COMP105P
1. Introducing the Robots

Version 2015-01

Mark Handley
M.Handley@cs.ucl.ac.uk

Contents

1 Introduction

On COMP1010, you will learn programming skills and gain practice at using programming to solve real-world problems by programming our robots to perform a range of tasks that increase in complexity as the term progresses.

In this note we introduce the robots that you will be working with, the simplest of the APIs that you will be programming against.

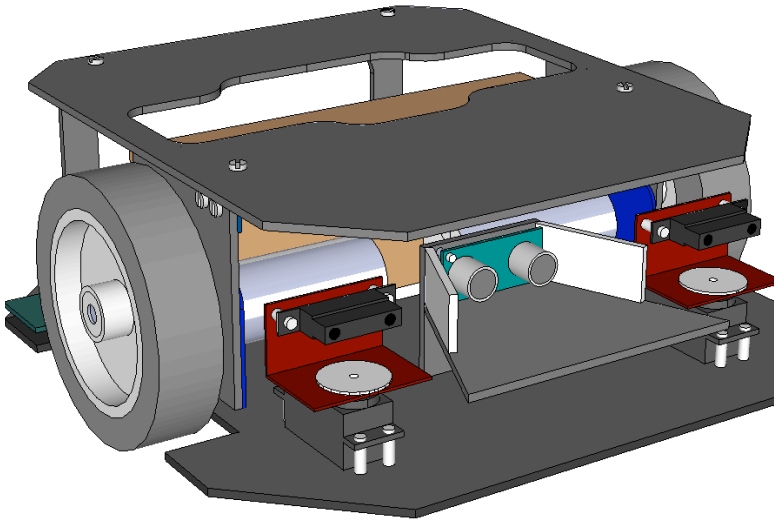

Figure 1: Front view of robot

Figure 1 shows the front view of one of the robots. Each robot has a range of sensors for discovering information about the external world, and two driving wheels that can be driven independently to maneuver the robot around. We will explain the robot's capabilities and limitations and how your software can access these capabilities.

2 Raspberry Pi

You will program a Raspberry Pi (RPi) computer that rides around on top of the robot. The Raspberry Pi is a 700MHz ARM-based computer, with 256MB of RAM. Ours have 4GB of flash storage in place of a hard drive, have WiFi networking, and run the Raspian version of Linux. You can remotely login to the RPi via ssh over WiFi using the command line in the same way you can remotely login to Linux or Mac computers.

You can upload your C source code onto the RPi, and use the C compiler to compile it to an ARM binary. You can then run your compiled code, and it can talk to the robot and cause it to drive, monitor sensors, and so on.

3 Firmware

The robot also has a second brain which handles all the low-level functionality. This is an Arduino Mega ADK microcontroller board, and all the robot hardware is wired into this microcontroller. The firmware and application programming interface (API) on the Arduino were developed specially for this course, and we can update the firmware as needed. If you have any suggestions for additional features that this firmware should provide, please let us know. Bear in mind though that we have deliberately limited the autonomous functionality provided on the Arduino to give you the opportunity to improve your programming and problem solving skills.

One of the purposes of the firmware is to allow the robots to run a self-test procedure, so we know if there is a problem with a particular robot. The self-test procedure runs entirely out of the Arduino, so it can be used if you're not sure if a problem is with the robot or with whatever code is on the Raspberry Pi. It also tests the communications between the Arduino and the Raspberry Pi. Self-test is described in more detail in Section 7.

The main purpose of the Arduino firmware is to implement the command interface. Your code running on the RPi sends commands to the Arduino. The firmware inteprets the comands, carries them out, and returns the results. Communication between the Arduino and the RPi occurs over USB using a simple ASCII command protocol. This same command protocol is supported by our robot simulator, so you can test out your ideas without needing access to the physical robots at all times. Beware though: although the command API is the same, the calibration of sensors, motors, and so forth will inevitably be different in the simulator.

4 Drive System

The robot has two DC motors controlled by a H-bridge motor controller that it uses to move forwards, backwards and turn. The robot's motor API allows you to send motor control commands to set the DC voltage on the each of the motors independently.

To a rough approximation, the speed of a DC motor is proportional to the voltage applied. However this is only a rough approximation. For example, commanding a low voltage might make the wheel turn slowly if the robot is lifted off the ground, but may be insufficient to turn the wheel at all when it has to carry the weight of the robot.

In principle, to drive forwards in a straight line, you would command the same voltage to both wheels. However, the motors are not precisely matched, so commanding the same voltage is likely to cause the robot to drive forward but also turn slightly to the left or the right. Also if the robot drives across an uneven floor, more drag can be exerted on one side causing the robot to slew to that side.

To combat this, the motors are equipped with motor encoders. These count the number of revolutions made by each motor. Your software can read back these counts and use it to compensate for the difference between the voltage you commanded and the speed you actually achieved. To drive in a straight line then typically requires a constant series of changes to the motor voltages to correct for deviations from what you wished the motor to do. The motor controller tries to do this for you, but doesn't always completely correct for errors, so if you want to know what the robot really did, you'll have to monitor the encoders yourself.

