2.5 Turning robots

2.5.1 Introduction

Linear robots are the simplest constructions that permit the analysis of basic rules of motion, fundamental control commands and how to integrate sensors to make the robot able to react to stimuli coming from the environment. In spite of their simplicity, they offer a didactic platform to investigate many fundamental concepts and relations, such as space, time, rotation, linear and angular velocity, acceleration, direct and inverse proportionality and others. But robots exploit all their flexibility when moving on a 2-D plane: pupils can design interesting strategies for avoiding obstacles and reaching target positions, use effectively all the information coming from the environment through sensors, reproduce behaviours more similar to those of the every day life. Therefore, the following step of development regards turning robots. With this term we speak about robots able both to move on a straight line and to perform nonlinear trajectories.

The first idea could be to build robots similar to cars or similar to live beings. Unfortunately these types of robots are excessively complex. Cars require critical mechanical subsystems to turn the steering wheels (steering-gear) and to reduce the lateral friction of the non-steering wheels (differential-gear): both such subsystems are very complex and do not add any interesting educational aspects, apart from their mechanical properties. Robots that emulate natural behaviours require several degrees of freedom, that is, several joints and motors, a structure hard enough to be made stable and very complex motions. If this last type of robots is of interest, it is advisable to use already built robots or kits specifically designed for it (‘animaloids’ or ‘humanoids’).

With Lego Mindstorms NXT, a turning robot can be more easily built by connecting one motor to each one of a couple of independent wheels and adding one or two free wheels to the robot to obtain a sufficient stability and to permit simultaneous application of different powers to the two motors. This produces different angular speeds for the two drive-wheels and, therefore, the robot can follow nonlinear trajectories with a limited friction (similarly to trolleys used in supermarkets). A free wheel can be also substituted by a sphere free to rotate in any direction within its seat (think of an old computer mouse).

2.5.2 General remarks and theory

Like in the case of linear robots, we are interested in the general potentialities of the class of robots that we call ‘turning robots’. As mentioned previously, we consider the most significant aspects of the robots of this class to have two independently motorized wheels and the possibility to turn. Several constructions could ex-

hibit these properties (fig. 2.5.1). In the following text, we will focus attention only on the motorized wheels.

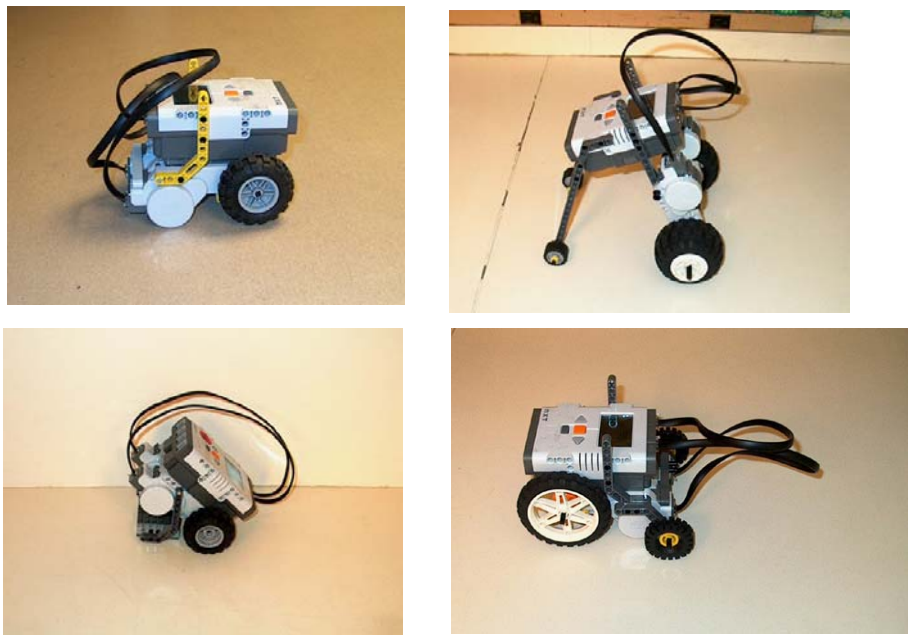

Fig. 2.5.1 *– Turning robots*

Considering positive a clockwise rotation of the wheels and supposing that this rotation makes the robot move forward, if $wheel_1$ has an angular speed ω_1 and $wheel_2$ has an angular speed ω_2, with $\omega_2 \geq \omega_1$ measured in rad/s, it can be easily verified that the robot turns right and the two wheels draw two concentric circular trajectories (fig. 2.5.2). Let r be the common radius of the wheels, d the (fixed) distance between the two wheels, in a short motion of Δt time units the two wheels track an arc of circle long respectively $\Delta A_1=\omega_1 r \Delta t$ and $\Delta A_2=\omega_2 r \Delta t$. Let R be the distance from $wheel_1$ and the hypothetic centre of the two arcs of circle, and $\Delta\Theta$ the angle corresponding to the two arcs, it results in:

$$\Delta A_1 = \omega_1 \, r \, \Delta t = r \, \Delta\theta_1 = R \, \Delta\Theta \qquad (2.5.1)$$

$$\Delta A_2 = \omega_2 \, r \, \Delta t = r \, \Delta\theta_2 = (R+d) \, \Delta\Theta \qquad (2.5.2)$$

and therefore:

$$\Delta\Theta = \omega_1 \, r \, \Delta t / R = \omega_2 \, r \, \Delta t / (R+d) \qquad (2.5.3)$$

$\Omega = \Delta\Theta / \Delta t$ = (the 'angular speed' of the robot) =

$= \omega_1 r / R = \omega_2 r / (R+d)$ (2.5.4)

$\omega_1 / R = \omega_2 / (R+d)$ (2.5.5)

$R = d\ \omega_1 / (\omega_2 - \omega_1)$ (2.5.6)

