#### Creating Movement Patterns

##### From Cartesian Coordinates to Differential Motor Outputs

Differential wheeled robots have two powered wheels, hence a motor drives the *left* wheel, and a motor drives the *right* wheel.

Cartesian Drive Mapping is the system by which motor output signals for both motors of a differential wheeled robot (left wheel *and* right wheel) are deduced from a Cartesian coordinate input. As explained earlier, the NXT framework motor output command consists of a *tachocount limit* (number of degrees of rotation, at which to stop), *absolute power* (indicative of speed)*,* and *polarity* (the direction of rotation). *Our implementation of the Cartesian Drive Mapping algorithm must calculate all.*

Fortunately, with little research, two implementations of such an algorithm where found. [1] [2] The later referenced implementation sourced from ‘a drop in the digital ocean’ is essentially a C and C# translation of the original ‘GoodRobot’ implementation in JavaScript. Note that the referenced implementations have a very rudimentary level ofmotor control. Because the implementation of the Cartesian Drive Mapping algorithm *only* varies Mpower, it is the *time* that the particular signal is *held* that determines the number of rotations of the wheel and hence the distance covered by that particular wheel (in either direction). As the reference implementations are for use in robotic systems designed to be manually controlled by a human user *through a joystick module* (which basically feeds the algorithm with a Cartesian coordinate), the human user acts as visual *feedback* - holding that *particular* joystick position (the calculated Mpower values for each motor) until he or she is satisfied with the distance and position achieved.

If the Cartesian Drive Mapping algorithm is to be used by an autonomous system, a greater level of control is required over the distance a wheel travels with any particular motor output signal (i.e. the time the signal is held should be just enough to achieve a particular distance).

Fortunately, as mentioned earlier, the LEGO® Mindstorms® NXT framework features a motor encoder, which allows the *tachocount* to be set to a constant. The encoder will allow a particular motor output signal to be applied to the motor until the *tachocount* (number of degrees) is reached. This will allow a greater level of control over the distance the motor covers with each motor output signal, though the distance will essentially be expressed as the *number of wheel rotations* until it is linked to the wheel’s *radius* –obviously both wheels would have the same radius. Linked to a radius, we can algebraically deduce the *unit distance* (e.g. centimetres or meters) the robot will travel - associating a step change in the Cartesian grid (Δx and Δy) with a step change in unit distance. [[1]](#footnote-1)

###### Relationship between Physical and Internal Variables

Before discussing the operations of the Cartesian Drive Mapping algorithm, let us make certain the relationship between the Cartesian coordinate *(input)*, the physical (real world) distance travelled, and the components of the generated motor output command.

**Let:**

*The Cartesian grid (exists only in software)*

Size of Cartesian grid from origin (0, 0) farthest points from origin would be (±G, ±G),   
i.e. in the NW, NE, SE, SW direction at polar radius *r*(±G, ±G).

*The physical distance the robot is expected to travel (hypothetically)*

*Real world* distance (cm)   
*Real world* time (s)   
*Real world* speed (cm/s)

*The physical wheel*

Wheel radius (cm)   
wheel diameter (cm)   
wheel circumference (cm) =

*Motor output signals (exist only in software / transmitted commands, but converted into physical signal by NXT hardware)*

The *tachocount limit* assigned to a motor output signal

The *power* assigned to a motor output signal (polarised) **But,** for the purpose of understanding relationships between the different variables involved, we will use:

The *magnitude power* assigned to a motor output signal (i.e. absolute of *power*)

The *polarity* of

Note that

The relationship between distance travelled, circumference of the wheel and tachocount limit

The relationship between the Cartesian grid and the distance travelled

As mentioned earlier:

The maximum possible *real world* distance that the robot can move with any single motor output command depends on the maximum possible value of. is of type *unsigned integer*, at least according the interfacing libraries that we utilize to establish a communication between the NXT and our PC (RWTH toolbox and MindSqualls library). The maximum value of an unsigned integer type in both MATLAB *and* C# is 99,999.

It may not actually be appropriate to have the Cartesian coordinate (±G, ±G) correspond to the colossal distance represented by Dmax as it may measure up to several meters, and therefore be inappropriate or inconvenient for certain applications. Hence it is necessary to allow the *change in* *distance per unit change in radius [of Cartesian coordinate input (x,y)]* to be manually assigned in code. Note that radius *r* is only a very rough linear approximation of the actual travel trajectory of the robot.