So long as you don't command very sudden changes in speed, and so long as the robot doesn't collide with any objects, the wheels will generally not slip. This means you can use the readings from the motor encoders

to map out the robot's path, turn 90 degree corners, and so forth. The encoders don't have arbitrary precision though, and small errors will accumulate over time, so you cannot be sure precisely where the robot is after a lengthy drive just based on the motor encoder readings. For any non-trivial mapping you do, you will have to combine information from the motor encoders with information from the other sensors.

The current being used by each motor can also be read back. This is broadly correlated with the load the motor is under, so if the robot is driving over a high-drag surface such as carpet, the current will be higher than on a smooth surface. You can use this as an indication that a wheel is under load before the load is evident from the wheel encoders. Unfortunately the current is only accurate to the nearest 100mA, which isn't very useful unless the robot is moving fairly fast. We don't know how useful being able to read current will be in practice, but the capability is there if you need it.

Finally the robot has some safety mechanisms. It's highly likely that your software will crash at some point, leaving the robot careering out of control. The robot firmware includes what is known as a *watchdog reset* — if you do not command the motor voltage for more than two seconds, the motors will shut themselves off. In addition, the robot has bump sensors on the front that detect collisions with objects. By default, the robot firmware will shut off the motors automatically if it detects a collision. This is useful when you're starting to explore the robot's capabilities, but it can get in the way for some tasks such as when you want to push a ball. For this reason, we provided a way for you to disable this safety mechanism if you really need to do so.

4.1 Battery Power

The robot is powered from a 12V 2.1Ah lead acid battery. These are sealed batteries and should not leak, but if you drop a robot it might be possible to break the battery and spill sulphuric acid. **Should this happen, do not attempt to clean it up yourself. Notify the lab demonstrator immediately.**

The battery should be charged when you receive the robot at the start of a lab and should provide enough battery life to last for the duration of the lab. When you're not actively using the robot but are programming it, switch the blue motor/sensor power switch off, leaving the red master power switch on. This reduces power drain in the motor control, servo and sensor circuitry. When you're ready for the robot to drive, switch the blue motor/sensor power back on. At the end of the lab, copy any code you've written off the robot and switch both power switches off.

The battery voltage will be displayed on the LCD screen when the robot is fully switched on. The reading is updated each time your software connects to the robot, but if you leave the robot switched on for a long time without connecting, the reading may be out of date. Your software can also poll the battery voltage. A well charged battery should read in the region of 13.5V when not under load, and should drop to around 12.0V when it's becoming discharged. Due to some voltage drop in the system, software monitor of battery voltage generally underreads by about 0.4V. Thus you should stop using a robot when the software reads a voltage below 11.5V. If the voltage gets much less than this, the robot may still function, but performance will get increasingly eratic. When the voltage drops below 12V, the LCD display will start flashing when the robot is idle, as a warning that the voltage is getting low.

Unlike with other battery types, the voltage drop is fairly linear as the battery discharges, so you can have some idea of how much battery time remains. When the motors are drawing a lot of current the measured voltage will drop somewhat — this is normal.

4.2 Emergency Stop

The large red button on the top of the robot is the Emergency Stop button. Press this to stop the robot. This button is monitored by the firmware, and if you press it, the firmware will switch off power to the drive motors. After you have pressed the stop button, the firmware will no longer respond to commands from the RPi until either you press the robot's green reset switch. Pressing reset will send a warning to your software that the Arduino has reset and any configuration changes have been lost. Emergency stop and reset do not reset the motor encoders, but turning the blue motor/sensor switch off will do so.

5 Sensors

The robots have a range of short and long range sensors to enable them to gather information about the world around them. On the front there are two infrared rangefinders, an ultrasonic rangefinder, and bump sensors that can detect collisions. On the sides behind the driving wheels there are sideways-pointing short range infrared sensors to detect whether an obstacle is immediately to that side of the robot.

5.1 Front Infrared Rangefinders

On the front are two Sharp GP2D12 infrared rangefinders. These sensors work by projecting a spot of infrared light onto objects directly in front of the sensor. They then use a CCD array to detect where the spot is seen - the output voltage of the GP2D12 sensor depends on the angle to the reflected dot, and by triangulation you can calculate the distance to an object in line of sight. The beam is narrow - if an object is just slightly to the side, it won't be seen. These sensors are fairly accurate for objects between 10cm and 80cm distance. They can't *reliably* see anything further than 80cm, though distances of at least 2 meters may be reported (sometimes with significant error). Objects closer than 8cm tend to cause the sensor to misread badly, reporting large distances.