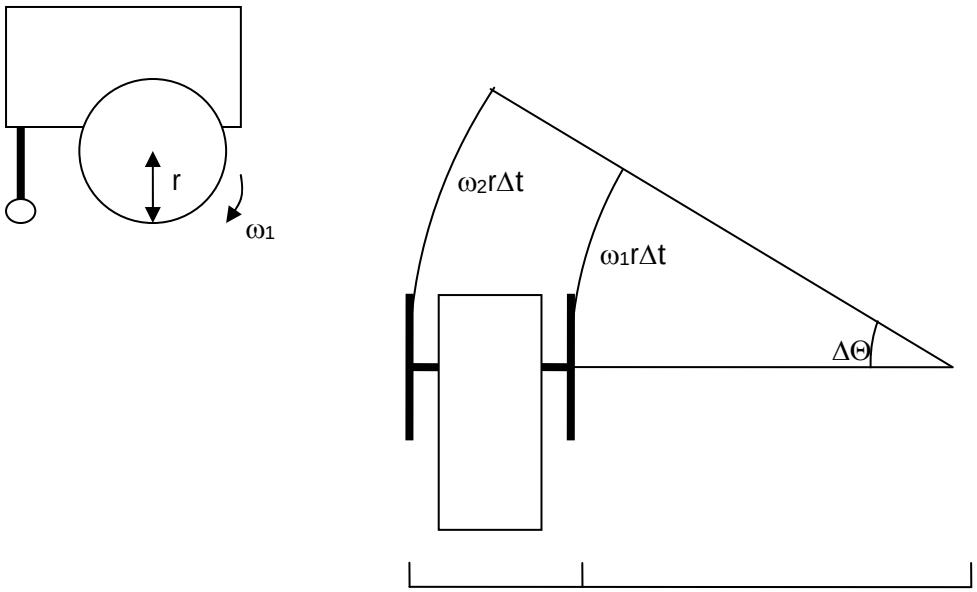

***Fig. 2.5.2** – Turning motion*

Therefore, the radius R does not depend on the radius r of the wheels, but only on the angular speeds and the inter-wheel distance. When $\omega_2 \rightarrow \omega_1$, then $R \rightarrow \infty$, the motion is straight and the equation (2.5.1) becomes simply $\Delta A = \omega\ r\ \Delta t = r\ \Delta\theta$. When $\omega_1 = 0$ then $R = 0$ and the robot pivots around wheel$_1$ that is not moving. When $\omega_2 = -\omega_1$ ($\omega_2 > 0$), it results in:

$R = - d /2$ (2.5.7)

and the robot pivots around the middle point between the two wheels, regardless of the value of the applied power. The angular speed of and the angle performed by the robot (ignoring the sign) are in this case:

$\Omega = 2\ |\omega_1|\ r / d$ (2.5.8)

$\Theta = |\theta_1|\ r / R = 2\ |\theta_1|\ r / d$ (2.5.9)

Now the problem is to provide suitable commands to impose the two ω_1 and ω_2 angular speeds required by the desirable trajectory. First of all, we recall that, for a

straight motion with one motor, the NXT-G Motor command actually provides a speed control, when the 'Motor power control' option is active (fig. 2.5.3).

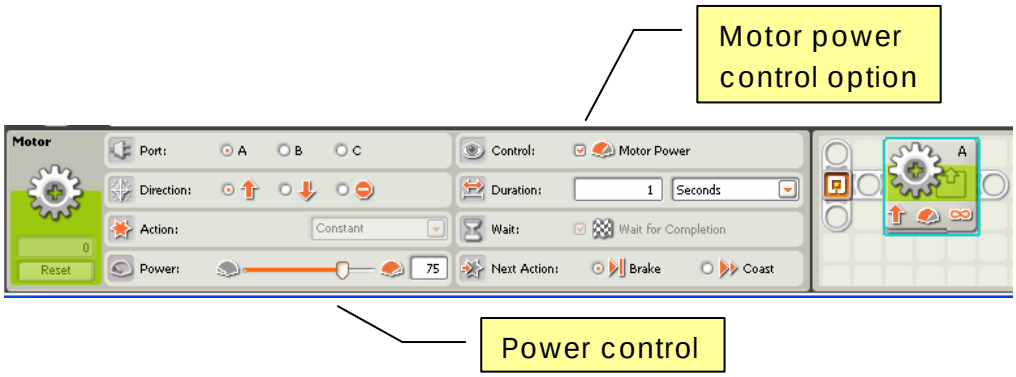

***Fig. 2.5.3** – The Motor block*

Empirically, it is easily possible to determine the relation between speed and power. We did this with a program that samples many times the rotation sensor integrated in the motor applying different powers with the 7.5 V rechargeable battery (for a description of this technique see section 2.5.7.1). Supposing that a negligible load is applied to the motor, this relation appears linear (fig. 2.5.4).

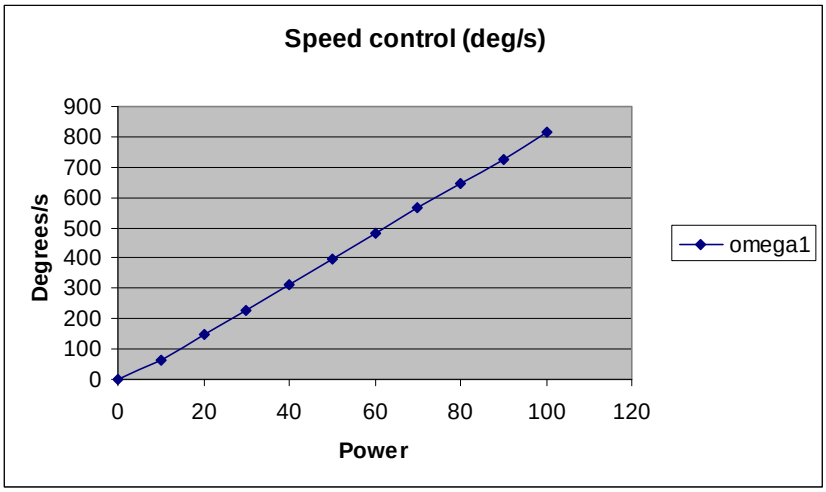

***Fig. 2.5.4** – Angular speed versus Power*

Let $\omega = k_{\omega P} P$ be the relation with P the applied power in percentage, as usually indicated: the constant of proportionality $k_{\omega P}$ has been determined as about 8.15 degrees/s, in the following formula:

$\omega = k_{\omega P} P = 8.15 P$ deg/s

Given that 1 deg = $2\pi/360$ rad and 1 deg/s = 1/6 rpm,

it results in $\omega = 0.1422 P$ rad/s = $1.36 P$ rpm (2.5.10)

From (2.5.6) we obtain also:

$$R = d\,\omega_1 / (\omega_2 - \omega_1) = d\,P_1 / (P_2 - P_1) \quad (2.5.11)$$

With a 9V battery $k_{\omega P} > 9.5$ degrees/s: thus, the actual constant depends heavily on the battery charge: it is advisable to replicate the analysis with the suggested method so that the teacher (and the students) can evaluate this constant with their own robots. With a significant load, the linearity in the relation speed-power is limited to the lower powers, whereas for higher powers and high load the control saturates and from one point to the maximum of power the speed remains almost constant.

With two motors, the first possibility is to provide separate commands to them with two successive Motor blocks, the first one necessarily not waiting for completion, in order to leave a very short (and negligible) time separation between the two commands.

