**Autonomous Rampaging Chariot – Control and Navigation 20/10/16**

**Aim.**

The autonomous Rampaging Chariot is required to undertake the Assault Course at the Scottish Robotic Games.

**Introduction**

In order to perform navigation an autonomous robot needs some sort of on-board sensor to obtain the motion and position of the robot. This can be conducted ‘Open Loop’ which is basically dead reckoning resulting in small errors adding up over time, or ‘Closed Loop’ where the errors are corrected at intervals by using an update of position obtained from some other sensor such as a camera, sonar, GPS etc. The rules for autonomous robots prohibit the use of special transmitter beacons and cameras external to the robot, but we can use general transmissions from commercial radio masts, the earth’s magnetic field, GPS or astronomy. Unfortunately as tests are to be conducted indoors the satellite Global Positioning System (GPS), and star sightings are unlikely to work.

**1.1 Method**

The robot will proceed round the course via a series of waypoints defined by X, Y, Φ arena coordinates.

(X and Y are Cartesian coordinates of the Waypoint and Φ is the required Heading at the Waypoint).

Waypoints are numbered consecutively from the start point.

Movement between Waypoints is via

1. A turn on the spot
2. A straight line
3. A circular arc. (This can be added by students themselves)

**1.2 Open Loop Navigation and Control**

The Navigation System is based on two odometers measuring the distance travelled by the two drive wheels from the start point. The difference between the distances measured by each drive wheel odometer is used to calculate the current heading relative to the heading at the start point. This technique enables a continuous estimate of raw position and heading to be calculated and is called ‘Open Loop Navigation’ as it relies on ‘dead reckoning’.

.

The distance travelled and the current heading are compared with those expected at the next waypoint and a (near) match is used to trigger a waypoint change.

The robot control parameters are demanded velocity ‘V’ and demanded heading rate ‘ω’. This is the same as the control parameters used in manual operation when using the standard radio control.

Slight differences in the performance of the two wheel motors may cause the robot to stray from its intended heading. This can be corrected by comparing the calculated heading regularly with the planned heading and applying a minor lateral correction to motor power (Demanded heading rate ‘ω’) to correct any heading discrepancy.

Unfortunately, if either wheel skids during a longitudinal or lateral movement, the odometers will be in error and hence the open loop distance and heading will be in error and the robot will diverge from its planned route. The actual robot position and heading in the arena will be different from the position and heading the autonomous system has calculated.

The Open loop navigation system can be tuned by adjusting specific constants such as wheel diameter to make it reasonably accurate over a short distance, but even small errors caused by skidding can build up over a number of seconds and cause the robot to stray from its intended track and hit an obstacle.

**1.3 Closed Loop Navigation and Control**

The navigation loop is closed by using position and heading fixes from other sensors to determine the actual position of the robot in the arena or relative to a known obstacle. This allows it to determine its position and heading errors and then calculate a new path to regain the planned route.

Our autonomous robot is provided with an IR distance sensor attached to a stepper motor which enables it to scan over a wide arc. This sensor is mounted on the front of the detachable lid and is used to detect and track specific obstacles.

An ultra-sonic distance measurer mounted on a servo motor is also provided and can also be used to detect and track obstacles. These two sensors have different characteristics which can be measured and compared.

By measuring the distance and associated angle of two obstacles, or the side of the arena, a geometry method called ‘triangulation’, can be used to calculate the ‘actual’ position of the robot in the arena and ‘actual’ robot heading.

The ‘actual position’ is then used to correct or update the estimated ‘open loop’ position and allow the robot to regain the planned track.

PJB

**2.0 Autonomous Software**

The Autonomous software works off, lots of different program files being connected together and sending information between one another to control the RC and then display what happens to the RC on the screen through a GUI. The program files themselves are connected together through the use of Control Loops.

To put them simply Control Loops are containers that blocks of code can be placed within. When within a Control Loop the block of code will run continuously and able to easily connect to other blocks of code that are also contained within Control Loops.

**2.1 Main.py**

The Main.py file is the one that contains the setup for the Control Loops. It is broken up into 5 sections.

1. First section is importing the program file that you want to place within Control Loops as well as other parameters you might want to use like math and time.
2. Second section is the calibration values to be passed into the program files that have been imported.   
   As well as initialisation of the program files themselves with parameters being passed in. This is where most calibration changes will be made for ease of the user so they don’t have to dig through files to find out where a variable needs to be changed.
3. Third section is where the imported program files are entered into their own personal Control Loops, ready to be connected to each other.
4. Fourth section is where the program files within their respective Control Loops are connected together to allow information to be pass between them.
5. Fifth section is where each of the Control Loops are started up on their own threads. Run is then called with makes all previously started threads run the code placed on them. In this case it will run all of the Control Loops that have been started up beforehand.

**2.1 OdometerControl.py**

The OdometerControl file is just like the test Odometer program (that you should have been through by now) with just a few changes. In the test program you have to print the values to the screen and handle all the information about the RC in just that program file. While in this file the Odometer just keeps track of the distance travel and the heading of the RC, then passes the values to other program files through the Control Loops to have them printed on the screen or used in other calculations.

Methods called Translators are used to pass values between Control Loops. In a translator you specify where it is going and what message is to be sent. For Odometer it is sending a message to trackControl

.