

Raspberry Pi communication on Pi Car Platform

Feiyang Jin

2018-07-24

Contents

1 Introduction 2

1.1 Pi Car 2

1.2 Sensors 2

1.3 Communication protocol 2

2 Project Goal 2

3 Sensors 3

3.1 TFmini Lidar 3

3.1.1 Introduction 3

3.1.2 Main Characteristics[1] 3

3.1.3 Communication protocol[1] 3

3.1.4 Communication with Pi 4

3.2 Inertial measurement unit(LSM9DS1) 4

3.2.1 Introduction[2] 4

3.2.2 Library 4

3.2.3 Communication with Pi 4

3.3 Pi Camera 5

3.3.1 Introduction 5

3.3.2 Fast Capture 5

3.3.3 Video 5

3.3.4 Fast capture on Pi car 5

4 Raspberry Pi and Arduino UNO communication 6

4.1 Serial 6

4.1.1 Introduction 6

4.1.2 Serial on arduino uno 6

4.1.3 Serial on Raspberry Pi 6

4.1.4 Serial protocol between the two 6

4.1.5 Speed 6

4.2 I²C 6

4.2.1 Introduction 6

4.2.2 I²C on arduino 6

4.2.3 I²C on raspberry pi 6

4.2.4 I²C between the two 7

4.2.5 Speed 7

4.3 SPI 8

4.3.1 Introduction[3] 8

4.3.2 SPI on arduino UNO[4] 8

4.3.3 SPI on raspberry pi 8

4.3.4 SPI between the two 8

4.3.5 Speed 8

4.4 Multiple devices to Pi by same protocol 9

4.5 How speed is tested? 10

5 Data analysis 11

5.1 Data synchronization 11

5.2 Data visualization 12

A Raspberry Pi 3 Model B pinout 13

B Arduino UNO pinout 14

C Raspberry Pi and arduino communication protocols summary 15

D PiCar github repository Link 16

1 Introduction

1.1 Pi Car

A multi-purpose, robotic, lab-scale, open-source, wheeled, autonomous research platform created at Washington University in St. Louis

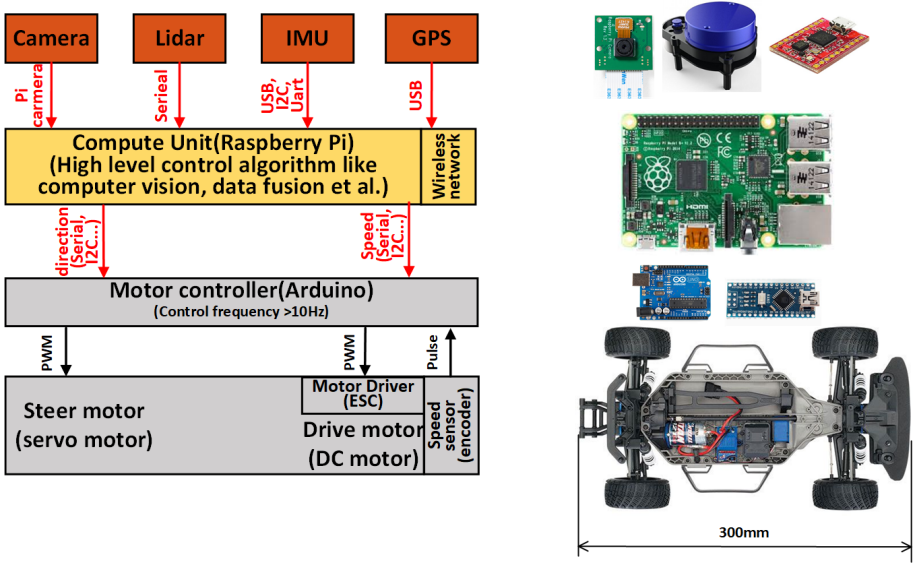


Figure 1: PiCar system architecture

1.2 Sensors

The sensors that are discussed in this report includes:

SparkFun 9DoF Sensor Stick(LSM9DS1)

Raspberry Pi Camera Module V2

TFMini - Micro LiDAR Module

For other sensors used on Pi car, please find reference on tutorial website or other reports

1.3 Communication protocol

There are three communication protocols between Raspberry Pi and arduino UNO in this report:

I²C: Inter-Integrated Circuit

SPI: Serial Peripheral Interface

Serial

2 Project Goal

1. Study the communication ports and bandwidth of the Raspberry Pi (I²C, SPI, and Serial). Compare and make a summary of these communications.
2. Read the camera, Lidar and IMU data to the Raspberry Pi.
3. Do basic data analysis.
4. Explore and compare the communication between Raspberry Pi and Arduino through I²C, SPI and Serial.

