PiCar Documentation

Release 2018

xz-group

Jan 02, 2019

Contents

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

Click on `Getting Started` tab on the sidebar, use the search-bar or use any of the following links to get started.

CHAPTER 1

Getting Started

The PiCar project is a miniature four-wheeled car powered by a Raspberry Pi 3 board. This lab-scale autonomous research platform is easy to build and modify. A camera and LIDAR mounted on the car allows for complex computer vision algorithms.

The PiCar GitHub Repository contains all the software and hardware source files required to duplicate the car, including:

- Chassis 3D printing, and CAD sources.
- Raspberry Pi 3 breakout PCB to connect peripherals and draw power from LiPo battery (v1).
- Source code for intergrating sensors like the encoder, IMU, Lidar and camera
- Source code for controlling the PiCar, networking, computer vision, etc.

The purpose of this documentation is to create a full fledged, clear formal guide for using the PiCar project. It will also serve as a place for showcasing results.

Where to begin:

1. A good place to start is to think about what you want your robot to do. The PiCar is intended to be a inexpensive way to build a robust mobile robot capable of running complex algorithms.

2. Usage -> Mechanical has a list of parts you would need to build the PiCar. It also contains instructions on how to assemble the PiCar.

Note: For your particular project, you may not need all the parts to assemble the PiCar.

- If the `Dromida Buggy` is not available, you may consider buying other similaryly sized (1/18th scale) chassis.
- If you are looking for having higher computational power, you could replace the `Raspberry Pi` with something like the `NVidia Jetson TX2`.
- You may not need the `Current Sensors` for your project.
- You could substitute the `TFMini Lidar` with a more powerful one like the `YDLidar F4` or a less powerful one like the `SR05 Ultrasonic sensor`.

3. On the software side of things, we currently use `Python 3` as the primary programming language for the `Raspberry Pi`. `Arduino` uses `Arduino C`. If you are unfamiliar with any of these hardware or software, the Tutorials section is a great place to start. The PiCar GitHub Repository contains all the code and also houses this documentation.

4. Once the PiCar has been assembled, the Usage secrtion has information about how the different modules work and how they can be controlled. It also has information about the `PiCar Module` or Class that is used for controlling any aspect of the PiCar.

5. If you would like to add your results to the Results page, kindly let us know. For contributing directly to the project, either by fixing or updating the codebase or documentation, kindly submit a GitHub pull request.

CHAPTER 2

Tutorials

To help you get acquainted with the PiCar platform, the following tutorials will help you bring to speed about the different components used.

2.1 Raspberry Pi Basics

A small crash course on setting up and using the Raspberry Pi (as a server, ftp storage, home-base, etc.)

2.1.1 What is a Raspberry Pi?

A Raspberry Pi is a credit card sized computer that can run Linux (and other OS) to do almost anything your computer can do. It can be simply connected to a monitor/TV, keyboard and mouse and be used as a regular computer. Because of its low power consumption, it is used as a web-server, file storage system, home automation system, etc. It can also be connected to an Arduino to aid in robotics such as the PiCar.

2.1.2 Setting up a Raspberry Pi with SSH

Materials Required

- Raspberry Pi 3 B+ (older models will also work)
- Pi compatible power adapter
- Micro SD Card (with alteast 16GB of memory; you may need an SD card adapter to connect it to your computer)
- USB keyboard and mouse
- Access to a monitor + HDMI cable
- Access to your router, or a new router
- A laptop for remote access to the Pi

Procedure

Note: If you are a Washington University student working on the PiCar project, you can skip to step 10 and use the given IP address of the Pi to communicate with the Pi. However it is recommended to atleast glance through the steps to see what was done.

1. Download the Raspbian Stretch with Desktop image.
2. Download Etcher
3. Install Raspbian OS to the Raspberry Pi:
   - Insert SD card into your computer.
   - Run *Etcher*.
   - In *Etcher*, choose the downloaded Raspbian zip or image file.
   - Choose the SD Card drive

     Warning: Ensure to select the correct drive (SD Card) because it will be formatted.

   - Flash the Raspbian image.
   - Eject SD card and put it into the Raspberry Pi.
4. Connect the mouse, keyboard and monitor to the Raspberry Pi. Finally connect the power cable to turn on the Raspberry Pi.
5. The default login credentials for the Raspberry Pi are:
   - username: `pi`
   - password: `raspberry`
6. Change the password to your liking by opening the terminal and typing:

```
sudo passwd pi
```

7. Use the following command to enable SSH (Secure Shell) which will be used to communicate to your computer wirelessly.

```
sudo raspi-config
```

   - Navigate to `Interfacing Options` >> `SSH` >> `Enable`
8. Connect the Pi to your router (which is connected to your company/university internet port) and reboot the Pi using:

```
sudo reboot now
```

9. Get the local IP address of your Pi using:

```
hostname -I
```

   - You will find your Pi's local IP (eg: *192.168.1.123*)
   - Alternatively you can navigate to the router admin page to check the IP addresses of connected devices.
10. On your laptop, connect to the router and use the following instructions based on your OS:

Windows:

- Download Putty
- Run *Putty*
- For the hostname, use the IP address you got for the Pi (eg: `pi@192.168.123`), and click Open

Mac/Linux:

- Open terminal and type (using the Pi's IP address):

```
ssh pi@192.168.1.123
```

11. Doing so will prompt you to enter the Pi's new password. Enter it.

Note: The default port used by Pi for SSH is 22. As long as your router and Pi password are strong, the security risk is minimized. Currently, SSH will only allow you to access the Pi when your computer and the Pi are connected to the same router.

Note: The IP addresses of devices including the Raspberry Pi may change each time you reboot it. To solve this you could either set up a static IP address for each Pi or create a way to email the IP to your email address.

2.1.3 Desktop Interface

Sometimes an terminal only interface does not suffice. We can alternatively connect to the Raspberry Pi using a VNC (Virtual Network Computing) Viewer to see the 'screen' of the Pi.

Procedure

1. Login to the Pi as usual using SSH
2. Enable VNC by using the following command:

```
sudo raspi-config
```

   - Navigate to `Interfacing Options` >> `VNC` >> `Enable`

3. Reboot the Pi

```
sudo reboot now
```

4. Install VNC Viewer on your laptop.
5. Open VNC viewer. Open a new connection: `File` >> `New Connection`
   - Use the local IP of the Pi and the SSH port (22 by default)
   - Use your credentials to login
6. You should be able to see the same screen that you saw when you initially connected to the Pi using HDMI

Note: For SSH connection to work, your laptop needs to be connected to the same WiFi (router) that the Raspberry Pi is connected to.

2.1.4 Password-less SSH

For SHH via a private computer, you can use an SSH key pair to login to the server or Raspberry Pi without a password

Procedure

1. Open a new terminal window and type the following command to generate a SSH key pair. You will keep the private key on your computer and send the public key to the server which will authenticate SSH connection without the password.

```
ssh-keygen -t rsa

* Follow through the process. If key pair has been generated previously,
choose a new file name. A passphrase is not necessary.
```

2. The following commands will create a SSH directory on the Pi, upload the generated public key to to the Pi and set the necessary permissions (replace `<Pi IP Address>`, `<Pi SSH Port (defaut: 22)>` and `<Pi username>` with their respective names/numbers):

```
ssh_ip=<Pi IP Address>
ssh_port=<Pi SSH Port (defaut: 22)>
ssh_user=<Pi username>

ssh $ssh_user@$ssh_ip -p $ssh_port mkdir -p .ssh
cat ~/.ssh/id_rsa.pub | ssh $ssh_user@$ssh_ip -p $ssh_port 'cat >> .ssh/
↪authorized_keys'
ssh $ssh_user@$ssh_ip -p $ssh_port "chmod 700 .ssh; chmod 640 .ssh/authorized_keys
↪"
```

Note: You will need to enter the SSH password for the above steps. Also, there is no space between `ssh_pi`, = and `<Pi IP Address>`, etc.

---

3. Now the SSH keys have been set up. To make the connecting via SSH even faster, do the following:

```
cat ~/.ssh/config
```

- If the `~/.ssh/config` file does not exist, create one using `nano`:

```
sudo nano ~/.ssh/config
```

- Fill it in the following format:

```
Host <some unique name>
   Hostname <Pi IP Address>
   User <Pi username>
   Port <Pi SSH Port (defaut: 22)>
```

- Or you can use the following command:

```
ssh_id=<some unique name>
echo "Host $ssh_id" >> ~/.ssh/config
echo "  Hostname $ssh_ip" >> ~/.ssh/config
echo "  User $ssh_user" >> ~/.ssh/config
echo "  Port $ssh_port" >> ~/.ssh/config
```

- Example (in `~/.ssh/config`):

```
Host pi-server
   Hostname 192.168.1.200
   User pi
   Port 22
```

- Save the file

4. Now you can SSH in to the Pi without a password using the command:

```
ssh pi-server # or whatever host identifier you chose
```

2.1.5 Resources

- Adafruit's Raspberry Pi Tutorial
- Instructables Raspberry Pi Projects

2.2 Arduino Basics

A small crash course on the Arduino micro-controller.

2.2.1 What is an Arduino?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects. We will be using the Arduino micro-controller to interface with the motor and servo(s).