Where is essentially the calculated to induce the change in distance per unit change in the magnitude value of x or y. Hence, re-arranging the above equation:

Wheredenotes an assigned distance value in centimetres. However, a validation statement would be required (in software) to only accept a value of if the value deduced is (maximum acceptable value for unsigned integers).

The relationship between the absolute value of motor power with speed, time and distance

In our software interface to the NXT, *power* is of type *signed byte*, therefore:

-128

Therefore, has a total of 256 different values.

Because is even, the length of the integer x-axis and y-axis would be odd, as the first value would be (0, 0) and max values would be (±G, ±G). If we are to vary

Because we ] expect the size of the x-axis and y-axis to be odd, from the origin to the maximum value (because we expect G to be an even number). We can scale |Mpower| to the following:

-127

The distance travelled by a wheel upon a command is ultimately determined by the *tachocount* *limit* associated with the motor command (N), and the circumference of the wheel.

*Ex 1.* The time taken to reach the required distance will be proportional to *magnitude power*. The greater the *magnitude power*, the *faster* the distance will be achieved.

Ex 2. However, the *exact* mathematical function that relates speed (cm/s) to *magnitude power* remains unknown, because the change in the time period taken to achieve a distance over a unit change in *magnitude power* remains unknown.

This makes *time* the variable that needs to be measured (as accurately as possible) before a *real world* speed (i.e. in cm/s) is associated with a *magnitude power*. In terms of implementation, *time* could either:

1. *Be assigned as a constant* in code *–* this would require the time taken for the robot to achieve a particular distance (point) to be measured, averaged and (eventually) deduced experimentally.
2. The time taken to achieve a particular distance (point) could be deduced *through a carefully designed calibration procedure*.

*However,* measuring time (and relating that to a distance) would not take into account changes in velocity (*acceleration* or *deceleration*)of the motor, particularly with regards to change in distance of the target point.

When the relationship between the change in *magnitude power* and *the time taken to achieve a particular distance* *(speed)* is determined, it may become easier to control the exact *real world* speed of the motor. But introducing another approximated variable (T) seems rather unnecessary, as it will not improve the function or performance of the system. Thus, real world speed and time should be left out of the equation, the assumption that is adequate, where is considered 0 cm/s in terms of *real world* speed, and is considered

Variable relationship summary: Cartesian Driving Mapping

The Cartesian coordinate input is ultimately supposed to be determine the *real world* angle (direction) and *radius* (distance) of the target position. This will obviously be different to the polar angle and radius deduced from the Cartesian input (to model the non-linear trajectory of each individual wheel would be very time consuming).

The target position is achieved through the action of the left motor and right motor; we can control the *tachocount limit*, *polarity,* and *motor power* of both. We know that:

**(a)** *Polarity* determines the direction (clockwise or anticlockwise) a wheel should move in order to achieve the target position (+1 = forward, -1 = reverse, 0 = no movement).

**(b)** T*achocount limit* determines the *number of degrees* the wheel should rotate in order to reach the expected distance. As calculated earlier: centimetres.

**(c)** *Magnitude power* determines the speed at which the target distance is achieved (through an arbitrary relationship that hasn’t been modelled). Inherently, Speed has *nothing* to do with *accuracy* of movement.  
  
Whether actual *real world* *time* (in seconds) taken to achieve a distance is deduced via calibration *or* via trial and experimentation (hard-coding), there *is* going to be a degree of inaccuracy.

Because *Tachocount limit* and *polarity* are sufficient is controlling the wheels to a position indicated by the Cartesian input coordinate, we leave |Mpower| to simply act as a scale of speed that can be adjusted by the user.