3 Sensors

3.1 TFmini Lidar

3.1.1 Introduction

Like other Lidar, TFmini Lidar give us the distance from Pi car to the obstacle ahead. The operating range of TFmini Lidar is from 0.3m to 12m, for current stage the operation range meets our need.

3.1.2 Main Characteristics[1]

Product Name	TFmini
Operating range	0.3m-12m
Maximum operating range at 10% reflectivity	5m
Average power consumption	0.12W
Applicable voltage range	4.5V-6V
Acceptance angle	2.3°
Minimum resolution ratio	5mm
Test frequency	100Hz
Test accuracy	1% (less than 6m), 2% (6m-12m)
Distance detection unit	mm
Operating center wavelength	850nm
Size	42mm×15mm×16mm
Operating temperature	-20℃-60℃
Anti-ambient light	70,000lux
Weight	6.1g
Communication interface	UART
Main applications	Drone altitude holding and terrain following Machine control and safety sensor Robot distance detection

Figure 2: TFmini Lidar main characteristic

3.1.3 Communication protocol[1]

TFMini Lidar communicates with Raspberry Pi through Serial port. More specially, the communication uses UART(Universal asynchronous receiver-transmitter) port. The baud rate for the Lidar is 115200. The following picture describes the protocol TFmini Lidar uses for output data

Byte1-2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8	Byte9
0x59 59	Dist_L	Dist_H	Strength_L	Strength_H	Reserved	Raw.Qual	Checksum_L

Figure 3: TFmini Lidar protocol

Table 1: Protocol byte explanation.

Byte	Interpretation
1	0x59, frame header, all frames are the same
2	0x59, frame header, all frames are the same
3	Low 8 bits of distance. Note: The distance value is a hexadecimal value.
4	High 8 bits of distance.
5	Low 8 bits of strength.
6	High 8 bits of strength.
7	Reserved bytes.
8	Original signal quality degree.
9	Checksum is the sum of the first 8 bytes of actual data; here is only a low 8-bit.

3.1.4 Communication with Pi

1. Raspberry pi needs to be configured to enable serial port.
In Pi configuration¹, we need to disable serial for login shell and enable serial for hardware device.
2. Wiring:

Raspberry Pi	TFmini Lidar
+5V	5V(Red)
GND	GND(Black)
TXD0	RX(White)
RXD0	TX(Green)

3. When opening the serial port on pi, we have to set the speed to 115200 to match Lidar speed.
4. As distance/strength are splitted into two bytes, the formula for calculating the two are:
 $distance = Low8bits + High8bits * 256$
 $strength = Low8bits + High8bits * 256$

3.2 Inertial measurement unit(LSM9DS1)

3.2.1 Introduction[2]

The LSM9DS1 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor.
The LSM9DS1 includes an I²C serial bus interface supporting standard and fast mode (100 kHz and 400 kHz) and an SPI serial standard interface.
Magnetic, accelerometer and gyroscope sensing can be enabled or set in power-down mode separately for smart power management.
The LSM9DS1 is guaranteed to operate within an extended temperature range from -40 celsius to +85 celsius.

3.2.2 Library

The LSM9DS1 has a fully developed C library for use. Our group also wrapped the C library into a python library for convenience. Please visit Pi Car Github repository to claim the library.

3.2.3 Communication with Pi

1. The IMU communicates Raspberry pi through I²C protocol. By default, I²C interface is not opened on Raspberry Pi, so we need to open it in pi configuration.
2. Wiring:

Raspberry Pi	IMU
+3.3V	Vcc
GND	GND
SDA	SDA
SCL	SCL

¹open the configuration by `sudo raspi-config`

3. The IMU also has a SPI interface, so if condition or requirement for the pi car changes, we can also use SPI to communicate.

