Pi Car Power Management

by

Yunshen Huang

Submitted in partial fulfillment of the requirements

for the Master Independent Study

Department of Electrical and System Engineering

WASHINGTON UNIVERSITY IN ST. LOUIS

May 2018

Contents

1. Proposal

There are many industrial robotics platforms, from large to small scale, from autonomous to manual manipulation. However, only a few studies the power monitoring and management. However, this filed will become increasingly important when robots are given more functionalities for various tasks desired. In this regard, the battery life could become an important criterion to judge whether a robot is suitable for the task.

By studying each electric component's power consumption, we can build a mathematical performance model and guide the user to decide which mode is supposed to operate to obtain a better performance. We believe, this model could also be used in the performance prediction field, which potentially saves more energy based on the same platform. Therefore, the success experiment could be extremely valuable for the industry.

2. Background

2.1 Hardware components.

To give the very first intuition of the small-scale robot power management, we decide to utilize the existing Pi Car, which consists of a plastic chassis, two layers of shelves (place components), a servo motor (control the direction of front wheels), an ESC and DC brushless speed motor (provide the steering power to the car), an Arduino, and a Raspberry Pi and a Pi camera.

Based on this car, we have added more components to give it more functionalities. Color sensor (guides the car to drive in a straight line), encoder (forms the closed loop speed controller by measuring the real speed of the car), current sensors (measure current of each component).

Below is the picture of the Pi Car

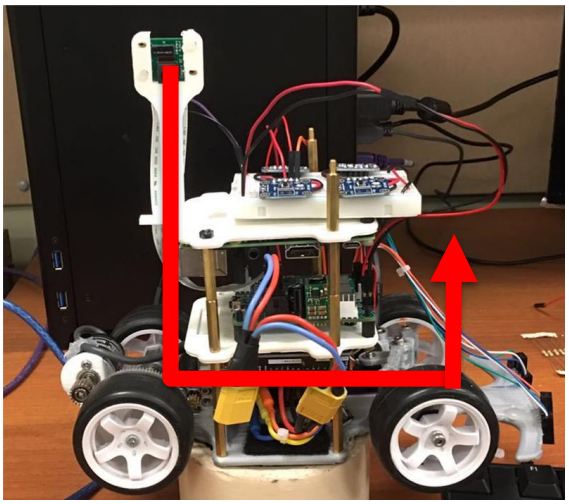


Figure 1. Pi Car appearance

Follow the array, each component is marked as respectively:

Pi camera, current sensor, Raspberry Pi, power board, Arduino, encoder, brushless motor, ESC (electric speed controller), battery, servo motor, color sensor.

The way how they are manipulated will be detailed described in the later chapters.

2.2 Wiring

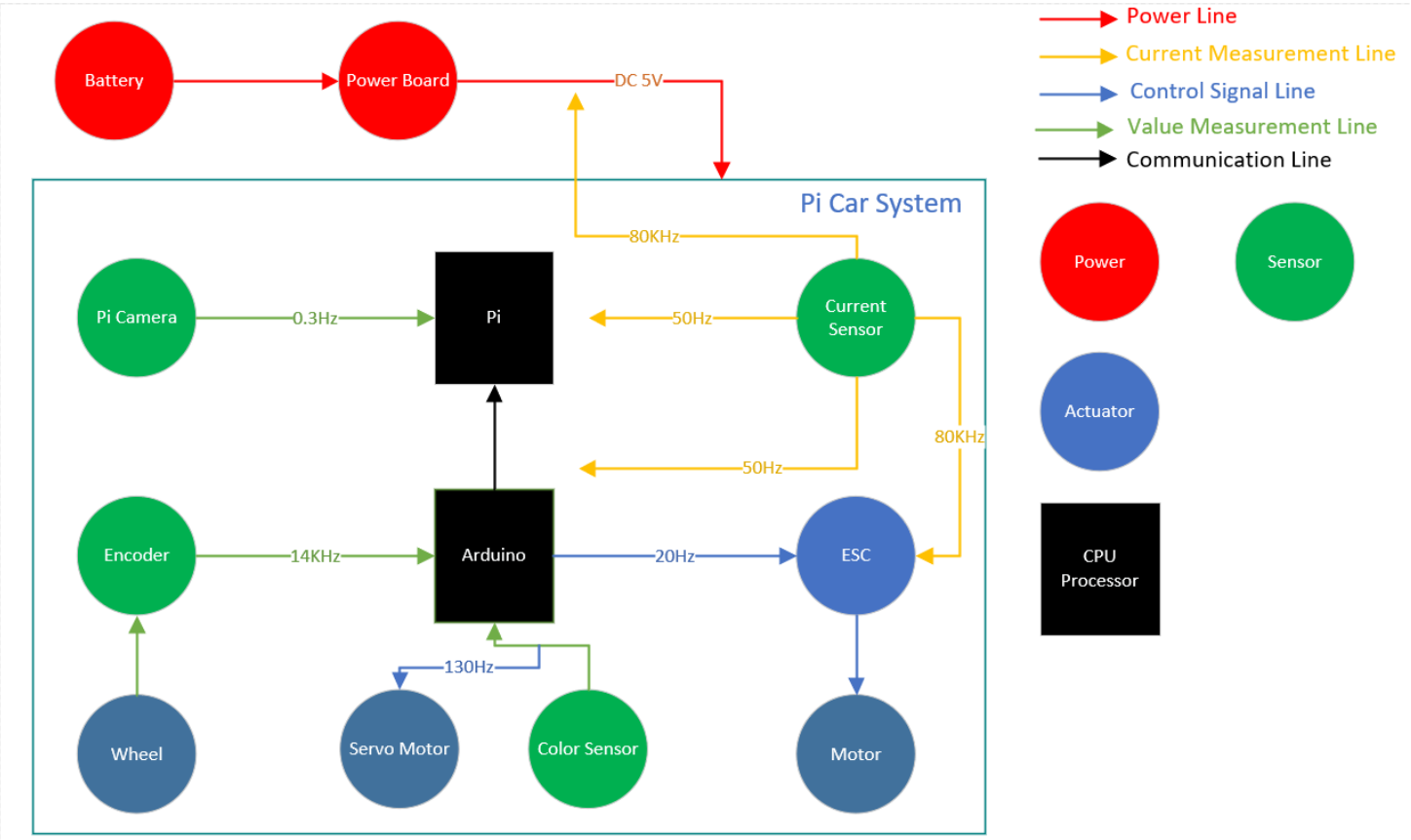


Figure 2. Internal wiring

The above diagram shows the internal wiring of each component. There are 5 types of units, power, sensor, actuator, low level controller, and Mirco-controller. Additionally, there are 5 basic wires classified by the functionality: power supply, current measurement, control signal (generated by the CPU), value measurement (normally generated by sensors) and communication (between Arduino and Pi).

- The LiPo battery is rated between 7.4-8.4V, which will be connected to the power board and ESC directly. The power board decreases the voltage to a stable 5V and then powers every component except ESC.
- The ESC takes in 8.4V and converts it to a periodic voltage signal to drive the three-phase brushless motor that works similarly with the AC motor.
- Pi camera takes the picture every three seconds and then transmits it to the Pi to be processed.
- The encoder rated 200 pulse/round measures the real time speed of the wheel and feeds back to the Arduino to form a closed loop speed controller, which will be discussed in chapter 3.2.
- The color sensor consists of a photodiode and a photo transmitter. Depending on the fact that distinct colors reflect different light intensity, the color sensor outputs different analog voltage value when pointing to different colors. Therefore, it is reasonable to use color sensor to guide the car to follow the straight blue line against the white floor.

- The current sensor consists of a shunt resistor, amplifier and a load resistor. By serials connected to the circuit, the voltage dropped on the shunt resistor will be amplified and reflected on the load resistor. The voltage dropped, which is related to the current of the circuit, on the load resistor is the output of the current sensor. In this project, I used it to measure three components, ESC, Pi and Arduino.

