FrostAV Final Report

December 19, 2019

Contents

0.1 Requirements Evaluation

0.1.1 Problem Domain Requirements

The main problem-domain requirements for this project are:

1. Given a lane, the car must travel approximately parallel to it, such that the car stays within its boundaries consistently, and, if the car is to accidentally leave it, it promptly returns.
2. Given a corner, the car must turn, continuing from the car's current lane to the next, such that the car stays within its boundaries consistently, and, if the car is to accidentally leave it, it promptly returns.
3. Given an obstacle, the car must stop until it is moved further from the car, or it is removed from circuit boundaries.
4. Given a sign, the car must respond to the event provided by it.
5. Given a circuit, the car must complete a full loop.

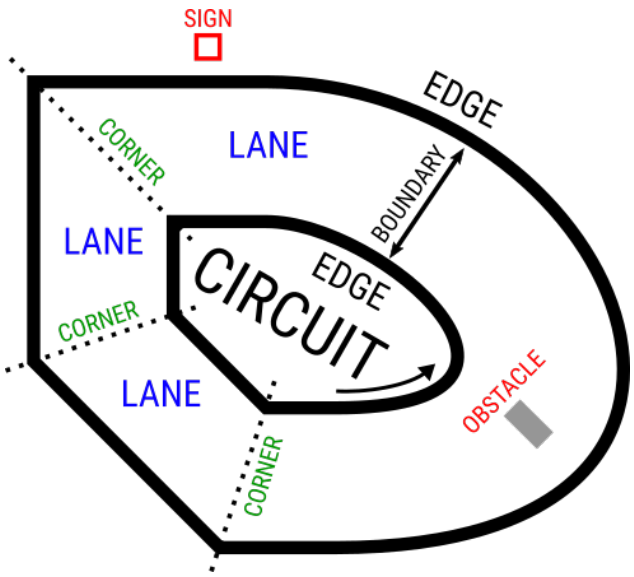


Figure 1: An example circuit which the car must navigate through.

The completed project is fully capable of meeting requirements 1, 2 and 5 and is unable to meet 3 and 4. The car can successfully navigate a straight path and recover from small excursions outside of the lane boundaries. It can also navigate corners and complete a circuit. Obstacle detection and sign detection was never implemented due to a lack of time.

0.1.2 Control System and Control Module Requirements

For the control system and control modules, the following requirements apply: The Control System shall:

1. Interface with vehicle peripherals, such as, motors, servos, batteries, etc.

2. Be the only system coupled to the vehicle.
3. Have a subset of Control Modules for each peripheral in need of control.

   Each Control Module shall:

4. Encapsulate a single purpose, such that, one controller controls a single peripheral.
5. Couple to a peripheral electronically, not mechanically.
6. Be able to communicate with other controllers via a wired bus.
7. Be able to communicate with non-controllers that depend on it, via a wired bus.
8. Communicate with other controllers and non-controllers via a single shared wired bus.

All the control system and control module requirements were met, except for the requirement that each control module must control a single peripheral. In the final design, the ATMEGA328 is used for both servo and throttle control instead of having a separate module for each. Otherwise, the control system was able to successfully interface with the vehicle peripherals using control modules, and the control modules share a single I2C bus. All control modules can communicate with the Raspberry Pi, which runs the main control system, and with each other (even though this capability was not utilized for the final design).

0.1.3 Autonomy Requirements

The vehicle's autonomy system shall:

1. Not be directly coupled to the Vehicle Interface
2. Be able to fit on the vehicle.
3. Allow the vehicle to navigate without user interaction.
4. Be able to communicate with the Control System.

The final design is capable of meeting all the autonomy requirements. The autonomy system can communicate with all modules that make up the control system, including communicating with multiple modules in quick succession. The system is not directly coupled to the vehicle interface, instead performing all vehicle functions using control modules. The system fits on the vehicle and allows the vehicle to navigate autonomously.

0.1.4 Vehicle Interface Requirements

The vehicle interface shall:

1. Include a way for the vehicle to be turned on and off, such that, the vehicle receives no power to the drive system or logic when off.
2. Include a way for the vehicle to have its logic turned on, while the drive system is off.
3. Provide a battery peripheral that powers the logic.
4. Provide a battery peripheral that powers the drive system.
5. The total power consumption of the logic and drive system cannot exceed the maximum capacity of the battery peripheral(s).

6. Provide a peripheral that moves the vehicle.
7. Provide a peripheral that steers the vehicle.
8. Provide an electronic interface from each peripheral.

All vehicle interface requirements were met with the final design. The design meets requirements 1 and 2 with a power switch on the logic battery, and a power switch and remote cutoff on the drive battery. This allows the logic to be powered independently of the drive system. Requirements 3 and 4 are met with the inclusion of the logic and drive batteries as well as the logic power supply. The batteries and power supply allow all the logic and drive system modules to be powered.

Requirement 5 is met by ensuring that the maximum power draw won't exceed the batteries maximum rating. All of the batteries used in this design are rated at 30C or greater, which is a measure of a batteries maximum discharge rate. For the 5200 mAh drive battery, a 30C rating corresponds to a maximum output of 156 A. The ESC used to power the drive motor and servo is rated for a maximum of 60A, so the maximum potential current draw is well within the battery's capabilities. The logic battery, which has a capacity of 2200 mAh and a discharge rating of 30C can supply a maximum of 66A. Since all logic is powered by the power supply (which has a current limiting function) the maximum current drawn from this battery will never exceed 3A.

Requirements 6 and 7 are met by the ATMEGA328, the motor, ESC and steering servo. The ATMEGA328 controls the servo and ESC, and the ESC regulates the motor's speed. These components allow the vehicle to be steered and moved by the autonomy system. Each peripheral also has an electronic interface. The ATMEGA328 communicates with the autonomy system, which is running on a Raspberry Pi using the I2C bus. The voltage and current draw of the logic battery are measured using the INA226 power monitoring chip built into the power supply and communicated to the autonomy system over the I2C bus.