3.3 Pi Camera

3.3.1 Introduction

Raspberry Pi camera gives us the ability to record what the car "see" through one experiment. The photos it provide are extremely helpful for a data analysis and logging.

3.3.2 Fast Capture

Beside standard photo mode, Pi camera is capable of capturing a sequence of images extremely rapidly by utilizing its video-capture capabilities with a JPEG encoder (via the `use_video_port` parameter).

3.3.3 Video

Pi Camera also supports recording video, and it is easy to stream the video to the internet.

3.3.4 Fast capture on Pi car

For the Pi car project, we chose to use the fast capturing mode to record photos. There are several benefits of it:

1. Change of resolution and frame rate comes out extremely straightforward for capturing photos (under fast-capture mode).
2. Photos can be easily timestamped (for our case we set time stamp as photo's name), which makes it convenient to synchronize data afterwards.
3. For future computer vision work, photo is a better choice than video.

4 Raspberry Pi and Arduino UNO communication

The three communication protocols, I²C, SPI and serial all have their own advantages and backsides. One thing to be noticed is that although the three protocols on Pi and arduino themselves have been well developed, there is still no detailed study or research that focus on the communication between the two. This report tries to give a summarize and basic analysis for the three protocols between Raspberry Pi and Arduino UNO.

4.1 Serial

4.1.1 Introduction

Serial protocol is probably the most widely studied protocol, and is very easy to connect and use.²

4.1.2 Serial on arduino uno

The code `Serial.begin(9600)` probably is the most widely known line for all arduino developer. From `Serial.write()` to `Serial.read()`, a great number of Serial functions are available from the Serial library.

4.1.3 Serial on Raspberry Pi

There are many python libraries for raspberry pi serial interface. After using and testing some of them, we recommend **pyserial** library for python programming serial port.

4.1.4 Serial protocol between the two

The most advantageous point for using serial communication is that we are able to read or write block of data by one statement. Neither I²C nor SPI could achieve this feature. Block data transmitting paves way for directly writing our own protocol. For instance, if we want to read a float on pi, after checking the header³, we can do code like `SerialRead(4)` to read all 4 bytes of the float, and afterwards unpacking them to a float number.

4.1.5 Speed

On arduino, the speed of serial is set by `Serial.begin()`. After testing, we found setting the speed to 2M is possible(although the actual speed is around 165KHZ). When we open the serial port in python on raspberry pi, we have to make sure the baud rate is the same as in `Serial.begin()` on arduino, otherwise the data will be garbled.

4.2 I²C

4.2.1 Introduction

I²C is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial computer bus invented in 1982 by Philips Semiconductor[5]. I²C requires slave and master agree on one address for communication; for pi car project, arduino is always claimed as I²C slave.

4.2.2 I²C on arduino

Wire library provides us some I²C related functions. We can set callback function for arduino being requested for data or receiving data. In this way, once the master(raspberry pi) asks for data or sends data to arduino, arduino will respond based on the callback function. Wire library also includes I²C read and write function for usage.

4.2.3 I²C on raspberry pi

The feature is disabled by default, so it needs to be enabled in pi configuration. One thing makes I²C complicated is that its speed can only be changed in Pi system file `/boot/config.txt`. As a result, i2c speed is not as flexible as serial or SPI did because their speed can be set when open the device in code.

Furthermore, as the default I²C bus to use is bus1, opening bus0 becomes necessary when some devices share the same address. However, this is not easy to do. After some research and testing, we find the best way to open bus0 is to change the I²C section under `/boot/config.txt` into

```
dtoverlay=i2c-arms
dtoverlay=i2c-vc
dtoverlay=i2c-baudrate=1000000
device_tree_param=i2c0=on
device_tree_param=i2c=on
```