The entire system is built on two-layer control, high level and low level.

Low level control: the most basic component is connected to the low level controller Arduino, which includes sensor and actuator. Since the Arduino has stable timer frequency, it would be better to use the Arduino to collect large raw data and control every actuator. Then during a certain period, the Arduino transmits processed data to Pi for further usage.

High level control: the four-core Raspberry Pi 3 has very powerful computation ability. In this regard, it would be better to be used to store the data transmitted from the Arduino, and do more complex computational task, like image processing and direction decision.

Based on various tasks, each component has a unique operating frequency, which is indicated in figure 2. And they will be discussed in the chapter 3 & 4.

2.3 Software strategy

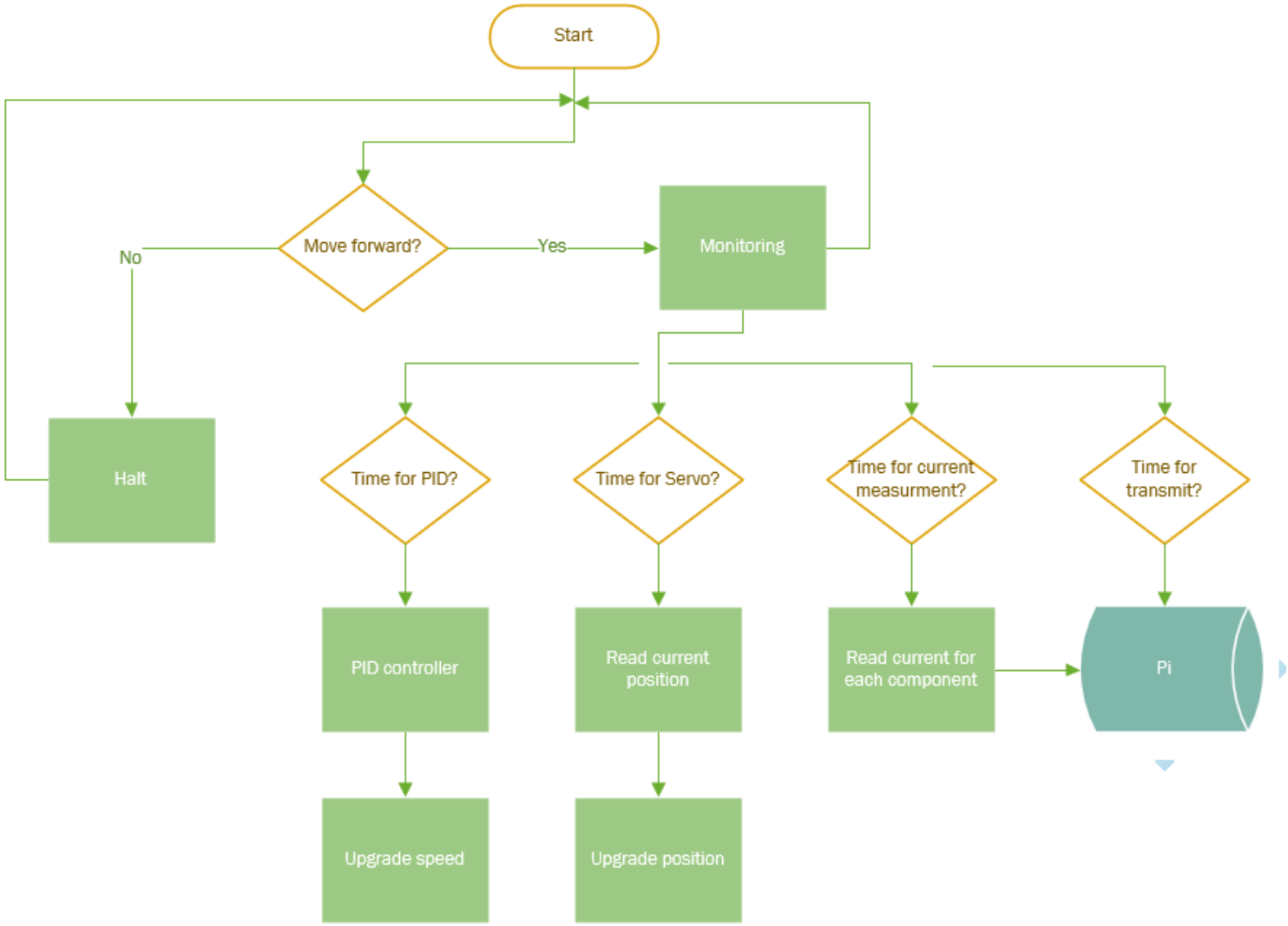


Chart 1. Arduino programming strategy

The flowchart 1 shows the general idea of the Arduino. In the current stage, two moving modes are considered, halt and forward. During the halt mode, nothing needed to be done. During the forward mode, the monitoring function will be activated to monitor any change. When it is the time to update the speed, the PID controller functions will be activated and then they will update the new speed to the motor. Similarly, when it is the time for changing the direction, the Arduino will firstly read the current position and react to it. When it is the time to record each component's current, the ADC will read each current sensor's voltage and store them in corresponding array. When it is the time for transmitting the current recorded to Pi, the Arduino transmits the stored current data to Pi via the serial connection.

Each functionality will be elaborated in the chapter 3 & 4.

3. Steering

3.1 Color sensor

Figure 3 indicates the components of the color sensor and indicates how it is connected to the circuit as well. The color sensor consists of a photodiode and a photo-controlled transmitter. By powered the 5V DC voltage, the photodiode emits UV ray in a certain power. After reflected by the certain type of color, a certain amount of UV ray is received by the transmitter's Base. Then this ray's energy controls the current flow through the circuit from the Collector to Emitter. A brighter color reflects more ray, which allows more current to flow through the circuit and decreases the output voltage. The value varies in 0-5V. By converted by the 10 bits ADC of Arduino, it is mapped to a digital value between 0-1024.

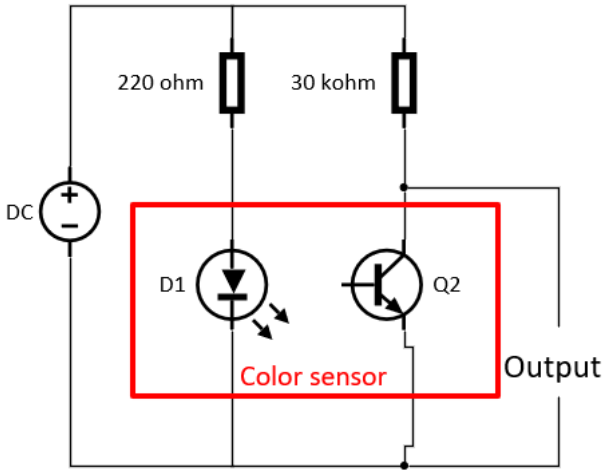


Figure 3. color sensor circuit

Two color sensors are horizontally attached on the head of the car, whose distance roughly equals to the width of the blue line. Therefore, by measuring the output voltage, two sensors will tell whether the car is on the blue line. If the car is off the line, the Arduino will be aware of and fix it by adjusting the front wheels' angle.

3.2 Closed loop control

3.2.1 Encoder

The rotary encoder is the electrical device that converts the angular position or motion of a shaft or axle to an analog or digital signal.

There are two main types of encoder: absolute and incremental encoder. While in this project, to decrease the cost the platform, and also considering only the forward movement is interested, the incremental encoder is used. The one has 200 $\mathrm{pulse/round}$, which feedbacks a quite high resolution speed information.

3.2.2 WHY Speed Control

Since the motor is a speedy motor, rated at 5050KV, which means 5050 rpm/V. By applying 8.4V, the highest speed can reach to $42420\ \mathrm{rpm} = 707\ \mathrm{rps}$, which is extremely fast for the ground steering robot. Therefore, when manipulating it in the low speed level, the characteristic of the

motor appears to be highly non-linear. Given a fixed step of input value increment leads to an unequal speed increment. Is this regard, it is extremely hard to precisely control the motor's speed.