0.1.5 Wireless Interface Requirements

The Wireless Interface shall:

1. Allow for wireless tunneling (e.g. via SSH)
2. Be able to access a server.
3. Provide bi-directional communication.

All of these requirements for the wireless system are met in the final design, using the Raspberry Pi 4's integrated WiFi. The Wifi connection allows for SSH tunneling as well as providing a HTTP server. The system is also capable of connecting to a server using SSH tunneling. The Wifi is also fully capable of bi-directional communication.

0.2 Additional Capabilities

During the development of the vehicle, a few additional capabilities and features were able to be implemented.

0.2.1 Status OLED Display

A SSD1306 OLED display module was added to the design towards the end of Milestone 4, which allows easier monitoring of the Raspberry Pi and battery parameters. The display shows the current

Figure 2: OLED display showing parameters.

IP address of the Pi, the CPU load and temperature, memory usage, and the voltage, current draw and power use from the logic battery.

This display has been useful in the development of the system because it allows important parameters to be monitored easily. Since the Clarkson University Wifi uses DHCP addresses, the Pi's IP changes periodically. The IP is needed for accessing SSH and the Pi's webserver so this is a very useful parameter to have. The voltage monitoring serves as a low battery indication for the logic battery.

0.2.2 HTTP Status Monitoring

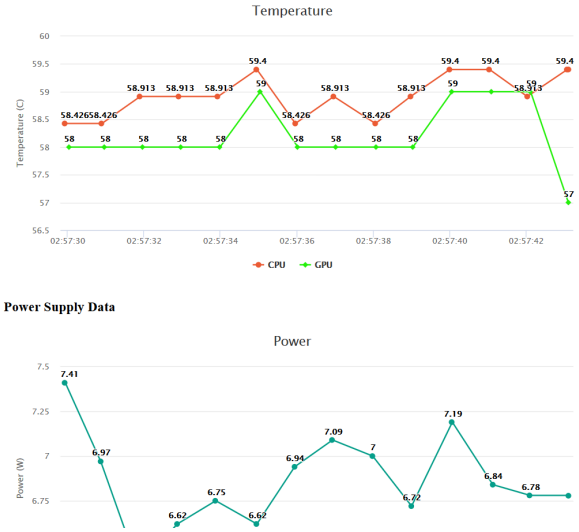


Figure 3: An example circuit which the car must navigate through.

Similar to the OLED display, a web server was implemented on the Pi to monitor various parameters. The parameters monitored are the Pi's CPU load, memory use, CPU and GPU temperature, and power, voltage and current from the logic battery. These parameters are graphed using JavaScript and the values are retrieved with PHP. The graphs update every 10 seconds and display the last 20 values. The webpage is accessed at the Pi's IP address.

This feature allows various parameters to be measured and allows the easy identification of trends in these values. The function of this webpage is like the OLED display, in that it's useful for the development of the system. This allowed for the identification of thermal throttling on the PI during testing.

0.2.3 Remote Motor/Servo Shutoff

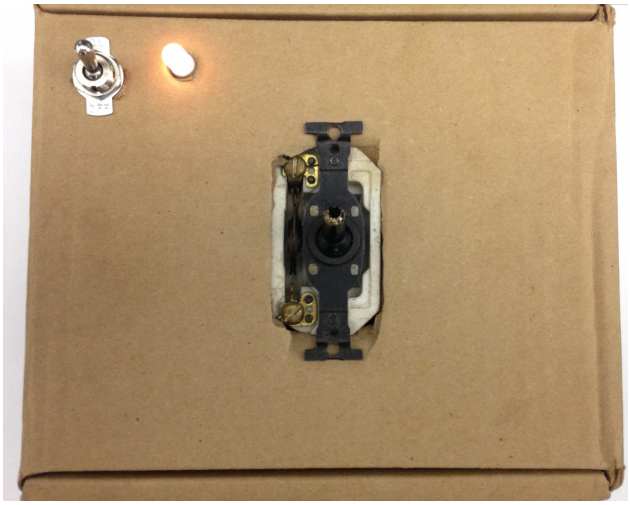


Figure 4: Shutoff remote control.

A remote shutoff system was implemented using Xbee radio modules and an IRF250 power MOSFET. The MOSFET cuts power to the motor speed controller and servo when the switch on the remote control unit is flipped. This system allows the car to be stopped remotely if it were to go out of control. The system is designed so that any failure will disconnect power to the motor and servo. If the Xbee on the car side loses signal or power, it will stop the motor.

0.3 Unique Innovations

0.3.1 Custom Power Supply

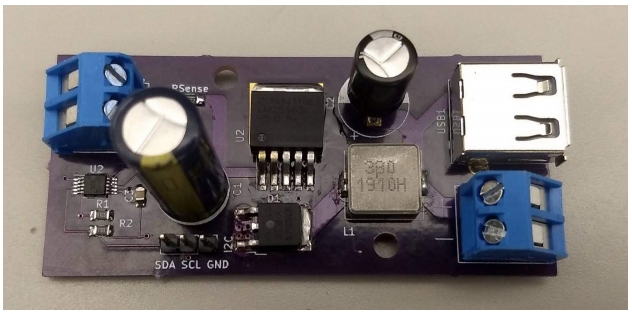


Figure 5: Power Supply complete.

A custom power supply was designed to convert the 7.2v from the LiPo batteries to 5v for powering the car's logic systems. The power supply is also capable of measuring battery power, voltage and current. The power supply is implemented using a LM2596-5.0 buck regulator IC, and an INA226 power monitoring IC. The power supply can supply the Raspberry Pi 4, which requires 5V at 3A as well as the ATMEGA328 board and other accessories.

For the design of the power supply, a TI LM2596-5.0 "Simple Switcher" buck converter was chosen because of its efficiency, 3A output and ability to accept an input voltage as low as 7v. Originally a LM7805 based design with a bypass transistor was considered, but it would have been much less

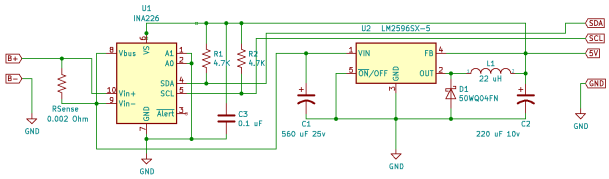


Figure 6: Power Supply schematic.

efficient. This would have reduced battery runtime and require a large heatsink which would have added weight. The values of L1, C1 and C2 were chosen based on the LM2596's datasheet. The output current was assumed to be 3A, and the maximum input voltage as 12v (providing the ability to operate using a 3 cell LiPo if needed).

