

Part 3

Testing the Motors, Sensors & Scanning Actuators

All the coding described in this Part 3 are **Green Modules** and should be within the capability of all pupils to undertake.

A number of dedicated test programmes are provided to test and calibrate the motors and sensors. 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. You will find the programmes in a Desktop folder called Test Programmes.

The test programmes are designed to:

- a. Simplify the programme and concentrate on the aim of testing and calibrating the chassis, motors, wheels and sensors to get the best possible performance
- b. Test each motor and sensor separately to check it is working properly. Later when the full autonomous programme is in operation the test programmes will enable you to confirm that a particular sensor is working correctly and thereby confirm whether an unexpected event is caused by hardware fault or a software programme error.
- c. To get you familiar with using Python to control motors and sensors in the real world. The programmes enable you to build up your knowledge in a progressive way.
- d. To enable you to calibrate each sensor to your robot which is essential if you want your robot to perform to its maximum capability. The calibration numbers obtained during the test procedure will later be transferred to the full autonomous programme.
- e. To get you used to Python programming we provide some extra coding you should do to modify each test programme to make it do extra things.

Try to resist the temptation to jump ahead and take time to do the extra coding as it will help you to understand the techniques of controlling the hardware. If you are really stuck after you have tried to do this, you will find some example code in a file called testProgramAnswers.

3.1 Testing the Chariot Raspberry Pi

When the switches, cables and sensors have all been connected ask your teacher to check that you have plugged all the cables, sensors and scanning motors (called Actuators) into the correct pins on the Sensors Interface Board. **If you have not got the cables connected correctly there is a risk that you will cause the R-Pi or the odometers to die; Burning smell, Kaput, RIP.**

- (1) Check that all the switches on the lid are off and the 20 wire grey ribbon cable is connected between the R-Pi and the sensors board.
- (2) Connect the R-Pi to Sensors board ribbon cable and 20 pin plugs
- (3) Plug in your own HDMI cable & monitor, USB keyboard and USB mouse into the chariot R-Pi
- (4) Insert one of the two pre-programmed SD cards (provided) into the Chariot R-Pi
- (5) Plug in the mains to 5.1v Universal Power Supply (provided) into the R-Pi and switch on the monitor and then this R-Pi Universal Power Supply.

You should see the Red LED on the R-Pi, and the Red LED 2 on the Sensors Interface Board illuminate (Red LED 1 may also be faintly lit). The Green LED on the R-Pi will then start to flash and a succession of Boot messages scroll up the monitor screen. When the boot process is complete the screen will go blank for about 15 seconds and then show the Desktop with a big Raspberry in the centre.

Cloning the SD card

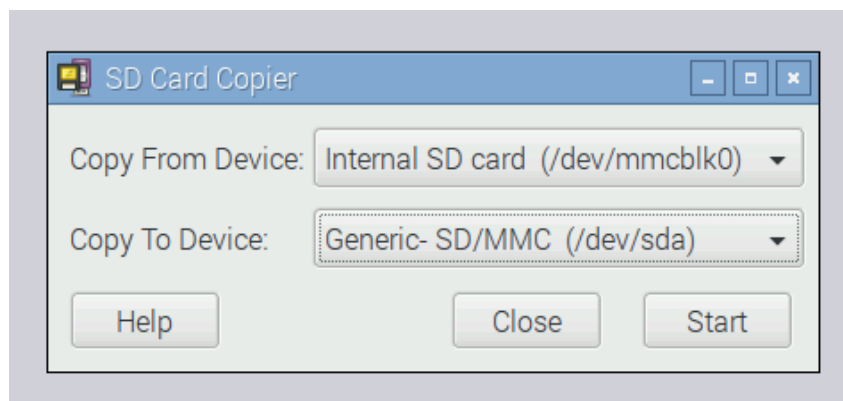
We provide two fully loaded SD cards for plugging into the Raspberry Pi(s) provided.

Before you start any experimenting with your Raspberry Pi you should clone the pre-programmed SD Card provided This is your “GET OUT OF JAIL FREE” card:

- a. It will provide a backup should the SD cards we provide get corrupted for any reason.
- b. If you inadvertently edit any of the issued programmes and they don't work you can replace individual files, or all of them, if you have had a programming disaster.
- c. You or a fellow member of your team can work off line to develop code and test it using the Simulate mode. The Rampaging Chariot Autonomous vehicle can then be shared by swapping the SD cards over.
- d. It is good practice to back up all your files regularly. If you don't, sooner or later you will experience a catastrophic hardware failure and loose everything.

The latest R-Pi software issued in 2016 contains an application that will copy all the files on your Raspberry Pi SD card onto another SD card.

- (1) Put the blank/target SD card into the USB- SD card writer (provided). This adapter plugs into one of the USB sockets on the R-Pi. Please bear in mind that everything on the destination card will be overwritten by this program, so do make sure you've got nothing you want to keep on the destination card before you hit Start!
- (2) On the Desktop screen, click on the Menu icon at top left and select Accessories > SD Card Copier > Enter



- (3) In the 'Copy From Device' box, select "Internal SD Card", and then select the USB card writer in the 'Copy To Device' box (where it will probably be the only device listed). Press 'Start', watch the messages on the screen and wait for the copy process to complete. In about 20 to 30 minutes you should have a clone of your current installation on the new SD card.
- (4) You can test it by putting the newly-copied card into the R-Pi's SD card slot and booting it. It should boot and look exactly the same as your original installation, with all your data and applications intact.

Geeks Box: The SD Card Copier program does not restrict you to only copying to a card the same size as the source; you can copy to a larger card if you are running out of space on your existing one, or even to a smaller card (as long as it has enough space to store all your files - the program will warn you if there isn't enough space). It has been designed to work with Raspbian and NOOBS images; it may work with other Operating Systems or custom card formats, but this can't be guaranteed.

The only restriction is that you cannot write to the R-Pi internal SD card reader, as that would overwrite the OS (Operating System) you are actually running, which would cause bad things to happen.

- (5) When the cloning is complete shut down the Raspberry pi by clicking on the small raspberry Menu icon at the top left of the screen. The shutdown command is at the bottom of the menu displayed. Select Shutdown in the central box that appears and wait for the Green LED on the R-Pi to stop flashing and go out (The Red LEDs will remain on). You can then switch off or unplug the power supply safely.

Warning: If you shut down the R-Pi by switching off the power supply without doing the formal shutdown process, there is a risk that the SD card can get corrupted.