In the same time, according to the characteristic of the motor's speed vs torque, the stall torque is much larger than the spinning torque, as shown in figure 4.

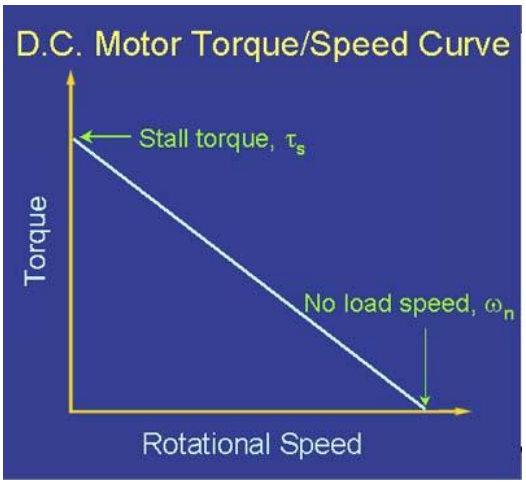


Figure 4. DC motor's torque vs. speed curve

In a nutshell, it is extremely hard to precisely control the motor's speed in the desired level without a speed controller.

Therefore, a closed loop reference-based speed controller is preferred. In this project, the classic and efficient PID controller is applied. The concept of the PID is demonstrated in the chart 2.

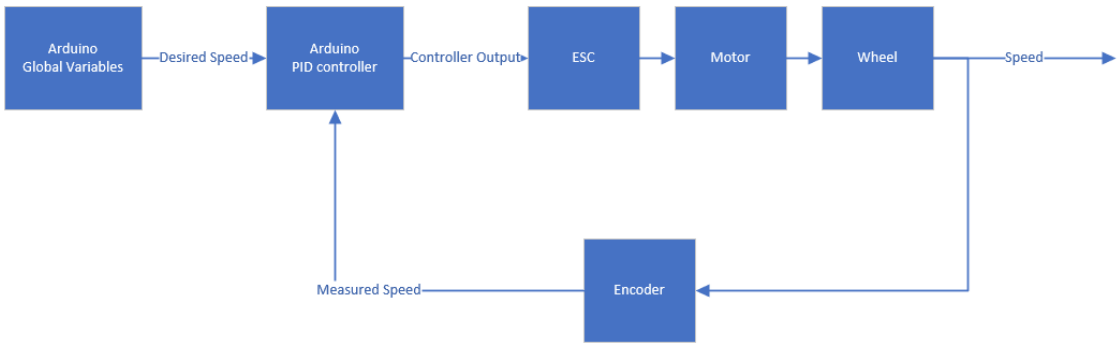


Chart 2. PID concept

3.2.3 Implementation

A digital PID controller is deployed in the Arduino, takes the difference between the desired speed and encoder feedbacked speed value as the input. Then computes the output signal as the controlled voltage value of ESC.

In our project, the detailed implementation is demonstrated in the chart 3.

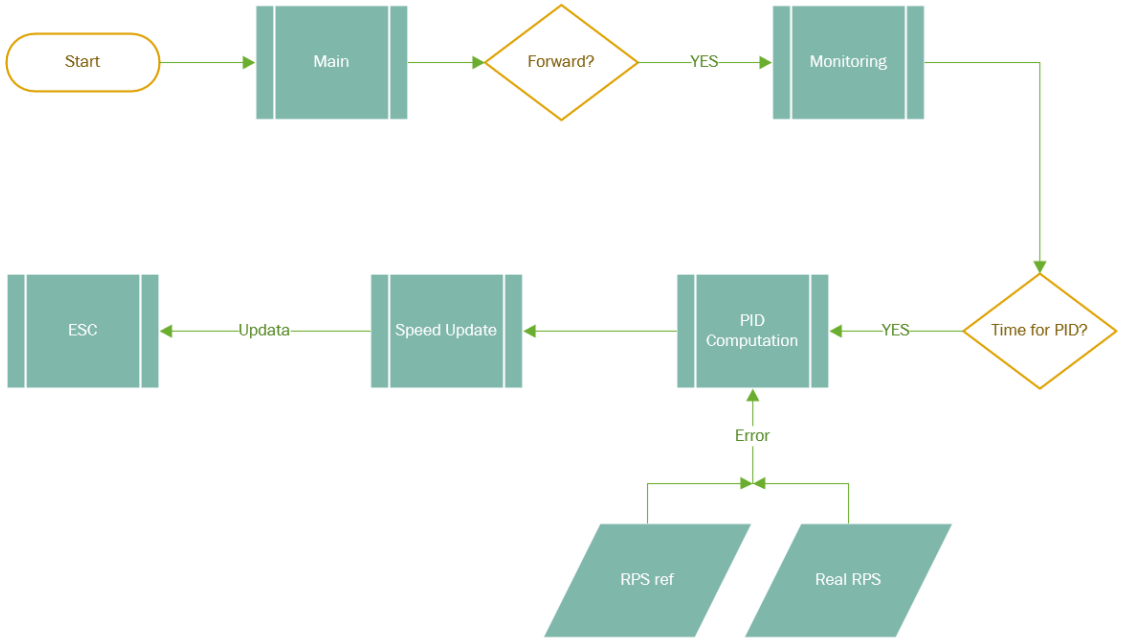


Chart 3. PID flow graph

When it is the time to update the speed, the PID computation function will be activated by taking the difference between the value of reference speed and real speed, and then computes the PID output as the ESC voltage. Eventually this value will be updated to the ESC to control the speed.

4. Current monitoring

4.1 Current Sensor

The general idea of the current monitoring is simple and basic, by using

$$U * I * \Delta t = W$$

Additionally, the power board provides steady 5V to each component. By given the current and time slide, the power consuming can be easily obtained.

However, there are several things need to be well considered

- To better characterize each components' current, the sampling rate is supposed to be as fast as possible.
- Due to the limitation of the hardware, the sampling rate cannot be infinitely high
- By considering serval existing approaches to measure the current, adding the shunt resistor to the circuit is the most proper way.
- Various current regulation largely differs among various components. Therefore, different resolution is needed, which means the resolution of the current sensor is supposed to be adjustable.

According to the attention above, a light weight and reliable current sensor INA 169 is chosen to achieve the measurement task.

The current sensor (ranging 0-5A) consists of three main parts: $0.1\ \Omega$ shunt resistor, $10\ \mathrm{K}\Omega$ load resistor, a $1\ \mathrm{K}$ gain amplifier. To measure several components simultaneously, multiple current sensors work together. The measured parameters' value is attached in the appendix table a.1&2.

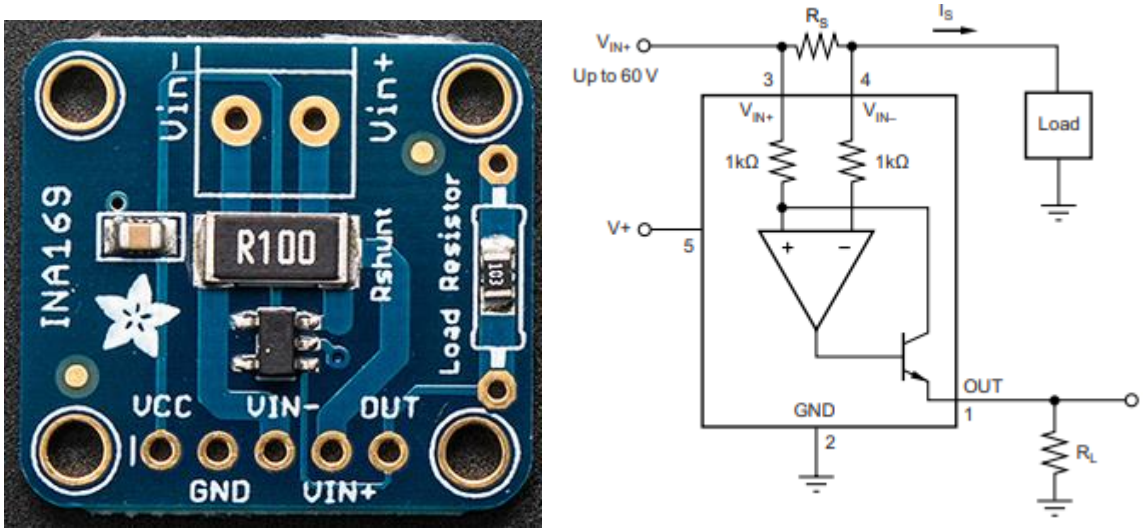


Figure 5. INA 169 appearance (left), circuit diagram (right)

Based on the designed parameter's value, the relation between the current and output voltage can be derived as:

$$\mathrm{V_o}/\mathrm{I_S} = \mathrm{R_S} * \mathrm{R_L}/1\mathrm{k}\Omega \quad \text{Voltage/Amps}$$

By changing the load resistor to a different value of resistance, the resolution can be adjusted.