###### Differential Wheel Drive and the Cartesian Grid

As illustrated in figure 1, the grid used in Cartesian Drive Mapping is always *square*. Therefore, both axes, x *and* y, will have the same absolute maximum value. The decision behind the size of the *input grid* will ultimately affect the fidelity of possible movements (and movement patterns) as a greater size would equate to a greater *resolution*. Thus: The larger the size of the grid, the greater the magnitude, therefore implying greater resolution, which would ultimately result in a greater number of possible movements i.e. motor output combinations (this may not necessarily be advantageous during development).

|  |
| --- |
|  |
| **Figure 1.** An illustration depicting the Cartesian grid. The robot is denoted by the block arrow facing north, positioned at the origin. As far the Cartesian Drive Mapping algorithm is concerned, the position of the robot does *not* change, this is regardless of any physical movement of the robot; hence any input coordinate presented will be respective to the robot’s *current* position. |

It is important to note that in differential mode driving, one wheel is always responsible for the distance the robot travels towards one of the four possible 45° points. The 45° point will correspond to the North East, South East, South West and North West direction.

|  |
| --- |
|  |
| Figure X. For a Cartesian coordinate to be of true NE, SE, SW or NW direction, |φ| = |ϴ| = 45°, because |x| = |y|. |

|  |  |  |
| --- | --- | --- |
| **Left Motor** | **Right Motor** | **Direction of Travel** |
| On (+) | Off | NE |
| On (-) | Off | SW |
| Off | On (+) | NW |
| Off | On (-) | SE |

Table x. The table shows the direction of travel (relative to the robot’s current position at the origin, facing north) in accordance to the motor active, and the rotational direction (- indicating reverse).

|  |  |
| --- | --- |
|  |  |
| Figure x (left) illustrates movement in NE and SW direction are controlled by movement of the left wheel alone.  Figure y (right) illustrates movement in NW and SE direction are controlled by movement of the right wheel alone.  Red wheel indicates the *distance wheel*. | |

Figure x and y illustrate that the polar radius of the coordinate corresponds to a real world distance ***D***. Note that the *real world* distance ***D*** at which the robot achieves the 45° angle is ultimately dependent on the *time* that motor power signal is held. If the signal is held longer than necessary, the robot would travel *beyond* the *real world* point at which the 45° angle is achieved. A motor encoder (via *tachocount limit*) automatically allows time to be limited by a given distance or number of degrees.

Observing and studying the role of the *distance wheel*, it becomes clear that the role of the opposite wheel is then, to deviate/turn-away from the |φ| = 45° angle: either towards |φ| = 0° - to force a sharp turn, or |φ| = 90° - to force straight travel with *no* turn at all. See figure x.

|  |
| --- |
|  |
| Figure x. The diagram illustrates the role of the *turn wheel* in modifying the direction (absolute angle) of the movement towards the direction of the x-axis (|φ|=90°), or the y-axis (|φ|=0°). The orange wheel is the appropriate *distance wheel* for the quadrant (i.e. closest associated 45° point). The *turn wheel* is black if inactive, blue if at *opposite* polarity to the *distance wheel*, red if at *same* polarity to the *distance wheel.* Because |φ| of all points is 45° apart – hence turn wheel, tachocount is kept the same. |

###### Understanding the Cartesian Drive Mapping Algorithm

The GoodRobot article briefly explains the derivation of the algorithm. [2]

The algorithm’s operation and derivation are discussed here forth.

(1) Prior Assumptions

Initially, a list of assumptions are made from prior knowledge of differential wheeled systems, and the direction of movement they induce: [2]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Direction of Movement | Point  (Figure 1) | Left Motor | | Right Motor | |
| **Magnitude Power** (indicating speed) | **Polarity** (indicating direction) | **Magnitude Power** (indicating speed) | **Polarity** (indicating direction) |
| N | d | P = Pmax | + | P = Pmax | + |
| S | f | P = Pmax | - | P = Pmax | - |
| E | h | P = Pmax | + | P = Pmax | - |
| W | b | P = Pmax | - | P = Pmax | + |
| NE | g | P = Pmax | + | 0 | 0 |
| SE | i | 0 | 0 | P = Pmax | - |
| SW | c | P = Pmax | - | 0 | 0 |
| NW | a | 0 | 0 | P = Pmax | + |

**Table 1.** A table of motor values we would assume for movement towards a particular direction.

***P*** is some arbitrary power value, indicative of the speed of motor rotation. Because we attempt to link this concept to the Cartesian grid in Figure 1, and the example points are at the edge of the grid (maximum of x and y, i.e. at + gridSize or - gridSize), we assume maximum speed due to greatest distance from origin. If the points remained in the same direction (angle from the origin) but decreased in magnitude, they would move closer to the origin, hence the power would decrease (tending to 0 as the point reaches the origin).