3.2 Switching on the Autonomous System

- (1) Fit the Auto/Manual jumper in the **Auto** position on the Master Motor Drive Board



Jumper in Auto Position

- (2) **Check the chassis is supported on a wooden block with the wheels free to rotate.**
- (3) Unplug the Universal Power Supply from the R-Pi and plug in the Chariot 5v power supply
- (4) Switch on the large 'Motor Power' Switch on the lid. The red LEDs on the two Motor Drive Boards should flash 5 times and then go steady Red. The two Red LED lights at the back of the Chariot should also illuminate. The Slave Board Red LED may go out.
- (5) Switch on the 'Auto System switch on the lid. The Red LED on the 18v to 5v UBEC Power Converter Module should illuminate.
- (6) Switch on the 'Sensor Motors' switch. The Red LED on the Sensors Board near the power plug should illuminate. Switch OFF the 'Sensor Motors' switch as this is not required until we are ready to test and calibrate the scanning sensors.
- (7) Switch on the Raspberry Pi switch. The red LED should illuminate and the green LED should flash intermittently. The R-Pi should boot up and the usual lines of boot statements should scroll up the screen and leave you (after 15 seconds) with the standard Raspberry icon and Desktop display.
The Red LED 2 on the Sensors board nearest to the R-Pi should illuminate to indicate that 5v is reaching the sensors board from the R-Pi (Red LED 1 may also be faintly lit).

Info Box: If the Raspberry Pi ever asks you to log in with your user name and password, the default login name is **pi** and the default password is **raspberrypi**. At the prompt **pi@raspberrypi ~ \$** type **startx**

```
raspberrypi login: pi
password:

pi@raspberrypi ~$ startx
```

The screen with the big Raspberry icon is the standard Raspbian Graphical User Interface (GUI) and you can simply click on the Desktop icons to access the software programs.

Geek's Box: As an alternative to the GUI (Graphical User Interface), you can communicate with the Raspberry Pi using text-based instructions, known as **commands**. This form of communication is called a **command-line interface**, and the window into which you type the commands is called a **terminal**. Although the GUI might be more user friendly and easier to understand than text commands, text commands can be faster when you become more experienced in using them.

(8) Click on the R-Pi Desktop icon **LXTerminal** using the mouse.

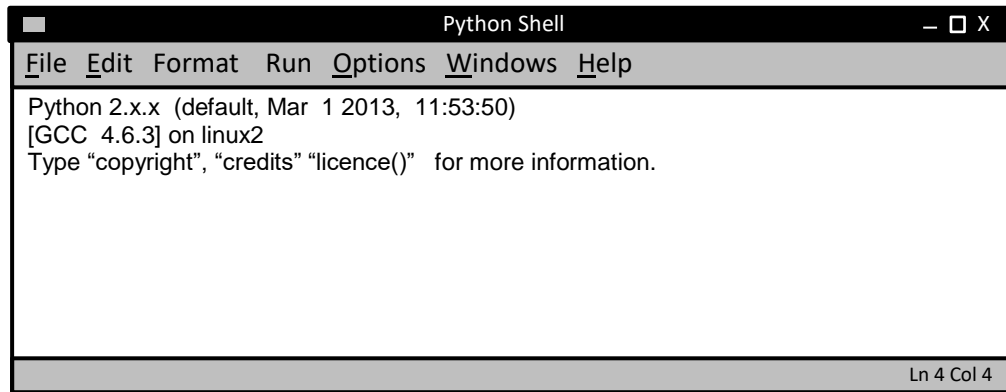
```
File Edit Tabs Help
pi@raspberrypi ~$
```

(9) After the prompt type **sudo idle** and press the Ent key.

```
File Edit Tabs Help
pi@raspberrypi ~$ sudo idle
```

Geek's Box: The word 'sudo' is short for 'super user' and allows you access to some R-Pi functions such as the **GPIO** (General Purpose In-Out) pins that are normally restricted. Be careful, however, as use of sudo also allows you to delete all your files. The programming environment 'Idle' is the user interface, called the IDE ('Integrated Development Environment', that we use for running Python programs. All the programmes we provide use Idle. This uses Python version 2 which is widely used. (Idle 3 uses Python version 3 which is not yet popular.). Idle allows you to create code and edit code as well as run or execute the code. It will also check your **syntax** which is a set of rules to check the code you have typed is valid Python code. When you make a mistake or a typo in your code, your programme may display a **syntax error** message. A syntax error will stop a programme from running because the R-Pi cannot understand the code. The most common reasons for syntax errors are a misspelt word, a character left out, an upper case instead of lower case character, or a missing colon at the end of loops and conditional statements.

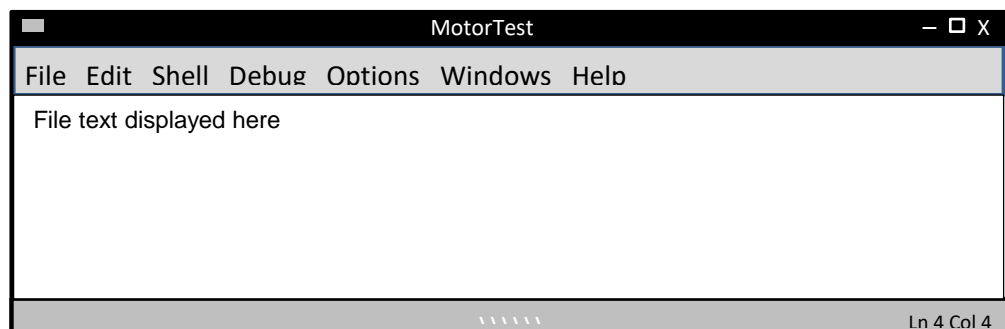
The Python 2 IDLE Shell screen should now appear:



The Python 2 IDLE Shell

3.3 Testing the Motors Under Autonomous Control [Desktop/TestPrograms/motorTest.py](#)

- (1) In the Python Shell window, Click on **File>** Click on **Desktop>** Click on **TestPrograms>** and then click on the file **“MotorTest.py”** to open it in the Python text editor.



- (2) Click **File > Save As** and save the file as **motorTest2** You should periodically save your files under progressive version numbers whenever you have made a significant alteration.
- (3) Run your version 2 file by clicking **Run > Run (or pressing F5)**

Speed Demand

The Chariot movement is controlled by two numbers we call **fwd** and **turn**. The first determines the forward speed and the second determines the rate of turn. Both numbers have to be between 0 and 255 which in binary form is 8 bits. The number 127 is the stationary datum for both forward speed and turn rate. Increasing forward speed is $127 + \text{fwd}$. Increasing backwards speed is $127 - \text{fwd}$.

Geek's Box: The PIC (Programmable Interface Controller) on the Chariot Master Motor Control Board works using positive whole numbers. We therefore have to send forward and back commands in this format. In common with most radio control systems we therefore use 127 as our zero, or stationary, datum. Increasing forward speed is $127 + \text{fwd}$ and increasing backwards speed is $127 - \text{fwd}$. For turn rate the same convention is used with increasing right turn rate being $127 + \text{turn}$ and increasing left turn rate $127 - \text{turn}$. The two numbers are sent to the Chariot Master Motor Control Board via a UART (Universal Asynchronous Receive/Transmit) serial data bus as two 8 bit Bytes.

The separate speed and turn rate numbers are displayed in the small 'pygame' black window that appears on the screen.

