Lab2 notes from lecture

How to infer the path that it flows from measurements we can get from its wheel rotations?

though unless I know where I started. If you know where A is along the curve you can figure out where B is. If you know where you start, how do you figure out where the destination is.

A -> B if we can figure out how to get each individual displacement along curve and add them up to get the total displacement. Calculate displacement from the last time you took a measurement.

How: periodically sample the displacement between the two wheels. Method to read back the techtometer. From the techtometer readings figure out the difference b/ween the last measurement and the current measurement.

Measure displacement of the left wheel

Measure displacement of the right wheel

Do the math necessary to turn that into the vector displacement from our last sample.

Picture of two wheeled robots.

Model the position as the point between the axels. Measure independently the rotation of the L and R wheels. We know the radii of the wheels so we can turn the rotations into a linear displacement. We also need to know the wheelbase of the wheels (distance b/ween the two wheels).

Inputs: rotations of the left and right wheels

Knowledge: the radii of wheels and the wheelbase of vehicle

Outputs (what we want to figure out) (model displacement as a vector): we are first going to model the displacement as a vector. A vector has a magnitude and a direction. We want to figure out the length of that vector (length of the displacement). Then we want to figure out the change in orientation (change in heading) of that vector from the last time we made a measurement. Then we want to add that vector to our last position to figure out our current position.

Picture of triangle:

We want to figure out displacement from point p to point p’. Wheelbase starts of at the first line position. For the interval that we’re measuring the left wheel moves over d1 amount to another position. And the right wheel moves over d2 – d1 amount to another position. We still want to figure out how p’ moves away from p.

We can represent this as a vector displacement. Magnitude is going to be the length between p’ – p. The displacement will be angle theta, which is the turning angle (change in heading).

How do we figure out d1, d2. Periodically read sensor (which are the techtometers on the left and right motors). So how are we going to do this: at each point the left techtometer count and left techtometer count will be measured. Assuming we kept track o f what the previous readings were. Take the current reading for one motor, subtract the previous distance and we get a difference, the delta degrees between previous and current; how many degrees the wheel rotated. Phi represents the actual differences form the current to the last reading.

To get the linear displacement d1: 2PIr (circumference of a circle, where r is the radius of the wheel) gives us the complete rotation of the wheel. Techtometer is calibrated in degrees, so a difference of 50 will be the difference of 50 degrees. To turn this into a number of turns we divide by 360. In the slides, he got rid of the 2PIr, got rid of 2 by dividing it into 360 and got 180. It makes sense. Where Rw is the radius of wheel. We are assuming they are the same but in fact they are not. Assume that the radii of the left and right wheel are the same (they are not there is slight differences between the two). There is also slip b/ween the wheel and the floor, the slip may also be different between each wheel which will introduce error into the process (this slip may not be the same for both wheels). As the distance from A to B is longer the more we are going to drift off.

What does all this tell me, since we have d1 and d2 the displacement (magnitude) of p’ is (d1+ d2) / 2 the average.

The bigger problem is figuring out the change in heading or change in orientation. From the triangle drawing we have to figure out the angle theta. Figure out how we have rotated relative to the initial orientation. Well if we knew the line s (difference between d2 -d1) . We know the wheelbase. So we use trig. But we don’t have d2 – d1, what we have is the arc length that we have drawn b/c we are moving in a curve (the red line in drawing). So theta is d / Wb where theta is in radians (you get tantheta), you wanna convert to degrees. All we have to do is figure out the arc length s. figuring out s is hard, so we are going to approximate s with d2-d1 when s is really small. For small displacements, the arc length we wanna calculate is close enuf. What is the typical precision that we’ll need. If you’re within a couple cm you’re close enuf. Define a max error that we’ll tolerate and its going to be 2 cm for ex. Analytically calculate how far off an angle I can have where the approximation violates this “2cm” distance. Now how fast do I have to have to take the measurements as they come in. If the displacement during the interval while we take measurement and wait exceeds the bound then not good. Fix problem by fixing more frequently. How do we figure how fast we have to measure. At some point we’ll have to go so fast you won’t even see a difference. We have a bound (we can’t do it to more than 1-2 cm). Optimize choice of parameters, smapling rate. Or the completxity of your model to meet the design of requirements. The silicon that the algorithm is going to run on is limited so obviously we wanna simplify the calculations as much as possible. Here the trig is simplified. But why do we care its just Math.function but in terms of energy the operations are expensive, so if we can simplify the model from the types of operations we have to perform that’s going to make it run better.