(2) Deduction of Relationships

Hence, the following relationship can be deduced between the Cartesian coordinate (input), its equivalent conversion to polar form, and the correct motor power signals to send (output). At this stage, we will assume that the scale of motor power signals (output) are equivalent to the scale of the axes of the square Cartesian grid (input): [[2]](#footnote-2)

* Variable x (i.e. the x axis) represents the *intended* *turn* (i.e. left or right).
* Variable y (i.e. the y axis) represents the *intended* backward/forward *movement*.
* When x = 0, this is indicative of no deviation from the centre line, i.e. the robot is to drive absolutely straight ahead, or reverse, with absolutely noleft or right turn. Hence, there would be *no* difference between left and right motor outputs (in either direction *or* magnitude power).
* When y = 0, this is indicative of no backward or forward motion, and therefore any movement produced is *solely* a turn/rotation. Thus, when y = 0, the magnitude power at both motors is the same, but at *opposite* polarity).
* When x = y, this indicates a 45° turn, where one motor is inactive (off / no motion), and the other motor is moving.

|  |
| --- |
|  |
| **Figure x**. Table of sample Cartesian coordinates (from figure 1), the equivalent polar coordinate, and the motor output values we expect to see from *prior* knowledge of differential wheeled driving. |

Hence we can also make assumptions about the algorithm’s operation before implementation.

* One wheel is used for turning, and the other for overall movement (to a distance). However,
  + When x = 0: *turning wheel* has the same polarity *and* absolute motor power as the *distance wheel*.
  + When y = 0: *turning wheel* has the same absolute motor power as the *distance wheel*, butwith *opposite* polarity.
* *Absolute φ* gives us as indication of how close the input coordinate lies to a 45° point (see figures x, x, and x). |φ| = 90° will therefore be indicative of no turn, i.e. *turn wheel* output = *distance wheel* output; |φ| = 0° will be indicative of full sharp turn, i.e. *turn wheel* output = - *distance wheel* output.
* Exactly which wheel is designated *turn wheel* or *distance wheel* is determined by the quadrant in which the Cartesian coordinate lies.

(3) Operation

1. Assume the range of all possible motor power outputs in the form of an array ranging between   
    and.
2. First, the input coordinate is classified as belonging to a particular quadrant (1,2, 3 or 4) using the following rule:

will allow us to determine which wheel is the *turn wheel* and which wheel is the *distance wheel* (later).

1. The *motor power output to the distance wheel* is determined essentially by the maximum absolute value of x and y.
2. Now, to calculate the *motor power output to the turn wheel.*  
   The input coordinates (x, y) is converted to polar form (radius *r*, angle θ). Essentially, it is the *magnitude* of θ that we are interested in – as its value is indicative of the point’s proximity towards the NE, SE, SW, or SE point (depending on the quadrant).
3. The angle θ is in radians, and should be converted into degrees (denoted φ).  
     
      
     
   where

This absolute angle would range anywhere from 0 to 90 degrees - i.e. scaled to fit to the 1st quadrant, with any point of being deemed to lie in the 45° direction appropriated by the actual quadrant of input (x, y).

1. We already know that**:**
   1. *The polarity of the turn wheel should be 0 when, as the movement towards a 45° point is controlled solely by the distance wheel.*
   2. *Polarity would be +1 when as the polarity of the turn wheel would be the same as the polarity of the distance wheel.*
   3. *Polarity would be -1 when as the polarity of the turn wheel would be the opposite to that of the distance wheel.*

Hence the output polarity of the *turn wheel* can be represented as a coefficient. The advantage of a coefficient with a range is that it can be used to scale the motor outputpower by the ‘severity’ of the turn (in order to accurately control deviation from the 45°):  
 (where polarity is ‘-‘, value of moves from -1 to 0)   
or (where polarity is ‘+’, value of moves from 0 to +1).

The ***turn coefficient*** denotes the degree to which the input coordinate deviates from the 45° line, and to which direction. Because a turn-coefficient of 0 denotes the point exactly 45° from the origin, means the turn is less than a 45° turn, and therefore the turn wheel would have to provide supplementary (same polarity) movement towards the direction of the distance wheel in order to *cut* from the 45° turn that the direction wheel would be making if it were to move *on its own* (this type of turn can be called an “soft” turn, as the turn < 45°) – hence when , the robot should appear to travel absolutely straight.

