3.7 Testing Straights and Turns Over a Simple Square Route Desktop/TestPrograms/ **routeTest.py**

This program combines MotorTest and OdometerTest to confirm the calibrations of the wheel/odometer combination in a simple square route back to the start position and heading.

**Method of Navigation**

Chariot navigation round an arena such as the assault course at the Rampaging Chariot Games is undertaken by navigating from waypoint to waypoint round the course. You designate these waypoints in x and y coordinates from the bottom left corner of the arena. The Autonomous Chariot travels from waypoint to waypoint in sequence by turns on the spot and straight lines. Curves/arcs can be added yourself at a later date to make the route faster and more efficient.

The planed route between waypoints is called the ‘required track’ and the robot position in the arena is called the ‘chassis position and heading’. Due to differences in motor power, wheel diameter and wheel friction, errors will build up between the required position in the arena and the actual position and heading of the chassis. Some of these differences can be measured and large errors corrected by applying calibration parameters, but we are operating in a real world, with a budget much less than NASA, and even they can never reduce the real world errors to zero (which would make the vehicle perform like a simulation on a computer screen).

It is these real world errors and uncertainties that make this project so interesting and provide us all with a real coding challenge. If it was simple it would be boring and you would not experience the satisfaction of beating this dumb Chariot with your human logic and innovation.

**“Far better it is to dare mighty things, to win glorious triumphs, even though chequered by failure, than to take rank with those poor spirits who neither enjoy much nor suffer much, because they live in the grey twilight that knows not victory nor defeat."**

**Teddy Roosevelt**

**Coordinate Scheme Used for the Autonomous Programme**

The test programmes are designed to simplify the programme and concentrate on the aim of testing and calibrating the chassis, motors, wheels and sensors to get the best possible performance. Where possible we have tried to isolate each major component so that it can be calibrated without being affected by other components. At the completion of the test programmes your calibration parameters will be inserted in the main programme and will therefore considerably affect the ultimate performance of your autonomous Rampaging Chariot.

For the purposes of simplicity we assume that the arena is positioned with North at the top and our zero origin is in the South West corner. Position in the arena is measured in millimetres along the x and y axes. Angles are measured in standard compass format with zero degrees being North and angles increasing positive clockwise so that East is 90 degrees.

**Coordinate System**

**y**

**x**

**θ**

**Chassis x,y,θ**

**hdg**

***Geek’s Box:*** **Cartesian positions or coordinates** are determined according to the *east/west* (x axis) and *north/south* (y axis) from the origin. Points or coordinates are indicated by

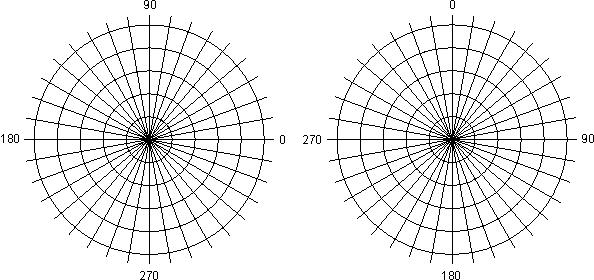
(x,y ). The origin is usually, assigned the value (0,0).

**Polar positions or coordinates** are determined according to the distance/ radius ‘r’, from the origin, and the angle determined relative to the reference axis ‘θ’ (Greek theta). (r, θ)

In the ’Maths’ scheme preferred by mathematicians, physicists, and engineers, the reference ray points off toward the right, and angles are measured anticlockwise from it (illustration at left).

In the ‘Compass’ scheme used by astronomers, navigators, military personnel, meteorologists, and robotics engineers, the reference ray points upward, and angles are measured clockwise from it (illustration at right). The radius coordinates are always positive. Angles can be specified in degrees from 0 to 360, or in radians from 0 to 2 pi.

The ‘Compass’ scheme is used for the autonomous robot, but most Python Maths functions use the ‘Maths’ scheme, so conversions between the schemes occur in our Python programmes.



‘Maths’ Scheme Compass’ Scheme

|  |
| --- |
|  |

**Calculation of Heading**

The Chariot chassis heading change in degrees is calculated using the difference in distance travelled by each drive wheel as measured by the two odometers. Unfortunately, for reasons of cost and simplicity of construction the odometers are mounted on the drive wheels which are positioned diagonally in the chassis. This means that for small heading corrections in a straight line the maths conversion from odometer distance in bits (1024 bits per rev) to heading change in degrees is a factor of 0.24

chassis heading change in degrees = difference in distance \*0.24 deg [180/pi/237]

For a turn-on-the-spot, because of the offset wheels, the radius of turn of the wheels and the distance travelled by the wheels is greater. The wheels also skid sideways. The maths conversion from odometer distance in bits (1024 bits per rev) to heading change in degrees is a factor of 0.14.

Because of the sideways skid the maths cannot cover unknowns such as the coefficient of friction between the rubber wheels and the arena floor surface, or the proportion of weight on each drive wheel when the balancing wheels touch the ground. We therefore provide two extra calibration factors for left and right friction which you will obtain by trial and error when testing the Chariot in actual turns. You can consider these friction calibration parameters as experimental ‘fiddle‘ factors. (The maths of odometer derived heading is covered in Appendix ??)

**Longitudinal and Lateral Speed and Turn Rate Profile**

The Chariot chassis has considerable inertia which will resist any change in its state of motion. This includes changes to its speed, direction or state of rest. Wheel skid will cause serious odometer errors, so an acceleration and deceleration phase is always desirable and is essential if the floor surface has low friction and you apply high forward or turn power.

A continuous deceleration to zero forward speed does not work as the chariot will stall and stop before it reaches the waypoint. We must therefore reduce speed when approaching the waypoint to a speed from which the robot will stop dead without the wheels skidding.

Both longitudinal speed and lateral turn rate profiles have five phases**:**

**Accelerate, Cruise, Decelerate, Creep, and Stop.**

Waypoint 1

Waypoint 2

Speed

Accel

Decel

Cruise

Creep

Stop

Distance

**Longitudinal & Lateral Speed/Turn-Rate Profile**

For short legs and turns the chariot may not have time to reach cruise speed and the acceleration phase and deceleration phase will merge.

**Forward and Turn Accuracy**

In a straight line the odometers measures distance to an accuracy of about 0.46 mm. and heading to an accuracy of 0.24 degrees.

The R-Pi motor speed and turn demands are transmitted every 16ms. Additional time delays occur in the UART serial data link, and motor drive board PIC. It is realistic to assume that a stop command will therefore take about 60 ms to reach the motors. At typical speeds the robot will have travelled about 30 mm or turned through an angle of 3 degrees in this time. We therefore cannot expect to get an accuracy much greater than these values.

Note: Heading errors are more serious than distance errors as an error of 3 degrees will cause the chariot to diverge from track by 50mm after travelling a distance of only one metre.

**Programme Logic.**