Make the assumption that the arc length is a close enuf approx. to the distance d = d2 -d1 so the theta is = d/ Wb.

Now that we’ve calculated the change in displacement, calculating the change in position is going to be easy. So change in position is going to be change in x and y. We we’re talking about x and y we mean movement in a plane where y is north and x is east west axis. The particular odometry model is relative b/c webots is different. So we are talking about our own motion with respect to the experiment. So if we have our displacement vector dh and change in heading thetah. So delta x = dh sinthetah and y = dh costthetah.

Current position is = xlast + delta x ; y current = ylast + delta y

Trajectory tracking algorithm (odometer):

Initialization

To start of initialization: set starting point (assume 0,0 and we’re pointing in the y direction).

We also need to know tacho counts (left and right techtometer measurements) save them as last\_tacho\_l and last\_tacho\_r

Updating

read left and right counts calculate phil and phir (current left – last left; current right – right). So now the last tacho count is going to be the current for left and same for right.

Calculating d1 ,d2 , d = d2 – d1, dh = average of d1/d2, theta is theta h.

Theta is taking the d which is d2-d1/Wb which is in radians , if you do ANY TRIG remember to convert to degrees.

Wait is delta t. Most embedded systems have a timer service that wakes up the particular step periodically (implemented in hardware). We implement this using a sleep b/c wait doesn’t exist in simulation.

In the model the parameters are set in Resources.java. The challenges with the simulator is getting the accurate dimensions on what the actual dimensions are. Look in the Resources file to see what the parameters are.

Variables:

lastTachoL

lastTachoR

nowTachoL

nowTachoR

X (current X position)

Y (current Y position)

Theta (current orientation) (we use doubles for x,y, theta b/c of the real valued math that we do)

// Resources

Set up the motor resources and getTextLCD() to output text in the ev3 console.

Set up display thing so that you can print what the machine is doing so the data can be analysed.

How does sampling affect performance: the more you sample, the more computation you’re doing, the more loaded the processor will be. What’s the minimum sampling rate i can get away with. The way to do this is to change the sampling rate parameter and observe how it will degrade the results you will get in terms of tracking the position. What the Timer myTimer = new Timer(SINTERVAL, new ODoDemo()); class does is set up the machinery).

What does the doSquare class do, it basically drives the cart in a square. For this lab just focus on tracking the trajectory. NExt class we’ll be talking about generating the trajectory.

After:

reset the tacho counts with resetTachoCount() method. Then we wanna use getTachoCount() to get the actual reading that’s there.

For understandin how to make threads work: how do you move information between the independently executing thread units. Generate a class variable which is a global and share that (AVOID this b/c if we have multiple threads that are accessing the same global variable in memory we have to worry about contention. If someone’s writing it and someone comes in and tries to write at the same time we’re gonna get unintended consequences). The preferred way of handling communication is getters and setters. If we want to access instance variable in a class we create a getter method to do that.