4.2 Arduino ADC Technical Specification

At the current stage, the Arduino ADC is used to capture the current sensors' output. Therefore, the specification of the Arduino ADC is a key area.

By referring to the datasheet and experiment, some critical indexes are specified below.

ADC (UNO, ATMega328) Parameters	Value
Processer Frequency /Pf	16MHz
Number of Channels	6
Resolution	10 bits
Range	0-1023 / 0-5V
Conversion Clocks /#CL	13
Default Pre-scale /Ps	128
Default Sampling Frequency /Sf	10KHz
Adjustable Pre-scale	16
Adjustable Sampling Frequency	80KHz

Table 1. Arduino ADC specification

Two noticeable indexes of the Arduino ADC are the pre-scale and number of conversion clocks factor. By referring to the datasheet, this is the on-chip ADC that affected by the chip characteristics, the highest sampling frequency is given by

$$SF = \frac{\mathrm{PF}}{\mathrm{PS} * \#\mathrm{CL}}$$

Since the CPU frequency and conversion clocks are fixed, decreasing the pre-scale, the limitation of the sampling rate can be increased. However, here are several aspects are supposed to be considered.

- The pre-scale can only be changed to the exponent of 2: $1, 2, 4, 8, \ldots, 128$,
- To maintain the best resolution as 10 bits, the minimal pre-scale is 16, the corresponding sampling frequency is 80 KHz.
- The real sampling rate is slightly smaller than the theoretical one. The comparison of the experimental and theoretical will be shown in the appendix table a.3&4.

4.3 Arduino Operating Principle

4.3.1 Operating Rate

According to the objective of the project. There are multiple tasks have to be done in the single core, which is the Arduino is designed as. Additionally, as mentioned in the chart 1, four main tasks need to be achieved in individual frequency. The table 2 gives a reasonable operating rate of each task: PID computation, line detecting, current measuring, and data transferring.

Functionality	Frequency
PID Controller Computation	15 Hz
Encoder detection	About 14 KHz
Direction Adjustment	20 Hz
Arduino, RasPi current monitor	100 Hz
Motor, Battery current Monitor	10 KHz
Data transferring to RasPi	10 Hz

Table 2. multiple tasks operating rates

The one should be noticed is that the encoder's pulse per round and the gear ratio between wheel gear is large. To increase the efficiency of the programming, an interruption is used to record the number of turns that encoder has been spun. Therefore, the number of interruption will be based on the rotating speed of the motor. Given a reasonable speed, the number of interruption will be 14 KHz.

Additionally, due to the characteristic of the ESC controlling, the current appears to be periodic in a very high frequency, (169 μs, the entire current graph is shown in the chapter 5). To fully capture the current signature, a higher sampling rate should be provided. Whereas, the CPU's current tends to be steady. Therefore, their current sampling rate can be relatively lower.

As shown in the 4.3 section, 80 KHz is sufficient to fully capture the motor current. Thus, it is also sufficient for the battery's capturing.

4.3.2 Operating Cost

Functionality	Cost /#instruction
PID Controller Computation	173
Encoder detection	22
Direction Adjustment	58
Arduino, RasPi current monitor	37
Motor, Battery current Monitor	37
Data transferring to RasPi	610

Table 3. multiple tasks operating cost

The table 3 shows the number of instructions cost for each task. These values are obtained in the disassemble file of the original Arduino .ino file, by counting the number of instructions of each related function. The file will be uploaded to the GitHub

Based on the disassemble file, some instructions take 4 clock-frequency, while most of the instructions take 2 clock-frequency. However, in this table, every instruction is assumed to have 2 clock-frequency, due to the overwhelming number of instructions to count. In this regard, the real number of instruction is slightly less than the assumption that is listed in the table. Whereas,

there are some other relevant instructions, like registration, repeating operation, etc., may not be counted. therefore, the correct real number of instruction should be roughly equal to the values listed in the table.

According to the table, the dominating cost is token by the data transmission that is almost 30 times of the one token by the encoder detection. Given the Arduino's processor clocks at 16 MHz, 610 instructions theoretically take about 0.07 ms to transmit 50 data in 500000 BPS. According to the disassemble file, as for the data transmission, most of the instruction is token by opening the wire and wait for it available. The second major instruction-consuming task is the PID computation which roughly takes 0.02 ms.

Additionally, the number of the instruction can be optimized by perfecting the code by making it efficient.

4.4 Current Data Manipulation

As mentioned in the chart 1, four tasks have individual frequency to be operated. In the chart 4, a more detailed chart demonstrates how the Arduino and Pi manipulate data.