The inductor L1 is a Bourns SRP1038A-220M which is shielded and has a saturation current of 5A. Since the components are in close proximity a shielded inductor is needed to prevent EMI from affecting the feedback line of the LM2596 (Pin 4) as well as the INA226 and I2C lines. The capacitors C1 and C2 are Nichicon aluminum polymer electrolytics, which have a very low (30 mΩ) ESR (equivalent series resistance). Low ESR is critical in a buck converter design because of the high current transients created by the switching. A low ESR allows the capacitor to charge and discharge faster to react to these transients. Diode D1 is a 50WQ04 Schottky diode which was used because of its fast switching time and low voltage drop.

For the power measurements, a TI INA226 current and power monitor was used because of its Linux driver support. The INA226 communicates with the Raspberry Pi over I2C, with R1 and R2 functioning as pull-up resistors. The INA226 measures current using a shunt resistor and measuring the voltage across it. The resistor's value is programmed in at startup and from there an accurate current measurement can be taken. The shunt resistor Rshunt is a 0.002Ω 1% resistor as recommended in the INA226 datasheet.

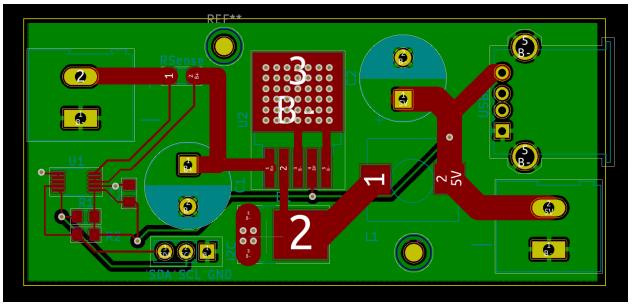


Figure 7: Power Supply PCB layout.

Since the buck converter operates at a frequency of about 150kHz and used many surface mounted components, a custom PCB (printed circuit board) was designed using KiCad. The power supply was originally constructed on a protoboard (without the INA226 power measurement IC) and while it was somewhat functional it had stability issues and the output was limited to about 2A at 5V. The custom PCB eliminated all of these issues as well as creating a more mechanically robust design.

High currents and frequencies are present in the power supply, so a ground plane design was used for the PCB. The back side of the board (shown in green) is copper filled and used as the ground connection for all components. Components on the front side of the board are connected to the

ground plane using through-hole vias, which are holes drilled and electroplated through the board. The ground plane is also used as a heatsink for the LM2596 buck controller IC. Multiple vias were placed on the heatsink pad of the device to help conduct heat and serve as a ground connection. Thicker traces and shorter trace lengths are used for high current paths.

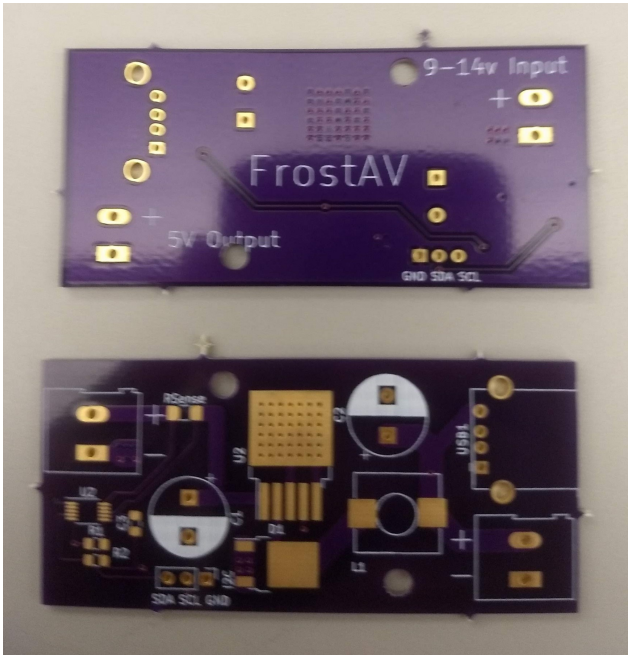


Figure 8: Power Supply PCB Before Assembly.

The PCB's were manufactured by OSHPark, and all components were hand soldered. The larger components such as the inductor, diode and buck regulator were hot air soldered and the smaller components were soldered using a fine soldering iron tip. Solder wick was used to remove excess solder and a few solder bridges between INA226 pins.

0.3.2 Reading Power Supply Data in Linux

To measure the battery voltage and power consumption, a TI INA226 power monitoring IC was added to the power supply design. Linux includes a driver to read data from this device, and in order to use it on the Raspberry PI configuration changes need to be made. Since the Raspberry Pi doesn't support hardware autodetection on any interface except USB, a system known as device tree is used to describe the system's hardware configuration. The device tree provides information on all the system's hardware, such as the addresses, registers, driver parameters and other information. The device tree is read by the Linux kernel at boot and then loads the required drivers and their parameters.

```
/dts-v1/;
/plugin/;
/ {
        fragment@0 {
                target = <&i2c1>;
                __overlay__ {
                        status = "okay";
```

```
                        ina226@40 {
                                compatible = "ti,ina226";
                                reg = <0x40>;
                                shunt-resistor = <2000>;
                        };
                };
        };
};
```

To add support for the INA226 a device tree overlay needed to be created to insert it into the system's device tree. The contents of this file are shown above. The bus that the INA226 is attached to is defined with `target = <&i2c1>`, the driver is set with `compatible = 'ti,ina226'` and the address with `reg = <0x40>`. The shunt resistor value is set with `shunt-resistor = <2000>`, with the resistance value in micro-ohms. The shunt resistor value is the resistance of the current shunt connected to the INA226, which is used for current and power calculations. In this case the current shunt's resistance is 0.02Ω. The device tree overlay is enabled in the Raspberry Pi's `/boot/config.txt` file.

Once the driver is set up the voltage, power and current measurements can be accessed by any Linux program. The `sensors` utility can be used to display these values.