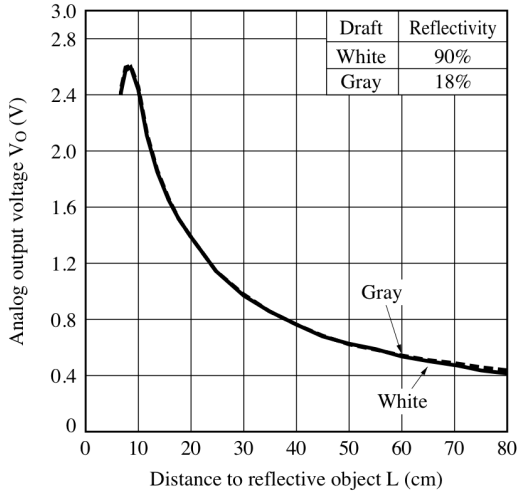

Figure 2: Response curve of Sharp GP2D12 rangefinder

The graph of output voltage vs distance for these sensors is shown in Figure 2. The output voltage is sampled by the Arduino giving a 10-bit value where 0 corresponds to 0V and 1024 corresponds to 5V. When your

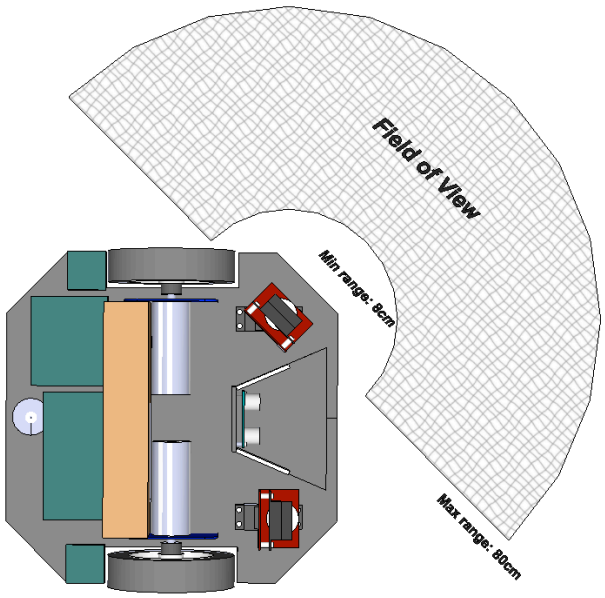

Figure 3: Field of view of the left IR rangefinder

software polls these sensors, it is this raw uncalibrated value that is returned. The full datasheet for these sensors is on Moodle.

As the beam width of these sensors is very narrow — imagine them as a laser rangefinder — then they are not really of very much use unless you can steer them to point in the direction you want. The IR sensors are therefore mounted on servomotors and can be panned left and right through about 180 degrees.

A servomotor is basically a small electic motor that internally includes a control circuit that works to actively move the motor to a specific position. The electronics feed a series of pulses to the motor, and the duration of each of these pulses corresponds to a particular position of the motor. This allows you to command the motor to move the sensor to a particular angle, and the motor will move to that angle and then attempt to hold the sensor at that angle even if a load tries to move it away. The servos we've used in these robots are fairly powerful — certainly overkill for moving the sensor around — but they're not indestructable so try to avoid anything getting in their way or manually forcing a servo to move position. Using a large servo means you can very quickly move the sensor from pointing in one direction to pointing in another, but there is still some delay. If you command the servo to move and immediately take a reading from the sensor, you'll get a reading from some angle in between where the sensor used to point and where you just commanded it to point. Thus if you want to know which direction an IR distance reading was taken from, you need to wait a short time for the servo to settle before reading from the sensor.

When you use the servo to scan the sensor backwards and forwared, with the left IR sensor the robot can see objects within roughly the field of view shown in Figure 3. The right sensor has a similar field of view on the right hand side. The sensors are approximately 7cm from the ground. Objects much above or below this height will not be seen. If is possible for the robot to see its own wheels if the sensor is rotated too far backwards - to avoid confusing your code it's best to avoid doing this.

The GP2D12 samples the distance approximately every 38ms. You can read the sensor reading from the firmware as often as you want, but the value won't update more often than this. This sample interval is fast

enough for most purposes, but if you want to scan the sensor backwards and forwards very quickly using the servo, you cannot read from the sensor fast enough to get a continous map of what is around the robot. If you want to scan using the servo, you'll need to experiment with how fast you can scan before you start missing objects of interest.

The IR sensors also only see objects which reflect infrared light. Glass doesn't reflect it well, polished surfaces may reflect it to a new position causing a misread, and some black surfaces may not reflect it well enough to get a good response, especially in bright sunlight. Most of the time though, these sensors are reasonably accurate within their specified range limitations.

Try to avoid pointing both IR sensors at the same object. The IR beam from one sensor may be picked up by the other sensor and cause incorrect distance readings. This may still happen if the sensors are pointed parallel if there is no object for several meters, because the beam pattern at that distance becomes large enough to overlap.

The biggest limitation of the IR sensors is at long distances. They're pretty reliable at distances of less that 80cm, but when there's no object for several meters they get very noisy, as they can no longer reliably detect the IR beam, and can sometimes and seemingly randomly read fairly low distances (down to 75cm or so).

