**Route Designation**

Route Designation must be simple, concise, unambiguous and comprehensive. It should not be influenced by hardware constraints or software elegance.

There are three common ways of designating a route:

1. **The Maze Method.** (No planned knowledge of arena or obstacles, but Start point and Finish point Defined). Robot plans its own route between these points avoiding obstacles it detects with its sensors.
2. **The Tom-Tom Method.** (Prior knowledge of arena size & shape and position of all known obstacles. Start point and Finish point Defined). Robot plans quickest or shortest route/waypoints (aka roads) between these points avoiding obstacles (eg toll bridges) it knows about. Re-plans route if unexpected obstacle (e.g. roadworks) is detected.
3. **The Waypoint Method.** (Prior knowledge of arena size & shape and position of all known obstacles).

Pre planned Waypoints define the route. Robot travels from waypoint to waypoint in sequence and refines its absolute position in the arena by navigation updates using sensors.

The Waypoint Method is considered the best to undertake the initial Robot Games Assault Course task as obstacles must be negotiated in a set order. Other Methods may be best for other tasks.

We designate the position and shape of the Arena, Obstacles and the Route using Cartesian Coordinates (X, Y). This requires a rectangular grid with its origin at the bottom left corner. The X axis is the horizontal axis from the origin (Positive to the right/East). The Y axis is the vertical axis from the origin (Positive up /North). The default values are X=4876 mm and Y=7315 mm (16ft x 24ft). This is the overall dimensions of the arena used for the Rampaging Chariots Scottish Robotic Games. The dimensions and scale of the grid can be altered from these values by keyboard input.

Another choice is the method of angular measurement.

**Angles** could be measured in ‘Degrees’ or ‘Radians’ using either:

1. The ‘Compass’ method where 0 degrees is up the Y axis (equivalent to ‘North’) and positive angles are defined 0 to 360 degrees clockwise from zero.
2. The ‘Mathematical’ method where 0 degrees is along the X axis to the right and positive angles are defined anticlockwise from that zero axis.

We have chosen to use the familiar ‘Compass’ method and degrees for the route input, but some internal code calculations are done in Python using the Mathematical method and radians.

**Distances** are measured and calculated using mm.

**Arena ‘Play Area’ Boundaries.** The arena boundaries or walls are input as a shape consisting of Points/Corners joined by straight lines. These Points/Corners can either be designated by consecutive left clicks of a mouse, or input as consecutive X, Y coordinates from the keyboard.

After the arena is defined with straight lines, any Point/Corner can be corrected or refined by updating its X, Y position from the keyboard. Any straight segment can be converted into an arc by inserting radius and arc-angle parameters from the keyboard. This action automatically corrects/refines the coordinates of the next point in the sequence as the end of the arc.

**Obstacles.** These are defined by ‘Type’ (Pole, barrel, etc.) and can either be input by X, Y coordinates, or by a click of the mouse. The default orientation is 0 degrees, but this can be altered from the keyboard.

**Route Waypoints**.The route is defined by Numbered Waypoints and consists of Straight Lines, Spot Turns and Arcs. The Waypoint position coordinates (X, Y) can either be designated by left clicks of the mouse or input manually from the keyboard. Other Waypoint parameters can either be selected from a right click drop-down menu or input manually from the keyboard.

The route is defined with straight lines, any Point/Corner can be corrected or refined by updating its X, Y position from the keyboard. Any straight segment can be converted into an arc by inserting radius and arc-angle parameters from the keyboard. This action automatically corrects/refines the coordinates of the next point in the sequence as the end of the arc.

Two input methods are available of defining the position of a Waypoint:

1. **Cartesian Coordinates.**

The route designation methods have similarities with the ‘G Code’ used to programme Computer Numerically Controlled (CNC) machines.

1. **Polar Coordinates.**

‘Polar’ coordinates are measured as a distance and angle (r, θ) from a known origin.

Distance and Heading parameters.

Obstacles and Waypoints defining the route are inserted in a specific Python “Waypoints” function in the programme. The ‘Next Waypoint’ parameters are then extracted whilst the programme is running. Some initial checking of illegal waypoint positions should be undertaken during the entry of parameters and both the course and the waypoints should be displayed on a course visualizer.

**What Inputs are needed to define the Waypoints and the route/actions?**

WP No, Posn X, Y, Hdg, Dirn, Radius. Leg Dist, Total Dist, Total Hdg, Speed, WaitTime,

Positions X and Y are measured from the origin of the arena to the centre of the robot chassis

Distances are in millimeters

Headings are in degrees (wrt. Arena North.)

Speed is in mm/s

Wait is in seconds

Sensor Distances R is in mm (wrt. Sensor Rotation Position on robot)

Sensor Angles θ is in degrees (wrt. Sensor Rotation Position on robot)

The Leg Distance, Total Distance since Start, and Total Heading Change since Start are calculated by the R-Pi

**Waypoint Data**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Waypoint**  **num** | **X**  **mm** | **Y**  **mm** | **Hdg**  **deg** | **Direct-ion** | **Radius**  **mm** | **Leg Dist**  **mm** | **Tot Dist**  **mm** | **Tot Hdg**  **deg** | **Speed**  **mm/s** | **Wait**  **sec** |
| 0 | 1000 | 500 | 090.0 | Start | -------- | -------- | 0.0 | 0.0 | 50 | 0.0 |
| 1 |  |  |  | Fwd | -------- | Calc | Calc | Calc | 50 | 4.0 |
| 2 |  |  |  | Back | -------- | Calc | Calc | Calc |  |  |
| 3 |  |  |  | Left | 0 | Calc | Calc | Calc |  |  |
| 4 |  |  |  | Right | 300 | Calc | Calc | Calc |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |

**WP (No,X,Y,Heading)** Waypoint Position and Robot Heading at arrival at that waypoint.

WP 0 is the Start Waypoint and the Robot will wait until it receives the ‘Go’ Command.

The End WP is the last WP in the Table and the Robot should stop and wait.

**Line (No,X,Y,Forward)**  Straight Line Forward/Back to next Waypoint (Cartesian coords)

**Turn (No,X,Y,Degrees,Dirn,Radius)** Turn Left/Right to Heading.

A Radius of Zero is a stop and turn on the Spot, otherwise it is an Arc.

The Default programme assumes a radius of zero and does not include code for an Arc.

**Speed (**Longitudinal Speed)

The Default programme only contains code for a fixed longitudinal speed. The lateral (turn) speed is also a fixed speed which is slightly more (x%) than longitudinal speed due to wheel skid friction.

**Wait (Sec)** Maximum Wait to undertake Sensor readings for Navigation update

**Sensor Data**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Waypoint**  **num** | **Sensor** | **Obstacle num** | **Update**  **Type** |  |  |  |  |
| 2 | I-R | 8 | Hdg |  |  |  |  |
| 2 | U-S | 9 | Y |  |  |  |  |
| 2 | U-S | Left Wall | X |  |  |  |  |
| 3 |  |  |  |  |  |  |  |

The Default programme only contains code for an Infra-Red sensor and an Ultra-Sonic sensor.

The table above describes what sensor updates should occur at particular waypoints.

e.g. At waypoint 2 the Infra-Red sensor should scan for obstacle 8 and the Ultra-Sonic sensor should scan for obstacle 9 and then for the arena left wall.

The wait at the waypoint should be sufficient for two/three scans or until the max wait time has been reached

The Default programme does not include code to detect the Left/Right edge of an obstacle.