced features of the eQEP into a part of your final project for this class. The only thing you need to add to the below code is a scale factor that converts the eQEP count value to a radian value.

Look at your robot's motors. Notice that there is a gear head between the motor and the wheel's shaft. This gear ratio is 30:1, so 30 rotations of the DC motor cause one rotation of the wheel. Also look at the back end of your motor and find the magnet wheel. There are 5 North/South magnetic poles in that small wheel. The Hall Effect sensors on the circuit board sense each of those poles as they pass by. So one rotation of the motor creates 5 square wave periods per rotation for both the A and B channels. Since the eQEP counts these pulses using the quadrature count mode, the total number of counts per revolution of the motor becomes 4 X 5 or 20 counts per revolution. *See my current lectures if you are not familiar with the A and B channels and quadrature count mode.* Then combining this with the gear

ratio of the motor, you can calculate the multiplication factor that converts eQEP counts to the number of radians the wheel has turned. Add this to both the ReadEncLeft() and ReadEncRight() functions so that they return radians of the wheel. With this multiplication factor added, cut and paste the below code into your C file. Make sure to create predefinitions of these three files.

```
void init_eQEPs(void) {

   // setup eQEP1 pins for input
   EALLOW;
   //Disable internal pull-up for the selected output pins for reduced power consumption
   GpioCtrlRegs.GPAPUD.bit.GPIO20 = 1;    // Disable pull-up on GPIO20 (EQEP1A)
   GpioCtrlRegs.GPAPUD.bit.GPIO21 = 1;    // Disable pull-up on GPIO21 (EQEP1B)
   GpioCtrlRegs.GPAQSEL2.bit.GPIO20 = 2;   // Qual every 6 samples
   GpioCtrlRegs.GPAQSEL2.bit.GPIO21 = 2;   // Qual every 6 samples
   EDIS;
   // This specifies which of the possible GPIO pins will be EQEP1 functional pins.
   // Comment out other unwanted lines.
   GPIO_SetupPinMux(20, GPIO_MUX_CPU1, 1);
   GPIO_SetupPinMux(21, GPIO_MUX_CPU1, 1);
   EQep1Regs.QEPCTL.bit.QPEN = 0;    // make sure eqep in reset
   EQep1Regs.QDECCTL.bit.QSRC = 0;   // Quadrature count mode
   EQep1Regs.QPOSCTL.all = 0x0;      // Disable eQep Position Compare
   EQep1Regs.QCAPCTL.all = 0x0;      // Disable eQep Capture
   EQep1Regs.QEINT.all = 0x0;        // Disable all eQep interrupts
   EQep1Regs.QPOSMAX = 0xFFFFFFFF;   // use full range of the 32 bit count
   EQep1Regs.QEPCTL.bit.FREE_SOFT = 2;  // EQep uneffected by emulation suspend in Code Composer
   EQep1Regs.QPOSCNT = 0;
   EQep1Regs.QEPCTL.bit.QPEN = 1;    // Enable EQep


   // setup QEP2 pins for input
   EALLOW;
   //Disable internal pull-up for the selected output pinsfor reduced power consumption
   GpioCtrlRegs.GPBPUD.bit.GPIO54 = 1;    // Disable pull-up on GPIO54 (EQEP2A)
   GpioCtrlRegs.GPBPUD.bit.GPIO55 = 1;    // Disable pull-up on GPIO55 (EQEP2B)
   GpioCtrlRegs.GPBQSEL2.bit.GPIO54 = 2;   // Qual every 6 samples
   GpioCtrlRegs.GPBQSEL2.bit.GPIO55 = 2;   // Qual every 6 samples
   EDIS;
   GPIO_SetupPinMux(54, GPIO_MUX_CPU1, 5); // set GPIO54 and eQep2A
   GPIO_SetupPinMux(55, GPIO_MUX_CPU1, 5); // set GPIO54 and eQep2B
   EQep2Regs.QEPCTL.bit.QPEN = 0;   // make sure qep reset
   EQep2Regs.QDECCTL.bit.QSRC = 0;  // Quadrature count mode
   EQep2Regs.QPOSCTL.all = 0x0;     // Disable eQep Position Compare
   EQep2Regs.QCAPCTL.all = 0x0;     // Disable eQep Capture
```

```
   EQep2Regs.QEINT.all = 0x0;      // Disable all eQep interrupts
   EQep2Regs.QPOSMAX = 0xFFFFFFFF;  // use full range of the 32 bit count.
   EQep2Regs.QEPCTL.bit.FREE_SOFT = 2;  // EQep uneffected by emulation suspend
   EQep2Regs.QPOSCNT = 0;
   EQep2Regs.QEPCTL.bit.QPEN = 1;   // Enable EQep
}

float readEncLeft(void) {
   int32_t raw = 0;
   uint32_t QEP_maxvalue = 0xFFFFFFFFU;  //4294967295U

   raw = EQep1Regs.QPOSCNT;
   if (raw >= QEP_maxvalue/2) raw -= QEP_maxvalue;  // I don't think this is needed and never true

 // 5 North South magnet poles in the encoder disk so 5 square waves per one revolution of the
// DC motor's back shaft.  Then Quadrature Decoder mode multiplies this by 4 so 20 counts per one rev
// of the DC motor's back shaft.  Then the gear motor's gear ratio is 30:1.
   return (raw*(?????));
}

float readEncRight(void) {

   int32_t raw = 0;
   uint32_t QEP_maxvalue = 0xFFFFFFFFU;  //4294967295U  -1 32bit signed int

   raw = EQep2Regs.QPOSCNT;
   if (raw >= QEP_maxvalue/2) raw -= QEP_maxvalue;  // I don't think this is needed and never true

 // 5 North South magnet poles in the encoder disk so 5 square waves per one revolution of the
// DC motor's back shaft.  Then Quadrature Decoder mode multiplies this by 4 so 20 counts per one rev
// of the DC motor's back shaft.  Then the gear motor's gear ratio is 30:1.
   return (raw*((?????);
}
```