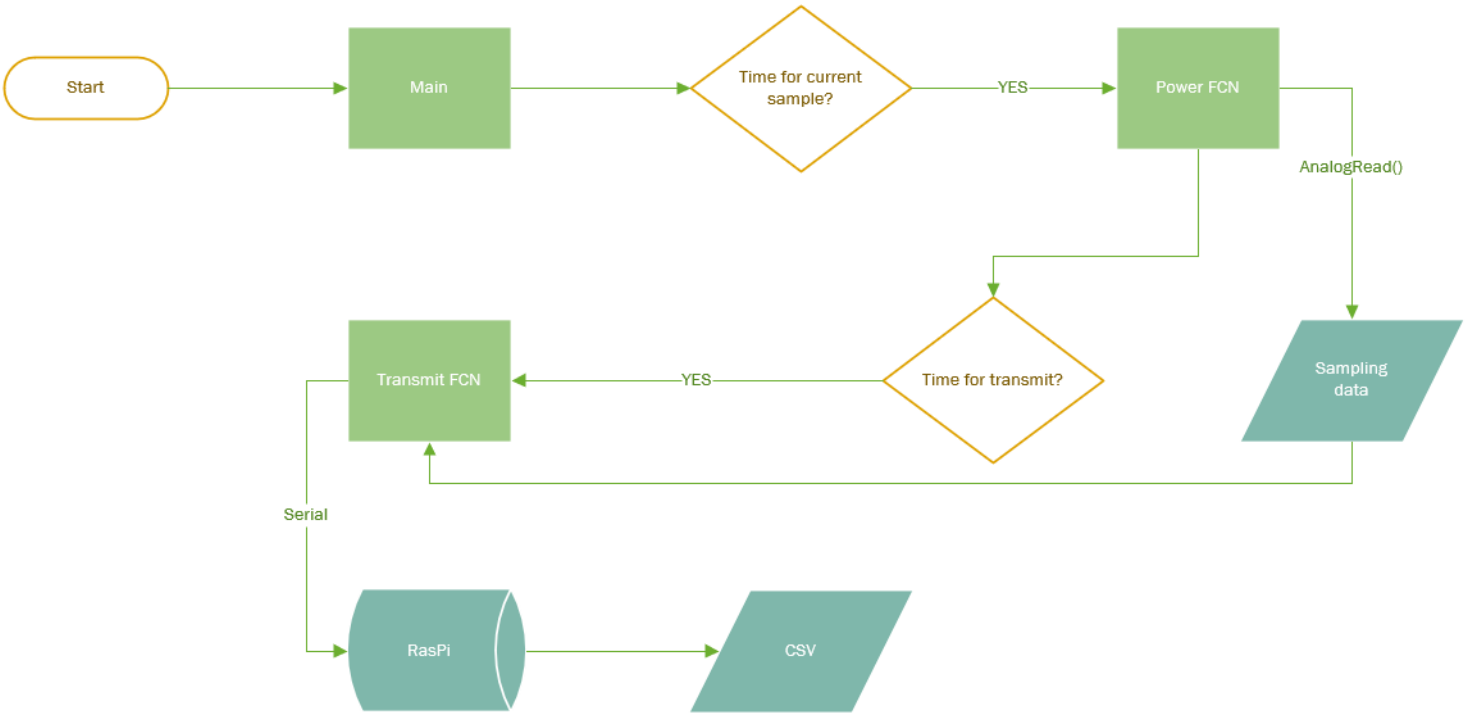


Chart 4. Current acquiring and transmitting

4.4.1 Current Sampling

When it is the time to acquire the current data, the ADC starts to work, and then stores them in an array corresponding to each measuring component. The frequencies for each component are listed in the Table 2.

4.4.2 Data Transmitting

When it is the time to transmit the data, the Arduino uses the serial to send the data to Pi. To better recognize which current measurement belongs to which component, the beginning of each data is attached an abbr. based on that component. For instance, if the current for the Pi is 0.76 A, then the data sent is P0.76.

Another noticeable aspect is the Baud Rate per Second of serial connection that largely affects the transmitting speed. The Table 4 comperes the speed difference between 9600 and 2000000 that is the maximum rate can be reached.

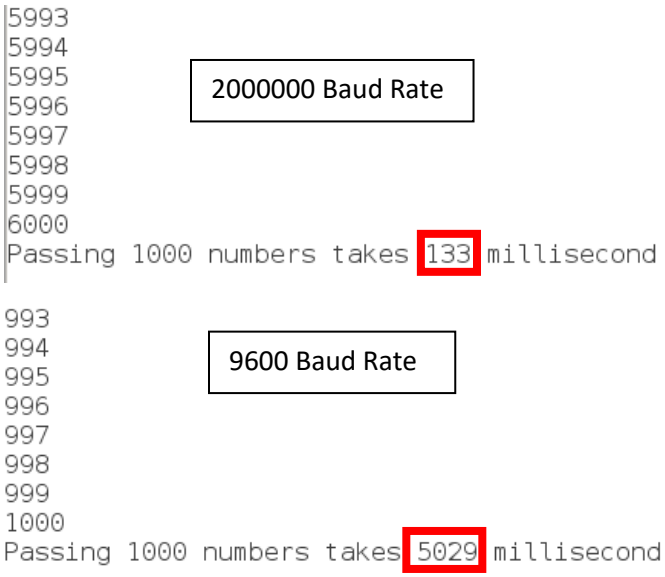


Table 4. different Baud rate causes different sending speed. 20000

The one should be noticed is that, under the same scenario it takes Arduino 6.7 ms to transmit 50 data. While as shown in the 4.3.2, theoretically it only takes 0.07 ms. The difference of 100 times cannot be ignored. This means either the counted number of instrument is wrong or there are some other issues cause this difference.

4.4.2 Pi Data Receive

Once the Pi receives the data, it will store them to the corresponding list based on the beginning alphabet. If it receives a data includes “end”, then the Pi will terminate the receiving and store them in a CSV file for future manipulation. The related code will be attached in the appendix code a.4.

5. Result

5.1 Color Sensor

Table 5 compares the value measured by the right and left color sensor, when it is above the white floor and the blue line.

right: 128.00	right: 395.00
left: 465.00	left: 133.00
right: 128.00	right: 395.00
left: 465.00	left: 133.00
right: 128.00	right: 395.00
left: 464.00	left: 132.00
right: 127.00	right: 394.00
left: 464.00	left: 132.00
right: 127.00	right: 394.00
left: 464.00	left: 132.00

Table 5. right sensor on the line (left), left sensor on the line (right)

5.2 Encoder & PID Controller

The encoder has a satisfying resolution. And it produces reliable pulses during the rotation.

The PID is quite efficient and fast-reacting if three parameters k_p, k_d, k_i are correctly set. According to the experiment, the PID controller is able to track and catch up the reference speed within 0.3 second. Additionally, it largely eliminates the sudden starting up when motor starts to rotate. In both ways, it brings a satisfying result.

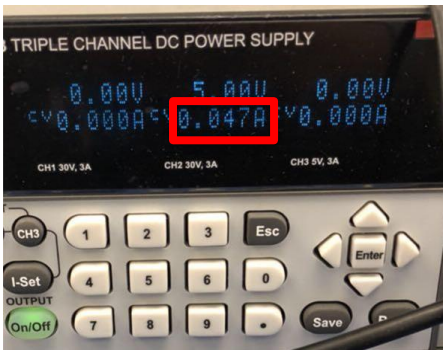
Table 6 shows the real speed the measured by the encoder and then regulated by the PID controller. The desired speed is 78 rps, the PID controller can limit the speed error within a certain level.

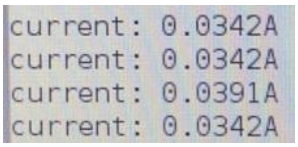
```
pid output: -0.22
motorSpeed: 79.05
escInput: 1593.41

pid output: 0.00
motorSpeed: 78.02
escInput: 1593.41

pid output: 0.91
motorSpeed: 76.22
escInput: 1594.32

pid output: -0.12
motorSpeed: 78.92
escInput: 1594.20

pid output: -0.63
motorSpeed: 80.77
escInput: 1593.57
```

Table 6. output of the PID, real motor speed, input value of ESC

5.3 Arduino Current measurement

The Arduino is a simple single core CPU. After powering up, it will continuously run the code within the '*void Loop()*' block. Therefore, if programmed properly, the Arduino never stops working. And a test was conducted when it continuously run the ADC reading program.