5.2 Front Ultrasonic Rangefinder

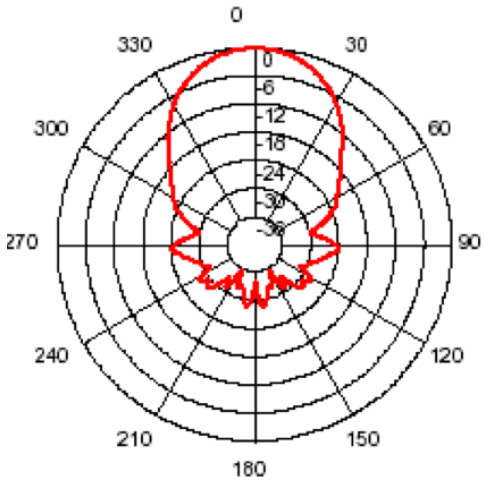

Figure 4: Field of view of SRF08 ultrasonic rangefinder

Figure 5: SRF08 ultrasonic rangefinder

In the centre of the front, the robots are fitted with a Devantech SRF08 ultrasonic rangefinder (Figure 5). This sensor emits a ping of 40KHz ultrasonic sound and times when the response is received. It is capable of detecting objects as close as 3cm, or as far away as 3m. It cannot tell you where in front of the robot an object is, but it can tell you how far away it is. The native beam pattern is more or less conical, with a response curve as shown in Figure 4. These sensors are not designed to be mounted as close to the floor as they are in our robots, so we've attempted to narrow their beam somewhat to avoid detecting reflections from ridges in the floor. This is why the ultrasound transducer is located in a fairly narrow horn structure. This works fairly well at short distances but can still be a problem at ranges of more than 1 metre.

These sensors are fairly slow - when you fire them, how long it takes to get an echo response depends on how far away the target is. If there's no target in range, it can take up to 70ms to receive a response. If there's a target close by, receiving a response should be faster. The firmware needs to poll the sensor until a response is available. This potentially large delay makes ultrasound a poor candidate for a synchronous request/response API. You would rather your code was doing something else while waiting for the response. The firmware compensates for this to a certain extent; for example, it polls the bump sensors to see if the robot has hit anything and checks if the emergency stop has been pressed while it is also checking for the ultrasound response. But the real solution is an asynchronous API, where responses to requests can arrive out of order or unsolicited. The robots also support an asynchronous API, but it is more difficult to use. This document only describes the simpler-to-use synchronous API.

In addition to being slow, the ultrasound sensor cannot tell the difference between the reflection of a ping your robot emits and a ping another robot emits. How much of a problem this is depends on how often everyone pings and whether you care about longer range responses. The SRF08 allows the sensor gain to be set, allowing the sensor to be used at reduced sensitivity when only short range measurements are needed. This can reduce sensor interference to some extent, and by default the firmware sets a relatively low gain. Last year, interference between robots did not seem to be a big problem.

Ultrasound can also misread. In particular, 40KHz sound can reflect off smooth hard surfaces and not bounce back to the robot. Our observations are that this tends to happen when the sound hits a wall section at a shallow angle (less than about 45 degrees). However, the ends and hinges in walls can still return an echo, so sometimes you'll get a reading, even at a shallow angle.

5.3 Side Mounted Infrared Sensors

On each side of the robot, just behind the driving wheel is a Sharp GP2D120 infrared rangefinder. These are very similar to the Sharp rangefinders on the front of the robot, except they are not mounted on servomotors, and they are designed for shorter range usage. Figure 6 shows the response curve for these sensors. As you can see, they're accurate from 3cm up to around 30cm. They cannot be used reliably at greater distances than this — though distances above 30cm may be reported you shouldn't trust them too much. Also an object closer than 3cm will cause a misread. The analog value is sampled by the Arduino, and the value returned is the raw value where 5V corresponds to 1024 and 0V corresponds to zero. Your software will need to convert this value into distance.

You cannot turn off the side IR sensors. *Try to avoid pointing the front IR sensors at the same place as the side sensors or both will misread.*