2.2.2 Getting Started

Materials Required

- Arduino UNO
- UNO compatible USB cable
- A laptop or computer for programming the Arduino

Procedure

1. Download and install the Arduino IDE (Integrated development environment) for your OS
2. Launch the Arduino IDE.
3. Connect the Arduino UNO to your computer via a USB cable.
4. Go to `File` >> `Examples` >> `01. Basics` >> `Blink`
5. Choose the correct board by navigating to `Tools` >> `Board` >> `Arduino/Genuino UNO`
6. Choose the correct board by navigating to `Tools` >> `Port` >> `COMx (Arduino UNO)` on Windows or `/dev/ttyACMx` on Linux
7. Click the `Upload` button (arrow pointing to the right).

8. You should see that the on-board LED on the Arduino (pin 13) blinks every second.
9. You can also hook up an external LED to the GND and digital pin 13 to make the that LED blink.
10. Try changing `delay(1000)` to `delay(3000)` and see what happens
11. You have successfully completed your first Arduino program.

2.2.3 Todo

- Fetch sensor data
- Program servo

2.2.4 Resources

- LadyAda's Arduino tutorial
- Adafruit's Arduino tutorial
- Instructables Arduino projects

2.3 GitHub Basics

2.3.1 What is GitHub?

GitHub is a code hosting platform for version control, backups and collaboration. It lets you and others work together on projects from anywhere.

2.3.2 Getting Started

Procedure

1. If you haven't already, create a GitHub account.
2. On a browser, navigate to the PiCar GitHub Repository
3. `Fork` the repo to create a copy of the master PiCar repo on your account.
4. Under your repositories navigate to your version of the PiCar repo.
5. `Clone` (download) the repo to your computer

   Windows

   - Download and install GitHub Desktop
   - Login to your GitHub account

   Linux / Mac

   - Copy the cloning link from your forked repo.
   - Install Git:

   ```
   sudo apt-get install git
   ```

- Make a new directory and navigate to it:

```
mkdir projects/github
cd projects/github
```

- Clone the repo (replace `<username>` with your GitHub username)

```
git clone https://github.com/<username>/PiCar.git
```

6. Once you have made changes to the code or documentation, you need to commit the changes to the remote repo.

   Windows

   - *GitHub Desktop* will automatically track changes you have made to the local repo.
   - Click on the `pull` repository icon to update your local branch with the remote branch
   - Enter a commit message and hit the `Commit` button
   - Click on the `push` repository icon to update the remote branch with your local branch

   Linux / MacOS

   - Inside the PiCar repository, using terminal:

```
git pull
git add *
git commit -m "your message here"
```

   - `git pull` updates your local repo with the remote repo
   - `git add *` checks your local repo for changes and aggregates them for commits
   - `git commit` saves the new version and has a unique hash identifier
   - If you want to find that hash, you can use `git rev-parse --short HEAD` to fetch it.

Note: It is helpful to leave useful commit messages so that other contributors can see what you have done.

---

7. Push your changes to the master branch of your forked repo.

```
git push
```

   - It will prompt you for your username and password, enter them.

8. Once the local changes have been pushed to remote successfully, go back to the original Picar GitHub Repository.
9. Click on `Pull Requests` >> `New Pull Request` >> `Compare against forks`
10. The base fork should be `xz-group/PiCar`; change the headfork to `<username>/PiCar`
11. You can see what changes (additions and deletions) will be created with the Pull Request. Add a title and a short description and submit the Pull Request.
12. If you are a direct contributor on the main repo, you can navigate to Picar GitHub Repository >> `Pull Requests` >> `Merge Pull Request` as long as there are no conflicts. If you're not a direct contributor, you will need to wait until your Pull Request is merged with the master branch.

Note: Google and StackOverflow are your friends. Use them when you run into an issue with git (merge conflicts, etc.).

2.3.3 Syncing a Fork

If the main branch of the repository is ahead of your forked repo, you will need to sync your repo with the main one.

1. Navigate to the directory containing your forked repo.
2. Add a new remote called `upstream` which essentially points to the main repository:

```
git remote add upstream https://github.com/xz-group/PiCar.git
```

3. Verify the remotes:

```
git remote -v
```

   You should see something like:

```
origin     https://github.com/username/PiCar.git (fetch)
origin     https://github.com/username/PiCar.git (push)
upstream   https://github.com/xz-group/PiCar.git (fetch)
upstream   https://github.com/xz-group/PiCar.git (push)
```

4. Grab the latest version of the `upstream` remote:

```
git fetch upstream
```

5. Merge your local branch with `upstream`:

```
git checkout master
git merge upstream/master
```

Warning: If there are conflicts between your local repo due to you having changed files that have other commits in the main repo, you will have to fix those conflicts before being able to merge.

6. Push your changes to your `remote` repository:

```
git push origin master
```

7. After making changes, submit a pull request as usual.

2.3.4 Resources

- GitHub Guide
- Forking repositories
- Syncing a fork
- Pushing to remote
- Do not be afraid to commit

2.4 Linux Basics

2.4.1 What is Linux?

Linux is an open-source operating system. It is the underlying operating system on which many popular operating systems like Android, Ubuntu, Raspbian and MacOS are based on.

2.4.2 Getting started

General Linux Commands

1. SSH into a Raspberry Pi or use a computer with Ubuntu.
2. Try out the following commands in the terminal:

Make (create) directory with name `foo`

```
mkdir foo
```

List Files

```
ls
```

Change directory to `foo`

```
cd
```

Print Working Directory

```
pwd
```

Create Python script

```
nano helloworld.py
```

Note: `nano` is the simplest Terminal based editor you can use. You can also use `vi`. If you are on the Desktop (via HDMI or VNC), you can use graphical editors like `gedit` and `Atom`.

Within `helloworld.py`, type the following:

```
print("Helloworld!")
```

- Save the file using `Ctrl + X` >> `Y` >> `Enter`

Run the Python script

```
python helloworld.py
```

- This should output `HelloWorld!`

Concatenate (get contents) of a file

```
cat helloworld.py
```

- This should output `print("Helloworld!")`

Copy file `helloworld.py` to `copy_of_helloworld.py`

```
cp helloworld.py copy_of_helloworld.py
```

- Try `ls` now.

Move file (`copy_of_helloworld.py`) to new directory `bar`

```
mkdir bar
mv copy_of_helloworld.py bar/
```

Note: Sometimes, typing the entire filename or command takes too long. In cases like this you can use `Tab Completion` to quickly type the commands. You write the partial file/directory name or command and press `Tab` to complete it (or choose from possible options by double tapping `Tab`).

Note: Use the `UP Arrow Key` to use the fetch the previously used command.

Rename file (`copy_of_helloworld.py`) to (`renamed_helloworld.py`)

```
cd bar/
mv copy_of_helloworld.py renamed_helloworld.py
```

Go back a directory level

```
cd ..
```

Delete a file or directory

```
rm bar/renamed_helloworld.py
rm bar -R
```

Manual for a command

```
man rm
man sudo
```

Update and upgrade your Linux packages

```
sudo apt-get update
sudo apt-get upgrade
```

Note: `sudo` is akin to an admin. Using it will sometimes ask you to enter the user's password.

Installing a new package like `htop`

```
sudo apt-get install htop
htop
```

Note: `htop` is a great terminal way of checking how much processing power and memory your computer is using.

Pinging a website like `www.google.com`

```
ping www.google.com
```

Show network configuration

```
ifconfig
iwconfig
```

Check date

```
date
```

Clear screen

```
clear
```

Check version of an installed package

```
htop -v
```

Get local and global IP

```
hostname -I
curl ifconfig.me
```

Disk space information

```
df -h
```

Raspberry Pi Specific Commands

Check the pinouts on the Raspberry Pi

```
pinout
gpio readall
```

Lists connected USB hardware

```
lsusb
```

Show Raspberry Pi CPU Temperature

```
vcgencmd measure_temp
```

Show CPU & GPU memory split

```
vcgencmd get_mem arm && vcgencmd get_mem gpu
```

Note: If you need more than one `Terminal` open at one time, and you do not want too many new Terminal windows, you can use `Ctrl` + `Shift` + `T`.

2.4.3 Resources

- Digital Ocean's Linux Intro

- Fundamental and common Linux commands

2.5 Python Basics

2.5.1 What is Python?

Python is a programming language that lets you work quickly and integrate systems more effectively. We will be using it for programming the Raspberry Pi, data aggregation, data transfer and data anaylsis. Python 2.7 and Python 3 are the most popular versions of Python.

2.5.2 Installation

Windows

- Download and install the Python setup from Python Releases for Windows
- Download Eclipse or a similar IDE and follow this tutorial for setting up Python on Eclipse

Mac

- Python 2.7 comes pre-installed with the Mac OS X 10.8 +.
- To install and use other versions of Python on a Mac, use the tutorial on Using Python on a Macintosh

Linux

- Python (2.7 and 3.4) usually comes preinstalled with major distributions of Linux. You can test if Python is installed using the following commands in the terminal:

```
python --version
python2 --version
python3 --version
```

- If you get a message saying no command found or package is missing, you can install it using:

```
sudo apt-get install python
sudo apt-get install python3
```

2.5.3 HelloWorld with Python

Create a new file called `helloworld.py` using the IDE for Windows/Mac or using `nano` on Linux and enter the following Python code:

```
print("HelloWorld!")
```

Save and run the file. On the IDE it would be via clicking a `Run Python Script` button and via terminal you need to type `python helloworld.py`. The output should simply be the following:

```
HelloWorld!
```

2.5.4 Installing Python Modules

What makes Python so powerful is the plethora of packages made to allow a programmer do a lot of things like web-parsing, plotting, simulation, computer vision, machine learning or simply getting the weather. Use the official guide for Installing Python Packages to get things set up.

