HullPixelbot Code Specification

Version 3.0

Rob Miles

Arduino Connection

The program monitors the primary serial port on an Arduino device. Statements that are entered are executed immediately or stored internally using the RM command.

The connection between the Arduino and the host is configured to run at 1200 baud. This is to allow successful storage of program code in the EEPROM in the Arduino.

Statement format

All statements are a single, zero terminated, string.

The type of the statement is given as the first two characters of the string. These are called the *command characters.* The first command character specifies the "class" of the statement, and the second command character the particular statement in that class.

> *As an example the two characters "MF" specify the Move Forwards statement. The Move commands also contain a rotation command; "MR".*

The command characters can be given in upper or lower case. In this document they will be expressed in upper case.

Numeric Values

A statement may make use of one or more numeric values. All numeric values are integer. Within the description in this document the character sequence "ddd" denotes a numeric value. There are a number of forms of numeric value.

Literal value

In this case the value is a sequence of digits, with an optional leading sign character which can be + or –

> *As an example the string:*
>
> ```
> 1234
> ```
>
> *- would mean "one thousand two hundred and thirty four.*

Variable

Programs can use variables to store information they may be processing. Each variable has a unique identifier, which is comprised of a letter (a-z) followed by one or more characters that can be letters or digits.

> *As an example the string:*
>
> ```
> fred
> ```
>
> *- would be a valid identifier, whereas "2be" would not.*

The maximum length of a variable identifier is 10 characters and the case of the variable name is significant. There are a limited number of variables available. Variables do not need to be declared before they are used, but they do have to be given an initial value.

Reading

A reading is a value that provides information about the robot status. Each reading has a particular name. A reading is given as the % character followed by the name of the reading.

As an example the reading:

```
%dist
```

- gives the current value being supplied by the distance sensor.

There are presently two reading values available:

dist	gives the current distance value as an integer in centimetres
light	gives the current light value (actually the reading on ADC 0)

Expression

There is support for simple expressions. An expression is given as two operands separated by an operator. An operands can be a literal values, a variable or a reading. The two operands are separated by a single arithmetic operator.

As an example the expression:

```
%dist+tol
```

- would compute the current reading of the distance sensor plus the contents of the variable tol

The result of an expression can be used in a command, in a variable assignment and in conditions.

Comments

A comment statement starts with a hash character (#) and is ignored by the robot.

As an example the statement:

```
# My first program
```

- would be ignored when the program runs

Information: Initial character I

The information commands provide information about the status of the robot. Some information commands can also be used to specify the amount of information to be produced during program execution.

Version: IV

```
IV
```

The robot responds with the version of the software followed by a carriage return and a linefeed character.

As an example the output:

```
Version 3.1
```

- would be produced if the version is 3.1

Distance: ID

```
ID
```

The robot responds with current reading from the distance sensor followed by a carriage return and a linefeed character. The distance is given in cm.

As an example the output:

```
6
```

- would be produced if there is an object 6 cm from the robot distance sensor.

Program: IP

```
IP
```

The robot responds with a listing of the current program. Each output line is terminated by a carriage return and a linefeed.

As an example the output:

```
CLloop
VSfred=%dist
CCfred<10,close
PC255,0,0
CJloop
CLclose
VVfred
PC0,255,0
CJloop
```

- would be produced if this was the program inside the robot.

Status: IS

```
IS
```

The robot responds with current program status as a single digit value:

```
0       PROGRAM_STOPPED
1       PROGRAM_PAUSED
2       PROGRAM_ACTIVE
3       PROGRAM_AWAITING_MOVE_COMPLETION
4       PROGRAM_AWAITING_DELAY_COMPLETION5
5       SYSTEM_CONFIGURATION_CONNECTION
```

A status value of 5 indicates that the serial connection is actually to a configuration program rather than a live robot. This is used during configuration of the network processor in the Hull Pixelbot. You will never see this status value during normal operation. The configuration options for the robot are described in the document "Hull Pixelbot Configuration".

Messaging: IM

```
IMddd
```

This command is used to set the level of messaging that is produced by the robot when programs are running. The command is followed by an 8 bit decimal value that sets the messaging level. The message levels are set as bits in the messaging level value.