- (4) For the Motor Test programme we use the arrow keys to increase speed or turn rate.
- (5) Press the up arrow key and the motors should start turning slowly forward at about the same speed. Subsequent presses increment the speed by 5 each press. Note: The first press may not be sufficient to overcome the static friction.
- (6) To slow down the speed press the down arrow key.
- (7) Press the left or right arrow keys to change turn rate. For a left turn the left wheel should turn backwards and the right wheel should turn forward and visa-versa.
- (8) A combination of forward and left demand should adjust the speed and direction of wheel rotation appropriately.
- (9) Experiment with the arrow keys.
- (10) To stop the movement quickly press the **space bar**.
- (11) To stop the motors and exit the programme; press the **'End'** key.

Geek's Box: Due to minor differences in manufacturing and electrical circuit resistance, the two drill motors may not turn at exactly the same speed. The right drill motor is also mounted the other way round and forward and backwards performance may not be exactly the same.
Welcome to the real world!



MotorTest Parameters Display Box

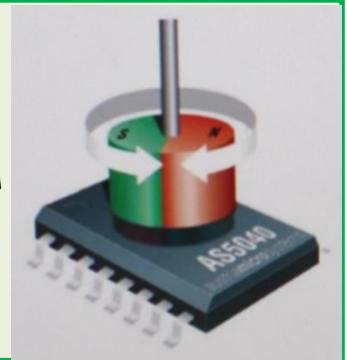
- (12) Modify the programme yourself so that when the key 'a' is pressed the wheels accelerate from stationary to 127+80 and then decelerate back to 127-80 and continue to cycle until stopped with the space key.

If you are not experienced with python, you will find a block(s) of code in a file called testProgramAnswers that will help you.

3.4. Testing the Odometers Desktop/TestPrograms/ odometerTest.py

- (1) Close the motorTest.py programme and then open the file “**odometerTest.py**” to open it in the Python text editor as before.
- (2) Click **File > Save As** and save the file as **odometerTest2** You should periodically save your files under progressive version numbers whenever you have made a significant alteration.
- (3) Run your version 2 file by clicking **Run > Run**

Geek's Box: The odometers work by having a magnet on the drive shaft that rotates over the centre of an IC. The IC measures the angle of the magnet to a resolution of 1024 positions per revolution which is about 1/3 degree. This angle information is sent over a serial data bus to the R-Pi. During every revolution of the magnet and wheel the angle increases from zero to 1024 and then starts at zero again. This is called a 'rollover'. To measure distance travelled we have to read the rotation angle over several rotations of the wheel, detect when each rollover occurs, and adjust the rotation angle count by 1024 each time.



The Pygame display box contains the odometer angles for each wheel:

Row 1 Displays the raw digital angle from the IC for each wheel.

Row 2 Displays the digital angle corrected for the zero start position

- (4) Rotate each wheel separately by hand and watch the angles count up to 1024 and then rollover to zero again. Notice it requires a significant force to move the wheels as you are also turning the gearbox and motors. When you reverse direction there is a sudden reduction in resistance due to significant slack called 'backlash' in the gearbox. The count then counts down to zero and rolls over to 1024.
- (5) Press 'c' on the keyboard to change to **Continuous** Count mode. In this mode the rollovers are detected and actioned to provide a continuous readout of odometer angle. Rotate the wheels to see the effect.

Rows 3 & 4 Display the continuous count for both wheels.

- (6) Notice that if you turn both the wheels in a forward direction, one odometer counts up and the other down. This is caused by the Right wheel driving in reverse.

Row 4 Displays the digital angles with the Right odometer digital angle reversed so that both count in the same direction.

	Left Odom	Right Odom	
Rollover	346	592	Raw Data
	414	641	Corrected Start Posn
Continuous	414	-380	Corrected Start Posn
	414	380	Rt Odom Reversed

- (7) If the wheels are 150 mm in diameter, how far does the chassis move forward for a change in digital angle of 1 count?
- (8) Modify the programme to display a row 5 which shows the distance moved by each wheel in millimetres.

If you are not experienced with python, you will find a block(s) of code in a file called testProgramAnswers that will help you.

3.5 Communicating with the Chariot by Wi-Fi

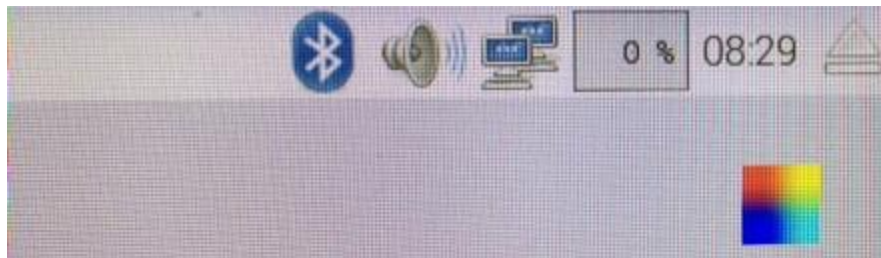
In competitions we are only allowed to communicate with the chariot to Start and Stop the Autonomous Motion and to receive telemetry back from the robot for monitoring, display and analysis purposes. In development, however, we can have full two-way communication with a ground station. This could be another R-Pi, laptop, tablet or smart phone. The Autonomous kit has two Raspberry Pi 3s to do this communications task.

The Raspberry Pi 3 has built in Wi-Fi and Bluetooth, but we are going to use Wi-Fi to communicate as it is more reliable in a crowded environment.

The '**Chariot**' R-Pi should already be configured as a Wi-Fi Hotspot/Server. To communicate, a second '**Communications**' R-Pi has to be configured as a Wi-Fi Slave/Client. The default SD card provided will work in both R-Pi s, but comes in the default 'Hotspot' mode.

Configure The Chariot R-Pi as a Hotspot

The Chariot R-Pi comes already configured as a Wi-Fi Hotspot/Server. You can identify it in this configuration by looking at the communications icon near the top right corner of the screen which should show two computer screens with three white dashes in each screen.



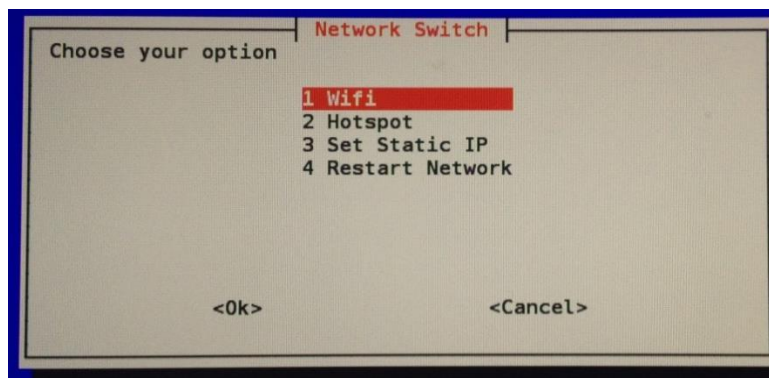
R-Pi Configured as a Hotspot/Server

If In Hotspot Mode

- (1) Shut down the 'Chariot R-Pi' and remove the HDMI cable, keyboard and mouse cables. Plug in the Chariot 5v power cable and switch on the sensors power and R-Pi power switches to re-boot the Chariot Pi in Hotspot mode as an untethered vehicle.

