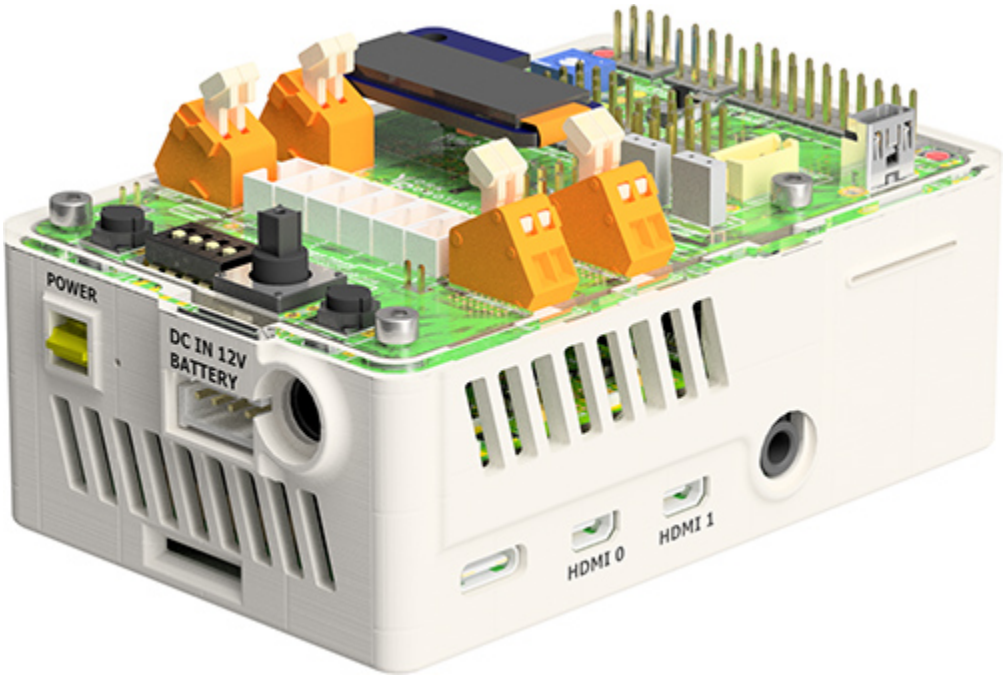
York Robotics Kit

Release 0.21

James Hilder

Feb 24, 2020

CONTENTS:



The **York Robotics Kit** is a platform design to allow the Raspberry Pi family of microcomputers to control a wide variety of robotics hardware with the minimum extra electronics assembly. It comprises a pair of PCBs which attach above a Raspberry Pi SBC *(single-board computer)*. This document describes the initial version of the York Robotics Kit, comprising the **YRL039** Power Supply Board and the YRL040 YRK Board, which are primarily designed to be used with the Raspberry Pi 4 B and Raspberry Pi 3 B(+) series of computers.

This is still very much a prototype and development design and should be regarded as such!

CHAPTER

ONE

CONSTRUCTION

YRL040 PCB Variants

1.0 *[Sep 2019]* Original version as seen in pictures, models and diagrams

1.1 *[Dec 2019]* Navigation switch changed to surface-mount model, support for 3.5mm screw terminals for motors added.

This document assumes PCBs have been manufactured and all components assembled (**see separate documentation for PCB construction**). The PCBs have been designed using Autodesk Eagle software and are both 2-layer construction, designed to conform to Eurocircuits Class 6 design guidelines. Information about the Eagle CAD source files, schematic diagrams and layout documents can be found in the *Schematic Diagrams* section. Both PCBs are two-layer boards and designed to be both relatively cheap to manufacture and relatively easy to assemble.

PCB Name	Description
YRL039	Power Supply PCB
YRL040	Main YRK PCB

CHAPTER

TWO

YRL039 POWER SUPPLY

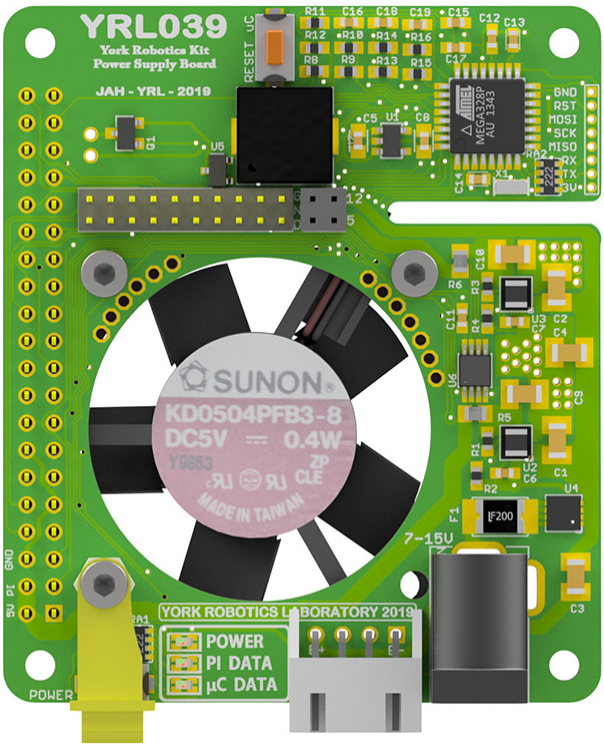


Fig. 1: YRL039 Power Supply

The YRL039 Power Supply Board as designed to serve six main functions:

- Take a battery or DC input and convert into stable, separate 5V supplies. Each of the 2 5V supplies is rated to provide at least **2.5A** current. One supplies the Raspberry Pi *(including USB peripherals)*, the other provides the **5V_AUX** rail for most other components.
- Provide an interface between Pi GPIO pins and respective pins on YRL040 PCB. It was desired to rotate and reduce in size the 40-pin GPIO header on the Pi to allow more space for edge-mount connectors on either side of the YRL040 PCB **(for motors, sensors and servos etc)**.
- Provide hardware monitoring of temperature, input voltage, output voltages and output currents.
- Provide a software on-off power button with forced shutdown option
- Provide an 35mm fan based active cooling for the Pi microcontroller, the power supplies and the YRL040 mounted above.
- Generate simple audio tones using a Piezo buzzer

All the components except for the fan are mounted to the top-side of the PCB. The **TPS82130** TI 3A Step-Down Converter Module is core components of each of the two 5V power supplies.

2.1 Programming the YRL039

The YRL039 uses at ATMega328P microcontroller to control the soft-power switch, make audio tones and report voltage, current and temperature readings using the I2C bus. The standard code is found in the `atmega_code/yrl039_arduino` subfolder of the GIT repository; if special functionality is required, such as lowering the battery-low cutoff voltage or reducing the temperature at which the fan operates, it may be necessary to reprogram the microcontroller. This is easiest to do with the **YRL030** *FTDI Interface and Programming Board* that contains the 8-pin, 1.25mm pitch connector used for programming and serial data; the **YRL030** can be used for both flashing the bootloader to the microcontroller using the **ICSP** interface and also uploading code and debugging data using a USB serial interface.

CHAPTER

THREE

YRL040 MAIN PCB

The main YRL040 PCB has been designed such that the vast majority of electronic components are surface mounted on the underside of the board.

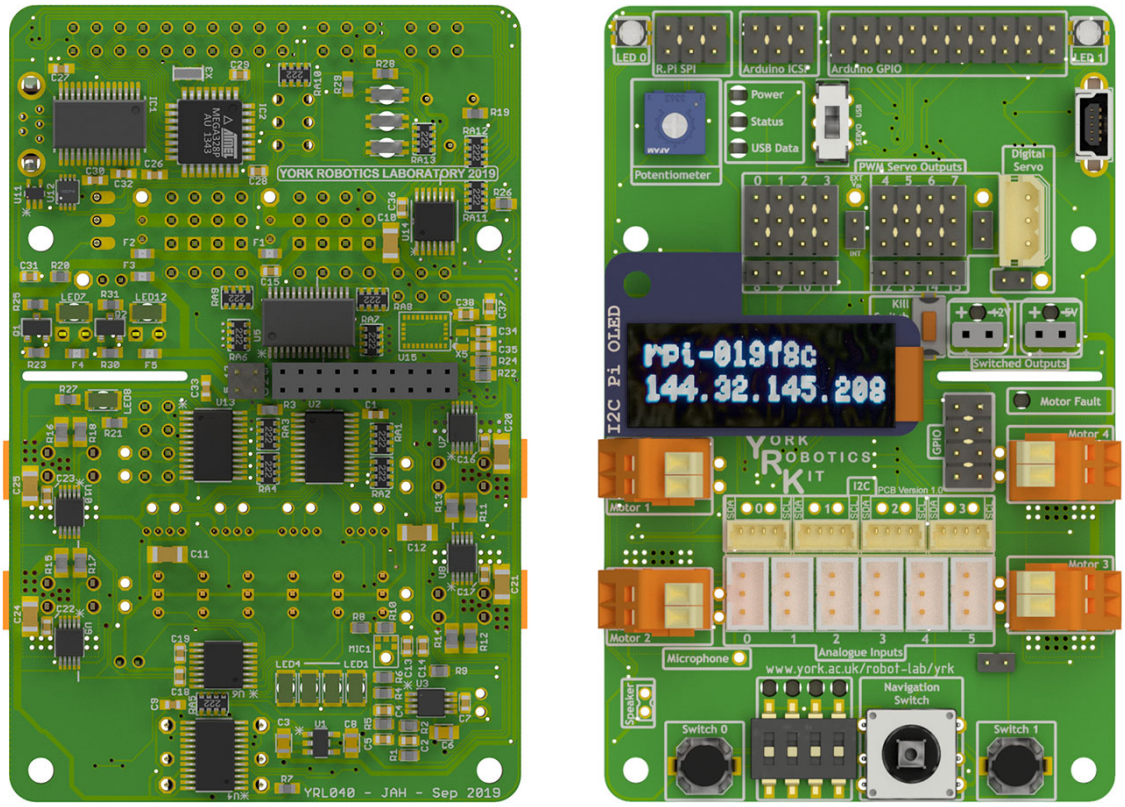


Fig. 1: YRL040 Main PCB *(Bottom and Top Views)*

The board contains the following interfaces:

- 8-way I2C switch providing 4 user busses at 3.3V, a 5V bus used by on board PWM driver and Arduino, a bus design for use with an OLED display module, a bus for communication with the YRL039 PSU and a bus for all other internal connections.
- 4 I2C H-Bridge Motor Drivers based on the TI **DRV8830** IC.
- 16-channel PWM Driver suitable for use with analogue servo motors
- 16-channel GPIO expander providing 5-way navigation switch, 4-way DIP switch, 2 push-button switches and LEDs

- Addition 16-channel GPIO expander providing motor driver fault monitoring, a 5V and a 12V switched output, a kill-switch and 8 user GPIO pins
- 8-channel, 8-bit ADC with 6 ports optimised for use with Sharp analogue distance sensors, a channel connected to a potentiometer and a spare channel.
- A mono audio amplifier connected to a PWM audio channel on the Raspberry Pi
- An I2S mono microphone module
- An Arduino compatible ATMega328P microcontroller for use with wheel encoding, digital servo motors and other tasks.

