PSI SWARM SYSTEM

Reference Manual

Manual Version **0.81** - API Version **0.7 (C)** and **0.8 (C++)** - PCB **1.5** - October 2016

This document contains information about the hardware and software for the Psi Swarm robotic platform developed by York Robotics Laboratory.

Psi Swarm System

For PCB Version 1.5 and API Version 0.7 & 0.8

16th October 2016

Contents

Preface

The Psi Swarm robotic hardware platform, the software sources and documentation are the work of the York Robotics Laboratory, University of York. This issue of the document is for internal use only and not for public release.

Licence and Copyright Notice

Authors and Contributors

James Hilder

Alan Millard

Jon Timmis

Homero Silva Elizondo

Alexander Horsfield

For queries and other correspondence please email james.hilder@york.ac.uk

Introduction

The Psi Swarm robot is a complete, low-cost, standalone robotics platform developed primarily for use as part of a fully autonomous robotic swarm. The robotic platform is designed to be controlled using an MBED LPC1768 rapid-prototyping microcontroller board, which allows code to be easily created on any system with a USB-port and web-browser, without the need for any dedicated software tool-chain or drivers. The MBED connects to a pair of 40-pin sockets on the top of Psi-swarm robot. Additional hardware modules such as the BlueSMIRF Bluetooth-Serial communication board can be added to additional sockets on the robot.

The robot has been designed in-house at YRL (www.york.ac.uk/robot-lab) and features a versatile arrangement of sensors, actuators and communications devices specifically selected to allow for swarming interactions to take place. The robot is a basic 2-wheel, skid-steered platform using miniature DC motors with metal gear-boxes. The robot is powered by a single 3.7V 14500 cell (these batteries are the same basic size as a conventional AA or LR6-cell, but use Lithium-Polymer technology). At the core of the design is the ability to self-recharge using base-mounted contacts, allowing the creation of simple low-cost recharging zones within the swarm's operational area.

This document describes both the hardware that makes up the Psi Swarm System, and the basic API and software libraries that can be used in the MBED online compiler (and also exported for offline development using a variety of C++ programming tool-chains and IDEs). This version of the document is specifically for version 1.5 of the Psi Swarm PCB and version 0.7 of the API and software libraries.

The Psi Swarm Robot

The MBED LPC1768 Rapid Prototyping Board

The MBED LPC1768 is a small PCB designed to allow rapid prototyping for general microcontroller applications. It is based around the NXP1768, a 32-bit ARM® Cortex™-M3 microcontroller which operates at a 96MHz clock frequency and includes 32KB RAM and at least 512KB FLASH memory (1MB on newer boards) which appears as a USB-FLASH drive when connected to a computer. It is available to purchase from many electronics suppliers, including Farnell (uk.farnell.com), for roughly £40. It features a wide variety of interfaces and busses, including built in Ethernet, USB (host- and device-), CAN, SPI, I^2C, 6 channels of ADC, 6 channels of PWM output, a DAC and up to 26 GPIO pins [*certain pins are shared between interfaces so not all will be available at one time*]. Additionally on-board LEDs are connected to other GPIO pins on the microcontroller, and the USB connector can be used as a virtual RS-232 port for PC communications.

One of the key differences between the MBED and many other similar microcontroller based prototyping boards is the availability of an online compiler at www.mbed.org. This compiler provides the tool-chain and libraries necessary to quickly create C++ programs which can be compiled and installed onto the mbed. The bootstrap loader on the mbed board simply loads the most recent binary file that has been saved onto the FLASH memory upon reset, so the process of uploading new code to an mbed board is simple:

1. Navigate to "mbed classic developer" site at https://developer.mbed.org/ on a web browser
2. Log in *(creating a new account if necessary)* and open the compiler (see Figure 1)
3. Create the code

4. Compile the code and download the compiled binary *(.bin)*
5. Connect the mbed board to the computer using a mini-USB – USB cable and save the *.bin* file in the flash drive folder
6. Press reset on the mbed board – the new code should now run

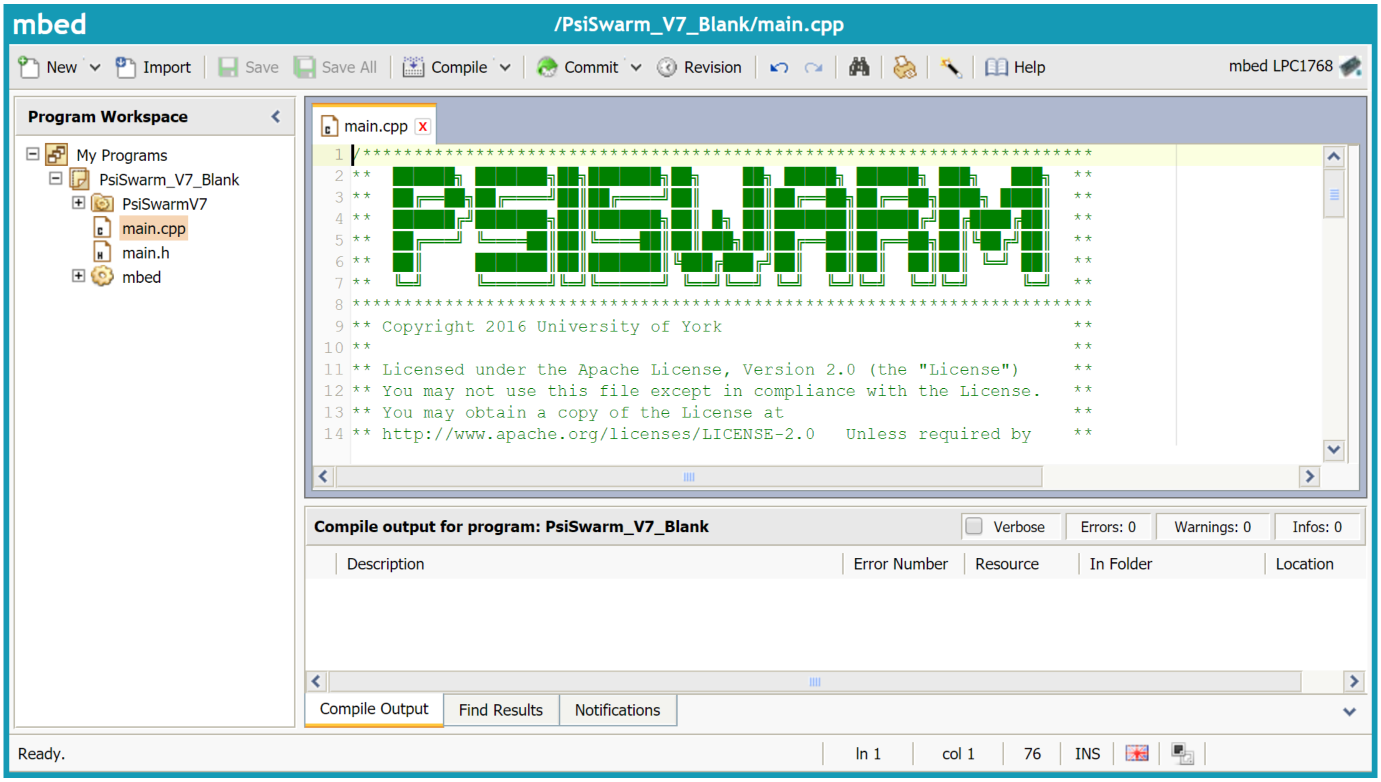


Figure 1: Screen shot of MBED Compiler at http://developer.mbed.org

In addition to appearing as a flash drive on a connected computer, the MBED can also be setup as a virtual serial port (this requires a driver to be installed in Windows but not needed for Linux). This makes it very easy to, for example, send debugging information back to a computer terminal using `printf` statements. Many different serial terminal emulators are available for different operating systems that can read data from the MBED; the authors personal preference is the free HTERM software available from http://www.der-hammer.info/terminal/. Baud-rates of up to 460,800bps are supported by the MBED (note that a lot of PC software will not recognise this baud-rate; 115,200 is recommended where maximum compatibility is needed). The MBED USB-Serial driver for Windows can be found at https://developer.mbed.org/handbook/Windows-serial-configuration.

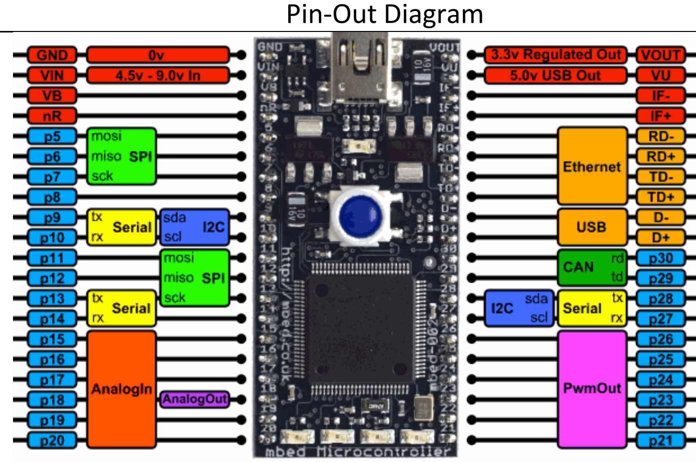
The mbed LPC1768 Rapid Prototyping Board	Pin-Out Diagram
	

Figure 2: The mbed LPC1768 Rapid Prototyping Board and its pin-out, highlighting available busses and GPIO pins.

The Psi Swarm PCB

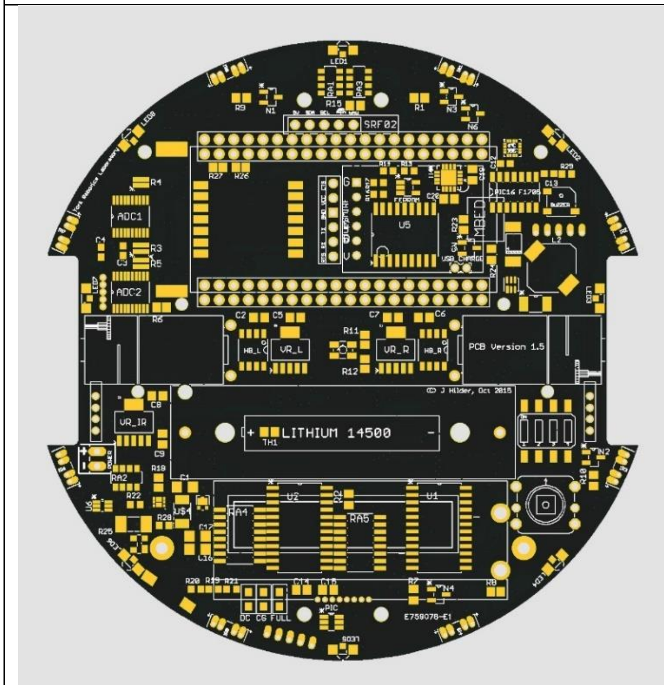
PCB Versions

The Psi Swarm PCB boards are designed in **Eagle** PCB design software (version 7.2.0) as 2-layer boards. They have been designed to meet **EuroCircuits** (www.eurocircuits.com) design requirements for class 6 boards; the minimum track width used is 0.2mm and the minimum drill size used in vias in 0.25mm. The board is ideally designed to be done in the standard pool on 1.55mm thick board, with optional black top and bottom soldermask, HAL or selective AU plated finish. The boards are approximately 108.3mm diameter circle with flattened sides to 103mm to allow for the wheel cut-outs. A silk-screen legend for both the top and bottom layers of the board is included. Version 1.5 of the PCB is dated October 2015; this revises the wiring of the wheel-encoders and the base colour sensor from the previous version 1.4, and provides a socket for an optional compensated 3D compass module. Version 1.4 revised the 5V power delivery stage with uprated components to prevent brown-outs that occasionally occurred on version 1.3. Version 1.3 added the secondary PIC controller, the speaker and revised the layouts of certain components from earlier versions.