Just to have a quantitative idea of the turning motion, taking from equation (2.5.10) 0.1422*P rad/s as the valid formula giving the angular speed with a rechargeable battery, mounting the standard intermediate size wheels of the NXT kit which have a diameter of $2r$=56 mm, and supposing to want to turn right with a radius R=100 mm and a distance d=70 mm, every second the robot performs:

- An internal arc of radius 100 mm and length of $\theta_1 = 28\ \omega_1$ mm (angular speed in rad/s) = $3.98*P_1$ mm
- An external arc of radius 170 mm and length of $\theta_2 = 28\ \omega_2$ mm (angular speed in rad/s) = $3.98*P_2$ mm
- It must hold: $P_1 / (P_2 - P_1) = R/d = 10/7 \Rightarrow P_2 = 1.7\ P_1$

Setting for example $P_1 = 30$ and $P_2 = 1.7\ P_1 = 51$, in 3 seconds the robot draws two arcs with:

$A_1 = 3.98*30*3 = 358.2$ mm $\qquad A_2 = 3.98*51*3 = 608.9$ mm

There is a second possibility integrated in the NXT-G Move block (fig. 2.5.5): this block permits to control concurrently two motors with a further parameter called ‘steering’ that you can set through a specific slider or, as usual, connecting an input data wire. This parameter controls the ratio between the angular speeds of the two motors, but unfortunately such relation is not well documented and it must be analyzed empirically. Let us assume that, when the Direction parameter in the Move block is set to straight forward, i.e. up arrow, in case of a straight motion the angu-

lar speed of the motors is positive. It is known that the steering parameter can vary from -100 and +100: 0 corresponds to a straight motion ($\omega_1 = \omega_2 > 0$), negative values are set to steer towards left motor ($\omega_1 > \omega_2$), positive values to steer towards right motor ($\omega_1 < \omega_2$). With the extreme values the robot pivots around the middle point ($\omega_2 = -\omega_1$, $\omega_1 > 0$ with steering=-100, $\omega_1 < 0$ with steering=+100).

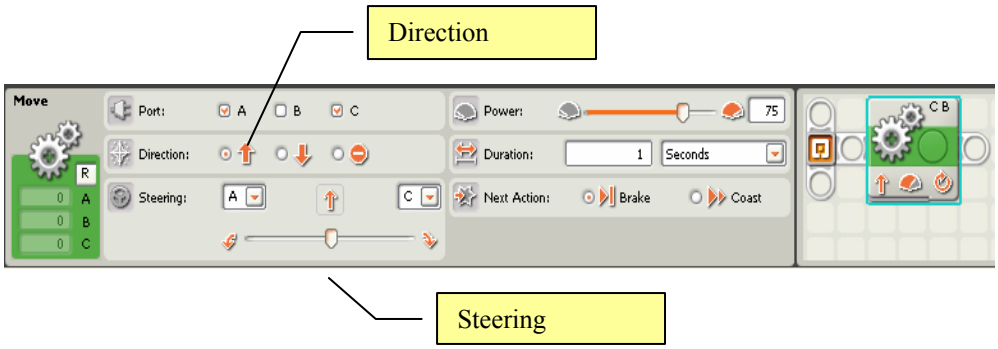

***Fig. 2.5.5** – The Move block*

To study the intermediate values, we initially prepared an observation experiment with a robot provided with a pen to draw on a paper sheet during the motion (fig. 2.5.6). Then we measured the radius of the drawn circles with different steering values, obtaining the results in fig. 2.5.7.

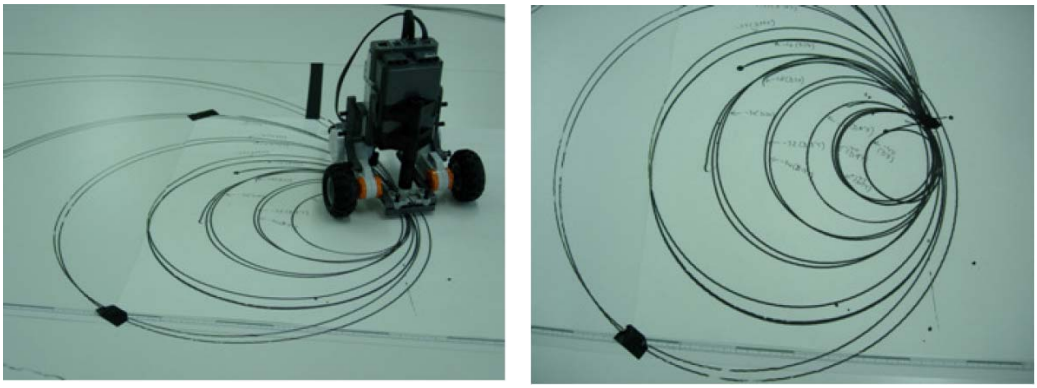

***Fig. 2.5.6** – The steering parameter*

The imprecision of this first approach counseled us to prepare an experiment similar to the previous one, used to measure the speed-power relation, to evaluate the average angular speed obtained with a fixed power (P=70) and a steering value varying from 0 to +100 (see section 2.5.7.2). The result is summarized in fig. 2.5.8.

As you can see in fig. 2.5.8, increasing the steering maintains almost constant ω_2 and decreases linearly ω_1 untill $\omega_1 \approx -\omega_2$. The ratio $R/d = \omega_1 / (\omega_2 - \omega_1)$ decreases

apparently in inverse proportion in respect of the steering value, as shown in fig. 2.5.9.

Steering	Diameter	Radius
44	70	35
42	95	47,5
40	100	50
38	115	57,5
36	135	67,5
34	155	77,5
32	175	87,5
30	200	100
28	240	120
26	270	135
24	310	155
22	365	182,5
20	410	205
18	470	235
16	540	270
14	680	340
12	750	375
10	1140	570

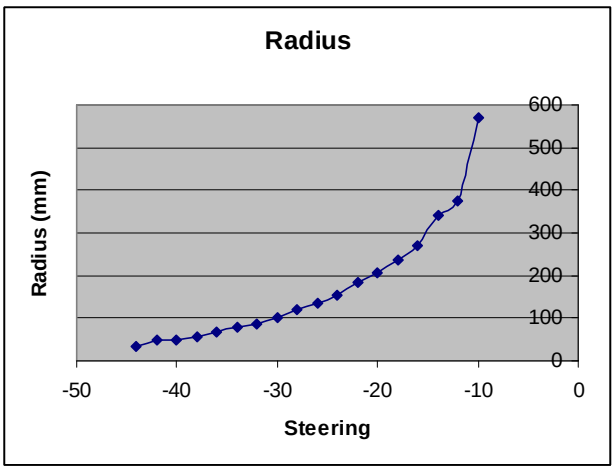

***Fig. 2.5.7** – Relation between Steering and Radius*

Steer.	ω_1	ω_2	ω_1 / (ω_2 - ω_1)	(-st+40) /st
0	564,7239	564,2873	-1293,5	
10	419,4042	555,563	3,080258	3
20	283,9617	554,582	1,0493	1
30	141,6721	554,0275	0,343568	0,333333
40	4,354432	552,7787	0,00794	0
50	-122,522	552,9425	-0,18139	-0,2
60	-277,014	552,4995	-0,33395	-0,33333
70	-414,975	551,408	-0,42941	-0,42857
80	-551,589	551,1325	-0,50021	-0,5
90	-574,262	550,4984	-0,51056	-0,55556
100	-573,674	550,3165	-0,51039	-0,6