CHAPTER

FOUR

ASSEMBLY

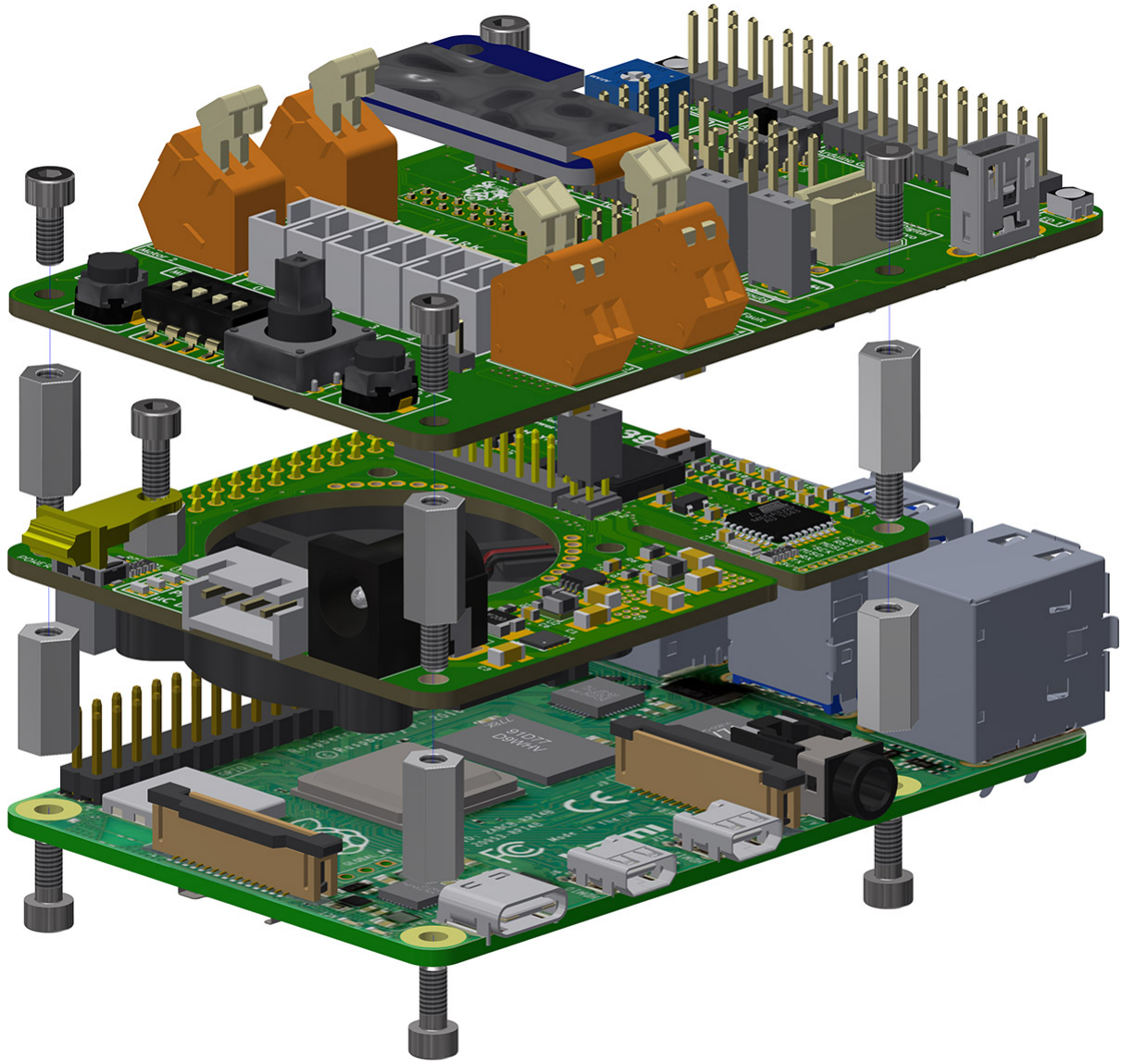


Fig. 1: Exploded view of YRK showing 11mm standoffs and M2.5 screws

The YRL039 attaches above the Raspberry Pi PCB using the 4 x 11mm length, M2.5 diameter standoffs. Another layer of 4 x 11mm standoffs is used to attach the *YRL040* PCB above the **YRL039** power supply PCB. It is recommended that

M:F **(male one end, female other)** standoffs are used in the upper layer, with the male threads pointing downwards through the *YRL040* PCB, then M:M standoffs **(sometimes called spacers)** below on the bottom layer.

The assembly can be mounted inside a further case, described below, or can be mounted directly onto a chassis or further standoffs. Consideration of the airflow path should be taken, particularly when fully enclosed inside a robot. The Raspberry Pi alone can generate a significant amount of heat and rapidly reaches a point at which it will throttle clock speed if it is not adequately cooled.

CHAPTER

FIVE

CASE

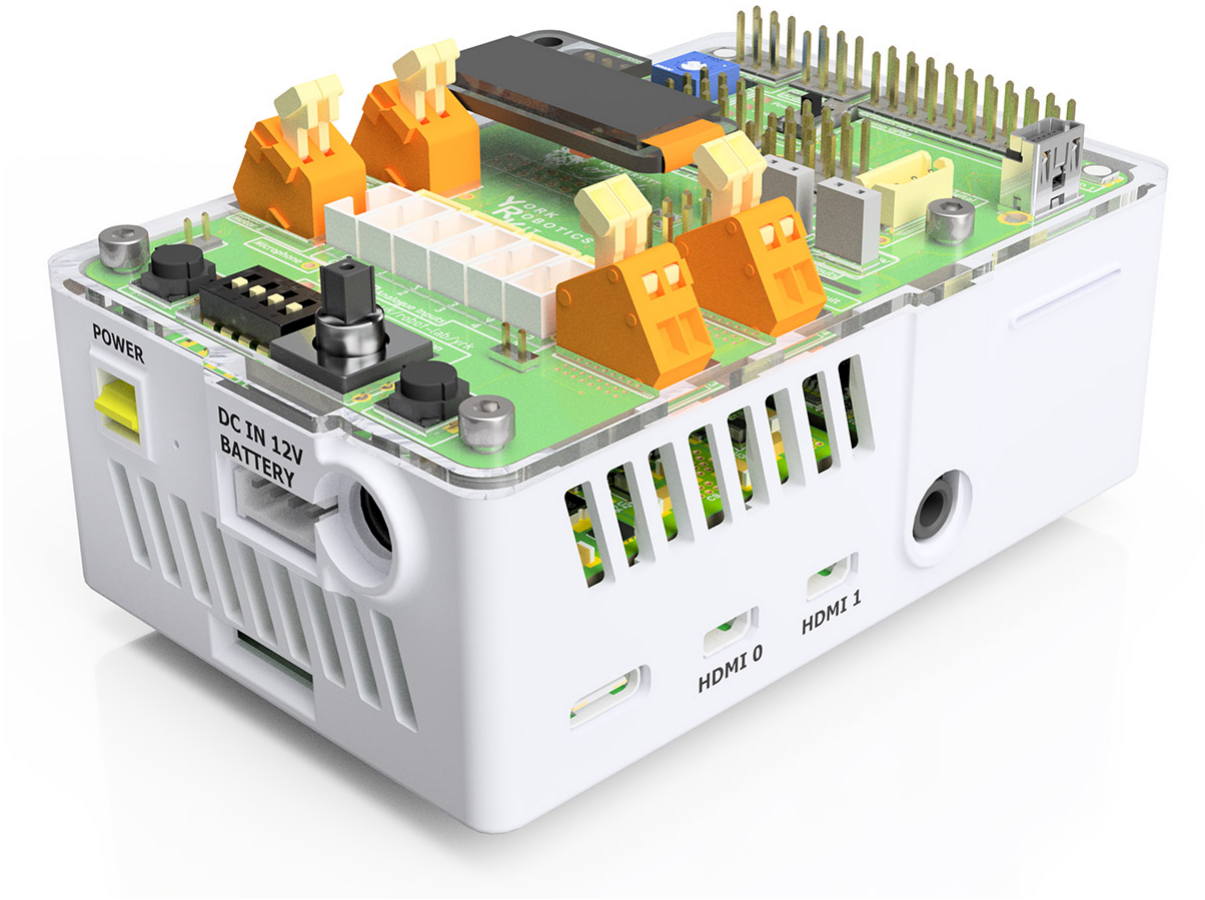


Fig. 1: York Robotics Kit mounted in Polyjet case

A case designed *specifically* for the **Raspberry Pi 4B** series of computer, the **YRL039** and the **YRL040** PCBs has been designed. This particular design is intended to be printed on a Polyjet class of 3D printer, with very fine tolerancs and gaps. A more general design for FDM could easily be implemented, but is not essential. If designed a new housing [or placing assembly within a robot chassis design etc], consider airflow route carefully. Pi 4 devices will generate substantial heat and the fan needs some route to direct air across the Pi 4 CPU, but also ideally the power supply elements on the top side of YRL039 and also components on the underside of the YRL040 PCB. Those components likely to dissipate the most heat [motor drivers, PWM driver, amplifier and active outputs] are all towards the lower half of the PCB, which should receive forced convection from the fan.

CHAPTER

SIX

CONNECTING HARDWARE

This section of the user guide explains how to connect hardware such as batteries, motors, servos and sensors to the **YRK** to make a complete electronics hardware design for a robot. The simplified pin-out diagram is shown below, a more detailed view is available in the `Documents` sections.

6.1 Power Supply

The YRK can be powered by any standard 12V power supply with a 2.1mm DC jack (centre positive) and 3A or greater current rating. Using a tethered power supply such as this is strongly recommended to be used as much as possible to preserve battery life if motion of the robot is not required. Do not connect a power supply when the battery is connected and vice-versa *(shutdown between changing power sources by pressing the power button)*.

6.2 Battery

The YRK is primarily inteded to be run using a 3-cell Lithium-Polymer battery pack, connected using the industry-standard **JST-XH** 4-pin 0.1" pitch connector used as the balance charging connector. These connectors and cables are typically rated for 2A current which is enough for normal use of the YRK [which has a **2.5A** input Polyfuse]. Where high currents are needed, such as powering multi servo motors, the high current cable from the battery should be soldered *(via a self-made adapter)* to the appropriate power points on the **YRL040** PCB.

The **YRL039** power supply board has a low-voltage dropout and should work effectively from voltages as low as **5.5V** to a **17V** high, enabling 2-cell and 4-cell operation Li-Po (and a range of other lead-acid, Ni-Mh, NiCad, LiFe and other rechargeable battery technologies, provided they can provide around 5A peak current at 5V). Relevant changes to the `yrk.settings` file should be made to reflect the battery used if not the default 3-cell configuration.

Only the outer-two connections of the **JST-XH** connector are used. It is also possible to use the 2.1mm DC jack as the input source. Always disconnect the battery after use and charge using a suitable charger (in a fire-safe bag if using a Li-Po battery pack). The power supply board is not presently optimised for ultra low-current and the residual current draw (from the ATMega microcontroller) will discharge the battery even when beyond a damagingly low value. General convention suggest a per-cell voltage of 3.0V is the absolute minimum a Li-Po battery should be allowed to discharge to *(ie 9V for 11.1V 3-cell battery)* before permanent damage is likely to be done. The Arduino code on the **YRL039** can be configured to automatically switch off power supplies below a critical low voltage but this needs to be correctly configured for the battery technology used. If the `core.py` software is *not* run the user must implement some method of periodically monitoring the battery voltage and provide suitable user warnings when low levels are reached, otherwise batteries can rapidly deteriorate.

The YRK connected to a Raspberry Pi 4 will typically consume approximately 800mA on the 5V rails at idle, most the current powering the Raspberry Pi and a smaller amount of the **5V_AUX** rail (mostly powering the fan and LEDs). Motors and sensors will add significantly to this load. This relatively high idle load means that battery capacity should be at least 1000mAH, and this capacity would provide at best around 1 hours use *(and significantly less in high load*