²The serial here refers to connect two devices by usb cable

³header means the byte that tells pi what kind of data will be transferred

4.2.4 I²C between the two

- 1. Wiring

Raspberry Pi	Arduino
GND	GND
SDA	SDA
SCL	SCL

- 2. Different from serial communication, the I²C interface between Raspberry Pi and arduino does not support transfer of block data. This does not mean I²C protocol failed to do so, but just the interface between the two failed to. We suspect the problem is caused by arduino side. For instance, in a python library called *pigpio*, there is a I²C function called *i2c_write_block_data*. This function requires a parameter specifies the **register** to write to on arduino. However, no matter which registers we passed to it, the function would not work. Indeed, as far as we researched, all python I²C library requires a **register** parameter for reading or writing block of data, and all of them failed to work. The reason, we thought, is that arduino does not open any feature related to **register** in I²C protocol.
- 3. In order to compensate the loss of block-data transmitting, we need to write our own protocol and send multi-bytes data by byte each time. For instance, assuming we set the header for a float to 35, when raspberry pi reads a byte and finds it turn to be 35, pi knows arduino is going to sent a float(4 bytes). After that, pi will call the function *readFloat* times, which reads one byte each time. As the pi car projects goes, this way of communication works perfectly.

4.2.5 Speed

As we increase I²C speed on raspberry pi, the communication speed did get improved between the two. The max speed is around 165KHZ when we configured the speed on raspberry pi to 1.5M.

4.3 SPI

4.3.1 Introduction[3]

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems. The interface was developed by Motorola in the mid 1980s and has become a de facto standard. SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave devices are supported through selection with individual slave select (SS) lines.

4.3.2 SPI on arduino UNO[4]

Arduino has a SPI library, but most functions of the library designed for master mode. Although there is no official library, we can still set it to slave mode and use it by programming on the corresponding registers – the SPI Control Register (SPCR), the SPI Status Register (SPSR) and the SPI Data Register (SPDR).

operation	code
Set arduino as spi slave	pinMode(MISO, OUTPUT) SPCR = _BV(SPE)
Check if there is input	(SPSR & (1 <<SPIF)) != 0 ⁴
Read input	i = SPDR
Write output	SPDR = j ⁵
Interrupt	arduino SPI library provides it and works in slave mode

4.3.3 SPI on raspberry pi

SPI needs to be enabled in Pi configuration. The speed is set when opening the device in code.

4.3.4 SPI between the two

1. Similar to I²C protocol, SPI between raspberry pi and arduino does not support transfer of block data. While I²C fails because it requires specifying registers, SPI basically works incorrectly with considerable data loss when we tried to transfer block data. The reason is not yet figured out.
2. The solution for transmitting multi-bytes data is the same as I²C did: writing simple protocols between the two. However, in order to read from arduino, raspberry pi has to write some "dummy bytes" first(here the byte is 0x00) because SPI reads and writes simultaneously. As a result, on arduino side we need to be careful enough to figure out which bytes are meaningless and which bytes are data we needed.
3. Wiring

Raspberry Pi	Arduino
GND	GND
MOSI	MOSI
MISO	MISO
SCLK	SCLK
cell0/cell1	SS

4.3.5 Speed

The SPI speed between the two is limited by arduino side. From the datasheet of arduino’s chip, we find the following sentences[6]
In SPI Slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the minimum low and high periods should be longer than two CPU clock cycles
This tells us how fast the SPI speed can be on arduino when claimed as slave. Typically, arduino UNO has a clock of 16MHZ, so the highest theoretical speed is 4MHZ.
After our test, the real speed is around 670KHZ in python and 900KHZ in C++(both achieved by set SPI speed to 7.5MHZ)

⁴The SPI End of Transmission Flag (SPIF) in SPSR is set when a byte has been received.
⁵Reading or writing SPDR clears SPIF.

4.4 Multiple devices to Pi by same protocol

Sometimses we may want to connect multiple arduinos to raspberry pi(currently we just use one arduino), we have spent a little time figuring out multiple devices using the three communication protocols.

- 1. If devices' required pins are less than pi's pins, things are not complicated to deal with. The following table describes how to connect two arduinos to pi by each of the three protocols

protocol	Arduinos
I ² C	SDA,SCL connected to same pins, but claimed different address; use bus0 if necessary
SPI	MOSI,MISO,SCL connected to same pins; one arduino wires to cell0, the other one to cell1.
Serial	Connect to different USB ports

- 2. If devices require more pins than raspberry pi can provide, we need to use some GPIO(General-purpose input/output) pins as simulative port. The code is available under my github repository.

Indeed, all other communications could use simulative ports if they use serial(by RX,TX pins), I²C or SPI.