0.4 Problems

0.4.1 Vehicle Issues

1. Motor and Drivetrain Issues

   Originally, the vehicle was intended to use a brushless motor and speed controller. There were issues with the brushless speed controller, so a brushless motor and speed controller were used instead. There were also issues getting the motor and center gear to mesh correctly, the motor kept shifting out of position. This was fixed by tightening the motor mounting screws. The 3D printed driveshaft coupling pieces broke a few times and the original center gear was warped. These pieces were reprinted in carbon fiber PLA.

2. Mounting and Fitment Issues

   The original steering servo, a Hitec HS-422 didn't fit in the steering servo mount. Since the base plate of the car was already printed, it couldn't be enlarged to fit the servo without the screw holes not lining up. Eventually another servo was found which was small enough to fit the mount. Another issue was encounterd when the Raspberry Pi 2 was replaced with a Raspberry Pi 4. The mounting holes on the Raspberry Pi are 1.9 mm in diameter but the smallest mounting hardware was 2 mm. The holes were drilled out to fit the mounting hardware.

   Another issue with the car was that since the body was slightly narrower than the original plans, the steering tie rods were slightly longer than needed. This caused the front wheels to angle in slightly. Shorter tie rods were printed but the wheels still had the same issue. Reprinting them again a few mm shorter would solve this issue.

0.4.2 Power Supply Issues

There were a few issues with the power supply during the development process. The original prototype of the power supply was built on a protoboard using through hole components. The inductor originally used for this design had the wrong value (68 uH vs 22 uH). Since this was a large toroidal inductor, the problem was resolved by taking turns off th inductor core until it measured at 22 uH. This iteration had stability issues where the power supply output voltage would randomly drop, even with no load. This iteration also had a max output current of about 1.5A. These issues were determined to be caused by the long lead lengths and EMI problems from the protoboard construction.

There was also a few issues with the second version mainly related to PCB layout. The USB port's power and ground lines were swapped, assuming it was top mounted. The solution to this problem was to desolder the port and install it upside down on the bottom of the board. Another issue was that the negative lead of the input capacitor (C1 on power supply schematic) was not connected to the ground plane, or anything else. The solution to this was to scrape off some solder mask from the ground plane and bridge the connection with solder.

0.4.3 Remote Shutoff Issues

A few problems were encountered with the remote shutoff system. With the original circuit, a JFET was used to drive the power MOSFET's gate. Sudden throttle changes and using the braking feature would cause the Xbee to reset, thus cutting power to the ESC. A decoupling capacitor was added to the Xbee power terminals, but this didn't solve the issue. The JFET was replaced with an optocoupler which electrically isolated the Xbee and power circuits. This solved the resetting issue, but the Xbee would still shut off ESC power after 60 seconds. This turned out to be a tim eout issue because the remote Xbee would only transmit when the switch's position was changed and the receiving ESC would time out after not receiving any transmission. After changing the Xbee programming to transmit every 10 ms this issue was solved.

1 Test Results

1.0.1 Autonomous Driving Tests

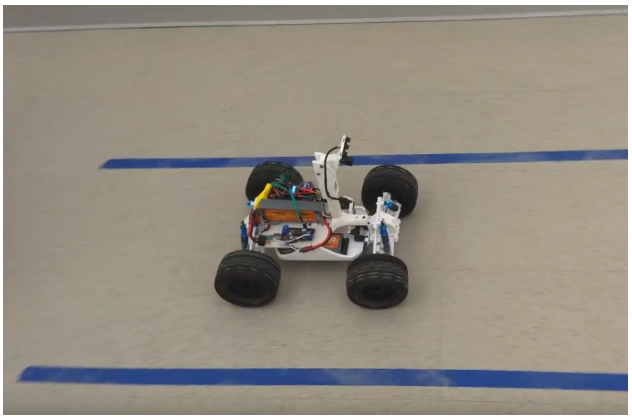


Figure 9: Vehicle navigating the wide section of test track.

In order to test the car's autonomous driving capabilities, a test track was constructed. The lines for

the testing track were marked with two parallel strips of blue masking tape. Two different widths were used for the track, 12 and 24 inches apart (based on the 12"x12" floor tiles).

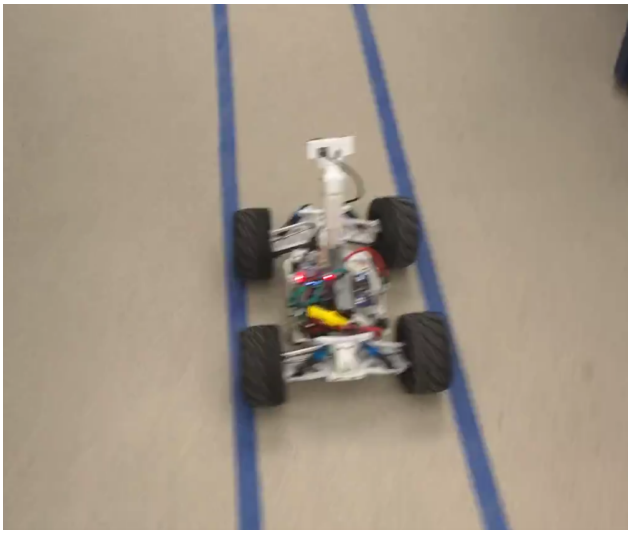


Figure 10: Vehicle navigating the narrow section of test track.

The vehicle was able to successfully navigate the test track multiple times. It did leave the track a couple of times during testing, and with more time these issues could be resolved. The vehicle could handle corners and varying track widths, as well as merging from the wide track to the narrow one.

1.0.2 OpenCV Lane Detection Testing Results

1.0.3 ATMEGA328 Firmware Testing Results

The ATMEGA328 firmware was tested in two parts, first testing the I2C interface itself and then verifying that the servo and throttle could be controlled using I2C commands. While the ATMEGA328 was originally intended to function as a PID controller as well, this function was eliminated due to a lack of development time.

For the first part of firmware testing, the ATMEGA was tested to ensure it would respond to its own I2C address. The ATMEGA was connected to the Raspberry Pi's I2C pins through a level converter, and the Linux utility "i2cdetect" was used to scan for I2C devices. This utility tries each possible I2C address, and shows the addresses of the devices that respond. An example of this is shown in figure 1.1.2-1, with the ATMEGA using I2C address 0x32.