Windows

- Use the `py` Python launcher in combination with the -m switch:

```
py -2   -m pip install SomePackage  # default Python 2
py -2.7 -m pip install SomePackage  # specifically Python 2.7
py -3   -m pip install SomePackage  # default Python 3
py -3.4 -m pip install SomePackage  # specifically Python 3.4
```

Mac / Linux

- Install `pip` which is a Python Package Installer

```
sudo apt-get install python-pip
sudo apt-get install python3-pip
```

- Install Python modules using `pip`:

```
pip2 install SomePackage # short hand installation for Python 2
pip3 install SomePackage # short hand installation for Python 2

# or

python2   -m pip install SomePackage  # default Python 2
python2.7 -m pip install SomePackage  # specifically Python 2.7
python3   -m pip install SomePackage  # default Python 3
python3.4 -m pip install SomePackage  # specifically Python 3.4
```

Note: If you get an `Permission denied` while using `pip`, you can append the command with `--user`. Example: `pip install matplotlib --user`. It is not recommended to use `sudo` to install packages using `pip`.

Note: It is highly recommended to install the Python module called IPython. It significantly improves upon the vanilla version of Python command line (terminal) interface.

2.5.5 Useful Modules

The official list of useful modules does not begin to cover the vast number of modules available for different tasks, but it is a good place to start. Some of them are listed below:

Computer Vision

- **openCV** (https://pypi.org/project/opencv-python/)

Cloud Intergration

- **Amazon Web Services** (https://aws.amazon.com/python/)
- **Google Cloud** (https://googlecloudplatform.github.io/google-cloud-python/)

GUIs (Graphical User Interfaces)

- **PyGObject** (https://pygobject.readthedocs.io/en/latest/)
- **tKinter** (https://docs.python.org/2/library/tkinter.html)
- **wxPython** (https://wxpython.org/)

Data Science & Scientific Computing

- **NumPy** (http://www.numpy.org/)
- **SciPy** (https://www.scipy.org/)
- **pandas** (https://pandas.pydata.org/)
- **parquet** (https://arrow.apache.org/docs/python/parquet.html)

Interactive Python

- **IPython** (http://ipython.org/)
- **Jupyter Notebook** (http://ipython.org/)

Games & Simulations

- **Pygame** (http://www.pygame.org/news.html)
- **Pyglet** (http://www.pyglet.org/)

Machine Learning

- **TensorFlow** (https://www.tensorflow.org/install/)
- **Keras** (https://keras.io/)

Networking

- **Twisted** (https://twistedmatrix.com/trac/)

Plotting & Data-visualization

- **matplotlib** (https://matplotlib.org/)
- **seaborn** (https://seaborn.pydata.org/)
- **plotly** (https://plot.ly/)

Web Scraping

- **BeautifulSoup** (https://www.crummy.com/software/BeautifulSoup/)
- **Scrapy** (http://www.scrapy.org/)

Miscellaneous

- **pint (https://pint.readthedocs.io/en/latest/)** Define, operate and manipulate physical quantities

2.5.6 Resources

- Style Guide for Python
- Automate the Boring Stuff

2.6 Read the Docs Basics

2.6.1 What is Read the Docs?

Read the Docs simplifies software documentation by automating building,versioning, and hosting of your docs for you. We use it to keep the PiCar documentation organized and updated. If you make a significant change to the PiCar repository or project, you are recommended to update the Read the Docs documentation for PiCar. It uses reStructuredText file format to build the HTML files using Sphinx.

2.6.2 How to update the Docs?

1. Fork and clone the PiCar Github repository
2. Navigate to the `../readthedocs` directory:

```
cd PiCar/docs/readthedocs
```

3. The documentation is currently ordered as the following:

```
index.rst
conf.py
chapters/
  introduction.rst
  tutorials.rst
  usage.rst
  tutorials/
    raspberry_pi_tutorial.rst
    arduino_tutorial.rst
    github_tutorial.rst
    linux_tutorial.rst
    readthedocs_tutorial.rst
  usage/
    mechanical
    electronics
    software
  changelogs.rst
  contributors.rst
```

4. It is recommended to use a comprehensive text editor like Atom or Sublime text. Atom can be installed by:

```
sudo add-apt-repository ppa:webupd8team/atom
sudo apt-get update
sudo apt-get install atom
```

   - Atom can be launched in the `../readthedocs` directory by:

```
atom .
```

Note: For more information on the different commands available for .rst type files, check out the Rest and Sphinx memo.

5. After making a change in an .rst , go back to ../readthedocs and enter the command to build the html pages:

```
make html
```

6. To preview the changes, navigate to `../readthedocs/_build/html` and open `index.html` in a browser.

Note: The PiCar Read the Docs is using the `sphinx_rtd_theme` theme. This can be change in the `../readthedocs/conf.y` file. The version number, project name, authors, language support can be changed here too.

Warning: ReadtheDocs is very strict with indentation and formating. Check warning messages (with the associated line number) to fix issues.

7. Once you have made changes without errors and warnings and are satisfied with the updated documentation, submit a pull request to the latest Github branch.

Warning: You have to run `make html` and check the HTML output before pushing your changes, otherwise the expected HTML changes will not be rendered.

Note: If you want to create readthedocs style documentation for an entirely new repository, or you want to test and see how the HTML pages looks online, you will need to create a readthedocs account (either import your GitHub account or create a new one), and import that specific repository. This ensures that when new commits are submitted, the docs are updated automatically as well.

2.6.3 Resources

- Rest and Sphinx Memo

Note: Some of the tutorials will be in the general sense of the components, like Linux, etc., while others will specific to the PiCar project.

CHAPTER 3

Usage

The project is split into three parts:

3.1 Mechanical

The mechanical documentation involves designing, 3D printing and assembling the PiCar chassis

> **Caution:** The mechanical design and assembly of the PiCar will continue being modified over the course of the research. The following guide reflects the earliest version of PiCar v2.0.

3.1.1 Design

For the base chassis of PiCar v2, we will be using the `Dromida 1/18 Scale Buggy`. To retrofit it with sensors and micro-controllers, we will be adding some 3D printed parts.

CAD

The parts are designed using Autodesk Fusion 360. We will be splitting the chassis into three layers, connected with spacers for better management:

Layer Zero

- Dromida buggy (without cover)
- DC Motor (drive)
- Servo (steer)
- Encoder
- ESC (Electronic Speed Controller)

Layer One

- Raspberry Pi
- Arduino
- Lipo Battery(s)
- Current sensors
- IMU (Intertial Measurement Unit)

Layer Two

- Servo (LIDAR)
- TFMini LIDAR
- PiCamera

Materials Required

Component	Price ($)	Quantity	Subtotal ($)	Store Link
Dromida 1/18 Buggy 4WD RTR	99.99	1	99.99	https://www.dromida.com/surface/didc0049-bx4wd/index.php
ISC25 Rotary Encoder	39.95	1	39.95	http://www.rotaryencoder-yumo.com/products/isc25-series-solid-shaft-incremental-rotary-encoder-ID84.html
20T 48P 4mm bore Pinion Gear	6.29	1	6.29	https://www.amazon.com/dp/B00A1E19VE
Arduino UNO Rev 3	24.95	1	24.95	https://www.sparkfun.com/products/11021
Raspberry Pi 3 B+	39.95	1	39.95	https://www.sparkfun.com/products/14643
32GB MicroSD Card	12.99	1	12.99	https://www.amazon.com/dp/B06XWN9Q99
IMU 9DoF Sensor Stick	14.95	1	14.95	https://www.sparkfun.com/products/13944
Raspberry Pi Camera Module V2	29.95	1	29.95	https://www.sparkfun.com/products/14028
TrackStar 5050kv Motor + ESC	37.94	1	5.76	https://hobbyking.com/en_us/trackstar-1-18th-scale-12t-brushless-power-system-5 html
TowerPro SG90 Micro Servo	3.72	1	3.72	https://www.amazon.com/dp/B01608II3Q
TFMini - Micro LiDAR Module	39.95	1	39.95	https://www.sparkfun.com/products/14588
Turnigy 1000mAh 2S 20C LiPo	14	1	14	https://www.amazon.com/Turnigy-1000mAh-Lipo-HobbyKing-Battery/dp/B0072AEHIC
M2.5 Standoffs Assortment	11.89	1	11.89	https://www.amazon.com/gp/product/B01L06CUJG/
Current Sensors (optional)	9.95	6	59.7	https://www.digikey.com/product-detail/en/adafruit-industries-llc/1164/1528-1807-ND/6565386
		Total	404.04	

Warning: The 7.4V LiPo battery must be used with care. Use a voltmeter or battery checker to ensure that the battery voltage does not drop below 30%.

Note: If the `48P 20T 4mm bore Pinion Gear` cannot be found, buy a 48P 20T Pinion Gear and use a drill to create a 4mm bore (shaft diamater).