`STATEMENT_CONFIRMATION 1`	outputs a confirmation message after each statement. This will either be xxOK, where xx is the command, or xxFAIL: reason. Some commands, for example conditional jumps, will give additional information.
`LINE_NUMBERS 2`	displays the program position of each statement prior to execution
`ECHO_DOWNLOADS 4`	echoes each downloaded statement during a remote download
`DUMP_DOWNLOADS 8`	dumps a downloaded program before executing it

If the corresponding bitfield is set the program will output information as described. Note that these commands can result in significant traffic on the serial connection and are only intended to be used for debugging.

When the robot is restarted the messaging level is set to 0 (i.e. all messages are turned off).

As an example the command:

```
IM5
```

- would cause the robot to confirm commands and echo downloads. The value 5 is made up of 1 (for STATEMENT_CONFIRMATION and 4 for ECHO_DOWNLOADS).

Read sensors: IR

```
IR
```

This command reads all the sensors on the robot and provides a json formatted string that gives their values. The present version presents the information in the following form:

```
{"version":1,"distance":[3],"lightLevel":[311,312,317]}
```

The version value allows the consumer to know the version of the data packet. The distance and lightLevel arrays are populated with distance and light level values. The distance value is nominally in cm and the light level values are in the range 0-1023 and reflect the analogue values on ports A0, A1 and A2 respectively.

As an example the command:

```
IR
```

- would cause the robot to produce the following (your values may vary)

```
{"version":3.2,"distance":[34],"lightLevel":[445,488,568]}
```

Movement: Initial character M

Forwards: MF

```
MFddd,ttt
```

The robot moves the number of millimetres given by the decimal value ddd. If the number is negative the robot moves backwards that distance. If the robot is already moving this command will replace the existing one. The command can be followed by a comma and an optional time value that gives the number of ticks (tenths of a second) that the move will take to complete. If the time value (and the comma) are omitted the robot will move as fast as possible.

As an example the command:

```
MF100
```

- would move the robot forwards 100 mm as quickly as possible.

The command:

```
MF150,100
```

- would move the robot 150 mm and take 10 seconds to complete (remember that a "tick" is a tenth of a second).

The robot starts moving as soon as the command is received. If the move can be performed and the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
MFOK
```

Note that this does not meant that the move command has been completed, rather that the robot has received and understood the command and has started moving.

If you send another move command before the robot has finished this move the new command will override the existing one. The command CA can be used to wait for a move to complete.

If the time requested is not possible because the robot cannot move that quickly (for example move 100 mm in 1 tick) the move will not take place. If the `STATEMENT_CONFIRMATION` flag is set the robot replies with an error message:

```
MFFail
```

As an example the command:

```
MF100,1
```

- would request the robot to move 100 mm in one tenth of a second. This is not possible (unless you have seriously modified your robot) and so the command would be ignored and the MFFail message would be sent to indicate this.

Rotate: MR

```
MRddd,ttt
```

The robot rotates clockwise the given number of degrees given by the value ddd (there are 360 degrees in a circle). If the number is negative the robot rotates anticlockwise that number of steps. If the robot is already moving this command will replace the existing one. If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
MROK
```

Note that this does not meant that the move command has been completed, rather that the robot has received and understood the command and has started moving.

If you send another move command before the robot has finished this move the new command will override the existing one. The command CA can be used to wait for a move to complete.

If the time is omitted the robot will perform the rotation as quickly as possible.

If the time requested is not possible because the robot cannot rotate that quickly (for example rotate 200 mm in 1 tick) the move will not take place. If the `STATEMENT_CONFIRMATION` flag is set the robot replies with an error message:

```
MRFail
```

As an example the command:

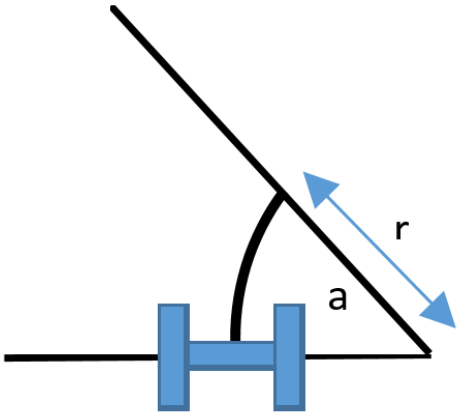
```
MR-90,50
```

- would move rotate the robot 90 degrees anti-clockwise and take 5 seconds to perform this move.

Move Arc: MA

```
MArr,aa,tt
```

This statement allows the robot to move in an arc. The first parameter is the radius of the arc, the second is the angular distance around the arc and the third is the time over which the move is to take place.