Simulative port is not as reliable as pi's built-in protocol pins, but we can save the built-in to more important devices, or connect lots of devices to pi. Furthermore, even though the simulative port is noisy, we can set filter in code to decrease the noise.

4.5 How speed is tested?

When we test speeds of the above three communication protocols, there are several things we need to consider and set carefully:

1. Make sure there is no data loss. As the speed goes higher the communications will become unreliable. As a consequence, it is necessary to check total data transferred to confirm the speed could be used.
2. How you measure speed is **important**. For example,

- (a) `startTime = currentTime`
- (b) transfer 1 byte
- (c) `endTime = currentTime`
- (d) `totalTime = endTime - startTime`
- (e) repeat the steps for n time

The sudo code may seem good to measure speed for n bytes data at first glance, but the statement a and c consume a great proportion of time as we just transfer 1 byte. To reduce the effect of getting time function, it is better to measure the time for transferring block of data.

5 Data analysis

5.1 Data synchronization

Once we have all the data and photo, we would like to synchronize them by timestamp so we can view the car drive and corresponding data simultaneously. The algorithm we currently use is very simple, as illustrated in the following figures.

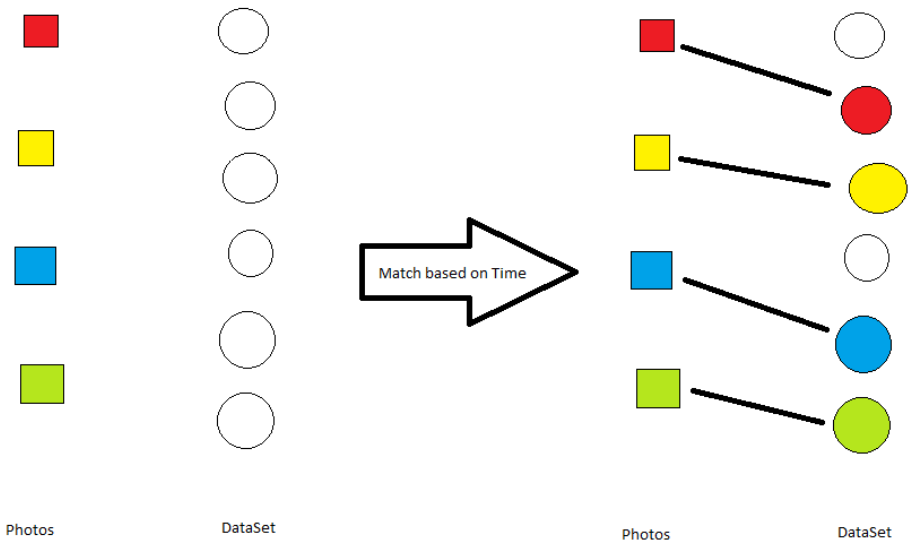


Figure 4: Pair all photos to one dataset by time

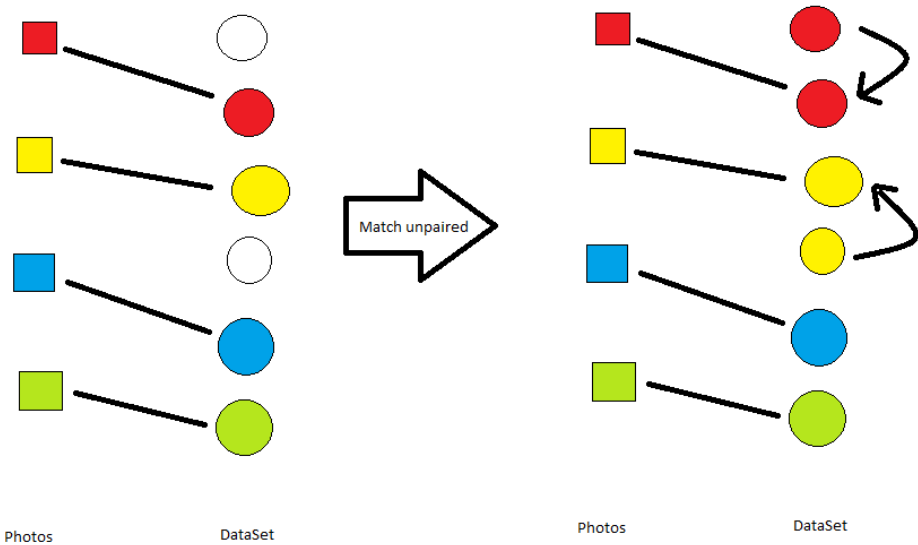


Figure 5: Match unpaired dataset

For unpaired dataset, first find its previous matched dataset A and take A's photo; if A does not exist, find its next matched dataset B and use B's photo.