During this testing, a few issues with the I2C controller were encountered. The first time "i2cdetect" was ran, the results were as shown above, with the ATMEGA responding to its own address and the other devices were able to respond as well. The second time it was run the ATMEGA held the data line to ground and no I2C communication was possible. When the ATMEGA was reset, the same issue would occur with a correct response the first time and a failure for all subsequent tries. This issue was found to be in an open source I2C library used on the ATMEGA which was not properly clearing the I2C interrupt bit. Once this issue was resolved the ATMEGA reliably responded to its I2C address.

For the next phase of testing the ATMEGA firmware was modified to print all data received over I2C to the USART. A small C++ program called "i2c-test" was written to send data to the ATMEGA

```
[alarm@alarmpi ~]$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- 32 -- -- -- -- -- -- -- -- -- 3c -- -- --
40: UU -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 11: Results of running 'i2cdetect' command while three devices were connected to the Raspberry Pi's I2C bus.

from the Raspberry Pi. When this test was performed, there was a minor issue with the ATMEGA firmware expecting a newline character (0x12) but "i2c-test" sending a null character (0x0) instead. Once this was fixed, the firmware performed as expected with all data sent over I2C being printed to the USART console. The final stage of testing was done to ensure that throttle and servo control over I2C were possible. The car was placed onto a stand and the motor speed controller and servo were connected to the ATMEGA board. The "i2c-test" code was used to send values for the throttle and servo angle. When this was tested the motor and servo responded as expected.

1.0.4 Power Supply Testing Results

Once the power supply was assembled, the first test was a load test to ensure that the power supply was capable of providing 3A at 5v with an input voltage of 7.4v (the lowest anticipated battery voltage). The ripple on the output of the power supply was also measured during load testing to ensure that it was less than 200 mV under all load conditions. Then the power monitoring circuit was connected to the Raspberry Pi over I2C and tested for functionality and accuracy.

Figure 12: Results of load testing, showing 3.4A current output at 5v with 7.4v input voltage

For the load test, the power supply was connected to a bench power supply (set to 7.4v) and the output terminals were connected to a load resistor. The voltage and current were measured at both the input and output. An oscilloscope probe, set to AC coupling was also connected to the output to measure ripple. The maximum current delivered by the power supply was 3.4A while maintaining a 5v output. Any higher current would trigger the overcurrent protection mode on the power supply.

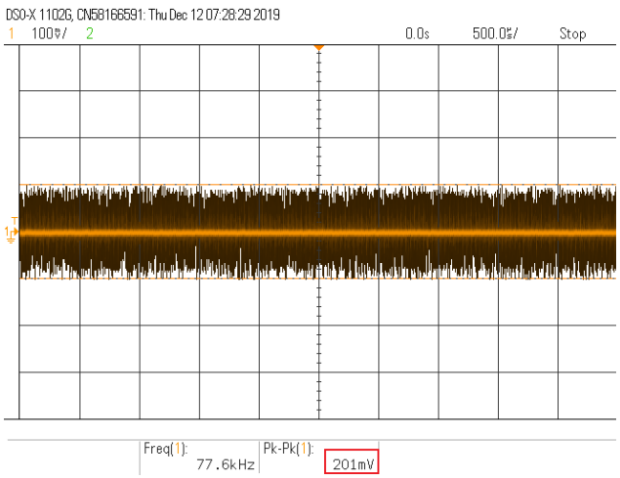


Figure 13: Results of ripple testing with a 3A load and 7.4v input voltage

The ripple with a 3A load was measured to be 201 mV.

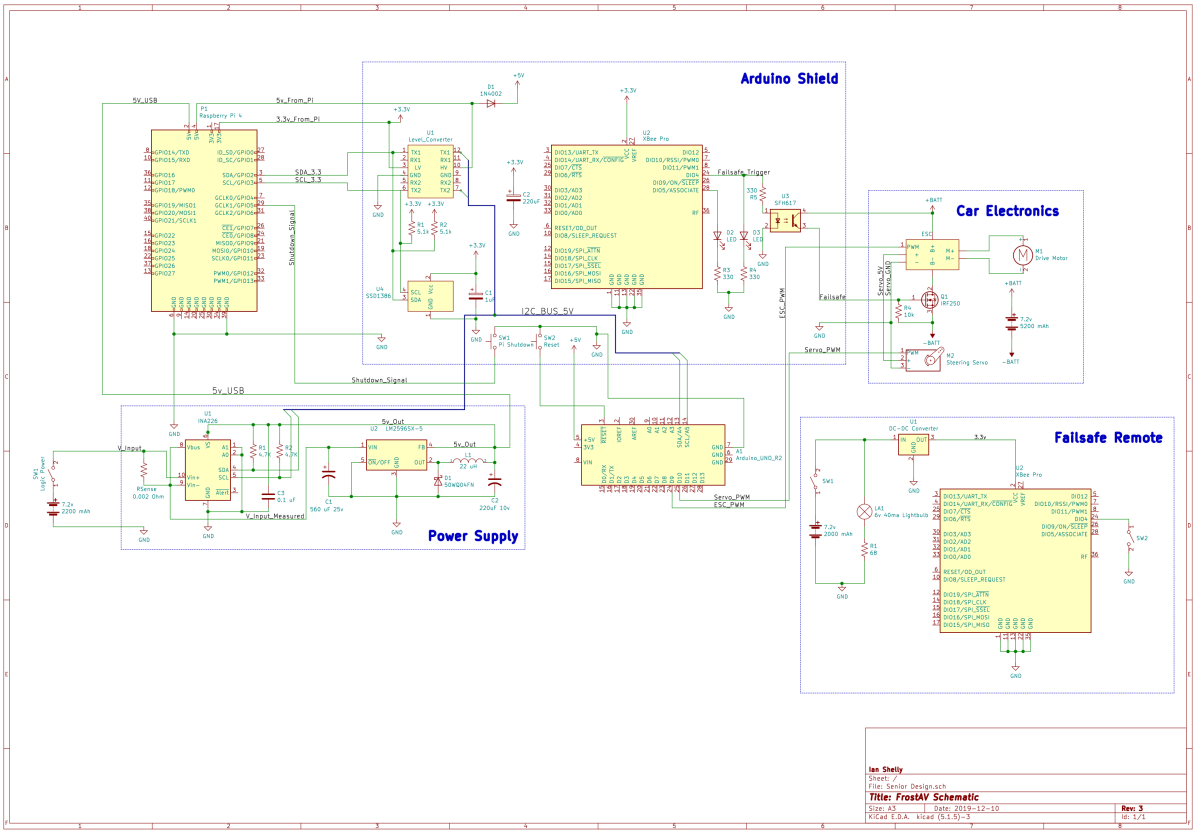
```
[alarm@alarmpi ~]$ sensors
ina226-i2c-1-40
Adapter: bcm2835 I2C adapter
in0:          1000.00 uV
in1:            7.42 V
power1:         4.34 W
curr1:        583.00 mA
```

Figure 14: Results of running "sensors" command, showing the voltage, current and power measurements from the power supply

The next phase of power supply testing was to ensure that the current and power monitoring section was functional. The power supply's I2C lines were connected to the Raspberry Pi through a level converter, and the Linux driver for the power monitoring chip was loaded. The Linux utility "sensors" was used to display the measurements. The measurements were verified with a multimeter and were within 0.01 of the multimeter reading for both current and voltage measurements.

2 Design Documentation (Appendix)

2.1 System Hardware Schematic



2.2 INA226 Device Tree Overlay (ina226.dts)