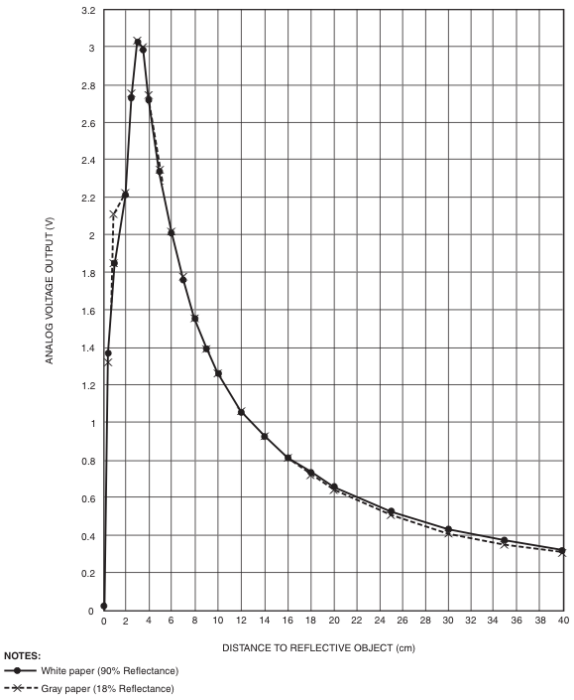

Figure 6: Response curve of Sharp GP2D120 side-mounted rangefinders

5.4 The Rear View

The robot has no sensors on the rear. As it can rotate on the spot, there really isn't much need for rear-facing sensors – just rotate the robot if you really need it to see what is behind. This doesn't stop you driving backwards, but it's probably not a good idea to drive backwards very far or very fast.

5.5 Bump Sensors

On the front the robot has two bump sensors that can detect if the robot has collided with an object. Of course, you are supposed to use the infrared and ultrasonic rangefinders to avoid objects, but sometimes the first you know of an obstacle is when you hit it.

By default, the firmware will turn off the motors when it detects a collision. If you send it a command to drive the motors forward and a collision is still detected, the command will be ignored so long as a collision is still being detected. The state of the collision sensors can be read by your code. By default, if a bump sensors detects a collision, it will continue to report a collision until you read the bump sensor state, even in the sensor is no longer triggered. The act of reading the sensor clears the latched collision state. This latching behaviour reduces the tendency to miss transient bump readings when waiting for a response to an ultrasound ping.

This default behaviour isn't always what you want. You can disable auto-stop if you need to allow the robot to push an object. You can also disable the latching of collision state. However, disabling both means that if you collide with an object and rebound slightly, if you are unlucky your reading of the sensor may be timed so as to miss the initial collision. This may delay stopping when you've collided with an object, resulting in wheelspin. If your code was using the wheel encoders to figure out its position, your estimate of location

can become much more inaccurate.

The bump sensors are digital - they report a value of 1 for collision and 0 for no collision.

6 Programming API

The robots will support two APIs over the USB interface:

- Synchronous ASCII
- Asynchronous ASCII

In this document we describe only the synchronous ASCII API. This is the simplest to use, but the lowest performance API.

6.1 Synchronous ASCII API

With this API, your program sends readable ASCII commands to the robot and the robot processes your command before processing any other command and then returns you a reply (if the command results in a reply) or an error message. ASCII here refers to the fact that the commands are in the form of printable characters (just like HTML is in printable characters). The alternative - a binary representation - would be more compact, but you can generate ASCII using the C sprintf command, and display responses using printf, so it's very easy to debug.

The USB interface to the robot is clocked at a relatively low data rate (USB can go faster, but the Arduino cannot). For this reason the commands are quite compact - we don't want to spend a lot of time just transmitting commands from the Raspberry Pi to the Arduino.

Commands take the form of a single letter, followed by a number or parameters separated by spaces. The command interpreter on the Arduino is written to be simple (and hence quick) but not to be robust. So only one space between the parameters, and all commands are terminated with a single newline character. Commands are case-sensitive - "`M`" is not the same as "`m`". All numbers in command parameters are decimal integers - no floating point numbers are allowed.

The commands supported are:

- `M` - set the voltage on a motor.
- `I` - set the position of an infrared rangefinder servo
- `S` - read a sensor value.
- `C` - configure the mode of something.
- `T` - display text on the LCD display.

6.1.1 Motor Commands

The `M` command can set the voltage on a motor. The format is:

`M <motor> <speed>`

Where `<motor>` can be either `L` or `R` to set the speed of the left or right motor respectively.

`<speed>` is a decimal integer in the range -127 to 127, where -127 is full reverse, 0 is stopped, and 127 is full forward. If you send values outside this range, the firmware will limit the speed accordingly. It is recommended you start off with lower speeds before you gain some confidence in your code.

For example, the following commands will make the robot turn on the spot in a clockwise direction:

```
M L 50
M R -50
```

In response to a motor command you will receive line containing a single "." from the robot so long as the parameters are sane. If the firmware does not understand your command it will return an error response. For example, if you sent:

```
M X 50
```

The firmware would respond with:

```
ERR "invalid motor name" M X 50
```

Often you'll want to change both motor speeds with minimal delay between changing the two. The API supports this with a single command:

```
M LR 50 -50
```

This sets the left motor speed to 50 and the right motor speed to -50, so the robot will rotate fairly quickly on the spot.

6.1.2 Infrared Servo Commands

The `I` command sets the angle of one of the two servos that pan the two front infrared rangefinders. The format is:

`I <servo> <angle>`

Where `<servo>` can be either `L` or `R` to set the angle of the left or right servo respectively.