5.2 Data visualization

We build a Pi car visualization platform based on html and javascript. There are two strong javascript packages we need to thank to: PapaParse.js and chart.js. PapaParse enables us to parse the synchronized data(in a csv file) from computer, and chart.js plots the graph for us. Currently, our platform supports two way of visualization: time-based display and data-based display. Time-based display will show photo and data that are paired to each other based on timestamp; data-based display will display the graph created by plotting all data(one plot for lidar, others for IMU). The two methods are explained by following figures:



Figure 6: Time-Based Display

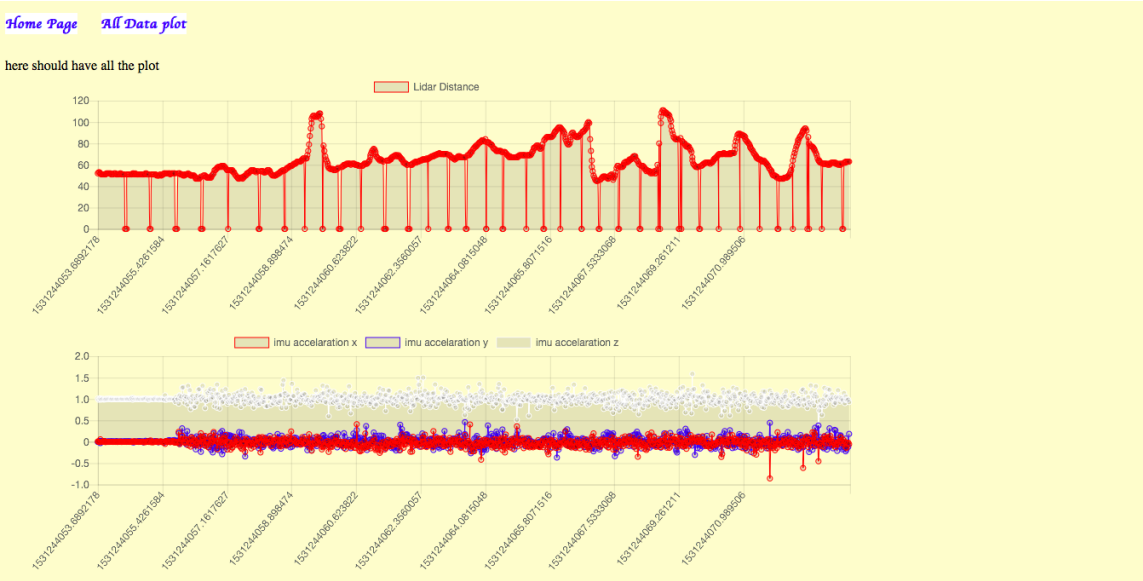




















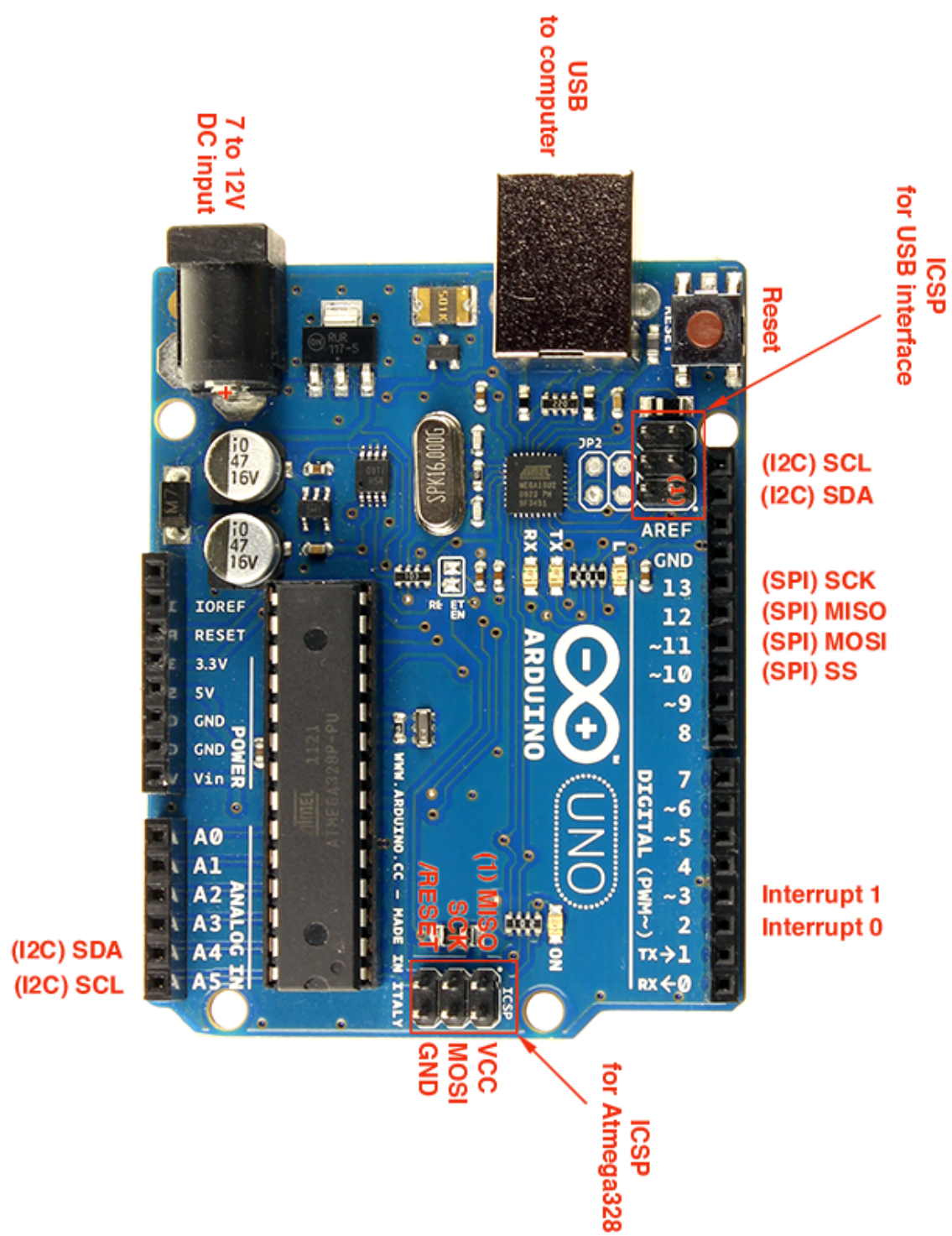


Figure 7: Data-Based Display