Call init_eQEPs() inside main() somewhere after the init_serial() function call but before the EINT line of code and the while(1) loop. Then set one of the unused CPU timer interrupts to timeout every 4 milliseconds. Inside that CPU timer interrupt function, simply call the two read functions and assign their return values to float variables like “LeftWheel” and “RightWheel”. Your existing code should be setting UARTPrint to have the main() while loop print your IMU values. Instead of printing the IMU readings, print your two wheel angle measurements. Make sure to print text indicating the left and right wheel. Make sure your motor ON/OFF switch is switched to OFF and then build and run this code. *(Make sure your encoder cables are connected to the LaunchPad.)* With your code running manually move your robot’s wheels. As a check, try to rotate one of the wheels just one turn. You should see an angle close to 2π. If not, you have the wrong multiplication factor in your read functions. Defining that the front of the robot car is the wheels and the back of the robot is the caster, does the labeling of left wheel and right wheel make sense? Forward speed will be defined as the front of the robot going forward. As you rotate your wheels you should see that if you rotate both motors in the forward

direction one will give a negative angle. Negate the multiplication factor in that wheel's read function so that both wheels read a positive angle when rotated in the forward direction.

In the next exercise you will calculate the speed at which your wheels are turning. In Lab 7, the control law will use the speed of the wheels in units of rad/s. This is the reason the read functions return radians. In this lab though, it will be nice to control the speed of the robot car in units of ft/s or tiles/s. Each of the tiles in the lab room are 1 foot by 1 foot and if we command the robot to move at 1 ft/s you will be able to use the tile divisions to approximately check if it is really running at that speed. Instead of using the diameter of the wheel, another easy way to convert between radians and feet is to simply put the robot on the floor and move the robot one foot without letting the wheels slip and that will tell you how many radians the wheel turns to reach one foot. Either put the robot on the floor or using some masking tape, measure 1 foot on your bench top. Line up your robot with the tape and push it forward 1 foot. Look at the radian values displayed in Tera Term to find the number of radians per foot. Create two more float variables and store the distance traveled by each wheel using this factor. Print these distances to Tera Term to check they are correct. **Does this factor make sense? It should be equal to the radius of the wheel in feet. Show your TA.**

Exercise 2:

For this exercise you will need to retrieve the functions you created in Lab 3 that commanded ePWM2A and ePWM2B with a command between -10 and 10. Copy these functions into your C file and create predefinitions at the top of your file. Also do not forget to set the "pinmux" for the PWM pins in main() along with the EPWM2 initializations for a 20Khz carrier frequency as you did in lab 3. Create two float variables "uLeft" and "uRight" that will be used as your control effort variables. For now just assign both of these to 5.0 when you create them as global variables. *Note: Since you will be spinning the wheels in the remainder of this lab make sure to put your robot on a piece of wood or box so that it does not drive off the bench top!!*

Inside the every 4ms CPU timer interrupt function that you setup in Exercise 1, calculate the left and right velocities that would be generated by the wheels assuming no slipping. Find these velocities in units of feet per second. To calculate these two velocities you will need global variables that store the current positions of the left and right wheel in addition to the positions of the wheels 4 milliseconds previous. We will call the current position of the wheel for example PosLeft_K and the previous position for example PosLeft_K_1. The velocity then can be easily calculated with the equation: VLeftK = (PosLeft_K – Posleft_K_1)/0.004. This will be called the "raw" velocity calculation as this can be a pretty noisy calculation of the velocity but also will have the least phase lag. When balancing the Segbot in Lab 7 we will have to add a filter to this velocity calculation to help with noise. For the speed control developed in exercise 3 and 4 this "raw" velocity works well. When writing your code to calculate these velocities, how do you handle saving the previous value of the wheels position? What should these previous variables be initialized to? Calculate the left and right velocities and print them to Tera Term. In addition, every 4ms call your setEPWM2A and setEPWM2B functions passing them the uLeft and uRight global float variables. Run your code and enable the motors. What are the velocity values when uLeft and uRight are 5? **Show your TA and Answer the Above Questions.**

You should see your velocities printing to Tera Term and they are probably turning in opposite directions. Here I want you to do some experimenting.

1. First off, by changing the values of uLeft and uRight and setting one to zero while the other one is set, figure out if EPWM2A drives the left or right motor and then of course the same for EPWM2B. If you got it opposite, make sure that setEPWM2A is passed the correct u value and setEPWM2B is passed the other u value. Here remember that left and right are determined as if you were driving the robot with the forward direction such that the caster is in the rear. **Show your TA which is the left and right wheel.**

2. Second figure out what u values cause the left and right wheels to spin in the positive direction. One will be negative due to the orientation of the motors. Negate the one u value that you find is negative to spin forward. This negative should be applied to the u value when passed to its setEPWM2? function and NOT inside the EPWM function. Once this is done in your code, if you set uLeft and uRight to 5 both motors will spin in the positive direction.

3. So now with all these changes, **demonstrate to your TA** that when you apply a "u" equaling 5 to both motors they spin with a positive velocity and in the robot's forward direction. In addition command both motors with -5 and show negative velocities.

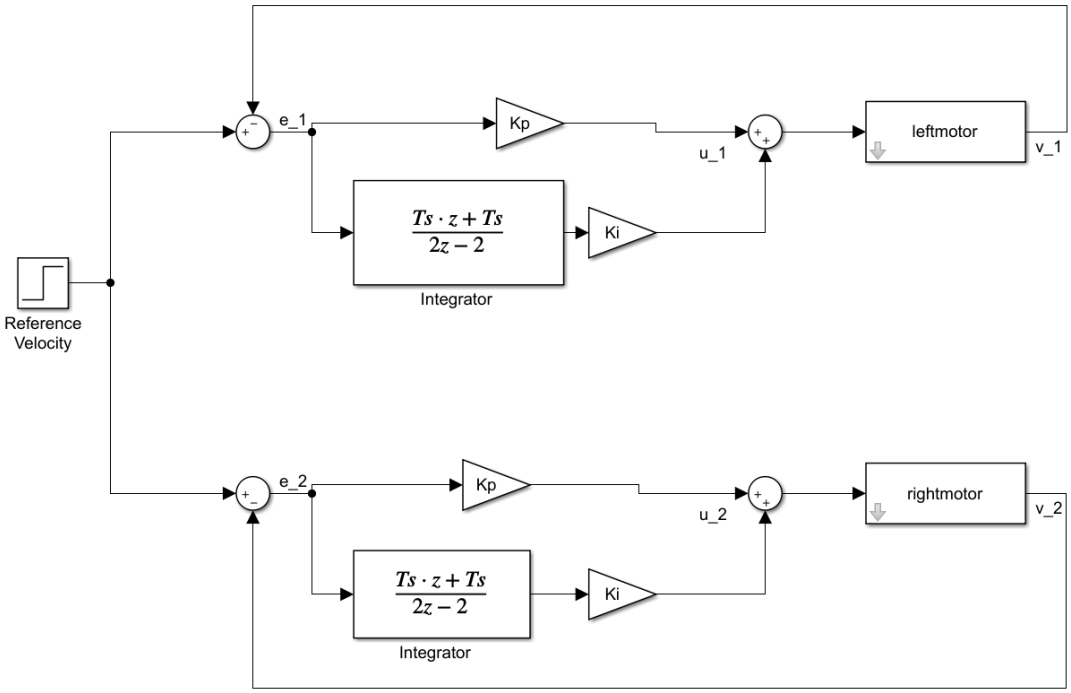
Exercise 3:

Using the below block diagram as a guide, implement the decoupled PI controller on both motors of the robot. Set Kp equal to 3 and Ki equal to 5. Use the following difference equations to form your control algorithm.

$$e_K = Vref - v_K$$

$$I_K = I_{K-1} + 0.004 * \frac{e_K + e_{K-1}}{2}$$

$$u_K = K_p e_K + K_i I_K$$



Decoupled PI controller diagram

Implement the controller and check to see that the speed matches various Vrefs in the range of -1.5 to 1.5 ft/sec. Also try a Ki gain of 15. Do you observe a difference in the motor's response? **Demonstrate to your TA different speeds and the difference between Ki = 5.0 and Ki = 25.0.**

It is necessary to note that integrators have "memory", which means they are affected by past behavior. Give the robot a Vref of 1 ft/s, then turn off the motor amp switch for a few seconds and switch back on. Note the behavior. The wheel spins faster than the set-point to "burn off" the extra integrated error accumulated while the wheel was turned off. Such *saturation* of the control input causes what is called *integral wind-up*. To prevent this behavior we must implement an *anti-windup controller*. One approach is to stop integrating when the control effort is saturated. In other words if your command is saturated at 10 or -10 set your Ik equal to the previous Ik 4 ms. ago. Try this method and check that the integral windup is fixed when the motor is spun both in the positive and negative directions. **Demonstrate to your TA.**

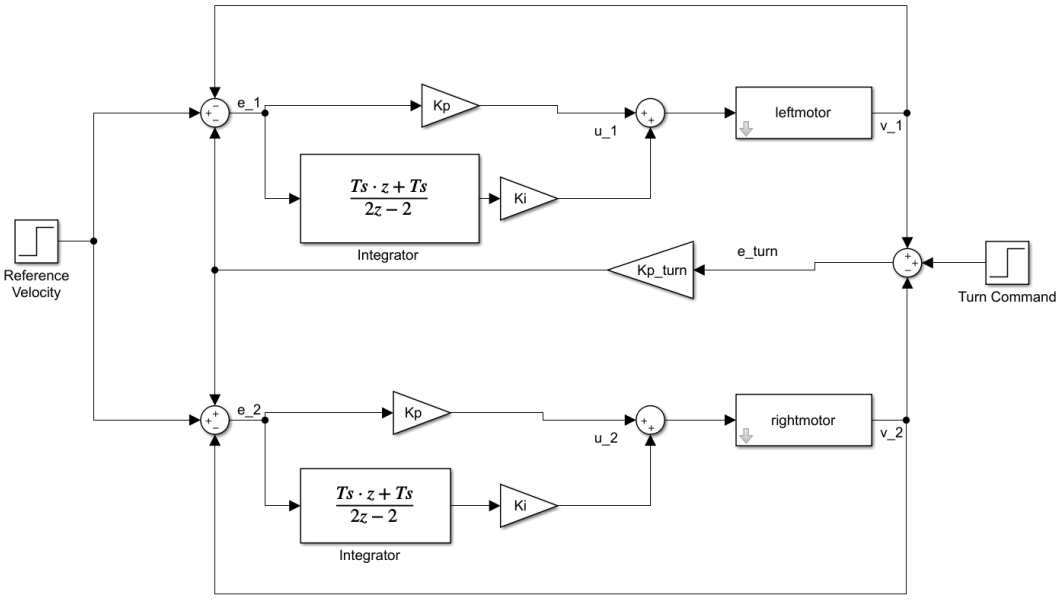
Exercise 4:

Implement a steering controller by coupling the motor control loops. Introduce a turn setpoint "turn" and form the turn error $e_{turn} = turn + (v_{left} - v_{right})$. You can see from this equation that the turn setpoint controls the amount by which one motor's speed exceeds the other motor's speed. Multiply the turn error by a gain K_{turn} (set to 3) and adjust the overall error signals as follows:

$$e_{Left} = Vref - v_{Left} - KP_{turn}e_{turn}$$

$$e_{Right} = Vref - v_{Right} + KP_{turn}e_{turn}$$

The approach is depicted in the block diagram below.



Coupled PI control structure

So the steering controller ensures that the average velocity tracks the reference and the turn command injects a difference in wheel speeds that is symmetric about the average velocity. Test your steering controller with several velocities and turn commands. Use the following code in your UARTA's receive function to command your robot by pressing the 'q', 'r' and '3' keys on your keyboard. Any other key sets turn back to 0 and Vref to 0.5 ft/sec.

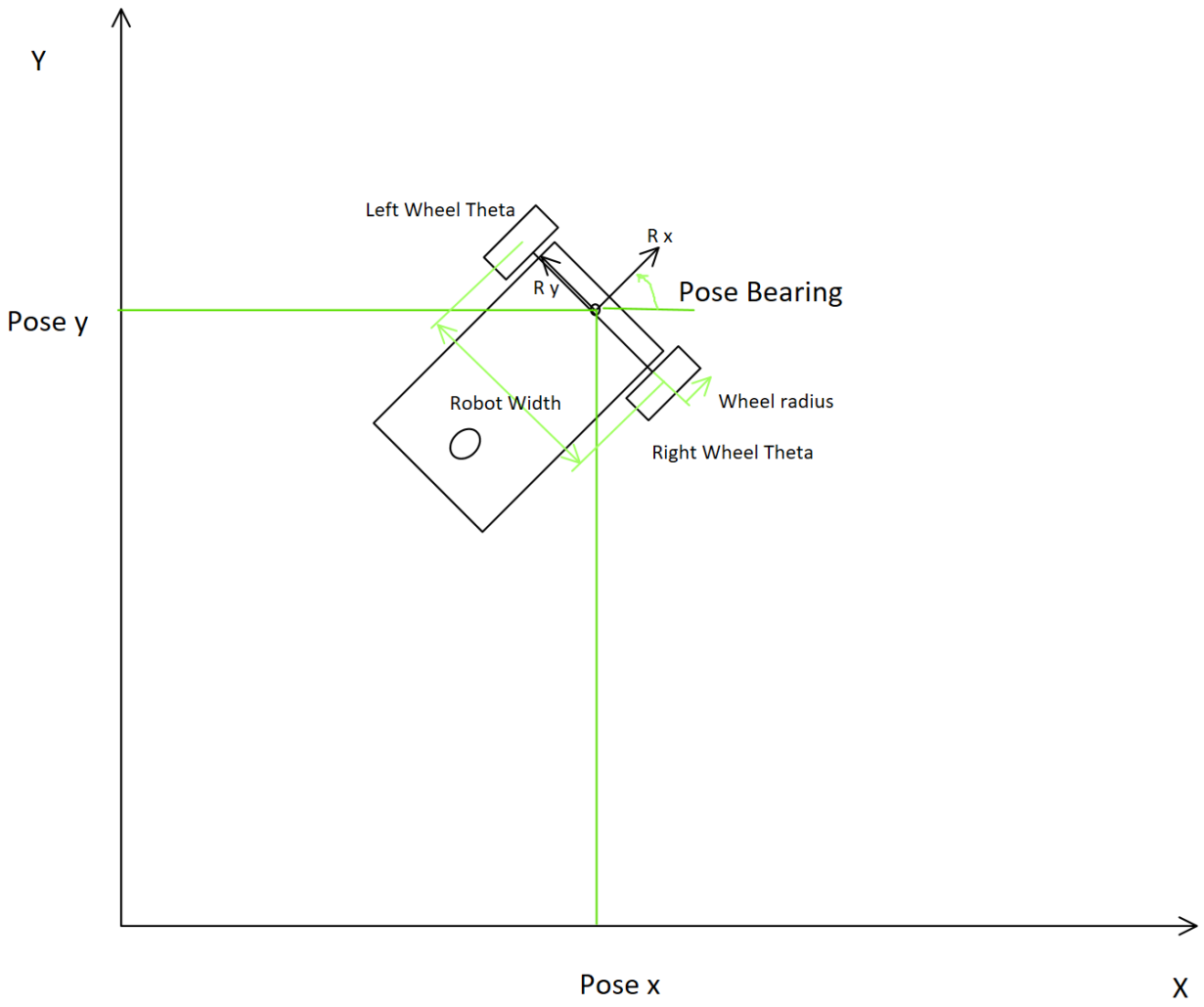
```
// This function is called each time a char is recieved over UARTA.

  numRXA ++;
  if (data == 'q') {
     turn = turn + 0.05;
  } else if (data == 'r') {
     turn = turn - 0.05;
  } else if (data == '3') {
     Vref = Vref + 0.1;
```

```
  } else {
    turn = 0;
    Vref = 0.5;
  }
}
```