3.1.2 Assembly

Tools Required:

- Dremel kit (with drill and sanding bits)
- Screw drivers
- Pliers

1. Download the Fusion 360 CAD files, convert them to STL and 3D print them.

- Encoder Mount: https://a360.co/2DUNNK6
- First Layer: https://a360.co/2RquoDs
- Second Layer: https://a360.co/2OBMGmP
- Camera: https://a360.co/2QQB4dp

2. For PiCar v2.0, the `Dromida 1/18th Scale Buggy` was used:

Fig. 1: Dromida 1/18th Scale Buggy

3. Remove the plastic covering and unplug the NiMh battery. We will be using a `LiPo battery` to power the PiCar.

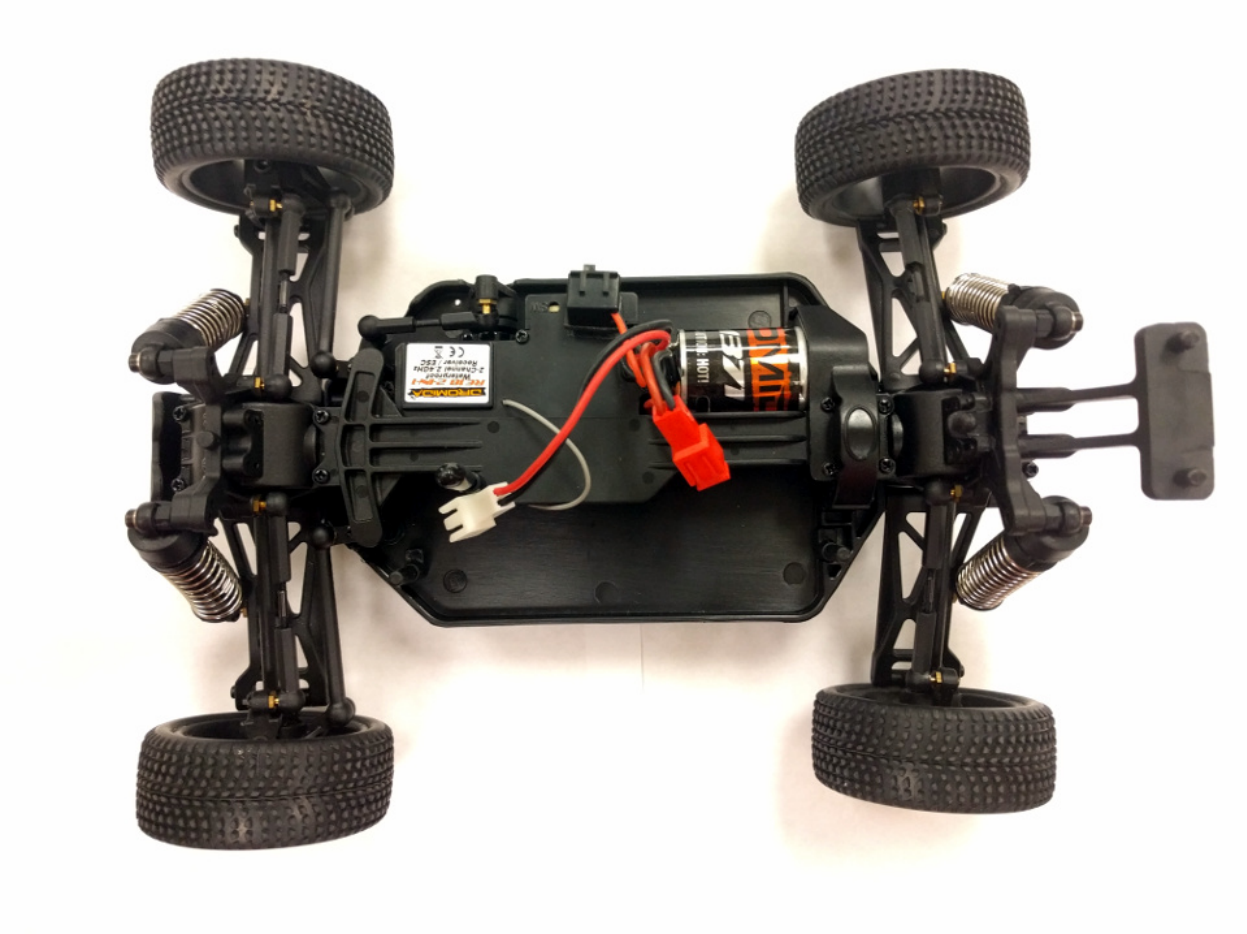


Fig. 2: Buggy with plastic casing and battery removed

4. Unscrew, and remove the rear gear covering and the plastic spline that goes along the center of the car.

5. Unscrew the plastic cover for the `ESC` (Electronic Speed Controller). Unplug the motor and servo connectors from the `ESC`. Remove the motor from the car. Do not remove the servo.

6. Unscrew the metallic motor mount. Pull out the plastic 'pillar' on the left of the rear gear.

7. Since we are using a rotary `encoder` for the low level speed controller, we need to ensure that the `encoder` meshes with the rear gear. Using the dremel and a sanding tool, carefully clear away the plastic from the gear as shown.

Ensure that the `encoder` with its `pinion gear` meshes with the rear gear and is not blocked by the plastic casing.

8. Screw in the `printed encoder mount` to the `encoder` and place it on the `chassis` as shown in the figure:

Ensure that the rear gear rotates along with the `encoder gear` with little to no friction. Holding the encoder in place, using a long narrow tipped screwdriver or nail or drill-bit, mark where the mounting holes would go. Drill 2mm holes in those points and mount the `encoder` either by using screws on the bottom of the `chassis` (recommened), or from the top.

9. Replace the `Dromida motor` with the `TrackStar Motor`. Screw the motor mount back in.

10. Now we are going to begin adding the layers that hold the electronics. Drill 2mm holes as specified in the following figure:

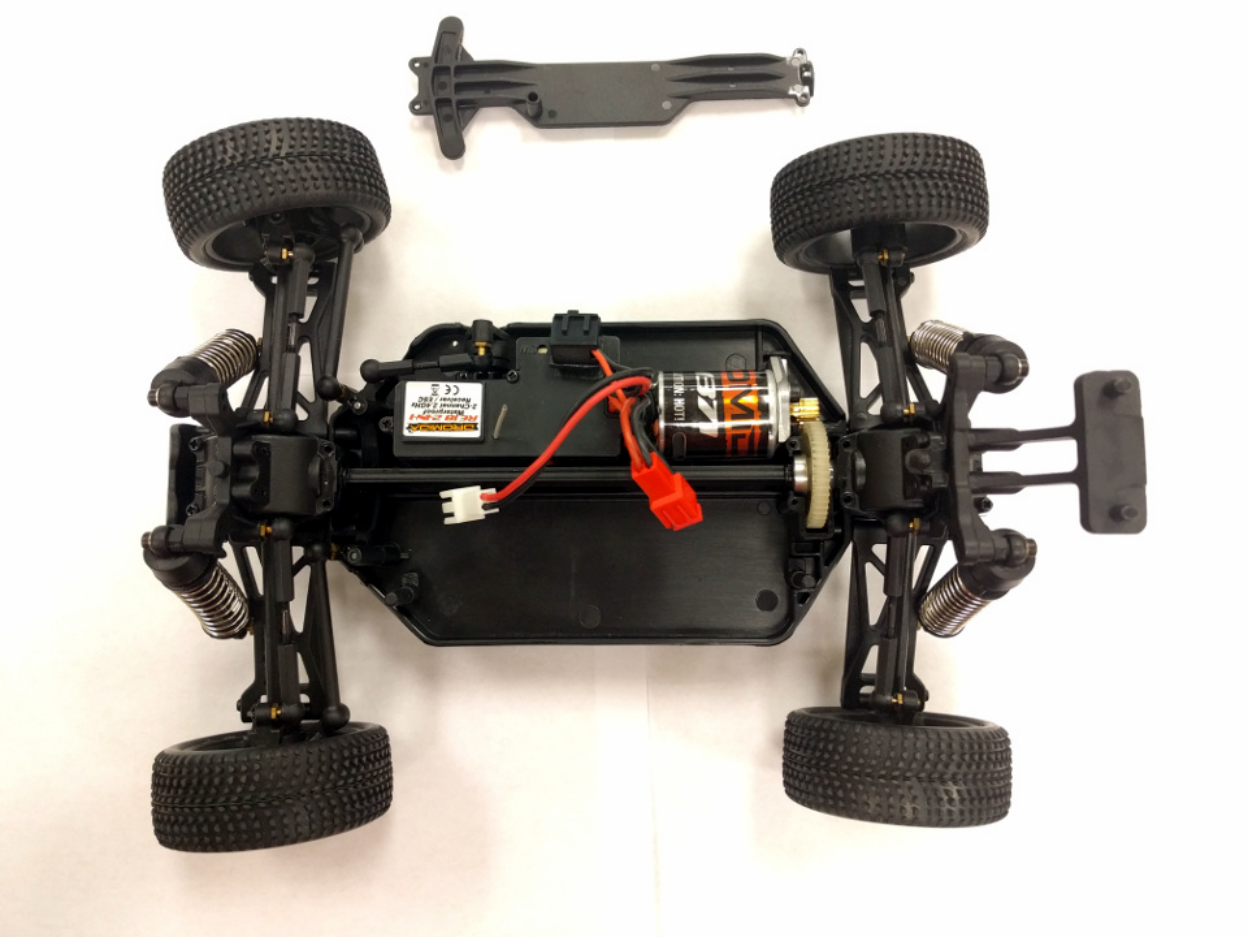


Fig. 3: Rear gear covering and spine removed

Fig. 4: ESC removed

Fig. 5: Motor removed

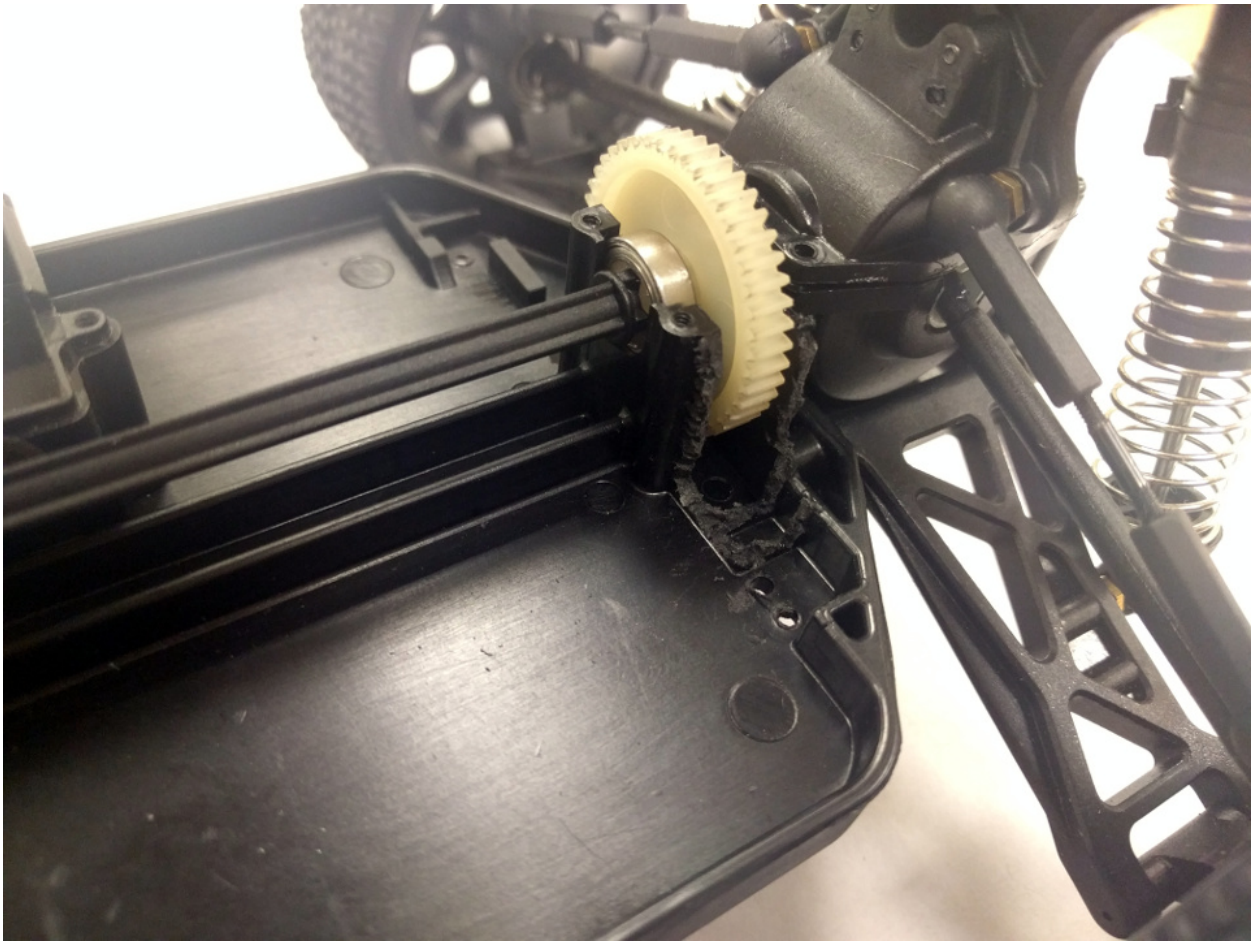


Fig. 6: Plastic cleared away for meshing Encoder

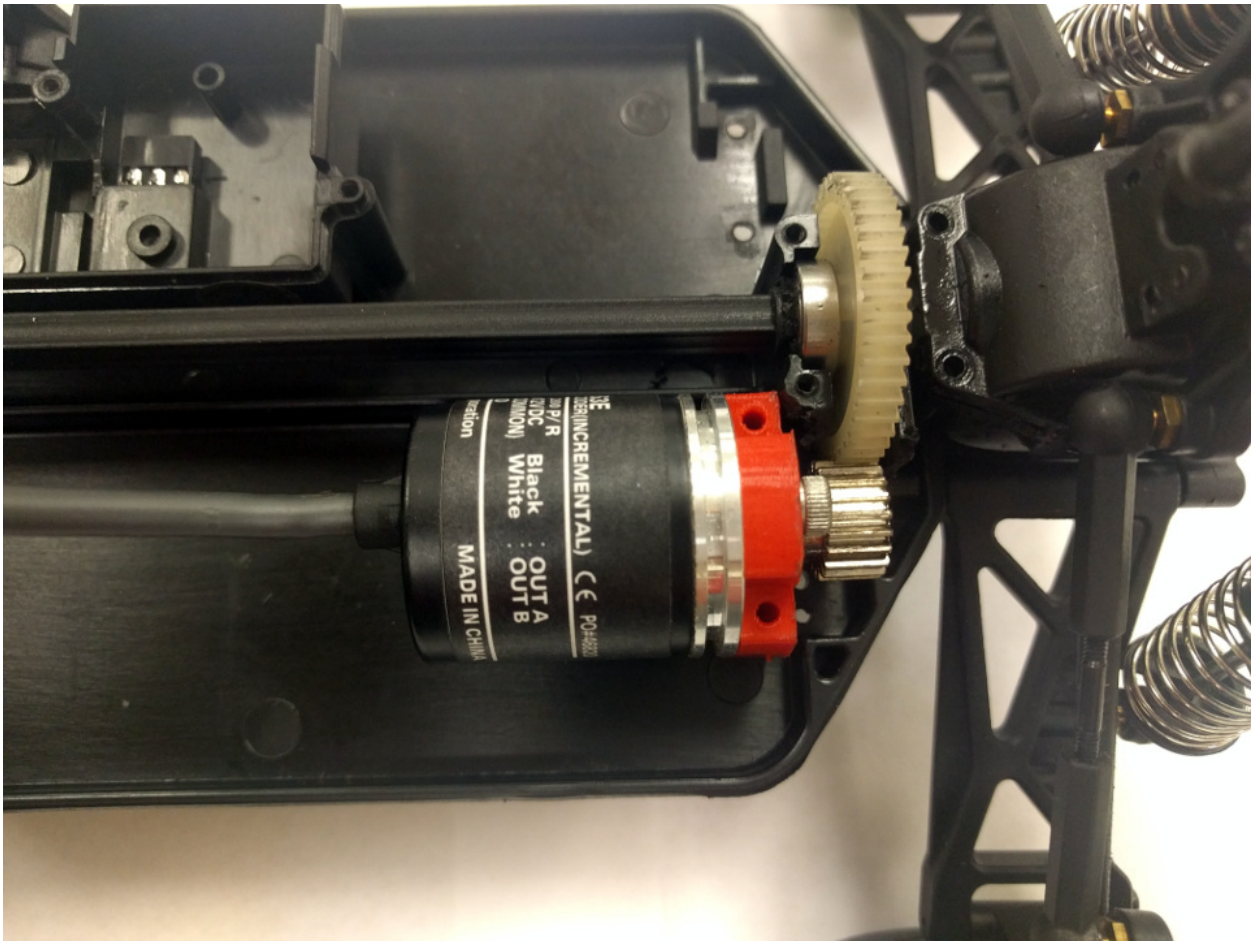


Fig. 7: Placed the encoder

Fig. 8: Replaced the default motor with the TrackStar motor.

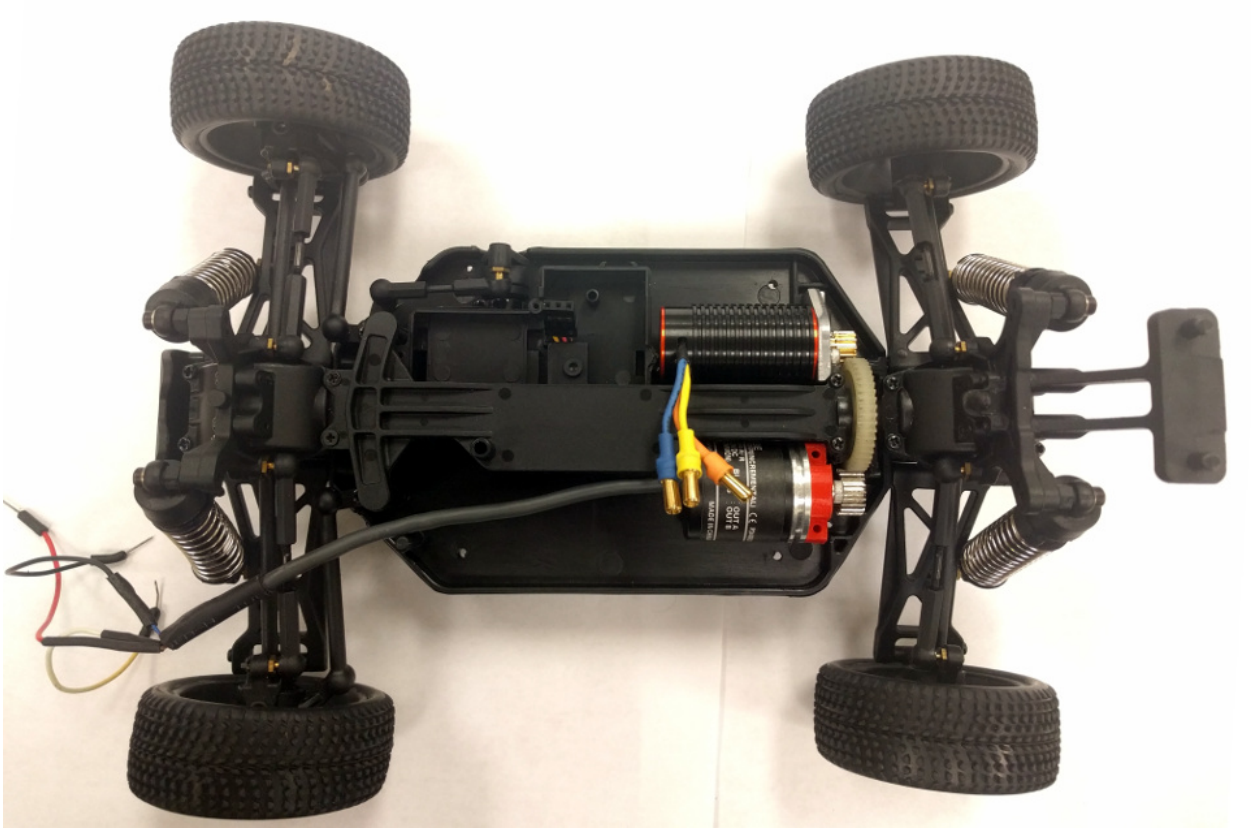


Fig. 9: Drilled holes to mount the first printed layer

Note: It may be more convenient to use the corner mounting holes as a guide to mark the locations of the holes on the base.