If Not in Hotspot Mode

- (2) Reconfigure the Chariot R-Pi as described in steps (3) to (5) below:
- (3) From Desktop > Enter 'Network-Switch' folder > Double click on switcher.sh
- (4) Click "Execute in Terminal" and you should get the screen below



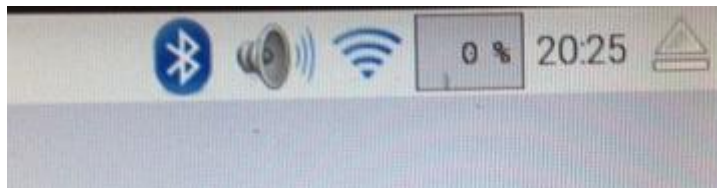
- (5) Move up and down with the Arrow keys to select **Hotspot** and press Enter.
After a pause, you should see an icon at the top right of the display change to the two computer screens. Each screen has three white dots to indicate signals are being sent out.

Configure the Communications R-Pi as a WiFi Slave/Client

The first time you boot up the Communications R-Pi it should be in Hotspot/Server configuration and you will need to reconfigure it as a WiFi Slave/Client. On subsequent booting it should configure in the WiFi mode.

- (6) Plug the Universal Power Supply cable, HDMI cable, keyboard and mouse cables into the '**Communications R-Pi**'.
- (7) Identify the WiFi configuration by looking at the communications icon near the top right corner of the screen.

When connected by WiFi the Communications R-Pi will show 3 concentric arcs.



WiFi Configuration Connected

When disconnected it will show two computer screens with a box containing a black cross.



WiFi Configuration Disconnected

If Not In WiFi Mode

- (8) From Desktop > Enter Network-Switch folder > Double click on switcher.sh
- (9) Click "Execute in Terminal" and you should get a visual screen
- (10) Move up and down with the Arrow keys to select WiFi and press Enter
- (11) Click OK and after a few seconds you should either have the two screens with a cross in a box or the concentric arcs icon.

If In WiFi Mode

- (12) Check that Communications R-Pi is connected to Chariot hotspot R-Pi
- (13) Click on icon in top right hand corner (either shown as 3 concentric arcs or 2 computer screens next to a box with an X in it)
- (14) A dropdown should appear with RPiHotspot on it. Check that there is a green tick next to RPiHotspot



- (15) If no green tick, select the RPiHotspot mode. A box should appear asking for a Pre Shared Key. For this enter 'raspberry' and press OK



- (16) If previously you had 2 screens with a cross, it should have changed to 3 concentric arcs now. The communications R-Pi is now connected to the chariot R-Pi.
- (17) If a problem occurs like RPiHotspot not showing in dropdown, or failure to connect after Pre Shared Key is entered; check that the Chariot Pi is up and running with green light flashing and then reboot the Communications Pi. Then try entering the Pre Shared Key again.

Connect VNC Viewer to make Communications R-Pi Screen Display the Chariot R-Pi Screen

- (18) From Desktop > Enter Network-Switch folder > Double click on vncconnector.sh
This should run vnc viewer and cause a new screen to appear within your existing one that represents the chariot pi. Anything run on this screen will affect the chariot pi.

The Chariot R-Pi will be running the robot systems, but the 'Communications R-Pi should now provide the interface to the Chariot R-Pi for development and control. On your monitor the Chariot R-Pi GUI (identified by the two computer screen icon) should be in a window on top of the Communications R-Pi GUI (identified by the concentric arcs icon).

- (19) Load the test file **MotorTest.py** and confirm the motors can be controlled via the Communications R-Pi over the WiFi.

3.6 Navigation

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.

Raw Navigation Using Odometers

Global Positioning System (GPS or SATNAV) is normally the answer to cross country navigation, but we are operating indoors and GPS does not work. The distance travelled by the two drive wheels is therefore measured by two odometers. 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."

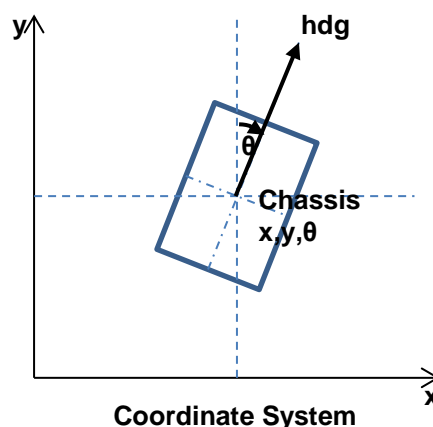
Teddy Roosevelt

The Test Programmes

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.

Coordinate Scheme Used for the Autonomous Programme

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 ' θ ' are measured in standard compass format with zero degrees being North and angles increasing positive clockwise so that East is 90 degrees.



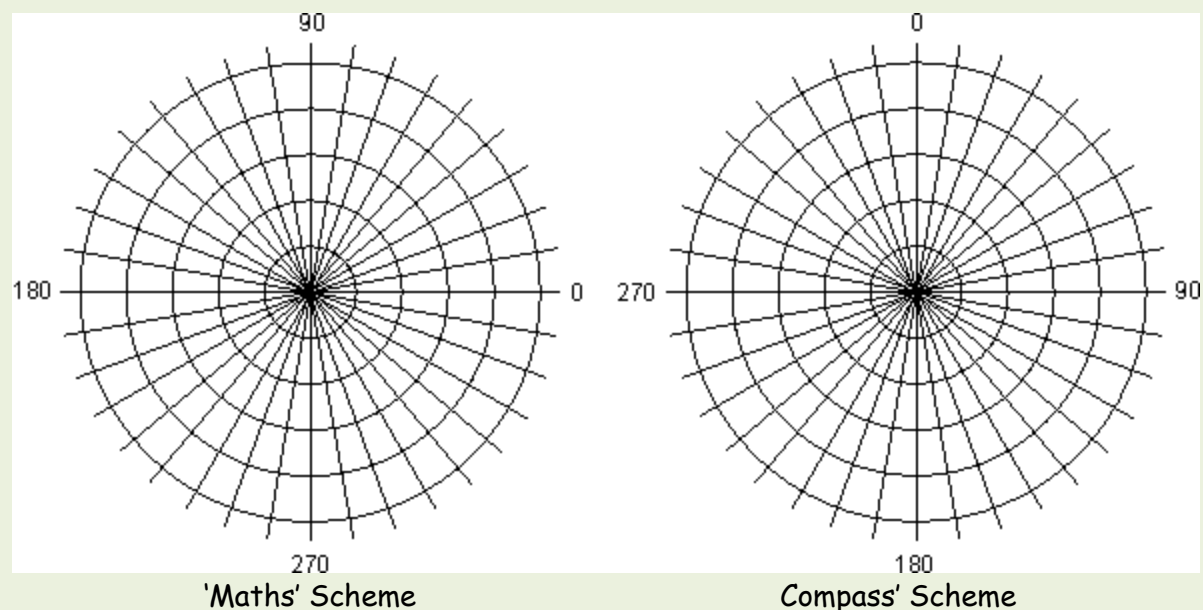
Geek's Box: Cartesian positions or coordinates are determined according to the *east/west* (x axis) and *north/south* (y axis) from the origin. Points or coordinates are indicated by (x,y). The origin is usually, assigned the value (0,0).