Demonstrate to your TA.

Exercise 5:

Use the below diagram and equations to, every four milliseconds, calculate the new pose of the robot car as you are steering it around. Tune the value for your wheel radius and robot width to make the pose calculation as accurate as you can. Pose is the x, y and bearing location of the robot car. Your car will be battery powered but still connected to your laptop through a six foot USB cable. Print to Tera-term the pose of the robot car. Use units of feet and radians for the pose. Use feet because the tiles on the floor are one square foot. Start out with the radius of the wheel equal to 0.19583 feet and the distance, width, between the wheels 0.57743 feet. For the angular rate of the wheels, I recommend you look at your velocity calculation for the PI speed control and calculate angular rate in the same fashion but not converting to ft/s. The equation below will use the radius of the wheel along with the distance between the wheels to convert to ft/s.



$W_R = Robot\ Width$

$R_{Wh} = Radius\ Wheel$

$\theta_r = Right\ Wheel\ Rotation\ Angle$

$\theta_l = Left\ Wheel\ Rotation\ Angle$

$\emptyset_R = Robot\ Pose\ Angle\ or\ Bearing$

$x_R = Robot\ Pose\ X\ coordinate$

$y_R = Robot\ Pose\ Y\ coordinate$

$$\emptyset_R = \frac{R_{Wh}}{W_R}(\theta_r - \theta_l)$$

$$\theta_{avg} = 0.5 * (\theta_r + \theta_l)$$

$$\dot{\theta}_{avg} = 0.5 * (\dot{\theta}_r + \dot{\theta}_l)$$

$$\dot{x}_R = R_{Wh}\dot{\theta}_{avg}\cos(\emptyset_R)$$

$$\dot{y}_R = R_{Wh}\dot{\theta}_{avg}\sin(\emptyset_R)$$

$x_R = \int \dot{x}_R$, Integrate with the Trapezoidal Rule in your C code

$y_R = \int \dot{y}_R$, Integrate with the Trapezoidal Rule in your C code

Lab Checkoff:

1. Demonstrate your Optical Encoder Sensors working.
2. Demonstrate your calculation of motor angular velocity and robot linear velocity.
3. Demonstrate your decoupled PI speed control.
4. Demonstrate your coupled speed control and the robot running on the floor steering right and left.
5. Demonstrate that your robot can keep track of its current pose (x,y,bearing) on the floor while the robot is manually steered.
6. Submit all your written code after adding comments explaining what you learned. Also be clear what code is for what exercise.
7. Submit your HowTo document with things to remember about working with the robot car and answer the following questions.
    a. Explain how the magnetic encoder works. Talk about the A channel and the B channel of the encoder and how the eQep peripheral monitors the state of these two signals. Explain how by monitoring the state of these two signals the eQep peripheral is able to up and down count its QPOSCNT register whose count value is proportional to the angle the encoder has turned. Draw some pictures for your explanation.
    b. What is the resolution/accuracy in feet of our encoder adding the 30:1 gear ratio of our motor and the radius of the wheel? In other words, when you start your code running the motors are at rest so the feet returned is 0? If you move the wheels super slowly in the positive direction, what is the next angle you would receive from your function if the encoder has only moved one count?