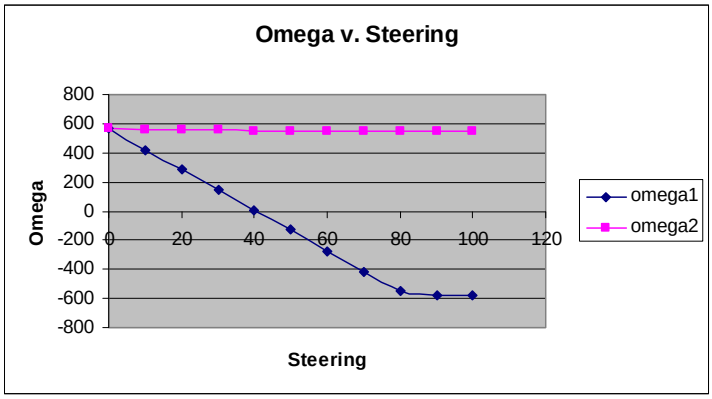

Fig. 2.5.8 *– Relation between angular speeds and steering*

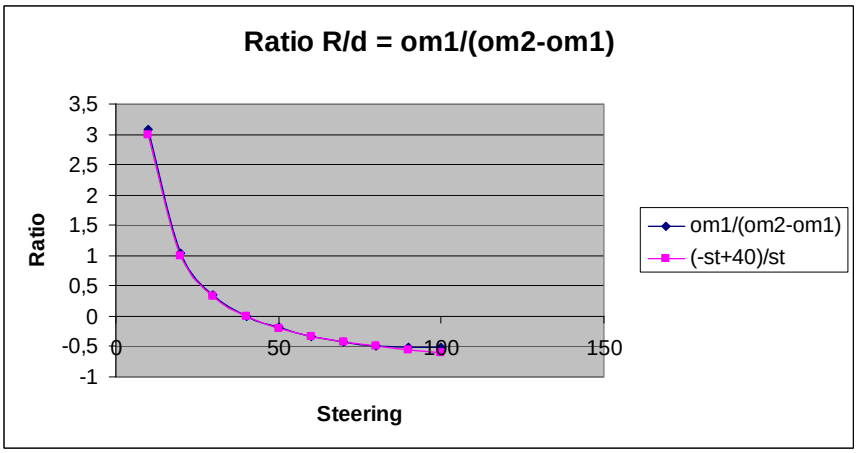

Fig. 2.5.9 *– Relation between R/d and steering*

In fact, assuming the following (S=steering):

$$\omega_1 = k_1 S + k_2 \qquad (2.5.12)$$

$$\omega_2 = k_3 \qquad (2.5.13)$$

$$R/d = \omega_1 / (\omega_2 - \omega_1) = (k_1 S + k_2) / (k_3 - k_1 S - k_2) =$$

$$= (S + k_A) / (-S + k_B) \qquad (2.5.14)$$

The estimation of these coefficients from the experimental data gives:

$$\omega_1 = -13.9 S + 558 \qquad (2.5.15)$$

$$\omega_2 = 552 \qquad (2.5.16)$$

$$k_A = k_2 / k_1 = (558 / -13.9) = -40 \qquad (2.5.17)$$

$$k_B = (k_3 - k_2) / k_1 = ((552 - 558)/ -13.9) \approx 0 \qquad (2.5.18)$$

$$R/d = (-S + 40) / S \qquad (2.5.19)$$

and the plot is an arc of an hyperbola with the vertical asymptote S=0 and the horizontal asymptote R/d=-1. To confirm the last result, in fig. 2.5.9 the obtained function is plotted together with the experimental data.

To conclude, even though the presented inspection made possible to evaluate the trajectory performed using a Move block with a given steering, it is even simpler to control the turning robot with two Motor blocks.

2.5.3 Didactical consideration

The following table summarizes the main problems that can be put when analyzing a turning robot.

	Problem	Pseudo-code of the action	Comments
1	How to build a robust turning robot?		
2	How to make the robot move forward with a given speed for a certain time?	GoForward (speed, time)	Just as for linear robots, the linear speed depends on the radius of the wheels. The power to be applied to the motors (i.e. the angular speed of the two wheels) is the same
3	How to make the robot move forward for a given distance with a given speed?	MakeStep (distance, speed)	Just as for linear robots, from the given parameters you should calculate the angle to be performed by both motors and the power to be applied to them.

4	How to make the robot pivot with a given angular speed Ω for a time t?	Pivot(Ω, t)	We know that for the robot angular speed it holds $\Omega=2\ \omega\ r / d \Rightarrow$ $\omega= \Omega\ d / (2\ r)$ $P = \omega / k_{\omega P} =$ $= \Omega\ d / (2\ r\ k_{\omega P})$
5	How to make the robot pivot for a given angle Θ with a given angular speed Ω?	Pivot(Θ, Ω)	We know that for the angle made by the robot it holds: $\Theta=2\ \theta\ r / d \Rightarrow$ $\theta= \Theta\ d / (2\ r)$ θ = motor angle$P = \omega / k_{\omega P} =$ $= \Omega\ d / (2\ r\ k_{\omega P})$
6	How to make the robot follow with its inner wheel (see fig. 2.5.2) an arc of circle of a given angle Θ, radius R and angular speed Ω?	Turn(Θ, R, Ω)	It is necessary to use the general formulas calculated in the previous section. The duration of the motion can be calculated in terms of θ (motor angle).

In the problems shown above, the wheel radius *r* and distance *d* have been conceptually considered constant parameters instead of function parameters because they are related to the building details. Their values will obviously influence the calculations internal to the functions to be realized.

There is a very general problem that might be mentioned at this point. The current standard NXT firmware does not support calculation and variables for fractional

numbers (neither fixed nor floating point). It supports only 32-bit signed integer that can contain values between -2^{31} = -2147483648 and $+2^{31}-1$ = +2147483647. This must be seriously considered, when you make complex calculations. Two general rules addressing this problem are:

- Scale the constants to obtain numbers not too large (to avoid overflow the capacity of an integer) and without significant fractional digits.
- Leave an operation that implies truncation (the division and, if installed as an extension block, the square root) as the latest operation, when possible.