A Raspberry Pi 3 Model B pinout

Raspberry Pi 3 Model B (J8 Header)					
GPIO#	NAME			NAME	GPIO#
	3.3 VDC Power	1		2	5.0 VDC Power
8	GPIO 8 SDA1 (I2C)	3		4	5.0 VDC Power
9	GPIO 9 SCL1 (I2C)	5		6	Ground
7	GPIO 7 GPCLK0	7		8	GPIO 15 TxD (UART) 15
	Ground	9		10	GPIO 16 RxD (UART) 16
0	GPIO 0	11		12	GPIO 1 PCM_CLK/PWM0 1
2	GPIO 2	13		14	Ground
3	GPIO 3	15		16	GPIO 4 4
	3.3 VDC Power	17		18	GPIO 5 5
12	GPIO 12 MOSI (SPI)	19		20	Ground
13	GPIO 13 MISO (SPI)	21		22	GPIO 6 6
14	GPIO 14 SCLK (SPI)	23		24	GPIO 10 CE0 (SPI) 10
	Ground	25		26	GPIO 11 CE1 (SPI) 11
30	SDA0 (I2C ID EEPROM)	27		28	SCL0 (I2C ID EEPROM) 31
21	GPIO 21 GPCLK1	29		30	Ground
22	GPIO 22 GPCLK2	31		32	GPIO 26 PWM0 26
23	GPIO 23 PWM1	33		34	Ground
24	GPIO 24 PCM_FS/PWM1	35		36	GPIO 27 27
25	GPIO 25	37		38	GPIO 28 PCM_DIN 28
	Ground	39		40	GPIO 29 PCM_DOUT 29

