

## **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 optical 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 robot's motors and find the black housing enclosed optical encoders. There are 100 slits on the small wheel inside this enclosure. The optical sensors of the optical encoder sense each slit as they rotate by. So one rotation of the motor creates 100 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  $4 \times 100$  or 400 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, below, 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 unaffected 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.QOSMAX = 0xFFFFFFFF; // use full range of the 32 bit count.
EQep2Regs.QEPCTL.bit.FREE_SOFT = 2; // EQep unaffected 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

    // 100 slits in the encoder disk so 100 square waves per one revolution of the
    // DC motor's back shaft. Then Quadrature Decoder mode multiplies this by 4 so 400 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

    // 100 slits in the encoder disk so 100 square waves per one revolution of the
    // DC motor's back shaft. Then Quadrature Decoder mode multiplies this by 4 so 400 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_serialSCIA()`, etc., function calls 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. 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  $2\pi$ . 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:  $V_{LeftK} = (PosLeft\_K - Posleft\_K\_1)/0.004$ . 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 below, 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.

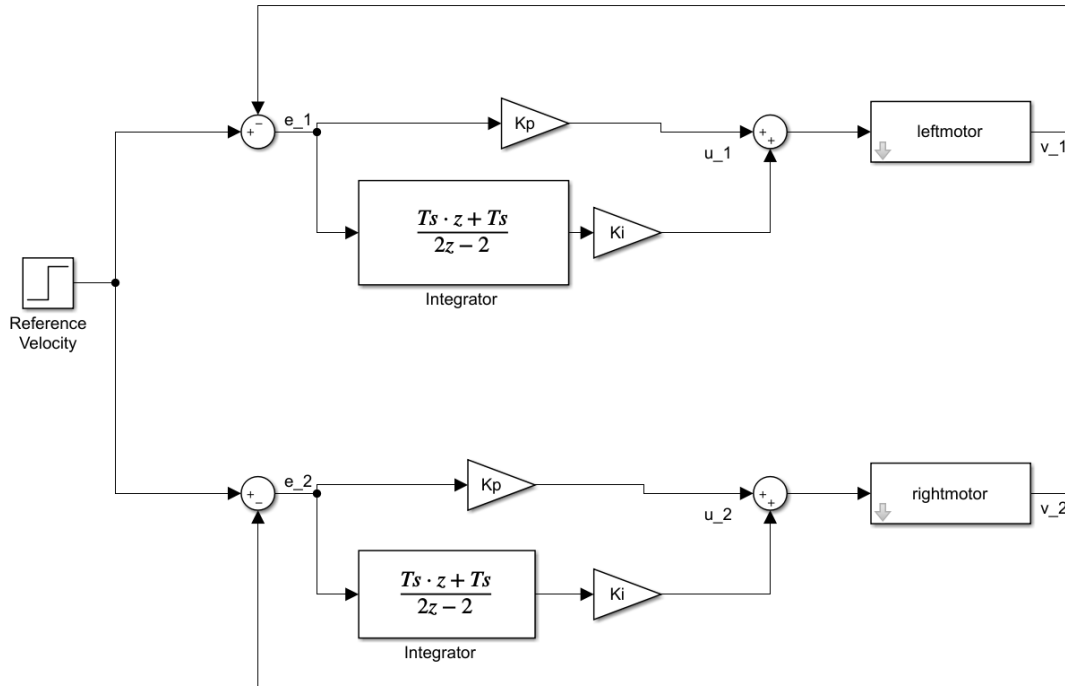
### **Exercise 3:**

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.