11. Connect the `TrackStar ESC` to the motor using the color coded wires. Reattach the spine:

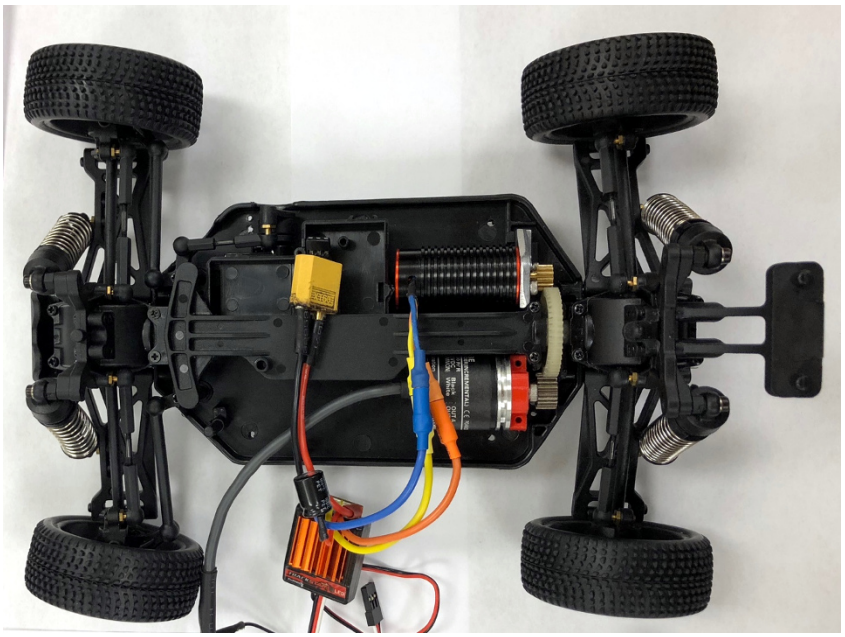


Fig. 10: ESC connected

12. Before we mount the `printed first layer` to the car using `spacers`, it may help to mount the `IMU`, `Raspberry Pi`, and the `Arduino` to the first layer.

Pre-requisites for this step:

- Create a common GND and +5V channel (we used a broken off piece from a small `breadboard`)
- Wire the `IMU` and mount it to the first layer using a screw.
- Mount the `Arduino` and `Raspberry Pi` in their respective positions using `spacers`.

Post-requisites for this step:

- Connect the `steering servo`, `ESC` and the `encoder` to the `Raspberry Pi` using usage/electronics.html
- Mount the `printed first layer` to the `chassis` using `spacers` (preferably metal ones)

13. Mount the `printed second layer` to the chassis using the `spacers`.

14. Again, using usage/electronics.html as a guide, complete the electrical assembly for the second layer.

This includes:

- Connecting a `relay` that acts as a kill switch
- Connecting the SPI / I2C communication between the `Raspberry Pi` and the `Arduino`
- Connecting the `IMU` to the `Raspberry Pi`

Now the PiCar is usable, and should look like this:

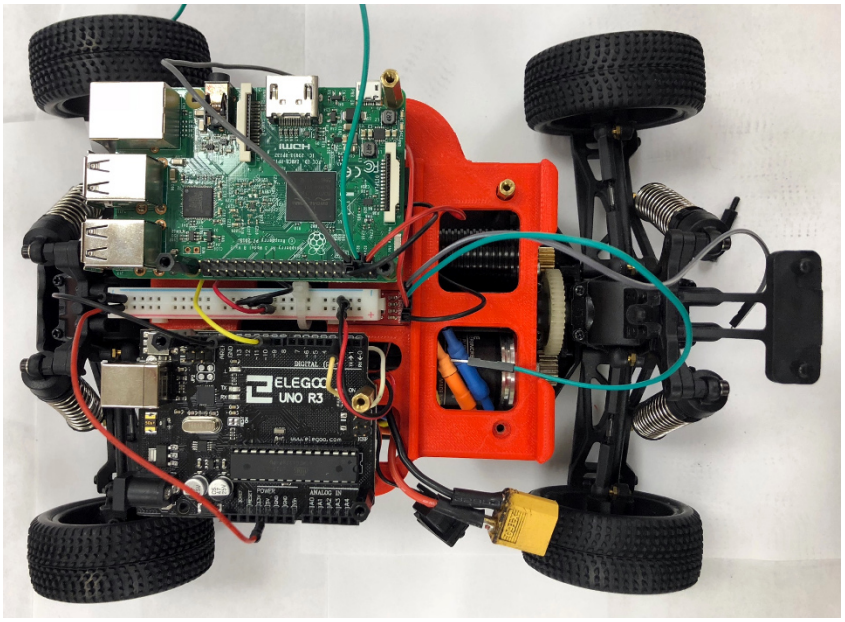


Fig. 11: First Layer Setup

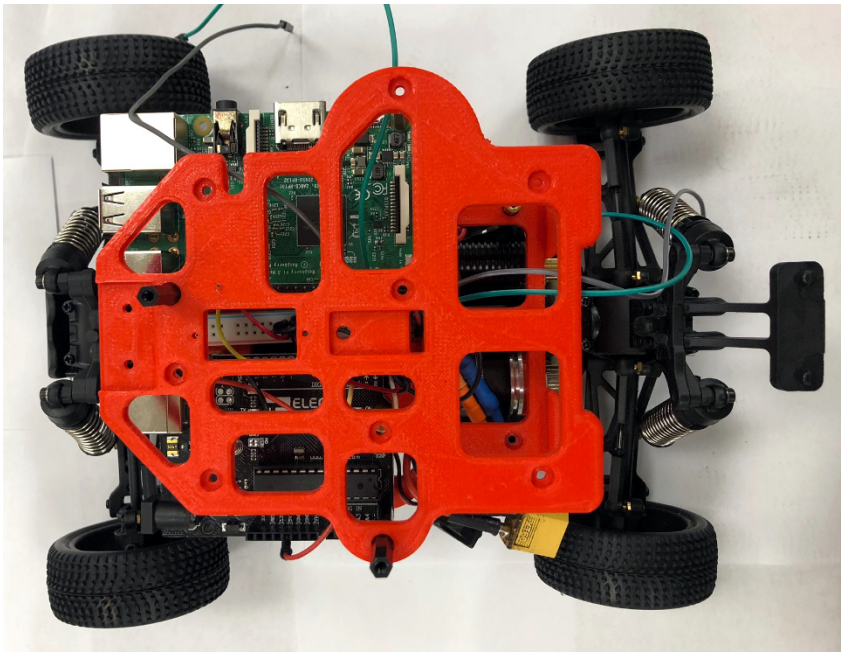


Fig. 12: Second Layer Setup

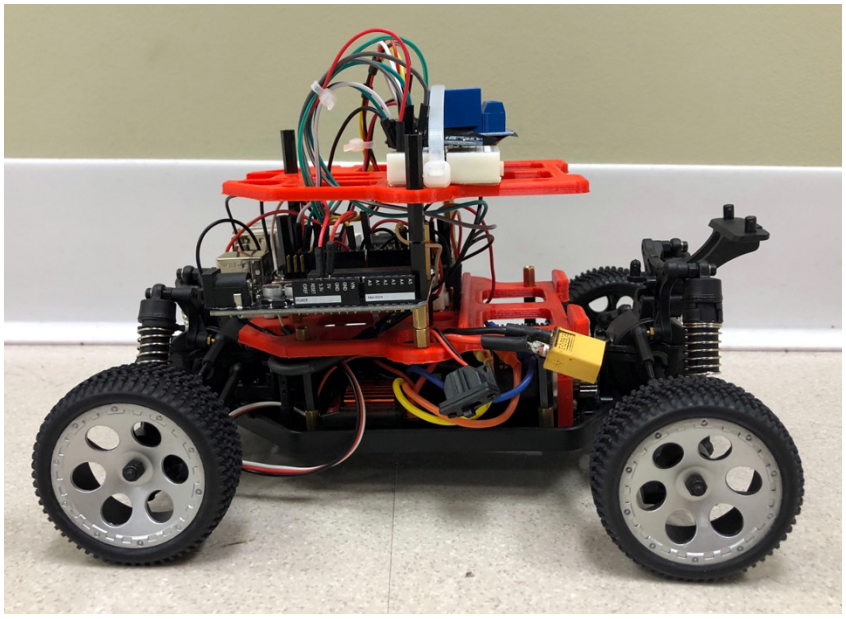


Fig. 13: PiCar: Side View

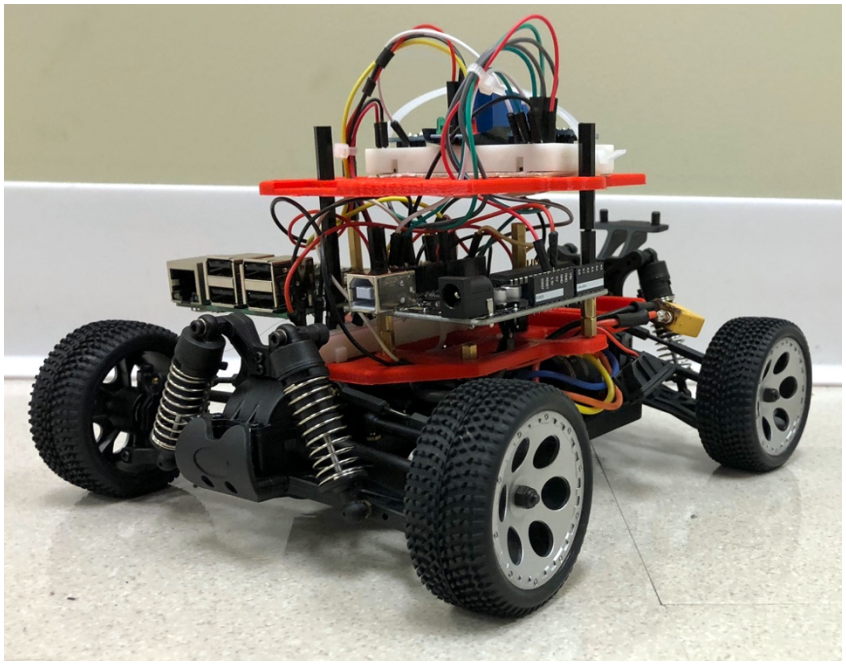


Fig. 14: PiCar: Isometric View

15. Once the base PiCar has been built, you can add the `Lidar`, `PiCamera`, etc. using the 3D printed mounts, and wire them accordingly.

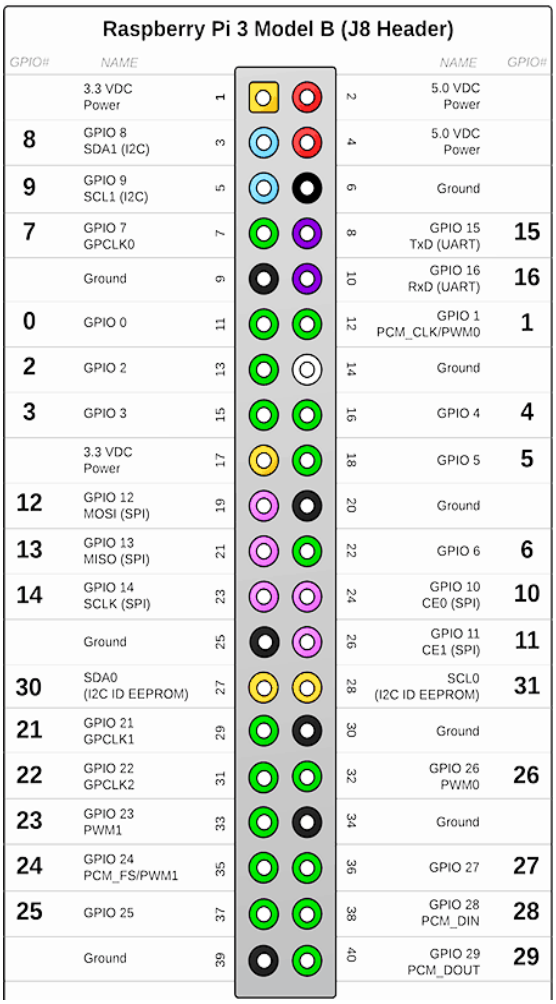
Ending notes:

- The `LiPo battery` sits in the first layer, behind the microcontrollers.
- For the time being, we are using a `compact power bank` to power the `Raspberry Pi`, which in turn powers the `Arduino` via USB.

3.2 Electronics

The electronics section will deal with interfacing the Raspberry Pi and Arduino with the sensors and actuators.

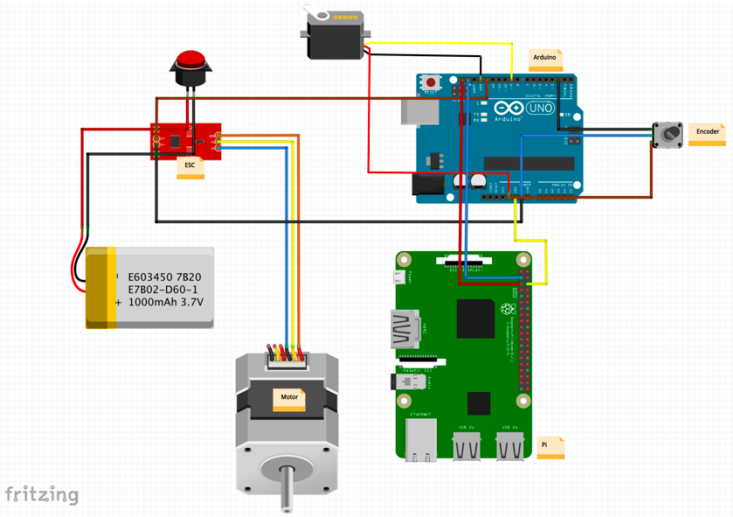
3.2.1 Raspberry Pi Pinout



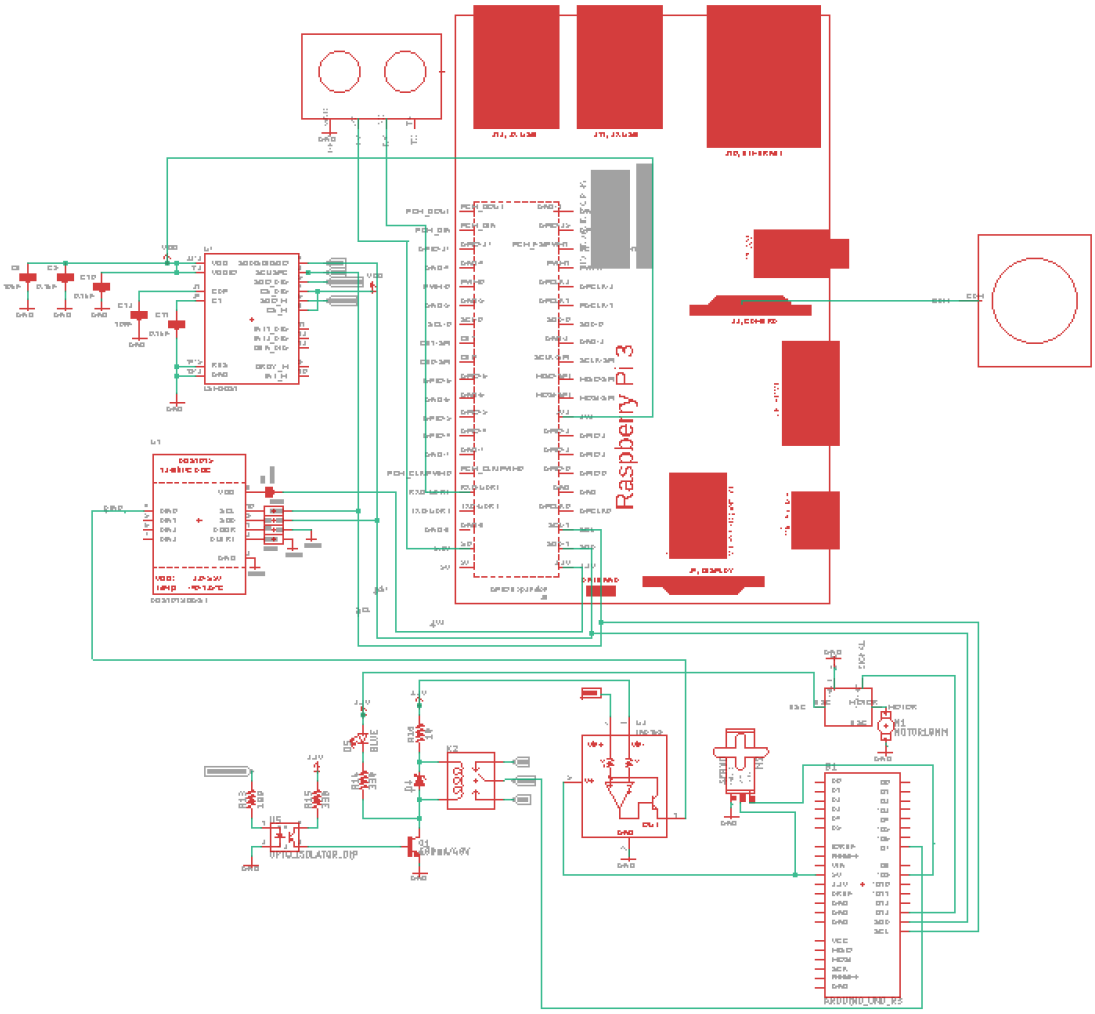
The Pi pinout image is often used as a reference because the GPIO pins on the board are often labelled.

3.2.2 Overall Circuitry

Wiring Simple



Wiring Schematics



Warning: The circuits are always prone to change, use with caution.

3.2.3 Pi and Arduino Communication

There are three commonly used to communicate between the Arduino and the Raspberry Pi.

1. I2C
2. SPI
3. Serial

Note: Most teams who used the PiCar platform recommended using Serial communication for its ease of use.

