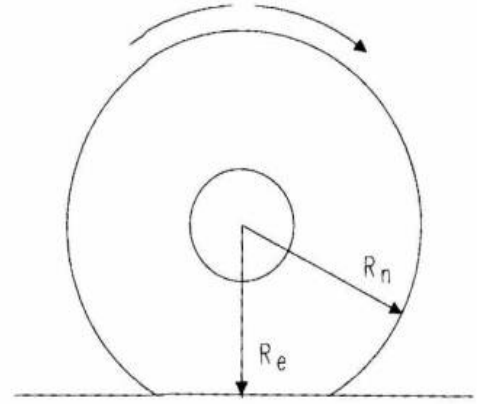
Position Tracking at the Next Level
Wesley Myers
wmyers@andrew.cmu.edu
Dr. Eugene Myers
cedarhouse@comcast.net

Position Tracking at the Next Level

Introduction

This paper presents how to navigate using a new version of position tracking. Our first version of position tracking was developed several years ago for Botball [1][2]. Position tracking is where the robot keeps track of its position by monitoring its motion. Our previous papers describe our old implementations of this system. This paper will discuss a new method learned during the Author's coursework at Carnegie Mellon University. This solution is for a differential drive robot.

Motivation



In the ever-changing conditions of Botball, an accurate method of keeping track of a robot's position is needed to achieve an edge over other teams. Our old position-tracking algorithm was relatively simple. However, this simplicity caused errors to accumulate every time the robot executed any degree of turning motion. Turning in a given direction resulted in an error in the robot's direction information and turning in the opposite direction was not sufficient to correct the error. The only way to correct the error was to align the robot along a PVC pipe by driving into it (for alignment) and then use the camera to localize. With the team's entry into the KIPR Open, a more accurate position tracking algorithm was necessary.

With the metal parts in the KIPR Open, we realized that weight is a significant factor, which causes more error in our navigation system. So we must achieve better navigation to perform our tasks. The figure above illustrates such an example that weight causes with rubber tires. What this means is that the movement of our robot can cause our wheel radius to change. By having better code for navigation, we can achieve more accurate navigation even with this varying condition. The VEX wheels do not have as much of a problem with this as the Lego wheels, none-the-less this new algorithm helps in either case.

Runge-Kutta

The Runge-Kutta Method is a method of approximating the solution of ordinary differential equations (ODEs). This particular method we chose involves the fourth order

method. What that means is that the error per step is t^5, while the total accumulated error is t^4. In this case t refers to the step size, or the size of the interval. This method provides for correction in the error caused by the turning motion and is implemented in our KIPR Open robots.

The Method

This section describes how to basically implement this method of position tracking [3]. A later section provides basic code to implement this in C.

Step 1: Write a loop to poll the motor encoders at intervals. Use these values to calculate the left and right wheel velocities in "ticks" or degrees per second. This can best be done using the equation below.

v_l = (current_left_motor_encoder_ticks – previous_left_motor_encoder_ticks)/(time_elapsed_since_last_poll)
v_r = (current_right_motor_encoder_ticks – previous_right_motor_encoder_ticks)/(time_elapsed_since_last_poll)

Step 2: Based upon these wheel velocities, we can convert them into linear velocities by multiplying the values by the radius of the wheels.

$V_l = v_l * R$
$V_r = v_r * R$

Step 3: With these values we can now get the linear and angular velocity of the robot. L is the distance between the contact point of the wheels on the ground.

$\nu = (V_r + V_l)/2$
$\omega = (V_r - V_l)/L$

Step 4: Given that we have the linear and angular velocity, we need to find (x, y, θ). That involves solving the below non-linear system of differential equations. Since the code cannot simply solve the ODE, we must use a numerical integrator like a fourth order Runge-Kutta to approximate this position vector. Let t be the time elapsed since we last ran the integration loop and n represent the current iteration of the loop.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \nu \cos(\theta) \\ \nu \sin(\theta) \\ \omega \end{pmatrix}$$

Runge-Kutta 4th Order Equations

Here are the equations to obtain the desired (x, y, θ) position of the robot using the Runge-Kutta 4th Order Equations [4].