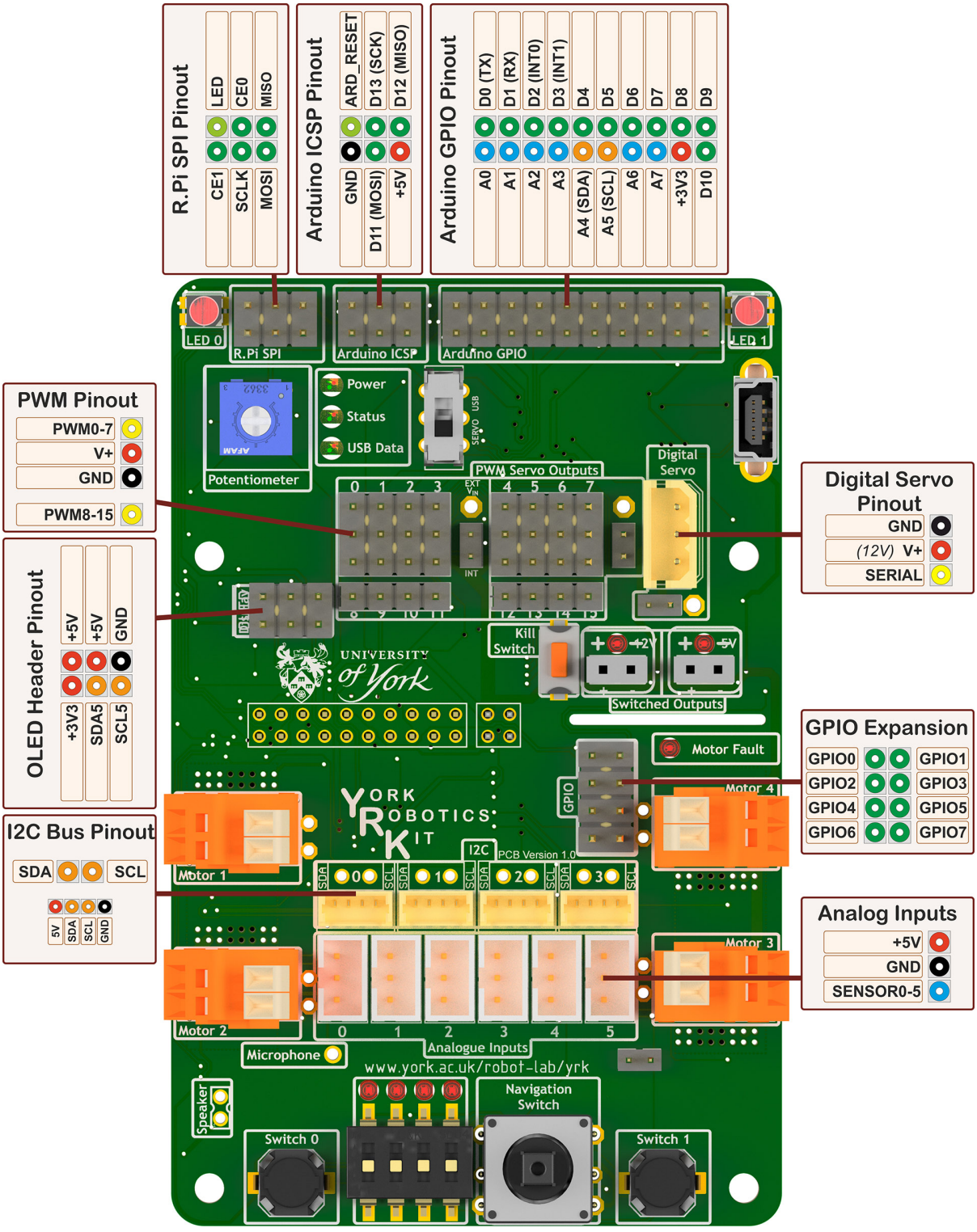


Fig. 1: Pin-out and wiring diagram for York Robotics Kit

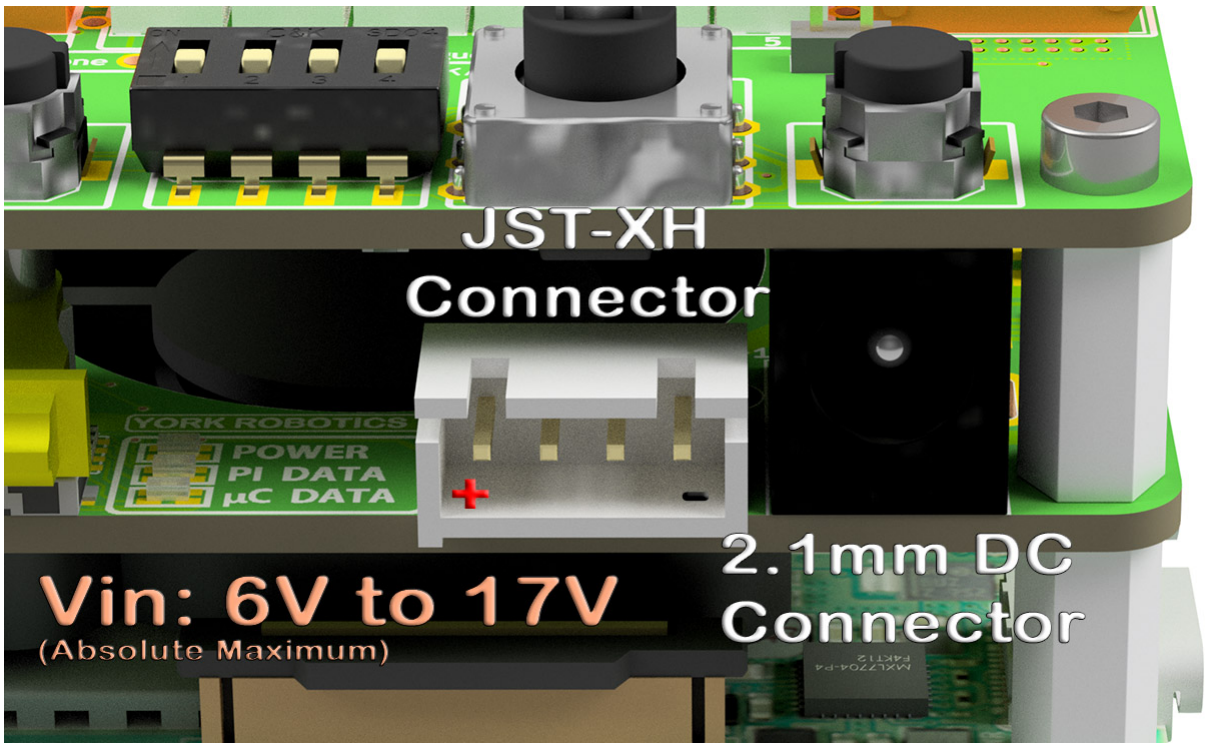


Fig. 2: Location of the 2.1mm DC jack and JST-XH battery connector

cases). The use of a lower-power Raspberry Pi, such as the model 3A, might be considered when long battery life on small batteries is desired, at the expense of memory, processing capability and expandability.

6.3 DC Motors

There are four serial H-Bridge motor drivers, based on the **DRV8830** TI Motor Driver. The PCB design limits each motor driver to approximately **800mA** current, powered from the *5V_AUX* supply. Having all four motors drawing this peak current for sustained periods will exceed the rating of the power supply. This current limit *(and voltage rating)* does restrict the motor driver to using small motors, such as the widely-available 3mm shafted **micro-metal gear motors**. Before using a different size of motor it is recommend to check *(such as by using a bench PSU)* what the stall and no-load currents at **5V**.

The holes on the unpopulated PCB allow the motors to be connected to either **Wago** push-fit terminals or **(on PCB version 1.1)** 3.5mm pitch screw terminals. With either connector, a remaining pair of holes will be accessible on the PCB should a direct soldered lead be required. The motor driver can detect fault conditions (undervoltage, overvoltage and overtemperature events) and when the `yrk.core` program is running these will be indicated by the red fault LED above the top-right motor output. If persistent fault conditions check the motor is working correctly and doesn't draw excessive current for the design; micro-metal-gear motors are relatively delicate.

6.4 Servos

The YRK can control both standard analogue servo motors (8 directly attachable, 8 further channels available via breakout), and digital servo motors via an Arduino-based software interface.

6.4.1 Analogue Servos

Analogue servos are operated using a **PCA9685** I2C LED driver IC. Whilst primarily designed to allow I2C brightness control of up to 16 LEDs, it can effectively work as a analogue servo controller. Analogue servos typically operate with a **20mS** period width *(50Hz PWM frequency)*, and expect a pulse width in the **1ms - 2ms** range [with **1.5ms** being the middle point of the servo rotation]. The **PCA9685** lets us fix the PWM frequency for all outputs and effectively becomes an I2C servo controller.

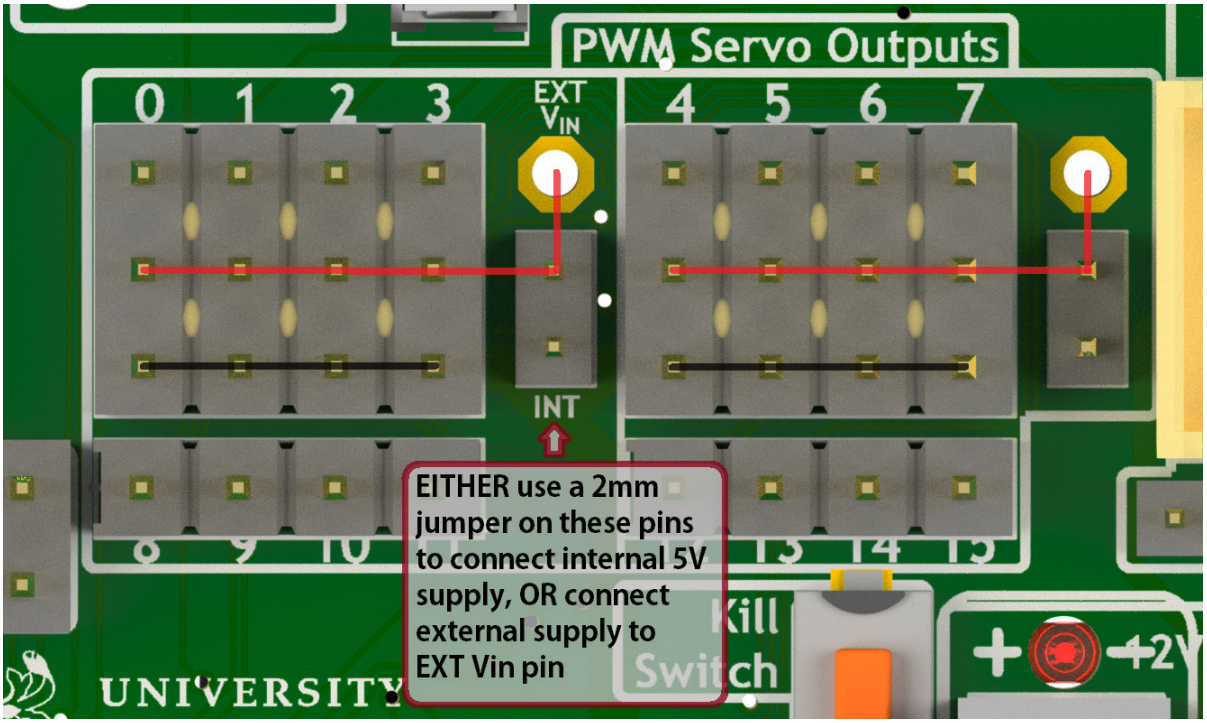


Fig. 3: Power connections for the 2-banks of PWM analogue servo outputs

There are 16 available outputs on the YRL040 PCB, located in the middle-top of the PCB. Eight of these are available as full 3-pin outputs, where DC power (**+** and **GND**) can be supplied to the servo. Most analogue servos come hardwired with a three-pin 0.1" pitch socket attached at the wire tail. Different colour schemes are used for the wiring, and it is important to be careful checking the orientation of the plug; as a general rule the lightest colour will the control signal (top side of connector) and the darkest will be ground.

Pin Number	Signal	Futaba	JR	Hitec
3 [Top]	Control	White	Orange	Yellow or White
2	V+	Red	Red	Red or Brown
1 [Bottom]	Ground	Black	Brown	Black

These 8 complete connections are split into two banks of four. Each of these banks can be supplied with DC either by the internal 5V supply **(by using a 2mm jumper)**, or to an external positive input, by soldering a suitable cable onto the hole on the board. If the internal supply is used, the total current for each bank **must not excede 1A** (a pair of

0603 fuses are included on the board . This is due to the overall current limitations on the board. For this reason it is strongly recommended to only use very small, low-current servos, and to spread load across both banks, if using the internal supply.

Another 8 PWM outputs are available just below the primary 8, but these cannot be used directly with a 3-pin connector. In situations where a large number of servos are required simultaneously, a small break-out board allowing direct power connection would be a sensible option. The circuit is the same as used on the Adafruit 16-channel PWM servo driver.

Code for the analogue servo control is in the `yrk.pwm` module. Examples of the use of the PWM driver to control servos can be found in `examples.console`.

6.4.2 Digital Servos

The York Robotics Kit is designed to support digital servos from the **(Dynamix AX- and MX- series)** via code on the Arduino microcontroller.

To do: This section and code not completed yet!

6.5 Analogue Inputs

The YRK includes an I2C based, 8-channel, 8-bit analogue to digital converter IC. Whilst this can be used for anything requiring analogue inputs, it is primarily intended for use with analogue distance sensors manufactured by Sharp, in particular the **2Y0A21** and **2Y0A41** models. The reference voltage is set to **2.5V**, meaning the returned value is approximately equal to the voltage mulitplied by 100.

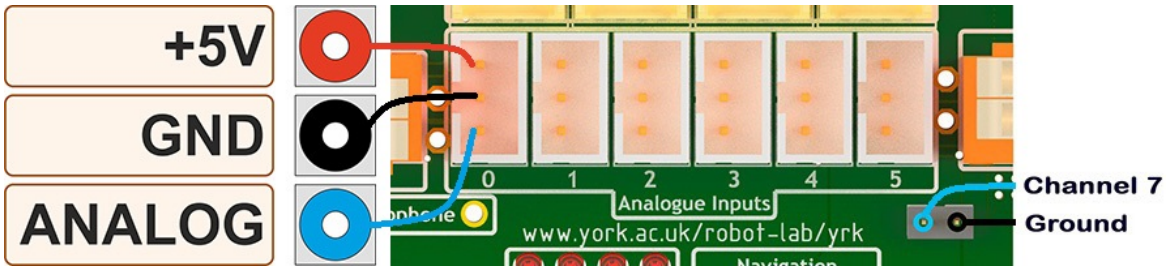


Fig. 4: Analogue input channels 0-5 *(JST PH sockets)* and channel 7

6.5.1 Cables

The Sharp distance sensors use a 3-pin JST PH series connection **(note the newest models use a JCTC connector instead of a JST)**. 6 matching JST-PH connections are available on the York Robotics Kit, each providing the analogue-input and 5V power supply required by the sensor. A suitable complete pre-made harness has not been sourced, but it is possible to buy pre-crimped leads from JST which make creating harnesses quick and simple *(if expensive)*.

JST Part Number	Farnell Part	Description	Unit Price *[per 100]*
01SPHSPH-26L150	2065431	150mm PH-PH Lead	0.416
01SPHSPH-26L300	2065432	300mm PH-PH Lead	0.439
PHR-3	3616198	3-pin PH Receptacle	0.032

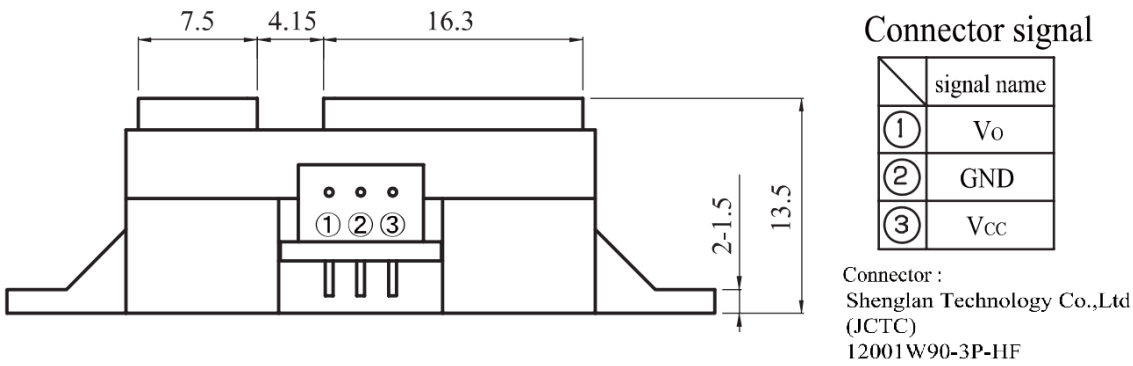


Fig. 5: Wiring diagram of Sharp Distance Sensors

To assemble the harness, place one receptacle face-up and the other face-down then connect top-to-top, middle-to-middle and bottom-to-bottom, as seen in the photograph below.