On the other hand, if, the turn is greater than 45°, and therefore the turn wheel would have to offset the angle *beyond* the 45° that the direction wheel would achieve on its own, this is accomplished by travelling in the reverse direction to the direction wheel (this type of turn may be referred to as an “sharp” turn, as the turn > 45°).

|  |
| --- |
|  |
| **Figure 2.** Plot illustrating the linear relationship between the magnitude angle () and the calculated turn coefficient. |

1. The power output value for the motor responsible for the *turn wheel* is calculated thus:

Where is the ***magnitude distance* from the 45° lines** (NE to SW, NW to SE), which (obviously) intersect at the origin. Note that. Ultimately, it is the*magnitude distance*scaled using the *turn coefficient* that gives us the magnitude of the motor output to the *turn wheel*.

When the coordinate is at a 45° angle from the origin, |x| and |y| are exactly the same, hence the difference is 0.Coincidently *and* when. However, as φ approaches 90° or 0°, is maximum, indicating maximum magnitude power for straight travel and sharp turns.   
  
Note that because the deduction of the motor output of both wheels is dependent solely on the *magnitude* values of (x, y) and the *magnitude* of φ of its equivalent polar form (r, θ).

|  |
| --- |
|  |
| **Figure x, Figure x.** A colour surface plot (top view and angled view, respectively) depicting the value of *magnitude distance*in relation to the value of x and y. Note that gridSize = 100, so that would be the absolute value (min and max) of x, y and the *magnitude distance*. More importantly, note that the value of magnitude difference tends to 0 as the absolute values of x and y become equal (indicating |φ| = 45°). Note the symmetry – the first quadrant of the Cartesian grid represents the *magnitude distance* of all coordinates. |

1. The ***quadrant*** of the input coordinate tells us the *rough* direction as to which the robot is to move. Hence, we can select the wheel that will be the *turn* wheel and the *distance* wheel.

|  |  |  |  |
| --- | --- | --- | --- |
| Quadrant of Input Coordinate | | Turn (- or + angular deviation from 45° point) | Distance to 45° point |
| Quadrant # | **Associated direction / |φ| = 45° point** |
| 1 | NE =  (G, G) | Right Wheel | Left Wheel |
| 2 | SE =  (G, -G) | Left Wheel | Right Wheel |
| 3 | SW =  (-G, -G) | Right Wheel | Left Wheel |
| 4 | NW =  (-G, G) | Left Wheel | Right Wheel |

Hence,

Flowchart

The following flowchart depicts the operation of the *original* (GoodRobot) algorithm.  


###### The AVINSoR Modification

Adjust *tachocount limit* and *polarity* not *absolute motor power* and *polarity*

Under ‘Relationship between Physical and Internal Variables’, we concluded that the AVINSoR implementation of the Cartesian Drive Mapping algorithm should adjust the *tachocount limit* and *polarity of motor power* in accordance to the position indicated by the Cartesian coordinate (input) rather than *absolute motor power* and *polarity*. This is because adjust *tachocount limit* is more effective at relating positions on a Cartesian grid with a ‘*real world’* distance.

Another reason to support this modification is that lower motor power values may not be suitable for a particular terrain, or may produce incredibly slow and ineffective movement. This is okay for robots whose movement is controlled by a human user, as the human user (acting as visual feedback) can always adjust the speed manually via the joystick; if the Cartesian inputs are being generated by an artificial intelligence algorithm or some other form of *code* than this will not be recognised. Hence the *absolute motor power* is defined as a suitable or practical range that the end-user may modify.

Operation

The operation of the modified Cartesian Drive Mapping algorithm can be described by the flowchart in figure x. Figure y describes how the data from the output *object* of the *CalculateDrive(…)* method is used to generate and execute motor output commands at variable values of magnitude motor power. The object oriented design of the Cartesian Drive Mapping algorithm is described in greater detail under ‘Object Oriented Design of the Cartesian Drive Mapping algorithm’.



Figure x. A flowchart describing the *CalculateDrive(…)* method of the ‘CartesianDriver’ object.



Figure y. A flowchart describing the *Play(…)* method of the ‘CartesianDriver’ object.