For example, if you have to calculate the following expression (in general, some values of the expression could be in variables):

124 * 13.2 / (5.7 – 2.5)

do actually the following ($\lfloor x \rfloor$ stands for floor(x), i.e. the greatest integer ≤ x):

57-25 = 32 -> var1

$\lfloor$(124 * 132) / var1$\rfloor$ = 511

whereas $\lfloor$124 / 32$\rfloor$ * 132 results in = 3 * 132 = 396

As a further example, let us now propose the general solution for the problem 6 above. We would realize the function *turn(Θ, R, Ω)* with:

Θ robot angle in degrees

R motion radius in mm

Ω robot angular speed in degrees/s

We must now calculate the requested powers P_1 and P_2 to be respectively applied to the motors and the wheel angles θ_1 and θ_2 to be performed. We use now formulas (2.5.4) and (2.5.10):

$$P_1 = \omega_1 / k_{\omega P} = \Omega R / (r\, k_{\omega P}) \qquad (2.5.20)$$

$$P_2 = \omega_2 / k_{\omega P} = \Omega (R+d) / (r\, k_{\omega P}) \qquad (2.5.21)$$

$$\theta_1 = \Theta R / r \qquad (2.5.22)$$

$$\theta_2 = \Theta (R+d) / r \qquad (2.5.23)$$

If d=80 mm, r=28 mm, $k_{\omega P}$ = 8.15 degrees/s, and you want Θ=40 degrees, R=200 mm, Ω=20 degrees/s we obtain:

$P_1 = \Omega R / (28 * 8.15) = 20 * 200 / 228.2 = 17.52$

$P_2 = \Omega (R+80) / (28 * 8.15) = 20 * 280 / 228.2 = 24.54$

$\theta_1 = \Theta R / 28 = 40 * 200 / 28 = 285.71$ degrees

$\theta_2 = \Theta\ (R+80) / 28 = 40 * 280 / 28 = 400$ degrees

We can check that:

$P_1 / (P_2 - P_1) = 17.52 / (24.54\text{-}17.52) = 2.495 \approx 2.5$

and

$R/d = 200/80 = 2.5$

Now, if we take into account the imprecision due to the integer calculations, we obtain:

$P_1 = \lfloor \Omega\ R\ 10/\ 2282 \rfloor = \lfloor 20 * 200 * 10 / 2282 \rfloor = \lfloor 40000 / 2282 \rfloor = 17$

$P_2 = \lfloor \Omega\ (R+80)\ 10/\ 2282 \rfloor = \lfloor 20 * 280 * 10 / 2282 \rfloor = \lfloor 56000 / 2282 \rfloor = 24$

$\theta_1 = \lfloor \Theta\ R / 28 \rfloor = \lfloor 8000 / 28 \rfloor = 285$ degrees

$\theta_2 = \lfloor \Theta\ (R+80) / 28 \rfloor = \lfloor 11200 / 28 \rfloor = 400$ degrees

Therefore, in addition to the intrinsic imprecision of the rotation sensor of the motor, the calculation causes a further error:

$\omega_1 = k_{\omega P}\ P_1 = 8.15 * 17 = 138.55$ $\Omega = \omega_1\ r / R = 138.55 * 28 / 200 = 19.397$

$\omega_2 = k_{\omega P}\ P_2 = 8.15 * 24 = 195.6$ $\Omega = \omega_2\ r / (R+d) = 195.6 * 28 / 280 = 19.56$

$\Theta = \theta_1\ r / R = 285 * 28 / 200 = 39.9$

$\Theta = \theta_2\ r / (R+d) = 400 * 28 / 280 = 40$

So we can conclude that the introduced errors are negligible. As a final remark, let us take into account also the imprecision in determining the two 'intrinsic' parameters *r* and above all *d* with a direct measure: it is advisable to use a first estimation of the two parameters for the calculations and then validate the parameters through a direct experience with the robot, tuning the initial estimation accordingly to the obtained motion. In particular, we can suggest to make the robot to pivot for a given angle (e.g. 360 degrees) with a simple program and to tune the factor $d/2r$ that multiplies the motor angle θ in order to obtain the desired result.

2.5.4 Building Instructions for the Tiny Turtle

We now present a very simple instance of the turning robot class, initially not provided with sensors (apart from, obviously, the integrated rotational sensor), for which we will define 4 basic motion commands as in the case of the well known Logo turtle (fig. 2.5.10). For this reason we call it *tiny turtle*. Actually, the presented layout is not mandatory: any instance of the class could be easily adapted to

this use, being based on two separately controlled motors. So, the building instructions are quite generic and aim at providing the simplest necessary robot.

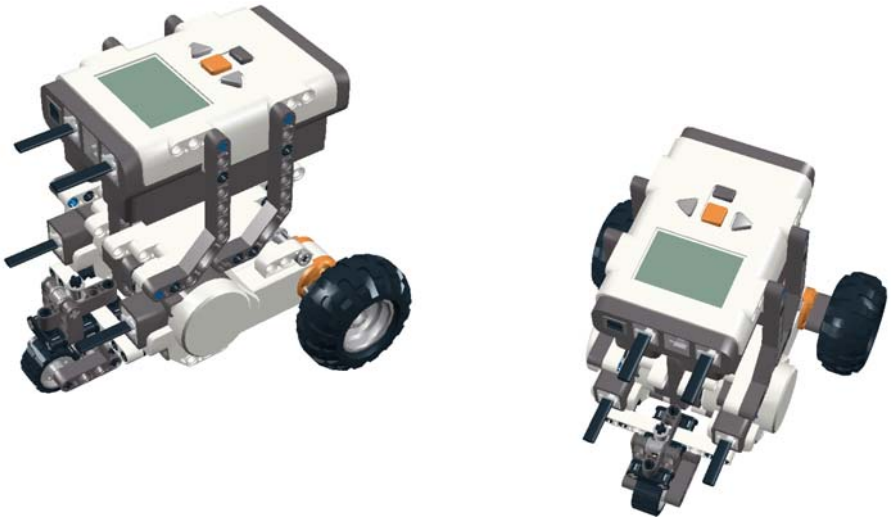

Fig. 2.5.10 – *The Tiny Turtle*

2.5.5 Programming the Basic Commands for the Tiny Turtle

We must prepare 4 basic commands (forward, backward, left and right) in terms of building blocks to permit the user to define in NXT-G the equivalent of a (simplified) Logo program. Therefore, we use the 'My block' technique of NXT-G to define such commands. Forward (Fd) and Backward (Bk) commands are straight motions and request a parameter specifying a distance measured in conventional units called '*steps*'. Left (Lt) and Right (Rt) commands make the robot pivot respectively counterclockwise and clockwise for an angle usually measured in degrees and specified by a command parameter. The duration of each movement is not relevant for a simple Logo-like turtle: thus, we could accept a standard speed profile and use an arbitrary power value even though we suggest to choose an intermediate value producing not too fast but also not too slow movements.