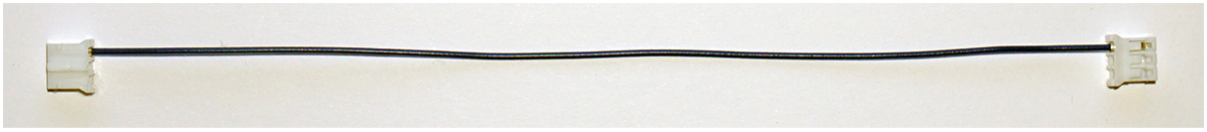


Fig. 6: Assembly of first wire in JST PH cable for use with Sharp Distance Sensors

6.5.2 Datasheets

Sharp 2Y0A21 [10-80cm]

Sharp 2Y0A41 [4-40cm]

6.5.3 Potentiometer

Channel 6 of the ADC is connected to a potentiometer *(variable resistor)* at the top-left of the PCB. As the pot' is rotated clock-wise from left to right the ADC output value will **decrease** from 255 to 0.

6.5.4 Other ADC Inputs

Channel 7 of the ADC is routed to the left pin of a 2mm pitch pin-header below the channel 5 connector. Any of channel 0-5 and 7 can be used as a general purpose ADC input (8-bit, with a 2.5V reference voltage) by using the raw reading value. There is also the potential to use any of the 8 available analogue inputs on the ATMega microcontroller which offers 10-bit resolution *(see section on Arduino below)*.

6.6 I2C Devices

The I2C interface is widely used for many robotics sensors and accessories. The YRK uses a **PCA9548** I2C switch, which splits the master I2C bus into 8 individual busses, allowing the use of repeated I2C addresses across multiple ports. This allows, for example, the use of multiple I2C distance sensors which share the same I2C address, by connecting them to different busses. Most I2C devices (capable of operating in **400KHz** *fast mode*) can be attached provided they do not use the address **0x70** which is used by the switch.

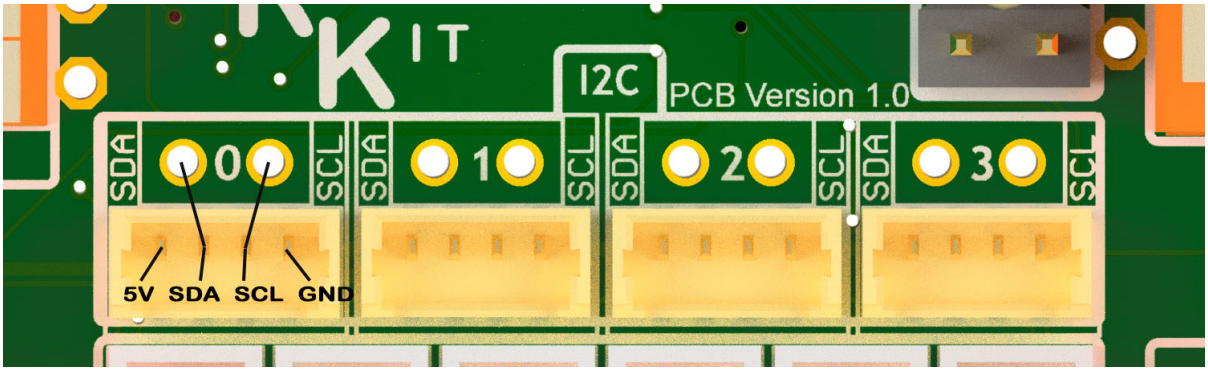


Fig. 7: Closeup of I2C (channel 0 - 3) Picoblade and 0.1" connectors

The I2C switch has a kernel-level driver, meaning that the individual switched busses appear to the user as different I2C root devices (each has its own file handle at `/dev/`i2c-XX`). The actual address of the bus is different when used on the Raspberry Pi 4 hardware and earlier versions, as the Raspberry Pi 4 hardware can support more native I2C busses (not used in the **YRK**). On the Raspberry Pi 4, the switched busses map to file descriptors `/dev/i2c-6` to `/dev/i2c-13` from channel 0 to 7, and to descriptors `/dev/i2c-3` to `/dev/i2c-10` on earlier version of the Pi.

The first four channels are unused and are intended for user additions. These SDA and SCL signals for these channels are each routed to unpopulated 0.1" pitch holes on the YRK PCB, and also to 4-pin Molex Picoblade headers. The Picoblade headers (1.25mm pitch) include a 5V and GND signal. These are directly compatible with sensor boards developed at York such as the **YRL013** multi-sensor board and the **YRL019** thermal-imaging sensor board. Note that all i2c channels except channel 4 are pulled-up to 3.3V; channel 4 is utilised by the PWM driver and the Arduino *(see below)* and is pulled-up to 5V.

6.7 Arduino

The YRK includes a **ATMega328P** microcontroller, running at **5V** and connected to both an FTDI serial to USB interface and to the I2C switch (on switch port 5, which is `/dev/i2c_11` on Pi 4). The microcontroller is effectively a clone of an Arduino Nano board (albeit with a different pin layout).

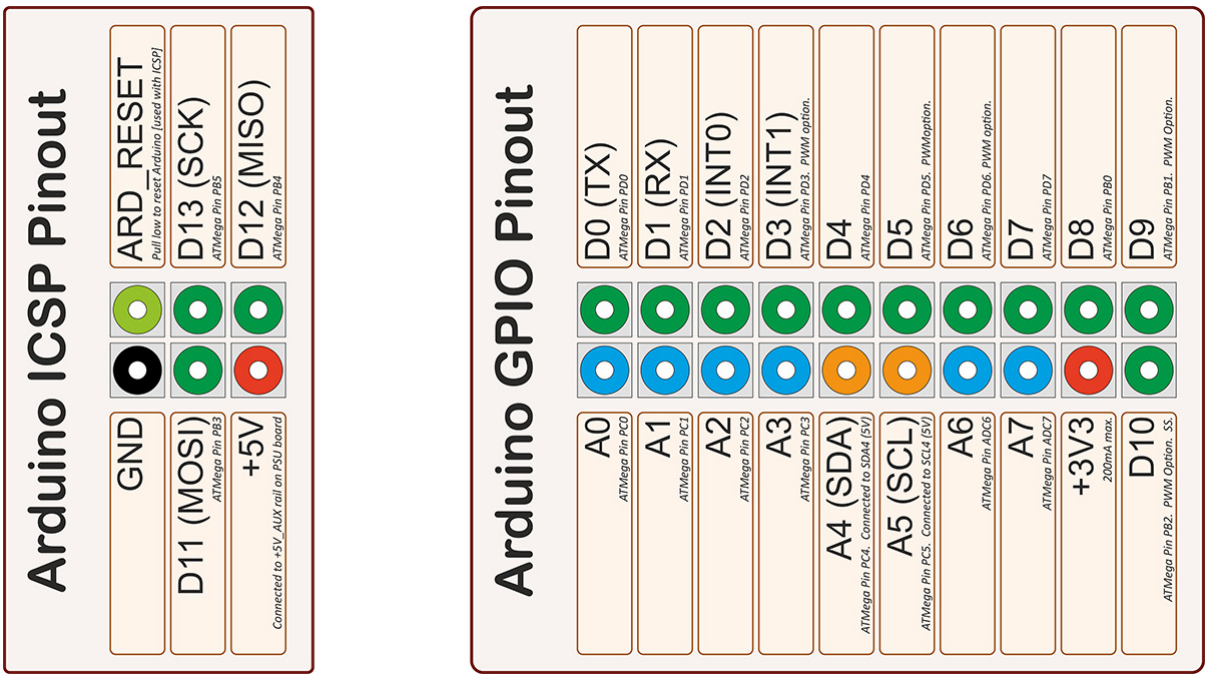


Fig. 8: Pin-out for the ATMega microcontroller *(Arduino nano clone)*

6.7.1 Programming Bootloader

Before normal use, the ATMega328P must have bootloader code uploaded to it which allows it to be programmed using the serial to USB interface. This can be done using various AVR programmers, but can also be done using a separate Arduino board and the Arduino as ISP program. The best settings in the Arduino IDE are to use `Board: Arduino Pro or Pro Mini` and `Processor: ATMega328P (5V, 16MHz)`.

6.7.2 Uploading Code

Once the bootloader has been uploaded, the Arduino can be programmed via the mini-USB port at the top-left of the board. Note that the 5V power on the mini-USB is not connected to the YRK *(this means the YRK board needs to be powered on if programming over USB). It is possible to program the Arduino from the Raspberry Pi, if a USB cable is connected from the mini-USB to the USBs on the Pi. This can be done either using Arduino IDE, or from the command line *(avoiding the need for X-windows)*.* The command line upload would look similar to this statement:

```
arduino --upload --port /dev/ttyUSB0 --board arduino:avr:pro:cpu=16MHzatmega328 --
↪verbose-upload my_code.ino
```

It is important to note that to use the mini-USB interface the slide-switch must be in its upper position. The switch directs the **TX** and **RX** serial output pins from the ATMega microcontroller to either the **FTDI** serial to USB interface *(and mini-USB port)*, if it is in its upper position, and combines the lines via a tri-state buffer for use with the digital servo port in its lower position.

6.8 Switched Outputs

The board contains a pair of **FET** driven switched outputs which can be used when it is necessary to turn on simple switched loads. Typical uses might be powering buzzers and sirens, LED light fittings and lamps, beacons, solenoids and relays. One output is connected to the **5V_AUX** supply, the other is marked as **12V** and is connected to the battery or DC input. Both switched outputs are protected by a 1A 0603 quick-blow fuse. The outputs are connected to 0.1" sockets *(preferred over header as harder to short-circuit)*.

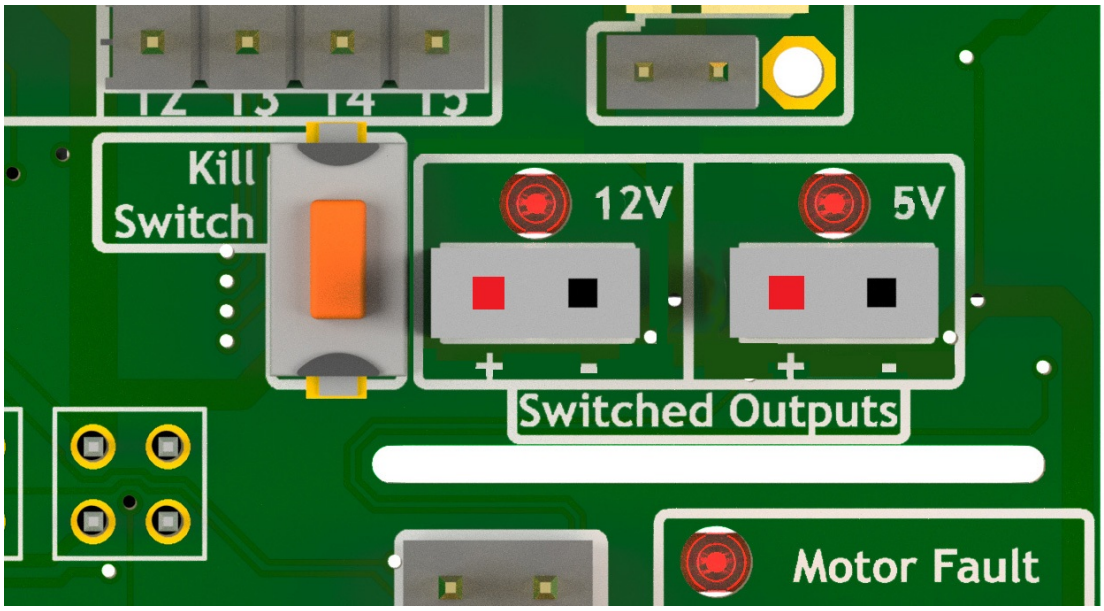


Fig. 9: Close-up view of 12V and 5V switched output connectors.

It is important to note that the switched outputs use low-side switching, meaning that the **+** output is connected directly to the *(5V or battery)* supply rail but the **-** is **not** connected to ground; never use the switched outputs on loads that require the grounds to be coupled together. It is recommended to limit the current on the switched outputs to below 500mA if possible. If a higher current (or circuit with coupled ground) is needed, consider using the switched load to drive a relay or solid-state equivalent. Note that the actual potential difference will be a little lower than the indicated amount due to the voltage drop across the **FET**. Consider using a flyback diode across inductive loads (such as relays and solenoids).

6.9 Raspberry Pi Interfaces

One consequence of the number of hardware features on the board is that very few Raspberry Pi GPIO pins are available for use. The 5 pins that are available (pins 19, 21, 23, 24 and 26) are the pins that can be used as the SPI0 interface on the Raspberry Pi, allowing SPI peripherals and expansion to be added to the YRK. These pins can also be used a general purpose IO pins if the SPI interface is not required.