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 (zero degree) ray points off toward the right (East), 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 (zero degree) ray points upward (North), 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.



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. For reasons of cost, accuracy 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 ??)

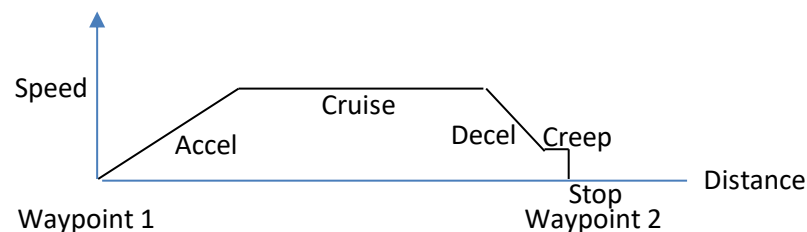
Longitudinal and Lateral Speed and Turn Rate Profile

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.



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.

3.7 **Calibrating the Motors and Wheels** [Desktop/TestPrograms/ forward-turnCal.py](#)

Open the file “**forward-turnCal.py**”, Click **File > Save As** and save the file as **forward-turnCal2.py**

This file combines motorTest.py and odometerTest.py to make the motors drive the chariot and display the odometer readings during some specific test manoeuvres. This enables you to calibrate the motors and wheels of your particular chariot in a straight line and a turn on the spot.

Two new functions have been added to handle the longitudinal and lateral control of the chariot. As the chariot moves forward the longitudinal function keeps track of the distance-to-go to the next waypoint and schedules an acceleration and deceleration that will try to stop the robot at the waypoint without skidding. The ‘Lateral’ function does the same thing for a turn-on-the-spot.

At the start of the ‘main’ function there are a number of parameters that you can alter to control the test manoeuvres. There are also some specific parameters to calibrate your robot. These are grouped under ‘Calibration parameters’.

Test 1. Calibrate Wheel Track, Wheel Diameter and Motor Bias

When you tested your Chariot in Manual Mode under radio control, you needed to adjust the forward-back trim and left-right trim slider controls on the transmitter to ensure the Chariot did not move when the two control sticks were in the central position. Adjustment of these trim sliders corrected for both a difference in drive wheel diameter and a difference in power output from the two motors. Without this trim adjustment you would have had great difficulty in controlling the chariot in a straight line as you would have to apply a constant small deflection to the left-right stick.

In the Autonomous Mode we have to do a similar trimming procedure and we have specific calibration parameters in the software to correct for differences in both wheel diameter and motor power.

First we need to stop (temporarily) the autonomous system trying to correct these parameters for us, as it will be much more accurate and efficient if we do as much as possible of these basic corrections ourselves. We do this by selecting the parameter (lateralCorrect = False).

If the two drive wheel diameters are different, the odometers will calculate different differences when the robot moves forward.

Calibrate Wheel Track by Measurement

- (1) Measure the distance in mm across the chassis between the centreline of the left and right drive wheel tyres. Change the wheelTrack variable number if required.

Test 1: Calibrate Motor Bias

- (2) Under #Longitudinal and Lateral Parameters that control the test manoeuvres: Set to:

lateralCorrect = “False”	#Automatic correction for heading drift in a straight line Off
reqdWptDist = 2000	#Distance to move forward in mm
reqdWptHdg = 90	#Heading to turn from current heading in degrees
fdwpd = 35	#Select a reasonable speed
turnSpd = 40	#Due to friction during a turn, you will require slightly more turn power

- (3) In a protected space mark out a straight course of at least 2m (2000mm) and stick a strip of masking tape across the course as a start line.

We want the chariot to start off with both wheels applying equal force. It is therefore important to ensure that both wheels are at the front limit of their deadband/gearbox backlash. With your

fingers, rotate the bottom of the drive wheels backwards until you feel significant resistance. Then place the centre of the Chariot across the start line.

- (4) Run the '**forward-turnCal**' file by pressing the Up arrow key. The chariot should move forward 2000mm and stop.
- (5) If it curves slightly to the right; under Chassis Parameters, change integer motorBias to 1. If it curves slightly to the left; under Chassis Parameters, change integer motorBias to -1.
- (6) Repeat the test and adjust motor bias until the chariot follows a roughly straight course. The motorBias integer (whole number) should be small; typically less than 4. If it is much more, investigate the motor electrical connections and tyre rubbing.
- (7) Repeat the test five times to get an average best result.

Test 2: Calibrate Wheel Diameters by Odometer Longitudinal Distance

- (8) Under #Calibration Parameters; change the parameter: lateralCorrect = "True"

This will apply a lateral control correction that will compare the two odometer readings and calculate the Chariot Heading. If the Heading diverges from the Track to the waypoint it will apply a small amount of turn to regain the correct heading and track. This should result in the Chariot maintaining a more accurate straight line to the waypoint.

- (9) Repeat the test above and measure the exact distance travelled with a tape measure. Record the two odometer readings. Repeat the test five times.

Actual Distance Measured mm	Left Odometer Reading mm	Left Error mm	Right Odometer Reading mm	Right Error mm	Notes
2050	2020	-30	2040	-10	Slight curve to left

- (10) Calculate the average difference between the actual distance travelled and the odometer readings. If there is an error it is probably due to a minor difference in wheel diameter due to tyre wear.

Note: The error between the actual wheel distance travelled and the odometer distance displayed in the screen box is what you are calibrating; The actual distance you asked the chariot to travel is not important for this test.

Fwd-Turn Calibrate			
	Fwd Speed	Turn Rate	Motor Bias
	127	127	1
	Left Odom	Right Odom	Heading
Odom Dist	2	7	
Wheel Dist mm	0	3	
ChassisDist & Hdg	0		0.0
Dist2Go & Hdg2Go	0		0.0
Fwd Phase :			
Turn Phase:			stop

Fwd-turnCal Parameters Display Box

- (11) Calculate the wheel diameter required to correct each odometer error. Under Chassis Parameters adjust wheelDiaLt and wheelDiaRt and repeat the test to confirm you have the most accurate values you can; ideally zero error.

Note a change of 1mm in wheel diameter will change the circumference of the wheel by 3.142mm (π). The wheel rotates approximately 4.2 times to travel 2000mm so a change of 1mm in wheel diameter will result in a change of approximately 13mm in the distance travelled.