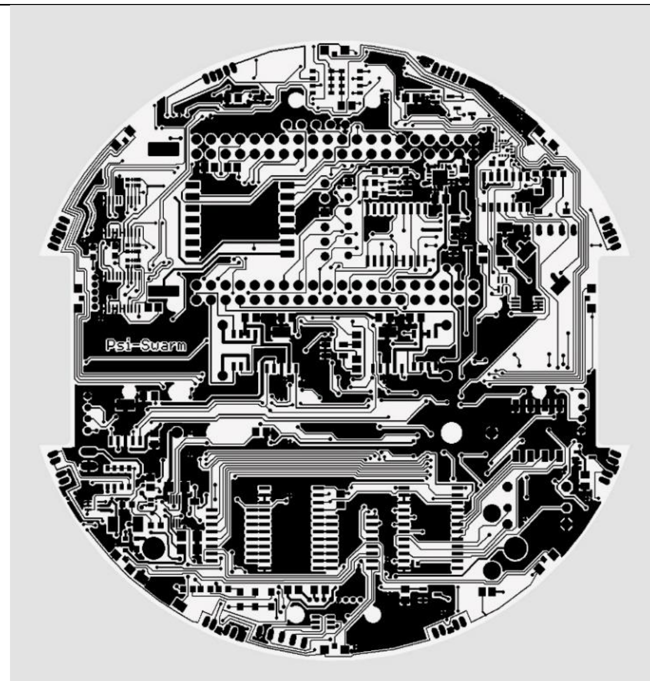
On the following pages, the image, copper layers and silk-screen for both sides of the PCB [as produced by EuroCircuits's automated verification tools] are shown. The PCB schematic layout diagram follows these.

PCB Views (Top Side)

PCB Image (TOP)



PCB Copper Layer (TOP)



PCB Silk Screen and Resist Outline (TOP)

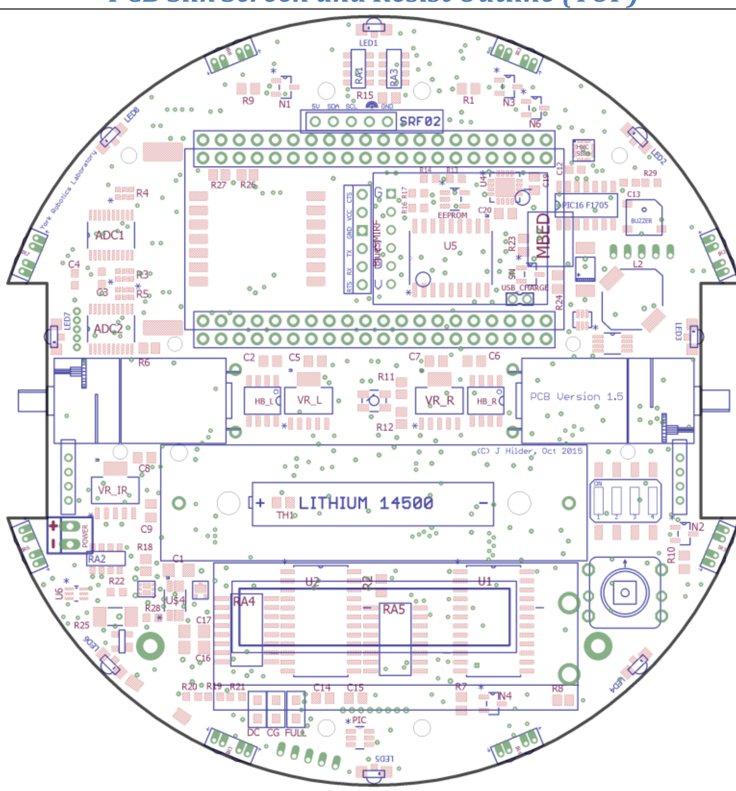
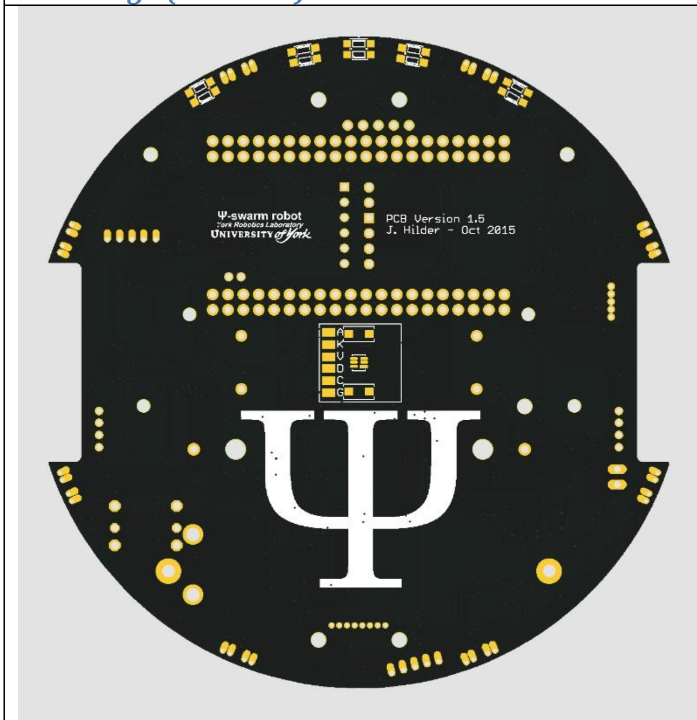


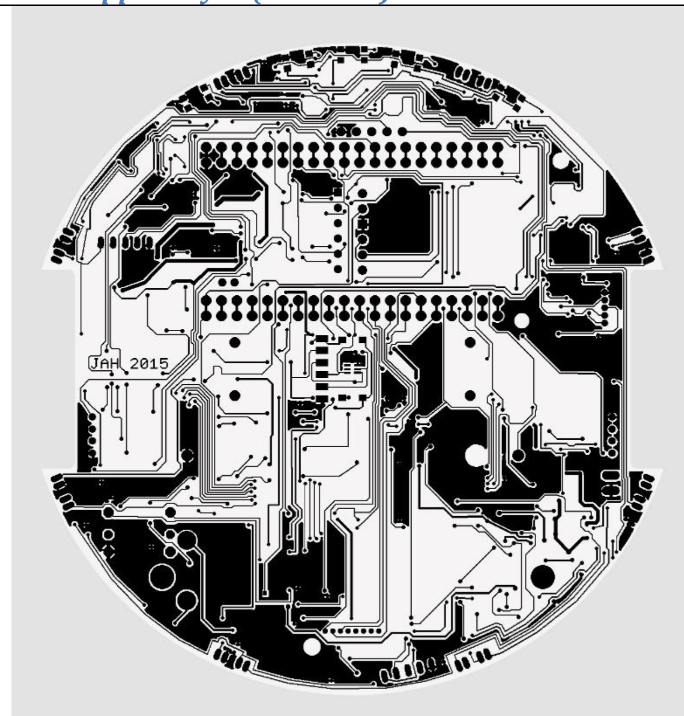
Figure 3: PCB Image for the top-side of the PsiSwarm PCB

PCB Views (Bottom Side)

PCB Image (BOTTOM)



PCB Copper Layer (BOTTOM)



PCB Silk Screen and Resist Outline (BOTTOM)

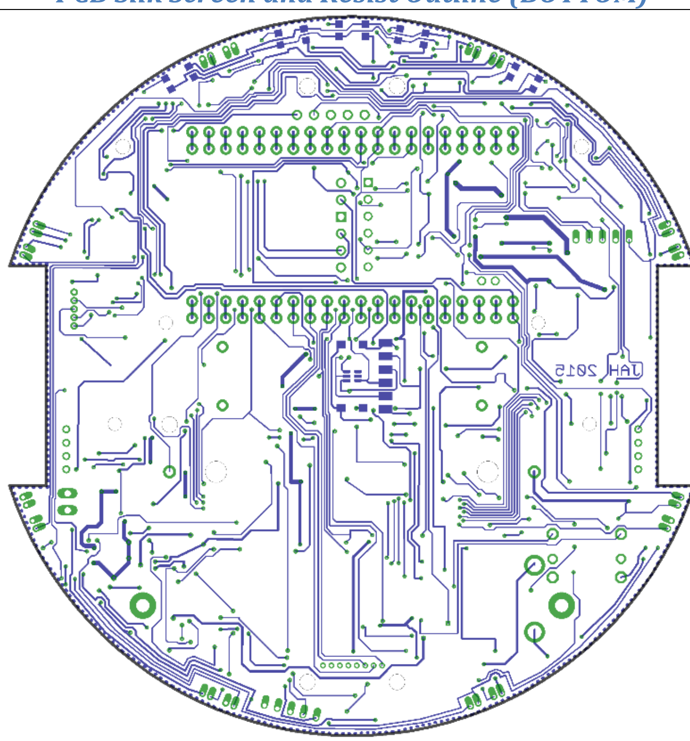


Figure 4: PCB Image for the bottom-side of the PsiSwarm PCB

Schematic for PCB Version 1.5

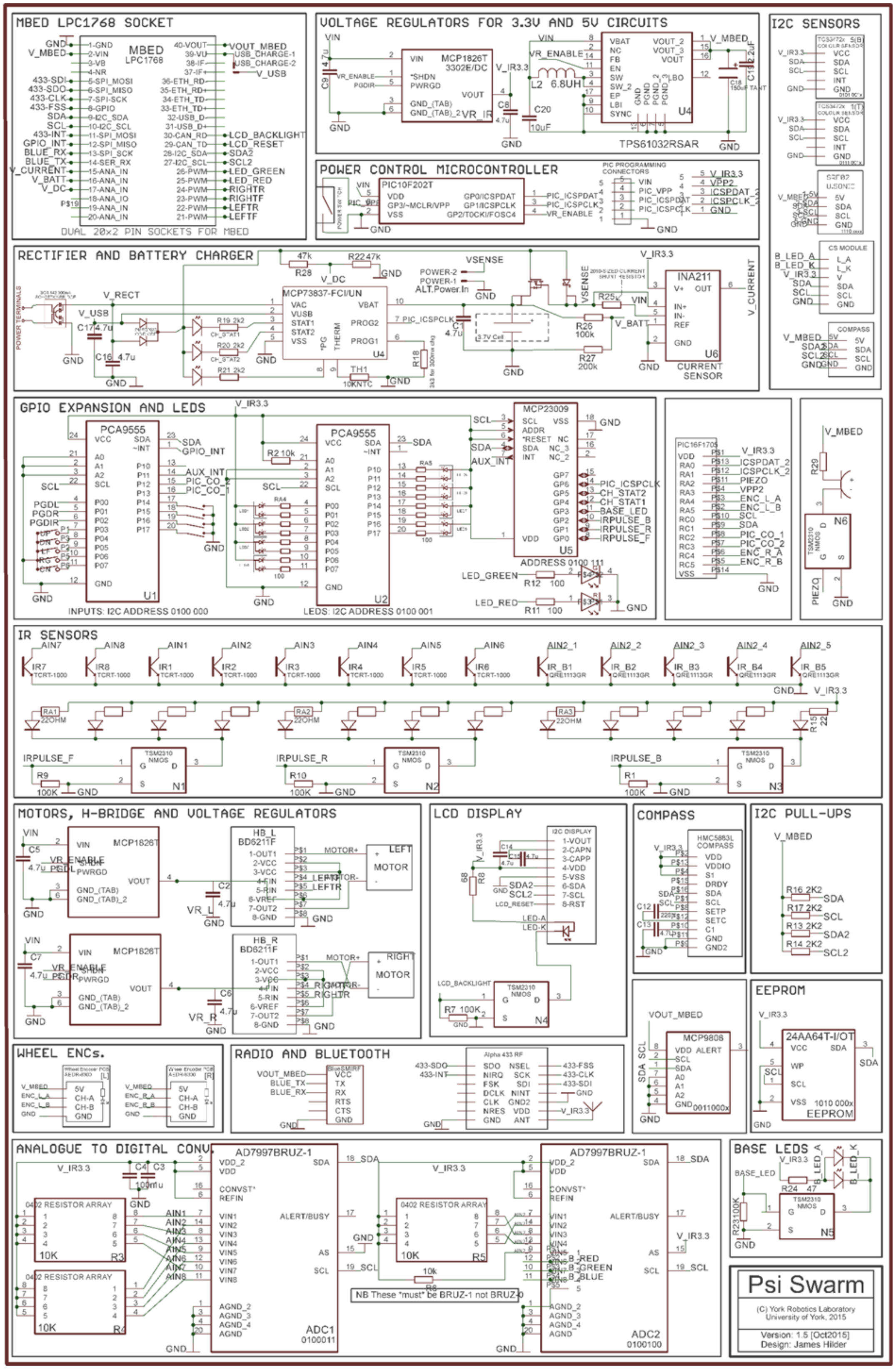


Figure 5: Schematic Diagram for the PsiSwarm PCB

Hardware

The board is equipped with the following sensors and actuators:

- A set of 8 bi-colour LEDs arranged in a ring around the edge of the board. They are equally spaced 45˚ apart from each other, with the exception of the East-West facing LEDs which are offset slightly North of center to allow light to shine through the ‘spokes’ of the wheels. Both colours of all these LEDs can be enabled/disabled individually. These LEDs are considered to be red-green in the remainder of this document, although different colour variants are available from the manufacturer (Kingbright KPBA-3010 series).
- An additional high-power red-green LED in the middle of the board, facing up. This has independent colour control from the edge LEDs, with the intensity of the red-green arrangement set using PWM outputs on the MBED.
- A set of 8 IR-proximity detectors arrange around the edge of the board. These are spaced almost equally around the board at approximately 45˚ intervals; the actual positions relative to North are: ±22.5˚, ±67˚, ±113˚ and ±157.5˚.
- A set of 5 IR-proximity detectors arranged on the underside of the front (North) of the robot, designed to be used as a line-following sensor.
- A set of 4-DIL switches, for the setting of robot ID within the swarm. This allows a standard swarm of up to 15 robots (the ID 0 is to be reserved). For larger swarms, the on-board EEPROM can be used to store additional ID bits.
- A 16x2 character MIDAS backlit LCD display connected to the secondary I^2C interface of the MBED. Different foreground\background colour combinations are available for this series of display; most the current robots are outfitted with black text-on-white displays.
- A 5-way directional switch which may be used to trigger interrupts and control the robot
- A 64 kilobit EEPROM (*24AA64T*)
- An (*optional)* MCP9808 I^2C based digital temperature sensor
- An (*optional*) 433MHz RF transceiver (*Alpha TRX-433*) and chip antenna connected to the SPI interface of the MBED
- An (*optional*) BlueSMIRF serial:BlueTooth transceiver. This is a stand-alone plug in module, available for approximately £25, which allows data to be sent and received using standard BlueTooth protocols whilst appearing to the MBED as a standard serial interface (ie allowing printf/scanf statements).
- An (*optional)* SRF-02 ultrasonic distance sensor facing forward at the front of the robot to detect obstacles in the 15cm – 2m range.
- An (*optional)* standalone tilt-compensated CMPS11 3D-compass module.

Note that some of the sensors and actuators are optional and may not be included on all the robots; some of the modules *[the Bluetooth module and compass]* are stand-alone modules which plug into sockets on the robot, some are standalone modules *[the 433MHz transceiver, the base colour sensor and the ultrasonic distance sensor]* which have to be soldered directly onto the PCB.

A number of different 3D-printed shims for both the underside and top of the robot have been developed. They have been designed to be made with the Objet range of printers by Stratasys but should be adaptable to a number of different printers. Alternative, a set of plastic shims which can be laser-cut from various widths of Perspex, and provide protection to the IR optocouplers from collisions with other robots and walls etc and protect the PCB from potential short-circuits, have also been designed. The underside shims include the provision for a pair of 8mm ball-bearings to improve stability of the robot in motion.

Basic Design

To augment the number of peripherals which can be attached to the MBED, the Psi Swarm PCB makes use of three separate I/O Expansion ICs. These devices connect to first the I^2C interface of the MBED (pin 9 and 10). Additionally, a pair of 8-channel ADC, which measure values from the IR sensor, and the EPROM, attach to this I^2C interface. A PIC microcontroller which is used for making audio tones and calculating wheel-encoder values is also attached to this interface. All these devices are compatible with high-speed 400KHz I^2C communications, which is enabled by default in the standard API.

The sockets for the MBED board includes an additional pair of 20 pin-sockets outside of the MBED pin-sockets to allow for debugging and expansion. As most of the GPIO pins on the MBED are actively used by the Psi Swarm board, the user must take care that any expansion does not conflict with existing circuitry; potential damage could occur to the robot or the MBED.

Infrared Proximity Sensors

The Psi Swarm robot contains a set of 8 transistor driven reflective optical sensors around the edge of the PCB. The optical component is a **TCRT1000** manufactured by *Vishay Semiconductor*, which combines a phototransistor and infrared emitter in a leaded package which blocks visible light. The phototransistors all feed into an *Analog Devices* **AD7997** 8-channel, 10-bit analogue to digital converter (ADC1), which is connected to the I^2C interface of the MBED (pin 9 and 10). The 7-bit address of the ADC is `0100011` *(0x46 in 8-bit hex format for MBED)* .

Each of the optocouplers also contains a 950nm infrared emitter. These are driven by a dedicated power supply stage, which draws directly from the battery supply on the Psi Swarm. The emitters are enabled in sets of 4 – those facing forwards form set 1, and those facing to the side and backwards form set 2. Set 1 is connected to signal line IRPULSE_F, which connects to pin GP0 of the MCP23009 GPIO expansion IC (U5), and set 2 to IRPULSE_R on pin GP1. A logic high value will enable the emitters. It is intended that the emitters are only enabled for short pulses last at most a few milliseconds at a time, as they draw significant current (approximately 50mA per emitter).

In addition, a second set of 5 similar Fairchild QRE1113GR infrared sensors are arranged on the underside of the PCB near the front of the robot. These are designed to allow the robot to detect and follow a black-on-white or white-on-black line, using PID (or similar) control algorithms. These sensors connect to ADC2 at address `0100100` *(0x48 in 8-bit hex format for MBED).* All five emitters are enabled by setting signal line IRPULSE_B high, by enabling GP2 on the MCP23009 expansion IC.

Software routines to enable and disable the emitters and take both background and reflective sensor readings for all the infrared sensors are included in the API, and generally it is recommended the end user uses these routines and doesn't interact with the ADCs or expansion ICs at a low-level. However, in the case where non-standard use of the IR is desired, such as using the IR for communication, this may be necessary.

Voltage, Current and Charge Sensors

The robot has a number of circuits and sensors that are designed to allow it to evaluate correct running operation. The battery voltage is monitored (though a potential-divider) into one of the analogue inputs on the board. The potential divider uses a 200KΩ:100KΩ ratio such that the voltage measured at the analogue input should equal 2/3rds of the current battery voltage. When an analogue input is measured on the MBED, it returns a floating point value between 0 and 1.0 which corresponds to a measured voltage of between 0V and 3.3V. Note that the nominal voltage of the Li-Po cell used is 3.7V: a fresh, fully-charged battery should measure 4.2V [*at minimal load*] and a discharged battery will be at below 3.6V. API routines exist to monitor the actual battery voltage; care should be taken to ensure the cells do not get into a deeply discharged state (below 3.5V) as this can damage the battery.

Similarly, any voltage present on either the USB input on the MBED or through the base-mounted recharging pickups (post-rectification) is also monitored [note that due to the charging circuit used, a small voltage is always present on this input which comes from the battery].

The main load passes through a current sense amplifier, which amplifies the voltage drop across a very small valued (2mOhm) resistor placed in series with the main load; this is amplified with a voltage gain of 500 so that the current being used by the load is equivalent to the voltage output of the amplifier in a 1:1 ratio. The peak current draw of the circuit, with both motors in stall state, and other sensors and actuators active, should never exceed 3A, except for very brief transients. The product of the battery voltage and current reading (or, when present, the DC-in voltage), should provide an accurate approximation of the system power draw.

Temperature and Colour Sensors

A Microchip **MCP9808** temperature sensor, placed close to the right-motor of the PsiSwarm, is connected to the primary I^2C bus using address `0011000` *(0x30 in 8-bit hex format for MBED)* . This provides 0.25˚C accuracy over a -40 to +125 ˚C range. API functions for reading the temperature in degrees Celsius are provided.

BlueTooth Transceiver

The primary for of inter-robot and robot-PC communication on the PsiSwarm robots is intended to be through the use of the BlueSMIRF Bluetooth transceiver, which connects to an RS-232 serial interface on the robot. When power is applied to the BlueSMIRF it automatically enters visibility mode, allowing a computer to connect to it [using the default passcode 1234]. Once connected, the computer should create a virtual Serial interface which can be communicated with using serial terminal software or programmatically using existing serial libraries for most programming languages. Other Bluetooth devices, such as Android and iOS phones\tablets, should also be able to communicate with the BlueSMIRF. Direct Inter-robot communication using the BlueSMIRF is technically possible by changing the operating mode of the BlueSMIRF (programmable using a special set of codes) but for most purposes using a computer effectively as a network switch, handling the passing of messages between connected BlueSMIRF modules, is probably an easier and more reliable approach.

RF Transceiver

An optional system for intra-swarm communication between Psi Swarm robots is using a 433MHz FM transceiver system, based on the low cost **Alpha-TRX433S** module by *RF Solutions*. The transceiver module connects to the MBED using the SPI interface and provides up to 115.2kbps data rate. The module and a dedicated chip antenna can be mounted on the PCB at the left-hand side of the MBED socket. Together they provide a typical range of several meters at the higher data rates and substantially further still (tested up to 20M) if data rates are lowered to 14400bps.

The transceiver module itself supports the sending of simple short-word packets. To enhance usability, the software API includes a simple communications stack which implements basic error-checking using CRC codes, and a simple instruction protocol, allowing various control messages to be sent between devices. The protocol includes the ability to broadcast messages (similar to UDP) to all listening robots, or to target individual device and form a reliable, acknowledged link (similar to concept TCP/IP, although much more basic in scope). The software stack is based on the implementation of the same RF solution in the Pi-Swarm robot.

ID Switch

A 4-position DIL switch is positioned just above the display window, which allows each robot in the swarm to be assigned its own unique ID. Whilst there are 16 different possible combinations, it is recommended that 0 is not used, as this is utilised by the serial stack (BlueTooth) and communications stack (433MHz RF) as the broadcast-to-all ID and as the ID for the radio modem. As a result it is possible by default to create swarms of up to 15 robots in size. In principle this could be expanded further by, for example, recording more significant bits in the on-board EEPROM, effectively allowing multiple sub-swarms of up to 15 robots each.

The ID switch is connected to pins `P1-4` to `P1-7` of the primary **PCA9555** I/O Expansion IC [U1], which communicates with the MBED using the I^2C interface. In the standard API these pins are not set to trigger an interrupt, instead they are read during the startup phase of the software and store the ID value in a local variable.

Direction Switch

To the right of the display window is a 5-direction push button switch. The direction switches are connected to pins `P0-3` to `P0-7` of the primary **PCA9555** I/O Expansion IC [U1]. In the standard API these pins are set to trigger an interrupt on GPIO_INT, which is connected to pin 12 to of the MBED. A small delay is associated to provide a simple software debounce of the switch and prevent multiple routines being triggered. An alternative solution to reading the switch would be to periodically poll the switch IO pins on the expansion IC or pin 12 on the MBED. Note that in practice it is generally recommended to avoid reliance on the center-push direction of the switch as it requires quite a lot of torque to activate the switch compared to the compass directions.

EEPROM

A 64-kilobit (8KB) EEPROM, the *Microchip Technology* **24AA64**, is connected to the I^2C interface of the MBED. It is accessed using the 7-bit address `1010000` *(0xA0 in 8-bit hex format for MBED)*. The EEPROM chip itself is sorted into 32-byte pages, and can be instructed to do either byte write or page write operations; it should be noted that a page write operation is restricted to a single physical page and that if it extends beyond the 32nd byte of a page the data will wrap over to overwrite the start of the page. An API routine abstracts this limitation and allows longer messages to be written regardless of where in a page they begin. It is important to remember that any write operation to EEPROM is relatively slow. The API follows the datasheet guidelines and adds a 5ms delay after any write operation (byte- or page-) which ensures that the data is correctly written; be aware that writing several hundreds of bytes will take several milliseconds. The last page of the EEPROM is reserved for system use storing the robots firmware settings (information about the robots hardware set and calibration data). The API abstracts this information away but it does mean that the entire 8KB area is not be available to the controller code.

Outer LEDs

The edge of the Pi Swarm PCB contains eight small, right-angled dual-colour LEDs, spaced at 45˚ intervals and numbered from 0-7[1] clockwise around the PsiSwarm starting from the North (0˚) position. Each colour LED has its own enable pin connected to a GPIO pin on the secondary **PCA9555** expansion IC [U2], that lets the outer LEDs be individually enabled or disabled. This arrangement allows each colour of each LED to be enabled or disabled as required. A number of different API routines exist for controlling these LEDs, allowing the state of all LEDs to be set at once, or the states of individual LEDs to be toggled without affecting the states of other LEDs.

By default the LEDs used are Kingbright KPBA-3010ESGC red\green, which offer are wide (140˚) viewing angle and 12mcd intensity; however any other KPBA-3010 series LED could be used as an alternative.

Central RGB LED

The central user LED is an upwards-facing, high intensity dual colour LED that sits dead in the center of the Psi Swarm PCB. This is directly connected to two PWM outputs on the MBED allowing for precise control of the output colour and brightness. By default the LED is a Wurth 150141RV73100 Red\Green LED with 120° viewing angle.