B Arduino UNO pinout



C Raspberry Pi and arduino communication protocols summary

Feature	I2C	SPI	Serial
Full Form	Inter-Integrated Circuit	Serial Peripheral Interface	serial
Pin on Arduino	SDA (pin above AREF) SCL (pin above SDA)	MOSI (pin11) MISO (pin12) SCLK (pin13) SS (pin10)	usb port
Pin on Pi	SDA (pin3) SCL (pin5)	MOSI (pin19) MISO (pin12) SCLK (pin13) cell0 (pin24) and cell1 (pin26)	usb port
addressing on pi	addressing needed pi and arduino should agree on one address	Slave select lines are used to address any particular slave connected ⁶	addressing is not needed. Use specific port directly
Check address on Pi	i2cdetect -y 1 i2cdetect -y 0 ⁸	ls /dev/*spi* ⁷	lsusb:list all usb device arduino port is usually ttyACM0 or ttyACM1
Theoretical rate on Arduino	100KHZ is usually the baseline 400KHZ is fast mode	Max is 4MHZ on UNO	Recommand rate 300;600;1200;2400;4800; 9600;14400;19200;28800; 38400;57600;115200 ⁹
Set Speed on Pi	set in /boot/config.txt change the line dtparam=i2c_baudrate=	set when open in code	set when open in code
Real speed between the two	Max is around 165KHZ when pi i2c speed set to 1.5M	Python:Max is around 670KHZ when open spi around 7.5MHZ ¹⁰ C++:Max around 900KHZ	Max is around 165KHZ when we open the port by 2M baud rate
Multiple arduino to Pi	Connect same Pin on pi, but claim different address	Connect to same MOSI,MISO and SCLK, but one to cell0,and one to cell1	Connect to different usb port
Advantage	1.several device can connect to same port, just declare different address 2.easy wiring, just two wires	1.Full-duplex communication, which means read and write simultaneously 2.higher data rate, longer distance theoretically, less power	1.easy to connect and coding 2.interface between the two is well developed
Disadvantage	1.It is half-duplex	1.Not as easy as other to use 2.There is no official library to use when claim arduino as slave	1.Baud rate must be the same for two devices; otherwise data will be garbled. 2.For baud rate over 115200, debug message will be unavailable in arduino serial monitor ¹¹

⁶pi has 2 lines for spi
⁷Make sure spi is enabled on pi
⁸Make sure i2c is enabled on pi
⁹May vary from arduino version
¹⁰higher speed not reliable
¹¹minicom is a good serial terminal for monitoring the serial communication

D PiCar github repository Link

1. Feiyang Jin PiCar repository: <https://github.com/FeiyangJin/PiCar-backup>
2. PiCar repository: <https://github.com/xz-group/PiCar>

References

- [1] Benewake(Beijing), “Tfmini infrared module specification.” <https://cdn.sparkfun.com/assets/5/e/4/7/b/benewake-tfmini-datasheet.pdf>. Accessed on June 2018.
- [2] STMicroelectronics, “Lsm9ds1 datasheet.” https://cdn.sparkfun.com/assets/learn_tutorials/3/7/3/LSM9DS1_Datasheet.pdf, Nov 2014. Accessed on June 2018.
- [3] Wikipedia-contributors, “Spi — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/wiki/Serial_Peripheral_Interface, Jun 2018. Accessed on July 2018.
- [4] R. Heymsfeld, “raspberry pi to arduino spi communication.” <http://robotics.hobbizine.com/raspiduino.html>. Accessed on June 2018.
- [5] Wikipedia-contributors, “I2c — Wikipedia, the free encyclopedia.” <https://en.wikipedia.org/wiki/I%C2%B2C>, Jul 2018. Accessed on July 2018.
- [6] Atmel-Corporation, “Atmega328/p datasheet.” http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, Nov 2016. Accessed on JULY 2018.