- (12) Repeat Test 1 (steps 2 to 7) to refine the motorBias calibration parameter
- (13) Reset lateralCorrect to 'True'

Test 3. Calibrate Longitudinal Acceleration and Longitudinal Deceleration

- (14) If the corrected odometers show a greater distance than 2000mm the Chariot has overrun the waypoint. Repeat the test above and again measure the exact distance travelled with a tape measure. Record the two odometer readings. Repeat the test five times.
- (15) If the average overrun or underrun is greater than 25mm and you think the acceleration is too quick causing the wheels too skid, adjust the accelLong parameter by say 0.1
- (16) If you think the deceleration is too abrupt and causing the wheels to skid, adjust the decelLong parameter by say 0.1

Test 4. Calibrate Lateral Turn Angle

- (17) Place the robot in the centre of the test area and exactly parallel to the left wall with the wheels at the front limit of their deadband/gearbox backlash. Press the Right arrow key to run the programme and the Chariot should turn about 90 degrees right. Measure the angle turned with a protractor and repeat the test five times.

Actual Angle Measured deg	Chassis Heading Digits deg	Heading to Go Digits deg	Heading Digits Error mm		Notes
94.2	97.2	-7.2	+3.0		Slight spiral outwards

- (18) If the average chassis heading digits error is not near zero, under #Calibration Parameters, adjust the frictionRt parameter and repeat the test. Reduce the friction number to increase the turn angle and visa-versa

Test 5. Calibrate Lateral Acceleration and Lateral Deceleration

- (19) If the Heading to Go shows the Chariot has overrun the turn and you think the turn acceleration is too quick and causing the wheels to skid, adjust the accelLat parameter by say 0.1
- (20) If you think the turn deceleration is too abrupt and causing the wheels to skid, adjust the decelLat parameter by say 0.1

Test 6. Repeat The Calibration Process for a Left Turn

- (21) Modify the programme to undertake a left turn and repeat the calibration process

If you are not experienced with python, you will find a block(s) of code in a file called `testProgramAnswers` that will help you.

You can repeat the odometer longitudinal and lateral calibration procedure at any time and particularly if you have been running the chariot on a rough surface that can remove rubber and reduce the diameter of the drive wheels. Also if you are operating on a new surface with a different coefficient of friction.

3.8 **Testing Straights and Turns Over a Simple Square Route** [Desktop/TestPrograms/routeTest.py](#)

Open the file “**routeTest.py**”, Click **File > Save As** and save the file as **routeTest2.py**
[Transfer the calibration parameters obtained in forward-turnCal into this programme.](#)

This file extends the previous file “**forward-turnCal.py** to command the chariot to follow a simple square course back to its start position. This is a very useful test programme as it builds on the previous test programs and confirms the calibrations obtained during the previous test programme. It also adds waypoint logic to show how the robot can follow a series of waypoints in sequence. It is therefore a very simple example of the main autonomous programme and is an introduction to the problems of navigation described in para 3.6.

The robot should follow a simple square course with straight legs of one metre and it should end up at the start position pointing in the same direction as when it set off. It will be a minor miracle if this happens the first time you run it as there are so many factors that can send the robot off course. If the start and end points are a long way apart do not be discouraged. Try to analysis the errors and see if they are due to a longitudinal error or a turning error. Go back and run the forward-turnCal test programme and adjust the appropriate parameters to bias the errors in the desired direction. Then repeat the routeTest programme. After several tests and tweaks you will better understand the parameters to change and the effect they have on robot movement.

For a Square to the Right, the route is defined by waypoints as follows:

Waypoint No	Waypoint X Coord	Waypoint Y Coord	Notes
0	0	0	Start Conditions
1	0	1000	Line to North
2	1000	1000	Turn Right & Line to East
3	1000	0	Turn Right & Line to South
4	0	0	Turn Right & Line to West
5	0	1	Turn Right to North
6	0	1	Stop at last Waypoint

The 1mm long line North at Waypoint 5 is required for the python maths function to calculate a turn back to North (zero degrees) and end up at the same heading as when it started.

Key programme parameters are displayed in a black window. In addition, a number of print commands display parameters in the python shell window which will help you analyse what the robot is doing during each phase of each leg.

Try adding a new route by changing the waypoints to undertake a figure of eight and end up at the start position.

Try selecting a sequence of waypoint coorinates that will take the chariot round obstacles such as desks and chairs and through fairly narrow gaps. If the chariot consistently diverges from planned track, tweak the waypoint coordinates. You will then have quite an impressive demonstration of the raw navigation possible before you start updating position using the sensors.

Please send us a video of your robot going round a simple obstacle course.

To enable you to better understand the code in this routeTest programme the following text takes you through the programme, and tries to explain the sequence and logic behind it.

Programme Logic.

The first part of the programme:

Imports python libraries,

Designates the waypoints of the desired route,

Initialises display parameters of the Pygame display window

Sets up the GPIO pins

Defines the specific programme functions.

setSerial()	Set up the serial interface UART used to communicate with the Chariot Master Motor Drive Board
read_Odometers()	Reads angle of the magnet in the wheel axle in bits
handle_rollovers()	Detects rollovers and calculates continuous odometer distance in bits
longCtrl()	Handles the longitudinal chassis movement algorithms and longPhases
latCtrl()	Handles the lateral chassis movement algorithms and latPhases
angleDiff()	Calculates the difference between two angles in compass coordinates

The main() function of the program first defines and initialises all the parameters and then enters into a continuous control loop.

It initialise the pygame display window, serial data link, local variables and Calibration Parameters.

It assumes the chariot is placed at the start waypoint 0 heading towards waypoint 1.

It calls the readOdometers function once to obtain the raw odometer distance and angle information for the start position (zero distance). It then stores the angles as previous values.

It zeros wptDist, chassisDist and odomDistLt & Rt

It Calculates the first wptHdg from the Waypoint array assuming the robot is heading towards waypoint 1 and then converts the wptHdg to compass notation wrt North +ve clockwise.

It Initialise chassisHdg and startLegChassisHdg as the initial wptHdg and selects wpNo 0.

The Continuous loop, within which commands are actioned endlessly until the program is stopped.

1. Read Odometers

Call read_Odometers() and handle_rollovers() functions

Correct sign of right odometer and convert odomDistLt & Rt (bits) into total distance moved by wheel in mm, totWheelDistLt & totWheelDistRt.

2. Change Waypoint

A waypoint change is actioned when both the turn & line are complete and longPhase & LatPhase = 'end'.

Note: The wpNo remains 0 until an arrow key is pressed to start the Chariot. This keypress selects the longPhase & latPhase to 'end'.

This section also calculates the distance and heading to the new waypoint (wptDist and wptHdg) and re-initialises the chassisDist and dist2Go for the new leg.

The new chassisDist does not include the overrunDist as this is usually small and the vectors are in different directions. The overrunHdg, however is a significant error and chassisHdg is therefore calculated as a continuous chassisHdg throughout the complete route.