$$e_K = V_{ref} - v_K$$

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

$$u_K = K_p e_K + K_i I_K$$



**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 and 25. 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.**

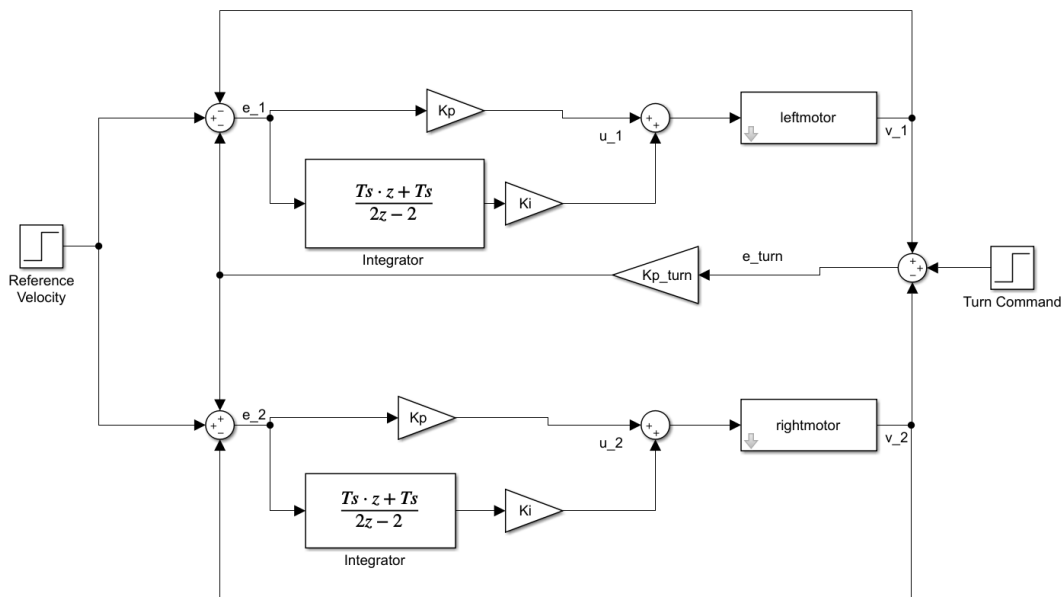
### Exercise 4:

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:

$$e_{Left} = V_{ref} - v_{Left} - K_{P_{turn}} e_{turn}$$

$$e_{Right} = V_{ref} - v_{Right} + K_{P_{turn}} e_{turn}$$

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 interrupt function, see instructions below, to command your robot by pressing the ‘q’, ‘r’ and ‘3’ keys on your keyboard. Any other key sets turn back to 0 and  $V_{ref}$  to 0.5 ft/sec. We will call this the “Cabled Way” to steer your robot as your USB cable needs to be connected to the F28379D board. In Exercise 5 you will steer the robot wirelessly.

In your Lab6 project, find and open the C file F28379dSerial.c. In this file, search for the function “RXAINT\_recv\_ready”. This is the interrupt function that is called when UARTA receives a single character. Copy the below code for the RXAINT\_recv\_ready function and paste it over the top of the default code for the RXAINT\_recv\_ready function. Look at this code and notice that when you press the ‘q’, ‘r’ and ‘3’ keys (in Tera Term) you change the turn and  $V_{ref}$  values. Pressing any other key sets  $V_{ref}$  to 0.5 ft/s and turn to 0. *NOTE: When you cut and paste C code from Word or PDF files sometimes the characters are not recognized by the C compiler. One common character is the “-”, so watch out for that one.*

Since Tera Term is running on your Lab PC you will have to test this steering at your bench with the robot sitting on a platform not allowing the wheels to touch the table. If in the future you would like to steer the robot with the USB cable connected, you could install Tera Term on your laptop.

```
extern float turn;
extern float Vref;
//This function is called each time a char is recieved over UARTA.
//for SerialA
#ifdef _FLASH
#pragma CODE_SECTION(RXAINT_recv_ready, ".TI.ramfunc");
#endif
__interrupt void RXAINT_recv_ready(void)
{
    RXAdata = SciaRegs.SCIRXBUF.all;

    /* SCI PE or FE error */
    if (RXAdata & 0xC000) {
        SciaRegs.SCICTL1.bit.SWRESET = 0;
        SciaRegs.SCICTL1.bit.SWRESET = 1;
        SciaRegs.SCIFFRX.bit.RXFIFORESET = 0;
        SciaRegs.SCIFFRX.bit.RXFIFORESET = 1;
    } else {
        RXAdata = RXAdata & 0x00FF;
        numRXA++;
        if (RXAdata == 'q') {
            turn = turn + 0.05;
        } else if (RXAdata == 'r') {
            turn = turn - 0.05;
        } else if (RXAdata == '3') {
            Vref = Vref + 0.1;
        } else {
            turn = 0;
            Vref = 0.5;
        }
    }
}

SciaRegs.SCIFFRX.bit.RXFFINTCLR = 1;
PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}
```

**Demonstrate this working to your TA.**



### Exercise 5:

As you have already completed in Homework #2, send commands from LabVIEW to steer your robot wirelessly using the ESP32's Wi-Fi. The Lab PCs are also connected to the same routers as the ESP32's Wi-Fi. You will want to review the LabVIEW program you created in Homework #2 and the C code you wrote. The LabVIEW program you created in Homework #2 is ready for this task and receives eight floating point values from the F28379D board and sends eight floating point values to the F28379D board. The first three of the eight values sent from the F28379D board to LabVIEW are the x, y and bearing pose of the robot. You will be calculating x, y, bearing in Exercise 6. The other five values can be whatever else you want to send to LabVIEW. The first two values of the eight values sent from LabVIEW to the F28379D board are "Vref" and "turn". The other six values sent from LabVIEW can be used however you please.