The figure above shows how this works. If the radius is positive the curve will be about a point which is to the right of the robot, and the robot will turn clockwise as it moves. If the radius is negative the curve will be about a point that is to the left of the robot, and the robot will turn counterclockwise. The centre point of the arc is on a line drawn between the two wheels.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
MAOK
```

Note that this does not meant that the move command has been completed, rather that the robot has received and understood the command and has started moving.

If the time requested is not possible because the robot cannot move that quickly the move will not take place. If the `STATEMENT_CONFIRMATION` flag is set the robot replies with an error message:

```
MAFail
```

As an example the following statement would cause the robot to traverse a complete circle and take a minute to perform this:

```
MA100,360,600
```

The radius of the circle is 100mm, the distance around the circle is 360 degrees and the move will take 600 ticks (remember that a tick is a 10th of a second)

Move Motors: MM

```
MMlll,rrr,ttt
```

This statement provides direct control of the speed and distance moved of each individual wheel motor. It is followed by three integer values that give the distance to be moved for each motor and the time to be taken for the move. When the motor has moved the specified distance it stops. The distance values can be signed, in which case the motor will run in reverse. This is the command that is the basis of the move and rotate commands above.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
MMOK
```

Note that this does not meant that the move command has been completed, rather that the robot has received and understood the command and has started moving. If the turn could not be performed a fail message is generated followed by a value which indicates the reason for the failure.

```
MMFail: d
```

The value of d gives the reason why the move could not take place:

```
1       Left_Distance_Too_Large
2       Right_Distance_Too_Large
3       Left_And_Right_Distance_Too_Large
```

As an example the command:

```
MM50,100,50
```

- would move the left wheel 50mm and the right wheel 100mm. The move would take 5 seconds to complete.

Check moving: MC

```
MC
```

This command can be used to determine whether the robot has completed a requested move operation. If the motors are still moving the robot replies with:

```
MCmove
```

If the motors are stopped the robot replies with:

```
MCstopped
```

Note that these messages are sent irrespective of the state of the `STATEMENT_CONFIRMATION` flag.

Stop robot: MS

```
MS
```

Stops any current move behaviour.

If the STATEMENT_CONFIRMATION flag is set the robot replies with:

```
MSOK
```

Wheel configure: MW

```
MWlll,rrr,sss
```

lll	left wheel diameter in mm
rrr	right wheel diameter in mm
sss	wheel separation in mm

This command can be used to configure the dimensions and spacing of the wheels fitted to the robot. The values are used to calculate all the robot movement. The values are stored in EEPROM inside the robot. The very first time the robot is turned on the dimensions are set to their default values:

Wheel diameter	69mm
Wheel spacing	110mm

These are the dimensions based on the stl files for the robot. If you fit wheels of a different size you will have to use this command to ensure that the distances moved by the robot are correct.

If the STATEMENT_CONFIRMATION flag is set the robot replies with:

```
MWOK
```

Note that these values are not validated in any way, so if you put silly dimensions you may find that silly things happen. And quite right too.

As an example the following statement would reset the wheel size and spacing to the default values:

```
MW69,69,110
```

View wheel configuration: MV

```
MV
```

This allows you to view the current settings of the wheel configuration values. The display is as follows:

```
Wheel settings
Left diameter: 69
Right diameter: 69
Wheel spacing: 110
```

Note that this message is sent irrespective of the state of the STATEMENT_CONFIRMATION flag.

Pixel control: Initial character P

The Hull Pixelbot can be fitted with a coloured “pixel”. This can be a single pixel, or it can be composed of a ring of 12 pixels. The Pixel control commands allow you to control individual pixels in the ring, or set an overall “flickering” colour. The rate at which the colour flickers can also be controlled.

Remote Coloured Candle: PC

```
PCrrr,ggg,bbb
```

rrr	red intensity in range 0-255
ggg	green intensity in range 0-255
bbb	blue intensity in range 0-255

Sets the pixel display to show a flickering candle of the given colour. This sets the colour of all the pixels in the display.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
PCOK
```

If any of the values are missing an appropriate message is displayed, for example:

```
PC255,
```

- would generate the error:

```
PCFAIL: mising colours after red in readColor
```

Note that this message is only output if the `STATEMENT_CONFIRMATION` flag is set.

As an example the command:

```
PC255,255,0
```

- would set the colour of the pixel to yellow (all the red and all the green).

Pixels Off: PO