`<angle>` is a decimal integer specifying the angle in degrees for the servo to point. The angle can be between -90 and 90 degrees, where 0 represents the middle position of the servos range - 45 degrees to the left of the robot for the left servo and 45 degrees to the right of the robot for the right servo. For both servos, negative values are to the left and positive values to the right.

For example, to point both front sensors straight ahead, you would use the two commands:

```
I L 45
I R -45
```

As with motors, you can also use a single command to set both servo positions. For example:

```
I LR 45 -45
```

This has the same effect as the two command above, but takes less time.

In response to a servo command you will receive line containing a single "." from the robot so long as the parameters are sane.

The servos are specifed as being able move 60 degrees in 0.18 seconds *under no load*. The IR sensors have some inertia, so slow the servos a little, but they're still quite fast. Even so, full left to full right takes over half a second - a long time by computer standards. If you do not give the servos time to reach the commanded destination before reading from the sensor, then you will still get a reading, but you won't know with any accuracy what direction corresponds to that reading. In some cases this may not matter, such as for object avoidance, but in others it will, such as mapping.

6.2 Reading from the Sensors

The `S` command is used to read from a sensor. The command takes the form:

```
S <sensor-name>
```

and the response is of the form:

```
S <sensor-name> <value>
```

Valid values for `sensor-name` are:

Name	Sensor type and location
IFL	Infrared rangefinder, Front Left
IFR	Infrared rangefinder, Front Right
ISL	Infrared rangefinder, Side Left
ISR	Infrared rangefinder, Side Right
US	Ultrasound, front
BFL	Bumper, Front Left
BFR	Bumper, Front Right
V	Battery Voltage
MEL	Motor Encoder, Left
MER	Motor Encoder, Right
MCL	Motor Current, Left
MCR	Motor Current, Right
IBL	Infrared reflectometer, Bottom Left
IBC	Infrared reflectometer, Bottom Centre
IBR	Infrared reflectometer, Bottom Right

The following combinations are also valid to read multiple sensors with a single command:

Name	Sensor type and location
IFLR	Infrared rangefinders, Front Left and Right
ISLR	Infrared rangefinders, Side Left and Right
BFLR	Bumper, Front Left and Right
MELR	Motor Encoders, Left and Right
MCLR	Motor Current, Left and Right
IBLC	Infrared reflectometer, Bottom Left and Centre
IBCR	Infrared reflectometer, Bottom Centre and Right
IBLR	Infrared reflectometer, Bottom Left and Right
IBLCR	Infrared reflectometer, Bottom Left, Centre and Right

No other combinations are valid.

and the response is of the form:

```
S <sensor-combination> <value> <value>
```

or

```
S <sensor-combination> <value> <value> <value>
```

where the multiple values correspond to the positions in the name. For example, reading BFLR, the first value is from the left bumper and the second value is from the right bumper.

6.2.1 Infrared Rangefinders

These are sensors `IFL, IFR, ISL` and `ISR`.

The front sensors are Sharp GP2D12 rangefinders and the side sensors are Sharp GP2D120 rangefinders. They are identical in operation, but the response curves for the front and side sensors are different.

To read from the front left rangefinder, you send the command:

```
S IFL
```

In response, you will receive a message such as:

```
S IFL 273
```

The command you sent is echoed back to you (later, this will allow the mixing of synchronous and asynchronous commands) followed by the value measured from the sensor.

In this example, the value is 273. This corresponds to an input voltage of $5 * 273/1024$ volts, or about 1.33v. From the curve in Figure 2, you can see this corresponds to a distance of about 20cm.

If, instead, you had issued the command:

```
S ISL
```

In response, you will receive a message such as:

```
S ISL 273
```

then this also indicates 1.33V, but on the side sensors the response curve in Figure 6 shows this indicates a distance of about 9cm.

To a rough approximation, to map the sampled GP2D12 value to distance in centimeters, you can use the function:

```
int gp2d12_ir_to_dist(int ir) {
  int dist;
  if (ir > 35)
    dist = (6787 / (ir - 3)) - 4;
  else
    dist=200;
  return dist;
}
```

This function was derived empirically by graphing the output of one sensor and curve fitting. You may be able to find a better approximation yourself; in particular, the limit at 200cm is artificial and probably rather optimistic. There will be some variation between sensors, so you may wish to calibrate your robot's sensors.

The mapping function for the GP2D120 sensors is similar, but with different constants.

A rough approximation can be obtained using:

```
int gp2d120_ir_to_dist(int ir) {
  int dist;
  if (ir > 80)
    dist = (2914 / (ir +4)) - 1;
  else
    dist=40;
  return dist;
}
```

You are, however, encouraged to experiment - for example, a table driven piecewise-linear fit may be more accurate.

6.2.2 Ultrasonic Rangefinder

To make a distance measurement with the ultrasonic rangefinder, you use the command:

```
S US
```

The response will be of the form:

```
S US 53
```

The returned value is the range in centimeters from the sensor (which is located a few centimeters back from the front of the robot). The ultrasonic sensor is slow to read - if you try to read it too often you'll slow down your control loop.