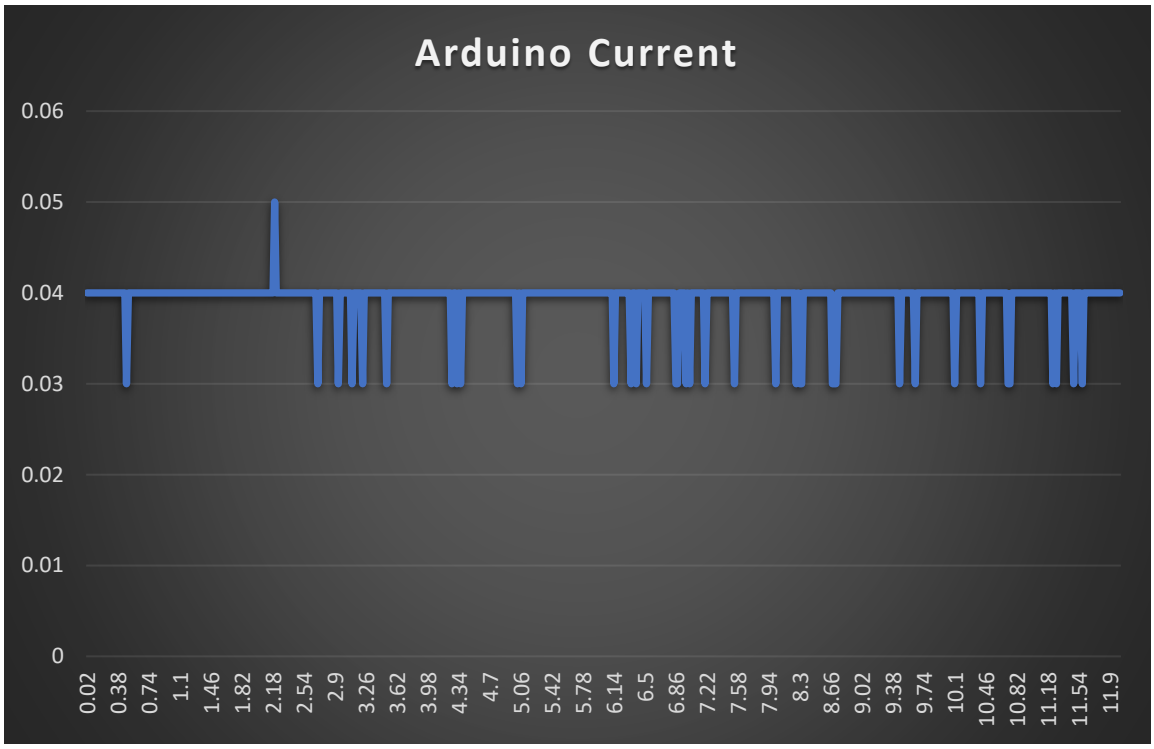


Figure 6. Arduino current measured by

DC source (top left), oscilloscope (top mid), Arduino ADC (top right)

Current vs. time (bottom)

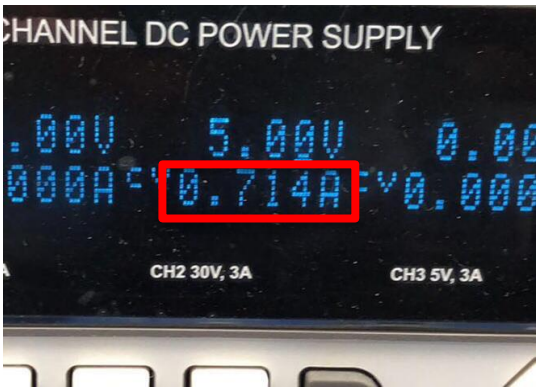
- Compared with the measurement by DC source, and oscilloscope, the result from Arduino ADC is slightly less. The reason can be explained by the fact that the current resistor is slightly less than 10 $k\Omega$, which will be shown in the appendix table a.1&2.

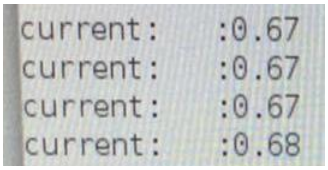
- The current flow through the Arduino is stable. Thus, the power consumption of Arduino is relatively stable along the time.

5.3 Raspberry Pi Current measurement

The Pi is a more complex four-core CPU. The power consuming highly depends on the program that is current executed.

Figure 7 shows the current through Pi when it continuously executes the image processing program. And the line chart shows the current changes during it is bosting.





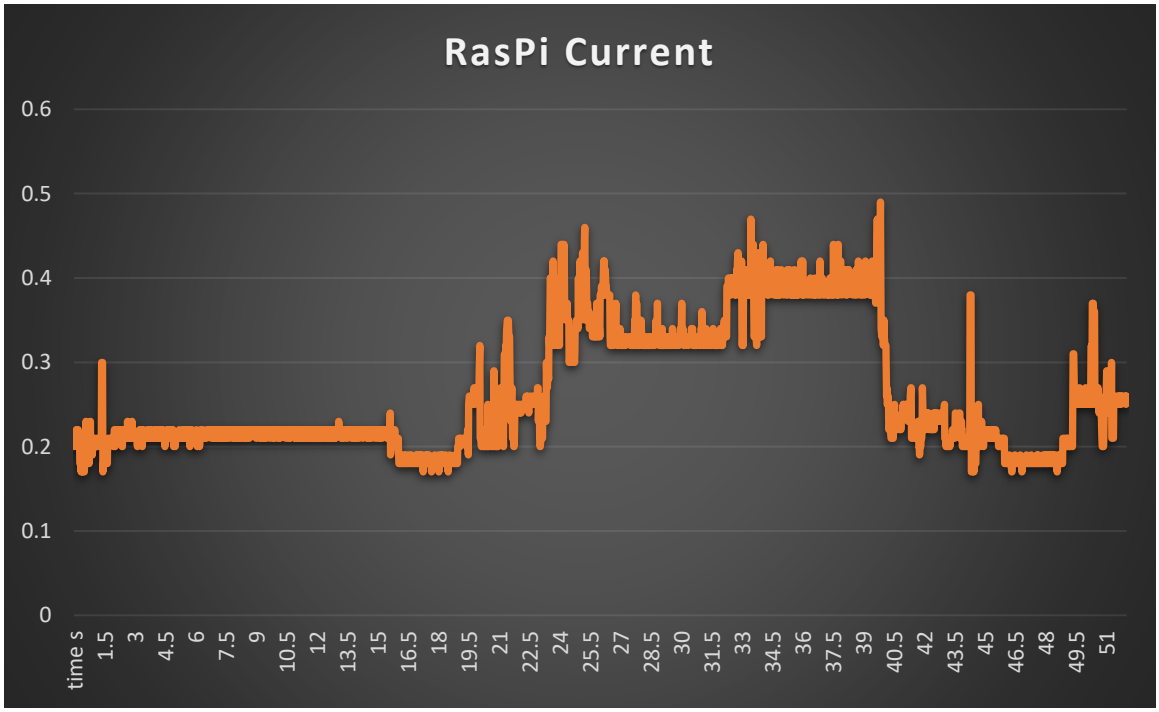


Figure 7. Pi current measured by

DC source (top left), oscilloscope (top mid), Arduino ADC (top right)

Current vs. time (bottom)

- Pi consumes much more power than the Arduino
- The current flow through the Pi largely varies during bosting
- Same reason why the ADC measured current slightly less than the other two's

5.4 Motor Current Measurement

Figure 8-10 shows the characteristic of the motor current with three types of speed that are dramatically different with the ones of micro-processors. These figures are captured by the oscilloscope, whose input is the output voltage of the current sensor.