Recalling the formulas obtained in section 2.5.3 for Fd(distance) and Bk(distance) commands, we must apply the same power to two successive Motor blocks or use one Move block with the steering set to 0. Apart from an initial and a final very short transition, the robot moves with a rather constant speed; from (2.5.1) we know that the common motor angle θ should be set to $k_{d\theta}$**distance* (*distance* measured in steps) with a reasonable constant $k_{d\theta}$. If there is no other specific reason to force this constant to some value, we suggest to set $k_{d\theta}$=1, i.e. 1 step = r 2 π / 360 mm with r the radius of the wheels in mm, corresponding to the minimum measurable rotation of the wheels of 1 degree.

The commands Lt(angle) and Rt(angle) must make the robot pivot for an angle Θ=*ang*. Because in this case from (2.5.9) it holds:

$\Theta = 2\ |\theta_{1,2}|\ r / d$

we must impose:

$|\theta_{1,2}| = ang\ d / 2\ r$

with *d* the distance between the two wheels.

Now, the description of the Lt block follows in the usual graphical form and then with the textual pseudo-code NXT-GTD. The other command blocks can be similarly programmed.

In a new (sub) program we add a numeric Variable block (the name for the moment is irrelevant, it is just a placeholder), representing the angle input parameter (fig. 2.5.11). Then, we must multiply for *d* and divide for 2*r*: it is very probable that these two values are integers with sufficient precision if measured in mm. Fig. 2.5.12 shows these operations with *d*=124 mm and 2*r*=56 mm. Notice the data wires necessary to connect the inserted blocks with each other.

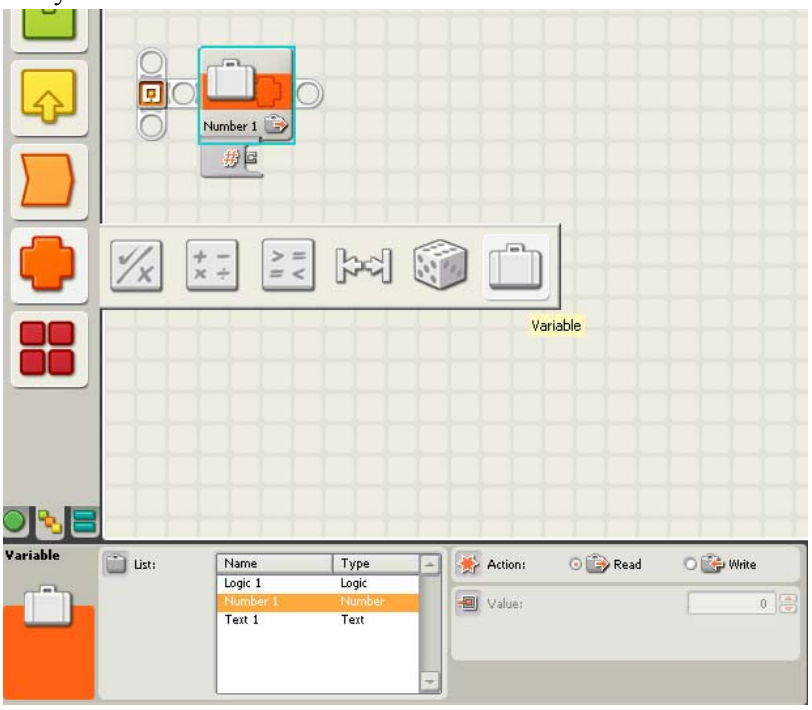

Fig. 2.5.11 *– The Lt block (I)*

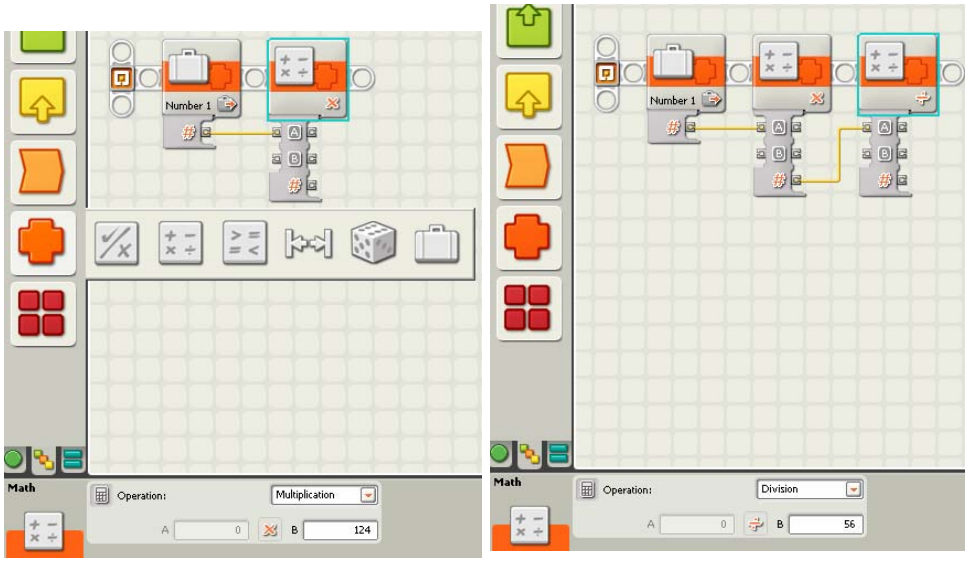

***Fig. 2.5.12** – The Lt block (II) (III)*

If either of the two is not an integer value, it is necessary to scale such values with the same constant K>1 for maintaining a sufficient precision:

$D = \lfloor d\ K \rfloor$ $R = \lfloor 2\ r\ K \rfloor$

and then calculate $\lfloor angle*D/R \rfloor$. For example, if angle=45, d=83.5 mm. and r = 28 mm:

$\lfloor 45*\lfloor 83.5 \rfloor / 56 \rfloor = \lfloor 3735 / 56 \rfloor = 66$

With K=10:

$\lfloor 45*835 / 560 \rfloor = \lfloor 37575 / 560 \rfloor = 67$

Now, if we decide to control the motors with two distinct Motor blocks, with an intermediate power of 50(%) and assuming A and C the two respective control ports, the two new blocks have the same calculated absolute value in degrees for the Duration parameter but opposite Direction, and only the first one does not wait for completion (fig. 2.5.13).

To define this 'code' as a new block using the 'My block' feature of NXT-G, select the last 4 blocks (Multiply, Division, MotorA and MotorC) and click on the 'Create My Block' button (or choose the equivalent menu item), assign the name Lt to the new user block (fig. 2.5.14 a) and a representative icon (fig. 2.5.14 b).