$$k_{00} = v\cos(\theta_{n-1})$$
$$k_{01} = v\sin(\theta_{n-1})$$
$$k_{02} = \omega$$

$$k_{10} = v\cos\left(\theta_{n-1} + \frac{t}{2}k_{02}\right)$$
$$k_{11} = v\sin\left(\theta_{n-1} + \frac{t}{2}k_{02}\right)$$
$$k_{12} = \omega$$

$$k_{20} = v\cos\left(\theta_{n-1} + \frac{t}{2}k_{12}\right)$$
$$k_{21} = v\sin\left(\theta_{n-1} + \frac{t}{2}k_{12}\right)$$
$$k_{22} = \omega$$

$$k_{30} = v\cos(\theta_{n-1} + t\,k_{22})$$
$$k_{31} = v\sin(\theta_{n-1} + t\,k_{22})$$
$$k_{32} = \omega$$

$$\begin{pmatrix} x_n \\ y_n \\ \theta_n \end{pmatrix} = \begin{pmatrix} x_{n-1} \\ y_{n-1} \\ \theta_{n-1} \end{pmatrix} + \frac{t}{6}\begin{pmatrix} k_{00} + 2(k_{10} + k_{20}) + k_{30} \\ k_{01} + 2(k_{11} + k_{21}) + k_{31} \\ k_{02} + 2(k_{12} + k_{22}) + k_{32} \end{pmatrix}$$

Using these equations, one can now find the (x, y, θ) position of the robot.

Go To Point

One of the most basic uses of position tracking is being able to tell the robot to go to a point. This solved a basic problem with our old algorithm. The old algorithm was based on keeping the robot pointed at the destination and stopping the robot when it came within a certain distance of the objective. Though simple to implement, the distance tracking and aiming involved two distinct operations, which had some corner cases when the robot got close to the destination. The usual manifestation was that the robot would turn slightly as it neared its destination, as it desired to stay pointed at the endpoint. This created some problems, as the robot may not necessarily be pointing in the desired direction.

The new algorithm is very simple. Basically the wheel speed varies by the distance between the current location and the desired location. It is as simple as that.

Example Code

In this section we give the position tracking algorithm in basic c-style form.

```
void positionTracking()
{
    int previousTime = nSysTime;
    int previousLeft = nMotorEncoder(motorA);
    int previousRight = nMotorEncoder(motorB);

    sleep(50);

    while(1)
    {
        int currentTime = nSysTime;

        float t = (currentTime-previousTime);

        int leftE = nMotorEncoder(motorA);
        int rightE = nMotorEncoder(motorB);

        float vl = ((leftE - previousLeft)/t) *(PI/180)*wheelRadius;
        float vr = ((rightE - previousRight)/t)*(PI/180)*wheelRadius;

        float v = (vr+vl)/2;
        float omega = (vr-vl)/wheelBase;

        //RK4

        float k00 = v*cos(robot_TH);
        float k01 = v*sin(robot_TH);
        float k02 = omega;

        float k10 = v*cos((robot_TH + (t/2 * k02)));
        float k11 = v*sin((robot_TH + (t/2 * k02)));
        float k12 = omega;

        float k20 = v*cos((robot_TH + (t/2 * k12)));
        float k21 = v*sin((robot_TH + (t/2 * k12)));
        float k22 = omega;

        float k30 = v*cos((robot_TH + (t * k22)));
        float k31 = v*sin((robot_TH + (t * k22)));
        float k32 = omega;

        robot_X += t/6 * (k00 + 2*(k10 + k20) + k30);
        robot_Y += t/6 * (k01 + 2*(k11 + k21) + k31);
        robot_TH += t/6 * (k02 + 2*(k12 + k22) + k32);

        //normalize heading

        if(robot_TH >= (2*PI))
        {
            robot_TH -= 2*PI;
        }
        if(robot_TH <= (-2*PI))
        {
            robot_TH += 2*PI;
        }

        sleep(50);
        previousTime = currentTime;
        previousLeft = leftE;
        previousRight = rightE;
    }
}
```

Results

What we got was a much more precision in our position tracking code. Like any position tracking, error accumulates and at some point we must localize. However with this code, we can go much longer without having to localize.

References

[1] Wesley Myers and Ethan Myers. "Navigation Using Position Tracking." Proceedings of the National Conference on Educational Robotics. Honolulu, Hawaii. July 10-13, 2007.

[2] Wesley Myers and Ethan Myers. "Position Tracking Using the XBC." Proceedings of the National Conference on Educational Robotics. Norman, Oklahoma. July 7-10, 2006.

[3] Choset, Howie. "Lab 3: Ded Reckoning." *16-311 Introduction to Robotics*. Carnegie Mellon University, 29 Dec 2010. Web. 13 Jun 2011. <http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/current/labs/Lab%203.html>

[4] Kaw, Autar. "Runge-Kutta 4th Order Method for Ordinary Differential Equations." *University of South Florida: Numerical Methods* 13 OCT 2010: 1-7. Web. 13 Jun 2011. <http://numericalmethods.eng.usf.edu/mws/gen/08ode/mws_gen_ode_txt_runge4th.pdf>