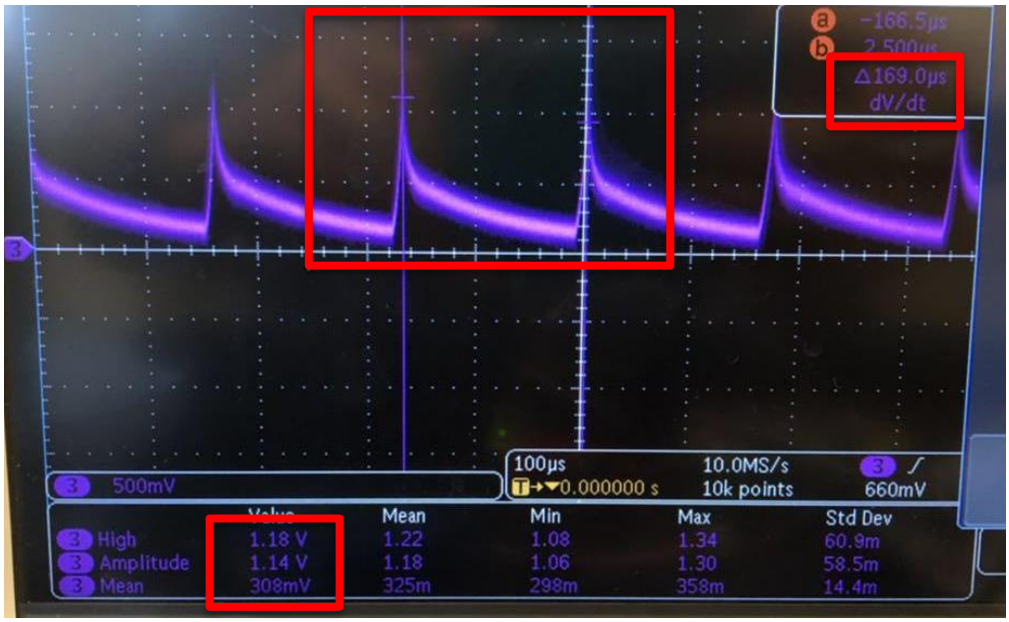


Figure 8. motor current, light loaded by encoder, rotating at 365 rps

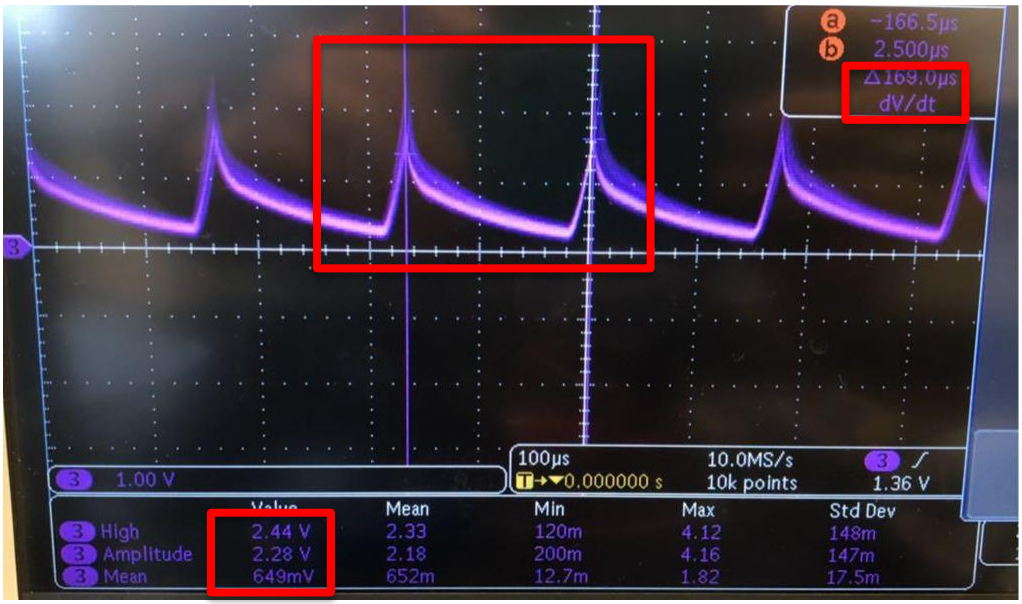


Figure 9. motor current, light loaded by encoder, rotating at 812 rps

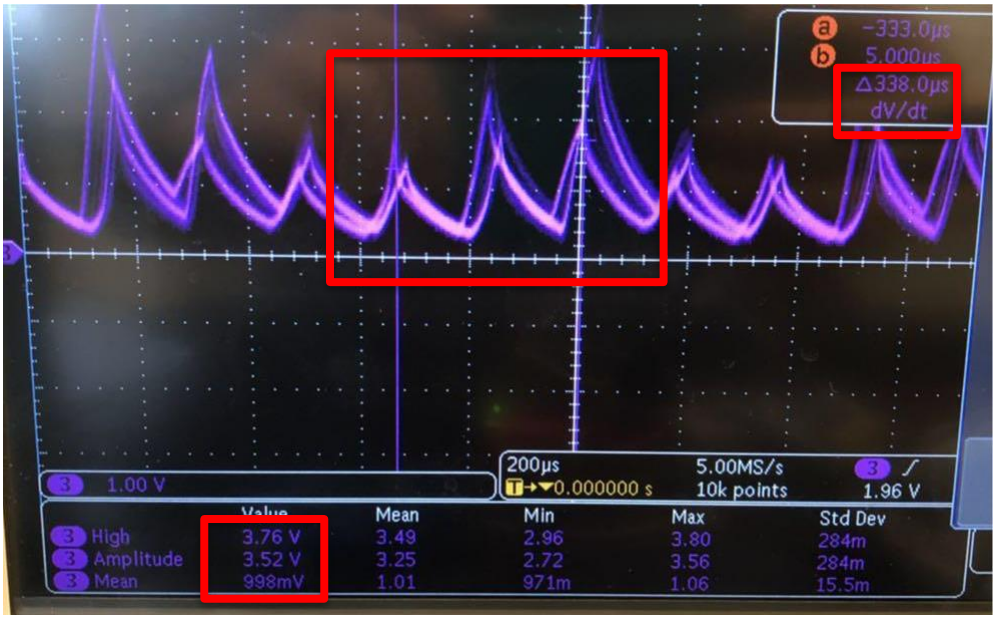


Figure 10. motor current, light loaded by encoder, rotating at 1168 rps

- No matter how much the speed that is rotating, the current changing rate always stays at 169 μs, or 5.9 KHz.
- During the experiment, the frequency of the input PWM has been changed. However, the change of the PWM frequency does not affect the ESC's current frequency, whose value stays the same from the beginning to the end.
- The speed of the motor relies on the mean value of the current. It is reasonable to assume with the normal load, the motor consumes 1-4 A, 8.4-33.6 W of power.
- By given the low speed that is what we desire in this project, the motor probably takes roughly 10 W.
- Based on figure 10, even the mean current value is 1 A, the highest value can reach to 3.76 A. Under the normal load, the highest value can beyond the range of the current sensor (5A), which is an issue should be considered. Then a larger range sensor could be an option.
- As mentioned in the 4.2, the boosted ADC 80 KHz can fully capture the motor's current. However, the current code does not guarantee the ADC to work every 125 μs, which means sometimes the ADC misses some of the data. Therefore, in this report, the ADC captured current will not be shown. However, this issue could be potentially solved by the software timer interruption, which will be shown in the future paper.

6. Conclusion

The original objective of monitoring each component's current has been basically fulfilled. On the way to achieve this objective, there are many issues to be solved, like: speed control, direction control, hardware limitation, noise signal, new way to verify the validation, etc. An effective way to conduct the experiment is to firstly divide the entire goal into tiny steps, and then break through them individually.

Because the final objective is to build a general model for the platform. Thus, the duplication would be a significant aspect. Therefore, the robustness and reliability of the platform is the most fundamental consideration. In this regard, the task that is mentioned in this report, same experiment has been conducted several times to guarantee the duplication.

- the closed loop control is important for an accurate platform.
- The color sensor guided direction system only sometimes successfully leads the car to move straightly. The potential explanation is that the servo motor does not rapidly respond the command in time, which can be observed by placing the static car on the line observing the outcome.
- Compared with motor's current, the processors' consuming current is steadier and, therefore, easier to capture.
- As a frequently moving robotic platform like the Pi Car, the motor demonstrates the power consumption.
- While when a heavy computation task is conducting by the advanced processor like the Raspberry Pi, the energy consumption cannot be neglected. Therefore, based on the type of task need to be done, a more energy-efficient strategy could be proposed.
- The low level components, like sensors and low level processor's energy consumption could be neglected when only the rough energy consumption is required.

7. Future work

Even though, the original objective has been basically fulfilled, there are still many works could be done.

- Conduct the same experiment on the different but similar platform, to see whether the similar result could be obtained.
- Improve the communication between Arduino and Raspberry Pi. The theoretical rate of SPI can reach to MHz, while the I^2C only has KHz frequency. In this way, the time spending on communication could be optimized largely.
- With the increase of the number of sensors and current sensors, 6 channels are insufficient for the task. Therefore, for saving space and easier extension, a mixed-up power board that consists of types of sensors, standalone ADC and power port is required.
- In this regard, the concept of the communication of the system is shown in figure 11.