[1] Note that these are numbered 0-7 in the API but 1-8 on the PCB silk-screen

Motors

The primary actuators on the Psi Swarm are the pair of MFA/Como DC motors which drive the platform. These are sub-miniature 12mm diameter 3V-rated motors with integrated all-metal gear boxes. The standard gear ratio used on the Psi Swarms is 102:1, which provides a 144 RPM no-load speed and 135g.cm max efficiency torque at 3V supply, although other ratios are available from the manufacture [including 60:1 for 250 RPM, 79g.cm torque and 298:1 for 52 RPM, 352g.cm torque). At the most direct programming level, a floating point value ranging from -1.0 to 1.0 is sent to the 3-Pi microcontroller to set the relative motor output.

Other Actuators

Other actuators on the 3-Pi include the 7x2 LCD display, which can be written to using standard C `printf` commands, and two user LEDs on the bottom of the Psi Swarm designed to be enabled to assist with the colour sensor. One some models there is a buzzer which will convert a ASCII string into conventional notes; this feature is to be activated in future API releases.

List of Components

Below is the list of components for PCB revision 1.5, along with Farnell\CPC and RS part numbers.

Part	Layer	Qnt	Type	Description	Size	MPN	Farnell	RS	Manufacturer
ANT	SMD-Top	1	Chip Antenna	AT62 ANTENNA 433MHz	CUSTOM	0433AT62A0020E	2148530		JOHANSON TECHNOLOGY
BR1	SMD-Top	1	Schottky Diode	30V 900mA	SOT-143	BAS 3007A-RPP E6327	1791469	752-7824	INFINEON
BT	TH-Top	1	Battery Holder	AA\14500 1 Cell Battery Holder	AA	1028	1650669		KEYSTONE
C1,16,17	SMD-Top	3	Capacitor	4.7uF 20V Tantalum	C1206	T491A475K020AT	2283560		KEMET
C10	SMD-Top	1	Capacitor	10uF 10V X7R	C0805	C2012X7R1A106K125AC	2346934		TDK
C11	SMD-Top	1	Capacitor	2.2uF 16V X7R	C0805	GCM21BR71C225KA64L	2494198		MURATA
C12	SMD-Top	1	Capacitor	220nF 16V X7R	C0603	MCSH18B224K160CT	1856336		MULTICOMP
C13	SMD-Top	1	Capacitor	4.7uF 10V X5R	C0603	MC0603X475K100CT	2320818		MULTICOMP
C18	SMD-Top	1	Capacitor	150uF 10V Tantalum	C2917	TPSD157K010S0050	2283870		AVX
C2	SMD-Top	1	Capacitor	4.7uF 10V X7R	C0805	C2012X7R1A475K125AC	2346936		TDK
C3	SMD-Top	1	Capacitor	1uF 16V X7R	C0603	MCB0603R104KCT	9406140		MULTICOMP
C4	SMD-Top	1	Capacitor	100nF 16V X7R	C0603	0603YC104JAT2A	1740612		AVX
C5-9, 14, 15	SMD-Top	7	Capacitor	4.7uF 10V X7R	C0805	C2012X7R1A475K125AC	2346936		TDK
D1	SMD-Top	1	Schotkky Rectifier	20V 2A 420mV	SOT-1061	PMEG2020CPA	1859909		NXP
DISP	TH-Top		I2C 16x2 Display	MCCOG21605B6W-FPTLWI	Custom	MCCOG21605B6W-FPTLWI	2218942		MIDAS
IR1-8	TH-Top	8	Infrared Optical Sensor	TCRT-1000	TCRT-1000	TCRT1000	1470064		VISHAY
IR9-13	SMD-Bottom	5	Infrared Optical Sensor	QRE1113GR	QRE1113GR	QRE1113GR	1471020		FAIRCHILD SEMI
J1	TH-Top	2	Socket	2 x 20 PIN 0.1”	Extended				
J10	Pad	1	PIC Programming Port (Co)	5-PIN-CON	5-PIN-2MM				

J11	TH-Top	1	BlueSMIRF Module Socket	01-ROBOT_BLUESMIRF-HOBBYTRONICS	6x 0.1" SOCKET				
J12	Pad	1	Alpha4333 Module Pads	POLOLU_LIBRARY_ALPHA433	POLOLU_LIBRARY_ALPHA433				
J13	TH-Top	1	Power Terminals	POWER-TERMS	P-TERMINALS				
J14	TH-Top	1	Power In Socket	ALT.Power.In	22-23-2021	22-23-2021	1462926		
J15	Pad	1	Debug\Expansion Port	5-PIN-0.5PITCH	1.27MMPITCH5PIN				
J16	TH-Top	1	USB Charge Jumper	87758-0216	87758-0216	87758-0216			
J2	TH-Top	1	Compass Socket	CMPS11-TILT-COMP-COMPASS	6x 0.1" SOCKET				
J3	Pad	1	Colour Sensor Pads	COLOUR_SENSOR_CONNECTOR	COLOUR_SENSOR_MODULE				
J4	TH-Top	1	SRF02 Module Socket		5x 0.1" SOCKET				
J5	SMD-Bottom	1	Colour Sensor (Base)	TCS34725FN	6-Pin FN	TCS34725FN		785-7746	ams
J6	SMD-Top	1	Colour Sensor (Top)	TCS34721FN	6-Pin FN	TCS34721FN			ams
J7	TH-Top	1	Wheel Encoder Socket (Right)	WHEEL-ENCODER-RIGHT	WHEEL-ENCODER-RIGHT				
J8	TH-Top	1	Wheel Encoder Socket (Left)	WHEEL-ENCODER-LEFT	WHEEL-ENCODER				
J9	Pad	1	PIC Programming Port (Power)	5-PIN-CON	5-PIN-2MM				
L1	SMD-Top	1	Wirewound SMD Inductor	6.8uH 3.5A	IND_ELL8TP	ELL8TP6R8NB	1865671	749-8457	PANASONIC
LED1-8	SMD-Top	8	Bi-colour Side SMD LED	KPBA3010-DUAL-LED	2mm	KPBA-3010ESGC-F01	8530165		KINGBRIGHT
LED10	SMD-Bottom	2	SMD LED (White)		2012 LED		2335796*		KINGBRIGHT
LED12	SMD-Top	1	Indicator LED (Red:DC)	RED LED	2012 LED	KP-2012SECK-J3	2335811		KINGBRIGHT
LED13	SMD-Top	1	Indicator LED (Yellow:CHG)	YELLOW LED	2012 LED	KP-2012-SYCK-J3	2335796		KINGBRIGHT
LED14	SMD-Top	1	Indicator LED (Green:FULL)	GREEN LED	2012 LED	KP-2012CGCK	2290331		KINGBRIGHT
LED9	SMD-Top	1	Bi-colour Top SMD LED	2322113-WURTH-DUALLED	2322113-DUAL-LED-WURTH1411	150141RV73100	2322113		WURTH ELEKTRONIK
LS1	SMD-Top	1	SMT-0540-T-2-R-BUZZERSMT	SMT-0540-T-2-R-BUZZERSMT	SMT-0540-T-2-RBUZZER				
M1,2	TH-Top	2	Geared Motor	12mm, 3V, 102:1 Ratio	PCB-Pin Mount	951D1021	MC02078		MFA
Q1-6	SMD-Top	6	NMOS	4A 20V 0.024Ohm 800mV	SOT-23-3	TSM2314CX	1864589		TAIWAN SEMICONDUCTOR
R1,4,7,9,10,23	SMD-Top	6	Resistor	100 K 0805	R0805				
R11,12	SMD-Top	2	Resistor	100 R 0805	R0805				
R13,14,16,17,19-21,26	SMD-Top	8	Resistor	2K2 0603	R0603				
R15	SMD-Top	A	Resistor	22 R 0805	R0805				
R18	SMD-Top	A	Resistor	3K3 0805	R0805				

R2,6	SMD-Top	2	Resistor	10 K 0805	R0805			
R22,25	SMD-Top	2	Resistor	47 K 0603	R0603			
R24	SMD-Top	1	Resistor	47 R 0805	R0805			
R3	SMD-Top	1	Resistor	0.002 Ohm	R2010	TLR2H15DR002FTDG	2332256	TE
R5	SMD-Top	1	Resistor	200 K 0805	R0805			
R8	SMD-Top	1	Resistor	68 R 0805	R0805			
RN1-3	SMD-Top	3	Resistor Array	22OHM x 4	2012	CRA12E08322R0FTR	2352773	VISHAY
RN4-6	SMD-Top	3	Resistor Array	10K x 4	RESCAXE50P210X110X45-8N	EXB28V103JX	2060058	PANASONIC
RN7,8	SMD-Top	2	Resistor Array	100 OHM x 8	BOURNS-4816P SOIC	4816P-1-101LF	1902652	BOURNS
RT1	SMD-Top	1	Thermistor	10KNTC	R0805	NTCS0805E3103JMT	2103177	VISHAY
SW1	TH-Top	1	Cursor Switch	5 Way, 0.05A, 12VDC	ALPS-SKQUCAA010	SKQUCAA010	1435775	Alps
SW2	SMD-Top	1	ID Switch	4-WAY-DIP-SM	4WAYDIL-MCEMR-04-T	MCEMR-04-T	1524007	Multicomp
SW3	SMD-Top	1	Power Switch	24V 50mA 180gf	4.5mm SM	FSMSM	1703878	TE Connectivity
U1,2	SMD-Top	2	GPIO Expander	PCA9555	SOIC-24	PCA9555DWR	2101298	Texas Instruments
U10,11	SMD-Top	2	Hbridge Driver	7V 1A	SOP8	BD6211F-E2	1716259	ROHM
U12-14	SMD-Top	3	LDO Converter	01-ROBOT_MCP1826T-3302E/DC	SOT223-5	MCP1826T-3302E/DC	1578429	Microchip
U15	SMD-Top	1	PIC Microcontroller	PIC10F202T	SOT-6	PIC10F202T-I/OT	9942874	Microchip
U16	SMD-Top	1	PIC Microcontroller	PIC16F1705	SOIC-14	PIC16F1705-I/SL	2422884	Microchip
U17	SMD-Top	1	MOSFET-SCHOTTKY	P-Ch 3A 20V 0.08ohm	SOT1118	PMFPB8032XP	2191755	NXP
U18	SMD-Top	1	Temperature Sensor	POLOLU_LIBRARY_MCP9808-E/MS	MSOP-8	MCP9808-E/MS	2080523	Microchip
U19	SMD-Top	1	HMC5883L Compass	HMC5883L	HMC5883L			
U3	SMD-Top	1	Battery Charge IC	01-ROBOT_MCP73837-FCI/UN	MSOP-8	MCP73837-FCI/UN	1675426	Microchip
U4	SMD-Top	1	Boost Conventer	TPS61032RSAR	QFN-16	TPS61032RSAR	1743713	Texas Instruments
U5	SMD-Top	1	GPIO Expander	01-ROBOT_MCP23009-E/SO	SOIC-18	McP23009-E/SO	1699841	Microchip
U6	SMD-Top	1	Current Monitor	INA211	SC-70-6	INA214AIDCKT	1754262	Texas Instruments
U7,8	SMD-Top	2	ADC	AD7997BRUZ-1	TSSOP-20	AD7997BRUZ-1	1078305	Analog Devices
U9	SMD-Top	1	Eeprom	24AA64T-I/OT	SOT23-5	24AA64T-I/OT	2101260	Microchip

Using the Psi Swarm

The power switch for the robot is located to the bottom left of the display on the edge of the robot and is used to turn on and turn off the Psi Swarm. A dedicated, low-power PIC microcontroller is responsible for managing the power-state; in the current firmware, a press on the power-switch approximately 0.125 seconds in duration should toggle the power state when the switch is released. When the power-on state is activated, the PIC sets the VR_ENABLE line high, which enables the various on-board voltage regulators and switch-mode regulators, providing power to the MBED and other peripherals. The MBED will then begin its standard boot-strap routine, checking for any new .bin files, flashing the program code if necessary, then starting the main ARM microcontroller.

Charging

To recharge the robot one can either remove the 14500 battery and use an external charger to recharge it, or use the internal charging circuit on the robot to provide the charging current. The internal charging circuit is designed to run of a **6V** DC power supply (that should be capable of providing at very least 1 amp). The actual charging current provided by the MCP73837 charging IC is determined by the value of resistor R18 on the PCB; in the standard configuration this is set to 3.3KΩ which produces a 300mA charging current [*see the MCP73837 datasheet for more information on this resistor*]. When charged in an external charger, a charging current rate of ½C (eg 400mA for a 800mAH battery) is recommended for safety and maximum battery life.