6.2.3 Motor Encoders

To read the count from a motor encoder, you use the command:

```
S MEL
```

Similarly `S MER` reads the right motor encoder. The response is of the form:

```
S MEL 7590745
```

The value corresponds to a 32-bit signed integer representing the number of encoder ticks since the robot was last reset. Negative values indicate the robot has driven backwards. Each tick corresponds to roughly 1 degree of motion of the driving wheel. If the count reaches the maximum number that can be represented in a 32-bit signed integer, namely 214748647 in decimal or 0x7fffffff in hexadecimal, then the next value to be returned will roll-over to -214748648. Such integer rollover is unlikely though, as the robot would have to drive about 170km for it to occur and the battery would have run flat first, but in general you should be careful about integer rollover when programming in C.

Should you need to, you can reset the motor encoder counts to zero using the `C RME` command. It is a bad idea to do this repeatedly while driving, because there will be a delay between the last reading taken and the reset, leading to underestimation of the distance driven.

Often you will want to poll the counts from both motor encoders frequently. To reduce delays you can use the command:

```
S MELR
```

The response is of the form:

```
S MELR 7590745 6752830
```

This indicates that left motor encoder is reporting 7590745 clicks and the right encoder is reporting 6752830 clicks.

6.2.4 Motor Current

To read the current used by a motor you use the command:

```
S MCL
```

The response is of the form:

```
S MCL 13
```

The value corresponds to the current in units of 100mA. Thus a value of 13 indicates 1.3 amps.

Units of 100mA are not very precise. If the motor is running unloaded (with the wheel off the ground), the current will not be enough to even register.

For a particular commanded value of motor voltage, the current will indicate the load the motor is under (how hard it is working).

6.2.5 Battery Voltage

The voltage of the 12V lead-acid battery providing the main power for the robot can be read using the command:

```
S V
```

The response is of the form:

```
S V 128
```

The value corresponds to the voltage in units of 0.1V. Thus a value of 128 indicates 12.8 volts.

The battery is becoming significantly discharged if the voltage falls below 11.5V. Try to avoid this happening by switching off the blue motor current switch when you do not need the robot to actually drive or use the IR servos, such as when you are reprogramming the robot.

6.2.6 Bumper Switches

The status of the front left bumper can be read with the command:

```
S BFL
```

Similarly use `BFR` for the front right bumper. The response is of the form:

```
S BFL 1
```

A value of 0 indicates the bumper has not hit anything; 1 indicates a collision.

The bumpers can operate in two modes - they can either latch a collision until the next read of the bumper state, or they can provide a instantaneous reading.

- In instantaneous mode, if the bumper switch triggers, releases, and then you read the bumper status, then the return value will be 0 as the bumper no longer detects the object.
- In latching mode, if the bumper switch triggers, releases, and then you read the bumper status, then the return value will be 1 as the bumper has latched the collision. If you subsequently read, the value will then be 0, as the first read cleared the latched state.

Both modes have their uses; you can choose whichever mode you prefer using the `C LB` configuration command. It is probably best to use latching mode when auto-stop is enabled, or you will not know why the robot has suddenly stopped.

Often you will want to poll the status of both bumpers frequently. To reduce delays, you can use the command:

```
S BFLR
```

The response is of the form:

```
S BFLR 0 1
```

This response indicates that the left bumper has not triggered and the right bumper has.

6.3 Text Command

You can send messages to the LCD display on the robot. The `T` commands will send a single line of text to the bottom row of the LCD:

```
T Hello World!
```

As with all commands, make sure it is terminated with a newline character.

6.4 Configuration Commands

The robots and simulator support a range of configuration commands.

Resetting Motor Encoders

RME resets the motor encoders to zero:

```
C RME
```

Configuring Bump Sensors

`LB` is the "Latch Bump sensors" configuratuion option. Setting it to zero disables latching of the bump sensors detection of a bump, whereas setting it to one enables bump latching.

```
C LB 0
```

Automatic Speed Regulation

The robot motor boards support automatic speed regulation, where the motor controller itself monitors the motor encoder readings and increases or decreases the voltage tothe motors to attempt to achieve the speed you commmand. This is normally what you want, so it defaults to "on", but you can disable it if you wish by setting it to zero:

```
C ASR 0
```

This only works on the robots.

Asynchronous Mode

This enables the asynchronous mode interface. I recommend you do not use this until you're comfortable with solving tasks in the (default) synchronous interface.

```
C ASYNC <parameter list>
```

Plotting the Robot's Trail

The `TRAIL` command only works on the simulator. It places a small dot at the robot's current position (at the center of the robot). You can use this to log the path the simulated robot takes.

```
C TRAIL
```

Plotting Specific Points

The `ORIGIN` command only works in the simulator. It sets the origin and direction of the X and Y axes for use by subsequent `POINT` commands.

```
C  ORIGIN
```

The `POINT` command only works in the simulator. It places a small dot at the specified absolute position.

```
C  POINT  <x>  <y>
```

Absolute position and orientation of the X and Y axes are determined relative to an origin set using the `ORIGIN` commands. The X and Y coordinates are in centimeters.