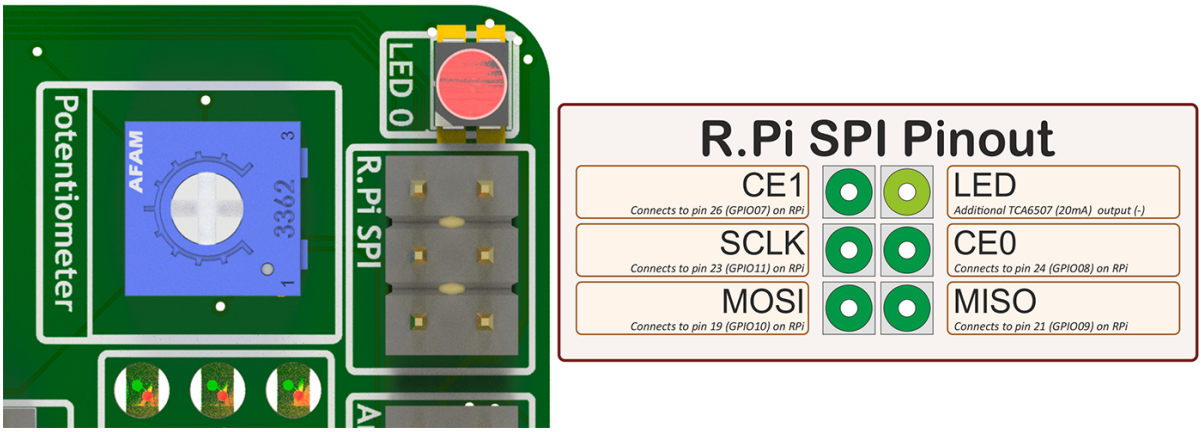


Fig. 10: Close-up view of Raspberry Pi SPI interface at top-left of board (rotated 90 degrees)

6.10 Additional GPIO

A bank of 8 user-GPIO pins connected to the bank 0 of the *(U13)* **PCA9555PW** GPIO expansion IC is available for use. The API for the pins is not yet written.

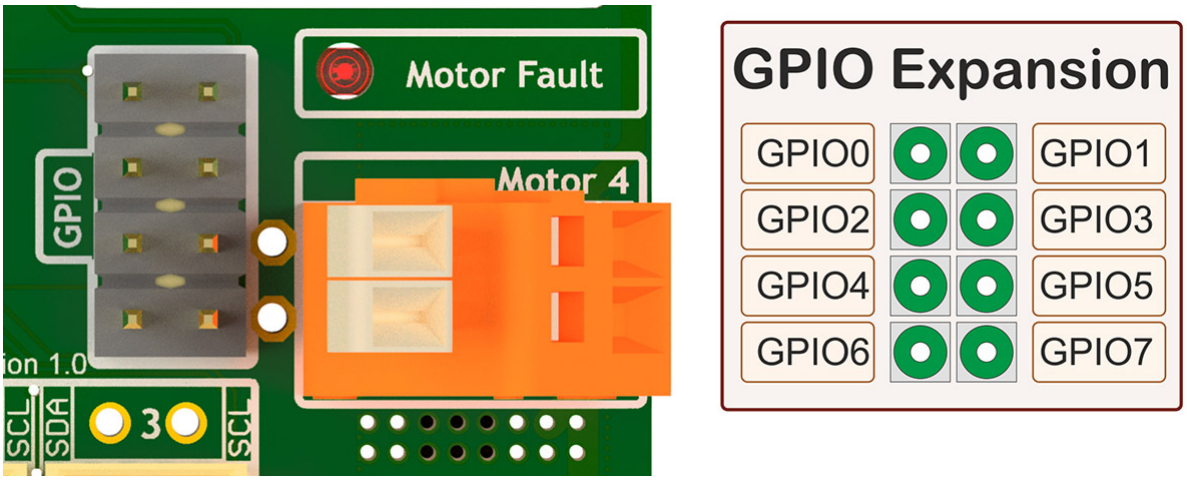


Fig. 11: Close-up view of 8 user GPIO expansion pins

There are several other expansion pins on the board that can be used as general purpose digital IO pins, for connecting extra hardware such as switches, LEDs, transistor switches and others. The **TCA6507** LED driver that drives the RGB LEDs has one additional output that is configured to give a 20mA drive current to an external LED *(or multiple LEDs in series or parallel)*. The cathode pin of the LED(s) should be attached to the LED pin of the `R.Pi SPI` header at the top-left of the PCB; the anode can be connected to either a 3.3V or 5V pin as needed *(blue and white LEDs may require 5V due to their greater forward voltage)*. The PWM (analogue servo) outputs can also be used to drive LEDs or other outputs if appropriate *(the PCA9685 driver is actually marketed as a LED driver)*.

6.11 Display

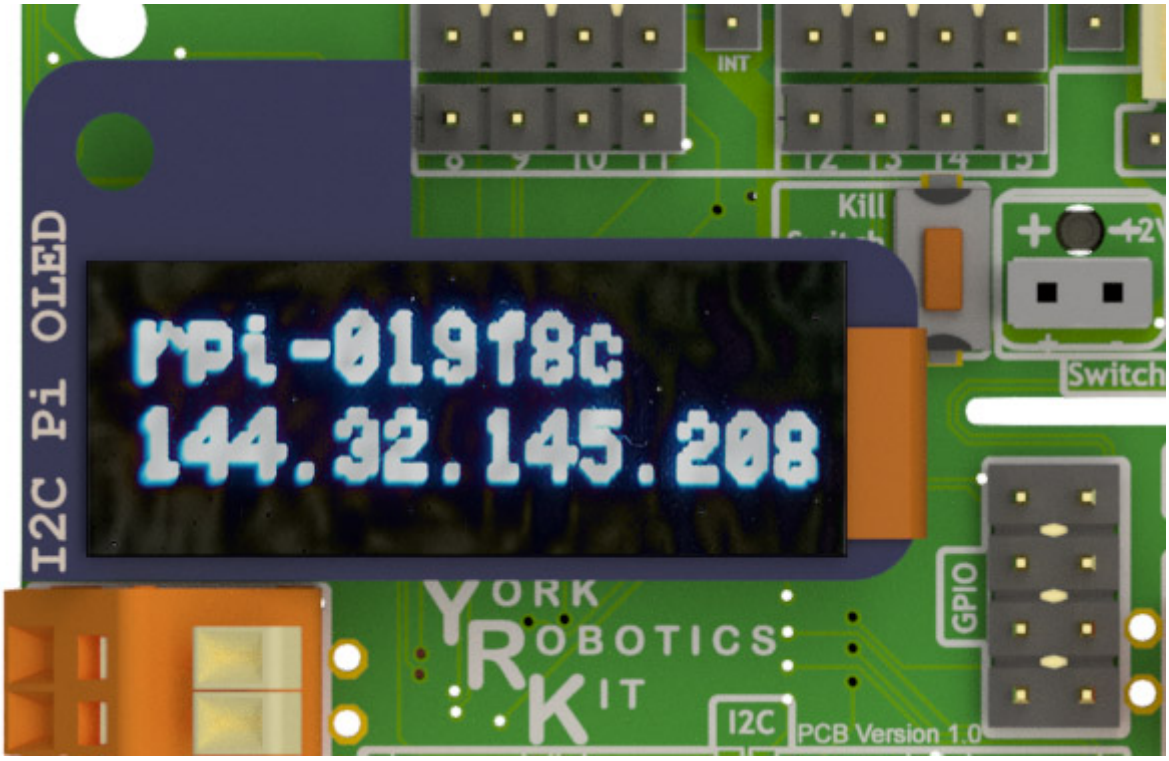


Fig. 12: Close-up view of Adafruit I2C OLED display module on YRK

The board has been designed such that an **Adafruit PiOLED** 128x32 pixel display module can be directly connected to the main board. The software library written by Adafruit has been adapted so that it performs more reliably on the switched I2C bus *(note that I2C is a relatively slow bus and even small displays take quite a lot of data to drive, so infrequent updates are recommended)*.

Obviously it may be desirable to relocate the display elsewhere on a robot chassis if the YRK and Pi are enclosed within; this simply requires the use of either 4 jumper leads or ideally a **3x2x0.1"** IDC patch cable from the header on the **YRK** to the receptacle on the display PCB.

It should be possible to use different I2C (and also SPI) displays but some alteration of code may be necessary to handle the i2c switch. It will not be possible to use any display modules which rely on a large number of GPIO pins on the Raspberry Pi as these are not available once the YRK is added. If a larger display is required, consider using the official Raspberry Pi touch display *(7" diagonal)* or a HDMI based solution.

6.12 Loudspeaker

The YRK includes a monoaural amplifier attached to one of the PWM outputs of the Raspberry Pi **(GPIO12)**. When correctly configured, the Raspberry Pi (using the **ALSA** audio system) can be set to play audio using its internal PWM. The **YRL040** PCB includes a Texas Instruments TPA2005 class-D audio amplifier IC, which is capable of producing up to 1.4W when using an 8-ohm speaker. It may be possible to use lower impedance speakers (down to 4-ohm) but note the amplifier is using the **5V_AUX** and would be operating out-of-specification at 4-ohms. Audio generated from PWM outputs is generally very noisy and low in fidelity but is adequate to generate simple sounds and speech synthesis.

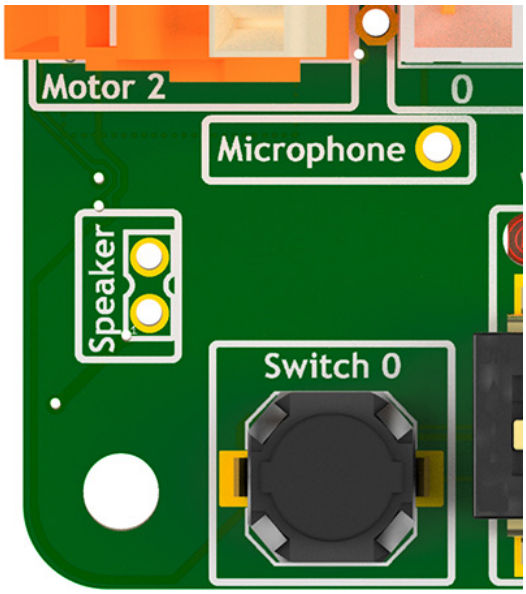


Fig. 13: Location of the speaker connection at bottom left of YRL040 PCB

The speaker output is routed to two pin holes near the bottom-left corner of the **YRL040 PCB** *(being mono the phase of the speaker doesn't matter)*. A 2-pin, 2mm pitch header can be soldered in place here to make attaching a removable speaker easier; it hasn't been done by default as the case has space for a 17mm x 11mm speaker to be connected which would be hard-wired to the underside of the *YRL040* PCB.

CHAPTER

SEVEN

SOFTWARE SETUP

This user guide covers the software setup of the **YRK**. It assumes that the micro-SD card in the Raspberry Pi is using the pre-built **Raspbian** installation (*YRK Raspbian*) created for the York Robotics Kit, that includes the **ROS Melodic**, **OpenCV 4** and the Python 3 virtual environment with all the prerequisite packages installed, along with a clone of the **Git** repository available at:

```
https://github.com/yorkrobotlab/yrk
```

The detailed software setup procedure followed to create the image is available in a different document. In the default image, the username is **pi** and the password is **robotlab**.

7.1 YRK Raspbian

This document is written for **YRK Raspbian Build 23/01/2020**. This build of Raspbian contains the following software installations:

- Raspbian Version: Buster *(Raspbian GNU/Linux 10)*. Output of `lsb_release -a`
- Kernel: Linux 4.19.75-v71+ #1270 Sep 24 2019 armv71. Output of `uname -a`
- OpenCV: 4.1.1 Output of `cv2.__version__` in Python
- ROS: Melodic Output of `rosversion -d`
- Arduino: 1.8.10 Output of `arduino --version`

The **yrk** PYthon virtual environment is preinstalled with a large number of required packages. The list of packages can be found in the `requirements.txt` file in the `/home/pi/yrk` folder, or by using the `pip freeze` command.

7.2 First Run

The image is preconfigured to work with the `robotlab` wi-fi network at York; if needed, make changes to `/etc/network/interfaces` before booting *(by editing the SD card in a different Linux system)*, or connect the Pi to a display and configure networking. Obviously in normal use the YRK is intended to be connected to remotely using SSH or VNC etc.

On first boot of a clean install of **YRK Raspbian Build 23/01/2020** it is recommended to update the system. Make sure all DIP switches are in their **OFF (down)** position. From the `/home/pi` folder execute the following script:

```
. update.sh
```

This will update Raspbian using `apt update` and `apt upgrade`, perform a Raspberry Pi firmware upgrade using `rpi-eeprom-update`, run `fixhostname.sh` to check the hostname has been update to the form **rpi-XXXX** *(where XXXX is last 4 digits of MAC address)*. It will then update to the latest codebase for **git** using `git pull`, and clean and rebuild the HTML documentation using `make clean` and `make html` from the docs folder.