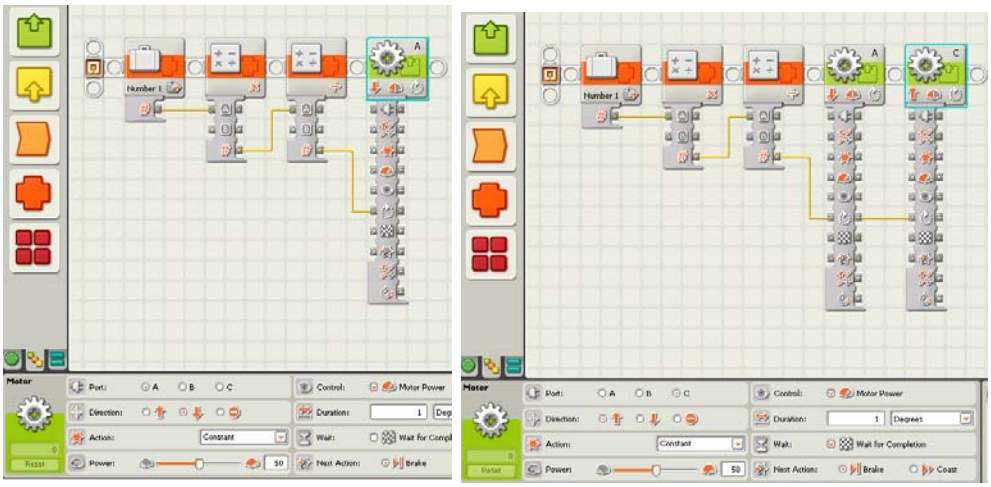

***Fig. 2.5.13** – The Lt block (IV) (V)*

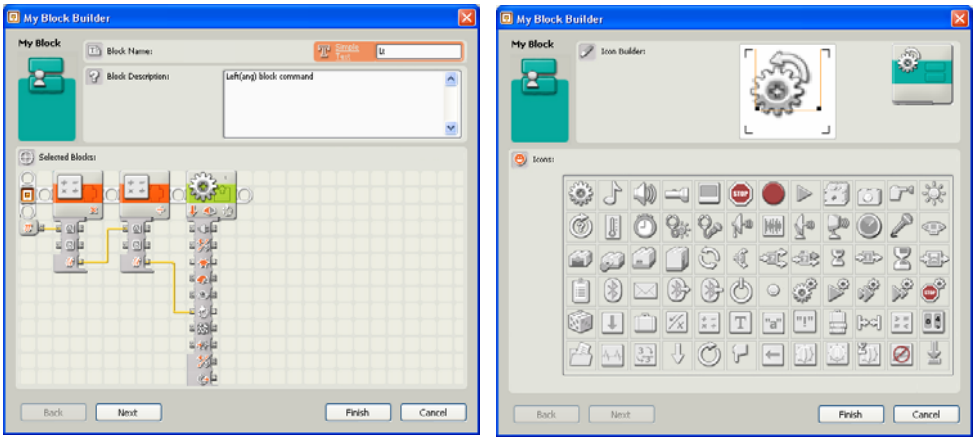

***Fig. 2.5.14** – The Lt block a (VI) b (VII)*

By double-clicking the Lt block, you can edit its definition and, specifically, the name of the input parameter, which appears now as an input connection ‘angle’ on which you can attach a data wire to provide the angle of rotation (fig. 2.5.15).

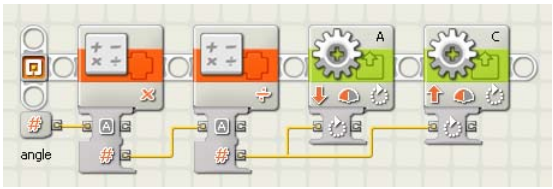

***Fig. 2.5.15** – The Lt block (VIII)*

This program expressed in NXT-GTD is as follows.

```
Lt(angle:degrees) : []
Mul1: MathOp(Type=MUL, A=Lt.angle, B=124)
Div1: MathOp(Type=DIV, A=Mul1.Res, B=56)
Mt1: Motor(Port=A, Dir=BK, Act=CONST, Pwr=50, PwrCtrl=ON,
  Dur=Div1.Res.DEG, Wait=OFF)
Mt2: Motor(Port=C, Dir=FD, Act=CONST, Pwr=50, PwrCtrl=ON,
  Dur=Mt1.Dur.DEG, Wait=ON, Next=BRK)
```

Once the 4 basic turtle commands are programmed, we can define a piece of code equivalent to some Logo code. For example, to make the turtle 'draw' a regular polygon of n edges each long l steps, in Logo we would say:

repeat n [fd l right 360 / n]

(in the turtle geometry, we know that 360/n is the amount of degrees the turtle must turn to have an internal angle of (n-2)*180/n degrees, which is required for a regular polygon).

In NXT-G the equivalent is shown in fig. 2.5.16, whereas the NXT-GTD version follows:

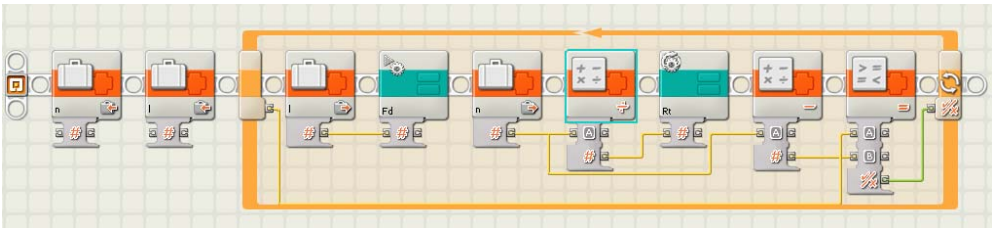

Fig. 2.5.16 *– Logo-equivalent code example*

```
VarDecl(Name=n, Type=NUM)
VarDecl(Name=l, Type=NUM)
Var(Name=n.NUM, Act=WR, Val=5)  -- n=5 pentagon
Var(Name=l.NUM, Act=WR, Val=300) -- l=300 steps
Loop1: Loop(Ctrl=LOGIC, Until=Cmp1.Res, ShowCnt=ON) [
        Vl: Var(Name=l.NUM, Act=RD)
        Fd(Vl.Val)
        Vn: Var(Name=n.NUM, Act=RD)
```

```
        D1: MathOp(Type=DIV, A=360, B=Vn.Val)
        Rt(D1.Res)
        S1:MathOp(Type=SUB, A=Vn.Val, B=1)
        C1:CmpOp(Type=EQ, A=Loop1.Cnt, B=S1.Res)
Loop1]
```

2.5.6 Some didactical Issues for the Tiny Turtle example

This example offers several 'foods for thought' that the teacher can exploit. First of all, the geometrical theory, which is on the basis of the definition of the 4 fundamental motion commands, is present in this section. Other interesting issues come from the 'turtle geometry' and regard the drawing of known figure (like the n-edge regular polygon described above), the simulation of physical phenomena, and others. Unfortunately, one of the most interesting aspects in Logo, the recursion, is (apparently) not supported by the NXT firmware.