If the battery is removed for charging, extreme care should be taken when reinserting it that the correct orientation of the battery is observed – the positive terminal must always be facing to the left of the PsiSwarm. The current PCB design does not feature reverse-polarity protection and the charging circuit will be damaged if the polarity is reversed.

Demo Mode

The demo-mode is accessed by holding the switch in any direction for 2-seconds when the robot is turned on (or the MBED is reset). If the switch is held in this way, the robot will run the special demo mode instead of the normal user code. Once the demo mode has started, various test menus for the different sensors and actuators can be accessed using the cursor to navigate the menus. One menu ("Code Demos") includes a number of simple built-in demo functions such as line following, obstacle avoiding, colour spin and "Stress Testing" which cycles through placing the robot in increasingly high-power states to ensure that the power delivery system is functional.

Firmware

The firmware is a small block of data stored at the end address of the non-volatile EPROM chip stored on the robot itself (so independent to the MBED). It is intended to store blocks of information about each robot such as its pcb version, serial number and available hardware and is read by the Psi Swarm MBED software library on boot-up. The main goal of the firmware is to ensure that MBED code written for one robot will work predictably on another, even if the PCB version or hardware setup of the robot is not identical. Naturally, this is not always possible, such as an algorithm relying on ultrasonic sensor data will not work on a robot that isn't equipped with the sensor. The firmware is written to the PsiSwarm using a special piece of MBED code that accesses the data block; the standard API prevents the user from accidentally writing code to this area to stop the firmware from being corrupted.

Software

The Psi Swarm robot has an MBED library that provides the API routines and internal functions to allow all of the main Psi Swarm sensors and actuators to be controlled. Two alternate versions of a API are currently available for use: version 0.7 and version 0.8 use the same core code base, however version 0.8 has been rewritten in C++ with the individual files all housing a Class. In version 0.7 the code base is primarily C with only the Display being implemented as a class. The core functions are the same between both implementations, however in the Class based API a function will need to be preceded by the class name.

For example, using API version 0.7, the following code will make the robot move and turn on an LED:

```
forward(0.25);

set_led(0,1);
```

Using API version 0.8, the following code would be needed for the same operation:

```
motors.forward(0.25);

led.set_led(0,1);
```

The motivation for this change is to make the software Doxygen compatible, more in line with the general structure of the MBED system, and more future proof. In future revisions further work will abstract away more code into the individual classes to make the code base more in line with object-oriented coding principles and reduce the reliance on global variables.

Core files in API version 0.7

This version of the library is written primarily in C. The exception with the LCD display and display.cpp which is implemented using a C++ class based structure. The user functions from the API itself are described in detail in the following section. The Psi Swarm Library comprises 14 core files plus respective header files which are outlined in the table below.

Filename	Description
`basic`	Contains the functions for the PsiBASIC interpreter (see section on PsiBASIC)
`colour`	Contains the functions for interacting with the bottom- (and top-) mounted colour sensors, if fitted
`dances`	Contains a library of simple predetermined movements
`demo`	Contains the functions for the built-in demo mode
`display`	Contains the Display class with functions for interfacing with the I2C display and its backlight
`eprom`	Contains the functions for writing-to and reading-from the dedicated EEPROM chip on the robot, including the firmware
`i2c`	Contains the internal functions for reading and writing messages to the peripherals connected to the primary I2C interface
`led`	Contains the functions to activate the various LEDs on the robot
`motors`	Contains the functions to activate the motors on the robot
`pic`	Contains the internal functions for communicating with the PIC cocontroller which handles audio
`psiswarm`	The main PsiSwarm C code
`sensors`	Contains the functions for reading information from the robots sensors
`serial`	Contains the functions that handle serial communication and message handling
`settings`	Contains the set of optional settings for the robot [header file only – no functions]

Core files in API version 0.8

This version of the library has been adapted to a class base structure in C++.

Filename	Description
`animations`	Contains the Animations class with functions for simple LED animations and dances *[renamed from dances.cpp and dances.h in version 0.7]*
`basic`	Contains the Basic class with functions for the PsiBASIC interpreter (see section on PsiBASIC)
`colour`	Contains the Colour class functions for interacting with the bottom- (and top-) mounted colour sensors, if fitted
`demo`	Contains the Demo class with functions for the built-in demo mode
`display`	Contains the Display class with functions for interfacing with the I2C display and its backlight
`eprom`	Contains the Eprom class with functions for writing-to and reading-from the dedicated EEPROM chip on the robot, including the firmware
`i2c_setup`	Contains the Setup class, with internal functions for reading and writing messages to the peripherals connected to the primary I2C interface
`led`	Contains the Led class with functions to activate the various LEDs on the robot
`motors`	Contains the Motors class with functions to activate the motors on the robot
`psiswarm`	The main Psiswarm class and initialisation, uptime, reset and debug functions
`sensors`	Contains the Sensors class with functions for reading information from the robots infrared, ultrasonic, temperature and power sensors
`serial`	Contains the SerialControl class with functions that handle serial communication and message handling
`settings`	Contains the set of optional settings for the robot *[header file only – no functions]*
`sound`	Contains the Sound class with functions for communicating with the PIC cocontroller which handles audio *[renamed from pic.cpp and pic.h in version 0.7]*

Communications

The code for the Alpha433 radio transceiver is not included in the current releases (0.7 & 0.8) of the PsiSwarm API; the code and documentation is available in the Pi Swarm manual and library (Alpha433.C, Alpha433.H and Communications.C, Communications.H).

PsiSwarm API

The API for the PsiSwarm is documented using the Doxygen 2.0 system, which produces online output for use within the MBED system, alongside the ability to export to LaTeX and HTML. The LaTeX output has been appended to this PDF file.

Animations Class Reference

`#include <animations.h>`

Public Member Functions

- void **vibrate** (void)
- void **led_run1** (void)
- void **set_colour** (char colour)

Detailed Description

The **Animations** class contains simple predefined LED animations and dances

Definition at line 32 of file animations.h.

Member Function Documentation

void Animations::led_run1 (void)

Patterns LEDs from back to front of robot 3 times then blinks at the front; animation takes about 1 second; restores LED states after action

Definition at line 36 of file animations.cpp.

void Animations::set_colour (char colour)

Sets the colour for single-colour LED animations (default = 1)

Parameters:

colour	The colour LED to use in the animation (1 = red, 2 = green, 3 = orange)

Definition at line 31 of file animations.cpp.

void Animations::vibrate (void)

Make the robot vibrate (turn rapidly left & right) for approximately 1 second with LED flashes; restores LED states after action

Definition at line 72 of file animations.cpp.

Basic Class Reference

```
#include <basic.h>
```

Public Member Functions

- void **read_list_of_file_names** (void)

Detailed Description

The **Basic** class contains the functions for the Psi **Basic** interpreter and file-handling
Definition at line 29 of file basic.h.

Member Function Documentation

void Basic::read_list_of_file_names (void)

Read the list of Psi **Basic** filenames from the MBED Flash memory

Definition at line 26 of file basic.cpp.

Colour Class Reference

```
#include <colour.h>
```

Public Member Functions

- void **set_base_colour_sensor_gain** (char gain)
- void **set_base_colour_sensor_integration_time** (char int_time)
- void **enable_base_colour_sensor** (void)
- void **read_base_colour_sensor_values** (int *store_array)
- char **IF_check_base_colour_sensor** (void)

Detailed Description

The **Colour** class contains the functions for reading the base-mounted and top-mounted I2C colour sensors (optional).

Definition at line 30 of file colour.h.

Member Function Documentation

void Colour::enable_base_colour_sensor (void)

Enable the base colour sensor

Definition at line 40 of file colour.cpp.

*void Colour::read_base_colour_sensor_values (int * store_array)*

Read the values from the base colour sensor

Parameters:

Pointer	to 3 x int array for r-g-b values

Definition at line 28 of file colour.cpp.

void Colour::set_base_colour_sensor_gain (char gain)

Set the gain of the base colour sensor

Parameters:

gain	The gain value for the sensor

Definition at line 32 of file colour.cpp.

void Colour::set_base_colour_sensor_integration_time (char int_time)

Set the integration time constant for the base colour sensor

Parameters:

gain	The gain value for the sensor

Definition at line 36 of file colour.cpp.

Demo Class Reference

`#include <demo.h>`

Public Member Functions

- void **start_demo_mode** (void)
- void **demo_handle_switch_event** (char switch_pressed)

Detailed Description

Demo class Build in demonstration

Test:

> mode for the robot that is enabled by holding the cursor switch in a direction for 1 second at boot-up time. The demonstration includes the ability to get readings from most of the on-board sensors, enable LEDs and motors, and run a number of basic build in demonstrations and tests, using cursor-navigated menus.

The demo can also be enabled by calling the **start_demo_mode()** function.

Definition at line 36 of file demo.h.

Member Function Documentation

void Demo::start_demo_mode (void)

Start the demonstration mode

Definition at line 80 of file demo.cpp.

Display Class Reference

```
#include <display.h>
```

Public Member Functions

- **Display** ()
- **Display** (PinName sda, PinName scl, PinName reset, PinName backlight)
- void **clear_display** (void)
- void **home** (void)
- void **write_string** (char *message)
- void **write_string** (char *message, char length)
- void **set_position** (char row, char column)
- void **set_cursor** (char enable)
- void **set_blink** (char enable)
- void **set_display** (char enable)
- void **set_backlight_brightness** (float brightness)
- void **debug_page** (char *message, char length)
- void **IF_restore_page** (void)
- void **IF_debug_multipage** (void)
- void **IF_backlight_toggle** (void)
- void **post_init** (void)
- void **post_post_init** (void)
- int **i2c_message** (char byte)
- void **init_display** (char mode)
- int **disp_putc** (int c)

Detailed Description

Display class Functions for use with the Midas 16x2 I2C LCD **Display** (MCCOG21605x6W) LCD Farnell part 2218942 or 2063206

Example:

```
#include "psiswarm.h"

int main() {
    init();
    display.clear_display;       //Clears display
    display.set_position(0,2);   //Set cursor to row 0 column 2
    display.write_string("YORK ROBOTICS");
    display.set_position(1,3);   //Set cursor to row 1 column 3
    display.write_string("LABORATORY");
}
```

Definition at line 53 of file display.h.

Constructor & Destructor Documentation

Display::Display ()

Create the LCD **Display** object connected to the default pins (sda = p28, scl = p27, reset = p29, backlight = p30)

Definition at line 47 of file display.cpp.

Display::Display (PinName **sda**, *PinName* **scl**, *PinName* **reset**, *PinName* **backlight***)*

Create the LCD **Display** object connected to specific pins

Parameters:

sda	pin - default is p28
scl	pin - default is p27
reset	pin - default is p29
backlight	pin - default is p30

Definition at line 44 of file display.cpp.

Member Function Documentation

void Display::clear_display (void)

Clear the display

Definition at line 230 of file display.cpp.

void Display::home (void)

Set cursor to home position

Definition at line 238 of file display.cpp.

void Display::set_backlight_brightness (float brightness)

Set the brightness of the backlight

Parameters:

brightness	- Sets the brightness of the display (range 0.0 to 1.0)

Definition at line 198 of file display.cpp.

void Display::set_blink (char enable)

Enable or disable cursor blink

Parameters:

enable	- Set to 1 to enable the cursor blinking mode

Definition at line 188 of file display.cpp.

void Display::set_cursor (char enable)

Enable or disable cursor

Parameters:

enable	- Set to 1 to enable the cursor visibility

Definition at line 183 of file display.cpp.

void Display::set_display (char enable)

Enable or disable display

Parameters:

enable	- Set to 1 to enable the display output

Definition at line 193 of file display.cpp.

void Display::set_position (char **row***, char* **column***)*

Set the row and column of cursor position

Parameters:

row	- The row of the display to set the cursor to (either 0 or 1)
column	- The column of the display to set the cursor to (range 0 to 15)

Definition at line 174 of file display.cpp.

*void Display::write_string (char ** **message***)*

Print string message

Parameters:

message	- The null-terminated message to print

Definition at line 131 of file display.cpp.

*void Display::write_string (char ** **message***, char* **length***)*

Print string message of given length

Parameters:

message	- The message to print
length	- The number of characters to display

Definition at line 154 of file display.cpp.