Once the networking is setup and the system updated, the should be able to use the YRK in its normal operating mode. The current release defaults to booting the X-server *(even without display attached) and auto-login; it is easy and recommended to change to command-line only using the ``raspi-config`` tool if graphical user interface isn't needed. Both SSH and VNC *(remote-desktop)* are enabled by default.

The documentation *(this file!)* is built in **HTML** format in the `/home/pi/yrk/docs/_build/html/` path, using the **Sphinx** document generation system.

7.3 Boot Procedure

The image contains entries in the `.bashrc` file that set the Python virtual environment to **yrk**. It then looks to see if the device `/dev/i2c_11` exists. If the **YRK** is connected and working correctly the i2c multiplexer device tree should be running, enabling 8 extra i2c busses (named `/dev/i2c_6` to `/dev/i2c_13` on the **Pi 4** and `/dev/i2c_3` to `/dev/i2c_11` on the **Pi 3**).

If the device is found, the shell scripts `/yrk/bootscript.sh` will be launched. It is important to note that both scripts are called every time a new session is started (such as every new **ssh** connection). A *core program* is run on the first call of `bootscript.sh` after each reboot, described in detail in the next section. The *core program* writes files to an area of temporary storage at */mnt/ramdisk/*. The **DIP** switches at the bottom of the **YRK** determine the operation mode at boot-up, described in more detail in the next section.

The default working directory is:

```
/home/pi/yrk
```

To kill Python processes, erase the ramdisk and restart `bootscript.sh` quickly use the script:

```
. rerun
```

7.4 Core Program

The core program `yrk.core` performs certain core functions that aim to improve usability and reliability of robot controllers. This include monitoring battery, temperature and fault conditions, and monitoring the user switches. It also provides the functionality to control the *ROS* service, a web service and a demo program. This functionality is provided through the 4-way **DIP** switch at the bottom of the kit.

- Switch 0 (*marked as 1 on the switch itself*) determines if the core program should be run on boot. If disabled, `core.py` will not be run. This is often useful for testing but user needs to remember to keep check on battery and temperature.
- Switch 1 enables the ROS service using ROS launch. If the switch is disabled after ROS has been launched the process will be killed, allowing a relaunch.
- Switch 2 enables the web service. This enables a **Flask** webserver running a **Dash** site with **DAQ** components and this manual. By default at `localhost:8080`.
- Switch 3 enables the demo program. [To do. . .]

7.5 Basic Programming Examples

Before using the ROS infrastructure to program the **YRK** it is worth considering some very simple low-level example programs. We shall look at how to write very basic programs that access low-level features; it is recommended that the core program, normal Python services and ROS services are **not** running when programming like this. Remember that if the core program isn't running, automatic battery, temperature and fault monitoring will not be running.

The code examples in the section are all Python programs that should be run in the **yrk** virtual environment (this should be enabled by default in the image and is indicated by a `(yrk)` before the filepath in the terminal console). The examples below can be written in a text-editor, or can be entered directly into the *Thonny* Python IDE using the VNC connection. For proper code development working on the Pi over VNC isn't recommended but it can be useful for quick tests such as this.

7.5.1 Motors Example

The first example program below sets the motor connected to driver 0 *(labelled as Motor 1 on the PCB)* to 50 percent forward duty-cycle. We import the *yrk.motors* module from the **yrk** library as `motors` and call the *yrk.motors.set_motor_speed()* method:

```
import yrk.motors as motors
motors.set_motor_speed(0,0.5)
```

As the **YRK** is designed to be flexible in robot topology, the *yrk.motors* is limited to functions that affect one motor at a time, with the exception of *yrk.motors.stop_all_motors()* which sets all 4 outputs to their stopped, high-impedance state. The following code shows an example of how simple drive functions for a two-wheels, skid-steered robot, with motors connected to driver 0 and 3 *(labelled as Motor 1 and Motor 4 on the PCB)*. By adjusting the speeds and the sleep times, it should be possible to make the robot move in a square path:

```
import yrk.motors as motors
import time

def forwards(speed):
      motors.set_motor_speed(0,speed)
      motors.set_motor_speed(3,speed)

def turn(speed):
      motors.set_motor_speed(0,speed)
      motors.set_motor_speed(3,-speed)

def brake_motors():
      motors.brake_motor(0)
      motors.brake_motor(3)

for i in range(4):
      forwards(0.5)
      time.sleep(0.5)
      brake_motors()
      time.sleep(0.1)
      turn(0.5)
      time.sleep(0.5)
      brake_motors()
      time.sleep(0.1)

 motors.stop_all_motors()
```

7.5.2 ADC Example

The module *yrk.adc* contains the methods for reading the analog:digital converter. The module *yrk.led* contains methods for controlling the RGB LEDs. We can combine all three to use the on-board potentiometer *(attached to ADC channel 6)* to set the motor speed for driver 0 and set the LED brightness proportional to speed:

```
import yrk.adc as adc, yrk.motors as motors, yrk.led as led, time
while(True):
      #Read raw value of pot. 255 is fully_left, 0 is fully_right
      pot_value = adc.read_adc(6)
      #Set motor 0 speed to be fully backwards [-1.0] at pot=fully_left and
      #fully forwards [1.0] at pot=fully_right
      motors.set_motor_speed(0, 1.0 - (0.007843 * pot_value))
      #Set led brightness to be proportional to speed (range is 0-15)
      led.set_brightness((abs(128-pot_value) + 6) >> 3)
      #Make the LEDs white
      led.set_colour_solid(7)
      #Add a short wait to keep system responsive
      time.sleep(0.01)
```

Warning: This code will spin motor 0 and set the LEDs to a high brightness unless the potentiometer is very close to its central position.

CHAPTER

EIGHT

API MODULES

The core API python modules are found in the `yrk` subfolder.

8.1 yrk.adc

This module provides functions for reading from the ADS7830 8-bit Analog:Digital Converter. 8-bit raw values can be read using the `read_adc` function and these values can be converted to distance measurements for various Sharp IR distance sensors based on look-up table values using the `get_model_value` function.

The ADC has 8 input channels. Channels 0-5 are broken out on the PCB to 3-pin JST-PH headers. +5V[aux] is the uppermost pin, GND in middle and V_sense at the bottom. The ADC is set to range from 0V [0] to 2.50V [255], approx 0.01V/scale. Using the `2y0a21` or `2y0a41` sensor models for the

GP2Y0A21YK0F distance sensor [10 - 80cm] https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf

GP2Y0A41SK0F distance sensor [4 - 30cm] https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a41sk_e.pdf

ADC channel six is connected to the potentiometer at the top-left of the board. This will produce a raw output of 255-0 as it is turned clockwise. Using the `inv_pct` sensor model will convert this from 0 - 100.0 %. ADC channel 7 is connected to the left auxiliary ADC pin, GND to the right. These pins are 2mm pitch [jumpers leads are available to convert to 0.1" pitch].

`yrk.adc.`**`get_model_value`**(*raw_value*, *sensor_model*)

A function that converts a raw ADC value to distance for Sharp IR sensor

Args: raw_value (int): The 8-bit raw ADC value [eg from read_adc()] sensor_model (str): The sensor model to use. '2y0a21' or '2y0a41' are valid sensors; other options are 'voltage','raw' and 'inv_pct'

Returns: float: Converted distance in mm based on look-up tables

`yrk.adc.`**`read_adc`**(*channel*)

A function that reads the raw 8-bit value [8-bit] from given channel of ADC

Args: channel (int): The ADC channel to read from [0-7]. 6 is potentiometer on YRL040.

Returns: int: ADC value [range 0-255]

8.2 yrk.audio

The YRL040 PCB contains a TPA2005 class-D 1.4W mono audio amplifier. The audio output from the Raspberry Pi is sent as a PWM signal over GPIO pin 12 and a logic-high on GPIO pin 16 enables the amplifier. PWM audio is relatively poor in audio quality and inherently noisy, so it is generally preferable to disable the output when audio is not being played.

The audio module is designed to run continuously and uses a seperate execution thread to process a queue of stored audio commands. The audio setup is started by calling the `setup_audio` function. To add a sound to the playback queue either `play_audio_file`, which plays a .mp3 or .wav file, or `say`, which uses the `espeak` program to read a message.

TPA2005 Data Sheet: https://www.ti.com/lit/ds/symlink/tpa2005d1.pdf

yrk.audio.**audio_queue_thread**()
Thread loop for handling queued audio files

yrk.audio.**kill_audio**()
Function to kill all running audio processes

yrk.audio.**mute**()
This function mutes the audio output

yrk.audio.**play_audio_file**(*file*)
Function to add an audio file to the audio Queue

Args: file (str): The filename for the [.mp3] or [.wav] audio file

yrk.audio.**say**(*message*)
Function to add a spoken message to the audio Queue

Args: message (str): The message to read out [using e-speak]

yrk.audio.**say_ip**()
This function will speak IP address using hostname subprocess and stored audio files

yrk.audio.**set_volume**(*volume*)
A function that sets the [PWM] volume output

Args: volume (int): The percentage volume [0-100]

yrk.audio.**setup_audio**()
Function to setup the audio system: sets volume, mutes output, starts thread

yrk.audio.**start_audio_thread**()
Function to start the audio thread

yrk.audio.**unmute**()
This function unmutes the audio output.

Note that audio out uses PWM GPIO which is inherently noisy, CPU activity noise should be expected when audio is unmuted. Where possible, use of the play_audio_file() function and queue system is recommended as this mutes audio when the queue is empty

8.3 yrk.core

8.4 yrk.display

There is a header on the YRL040 PCB to which an Adafruit PiOLED 128x32 pixel display can be attached, either directly or via a ribbon cable. This module augments the functionality of the `Adafruit_SSD1306` library, providing functions to display graphics and text on the display. Bus 5 of the I2C switch is routed (exclusively) to the display header.

The init_display() function should be called once before drawing functions are used to initialise the display (this is done by core.py if being used).

`yrk.display.`**`clear`**`()`
A function to clear the display

`yrk.display.`**`display_image`**`(`*image*`)`
Function to display a 128x32 pixel PIL image on the display

The display functions use the Python Image Library (Pillow) to store the image bitmaps that are to be displayed. All other drawing functions generate a PIL image then call this function to display the generated image.

Args: image (PIL image): The 128x32 PIL image

`yrk.display.`**`display_image_file`**`(`*image_filename*`)`
Function to display a 128x32 pixel monochrome PBM image from file on the display

PBM are portable bit-map files; the file size should be 522 bytes (10 bytes header + 512 bytes data). Files can be generated using bitmap editing graphics applications such as Gimp or Adobe Photoshop.

Args: image_filename (str): The filename for the 128x32 PBM image

`yrk.display.`**`display_stats`**`()`
Function to display current system stats (IP, cpu load, temperatures, memory usage)

`yrk.display.`**`get_ip`**`()`
Function to get IP address as a string using hostname system process

`yrk.display.`**`init_display`**`()` → bool
A function to initialise and clear the display

`yrk.display.`**`one_line_text`**`(`*text*`)`
Function to display one line of 32-pixel high text on display [no wrapping]

Args: text (str): The message to display [8-chars max to be displayed]

`yrk.display.`**`one_line_text_wrapped`**`(`*text*`)`
Function to display one message of text on display

32-pixel high [1-line] font will be used unless message > 14 characters long, in which case either 16-pixel or 8-pixel high fonts will be used.

Args: text (str): The message to display [64-chars max to be displayed]

`yrk.display.`**`two_line_text`**`(`*line1_text*, *line2_text*`)`
Function to display two lines of text on display in 16-pixel high font [no wrapping]

Args: line1_text (str): The first line of text to display [14-chars max to be displayed] line2_text (str): The first line of text to display [14-chars max to be displayed]

`yrk.display.`**`two_line_text_wrapped`**`(`*line1_text*, *line2_text*`)`
Function to display two lines of text on display

Where lengths are < 15 characters, 16-pixel font used, else 8-pixel font used

Args: line1_text (str): The first line of text to display [32-chars max to be displayed] line2_text (str): The first line of text to display [32-chars max to be displayed]

yrk.display.**warning**(*text*)
Function to display a warning graphics alongside a text message on display

Args: text (str): The text message to display (limited to about 8 characters)

8.5 yrk.gpio

The YRL040 PCB contains two PCA9555 16-bit GPIO expanders, one is connected to the switches [see `switch` module], the other provides 8 user GPIO pins on a 0.1" pitch 4x2 header. It also uses 4 GPIO input to detect fault conditions in the four H-bridge motor drivers, and 1 more to provide the motor fault LED output. Another input is used to provide the "kill switch" input, which might be configured to stop all running code etc.

The final pair of GPIO outputs are used to provide an NMOS-driven 5V and 12V switched output. These outputs can be used to turn on a DC load [fused at 1A], but note the negative terminal is not a common ground with the main PCB and should not be connected. This makes the output most suited to drive things such as a buzzer or lamp, or potentially a relay. The 5V output is connected to the 5V_AUX supply [though will be slightly under 5V due to the FET losses]; the 12V is connected to V_IN.