With your Homework #2 LabVIEW program ready to go, all you need to do is add some C code to your F28379D project. First define these global variables towards the top of your C-file. Note how "printLV1" and "printLV2" are not defined. This is because those two values are now "Vref" and "turn".

```
float printLV3 = 0;
float printLV4 = 0;
float printLV5 = 0;
float printLV6 = 0;
float printLV7 = 0;
float printLV8 = 0;
float x = 0;
float y = 0;
float bearing = 0;
extern uint16_t NewLVData;
extern float fromLVvalues[LVNUM_TOFROM_FLOATS];
extern LVSendFloats_t DataToLabView;
extern char LVsenddata[LVNUM_TOFROM_FLOATS*4+2];
extern uint16_t newLinuxCommands;
extern float LinuxCommands[CMDNUM_FROM_FLOATS];
```

Next inside your timer interrupt function that is being called every 4ms, add this below code that receives the values sent from LabVIEW and every 248ms sends floats to LabVIEW.

```
if (NewLVData == 1) {
    NewLVData = 0;
    Vref = fromLVvalues[0];
    turn = fromLVvalues[1];
    printLV3 = fromLVvalues[2];
    printLV4 = fromLVvalues[3];
    printLV5 = fromLVvalues[4];
    printLV6 = fromLVvalues[5];
    printLV7 = fromLVvalues[6];
    printLV8 = fromLVvalues[7];
```

```

}

if((numTimer??????calls%62) == 0) { // change to the counter variable of you selected 4ms. timer
    DataToLabView.floatData[0] = x;
    DataToLabView.floatData[1] = y;
    DataToLabView.floatData[2] = bearing;
    DataToLabView.floatData[3] = 2.0*((float)numTimer0calls)*.001;
    DataToLabView.floatData[4] = 3.0*((float)numTimer0calls)*.001;
    DataToLabView.floatData[5] = (float)numTimer0calls;
    DataToLabView.floatData[6] = (float)numTimer0calls*4.0;
    DataToLabView.floatData[7] = (float)numTimer0calls*5.0;
    LVsenddata[0] = '*'; // header for LVdata
    LVsenddata[1] = '$';
    for (int i=0;i<LVNUM_TOFROM_FLOATS*4;i++) {
        if (i%2==0) {
            LVsenddata[i+2] = DataToLabView.rawData[i/2] & 0xFF;
        } else {
            LVsenddata[i+2] = (DataToLabView.rawData[i/2]>>8) & 0xFF;
        }
    }
    serial_sendSCID(&SerialID, LVsenddata, 4*LVNUM_TOFROM_FLOATS + 2);
}

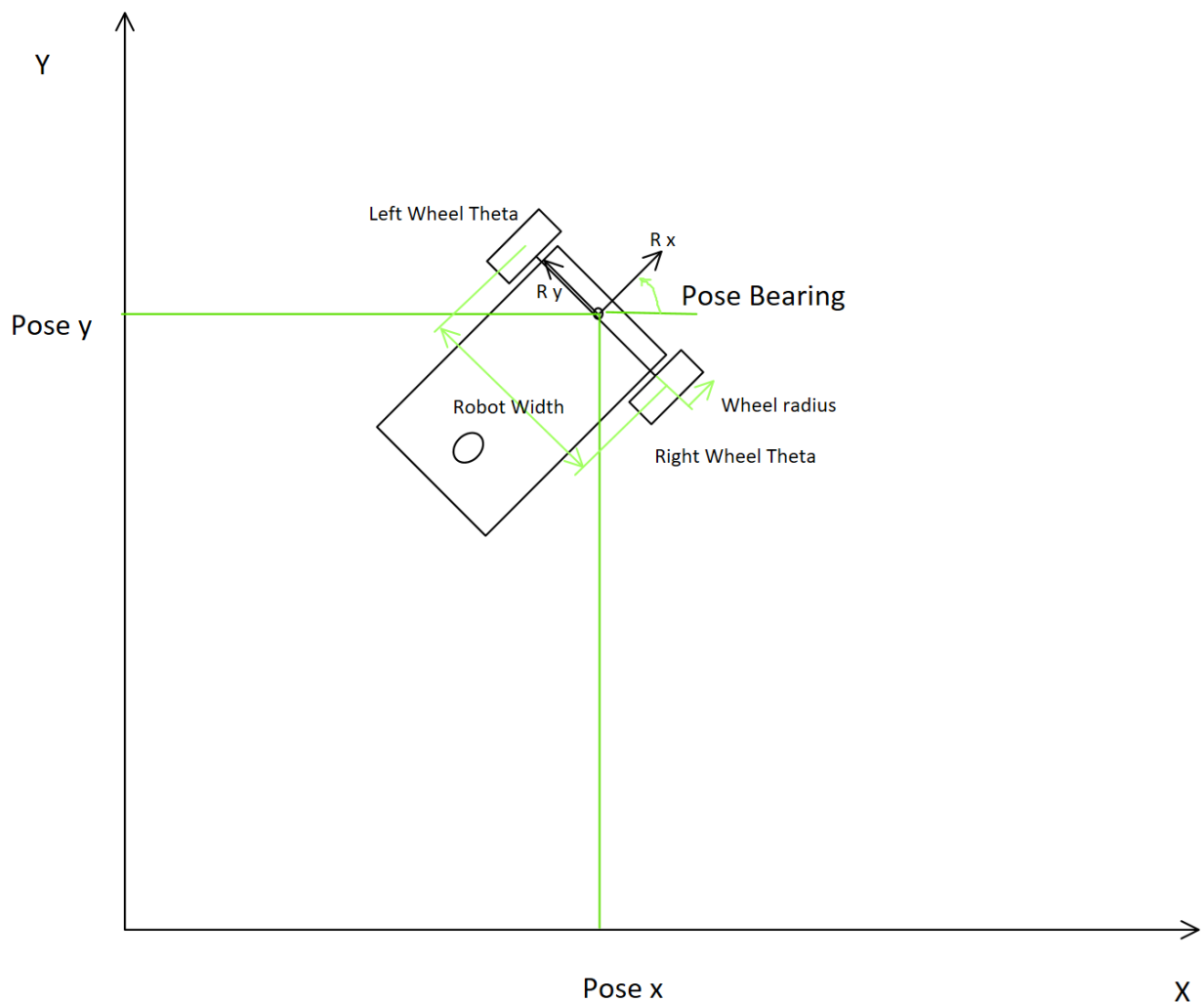
```