Once we start everyone running, (LOOK ON CODE IN MYCOURSES)

What’s really important is the timeOut() method. When the timer files it generates an exception and transfers context to your particular thread the timer service is expecting to transfer control to a method called timed out. In that method you wanna write everything you want to do every time that the timer service returns control to you. Algorithm we talked about:

read current value of L and R tachos. Then figure out the out the left and the right displacements which is the pi \* Wr \* (nowTachol – lastTachoL)/180 and same for right. Now the lastTachoL = nowTachoL and same for R. The actual displacement is deltaD = disL +. disR /2. Change in heading is the disL- distR/Wb. And the theta is going to be theta + deltaT. Theta is represented in radians. If we need degrees only convert when you need to use it. Because the execution is periodical we want to eliminate having another transcendental function there. So accumulate the change in heading of radians. To calculate dX its going to be deltaD \* sin(Math.sin(theta) theta in radians. same for deltaY for components of displacement. Then we wanna calculate X = X + dX and same for Y update estimates of X and Y position.

if we want to run timedOut() really fast the transcendental functions will take time. We want to look at our robot and think what’s taking all the time, if it’s the transcendental math then we want to have a lookup table. But a lookup table consumes memory, we try and balance off the competing requirements to find the happy medium. Always look at how something performs. The problem is if you sample so fast that you don’t have time to get through one iteration of the timedOut() function then we are not in a good place. We want to adjust the sampling rate intervals. We need to know what the time it takes to go through timedOUt() once by running it a million times in a loop look at the wall clock on your watch and divide by a million to figure out how long one iteration takes. Determine this as part of the experiment.

This CLASS Was the odometer class. And next class we will learn how to write the squareDriver class. Monday we’ll learn about the actual trajectory generation.

NOTES ON SQUARE DRIVER

Path in red , assume its given to us, we wanna make a robot that traverses this path. We have to break this path into segments and approximate each by its tangent direction. The issue is, how many segments should we break the path in, you want to put the marks in the locations where the path is changing directions.

How do we solve the moving problem: we are at particular point, what strategy is to do vector displacement to next point along path. We did opposite last time, we measured wheel rotation and we worked backwards to figure out change in orientation and displacement. This time we wanna generate this change in orientation and change in displacement, using the kinematic model. The more complex the machine the more complex the model so its another challenge in design. Simple is better.

LINEAR MOTION

To move a motor in a straight line we rotate the left/right wheels at same rate, rotate for the right number of revolutions that moves you the correct linear distance. We know the distance that robot travels is (2pi Wr \* #rotations) how do we specify number of rotations? Cuz we can have half a turn, quarter of a turn. There’s a method called rotate() which says rotate so many degrees. We can incrementally rotate from 1 to 360 degrees. We can specify rotations as degrees : ex 720 degrees is 2 rotations. summarize that as 2Piwheelradius\*totalnumberofdegrees that the motor turns /360. This gives us the translation from wheel rotations to linear distance. So the number of rotations a wheel has to make to cross a certain distance would be = (distance / (2PiWr)) \* 360.

ROTATION IN PLACE (no displacement)

We want to be able to figure out how to turn so many degrees in place how many rotations do we need from left and right wheel. To rotate in place one wheel goes forward and the other goes backward at exactly the same rate. What’s the model for this.?

OmegaC is the rotation the distance (semicircle thing) is omegaC in radians \* the radius of the wheel base. So how many turns do we hav3 to make in left and right wheels to cover that omegaC distance. So, its going to be Rw \* omegaW (remember that omegaW is the number of degrees total our wheel has rotated). So Rw \* Ωw = Rc \* Ωc. So Ωw = Ωc \* (Rc / Rw). The units for Rw and distance d have to be the same. Preferably distance d in cm.

Ok now we gotta think about the physical components that are actually run the program. So computer has double and float arithmetic, but the ev3 in physical world is not strong enuf to do this. So there is a price to pay for doing calculations with real numbers. So its faster to do integer arithmetic. But if you keep dropping the fractional part, over a longer distance the errors add up so you wanna scale the numbers. Multiply by 100 the amount so that you get an integer (instead of 20.34 we get 2034) so at the end of all the computations you wanna divide by 100 to get a pretty accurate number. Scale to maintain precision and unscale when your calculation is done.