Eprom Class Reference

```
#include <eprom.h>
```

Public Member Functions

- void **write_eeprom_byte** (int address, char data)
- char **read_eeprom_byte** (int address)
- char **read_next_eeprom_byte** (void)
- char **read_firmware** (void)

Detailed Description

Eprom Class Functions for accessing the 64Kb EPROM chip and reading the reserved firmware block

Example:

```
#include "psiswarm.h"

int main() {
    init();
    eprom.write_eeprom_byte(0,0xDD);    //Writes byte 0xDD in EPROM address 0
    char c = eprom.read_eeprom_byte(0); //c will hold 0xDD
    //Valid address range is from 0 to 65279
}
```

Definition at line 41 of file eprom.h.

Member Function Documentation

char Eprom::read_eeprom_byte (int address)

Read a single byte from the EPROM

Parameters:

address	The address to read from, range 0-65279

Returns:

The character stored at address

Definition at line 62 of file eprom.cpp.

char Eprom::read_firmware (void)

Read the data stored in the reserved firmware area of the EPROM

Returns:

1 if a valid firmware is read, 0 otherwise

Definition at line 88 of file eprom.cpp.

char Eprom::read_next_eeprom_byte (void)

Read the next byte from the EPROM, to be called after read_eeprom_byte

Returns:

The character stored at address after the previous one read from

Definition at line 77 of file eprom.cpp.

void Eprom::write_eeprom_byte (int address, char data)

Write a single byte to the EPROM

Parameters:

address	The address to store the data, range 0-65279
data	The character to store

Definition at line 42 of file eprom.cpp.

Led Class Reference

```
#include <led.h>
```

Public Member Functions

- void **set_leds** (char green, char red)
- void **set_green_leds** (char green)
- void **set_red_leds** (char red)
- void **set_led** (char led, char state)
- void **set_base_led** (char state)
- void **blink_leds** (float timeout)
- void **set_center_led** (char state)
- void **set_center_led** (char state, float brightness)
- void **set_center_led_brightness** (float brightness)
- unsigned short **get_led_states** (void)
- void **save_led_states** (void)
- void **restore_led_states** (void)
- void **IF_init_leds** (void)
- void **IF_update_leds** (void)

Detailed Description

Led class Functions to control the various LEDs on the robot

Example:

```
#include "psiswarm.h"

int main() {
    init();
    led.set_led(0,1);       //Set the outer LED number 0 (North) to red
    led.set_led(4,3);       //Set the outer LED number 4 (South) to orange (red+green)
}
```

Definition at line 42 of file led.h.

Member Function Documentation

void Led::blink_leds (float timeout)

Turns on all outer LEDs for a period of time defined by timeout then restore their previous state

Parameters:

timeout	- The time (in seconds) to keep LEDs on

Definition at line 71 of file led.cpp.

unsigned short Led::get_led_states (void)

Returns the current state of the outer LEDs

Returns:

A 16-bit value when MSB represent the green states and LSB the red states of the 8 LEDs

Definition at line 33 of file led.cpp.

void Led::restore_led_states (void)

Restore the LED states to those set usign store_led_states()

Definition at line 118 of file led.cpp.

void Led::save_led_states (void)

Store the current LED states for use with **restore_led_states()**

Definition at line 112 of file led.cpp.

void Led::set_base_led (char state)

Set the state of the base LEDs [if fitted]

Parameters:

state	- 0 for off, 1 for on

Definition at line 45 of file led.cpp.

void Led::set_center_led (char state)

Set the state the center LED

Parameters:

state	- 0 for off, 1 for red, 2 for green, 3 for orange

Definition at line 78 of file led.cpp.

void Led::set_center_led (char state, float brightness)

Set the state the center LED with brightness control

Parameters:

state	- 0 for off, 1 for red, 2 for green, 3 for orange
brightness	- brightness of LED [PWM duty cycle] - range 0.0 to 1.0

Definition at line 83 of file led.cpp.

void Led::set_center_led_brightness (float brightness)

Set the brightness of center LED without changing state

Parameters:

brightness	- brightness of LED [PWM duty cycle] - range 0.0 to 1.0

Definition at line 107 of file led.cpp.

void Led::set_green_leds (char green)

Set the green component of all 8 outer LEDs to the defined colour sequence

Parameters:

green	- An 8-bit description of the green leds eg(0b00000001) means that LED 7 green is on, rest are off

Definition at line 50 of file led.cpp.

void Led::set_led (char led, char state)

Set the state of an invididual outer LED without affecting other LEDs

Parameters:

led	- The LED to change state of (range 0 to 7)
state	- 0 for off, 1 for red, 2 for green, 3 for orange

Definition at line 62 of file led.cpp.

void Led::set_leds (char **green***, char* **red***)*

Set all 8 outer LEDs to the defined colour sequence

Parameters:

green	- An 8-bit description of the green leds eg(0b00000001) means that LED 7 green is on, rest are off
red	- An 8-bit description of the red leds eg(0b11111110) means that LED 7 red is off, rest are on

Definition at line 38 of file led.cpp.

void Led::set_red_leds (char **red***)*

Set the red component of all 8 outer LEDs to the defined colour sequence

Parameters:

red	- An 8-bit description of the red leds eg(0b11111110) means that LED 7 red is off, rest are on

Definition at line 56 of file led.cpp.

Motors Class Reference

`#include <motors.h>`

Public Member Functions

- void **set_left_motor_speed** (float speed)
- void **set_right_motor_speed** (float speed)
- void **brake_left_motor** (void)
- void **brake_right_motor** (void)
- void **brake** (void)
- void **stop** (void)
- void **forward** (float speed)
- void **backward** (float speed)
- void **turn** (float speed)
- void **init_motors** (void)
- void **time_based_forward** (float speed, int microseconds, char **brake**)
- void **time_based_turn** (float speed, int microseconds, char **brake**)
- int **time_based_turn_degrees** (float speed, float degrees, char **brake**)
- float **get_maximum_turn_angle** (int microseconds)
- int **get_time_based_turn_time** (float speed, float degrees)

Detailed Description

Motors class Functions to control the Psi Swarm robot motors

Example:

```
#include "psiswarm.h"

int main() {
    init();
    motors.forward(0.5);      //Set the motors to forward at speed 0.5
    wait(0.5);
    motors.brake();           //Enable the hardware brake
    wait(0.5);
    motors.turn(0.5);         //Turn clockwise at 50% speed
    wait(0.5);
    motors.stop();            //Sets motor speed to zero (but not hardware brake)
}
```

Definition at line 46 of file motors.h.

Member Function Documentation

void Motors::backward (float speed)

Sets both motors to the specified inverted speed

Parameters:

speed	- Set the motors to the specified speed (range -1.0 for max. forward to 1.0 for max. reverse)

Definition at line 83 of file motors.cpp.

void Motors::brake (void)

Enable the active brake on the both motors

Definition at line 56 of file motors.cpp.

void Motors::brake_left_motor (void)

Enable the active brake on the left motor

Definition at line 42 of file motors.cpp.

void Motors::brake_right_motor (void)

Enable the active brake on the right motor

Definition at line 49 of file motors.cpp.

void Motors::forward (float speed)

Sets both motors to the specified speed

Parameters:

speed	- Set the motors to the specified speed (range -1.0 for max. reverse to 1.0 for max. forward)

Definition at line 74 of file motors.cpp.

void Motors::init_motors (void)

Initialise the PWM settings for the motors

Definition at line 299 of file motors.cpp.

void Motors::set_left_motor_speed (float speed)

Set the left motor to the specified speed

Parameters:

speed	- The set motor to the specified (range -1.0 for max. reverse to 1.0 for max. forward)

Definition at line 28 of file motors.cpp.

void Motors::set_right_motor_speed (float speed)

Set the left motor to the specified speed

Parameters:

speed	- The set motor to the specified (range -1.0 for max. reverse to 1.0 for max. forward)

Definition at line 35 of file motors.cpp.

void Motors::stop (void)

Stop both motors This sets the speed of both motors to 0; it does not enable the active brake

Definition at line 65 of file motors.cpp.

void Motors::time_based_forward (float speed, int microseconds, char brake)

Make the robot move forward for a predetermined amount of time

Parameters:

speed	- Sets the motors to the specified speed (range -1.0 for max. forward to 1.0 for max. reverse)
microseconds	- The duration to keep moving
brake	- If set to 1, the brake instruction will be applied at the end of the move, else motors are just set to stop

Definition at line 103 of file motors.cpp.

void Motors::time_based_turn (float **speed***, int* **microseconds***, char* **brake***)*

Make the robot turn for a predetermined amount of time

Parameters:

speed	- Sets the turning speed (range -1.0 for max. counter-clockwise to 1.0 for max. clockwise)
microseconds	- The duration to keep moving
brake	- If set to 1, the brake instruction will be applied at the end of the move, else motors are just set to stop

Definition at line 115 of file motors.cpp.

void Motors::turn (float **speed***)*

Turn the robot on the spot by setting motors to equal and opposite speeds

Parameters:

speed	- Sets the turning speed (range -1.0 for max. counter-clockwise to 1.0 for max. clockwise)

Definition at line 92 of file motors.cpp.

Psiswarm Class Reference

```
#include <psiswarm.h>
```

Public Member Functions

- void **init** (void)
- float **get_uptime** (void)
- void **pause_user_code** (float period)
- void **reset_encoders** (void)
- void **debug** (const char *format,...)

Detailed Description

Psiswarm Class The main class to define a robot

Example code for main.cpp:

```
#include "psiswarm.h"
Psiswarm psi;
char * program_name = "Example";
char * author_name  = "Name";
char * version_name = "0.8";
void handle_switch_event(char switch_state){}
void handle_user_serial_message(char * message, char length, char interface) {}
int main(){
   psi.init();
   while(1) { //Do something!
   }
}
```

Definition at line 98 of file psiswarm.h.

Member Function Documentation

*void Psiswarm::debug (const char * format, ...)*

Send a string (in printf format) to the preferred debug stream, specified in **settings.h** [of overridden programmatically]

Parameters:

The	string to send to output stream

Definition at line 325 of file psiswarm.cpp.

float Psiswarm::get_uptime (void)

Get the uptime for the robot

Returns:

The amount of time in seconds that the MBED has been active since last reset

Definition at line 339 of file psiswarm.cpp.

void Psiswarm::init (void)

Main initialisation routine for the PsiSwarm robot

Set up the GPIO expansion ICs, launch demo mode if button is held

init()

Main initialisation routine for the PsiSwarm robot

Set up the GPIO expansion ICs, launch demo mode if button is held

Definition at line 169 of file psiswarm.cpp.

void Psiswarm::pause_user_code (float period)

Pause the user code for a defined amount of time

Parameters:

The	amount of time in seconds to pause user code

Definition at line 344 of file psiswarm.cpp.

void Psiswarm::reset_encoders (void)

Reset the wheel encoder counters

Definition at line 319 of file psiswarm.cpp.

Sensors Class Reference

```
#include <sensors.h>
```

Public Member Functions

- float **get_battery_voltage** (void)
- float **get_current** (void)
- float **get_dc_voltage** (void)
- float **get_temperature** (void)
- void **enable_ultrasonic_ticker** (void)
- void **disable_ultrasonic_ticker** (void)
- void **update_ultrasonic_measure** (void)
- void **IF_read_ultrasonic_measure** (void)
- void **store_background_raw_ir_values** (void)
- void **store_illuminated_raw_ir_values** (void)
- void **store_ir_values** (void)
- unsigned short **get_background_raw_ir_value** (char index)
- unsigned short **get_illuminated_raw_ir_value** (char index)
- unsigned short **calculate_side_ir_value** (char index)
- unsigned short **read_illuminated_raw_ir_value** (char index)
- void **store_background_base_ir_values** (void)
- void **store_illuminated_base_ir_values** (void)
- void **store_base_ir_values** (void)
- unsigned short **get_background_base_ir_value** (char index)
- unsigned short **get_illuminated_base_ir_value** (char index)
- unsigned short **calculate_base_ir_value** (char index)
- void **store_reflected_ir_distances** (void)
- float **read_reflected_ir_distance** (char index)
- float **get_reflected_ir_distance** (char index)
- float **calculate_reflected_distance** (unsigned short background_value, unsigned short illuminated_value)
- int **get_bearing_from_ir_array** (unsigned short *ir_sensor_readings)
- void **store_line_position** (void)
- void **calibrate_base_ir_sensors** (void)

Detailed Description

Sensors class Functions to read values from the Psi Swarm infrared, ultrasonic, temperature and power sensors

Example:

```
#include "psiswarm.h"

int main() {
    init();

}
```

Definition at line 39 of file sensors.h.