Make sure to flash your C code to the robot so that you can unplug the USB cable and put the robot on the floor. You will need a battery for the robot so ask your TA how to connect a battery. Once the robot is battery powered and on the floor you will need to “telnet” into Nuttx on the ESP32 and start the wireless communication. Use these steps from Homework #2.

1. When the robot is powered on, the ESP32 board is also powered. It takes about 5 to 10 seconds for Nuttx to boot on the ESP32 board. After waiting 10 seconds, open a CMD prompt in Windows and test that Nuttx is ready by pinging your robot’s IP.  
ping 192.168.1.<yourRobotsIPnumber>, i.e. ping 192.168.1.55  
You should receive a response. Press “<CTRL> c” to end ping. If you do not receive a response with ping, check that you typed the correct IP. Also you can press the small “EN” button on the robot’s ESP32 board. Wait 10 seconds and try again.  
*NOTE: you do not need to do this ping command every time you want to connect. It is a debug step if LabVIEW cannot connect to Nuttx.*
2. Then from the Windows CMD prompt connect to Nuttx over telnet by typing  
“putty -telnet 192.168.1.<yourRobotsIPnumber>” without the quotes.  
This should give you a Nuttx Shell, nsh> prompt. At this prompt type  
tcpLVCOM. tcpLVCOM is now waiting for your LABVIEW program to connect.
3. Run your LabVIEW program.
4. Switch the Motor Enable switch on your robot to enable the motors.
5. Use LabVIEW to steer around your robot.

### Exercise 6:

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. You will need to battery powered your robot so ask your TA on how to connect the battery. 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 0.19460 feet and the distance, width, between the wheels 0.56759 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.



$$W_R = \text{Robot Width}$$

$$R_{Wh} = \text{Radius Wheel}$$

$$\theta_r = \text{Right Wheel Rotation Angle}$$

$$\theta_l = \text{Left Wheel Rotation Angle}$$

$$\phi_R = \text{Robot Pose Angle or Bearing}$$

$$x_R = \text{Robot Pose X coordinate}$$

$y_R = \text{Robot Pose Y coordinate}$

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

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

$$\phi_R = \frac{R_{Wh}}{W_R} (\theta_r - \theta_l) \text{ * See Note Below}$$

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

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

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

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

### **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 optical 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 and how far have you traveled in feet?

Note\* This equation is a bit of a brain teaser to figure out. I think the easiest way to derive this equation is to first just move the right motor's theta and calculate the bearing angle. Then starting back at the same initial position just move the left motor's theta and calculate the bearing angle. Then to find the equation for both motors moving, superimpose (sum) the two solutions together to get the final equation that calculates bearing given the angles both motors have turned. Of course this equation is assuming there is no slippage of the wheels except to rotate at a point.