PCA5555 GPIO Expander Datasheet: https://www.nxp.com/docs/en/data-sheet/PCA9555.pdf

yrk.gpio.**read_user_gpio**()
A function to read the input registers of the user GPIO Expander

Returns: int: 16-bit value indicating user GPIO states.

yrk.gpio.**set_motor_fault_led**(*state*)
A function to enable or disable the motor fault LED

Args: state (bool): Enable or disable the motor fault LED

yrk.gpio.**set_switched_output_12V**(*state*)
A function to enable the 12V switched output

Note the PD will actually be a little under Vin. V- is not GND and should not be connected to ground.

Args: state (bool): Enable or disable the 12V output

yrk.gpio.**set_switched_output_5V**(*state*)
A function to enable the 5V switched output

Note the PD will actually be a little under 5V. V- is not GND and should not be connected to ground.

Args: state (bool): Enable or disable the 5V output

yrk.gpio.**setup_user_gpio**()
An initialisation function for the PCA9555 GPIO Expander for the user GPIO and motor fault detection

Sets pins IO0_0 : IO0_7 based on values from settings.py Sets pins IO1_0 : IO1_3 as inverted inputs for motor fault detection Sets pins IO1_4 : IO1_6 as outputs [off] for switched outputs and motor fault LED Sets pin IO1_7 as inverted input for kill switch

yrk.gpio.**update_switched_gpio_outputs**()
Sends i2c command to set the switched outputs based on stored variables

8.6 yrk.led

The YRL040 PCB contains a TCA6507 7-way I2C LED driver. This IC is connected to the two RGB LEDs at either corner of the top of the PCB, plus one extra output is routed to the top-left expansion pin at the top of the PCB.

The TCA6507 allows 2 programmable channels of PWM output, allowing blinks and pulse effects to be programmed using a single I2C message. However, as this is limited to 2 channels, full independent RGB control of the LEDs is not possible. To simplify use, this module contains a number of functions to allow programmed single colours and animated effects to be shown on the LEDs.

TCA6507 Data Sheet: https://www.ti.com/lit/ds/symlink/tca6507.pdf

`yrk.led.`**`animation`**(*index*)
Displays the given animation from `body_animations` list

Args: index (int): The `body_animations` entry to use [range 0-8]

`yrk.led.`**`set_brightness`**(*brightness_int*)
Sets stored target brightness value to given value

NB Brightness is only changed on next call to set led

Args: brightness_int (int): The target brightness [range 0-15]

`yrk.led.`**`set_colour_pulse`**(*index*, *speed=4*)
Sets both LEDs to pulse, at given speed and colour

Args: index (int): The solid_colours entry to use [range 0-8] speed (int): The speed of the pulse [range 0-15]

`yrk.led.`**`set_colour_solid`**(*index*)
Sets both LEDs to solid colour

Args: index (int): The solid_colours entry to use [range 0-8]

`yrk.led.`**`set_left_colour_pulse`**(*index*, *speed=4*)
Sets the left LED to pulse, at given speed and colour

Args: index (int): The solid_colours entry to use [range 0-8] speed (int): The speed of the pulse [range 0-15]

`yrk.led.`**`set_left_colour_solid`**(*index*)
Sets the left LED to a solid colour

Args: index (int): The solid_colours entry to use [range 0-8]

`yrk.led.`**`set_right_colour_pulse`**(*index*, *speed=4*)
Sets the right LED to pulse, at given speed and colour

Args: index (int): The solid_colours entry to use [range 0-8] speed (int): The speed of the pulse [range 0-15]

`yrk.led.`**`set_right_colour_solid`**(*index*)
Sets the right LED to a solid colour

Args: index (int): The solid_colours entry to use [range 0-8]

`yrk.led.`**`stop_animation`**()
Stops the animation [by calling `animation(0)`]

`yrk.led.`**`timed_animation`**(*index*, *time*)
Starts a `Timer` thread to run animation for given duration then stop

Use carefully as no checking is done in the case of repeated calls

Args: index (int): The body_animations entry to use [range 0-8] time (float): The duration in seconds before calling `stop_animation`

8.7 yrk.motors

The YRL040 PCB contains 4 DRV8830 H-Bridge motor drivers, powered from the 5V_AUX supply. Each driver has an 800mA current limit. It is primarily intended for use with small brushed DC motors such as the 12mm micro-metal gear motors available from a number of suppliers including MFA Como, Pimoroni and Pololu. Motors are connected using the push-level Wago terminals at either side of the YRL040 PCB.

The motor driver allows the effective motor voltage [in either direction] to be programmed using an I2C message. It also allows the motor to be put in a coast [*high-impendance*] or brake [*short-circuit*] state. The ICs contain fault [*over-current* and *over-temperature*] conditions to trigger a logic low output, which is connected the GPIO expander [see `gpio`].

DRV8830 Data Sheet: https://www.ti.com/lit/ds/symlink/drv8830.pdf

`yrk.motors.`**`brake_motor`**(*motor*)
Turns on brake mode for given motor

Brake state effectively short-circuits the motor windings, resisting motion. This is different to coast state [*high-impendance*] which is set by calling `set_motor_speed(motor,0)`

Args: motor (int): The motor driver [*range 0-3*]

`yrk.motors.`**`get_brake_state`**(*motor*)
Gets current brake state of the given motor driver

Args: motor (int): The motor driver [*range 0-3*]

Returns: bool: True if the motor is in brake [*short-circuit*] state

`yrk.motors.`**`get_motor_speed`**(*motor*)
Gets current speed of the given motor driver

Args: motor (int): The motor driver [*range 0-3*]

Returns: float: The target speed of the motor [*range -1.0 to 1.0*]

`yrk.motors.`**`set_motor_speed`**(*motor*, *speed*)
Sets the motor to given speed

Args: motor (int): The motor driver [*range 0-3*] speed (float): The target speed for the motor [*range -1.0 to 1.0*]

`yrk.motors.`**`stop_all_motors`**()
Sets all 4 motor drivers to coast state

8.8 yrk.power

The YRL039 PCB provides the main power supplies for the YRK, using a pair of 3A, 5V buck [*step-down*] converters. These power supplies are controlled by a ATMega328 microcontroller, which also monitors the voltage of the supply plus the voltages and currents of both 5V outputs. The ATMega also monitors the temperature of the PCB and controls a small fan, positioned directly above the CPU of the Raspberry Pi.

This module provides the functions to read the data values from the ATMega microcontroller [using the I2C bus].

`yrk.power.`**`get_aux_current`**()
Returns 5V_AUX current (float) stored on last call of `read_all_values`

`yrk.power.`**`get_aux_voltage`**()
Returns 5V_AUX voltage (float) stored on last call of `read_all_values`

`yrk.power.`**`get_battery_voltage`**()
Returns battery voltage (float) stored on last call of `read_all_values`

yrk.power.**get_pcb_temperature**()
Returns PCB temperature (float) stored on last call of `read_all_values`

yrk.power.**get_pi_current**()
Returns 5V_PI current (float) stored on last call of `read_all_values`

yrk.power.**get_pi_voltage**()
Returns 5V_PI voltage (float) stored on last call of `read_all_values`

yrk.power.**read_all_values**()
Reads and stores all voltage, current and temperature readings in a single i2c request

8.9 yrk.pwm

This module provides functions for controlling the PCA9685 16-channel 12-bit PWM driver, which is primarily intended for use with analog servo motors.

This module has been quickly put relatively quickly to provide functionality for [exclusively for] use with analog servos. If wanted for other purposes it would be worth investigating the existing Adafruit library, which is more polished, but requires circuitpython amongst other libraries and might take a bit of adaptation to work using the switched i2c bus.

PCA9685 Datasheet https://www.nxp.com/docs/en/data-sheet/PCA9685.pdf

yrk.pwm.**calculate_nearest_duty_cycle_to_period**(*period_seconds: float*) → int
Calculate the value for raw cycle for a given period

Args: period_seconds (float): The target on-time in seconds.

Returns: int: The raw duty cycle value [*for use with set_duty_cycle_raw*]

yrk.pwm.**estimate_on_time**(*dutycycle_raw: int*) → float
Estimate the on-time based on the raw duty cycle value

Args: dutycycle_raw (int): The target duty-cycle [*range 0 - 4095*]

Returns: float: The approximate on-time value in seconds

yrk.pwm.**set_duty_cycle**(*output: int*, *dutycycle_pct: float*)
Sets the duty cycle (on period) of a PWM output as a percentage

Args: output (int): The servo output to use [*range 0-15*] dutycycle_pct (float): The percentage [*0-100*] of on-time

yrk.pwm.**set_duty_cycle_raw**(*output: int*, *dutycycle_raw: int*)
Sets the raw on-period value for a given PWM output

Args: output (int): The servo output to use [*range 0-15*] dutycycle_raw (int): The on-time period [*range 0-4095*]

yrk.pwm.**set_normal_mode**()
Disables sleep mode on PWM driver

yrk.pwm.**set_prescale_value**(*psv: int*)
Sets the prescale register to set the PWM frequency

PWM frequency approximately equal to `6104 / (psv + 1)`

Args: psv (int): The target prescale register value [*range 3-255*]

yrk.pwm.**set_pwm_frequency**(*freq: float*)
Sets the PWM frequency

The PWM will be set as close as it can be to the requested frequency. It calculates the closest `prescale` value and calls `set_prescale_value` with this value. All outputs have the same frequency.

Args: freq (float): The target frequency in hertz [*effective range 24 - 1526*]

yrk.pwm.**set_sleep_mode**()
Enables sleep mode on PWM driver

8.10 yrk.settings

yrk.settings.**ADC_MODELS = ['voltage', 'voltage', 'voltage', 'voltage', 'voltage', 'voltage**
List of model types for the 8 ADC inputs

Sensor type in this list. Should have 8 entries. Note 7th is for potentiometer. Valid models are voltage, pct, inv_pct, raw, 2y0a21 and 2Y0a41

Eg: ADC_MODELS = ['2y0a21','2y0a41','voltage','raw','raw','raw','inv_pct','raw']

yrk.settings.**AUDIO_VOLUME = 100**
Volume to set Alsa mixer to on setup of audio (range 0-100, but 80+ recommended)

yrk.settings.**BATTERY_CELLS = 2**
Number of Li-Ion or Li-Po cells in battery (2,3 or 4)

Sets the BATTERY_LOW_VOLTAGE, BATTERY_CRITICAL_VOLTAGE and BATTERY_SHUTDOWN_VOLTAGE parameters based on the number of cells described. The values for 2 cell batteries are a little more generous to get decent usable life of of battery, although voltage drop may be too great in high current drain use cases. If using other battery technology (eg Ni-Mh) set values manually within settings.py

yrk.settings.**BATTERY_CHECK_PERIOD = 2.0**
Period (s) between battery state checks

yrk.settings.**BATTERY_CRITICAL_SHUTDOWN = True**
If enabled, system will force shutdown when Vbatt<BATTERY_SHUTDOWN_VOLTAGE

yrk.settings.**CONSOLE_LOGGING_MODE = 20**
Set the Python logging level for console output.

The recommend setting is logging.INFO for deployment and logging.DEBUG for debugging. Cannot be at a lower level than FILE_LOGGING_MODE.

yrk.settings.**CPU_CRITICAL_TEMP = 75**
CPU critical temperature, recommend ~75C for Pi 4 and ~65C for Pi 3

yrk.settings.**CPU_SHUTDOWN_TEMP = 82**
CPU shutdown temperature, recommend ~82C for Pi 4 and ~72C for Pi 3

yrk.settings.**CPU_WARNING_TEMP = 65**
CPU warning temperature, recommend ~65C for Pi 4 and ~60C for Pi 3

yrk.settings.**DISPLAY_ROTATED = False**
Set to True if the OLED module is rotated to flip image

yrk.settings.**ENABLE_BATTERY_MONITOR = True**
If enabled yrk-core.py will display visual+audible warnings when battery low

yrk.settings.**ENABLE_TEMPERATURE_MONITOR = True**
If enabled yrk-core.py will display visual+audible warnings when cpupcb temperature high

yrk.settings.**FILE_LOGGING_MODE = 10**
Set the Python logging level for saved log files.