3. Determine the legMode

This code section determines whether a turn and/or straight line is required to reach the next waypoint. The turn is always actioned first.

If the waypoint heading change is not zero and a turn has not already been done; a turnRt or turnLt is actioned.

Converting odometer readings into chassis headings is different for lines and turns due to the offset wheel configuration. chassisHdg must therefore be reset at every change of leg mode and the heading calculated as if it was the first turn in the programme.

After the turn is complete, if the new wptDist is greater than zero a lineFwd is then actioned.

The startLegDistLt & Rt, chassisDist, startLegChassisHdg and hdg2Go are all initialised for the new leg.

3. Calculate Longitudinal and Lateral control corrections.

First the legWheelDistLt & Rt are updated so that the correct heading is obtained during turns.

Depending on the legMode, key parameters are passed to the longCtrl() and latCtrl() functions and returned.

The longCtrl() function handles the longitudinal chassis movement algorithms.

The speed profile has 5 phases: Accelerate, Cruise, Decelerate, Creep, and then Stop(end) at the next waypoint without skidding.

For a 'lineFwd' the new chassisDist derived from the odometer readings and the dist2Go is calculated.

The accelSpd is a function of the chassisDist from the waypoint the chariot has just left.

The decelSpd is a function of the dist2Go to the next waypoint. Both can be adjusted by calibration parameters.

The function then determines what phase of the leg profile is actioned and the fwd speed demand required.

The latCtrl() function handles the lateral chassis movement algorithms.

If the legMode is lineFwd,

A deviation in heading from the wptHdg (=hdg2Go) is calculated from the difference in distance travelled by each wheel:

legChassisHdg in Radians = difference in distance / wheeltrack

legChassisHdg in degrees = difference in distance *0.24 deg [180/pi/237]

The heading error (hdg2Go) is addressed by a turn correction which is in proportion to hdg2Go and decelLat.

If the legMode is turnRt or turnLt,

The chassis heading in degrees with respect to North (y axis) is calculated using the difference in distance travelled by each wheel.

Due to the offset position of the two drive wheels and odometers, the wheels skid laterally during a turn-on-the-spot and an additional correction factor of 0.58 is required. Finally a small correction for friction is used to calibrate the turn. (The maths is covered in Appendix ?)

The turn rate profile is Accelerate, Cruise, Decelerate, Creep, and Stop(end) the turn at the next waypoint heading without skidding.

The accelRate is a function of the legChassisHdg and accelLat.

The decelRate is a function of the hdg2Go and the decelLat.

4. Action Key Presses

'Up arrow key' starts the movement by setting the latPhase and longPhase to "end" which actions a waypoint change from 0 to 1

'space' will pause the movement by causing the programme to loop around the Action Key presses code block until another key is pressed.

"g" restarts the movement. Depending on which phase is in progress the wheels may skid when the programme pauses and restarts.

'End' will exit the program and also exit the continuous programme loop1.

5. Send Output to Motors

This is a red code block as it contains safety code to ensure that the fwd and turn integers sent to the chariot master motor drive board are always positive and with the range of a single 8 bit byte to avoid a dangerous overflow that could cause a full power runaway.

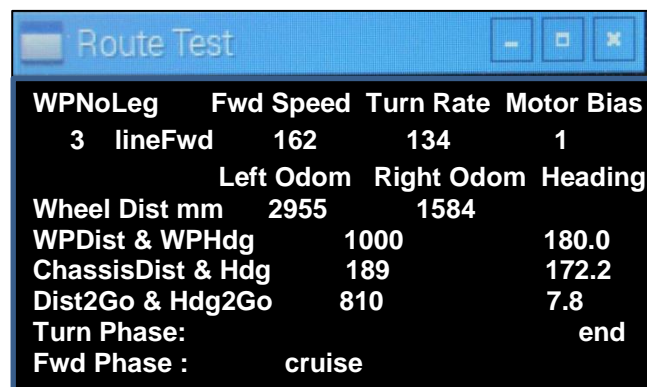
6. Display Parameters

This block displays the captions and parameter values for the programme and allows you to view the progress of the chariot as it proceeds from waypoint to waypoint and cycles through the leg phases.

Loop Timing

The Master Motor Drive Board expects regular updates of the fwd and turn parameters every 16ms. If there is a significant delay in receiving these parameters the Master Motor Control Board will assume a programme error and start a safety shut down. The programme will also be extremely sluggish. Waypoints will be overrun and heading corrections may cause a large divergent oscillation.

Printing data to the screen takes a significant time but a minimum of 16 msec is maintained between updates to ensure commands are actioned correctly by the Master Motor Control Board. The elapsedTime print out will show you the current loop iteration time. The increase in time during a waypoint change calculation is OK because the chariot is stationary at this time.

A screenshot of a software window titled "Route Test" with a black background and white text. The window displays various parameters for a robot's movement, including waypoints, speed, turn rate, motor bias, and distances. The text is organized into a table-like structure with some parameters spanning multiple rows.

WPNoLeg	Fwd Speed	Turn Rate	Motor Bias
3 lineFwd	162	134	1
Left Odom		Right Odom	Heading
Wheel Dist mm	2955	1584	
WPDist & WPHdg	1000		180.0
ChassisDist & Hdg	189		172.2
Dist2Go & Hdg2Go	810		7.8
Turn Phase:			end
Fwd Phase :	cruise		

RouteTest Parameters Display Box

3.9 Testing the Sensors and Actuators

The Stepper motor, Servo motor, Infra-Red sensor and Ultra sonic sensor are all controlled from the R-Pi via a serial bus called I2C.

Geek's Box: The I²C Data Bus is a Bi-directional Synchronous Data Bus that is often used to connect peripherals such as sensors or memory chips to a microprocessor. It is extremely versatile and can work in either direction with multiple Master Controllers and Slave Devices. We use this type of bus with one Master Bus Manager (The R-Pi) to control the Sensors Interface Board (called the Slave). This board has two additional 3 pin header pins to allow you to interface additional 3v or 5v I2C buses to control several slaves such as interface boards or other commercial sensors.

The R-Pi I2C bus operates at 3.3v so we route the bus through the voltage converter to be compatible with the 5v used by the Sensors Board PIC. The I2C bus uses two wires plus a ground wire. The data line is called SDA and the clock line is called SCL. Each of the two wires normally require a 10K 'pull-up' resistor to be fitted between each wire and the positive voltage (5v or 3.3v). This is to allow either end of the wire to pull the signal voltage low. In our application the R-Pi has these resistors already fitted internally to the two designated I2C pins.

The first byte transmitted by the master (R-Pi) contains the destination **Address** of the bus and which direction the data is to be sent. The Sensors Interface board (I2C slave) will match the address sent by the R-Pi and if it is a match it will interrupt what it is doing and receive and action the I2C instructions. Our Sensor Interface Board (Slave) has an address of 4. The least significant bit of the address byte (LSB) is a zero for a WRITE command (R-Pi to Sensors) and a 1 for a READ command (Sensors to R-Pi).