For example:

```
C  ORIGIN
C  POINT  0  13
C  POINT  11  11
```

This sequence will set the origin to the current position of the robot, orient the X and Y axes so that positive values of Y are forwards and positive values of X are to the robot's right, and then plots two points, one right in front of the robot and one to the front right.

One use for this is to plot where the IR sensors have seen objects. You'll have to calculate the positions using some trigonometry though, based on an estimate of the sensors position on the (simulated) robot and an estimate of which direction the IR sensors were facing at the time the reading was taken.

6.5 Mode Change Warnings

The API is not completely synchronous - the robot will sometimes proactively send mode change warnings. Typicaly your code will receive one of these in place of the response it was expecting from some other command. All such warnings start with a "`W`".

The current set of warnings include:

Name	Meaning
W START	Firmware has restared
W RUN	Robot has switched to normal mode (gives firmware version)
W STOP	Emergency stop has been pressed
W TEST	Self-test has been selected

The robot will also send some informative warnings at startup, including giving battery voltage and firmware version number. When your code is waiting for a response from a sensor and receives a warning, the simplest thing to do is to print out the warning, and continue to wait for the response it expected.

7 Self Test Mode

The robots have a built-in self-test function to ensure that the motors and sensors are all working properly. Place the robot about 20cm away from a wall, facing the wall. Press the square black self-test button at the

back right of the top of the robot until it latches. A self-test message should appear on the LCD display, and the test should begin.

The robot first performs a number of scans with the front IR sensors to establish with some accuracy how far away the wall is. If the robot is not directly facing the wall, the robot will attempt to adjust position until it is facing the wall.

Once it is happy it is facing the wall, it will turn 90 degrees to the left, take a reading with the right side sensor, turn 180 degress to the right, take a reading with the left side sensor, and turn back to face the wall. If all the sensors more or less agree, the test passes.

The test is not guaranteed to terminate if there is a problem, but you should easily be able to observe that it's not working properly.

You can terminate the test at any point by pressing the square button again.

8 Quirks

Being low-level hardware, the robots have a variety of unintentional quirks. Where possible, we've worked around them in firmware, but this isn't always possible. You need to help us keep the robots serviceable by reporting anything that isn't working right.

From the last two years experience, here are what you might see:

- The IR sensors may misread when pointed at anything reflective, or into tight corners.
- The IR sensors will misread if two are pointed at the same place, even is that place is several meters away.
- Ultrasound will bounce off walls it it hits them at a shallow enough able and not detect the wall.
- The ultrasound may stop responding and read 500cm constantly, even if objects are much closer. If so, please note it in the logbook, and report it to the demonstrator. We hope this problem has been solved in the current firmware.
- The ultrasound may stop responding and read 13cm constantly, even if objects are much closer. If so, please note it in the logbook, and report it to the demonstrator. We hope this particular problem has been solved by improving the ultrasound horns.
- Wheels may fall off! The wheels are held on by a grubscrew; even when done up tight, these have been known to work loose. When this happens the wheel will become sloppy on the shaft when changing direction, and mess up the robot's driving. In the extreme, a wheel may simply fall off. Please report loose wheels to the demonstrator and note it in the logbook. Please check the wheels are tight on the shaft when you receive a robot.
- Wifi may be flakey. The RPi boards and Wifi are new. In testing we discovered a few Wifi dongles that refused to connect or disconnected frequently, and these have been replaced. If you see similar behaviour, please note it in the logbook. If it is persistent, we'll replace the Wifi dongle.
- Blown motor board. Last year four motor boards failed, all under seemingly benign conditions. The symptom is that a wheel can no longer move, or can only move in one direction. This is different

from having a loose wheel, but may appear similar. If you think this has happened, please tell the demonstrator and note in the logbook what the robot was doing when it failed. We're trying to build up a picture of why this might happen.

- Bumpers may wedge. Occassionally we've seen the front bumpers wedge in the bumped position after a particularly bad impact. The symptom will be that the robot can only drive backwards, because the firmware is preventing it driving forwards. It should be obvious how to unwedge the bumper once you recognise this condition.
- Servos may get hot. We've had a lot of servos starting to get stiff and no longer rotate to the correct position, or rotate more slowly that desired. The first symptom is that the servo will get hot (and the battery will drain quickly). If a servo is hot, report it to a demonstrator, and we'll replace the servo.
- Sensor cables may break. We had two front IR sensor cables break last year, giving rogue front IR readings. If you suspect a sensor, run the "sensors" program on the Raspberry Pi and check if everything is working as expected.

In general, if the robot has a quirk or problem, please note it in the logbook, even if you think it isn't worth bothering the demonstrator. We're trying to build up a picture of whether any particular robot has intermittent problems.