###### Object Oriented Design of the Cartesian Drive Mapping algorithm

As all output commands to the NXT-based robot assembly are computed from a client-pc (via a Bluetooth® connection). This means that the Cartesian Drive Mapping algorithm will be embedded with the client-side *application*, which is designed and implemented using the object-orientation paradigm.



Figure x. A UML diagram describing the classes to be implemented to allow Cartesian Drive Mapping in the client-side *application.*

CartesianDriver

Properties

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Controller | NxtController | Points to the ‘NxtController’ object used by the *play(…)* method to generate and execute motor output commands. *Default value is null.* |
| minAbsMotorPower | Unsigned Integer | The minimum allowable motor power to be set when wheel(s) are moving. Default is 30. |
| maxAbsMotorPower | Unsigned Integer | The maximum allowable motor power to be set when wheel(s) are moving. Default is 127. |

Behaviours

CartesianDriver(…)

Class Constructor. Creates a CartesianDriver object. Automatically sets motor power (motorPower) to maximum allowable motor power value.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| gridSize | Unsigned Integer | The size of the Cartesian grid from the origin. This will determine the minimum and maximum value of x and y.  Hence determines internal properties: minXYVal (= -gridSize), maxXYVal (= +gridSize). |
| wheelRadius | Double | The radius of the robot’s wheels, in centimetres.   Hence determines internal properties: wheelRadiusCM (= wheelRadius), distancePerDegreeCM (= wheelRadius/360). |
| minAbsMotorPower | Unsigned Integer | *Optional.* The minimum allowable motor power to be set when wheel(s) are moving. Default is 30. |
| maxAbsMotorPower | Unsigned Integer | *Optional.* The maximum allowable motor power to be set when wheel(s) are moving. Default is 127. |

**Returns** the created ‘CartesianDriver’ object.

Calibrate(…)

Calibrates the ‘CartesianDriver’ object to determine the number of degrees of wheel rotation per unit change in x or y of the input coordinate. Hence, this method determines the internal property ‘degreesPerStep’. Note that this method should throw an exception if the calibration fails to determine an appropriate value for ‘degreesPerStep’.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| delta | Object | This should be an unsigned integer if mode is passed ‘SetDeltaDPerUnitAbsXY’ or ‘SetDeltaDOfMaxXY’, or double otherwise. If the parameter is passed an object which is not of the correct type, an exception should be raised before ‘degreesPerStep’ is calculated. |
| mode | CalibrationMode | Determines how ‘degreesPerStep’ should be calculated. |

Drive(…)

Generates a motor output command for the left and right motors in accordance to the Cartesian coordinate *input*. Gets NXT to execute motor output commands to induce movement.

Method should raise an exception if x or y value < minXYval or > maxXYval.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| x | Integer | The x value of the Cartesian coordinate. |
| y | Integer | The y value of the Cartesian coordinate. |
| inhibitPhysicalMovement | Boolean | When ‘true’, the execution of motor output commands on the NXT are inhibited (no physical movement of the robot). |

**Returns** a ‘CartesianDriverOutputs’ object consisting of calculated motor outputs.

SetAbsMotorPower(…)

Sets value of (absolute) motor power to use when motor output commands are being executed by the NXT.

The method should throw an exception if [absPower < minAbsMotorPower] or [absPower > maxAbsMotorPower].

Hence this method determines internal property ‘absMotorPower’.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| inhibitPhysicalMovement | Unsigned Integer | When ‘true’, the execution of motor output commands on the NXT are inhibited (no physical movement of the robot). |

Play(…)

Execute the motor output commands represented by a ‘CartesianDriverOutputs’ object.

Execution of motor commands can only take place if ‘Controller’ is not *null* (i.e. an ‘NxtController’ object is associated with *this* object), otherwise an exception is thrown.

See figure y - flowchart describing the method’s operations.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| cartOutputObj | CartesianDriverOutputs | The ‘CartesianDriverOutputs’ object that represents the motor output commands to be generated and executed. |

Internal Behaviours

CalculateDrive(…)

Essentially encapsulates the mathematical operation of the Cartesian Drive Mapping algorithm, particularly the conversion of Cartesian coordinate to an output variable for the left and right motors.

In our case, the output variables will be the *tachocount limit* for the left and right motor.