Member Function Documentation

unsigned short Sensors::calculate_base_ir_value (char index)

Returns the subtraction of the background base IR value from the reflection based on last call of **store_base_ir_values()**
For most purposes this is the best method of getting values from the base IR sensor as it mitigates for background levels of IR

Parameters:

index	- The index of the sensor to read (range 0 to 4, sensor from left to right viewed from above - 2 is in middle of front)

Returns:

Unsigned short of compensated ir value (illuminated value - background value) (range 0 to 4095)

Definition at line 407 of file sensors.cpp.

unsigned short Sensors::calculate_side_ir_value (char index)

Returns the subtraction of the background side IR value from the reflection based on last call of **store_ir_values()** For most purposes this is the best method of detected obstacles etc as it mitigates for varying background levels of IR

Parameters:

index	- The index of the sensor to read (range 0 to 7, clockwise around robot from front-right)

Returns:

Unsigned short of compensated ir value (illuminated value - background value) (range 0 to 4095)

Definition at line 420 of file sensors.cpp.

void Sensors::disable_ultrasonic_ticker (void)

Disables the ultrasonic ticker

Definition at line 37 of file sensors.cpp.

void Sensors::enable_ultrasonic_ticker (void)

Enables a 10Hz ticker that automatically takes readings from the SRF-02 ultrasonic sensor (if fitted)

Definition at line 32 of file sensors.cpp.

unsigned short Sensors::get_background_base_ir_value (char index)

Returns the stored value of the non-illuminated base IR sensor value based on last call of store_background_base_ir_values Call either **store_base_ir_values()** or **store_background_base_ir_values()** before using this function to update reading

Parameters:

index	- The index of the sensor to read (range 0 to 4, sensor from left to right viewed from above - 2 is in middle of front)

Returns:

Unsigned short of background IR reading (range 0 to 4095)

Definition at line 299 of file sensors.cpp.

unsigned short Sensors::get_background_raw_ir_value (char index)

Returns the stored value of the non-illuminated side-facing IR sensor value based on last call of store_background_raw_ir_values Call either **store_ir_values()** or **store_background_raw_ir_values()** before using this function to update reading

Parameters:

index	- The index of the sensor to read (range 0 to 7, clockwise around robot from front-right)

Returns:

Unsigned short of background IR reading (range 0 to 4095)

Definition at line 175 of file sensors.cpp.

float Sensors::get_battery_voltage (void)

Returns the current battery voltage for the robot

Returns:

The voltage (in V); this should range between 3.5V for a discharged battery and 4.2V for a fully charged battery

Definition at line 87 of file sensors.cpp.

float Sensors::get_current (void)

Returns the current being used by the robot

Returns:

The current (in A)

Definition at line 93 of file sensors.cpp.

float Sensors::get_dc_voltage (void)

Returns the voltage sensed from the DC input (post rectification)

Returns:

The voltage (in V); note some back-voltage from the battery is expected even when no DC input is detected

Definition at line 101 of file sensors.cpp.

unsigned short Sensors::get_illuminated_base_ir_value (char index)

Returns the stored value of the illuminated base IR sensor value based on last call of store_illuminated_base_ir_values Call either **store_base_ir_values()** or **store_illuminated_base_ir_values()** before using this function to update reading

Parameters:

index	- The index of the sensor to read (range 0 to 4, sensor from left to right viewed from above - 2 is in middle of front)

Returns:

Unsigned short of illuminated IR reading (range 0 to 4095)

Definition at line 307 of file sensors.cpp.

unsigned short Sensors::get_illuminated_raw_ir_value (char index)

Returns the stored value of the illuminated side-facing IR sensor value based on last call of store_illuminated_raw_ir_values Call either **store_ir_values()** or **store_illuminated_raw_ir_values()** before using this function to update reading

Parameters:

index	- The index of the sensor to read (range 0 to 7, clockwise around robot from front-right)

Returns:

Unsigned short of illuminated IR reading (range 0 to 4095)

Definition at line 183 of file sensors.cpp.

float Sensors::get_temperature (void)

Returns the temperature sensed by the digital thermometer placed near the front of the MBED socket

Returns:

The temperature (in degrees C)

Definition at line 77 of file sensors.cpp.

unsigned short Sensors::read_illuminated_raw_ir_value (char index)

Enables the IR emitter then returns the value of the given side-facing IR sensor This function is used when just one sensor is needed to be read; in general using **store_ir_values()** and get_illuminated_raw_ir_value(index) is preferable

Parameters:

index	- The index of the sensor to read (range 0 to 7, clockwise around robot from front-right)

Returns:

Unsigned short of illuminated IR reading (range 0 to 4095)

Definition at line 256 of file sensors.cpp.

void Sensors::store_background_base_ir_values (void)

Stores the raw ADC values for all 5 base IR sensors without enabling IR emitters

Definition at line 325 of file sensors.cpp.

void Sensors::store_background_raw_ir_values (void)

Stores the raw ADC values for all 8 IR side-facing sensors without enabling IR emitters

Definition at line 207 of file sensors.cpp.

void Sensors::store_base_ir_values (void)

Stores the raw ADC values for all 5 base IR sensors both before and after enabling IR emitters Calls **store_background_base_ir_values()** then **store_illuminated_base_ir_values()**

Definition at line 315 of file sensors.cpp.

void Sensors::store_illuminated_base_ir_values (void)

Stores the raw ADC values for all 5 base IR sensors after enabling IR emitters

Definition at line 334 of file sensors.cpp.

void Sensors::store_illuminated_raw_ir_values (void)

Stores the raw ADC values for all 8 IR side-facing sensors after enabling IR emitters

Definition at line 216 of file sensors.cpp.

void Sensors::store_ir_values (void)

Stores the raw ADC values for all 8 IR side-facing sensors both before and after enabling IR emitters Calls **store_background_raw_ir_values()** then **store_illuminated_raw_ir_values()**

Definition at line 200 of file sensors.cpp.

void Sensors::update_ultrasonic_measure (void)

Sends a message to SRF-02 ultrasonic sensor (if fitted) to instruct it to take a new reading. The result is available approx 70ms later

Definition at line 42 of file sensors.cpp.

SerialControl Class Reference

```
#include <serial.h>
```

Public Member Functions

- void **setup_serial_interfaces** (void)

Detailed Description

SerialControl class Functions to handle command and user messages sent over the PC or BT serial interfaces. Most of the functions within this class are not intended to be called by user applications; once the **setup_serial_interfaces()** function has been called the enabled serial ports are attached to listeners which handle any received messages. A predefined message structure for commands has been created which allows most functions on the robot to be externally called, either using a PC-robot or Bluetooth connection.

For user functions, the main.cpp file should include a void handle_user_serial_message(char * message, char length, char interface) function to handle user-defined messages.

Definition at line 38 of file serial.h.

Member Function Documentation

void SerialControl::setup_serial_interfaces (void)

Sets the baud rates and enables the serial interfaces (PC and BT) as defined in the **settings.h** file Attaches listeners to both the serial ports that trigger when a message is received

Definition at line 54 of file serial.cpp.

Setup Class Reference

```
#include <i2c_setup.h>
```

Public Member Functions

- char **get_dc_status** (void)
- char **IF_setup_led_expansion_ic** (void)
- void **IF_setup_gpio_expansion_ic** (void)
- void **IF_read_aux_ic_data** (void)
- void **IF_parse_gpio_byte0** (char byte)
- void **IF_parse_gpio_byte1** (char byte)
- void **IF_handle_gpio_interrupt** (void)
- void **IF_update_gpio_inputs** (void)
- void **IF_set_base_LED** (char state)
- void **IF_set_IR_emitter_output** (char emitter, char state)
- unsigned short **IF_read_IR_adc_value** (char adc, char index)
- char **IF_is_switch_pressed** (void)
- char **IF_get_switch_state** (void)
- void **IF_write_to_led_ic** (char byte_0, char byte_1)
- void **IF_setup_temperature_sensor** (void)
- float **IF_read_from_temperature_sensor** (void)

Detailed Description

The **Setup** class contains internal functions that initiate the I2C components on the robot and send the low level messages to read to these components. The functions within this class are intended to be used by other classes to provide higher level functionality, so are not documented in the API.

Definition at line 32 of file i2c_setup.h.

Sound Class Reference

```
#include <sound.h>
```

Public Member Functions

- void **play_audio_string** (char *tune)
- void **play_tune** (char *tune, char length)
- char **IF_check_pic_firmware** (void)

Detailed Description

Sound class Functions that generate audio tones using the sound module on the PIC coprocessor, where used

Definition at line 32 of file sound.h.

Member Function Documentation

*void Sound::play_audio_string (char * * **tune***)*

Play a tune defined by the given null terminated string

Parameters:

tune	- The tune to play

Definition at line 27 of file sound.cpp.

*void Sound::play_tune (char * * **tune**, *char* **length***)*

Play a tune defined by the given string

Parameters:

tune	- The tune to play
length	- The number of characters in the string

Definition at line 33 of file sound.cpp.

File Documentation

D:/GoogleDrive/PsiSwarm/Code/MBED Code/PsiSwarmV8_CPP/settings.h File Reference

Header file containing PsiSwarm define headings.

Macros

- #define **USE_MOTOR_CALIBRATION** 1
  Use EPROM stores motor calibration values 0=off 1=on
- #define **OFFSET_MOTORS** 1
  Offset the motors to prevent stalling at PWM values below 0.2 . 0=off 1=on.
- #define **ENABLE_DEMO** 1
  Enable if demo mode can be used at turn-on. 0=off 1=on.
- #define **ENABLE_BASIC** 1
  Enable if the Basic interpreter is being used. 0=off 1=on.
- #define **ENABLE_BLUETOOTH** 1
  Enable if the BlueSmirf module is being used. 0=off 1=on.
- #define **ENABLE_PC_SERIAL** 1
  Enable the PC serial connection. 0=off 1=on
- #define **BLUETOOTH_BAUD** 115200
  Bluetooth baud rate
- #define **PC_BAUD** 115200
  PC baud rate
- #define **DEBUG_MODE** 1
  Show extended debug messages over debug output stream
- #define **SHOW_VR_WARNINGS** 0
  Enable voltage regulator warning messages
- #define **USE_LED3_FOR_INTERRUPTS** 1
  Use MBED LED3 to show ISR entry
- #define **USE_LED4_FOR_VR_WARNINGS** 1
  Use MBED LED4 to show VR warnings
- #define **HALT_ON_GPIO_ERROR** 1
  Halt the system if no GPIO IC is detected
- #define **HALT_ON_ALL_VREGS_LOW** 0
  Halt the system if all VREGs give a low output
- #define **DEBUG_OUTPUT_STREAM** 1
  Output stream for debug messages; 1=PC, 2=BT, 4=Onboard Display [combinations possible]

List of Serial Commands

The following table contains the list of predefined Serial commands in `serial.cpp` that