The recommend setting is logging.INFO for deployment and logging.DEBUG for debugging

yrk.settings.**HAS_DISPLAY = False**
Set to True is OLED module is being used

yrk.settings.**I2C_5V_BUS = 11**
The PWM driver and the Arduino are on the 5V I2C Bus [bus 4 on switch]

yrk.settings.**OLED_BUS = 12**
The /dev/i2c_XX bus which the OLED module is attached to

yrk.settings.**PCB_CRITICAL_TEMP = 45**
PCB critical temperature, recommend ~45C

yrk.settings.**PCB_SHUTDOWN_TEMP = 50**
PCB forced shutdown temperature, recommend ~50C

yrk.settings.**PCB_WARNING_TEMP = 40**
PCB warning temperature, recommend ~40C

yrk.settings.**PWM_FREQUENCY = 50**
PWM (Analog Servo) target frequency in hertz

yrk.settings.**RASPBERRY_PI_MODEL = 4**
Sets the Raspberry Pi version being used

Set to 4 if using Raspberry Pi 4 (recommended) If set to different value, default BUS values are overwritten with values compatible with the earlier Pis.

yrk.settings.**TEMPERATURE_CHECK_PERIOD = 2.0**
Period (s) between temperature checks

yrk.settings.**TEMPERATURE_CRITICAL_SHUTDOWN = True**
If enabled, system will force shutdown when CPU Temp<CPU_SHUTDOWN_TEMP or PCB Temp<PCB_SHUTDOWN_TEMP

yrk.settings.**USER_GPIO_INVERTED = 0**
Initialisation inversion state for the 8 user GPIO pins [1=Invert input]

yrk.settings.**USER_GPIO_MODE = 255**
Initialisation mode for the 8 user GPIO pins [1=Input [with pull-up], 0=Output].

yrk.settings.**USER_GPIO_OUTPUT_STATE = 0**
Initialisation output state for the 8 user GPIO pins

yrk.settings.**USE_DIP_FUNCTIONS = True**
If True, yrk-core uses DIP 2 for ROS, DIP 3 for DASH server and DIP 4 for DEMO

yrk.settings.**USE_DIP_LEDS = True**
If true, yrk-core will set DIP LEDs based on program state

yrk.settings.**YRL039_ADDRESS = 57**
The I2C address for the ATMega on the YRL039 Power Supply Board (could be reprogrammed to different address if needed).

yrk.settings.**YRL039_BUS = 14**
The /dev/i2c_XX bus the YRL039 Power Supply board's ATMega is attached to

yrk.settings.**YRL040_BUS = 13**
The /dev/i2c_XX bus the YRL040 [3.3V] I2C devoices are attached to

8.11 yrk.switch

This module contains function for the switch GPIO expander [one of two PCA9555 GPIO expanders on the YRL040 PCB]. The switch module, at the bottom of the YRL040 PCB, contains a 4-way DIP switch with associated LEDs, a 5-way directional switch and 2 push-buttons at either side. The green power LED is also connected to the GPIO expander.

PCA5555 GPIO Expander Datasheet: https://www.nxp.com/docs/en/data-sheet/PCA9555.pdf

yrk.switch.**read_dip_switch**()
A function to read the state of the 4-way DIP switch

Returns: int: 4-bit value indicating switch states

yrk.switch.**read_input_registers**()
A function to read the state of the switches at the bottom of the YRL040 PCB

Returns: int: 11-bit value indicating switch states.

yrk.switch.**set_dip_leds**(*nibble*)
A function to set the state of the yellow LEDs above the DIP switch

Args: nibble (int): A four-bit value indicating the target state of the LEDs

yrk.switch.**set_power_green_led**(*state*)
A function to set the state of the green power LED at top of YRL040 PCB

Args: state (bool): Enable or disable the LED

yrk.switch.**setup_switch_gpio**()
An initialisation function for the PCA9555 GPIO Expander controlling the switches

Sets pins IO0_0 : IO0_7 and IO1_0 : IO1_2 as inverted inputs [ie 0V = True] Sets pins IO1_3 : IO1_7 as outputs. IO1_3 : IO1_6 are DIP LEDs, IO1_7 is green power LED

yrk.switch.**update_switch_gpio_output**(*new_states*)
A function to update the output pins on the PCA9555 GPIO Expander

Args: new_states (int): A 1-byte value [MSB=1_7, Green LED].

8.12 yrk.utils

CHAPTER

NINE

EXAMPLE PROGRAMS

Some simple example code programs and utilities are found in the `examples` subfolder.

9.1 examples.console

A terminal-based console that provides feedback on sensor information and allows keyboard based control of actuators. The console.py example uses the **curses** library to allow console over-writing. It uses line-drawing characters, which may not render correctly on default settings on Windows, *kitty* (a fork of *putty*) has an `Allow ACS line drawing in UTF` setting which allows the correct rendering.

It provides a useful quick test for hardware and also a useful example of how to do many low-level API calls. The console should be run without any other code (include `core.py`) running. It can be run as follows:

```
cd ~/yrk/examples
python console.py
```

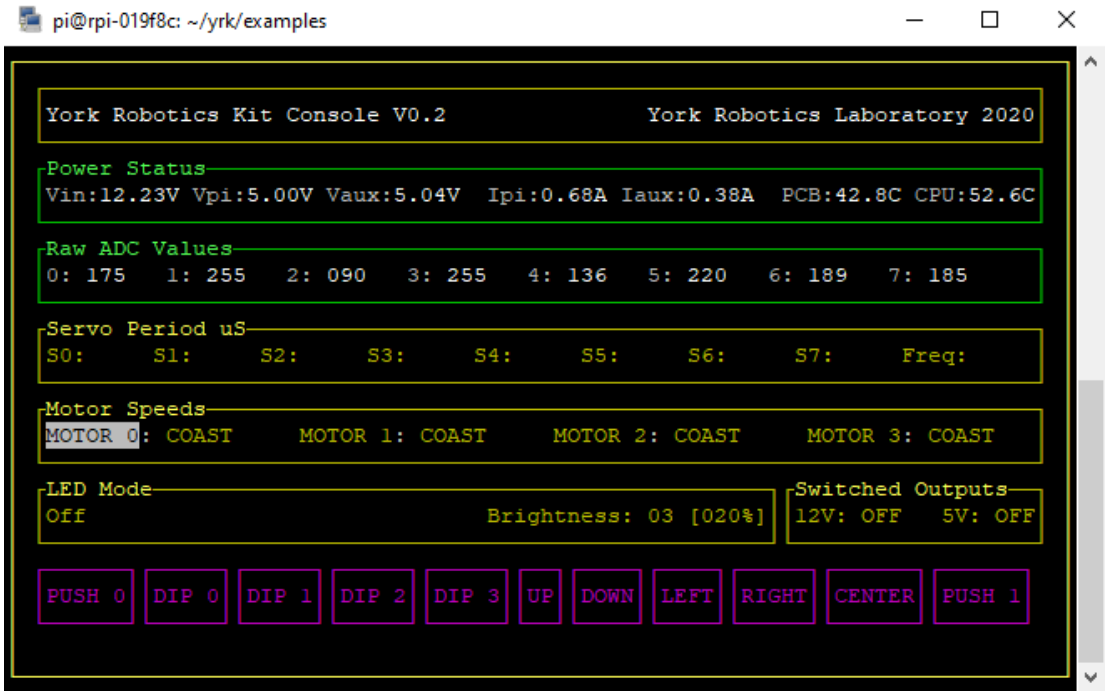


Fig. 1: Screen shot of `console.py`

The cursor keys on the keyboard can be used to move between the different motor, LED and servo options (*the console sets the PWM driver up for 1.5mS analogue servos*). As the console can enable and motors and servos it should be used with caution.

9.2 examples.potmotor

An simple script showing how to set the speed of motor 0, and the brightness of the LEDs, proportional to the potentiometer setting. Attach a motor to motor port 0 and set pot roughly to middle position before running!

To run:

```
python potmotor.py
```

Use `Ctrl-C` to exit, then run:

```
python stop.py
```

to reset motors and leds.

9.3 examples.potservo

An simple script showing how to set PWM duty cycle (on-time) of one of the servo outputs, proportional to the potentiometer setting. Attach a small analog servo to port 0. The on-time will be vary between *1.0 mS* and *2.0 mS* as the potentiometer is rotated, which is a typical range for most analog servos. The PWM frequency is set to about 50Hz, again typical for most servo motors.

To run:

```
python potservo.py
```

Use `Ctrl-C` to exit

9.4 examples.stop

An example script showing how to stop and reset everything. The following sequence takes place:

- Call `motors.stop_all_motors()` to stop motors
- Call `led.set_colour_solid(0)` to turn off RGB LEDs
- Call `display.clear()` to clear OLED display

To run:

```
python stop.py
```

`examples.stop.`**`stop_all`**`()`
Stops and resets all actuators

CHAPTER

TEN

YRK ROS API

The ROS script, message and service files are found in the catkin workspace subfolder `catwin_ws/src/yrk_ros`.

This can be quickly accessed from a terminal using the following ROS command:

```
roscd yrk_ros
```

10.1 yrk_ros.adc_publisher

Reads the values from the 8-port ADC and publish as a rostopic.

To run:

```
rosrun yrk_ros adc_publisher.py
```

10.2 yrk_ros.button_publisher

Reads the values from the buttons and switches and publishes as a rostopic.

To run:

```
rosrun yrk_ros button_publisher.py
```

10.3 yrk_ros.display_server

The `display_server.py` script contains the code for the *display_service* ROS service in the *yrk-ros* library. This provides the functionality to display text, warnings and to clear the I2C OLED module, using functions from the *yrk.display* core module.

The service message is defined in the file *Display.srv* and contains 5 input fields, shown in the table below, and one response field (*bool* **success**).

Type	Name	Notes
string	message_1	
string	message_2	Only used when *number_of_lines==2* and *warning==false*
unit8	number_of_lines	Range 0-2, 0=clear display
bool	wrap_text	Shrinks font, wraps text over 2 lines if needed
bool	warning	Displays ~10 chars text [message_1] with warning icon

The Display service message is processed using the following logic:

- If *number_of_lines*>2 exit with failure
- If *number_of_lines* =0 clear display
- If *warning* is true, display *message_1* as a warning (ignore *number_of_lines*, *wrap_text*, *message_2*)
- Otherwise call 1- or 2- line wrapped or normal functions based on settings

To run outside of launch file:

```
rosrun yrk_ros adc_publisher.py
```

Usage Example:

```
rosservice call /display_service "{message_1: '', message_2: '', number_of_lines: 0,␣
↪wrap_text: false, warning: false}"
```

Note: Use tab completion on rosservice call */display_service* to prepare above macro, works elsewhere too!

10.4 yrk_ros.led_server

Controls the colour, mode and brightness of the RGB LEDs on the YRK. The message expects 3 arguments: uint8 index : Colour index uint8 brightness: Brightness value [0-15]

Output is a 'name' string describing LED colour or animation mode

To run:

```
rosrun yrk_ros led_server.py
```

10.5 yrk_ros.motor_server

Sets the speed and brake mode [highlow impedance] for the motors on the York Robotics Kit. The message expects 3 arguments: uint8 motor_index : Motor index [0-3] float32 speed: Motor speed [-1.0 to 1.0, 0=brake or coast] bool brake_mode: If speed=0.0 and brake_mode=true, will force low-impedance brake

Returns: bool success

Output is a 'name' string describing LED colour or animation mode

To run:

```
rosrun yrk_ros led_server.py
```

10.6 yrk_ros.power_monitor

Reads the values from the Atmega328 microcontroller on the YRL039 power supply board and publishes as a ROS topic using the yrk_ros.msg.power_status message format.

To run:

```
rosrun yrk_ros power_monitor.py
```

10.7 yrk_ros.switched_output_server

Enablesdisables the 5V and 12V switched outputs.

To run:

```
rosrun yrk_ros switched_output_server.py
```

CHAPTER

ELEVEN

ADDITIONAL DOCUMENTS

This section contains links to other useful documents such as electronic design files, 3D models files and datasheets for many of the core components use on (or designed for use with) the York Robotics Kit.

11.1 Schematic Diagrams

The Eagle design files and bill-of-material files for the **YRL039** and **YRL040** PCB The schematic diagrams and Eagle design files are available in the `pcb_documents` subfolder of the YRK repository.

File	Description
yrl039-schematic.pdf	Schematic diagram for the **YRL039** Power Supply Board
yrl039-top.pdf	Top silkscreen view of the **YRL039** Power Supply Board
yrl039-toptraces.pdf	Top copper layer view of the **YRL039** Power Supply Board
yrl039-bottomtraces.pdf	Bottom copper layer view of the **YRL039** Power Supply Board
yrl040-schematic.pdf	Schematic diagram for the **YRL040** York Robotics Board
yrl040-top.pdf	Top silkscreen view of the **YRL040** York Robotics Board
yrl040-bottom.pdf	Bottom silkscreen view of the **YRL040** York Robotics Board
yrl040-toptraces.pdf	Top copper layer view of the **YRL040** York Robotics Board
yrl040-bottomtraces.pdf	Bottom copper layer view of the **YRL040** York Robotics Board
yrl040-alltraces.pdf	Bottom view of the **YRL040** York Robotics Board showing both copper layers

11.2 Datasheets

This table contains internet URLs to most of the core intergrated circuits and modules in use on the **YRL039** and **YRL040** PCBs.

Component	Part Name	Datasheet URL
Arduino IC	ATMega328P	https://www.microchip.com/wwwproducts/en/ATmega328p
I2C Switch IC	PCA9548	http://www.ti.com/lit/ds/symlink/pca9548a.pdf
Serial:USB IC	FTDI FT232RL	https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf
Power Supply IC	TPS82130	http://www.ti.com/lit/ds/symlink/tps82130.pdf
GPIO Expander IC	PCA9555	https://www.nxp.com/docs/en/data-sheet/PCA9555.pdf
LED Driver IC	TCA6507	https://www.ti.com/lit/ds/symlink/tca6507.pdf
Motor Driver IC	DRV8830	https://www.ti.com/lit/ds/symlink/drv8830.pdf
PWM (Servo) IC	PCA9685	https://www.nxp.com/docs/en/data-sheet/PCA9685.pdf
Audio Amp IC	TPA2005	http://www.ti.com/lit/ds/symlink/tpa2005d1.pdf
Distance Sensor	GP2Y0A21	https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf
Distance Sensor	GP2Y0A41	https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a41sk_e.pdf

CHAPTER

TWELVE

INDEX

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

K

L

M

O

P

R

S

T

U

W

Y