See figure x - flowchart describing the method’s operations.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| x | Integer | The x value of the Cartesian coordinate. |
| y | Integer | The y value of the Cartesian coordinate. |
| minOutputVar | Integer | The minimum tachocount limit. This should be passed a 0. |
| maxOutputVar | Integer | The maximum tachocount limit. This should be passed degreesPerStep × maxXYVal. |

**Returns** an integer array consisting of an integer output calculated for the left motor, and an integer output calculated for the right motor (tachocount limits).

ReMap(…)

This is borrowed as-is from *a drop in the digital ocean*’s “tank-drive” implementation. [1] The method is used in *CalculateDrive(…)* to map values from the Cartesian range to the appropriate range for tachocount limit.

Note that the input parameters and the output are *floating-point values*.These are rounded to the nearest integer in *CalculateDrive(…)*.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Value | Float | The number in the *input scale* whose equivalent is to be sought in the *output scale*. |
| Min | Float | The minimum value of the *input scale*. |
| Max | Float | The maximum value of the *input scale*. |
| From | Float | The minimum value of the *output scale*. |
| To | Float | The maximum value of the *output scale*. |

**Returns** a floating-point value of the *output-scale* equivalent.

CalibrationMode

This is essentially an enumeration of the different ways that a CartesianDriver object can be calibrated – in order to deduce the number of degrees the motor should rotate per unit change of (magnitude) x or (magnitude) y on the Cartesian grid.

SetDeltaDPerUnitAbsXY

Denotes that the CartesianDriver should be calibrated such that *a particular distance is achieved for a unit change in (magnitude of) x or y*.

SetDeltaNPerUnitAbsXY

Denotes that the CartesianDriver should be calibrated such that the *wheel rotates a particular number of degrees per a unit change in (magnitude of) x or y*.

SetDeltaDOfMaxXY

Denotes that the CartesianDriver should be calibrated in such a way that *the particular distance per unit change in (magnitude of) x or y* allows a certain distance to be achieved when (x, y) = (±G, ±G).

SetDeltaNOfMayXY

Denotes that the CartesianDriver should be calibrated in such a way that *the particular number of degrees a wheel rotates per unit change in (magnitude of) x or y* allows a certain number of degrees rotation to be achieved when (x, y) = (±G, ±G).

CartesianDriverOutputs

This is essentially the object that stores the relevant data to produce the correct left and right motor output commands for the position indicated by the particular Cartesian coordinate for which it was produced.

Properties

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| MPolLeft | MotorPolarity | *Read only.* The polarity of the left motor. |
| MPolRight | MotorPolarity | *Read only.* The polarity of the right motor. |
| TachoLimitLeft | Unsigned Integer | *Read only.* The number of degrees for the left motor to rotate. |
| TachoLimitRight | Unsigned Integer | *Read only.* The number of degrees for the right motor to rotate. |
| DelayInSeconds | Unsigned Integer | *Read only.* The delay (in seconds) before this command is sent to the NXT for execution. This is for use when the object is a part of a collection to form a movement pattern, and is only relevant when the object is *not* the first in a collection. |

Behaviours

CartesianDriverOutputs(…)

Class constructor. Creates a ‘CartesianDriverOutputs’ object.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| lMotPol | Integer | The x value of the Cartesian coordinate. |
| rMotPol | Integer | The y value of the Cartesian coordinate. |
| lMotTachoLim | Integer | The minimum tachocount limit. This should be passed a 0. |
| rMotTachoLim | Integer | The maximum tachocount limit. This should be passed degreesPerStep × maxXYVal. |
| delayS |  |  |

**Returns** the created ‘CartesianDriverOutputs’ object.

1. Furthering our interest in more accurate control of the motors, and therefore the accuracy of *our* implementation of the Cartesian Drive Mapping algorithm, a particular RXE executable is used to read motor control commands and act as a synchronous *feedback* control system (i.e. for both motors) – producing a visible difference in the accuracy of movements (see *testing*). This method of motor control is covered in greater depth under System Implementation (page x). [↑](#footnote-ref-1)
2. The Cartesian grid’s absolute minimum value (-gridSize) and maximum value (+gridSize) may be mapped to the integer range indicated by minimum power output to the motor (motorOutputMin) and maximum power output to the motor (motorOutputMax).  
   *This will be particularly useful in converting a grid induced value (ranging from –gridSize to +gridSize) into a motor output value (ranging from motorOutputMin to motorOutputMax).* [↑](#footnote-ref-2)