```
/dts-v1/;
/plugin/;
/ {
        fragment@0 {
                target = <&i2c1>;
                __overlay__ {
                        status = "okay";
                        ina226@40 {
                                compatible = "ti,ina226";
                                reg = <0x40>;
                                shunt-resistor = <2000>;
                        };
                };
        };
};
```

2.3 Raspberry Pi Boot Configuration (/boot/config.txt)

```
# See /boot/overlays/README for all available options

gpu_mem=64
initramfs initramfs-linux.img followkernel
dtparam=i2c1=on
dtparam=i2c_arm=on
disable_overscan=1
dtoverlay=ina226
hdmi_force_hotplug=1
dtoverlay=gpio-shutdown,gpio_pin=5
```

2.4 OpenCV Lane Detection (Raspberry Pi)

2.4.1 video.py

```
import numpy
import cv2
import math
import sys

def detect_edges(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    lower_blue = numpy.array([60, 40, 40])
    upper_blue = numpy.array([150, 255, 255])
    mask = cv2.inRange(hsv, lower_blue, upper_blue)
    edges = cv2.Canny(mask, 200, 400)
    return edges
def display_lines(frame, lines, line_color=(0, 255, 0), line_width=2):
    line_image = numpy.zeros_like(frame)
    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv2.line(line_image, (x1, y1), (x2, y2), line_color, line_width)
    line_image = cv2.addWeighted(frame, 0.8, line_image, 1, 1)
    return line_image
def region_of_interest(edges):
    height, width = edges.shape
    mask = numpy.zeros_like(edges)

    # only focus bottom half of the screen
    polygon = numpy.array([[
        (0, height * 1 / 2),
        (width, height * 1 / 2),
        (width, height),
```

```
        (0, height),
    ]], numpy.int32)

    cv2.fillPoly(mask, polygon, 255)
    cropped_edges = cv2.bitwise_and(edges, mask)
    return cropped_edges
def detect_line_segments(cropped_edges):
    # tuning min_threshold, minLineLength, maxLineGap is a trial and error process by hand
    rho = 1  # distance precision in pixel, i.e. 1 pixel
    angle = numpy.pi / 180  # angular precision in radian, i.e. 1 degree
    min_threshold = 10  # minimal of votes
    line_segments = cv2.HoughLinesP(cropped_edges, rho, angle, min_threshold,
                                    numpy.array([]), minLineLength=8, maxLineGap=4)

    return line_segments
def make_points(frame, line):
    height, width, _ = frame.shape
    slope, intercept = line
    y1 = height  # bottom of the frame
    y2 = int(y1 * 1 / 2)  # make points from middle of the frame down

    # bound the coordinates within the frame
    x1 = max(-width, min(2 * width, int((y1 - intercept) / slope)))
    x2 = max(-width, min(2 * width, int((y2 - intercept) / slope)))
    return [[x1, y1, x2, y2]]

def average_slope_intercept(frame, line_segments):
    """
    This function combines line segments into one or two lane lines
    If all line slopes are < 0: then we only have detected left lane
    If all line slopes are > 0: then we only have detected right lane
    """
    lane_lines = []
    if line_segments is None:
        return lane_lines

    height, width, _ = frame.shape
    left_fit = []
    right_fit = []

    boundary = 1/3
    left_region_boundary = width * (1 - boundary)  # left lane line segment should be on left 2
    right_region_boundary = width * boundary # right lane line segment should be on left 2/3 of

    for line_segment in line_segments:
        for x1, y1, x2, y2 in line_segment:
```

```
            if x1 == x2:
                continue
            fit = numpy.polyfit((x1, x2), (y1, y2), 1)
            slope = fit[0]
            intercept = fit[1]
            if slope < 0:
                if x1 < left_region_boundary and x2 < left_region_boundary:
                    left_fit.append((slope, intercept))
            else:
                if x1 > right_region_boundary and x2 > right_region_boundary:
                    right_fit.append((slope, intercept))

    left_fit_average = numpy.average(left_fit, axis=0)
    if len(left_fit) > 0:
        lane_lines.append(make_points(frame, left_fit_average))

    right_fit_average = numpy.average(right_fit, axis=0)
    if len(right_fit) > 0:
        lane_lines.append(make_points(frame, right_fit_average))

    return lane_lines
def detect_lane(frame):

    edges = detect_edges(frame)
    cropped_edges = region_of_interest(edges)
    line_segments = detect_line_segments(cropped_edges)
    lane_lines = average_slope_intercept(frame, line_segments)

    return lane_lines
def display_heading_line(frame, steering_angle, line_color=(0, 0, 255), line_width=5 ):
    heading_image = numpy.zeros_like(frame)
    height, width, _ = frame.shape

