Chapter 1 (ctd-ctd)
Robot Kinematics

A brief Introduction

Robots move using their actuators (motors). We can classify actuators in several ways. First, we can consider how they move through space; most actuators we have experienced use rotation, which attached to wheels produce linear motion through space (along a straight or along a curve). But there are also *pure* linear actuators, which move in a straight line. Rotational actuators may be placed in sub-classes; there are *dc-motors* which when connected to a 5Volt source rotate with a certain angular speed, there are *servomotors* where you control their angular speed by giving them pulses of varying widths, and finally there are *stepper-motors* which move through a discrete angle when you give them a pulse. Stepper motors are capable of very accurate (can move small distances) and repeatable motion; they find application in laser-cutters, 3D printers, photocopiers and in robotic surgery.

Recent technology provides us with extended actuator possibilities, one example is shape-memory alloy. Think of a wire which can be long or short depending on its temperature; send a current through it (to heat it up) and it could be short, stop the current (it cools) and it will be long. So, you can control its length electronically. Applications are found in limb prosthetics.

Electro-mechanics of a Stepper Motor

The motor shaft is connected to an armature which rotates; think of this as having small magnets attached. Surrounding the armature are coils which produce a magnetic field when driven by a current. Fig.1 shows a simple example. At the top the armature is held in its position by the south pole coil attracting the north pole on the armature. Then the coil south

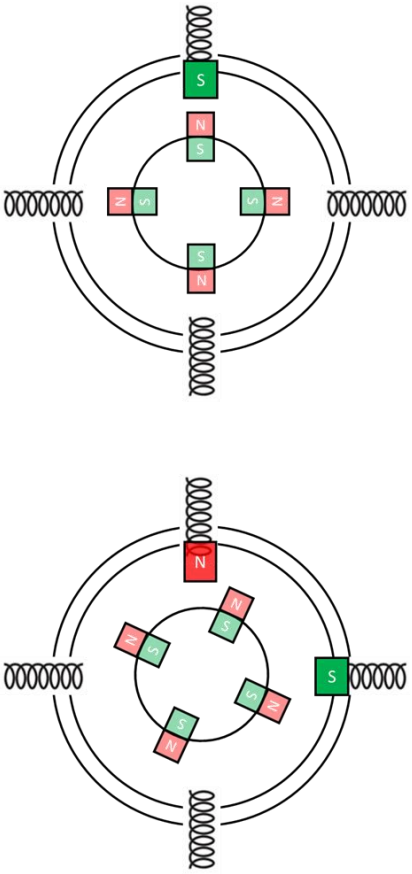

Figure 1 Stepper motor starting one clockwise step driven by magnetic attraction and repulsion

pole is rotated clockwise, and a coil north pole is placed at the top, so the armature will rotate 90 degrees then stop there. So, steppers work by rotating a magnetic field around the armature, dragging it around. Clearly the rotation occurs in steps defined by the number of magnets.

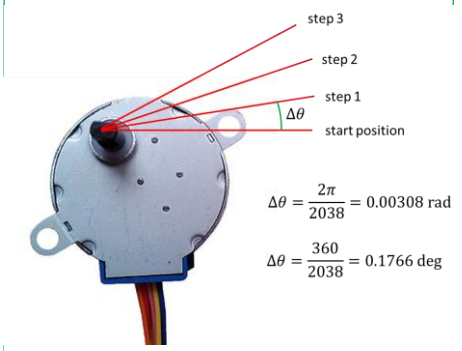

Figure 2 Our stepper showing a few clockwise steps

Our motors also have a gearbox which reduces the angle moved, this results in one revolution having 2038 steps. The angle of a single step (radians) is then

$$\Delta\theta_1 = \frac{2\pi}{2038}$$

which is about 0.0031 radians or 0.177 degrees, quite small. The distance covered by a wheel of radius r for one step is just

$$\Delta s_1 = r\Delta\theta_1$$

for our wheel of radius about 25mm this is around 0.08mm which is quite a small distance. This raises the hope of being able to control the position of a robot very accurately.