During a preparatory stage, the teacher can propose the general problem of turning robots to the students and discuss with them if the building of a car with a steering sub-system is realizable. The teacher should help the students to identify the simplest two-motorized solution, also on the basis of real experiences (supermarket trolleys, agricultural machines, segways, etc.). A further investigation can make the student discover some general relations based on geometry and trigonometry. Then, the teacher suggests the realization of the turtle as an example of unsensorized turning robots. The discussion goes on about how to realize the 4 basic commands. After the actual realization of the turtle, the teacher suggests to prepare some Logo-style demo programs (for example, to verify that the turtle can follow a polygon already drawn on the plane). Finally, the students can discuss the correspondence between the theory and the practical realization. Some suggestions could be stimulated to add useful sensors to the turtle and to provide a reasonable way to use them.

2.5.7 Appendix

2.5.7.1 Investigation of the relation power - ω

The program to evaluate the relation between the power (0 .. 100) applied to both motors (steering=0) and the common angular speed obtained is based on the data logger schemata of section 2.7 and presented in fig. 2.5.17.

The equivalent NXT-GTD description follows.

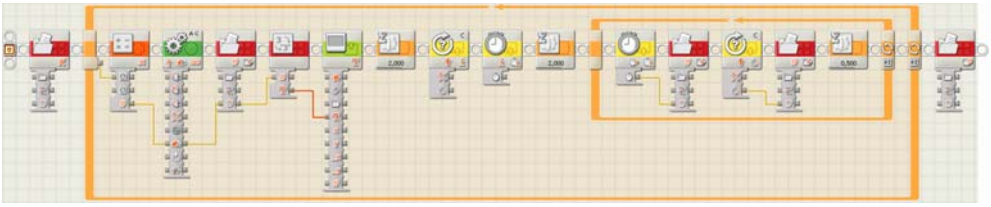

***Fig. 2.5.17** – ω vs. P 'plotting' program*

```
File(Act=DL, Name=omega)
Lo1: Loop(Ctrl=COUNT, Until=11, Dis=ON) [
  Mu1:  MathOp(Type=MUL, A=Lo1.Cnt, B=10)
  Mv1:  Move(Ports=AC, Dir=FD, StLt=A, StRt=C, Steer=0, Pwr=Mu1.Res,
        Dur=FOREVER)
  F1:   File(Act=WR, Name=omega, Type=NUM, Val=Mv1.Pwr)
  Nt1:  N2Txt(Num=F1.Num)
        Display(Type=TXT, Clr=ON, Txt=Nt1.Txt, PosX=8, PosY=32,
          Line=4)
        Wait(Ctrl=TIME, Until=2)  -- wait for motion stabilization
        RotSens(Port=C, Act=RESET, Cmp=??)
        Timer(Num=1, Act=RESET, Cmp=??)
        Wait(Ctrl=TIME, Until=2)  -- wait for motion stabilization
  Lo2:  Loop(Ctrl=COUNT, Until=10, Dis=OFF) [
        Ti1:    Timer(Num=1, Act=READ, Cmp=??)
        F2:     File(Act=WR, Name=omega, Type=NUM, Val=Ti1.Val)
        Ro1:    RotSens(Port=C, Act=READ, Cmp=??)
        F3:     File(Act=WR, Name=omega, Type=NUM, Val=Ro1.Val)
                Wait(Ctrl=TIME, Until=0,5)  -- wait for next sample time
        Lo2]
Lo1]
File(Act=CL, Name=omega)
```

The program repeats 11 times, applying orderly powers 0, 10, 20, . . ., 100, the acquisition of 10 samples of the motor angular position through its integrated rotation sensor. In the internal file "omega" such data appear as follows:

. . .

20 -- power applied

2002 -- sample time 1

293 -- sample value 1

2514 -- sample time 2

366 -- sample value 3

. . .

6606 -- sample time 10

963 -- sample value 10

30 -- power applied

. . .

The ω angular speed for a certain power is estimated as the average value of the 9 ratios:

$v_i - v_{i-1} / t_i - t_{i-1}$ with $1 \leq i \leq 10$

2.5.7.2 Investigation of the relation steering - ratio $\omega_1 / (\omega_2 - \omega_1)$

Similarly, as with the previous relation, we prepared a sampling program evaluating both average angular speeds at a sequence of different steering values (fig. 2.5.18).

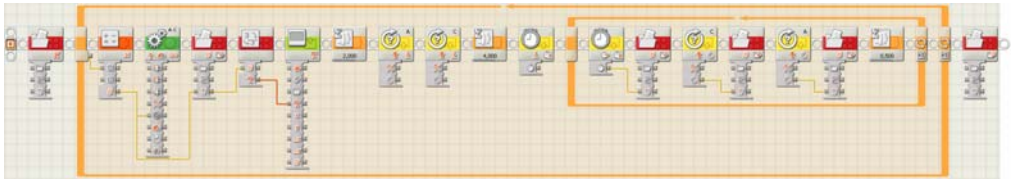

Fig. 2.5.18 – *ω vs. P 'plotting' program*

The equivalent NXT-GTD description is as follows:

```
File(Act=DL, Name=steer)
Lo1: Loop(Ctrl=COUNT, Until=11, Dis=ON) [
```

```
Mu1: MathOp(Type=MUL, A=Lo1.Cnt, B=10)
Mv1: Move(Ports=AC, Dir=FD, StLt=A, StRt=C,
          Steer=Mu1.Res, Pwr=70, Dur=FOREVER)
F1:  File(Act=WR, Name=steer, Type=NUM, Val=Mv1.Steer)
Nt1: N2Txt(Num=F1.Num)
     Display(Type=TXT, Clr=ON, Txt=Nt1.Txt, PosX=8, PosY=32,
      Line=4)
     Wait(Ctrl=TIME, Until=2)  -- wait for motion stabilization
     RotSens(Port=A, Act=RESET, Cmp=??)
     RotSens(Port=C, Act=RESET, Cmp=??)
     Wait(Ctrl=TIME, Until=4)  -- wait for motion stabilization
     Timer(Num=1, Act=RESET, Cmp=??)
Lo2: Loop(Ctrl=COUNT, Until=10, Dis=OFF) [
       Ti1:  Timer(Num=1, Act=READ, Cmp=??)
       F2:   File(Act=WR, Name=steer, Type=NUM, Val=Ti1.Val)
       Ro1:  RotSens(Port=C, Act=READ, Cmp=??)
       F3:   File(Act=WR, Name=steer, Type=NUM, Val=Ro1.Val)
       Ro2:  RotSens(Port=A, Act=READ, Cmp=??)
       F4:   File(Act=WR, Name=steer, Type=NUM, Val=Ro2.Val)
             Wait(Ctrl=TIME, Until=0,5)  -- wait for next sample time
     Lo2]
Lo1]
File(Act=CL, Name=steer)
```