    # figure out the heading line from steering angle
    # heading line (x1,y1) is always center bottom of the screen
    # (x2, y2) requires a bit of trigonometry

    steering_angle_radian = steering_angle / 180.0 * math.pi
    x1 = int(width / 2)
    y1 = height
    x2 = int(x1 - height / 2 / math.tan(steering_angle_radian))
    y2 = int(height / 2)

    cv2.putText(frame, f"{steering_angle} deg", (int(width/2) - 40, y2-10), cv2.FONT_HERSHEY_SIN
    cv2.line(heading_image, (x1, y1), (x2, y2), line_color, line_width)
    heading_image = cv2.addWeighted(frame, 0.8, heading_image, 1, 1)
```

```
    return heading_image
def stabilize_steering_angle(
          curr_steering_angle,
          new_steering_angle,
          num_of_lane_lines,
          max_angle_deviation_two_lines=5,
          max_angle_deviation_one_lane=1):
    """
    Using last steering angle to stabilize the steering angle
    if new angle is too different from current angle,
    only turn by max_angle_deviation degrees
    """
    if num_of_lane_lines == 2 :
        # if both lane lines detected, then we can deviate more
        max_angle_deviation = max_angle_deviation_two_lines
    else :
        # if only one lane detected, don't deviate too much
        max_angle_deviation = max_angle_deviation_one_lane

    angle_deviation = new_steering_angle - curr_steering_angle
    if abs(angle_deviation) > max_angle_deviation:
        stabilized_steering_angle = int(curr_steering_angle
            + max_angle_deviation * angle_deviation / abs(angle_deviation))
    else:
        stabilized_steering_angle = new_steering_angle
    return stabilized_steering_angle

def getMp4Colorless(name: str, fps: int) -> cv2.VideoWriter:
    return cv2.VideoWriter(
        filename = f"{name}_{fps}fps.mp4",
        fourcc = cv2.VideoWriter_fourcc(*'X264'),
        fps = fps,
        frameSize = (640,480),
        isColor = False)

def getMp4Color(name: str, fps: int) -> cv2.VideoWriter:
    return cv2.VideoWriter(
        filename = f"{name}_{fps}fps.mp4",
        fourcc = cv2.VideoWriter_fourcc(*'X264'),
        fps = fps,
        frameSize = (640,480),
        isColor = True)


frame = cv2.imread(sys.argv[1])
lane_lines = detect_lane(frame)
height, width, _ = frame.shape
```

```
#2 lane detected
_, _, left_x2, _ = lane_lines[0][0]
_, _, right_x2, _ = lane_lines[1][0]
mid = int(width / 2)
x_offset = (left_x2 + right_x2) / 2 - mid
y_offset = int(height / 2)

#Only 1 lane detected
# x1, _, x2, _ = lane_lines[0][0]
# x_offset = x2 - x1
# y_offset = int(height / 2)

angle_to_mid_radian = math.atan(x_offset / y_offset)  # angle (in radian) to center vertical li
angle_to_mid_deg = int(angle_to_mid_radian * 180.0 / math.pi)  # angle (in degrees) to center v
steering_angle = angle_to_mid_deg + 90  # this is the steering angle needed by picar front wheel

lane_lines_image = display_lines(frame, lane_lines)
cv2.imshow("lane lines", lane_lines_image)
cv2.imshow("", display_heading_line(frame, steering_angle))
cv2.waitKey(0)
```

2.5 Web Status Page

2.5.1 index.html

```
<!DOCTYPE HTML><html>

<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <script src="/highcharts.js"></script>
  <title>Pi Status</title>
  <link rel="icon" href="/favicon.jpg">
  <style>
    body {
      min-width: 310px;
            max-width: 800px;
            height: 400px;
      margin: 0 auto;
    }
    h2 {
      font-family: Arial;
      font-size: 2.5rem;
      text-align: center;
    }
  </style>
```

```
</head>
<body>
  <h2>Vehicle Data</h2>


</form>
  <h3>Pi Data</h3>
  <div id="chart-load" class="container"></div>
  <div id="chart-mem" class="container"></div>
  <div id="chart-temperature" class="container"></div>
  <h3>Power Supply Data</h3>
  <div id="chart-power" class="container"></div>
  <div id="chart-voltage" class="container"></div>
  <div id="chart-current" class="container"></div>

</body>
<script>
var chartC = new Highcharts.Chart({
  chart:{ renderTo : 'chart-temperature' },
  title: { text: 'Temperature' },
  series: [{
    name: 'CPU',
    showInLegend: true,
    data: [],
    color: '#eb5b34'
  }, {
    name: 'GPU',
    showInLegend: true,
    data: [],
    color: '#24f00a'
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#059e8a' }
  },
  xAxis: { type: 'datetime',
    dateTimeLabelFormats: { second: '%H:%M:%S' }
  },
  yAxis: {
    title: { text: 'Temperature (C)' }
  },
  credits: { enabled: false }
});
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
```

```
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var x = (new Date()).getTime(),
          y = parseFloat(this.responseText);
      //console.log(this.responseText);
      if(chartC.series[0].data.length > 40) {
        chartC.series[0].addPoint([x, y], true, true, true);
      } else {
        chartC.series[0].addPoint([x, y], true, false, true);
      }
    }
  };
  xhttp.open("GET", "/temperature_c.php", true);
  xhttp.send();
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var x = (new Date()).getTime(),
          y = parseFloat(this.responseText);
      //console.log(this.responseText);
      if(chartC.series[1].data.length > 40) {
        chartC.series[1].addPoint([x, y], true, true, true);
      } else {
        chartC.series[1].addPoint([x, y], true, false, true);
      }
    }
  };
  xhttp.open("GET", "/temperature_g.php", true);
  xhttp.send();

}, 1000 ) ;

var chartL = new Highcharts.Chart({
  chart:{ renderTo : 'chart-load' },
  title: { text: 'System Load' },
  series: [{
    showInLegend: false,
    data: []
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#059e8a' }
  },
  xAxis: { type: 'datetime',
    dateTimeLabelFormats: { second: '%H:%M:%S' }
```