Diversion: b/c when you tell a motor to rotate so much , you should have an idea what’s going on. We want to have a control loop. Controller is the ev3. Ev3 specifies the voltage to the motor, the higher the voltage the faster the motor turns. Our motor spins and the sensor is an encoder, it spits out a pulse anytime a motor turns by one degree. If we rotate 15% encoder will general 15 pulses. getTachoCounts() is associated with the motor class to read the current position of the motor essentially that’s where its coming from.

To write the code to operate the motor we

read init encoder values

apply voltage to motor

loop

read current encoder value

diff = initial – current (initial count when motor started vs what’s the encoder saying now). Check if the difference less than number of degrees I was told to rotate. If not less than go back and do this again, otherwise we stop the motor from turning…

Summarize the main methods we will use in SquareDriver

rotate(angle, return) – integer argument angle (degrees we want to rotate, positive number means motor rotates clockwise, negative means motor rotates counterclockwise). Second argument is really important, if it’s true the rotate method comes back right away so you can continue the next step in the program, if its falls then the rotate method will only return once the motor has completed the command that you gave it. Let’s think: If we want to move forward, if we did not supply the second argument or simply said false. Then what would happen is we would rotate left and then right wheel but the behavior we want is for them to run at the same time. Which is not what we get. This is why we have to specify whether we want to return immediately or return upon completion.

This also implies something in terms of the rotate method. It says that the instance of the class that it belongs to means that it has to be a thread. For the motors to be able to execute at the same time (the loop), the motor class has to be implemented as a thread so that they can run concurrently with everything else we want to do in the program.

Two flavours of rotate. One is rotate relative to where you happen to be now. And rotateTo specifies to go to an absolute position. We rarely use rotateTo. Then there’s setSpeed which sets the speed in degrees/sec. Another method regulateSpeed() : if you’re moving in a straight line, we’d like to control the left/right motors so that they’re in sync w/ each other and regulatespeed does this for you. SmoothAcceleration ramps up the motor in such a way to minimize the wheel slip.

SquareDRIVER

Parameters: (in the eclipse code given those parameters live in the resources file).

DISTTODEG: an important scale factor. When you calculate the # of degrees that you need for a certain linear distance, this is the parameter you multiply the distance by to get the wheel rotation.

ORIDENTATIONTODEG: 2nd scale factor which does exactly same thing as DISTTODEG but to the cart.

We want to be able to speicify linear distance. The SIDE is in units of 10th of a milimeter so 5000 units corresponds to half a meter.

FWDSPEED: speed robot travels when it moves in straight line.

TRNSPEED: turn speed, when you’re orienting in place you wanna slow down a bit b/c you wanna minimize the slip. If you turn too fast you’ll slip the wheels. The second parameter is only when we’re moving in place . To take into account that the moving straight and turn in place affect the wheel slip differently.

OpenLoop is basically square driver. Notice it hasn’t been made an extension of the thread class. B/c its code that is run on ev3 brick so by definition the default class is already a thread. Inside webots/simlejos you have to specifically specify that. In the sample code for assignment 2 it makes this distinction very clear. WHERE IS THE SAMPLE CODE???????

moveDistFwd() two parameters SIDE (linear distance to move, 5000 mm or 1/2 a m) and the velocity is determined by forward speed. Then we rotate 90 degrees which is same as moveDisFwd but it uses a different scale factor. And then we do it 4 times to make each side of the square.

METHODS

MoveDisForward() :

1. calculate the equivalent rotation in terms of the motors. So distance \* DISTTODEG/100 (/100 takes scaling into account).

Rotate():

1. rotation angle is the input angle \* ORIENTODEG/100 . These are all integer operations. We want to be as efficient as we can to do it quickly. Difference between going straight and rotating. It’s that the minus sign. The leftMotor rotates clockwise, while the rightMotor rotates counter clockwise.