```
PO
```

Turns off all the pixels in the pixel display.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
POOK
```

Pixels Random: PR

```
PR
```

Sets each pixel to a different, random, colour.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
PROK
```

Pixels Flicker update speed: PF

```
PFnn
```

Sets the flicker update speed for the pixels. The larger the number, the faster the pixels will change colour. The speed is given in the range 1 to 20. A speed of 1 is very gentle, a speed of 20 is manic. When the program starts the speed is set to 8. Values outside the range 1-20 are clamped, i.e. anything less than 1 is set to 1 and anything greater than 20 is set to 20.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
PFOK
```

As an example the command:

```
PF20
```

- would set the flicker speed to the maximum value (and make the robot appear angry).

Remote Set Individual Pixel: PI

```
PIppp,rrr,ggg,bbb
```

ppp	number of the pixel to be set in the range 0 to n-1, where n is the number of pixels
rrr	red intensity in range 0-255
ggg	green intensity in range 0-255
bbb	blue intensity in range 0-255

Set Individual to the given colour. The state and colours of the other pixels are not affected by this.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
PIOK
```

If any errors are detected in the values supplied an appropriate message is displayed instead of the OK message. See the PC command for details of the errors that can be produced.

As an example the command:

```
PI0,255,255,0
```

- would set pixel 0 (the first pixel) to be yellow.

Remote Crossfade color: PX

```
PXss,rrr,ggg,bbb
```

ss	face speed in range 1-20
rrr	red intensity in range 0-255
ggg	green intensity in range 0-255
bbb	blue intensity in range 0-255

Sets the pixel display to cross fade to a flickering candle of the given colour. This sets the colour of all the pixels in the display.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
PXOK
```

If any of the values are missing an appropriate message is displayed, for example:

```
PX20,
```

- would generate the error:

```
PXFAIL: mising colours after red in readColor
```

Note that this message is only output if the `STATEMENT_CONFIRMATION` flag is set.

As an example the command:

```
PX1,255,0,0
```

- would slowly fade the colour of the pixel to red.

Program Control: Initial character C

These commands are performed when the robot is running a program sequence. The can be entered directly into the robot, but they won't do much.

Pause when the motors are active: CA

```
CA
```

The program pauses while the motors are active. This allows a program to wait until a movement has completed. If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
CAOK
```

Note the reply is sent when the command has been received **not** when the robot has completed the pause.

Delay: CD

```
CDddd
```

The program pauses for the number of *ticks* given by the decimal value ddd. The number of ticks can be omitted:

```
CD
```

The program repeats the previous delay. If there was no previous delay the program does not delay. The delay starts as soon as the command is received.

A tick is a tenth of a second.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
CDOK
```

Note the reply is sent when the command has been received **not** when the robot has completed the delay. Ongoing move commands will continue to complete, and the robot will respond to other direct commands.

As an example the command:

```
CD50
```

- would cause the program to pause for five seconds.

Label: CL

```
CLcccc
```

This statement declares a label which can be used as the destination of a jump instruction. The label can be any number of characters and will be terminated by the end of the statement.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
CLOK
```

The robot will not presently detect if the same label has been declared multiple times in a program. In this case branches will always arrive at the **first** declaration of the label.

As an example the command:

```
CLloop
```

- declares a label called loop. This label can be used as a destination for jump instructions.

Jump: CJ

```
CJcccc
```

This statement causes execution of the program to continue from the given label. The label can be any number of characters and will be terminated by the end of the statement. The program must contain the label requested, or the program will stop.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with the following messages:

If the label is found and the jump performed the robot replies:

```
CJOK
```

If the label is missing from the program the robot replies with:

```
CJFAIL: no dest
```

As an example the command:

```
CJloop
```

- would cause the program to continue execution at the label Loop.

Coin toss jump: CT

```
CTcccc
```

This statement causes execution of the program to continue from the given label fifty percent of the time. The rest of the time the program continues. The label can be any number of characters and will be terminated by the end of the statement. The program must contain the label requested, or the program will stop.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with the following messages:

If the label is found and the jump performed the robot replies:

```
CTjump
```

If the label is found and the jump not performed the robot replies:

```
CTcontinue
```

If the label is missing from the program the robot replies with:

```
CTFAIL: no dest
```

As an example the command:

```
CTloop
```

- would cause the program to continue execution at the label Loop half the times this instruction was performed. The rest of the time execution would continue with the next instruction.

Jump when motors inactive: CI