In our application, following the Address with a WRITE command, the Slave will receive 4 Bytes of 8 bits each. The first Byte is the **Control** byte which identifies the sensor and tells it what type of action it should do. (e.g. Scan to a particular angle and send back the minimum range it measures). The final 3 Bytes then contain data specific for that sensor or actuator (e.g. The two angles which the stepper motor should scan between).

In our application, following the Address with a READ command, the Slave will reply by loading the bus with 4 bytes of data specific to the sensor being addressed (e.g. the current angle and range of the I-R Sensor and its stepper motor).

As we only have one sensors board the Address is always 4.

The Control codes for addressing various sensors, functions and data are contained in Appendix ?. Typical Control Codes that you will use in this programme are:

Common R-Pi I2C WRITE Control Codes:

Stepper Motor Reinitialise to centre datum	32+4+1=	37
Stepper Motor to go to fixed angle at three speeds	32+8+3=	41-43
Stepper Motor to scan between two angles at three speeds	32+12+2=	45-47
Servo Motor to go to fixed angle at three speeds	64+8+3=	73-75
Servo Motor to scan between two angles at three speeds	64+12+2=	77-79

Common R-Pi I2C READ Control Codes

Send Back Current I-R Range and Angle	96+16=	112
Send Back Nearest I-R Range and Angle of object in FOV	96+20=	116
Send Back Current Ultra-Sonic Range and Angle	128+16=	144
Send Back Nearest Ultra-Sonic Range and Angle of object in FOV	128+20=	148

If it is an **R-Pi READ** The Sensors Board will reply with n bytes of data as requested by the control byte and numbytes.

For the **Range Sensor** this will be 4 bytes containing the high and low bytes of the angle (approximate +/- tenths of a degree) Followed by the high and low bytes of the unscaled sensor distance (approximate mm).

3.9 Testing the Stepper Motor and Infra-Red Sensor

1. Open the file "**sensorsTest.py**", Desktop/TestPrograms/ **sensorsTest.py**
Click **File > Save As** and save the file as **sensorsTest2.py**

This test programme will be used to test and calibrate the Stepper Motor, the Stepper motor datum micro-switch and the Infra-Red range sensor.

The file contains new code to set up the I2C interface and two new functions to

- a. WRITE to the sensors board
- b. READ data back from the sensors

The main() function allows you to set up some test parameters.

2. Change the control number to action the required test in the order set out at the start of the function.
3. **Calibrate the Stepper motor datum.**
Control = 37 should cause the stepper motor to rotate anticlockwise until it hits the datum micro switch. It should then rotate clockwise and stop facing forwards. Depending in the exact position you have mounted the micro switch you will need to calibrate the parameter `steprDatum` by approximately 11 for each degree it is off centre.
4. **Calibrate the Stepper Motor Scaling**
The stepper motor scaling is determined by the mechanical construction of this sensor and the calibration number should not be changed from 11.32 steps per degree unless you are using a different motor to that provided.
5. **Calibrate the Infra-Red Sensor**
The infra-red sensor returns the range of the object in its field of view and displays it in the window.

Geek's Box: The IR Sensor measures the range of the object in its field of view as an analogue voltage. The PIC on the sensors board uses an AtoD (analogue to digital converter) to convert the analogue voltage into a digital word. This word is split into two 8 bit bytes and transmitted to the R-Pi where the bites are recombined into the original word and displayed in the test screen window.

Place a white obstacle pole in front of the sensor at a range of about 1m and move it sideways to get the minimum range reading. If this is not directly in front of the sensor it is likely the sensor has a squint. If your local optician is unable to provide appropriate glasses, you should place the obstacle pole directly in front of the chariot and adjust the zero position of the stepper motor as in paragraph 3 until the minimum range coincides with the centre of the pole.

6. Place the pole at a range of 2500mm range from the pivot point of the I-R sensor. Move the pole left and right and record the minimum range reading. Move the pole towards the chariot and repeat this test every 100mm.
7. Draw a graph of actual distance 'y' against the range reading 'x'. You should expect it to be approximately a straight line until you get to actual distances less than 300mm.

8. Draw the best fit straight line through your data points above 300mm. Chose two points on this line at $x = 500\text{mm}$ and $x = 1500\text{mm}$ and record the x and y coordinates of the line.

Using the straight line equation $y = mx + c$ substitute your numbers to obtain two simultaneous equations Solve these equations to obtain the slope of the line ' m ' and the offset ' c '
(If you are not familiar with simultaneous equations your maths teacher will be keen to help you)

You now have two calibration parameters to adjust the range of the sensor:

$$\text{True range 'y' = range''x' * m + c}$$

Enter these parameters in the programme as `IRslope` and `IRoffset` and repeat your test at a few ranges to confirm your I-R sensor is giving accurate readings.

3.10 Testing the Servo Motor and Ultra-Sonic Sensor

1. Save the file "**sensorsTest2.py**" and then save it again as "**sensorsTest3.py**"

This test programme will be used to test and calibrate the Servo Motor and the Ultra-Sonic range sensor. Your challenge is to write code very similar to that for the IR sensor to enable this programme to test both types of motor and both types of sensor. If you get stuck you will find suggested blocks of code in the file **testProgramAnswers.py**

2. In the function `screen_display(angle, distance, caption)` insert a block of code starting if `caption == "U-S"`:

Where you decide to add the digits for U-S Angle and U-S Range in the display, you must position a solid rectangle to blank the existing digits before writing new values.

Geek's Box: The Servo Motor angle depends on the length of the pulse sent to it. A pulse length of 1500 micro seconds should move it to a central position. A pulse length of about 1000 us should move it to the left limit and a pulse length of 2000us should move it to the right limit. We need to calibrate these angles to ensure we can move it the number of degrees we want. We therefore need both a scale factor and an offset parameter.

```
servoFixedAngle = int(servoFixedAngle+servoOffset)*servoScaling + 1500
```

3. In the code block `#Convert control codes into bytes for transmission` add code blocks to handle the Servo.

```
The servoFixedAngle = int(servoFixedAngle+servoOffset)*servoScaling + 1500
```

4. At the start of the while True loop add a code block to change the display between I-R and U-S each read to display both parameters.

5. After the heading `#select and scale readings for rangeUS and angleUS` add code blocks to
Get Current U-S Range & Servo Angle
Get nearest U-S Range & Servo Angle

Geek's Box: The Ultra-Sonic Range Sensor sends a pulse of ultrasound and measures the time between sending the pulse and receiving back an echo from an obstacle.

The PIC measures this time to an accuracy of one micro second which is the time taken for sound at sea level to travel 0.34mm (Speed of sound = 340 m/s at Sea Level (760mph))

The range of the obstacle in mm is therefore the time delay * 0.17 (remember the sound has to travel out to the object and back again)