```
  },
  yAxis: {
  },
  credits: { enabled: false }
});
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var x = (new Date()).getTime(),
          y = parseFloat(this.responseText);
      //console.log(this.responseText);
      if(chartL.series[0].data.length > 40) {
        chartL.series[0].addPoint([x, y], true, true, true);
      } else {
        chartL.series[0].addPoint([x, y], true, false, true);
      }
    }
  };
  xhttp.open("GET", "/load.php", true);
  xhttp.send();
}, 1000 ) ;

var chartM = new Highcharts.Chart({
  chart:{ renderTo : 'chart-mem' },
  title: { text: 'Memory' },
  series: [{
    name: 'Used Memory',
    color: '#00FF00',
    showInLegend: true,
    data: []
  }, {
  name: 'Free Memory',
  color: '#FF00FF',
  showInLegend: true,
  data: []
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#059e8a' }
  },
  xAxis: { type: 'datetime',
    dateTimeLabelFormats: { second: '%H:%M:%S' }
  },
  yAxis: {
```

```
    title: {text: "Memory (Mb)" }
  },
  credits: { enabled: false }
});
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var x = (new Date()).getTime(),
          y = parseFloat(this.responseText);
      if(chartM.series[0].data.length > 40) {
        chartM.series[0].addPoint([x, y], true, true, true);
      } else {
        chartM.series[0].addPoint([x, y], true, false, true);
      }
    }
  };
  xhttp.open("GET", "/memory.php", true);
  xhttp.send();
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var x = (new Date()).getTime(),
          y = parseFloat(this.responseText);
      if(chartM.series[1].data.length > 40) {
        chartM.series[1].addPoint([x, y], true, true, true);
      } else {
        chartM.series[1].addPoint([x, y], true, false, true);
      }
    }
  };
  xhttp.open("GET", "/memfree.php", true);
  xhttp.send();

}, 1000 ) ;


var chartT = new Highcharts.Chart({
  chart:{ renderTo : 'chart-power' },
  title: { text: 'Power' },
  series: [{
    showInLegend: false,
    data: []
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
```

```
    },
    series: { color: '#059e8a' }
  },
  xAxis: { type: 'datetime',
    dateTimeLabelFormats: { second: '%H:%M:%S' }
  },
  yAxis: {
    title: { text: 'Power (W)' }
  },
  credits: { enabled: false }
});
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var x = (new Date()).getTime(),
          y = parseFloat(this.responseText);
      //console.log(this.responseText);
      if(chartT.series[0].data.length > 40) {
        chartT.series[0].addPoint([x, y], true, true, true);
      } else {
        chartT.series[0].addPoint([x, y], true, false, true);
      }
    }
  };
  xhttp.open("GET", "/power.php", true);
  xhttp.send();
}, 1000 ) ;

var chartH = new Highcharts.Chart({
  chart:{ renderTo:'chart-voltage' },
  title: { text: 'Battery Voltage' },
  series: [{
    showInLegend: false,
    data: []
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    }
  },
  xAxis: {
    type: 'datetime',
    dateTimeLabelFormats: { second: '%H:%M:%S' }
  },
  yAxis: {
    title: { text: 'Voltage (V)' }
```

```
  },
  credits: { enabled: false }
});
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var x = (new Date()).getTime(),
          y = parseFloat(this.responseText);
      //console.log(this.responseText);
      if(chartH.series[0].data.length > 40) {
        chartH.series[0].addPoint([x, y], true, true, true);
      } else {
        chartH.series[0].addPoint([x, y], true, false, true);
      }
    }
  };
  xhttp.open("GET", "/voltage.php", true);
  xhttp.send();
}, 1000 ) ;

var chartP = new Highcharts.Chart({
  chart:{ renderTo:'chart-current' },
  title: { text: 'Battery Current' },
  series: [{
    showInLegend: false,
    data: []
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#18009c' }
  },
  xAxis: {
    type: 'datetime',
    dateTimeLabelFormats: { second: '%H:%M:%S' }
  },
  yAxis: {
    title: { text: 'Current (mA)' }
  },
  credits: { enabled: false }
});
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
```

```
      var x = (new Date()).getTime(),
          y = parseFloat(this.responseText);
      //console.log(this.responseText);
      if(chartP.series[0].data.length > 40) {
        chartP.series[0].addPoint([x, y], true, true, true);
      } else {
        chartP.series[0].addPoint([x, y], true, false, true);
      }
    }
  };
  xhttp.open("GET", "/current.php", true);
  xhttp.send();
}, 1000 ) ;
</script>
</html>
```

2.5.2 power.php

```
<?php

// Use ls command to shell_exec
// function
$output = shell_exec('(sensors | grep power)'); //'bash -c 'sensors');// | grep power | sed 's/

$output = preg_replace('/^[^:]+:\s*/', '' ,$output);

$output = preg_replace('/[^0-9^.]*/', '', $output);

//$output2 = preg_replace( '[^0-9]*', '', $output);
// Display the list of all file
// and directory
echo "$output";
?>
```

2.5.3 load.php

```
<?php

// Use ls command to shell_exec
// function
$output = shell_exec('cat /proc/loadavg');

$output = substr ( $output, 0, 4);
```

```
echo "$output";
?>
```

2.5.4 temperature_c.php

```
<?php

// Use ls command to shell_exec
// function
$output = shell_exec('cat /sys/class/thermal/thermal_zone0/temp');

$output = $output / 1000;

echo "$output";
?>
```

2.5.5 temperature_g.php

```
<?php

// Use ls command to shell_exec
// function
$output = shell_exec('/opt/vc/bin/vcgencmd measure_temp');

$output = preg_replace("/[^0-9]/", "",$output);

$output = $output / 10;

echo "$output";
?>
```

2.5.6 voltage.php

```
<?php

// Use ls command to shell_exec
// function
$output = shell_exec('(sensors | grep in1)'); //'bash -c 'sensors');// | grep power | sed 's/[^

$output = preg_replace('/^[^:]+:\s*/', '' ,$output);

$output = preg_replace('/[^0-9^.]*/', '', $output);

//$output2 = preg_replace( '[^0-9]*', '', $output);
```

```
// Display the list of all file
// and directory
echo "$output";
?>
```