```
CIccc
```

If the STATEMENT_CONFIRMATION flag is set the robot replies with the following message:

```
CIOK
```

As an example the command:

```
CIcomplete
```

- would cause the program to continue execution at the label complete if the motors are not active.

Measure Distance: CM

```
CMddd,cccc
```

This statement causes execution of the program to continue from the given label (ccc) if the distance sensor reading is less than the given distance value (ddd). The value is given in centimetres.

The label can be any number of characters and will be terminated by the end of the statement. The program must contain the label requested, or the program will stop. If the distance measured is greater than the given value, the program continues at the next statement.

If the STATEMENT_CONFIRMATION flag is set the robot replies with the following messages:

If the label is not present in the instruction the robot replies with:

```
CMFail: missing dest
```

If the label is found the robot replies with:

```
CMFail: label not found
```

If the label is found and the distance is less the robot replies with:

```
CMjump
```

If the label is found and the distance is greater the robot replies with:

```
CMcontinue
```

As an example the command:

```
CM10,close
```

- would cause the program to continue execution at the label close if an object is detected within 10cm of the robot.

Command Compare: CC

```
CCttt,cccc
```

This statement causes execution of the program to continue from the given label (ccc) if the given logical test (ttt) succeeds.

The label can be any number of characters and will be terminated by the end of the statement. The program must contain the label requested, or the program will stop. If the distance measured is greater than the given value, the program continues at the next statement.

If the STATEMENT_CONFIRMATION flag is set the robot replies with the following messages:

If the label is not present in the instruction the robot replies with:

```
CMFail: missing dest
```

If the label is found the robot replies with:

```
CMFail: label not found
```

If the label is found and the distance is less the robot replies with:

```
CMjump
```

If the label is found and the distance is greater the robot replies with:

```
CMcontinue
```

As an example the command:

```
CCcount==10,done
```

- would cause the program to continue execution at the label done if the variable count contains the value 10.

Variables: Initial character V

These commands provide simple data processing behaviours. They can be used interactively, but they are intended to be used by the HullPixelbotScript language which provides a higher level programming interface.

Variables clear: VC

VC

Removes all the variables from storage. This command is issued at the start of program execution. If your program wants to clear the variable storage it can do this as it runs.

Variable set: VS

VSnnnn=dddd

Sets the variable with the name to the given value. If the variable doesn't exist it is created automatically.

As an example the command:

```
VScount=0
```

- would cause the program to set the value of the variable count to zero. If the variable does not exist it will be automatically created.

Variable view: VV

VVnnnn

Display the value in the given variable.

Remote Management: Initial character R

These commands are performed to allow a new stored program to be downloaded into the EEPROM in the robot and to control program execution by the robot drive system.

Start Program: RS

This statement starts the execution of the current program (if present).

```
RS
```

This statement is obeyed immediately upon receipt. The program is started from the first statement in the program.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
RSOK
```

Halt Program: RH

This statement halts the execution of the current program (if present).

```
RH
```

This statement is obeyed immediately upon receipt. The program is halted. It cannot be resumed, it must be restarted.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
RHOK
```

Pause Program: RP

This statement pauses the execution of the current program (if present).

```
RP
```

This statement is obeyed immediately upon receipt. The program is paused. It can be resumed using the RR statement.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with:

```
RPOK
```

Resume Running: RR

This statement resumes the execution of the current program (if it has been paused).

```
RR
```

This statement is obeyed immediately upon receipt. The program is resumed.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with the following messages:

```
RROK
```

This is displayed if the program has been resumed.

```
RRFail:d
```

This is displayed if the program cannot be resumed. The single digit value following the Fail: gives the current program state as for the IS command.

Remote Download: RM

This statement stops the execution of the current program and switches off the pixel display ready for the receipt of a new program.

```
RM
```

The program is sent as a sequence of statements that directly follow the RM command. Each statement is stored in EEPROM for execution by the robot when the download is complete. A standard Arduino has enough internal storage for around 900 bytes of remote program storage.

Note that the RM command is terminated by a Carriage Return (CR) character (0x0D) as are all commands. Each statement in the program is separated from the next by the CR character. The end of the program is specified by an RE command (see next).

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with the following message:

```
RMOK
```

If the `ECHO_DOWNLOADS` flag is set the robot echoes each statement as it is received.

Remote Download Exit: RX

This statement completes a download started by the RM command.

If the `STATEMENT_CONFIRMATION` flag is set the robot replies with the following message:

```
RXOK
```

If the program was successfully received it is executed immediately.

If the `DUMP_DOWNLOADS` flag is set the robot dumps a listing of the received program as soon as it has been successfully received.

Sample Programs

SimpleDance

This program contains a single loop that will run indefinitely. It makes the robot move back and forth. When the robot is moving forwards the pixel is green, when the robot is moving back the pixel is red.