There are some limitations where this accuracy may not be realizes. One is related not to the motor, but to the wheels which may slip, especially if the motors are driven from rest to a high angular speed; to avoid this *speed ramping* will be used. Others are related to the motor, the motor may miss a step, especially when its speed is changed dramatically; ramping may help this. The other issue relates to driving two motors at the same time to make the robot move forward in a straight line. Assuming the following function is available (where the arguments are steps to the left and right motors respectively); compare the two coding solutions to get the robot to move *forwards* 100 steps.

`stepMotors(100,100);`	`int i=0;`
	`while(i<100) {`
	`  stepMotors(1,1);`
	`  i++;`
	`}`

The solution on the left will not work, since the internals of the function stepMotors(…) cannot drive both motors at the same time, and almost certainly drive one motor 100 steps then the other motor 100 steps. So, the robot would waddle forward in a series of arcs. While the same is true for the robot on the right, the waddle is limited to single steps and will be barely noticeable.

Forward Motion in a Straight Line

This is straightforward; we must arrange two things, (i) both left and right motors take the same number of steps, (ii) we must drive them with the same speeds. The API has two functions

```
setStepperSpeeds(speedL,speedR);

stepMotors(nL,nR);
```

where **nL** and **nR** are the numbers of steps sent to the left and right motors. The calculation of these is straightforward; if we want to drive the robot **dist** forward then we have

$$n_L = \frac{dist}{\Delta x}, \qquad n_R = \frac{dist}{\Delta x},$$

where Δx (**dx** in code) is the distance travelled for one motor step. Fig 3. summarizes this.

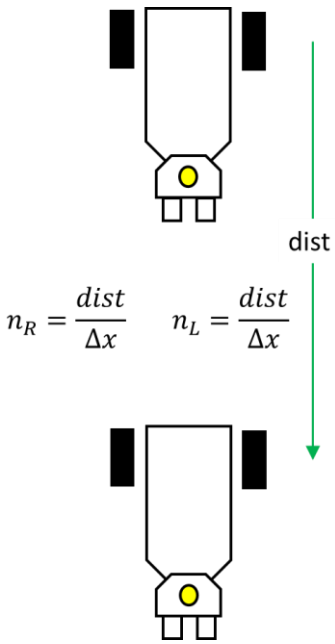

Figure 3 Motion on a straight line; same speeds, same number of steps.

Motion on an Arc

This is a little more complicated to analyze, but the implementation of the algorithm in code is really tricky and requires some thought. Let's say we want to move the robot along an arc of radius R and angle $\Delta\theta_{rob}$ in a clockwise manner. We have seen something similar before; the arrangement is shown in Fig.4. The left wheel moves further than the right wheel, and since both wheels must finish their journeys *at the same time*, the left wheel must move proportionally faster.

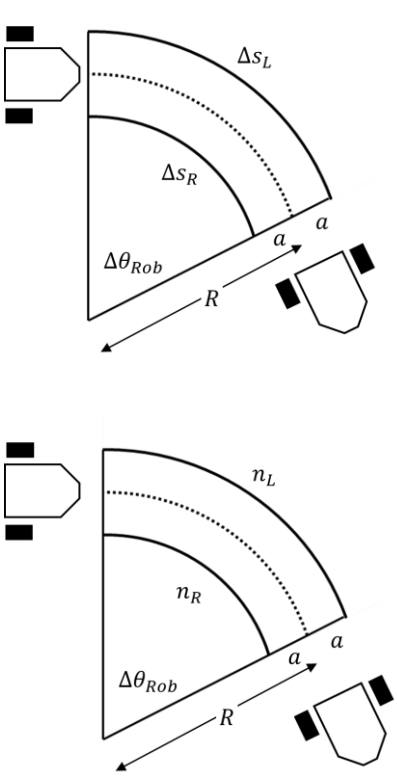

Figure 4 Driving on an arc: Top shows the distances, bottom expresses these in number of left and right steps.