Command Byte		Function	Notes	Example (Hex Bytes)	Example Description
1	`0x01`	Set Left Motor	MSB of Value[0] indicates direction (1 = Forward)	`1D,01,7F,FF,1D`	Sets left wheel turning backwards at full speed
2	`0x02`	Set Right Motor	MSB of Value[0] indicates direction (1 = Forward)	`1D,02,BF,FF,1D`	Sets right wheel turning forward at half speed
3	`0x03`	Set Both Motors	MSB of Value[0] indicates direction (1 = Forward)	`1D,03,3F,FF,1D`	Sets both wheels turning backwards at half speed
4	`0x04`	Brake Left Motor		`1D,04,00,00,1D`	Actively brake left wheel
5	`0x05`	Brake Right Motor		`1D,05,00,00,1D`	Actively brake right wheel
6	`0x06`	Brake Both Motors		`1D,06,00,00,1D`	Actively brake both wheels
7	`0x07`	Stop Both Motors		`1D,07,00,00,1D`	Stops the turning of the wheels, allowing roll
8	`0x08`	Turn on the Spot	MSB of Value[0] indicates direction (1 = Clockwise)	`1D,08,FF,FF,1D`	Turn clockwise on the spot at full speed
9	`0x09`	Set Each Motor	Value[0] sets the left motor speed, Value[1] the right motor speed. The MSB of each indicates the respective direction (1 = Forward)	`1D,09,BF,7F,1D`	Sets the left motor to half speed forwards, and the right motor to full speed backwards
10	`0x0A`	Set All Outer LED States	Value[0] and Value[1] set the green and red states of the outer LEDs respectively, each bit representing an LED	`1D,0A,FF,03,1D`	Enables the red sections of all outer LEDs, and the green section of three of these
11	`0x0B`	Set All Outer Red LED States	Value[0] sets red states of the outer LEDs, each bit representing an LED	`1D,0B,2F,00,1D`	Enables the red sections of five of the outer LEDs
12	`0x0C`	Set All Outer Green LED States	Value[0] sets green states of the outer LEDs, each bit representing an LED	`1D,0C,FF,00,1D`	Enables the green sections of all the outer LEDs
13	`0x0D`	Set Outer LED State	Value[0] contains the index of the LED (0 indexed) and Value[1] the state (0 = Off, 1 = Red, 2 = Green, 3 = Red and Green)	`1D,0D,02,01,1D`	Sets LED 3 to the red state
14	`0x0E`	Set Centre LED State	Value[0] contains the state to set the centre LED (0 = Off, 1 = Red, 2 = Green, 3 = Red and Green)	`1D,0E,02,00,1D`	Sets the centre LED to the green state
15	`0x0F`	Set Centre LED Brightness	Value[0] and Value[1] make up an unsigned float for the brightness value	`1D,0F,7F,FF,1D`	Sets the centre LED to half brightness
16	`0x10`	Set MBED LED States	The four most significant bits of Value[0] indicate set the four MBED LED states	`1D,11,7F,FF,1D`	Enables three of the four MBED LEDs
17	`0x11`	Blink Outer LEDs	Value[0] and Value[1] make up an unsigned float for the blink time (65535 = 1 sec)	`1D,12,01,00,1D`	Blink the outer LEDs for half a second
18	`0x12`	Set Base LED State	Value[0] sets the state of the base LED (1 = On)	`1D,13,03,FF,1D`	Turn on the base LED
19	`0x13`	Set Centre LED State and Brightness	Value[0] contains the state to set the centre (0 = Off, 1 = Red, 2 = Green, 3 = Red and Green) and Value[1] the brightness	`1D,14,01,00,1D`	Sets the centre LED to the red and green state at full brightness
20	`0x14`	Set Display	Value[0] sets the state of the Psi Swarm LCD screen (0 = clear screen, 1 = PC Connection Started, 2 = PC Connection Terminated, 3 = Android Device Connected, 4 = Android Device Disconnected)	`1D,14,01,00,1D`	Display “PC Connection Started” on the Psi Swarm LCD
21	`0x15`	Set LCD Cursor	Value[0] sets the line number (0 indexed, 0-1), and Value[1] sets the character number (0 indexed, 0-15)	`1D,15,01,00,1D`	Set the cursor to the first character of the second line
22	`0x16`	Print Characters	Value[0] and Value[1] set two ASCII characters to be printed on the LCD	`1D,16,4F,4E,1D`	Print the characters ”ON” to the LCD screen
23	`0x17`	Set LCD	Value[0] and Value[1] make up an unsigned float for the Psi Swarm LCD	`1D,17,FF,FF,1D`	Set the LCD brightness to

		Brightness	brightness		100%
30	0x1E	Set Debug Mode	Value[0] sets the debug state (1=On) and Value[1] sets the debug output mode (1 = PC, 2 = Bluetooth, 4 = LCD)	1D,1E,01,02,1D	Enables the debug mode and sets the output mode to the Bluetooth serial connection
31	0x1F	Set Demo Mode	Value[0] sets the demo state (1=On)	1D,1F,01,00,1D	Enable demo mode
32	0x20	Enable User Code	Value[0] sets the state of the execution of user code (1 = Enabled)	1D,20,00,00,1D	Disable user code execution
33	0x21	Pause User Code	Value[0] and Value[1] make up an unsigned float for the user code pause time (Max 10s = 65535)	1D,21,03,FF,1D	Pause the user code for 2.5 seconds
34	0x22	Reset Wheel Encoders		1D,22,00,00,1D	Reset the value of the wheel encoder counters to zero
35	0x23	Enable Bluetooth Commands	Value[0] sets the state of the execution of bluetooth commands (1 = Enabled)	1D,23,01,00,1D	Enable the execution of Bluetooth commands
36	0x24	Set Outer IR Pulse Delay	Value[0] and Value[1] make up an integer for the pulse delay time in milliseconds)	1D,24,00,32,1D	Set the outer IR pulse delay to 50ms
37	0x25	Set Base IR Pulse Delay	Value[0] and Value[1] make up an integer for the pulse delay time in milliseconds)	1D,25,00,64,1D	Set the base IR pulse delay to 100ms
40	0x28	Get Left Motor Speed	Returns four ascii characters, making up an integer value (4095 = 100%)	1D,28,00,00,1D	Response: 1F,04,33,30,37,31 Left Motor = 3071(75%)
41	0x29	Get Right Motor Speed	Returns four ascii characters, making up an integer value (4095 = 100%)	1D,29,00,00,1D	Response: 1F,04,34,30,39,35 Right Motor = 4095(100%)
42	0x2A	Get Brake States	Returns three ascii characters, where the first is the left brake state (1 = On), the second a comma, and the third the right brake state	1D,2A,00,00,1D	Response: 1F,03,30,2C,30 Left and right brakes are off (0,0)
43	0x2B	Get Motor States	[Not yet implemented]	1D,2B,00,00,1D	
44	0x2C	Get Wheel Encoders	Returns a minimum of three ASCII characters, which will contain the left encoder value, a comma, then the right encoder value	1D,2C,00,00,1D	Response: 1F,04,32,2C,31,31 Left encoder = 2, Right encoder = 11 (2,11)
50	0x32	Get LED States	Returns four ascii characters, where the first two make up a hexadecimal value of the green states of the outer LEDs (LSB = LED 1), and the second two the red states	1D,32,00,00,1D	Response: 1F,04,30,63,31,30 LED 3 and 4 are green and LED 5 is red (0x0C 0x10)
60	0x3C	Get Software Version	Returns a minimum of four asci characters, making up a float value of the software version	1D,3C,00,00,1D	Response: 1F,04,30,2E,32,30 Software Version = 0.20
61	0x3D	Get Running Time	Returns ascii characters, making up a float value of the running time (seconds)	1D,3D,00,00,1D	Response: 1F,05,32,37,2E,35,30 Running time = 27.50s
62	0x3E	Get Robot ID	Returns ascii characters, making up an integer value of the psi swarm ID	1D,3E,00,00,1D	Response: 1F,01,34 Robot ID = 4
63	0x3F	Get Switch States	Returns two ascii characters, making up a hexadecimal value of the switch state (LSB = Switch 1)	1D,3F,00,00,1D	Response: 1F,02,30,32 Switch 2 is on, all other switches are off
64	0x40	Get User Code State	Returns an ascii character which is an integer value of the state of user code execution (1 = Enabled)	1D,40,00,00,1D	Response: 1F,01,31 User code execution is enabled
65	0x41	Get Test Response	Returns three ascii characters making the string "PSI", allowing the serial connection to be tested	1D,41,00,00,1D	Response: 1F,03,50,53,49 Correct serial read returns "PSI"
66	0x42	Get Program Name	Returns ascii characters making a string of the program name	1D,42,00,00,1D	Response: 1F,04,54,65,73,74 Program Name = "Test"
67	0x43	Get Author Name	Returns ascii characters making a	1D,43,00,00,1D	Response:

			string of the author name		1F,03,59,52,4C Author Name = “YRL”
68	0x44	Get Debug Mode	Returns two ascii characters, the first is the debug state (1 = On), and the second the debug output mode (1 = PC, 2 = Bluetooth, 4 = LCD)	1D,44,00,00,1D	Response: 1F,02,31,34 Debug state = Enabled, Debug output mode = LCD
69	0x45	Get System Warnings	Returns an ascii characters, which will make a hexadecimal value of the system warnings (1 = Error) (Bit 0 = I2C LED Driver, Bit 1 = I2C Main GPIO IC, Bit 2 = I2C Aux GPIO IC)	1D,45,00,00,1D	Response: 1F,01,32 Error with Main GPIO IC I2C Acknowledge
80	0x50	Store Outer IR Background Values		1D,50,00,00,1D	Update the stored back ground values for the outer IR sensors
81	0x51	Store Outer IR Reflection Values		1D,51,00,00,1D	Update the stored reflection values for the outer IR sensors
82	0x52	Store Outer IR Values		1D,52,00,00,1D	Update the stored background and reflection values for the outer IR sensors
83	0x53	Store Base IR Background Values		1D,53,00,00,1D	Update the stored background values for the base IR sensors
84	0x54	Store Base IR Reflection Values		1D,54,00,00,1D	Update the stored reflection values for the base IR sensors
85	0x55	Store Base IR Values		1D,55,00,00,1D	Update the stored background and reflection values for the base IR sensors
86	0x56	Store Outer and Base IR Values		1D,56,00,00,1D	Update the stored background and reflection values for the outer and base IR sensors
87	0x57	Get Obstacle and Line Presence	Returns 4 ascii characters making up two 8-bit values in hexadecimal, where the first is the binary obstacle states (LSB = Outer Sensor 1), and the second is the binary line states (LSB = Base Sensor 1) (1 = Line/Obstacle detected)	1D,57,00,00,1D	Response: 1F,04,31,30,30,36 Obstacle on Sensor 5, Line on Sensor 2 and 3
90	0x5A	Get Outer IR Background Value	Value[0] contains the index of the LED (0 indexed) (0 - 7). Returns 3 ascii characters making up a hexadecimal value (Max = 4095)	1D,5A,01,00,1D	Response: 1F,03,32,38,61 Sensor 2 BG = 650 (16%)
91	0x5B	Get Outer IR Reflection Value	Value[0] contains the index of the LED (0 indexed) (0 - 7). Returns 3 ascii characters making up a hexadecimal value (Max = 4095)	1D,5B,04,00,1D	Response: 1F,03,30,64,63 Sensor 5 Ref = 220 (5.4%)
92	0x5C	Get Outer IR Background Values	Returns 27 ascii characters, making up eight, three character, hexadecimal values (First value = Sensor 1) (Max = 4095)	1D,5C,00,00,1D	Response: 1F,18,30,65,31,30,36,34.. Sensor 1 BG = 225 (5.5%) Sensor 2 BG = 100 (2.4%)
93	0x5D	Get Outer IR Reflection Values	Returns 27 ascii characters, making up eight, three character, hexadecimal values (First value = Sensor 1) (Max = 4095)	1D,5D,00,00,1D	Response: 1F,18,62,62,38,31,66,34.. Sensor 1 Ref = 3000 (73%) Sensor 2 Ref = 1000 (24%)
94	0x5E	Get Base IR Background Value	Value[0] contains the index of the LED (0 indexed) (0 - 4). Returns 3 ascii characters making up a hexadecimal value (Max = 4095)	1D,5E,01,00,1D	Response: 1F,03,32,38,61 Sensor 2 BG = 650 (16%)
95	0x5F	Get Base IR Reflection Value	Value[0] contains the index of the LED (0 indexed) (0 - 4). Returns 3 ascii characters making up a hexadecimal value (Max = 4095)	1D,5F,04,00,1D	Response: 1F,03,30,64,63 Sensor 5 Ref = 220 (5.4%)
96	0x60	Get Base IR Background Values	Returns 15 ascii characters, making up five, three character, hexadecimal values (First value = Sensor 1)(Max = 4095)	1D,60,00,00,1D	Response: 1F,0F,30,65,31,30,36,34.. Sensor 1 BG = 225 (5.5%) Sensor 2 BG = 100 (2.4%)
97	0x61	Get Base IR Reflection Values	Returns 15 ascii characters, making up five, three character, hexadecimal	1D,61,00,00,1D	Response: 1F,0F,62,62,38,31,66,34..

			values (First value = Sensor 1)(Max = 4095)		Sensor 1 Ref = 3000 (73%) Sensor 2 Ref = 1000 (24%)
98	`0x62`	Get Base IR Corrected Reflection Values	Returns 15 ascii characters, making up five, three character, hexadecimal values, where the background IR has been subtracted from the reflection (First value = Sensor 1) (Max = 4095)	`1D,62,00,00,1D`	Response: `1F,0F,62,62,38,31,66,34..` Sensor 1 C.Ref = 3000 (73%) Sensor 2 C.Ref = 1000 (24%)
99	`0x63`	Get Outer IR Corrected Reflection Values	Returns 27 ascii characters, making up eight, three character, hexadecimal values, where the background IR has been subtracted from the reflection (First value = Sensor 1) (Max = 4095)	`1D,63,00,00,1D`	Response: `1F,0F,62,62,38,31,66,34..` Sensor 1 C.Ref = 3000 (73%) Sensor 2 C.Ref = 1000 (24%)