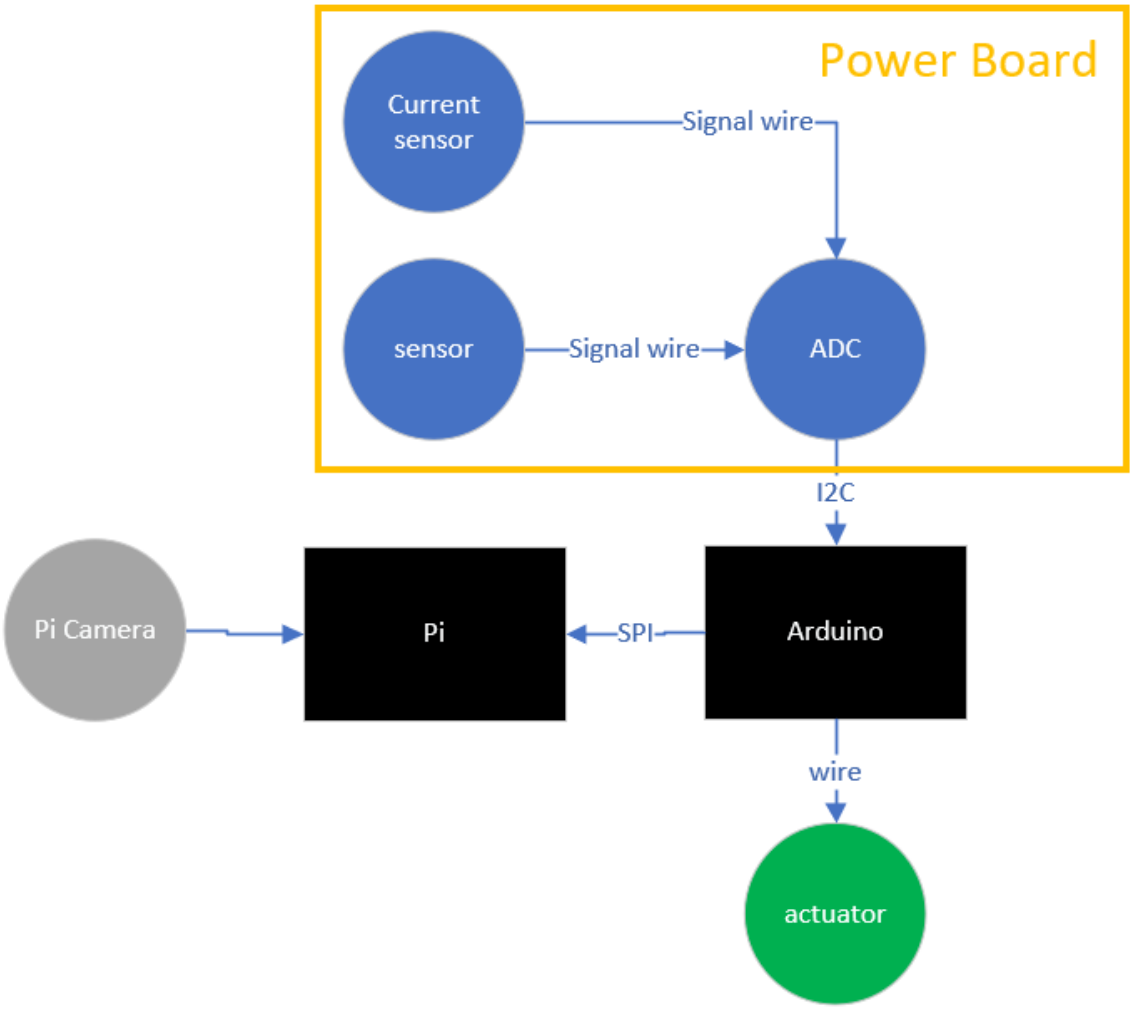


Figure 11. system internal communication wire

- Find an innovative approach to capture the high frequency motor current

- Find a more effective way to find out the accurate number of instruction of a function. Using it to guide the study of the potential of the platform.
- Instead of using the color sensor to trace the straight line, but using other sensors like Pi camera, Lidar, sonic sensor to perceive the environment and achieve various movement. This could involve with more motor speed mode and be more valuable for studying the energy consumption behavior.
- Based on the data collected from platform, a small scale robotic platform power consumption database could be built.
- Based on the database, more advanced algorithm like machine learning could be applied to optimize the power consumption.

Appendix

1. Code

```
//----------------------------PID-----------------------------------------
double pidCalculate(double in, double ref, double dt){
  double errNew, dErr, iErrNew, pidOut;
  errNew = ref - in;   //current error
  dErr = (errNew - err0ld)/dt;   //differential of error
  iErrNew = (errNew)*dt + iErr0ld;   //new integral error
  pidOut = errNew*kp + dErr*kd + iErrNew*ki;   //output of PID
  err0ld = errNew; iErr0ld = iErrNew;  //update err

  // multi step PID. errNew--->d_rps
  if (abs(errNew)>2){
  }
  else if (abs(errNew)<=2 && abs(errNew)>=1.5){
    pidOut *= 0.75;
  }
  else if (abs(errNew)<1.5 && abs(errNew)>=1){
    pidOut *= 0.5;
  }
  else if (abs(errNew)<1 && abs(errNew)>=0.5){
    pidOut *= 0.25;
  }
  else {
    pidOut = 0;
  }

  return pidOut;
}
```

Code a.1. Digital PID computation

```
//-------------------------------------------------------
//-----------------------power---------------------
//-------------------------------------------------------
void power() {
  /*
  #define batteryPowerPin A0
  #define motorPowerPin A1
  #define arduinoPowerPin A2
  #define piPowerPin A3
  */
  if (i_count == i_size) {
    i_count = 0;
    transmit();
    i_arduino[i_count] = analogRead(arduinoPowerPin);
    i_pi[i_count] = analogRead(piPowerPin);
  }
  else {
    i_arduino[i_count] = analogRead(arduinoPowerPin);
    i_pi[i_count] = analogRead(piPowerPin);
    i_count ++;
  }
}

//-------------------------------------------------------
//-----------------------transmit---------------------
//-------------------------------------------------------
void transmit() {
  for (int i=0; i<i_size; i++) {
    Serial.println(String('A') + i_arduino[i]/1024*(5/i_arduino_rez));
    Serial.println(String('P') + i_pi[i]/1024*(5/i_pi_rez));
  }
}
```

Code a.2. Current capture and transmission

```
//-------------------------------------------------------
//---------------------------direction--------------------
//-------------------------------------------------------

//-------colour sensor-----------------
void leftSensor() {
  servoAngle -= 10;
  if(servoAngle <= servoMiddle-200) {
    servoAngle = servoMiddle-200;
  }
  Serial.println("left touch");
  servo.writeMicroseconds(servoAngle);
}

void rightSensor() {
  servoAngle += 10;
  if(servoAngle >= servoMiddle+200) {
    servoAngle = servoMiddle+200;
  }
  Serial.println("right touch");
  servo.writeMicroseconds(servoAngle);
}
//-------------------------------------------------------
//---------------------------power--------------------
//-------------------------------------------------------
```

Code a.3. Servo motor direction control

Here are some of the main code for four tasks. The complete code will be uploaded to the GitHub.

2. Result

INA 169 Sensor	1	2	3	4	5	6
load resistance (k ohm)	9.98	9.96	5.6	5.6	9.96	10.01
shunt resistance (ohm)	0.45	0.48	0.52	0.45	0.62	0.61

Table a.1. measured resistance of INA 169

The shunt resistor's value could largely differ from the real one. While even the resistance of 10 cm wire's is measured as 0.7 Ω

	sensor	1	2	3	4
V=5 R=98	sensor val	45.3	45.9	47.9	46.8
	multimeter val	48.4	48.4	48.4	48.4
	computed val	50.8	50.8	50.8	50.8
	difference (a-c)/c	-0.108	-0.096	-0.057	-0.079

Table a.2. comparison of current measurement of INA 169

```
Sampling rate:    8.93KHz
Sampling rate:    8.93KHz
Sampling rate:    8.92KHz
Sampling rate:    8.93KHz
```

Table a.3. ADC sampling rate with default pre-scale of 128

```
Sampling frequency: 76.92 KHz
Sampling frequency: 76.88 KHz
Sampling frequency: 76.92 KHz
Sampling frequency: 76.88 KHz
```

Table a.4. ADC sampling rate with pre-scale of 16