```
# Simple dance program
# Makes the robot move back and forth
# Rob Miles
CLloop
PC0,255,0
MF100
CA
MF-100
PC255,0,0
CA
CJloop
```

Move and turn

This program moves the robot forwards until an object is detected. When the object is detected the robot turns 90 degrees before continuing forwards.

```
#avoider
# move and turn away from objects
# Rob Miles
CLtop
CIstartmotors
CJdistanceTest
CLstartmotors
PC255,0,0
MF100
CLdistanceTest
CM10,hit
CJtop
CLhit
PC255,255,0
MR90
CA
CJtop
```

Move and turn five times

This program moves the robot forwards until an object is detected. When the object is detected the robot turns 90 degrees before continuing forwards. It does this five times. The fifth time an objects is detected the robot “gives up” and stops moving.

```
#avoider 5 times
# move and turn away from objects
# Rob Miles
VScount=0
CLtop
```

```
CIstartmotors
CJdistanceTest
CLstartmotors
PC255,0,0
MF100
CLdistanceTest
CM10,hit
CJtop
CLhit
VScount=count+1
CCcount==5,done
PC255,255,0
MR90
CA
CJtop
CLdone
PC0,0,0
```

Command Reference

Information Commands	
IV	Display version
ID	Return distance value in cm
IP	List the current program
IS	Display robot status
IMnn	Set Messaging level to nn. Value can be composed of: STATEMENT_CONFIRMATION 1 LINE_NUMBERS 2 ECHO_DOWNLOADS 4 DUMP_DOWNLOADS 8
IR	Read sensors and produce JSON string
Movement Commands	
MFddd,*ttt*	Move distance ddd in time ttt ticks (a tick is a 10^{th} of a second). Negative distance to move back.
MRddd,*ttt*	Rotate angle ddd clockwise in time ttt ticks. Negative angle for anti-clockwise
Marrr,aaa,*ttt*	Move in a clockwise arc of radius rrr, angular distance aaa in time ttt ticks. Radius is measured from the mid-point of the robot between the two wheels. A negative radius will result in an anti-clockwise arc.
MMlll,rrr,*ttt*	Move left motor lll and right motor rrr in time ttt ticks.
MC	Check robots moving. Produced message to indicate movement state
MS	Stops current robot move
MWlll,rrr,sss	Wheel configure. Sets left wheel diameter to lll, right wheel diameter to rrr and motor spacing to sss.
MV	View motor configuration.
Pixel Control	
PCrrr,ggg,bbb	Pixel candle of colour rrr,ggg,bbb
PO	Turn pixels off
PR	Set each pixel to a random colour
PFnnn	Set pixel flicker speed to value nnn. The value is between 1 and 20. 20 is fastest
PIppp,rrr,ggg,bbb	Set pixel ppp to the colour rrr,ggg,bbb. Pixels are numbered starting from 0
PXss,rrr,ggg,bbb	Pixel crossfade to specified colour. Speed of the fade is sss (in the range 1 to 20). Values of rrr,ggg,bbb given in the range 0 to 255
Program Control	
CA	Pause the program while the motors are active
CDnnn	Pause the program for the given number of ticks
CLcccc	Declare a label called cccc
CJcccc	Transfer program execution to cccc
CTcccc	Transfer program execution to cccc half the time
CIcccc	Transfer program execution to cccc if the motors are inactive
CMddd,cccc	Transfer program execution to cccc if the distance value is less than dddd
CCttt,ccc	Transfer program execution to cccc if the test tttt is true
Variables	
VC	Clear all variables
VSnnnd=dddd	Set variable name nnnn to the value dddd
VVnnnn	View the contents of the given variable

Remote Management	
RS	Start running the program loaded into the robot
RH	Halt running the program loaded into the robot
RP	Pause the program.
RR	Resume running a paused program
RM	Remote download. Commands that follow are stored in the robot for later execution.
RX	Remote download exit. The program storage ends, the robot will now respond to direct commands. The internal program is executed.