The distances travelled (Fig.4 top) are calculated as usual; for the left wheel we have

$$\Delta s_L = (R + a)\Delta\theta_{Rob} \qquad (1)$$

and for the right wheel

$$\Delta s_R = (R - a)\Delta\theta_{Rob} \qquad (2)$$

Therefore, using the relation

$$n_L = \frac{\Delta s_L}{\Delta x} \qquad (3)$$

and an equivalent one for the right wheel, we find

$$n_L = \frac{\Delta\theta_{Rob}}{\Delta x}(R + a) \qquad (4)$$

and

$$n_L = \frac{\Delta\theta_{Rob}}{\Delta x}(R - a) \qquad (5)$$

Everything on the right side of these equations is known, so we can compute the numbers of steps needed by left and right motors.

As mentioned, both motors need to complete their rotations at the same time, let's call this time Δt. Since distance is velocity time we have

$$\Delta s_L = v_L \Delta t, \qquad \Delta s_R = v_R \Delta t$$

and dividing these

$$\frac{\Delta s_R}{\Delta s_L} = \frac{v_R}{v_L} \qquad (6)$$

and using expression (3) we have

$$\frac{v_R}{v_L} = \frac{n_R}{n_L} \qquad (7)$$

so, the speeds are in proportion to the number of steps taken, in this case the right motor has a lower speed, agreeing with Fig.4.

We need to use expressions (4,5, and 7) in our code. These are used once to calculate the number of steps and the speeds of both motors. The problem is, once we have these values, how to use them to get the motors to turn correctly.

The approach we take (which is successful) is as follows. The right wheel moves a shorter total distance than the left wheel. So, when the right wheel has taken *one step*, we *imagine* that the right wheel has taken a *fractional step*. Of course, it can't but we imagine it can. When the left wheel takes another step, we imagine the right wheel as taken another fractional step.

At some time, the right wheel fractional steps will add up to a whole step, so at this point we make the right wheel take a step. How do we calculate the size of this imaginary step? Well, it is simply the ratio

$$\beta = \frac{n_R}{n_L} \qquad (8)$$

so, when the left motor has finished its n_L steps, the total steps taken by the right motor is

$$\beta n_L = \frac{n_R}{n_L} n_L \qquad (9)$$

which is just n_R, exactly what we want! Fig.5 shows this algorithm expressed as a flow diagram.

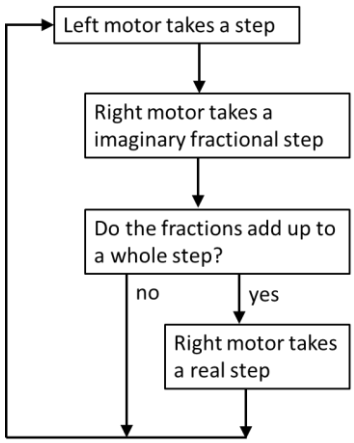

Figure 5 Arc algorithm

One solution to coding this is shown below. This assumes required step numbers and speeds have been computed. The loop runs over the left steps, and the variable **prepR** ('prepare the right motor') accumulates the fractional imaginary right motor steps, **beta** since $n_L = 1$ in (8). When **prepR** is greater than one step, the right motor is stepped. Note when the loop over the left steps is finished, there is a little check to see if there is an outstanding step for the right motor.

```
while (doneL < nL) {  // --------

  stepMotors(1,0);
  doneL += 1;
  prepR += beta;

  if(prepR > 1.0) {
      stepMotors(0,1);
      doneR += 1;
      prepR -= 1.0;
  }

} // End while -----------------

// Check if any outstanding Right
if (prepR > 0.5) {
  stepMotors(0,1);
  doneR += 1.0;
  prepR -= 1.0;
}
```

The variables **doneL** and **doneR** count the steps actually taken while **nL**and **nR** refer to the total steps to take. This code will only work for clockwise turns, and needs to be generalized for all arcs.