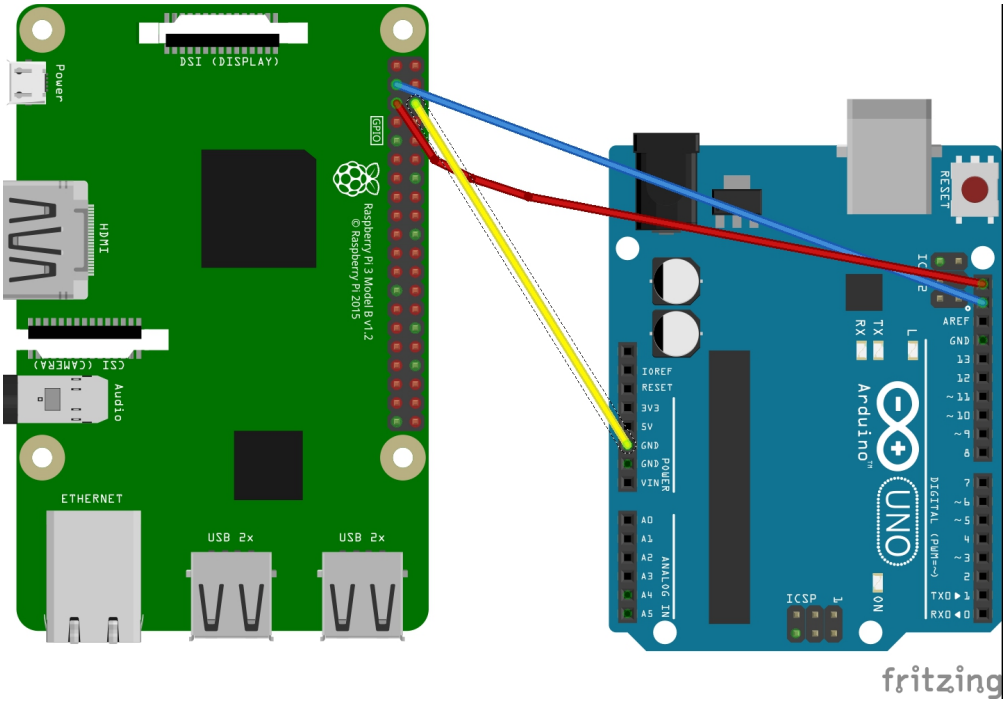
I2C Method

The code is from here, with slight changes to accommodate Python 3 instead of Python 2.

Wiring

Rasberry Pi 3	arduino Uno
GND	GND
SDA (pin3)	SDA (The pin above AREF)
SCL (pin5)	SCL (The pin above SDA)

And you can power arduino by usb on pi or on your labtop



Upload Arduino code to Arduino board

The testing code is:

```
#include <Wire.h>
#define SLAVE_ADDRESS 0x04
int number = 0;
int state = 0;

void setup() {
  pinMode(13, OUTPUT);
  Serial.begin(9600);

  // initialize i2c as slave
  Wire.begin(SLAVE_ADDRESS);

  // define callbacks for i2c communication
  Wire.onReceive(receiveData);
  Wire.onRequest(sendData);
  Serial.println("Ready!");
}

void loop() {
  delay(100);
```

(continues on next page)

(continued from previous page)

```
}

// callback for received data
void receiveData(int byteCount){
  while(Wire.available()) {
    number = Wire.read();
    Serial.print("data received: ");
    Serial.println(number);
    if (number == 1){
      if (state == 0){
        digitalWrite(13, HIGH); // set the LED on
        state = 1;
      }
      else{
        digitalWrite(13, LOW); // set the LED off
        state = 0;
      }
    }
  }
}

// callback for sending data
void sendData(){
  Wire.write(number);
}
```

Run the python code on the Raspberry Pi

The testing code is:

```
import smbus
import time

bus = smbus.SMBus(1)

# This is the address we setup in the Arduino Program
address = 0x04

def writeNumber(value):
  bus.write_byte(address, value)
  return -1

def readNumber():
  number = bus.read_byte(address)
  return number

while True:
  var = int(input("Enter 1  ^ ^  9: "))
  if not var:
      continue

  writeNumber(var)
  print("RPI: Hi Arduino, I sent you ", var)
  # sleep one second for debug
  time.sleep(1)

  number = readNumber()
```

(continues on next page)

(continued from previous page)

```
28  print("Arduino: Hey RPI, I received a digit ", number)
29  print()
```

See Also:

- SMBus Package

Tip: To open i2c bus0 on raspberry pi, you need to change the file /boot/config.txt

Under the i2c section, the txt should be

#Uncomment some or all of these to enable the optional hardware interfaces

dtparam=i2c_arm=on

dtparam=i2c_vc=on

dtparam=i2c_baudrate=1000000

#dtparam=i2s=on

device_tree_param=i2c0=on

device_tree_param=i2c=on

dtparam=spi=on

Then you can use the bus0 for i2c.

SPI Method

Wiring

Rasberry Pi 3	Arduino UNO
GND	GND
MOSI (Pin 19)	MOSI (Pin 11)
MISO (Pin 21)	MISO (Pin 12)
SCLK (Pin 23)	SCLK (Pin 13)
cell0 (Pin 24)	SS (Pin 10)

and you can choose to power the arduino using USB cable on Pi or on your laptop.

SPI on arduino

First the MISO pin has to be defined as an output pin. All other pins are configured automatically as input pins if the SPI is enabled:

```
pinMode(MISO, OUTPUT);
```

Second the SPI enable bit needs to be set:

```
SPCR |= _BV(SPE);
```

Reading and writing of SPI data is performed through SPDR. Programmatically you can treat SPDR as you would a variable. To read the contents of SDPR, it can either be accessed directly, or another variable can be set equal to it:

```
i = SPDR;
```

To load the data register with a value to transmit back to the master, the statement is reversed:

```
SPDR = i;
```

At the hardware level SPDR includes both an 8-bit shift register and an 8-bit receive buffer. When the slave is receiving data, that data is shifted into the shift register one bit at a time while the original 8-bits in the register are shifted back to the master. When a complete byte has been shifted into the register, that byte is then copied into the receive buffer. The receive buffer won't be updated again until the next complete byte is received.

Note: This means if the pi(master) wants to read from arduino(slave), it has to send something first !!

Code:

code on arduino

```
/*************************************************************
 SPI_Hello_Raspi
   Configures Arduino as an SPI slave and demonstrates
   bidirectional communication with an Raspberry Pi SPI master
*************************************************************/

#include <SPI.h>

byte c = 0;

/*************************************************************
 Setup SPI in slave mode (1) define MISO pin as output (2) set
 enable bit of the SPI configuration register
*************************************************************/

void setup (void)
{
  Serial.begin(9600);
  pinMode(MISO, OUTPUT);
  SPCR |= _BV(SPE);

}

/*************************************************************
 Loop until the SPI End of Transmission Flag (SPIF) is set
 indicating a byte has been received.  When a byte is
 received, load the byte,print it, and put 0x08 into SPDR for pi
 to read
*************************************************************/

void loop (void)
{

  if((SPSR & (1 << SPIF)) != 0)
  {
    //arduino should receive 3 and 4
    //and send 8 to pi
    c = SPDR;
    Serial.print("we received: ");
```

(continues on next page)

(continued from previous page)

```
    Serial.println(c);
    SPDR = 8;
  }

}
```

Python code on Pi(make sure you have pigpio installed and running by sudo pigpiod in terminal):

```
#!/usr/bin/env python

import time,pigpio

#open spi
pi = pigpio.pi()

if not pi.connected:
   exit(0)

h = pi.spi_open(0, 40000)


#function for communicating with arduino
def communicate():
   while True:
      #first send byts to arduino
      pi.spi_write(h,b'\x03\x04')

      #sleep 1 second and read 1 byte
      time.sleep(1)
      #pi shoudl receive 0x08, which is sent from arduino
      #spi_read returns a tuple, first is the number of bytes read,
      #second is the byte array contains the bytes
      (count,data) = pi.spi_read(h,1)
      #at the same time for reading, arduino will receive 1 byte, which is 0x00
      #Why? remember in order to read, the pi has to send something to the arduino␣
↪first !
      #By default, it will write 0 to arduino in order to read.
      print("we get %s" % data)


if __name__ == '__main__':
   try:
      communicate()
   except:
      pi.spi_close(h)
      pi.stop()
```

The arduino should continueously print 3,4 and 0 (for pi reading purpose) and pi should receive and print 0x08.

Resources

- Pi_Arduino_SPI_communication

Serial Method

Wiring

Connect arduino USB port to one of the USB port on raspberry pi

Code

The code is under `PiCar/src/Pi_Arduino_Communication/serial`

On python side, it will continuously ask you to input a float, send it to arduino.

On arduino side, once the float is sent, it will recive the data and then send it back to pi.

Difference compared with I2C and SPI

As Serial communication is well studied, we are able to send and read block of bytes on pi side.

As a result, it is much more convenient to send data more than 1 byte (discussed in next section).

Sending more than one byte between Pi and Arduino

Reason

The above basic communication (i2c,spi) allows us to send one byte between pi and arduino. However, if we want to send data that is more than one byte, such as float, the above method does not work. We first thought this is a well developed problem, and there should be easy function being called to send block of data. However, the truth is that as far as we searched, none of the proposed solution works. We come out this example for sending float between pi and arduino. If you want to develop data other than float, you are welcomed to do so.

Wiring

Same as I2C section or SPI section did

Code

The code for this is under `PiCar/src/Pi_Arduino_Communication` each subfolder(i2c,spi,serial) contains two files, .ino file should run on arduino, and .py file should run on raspberry pi.

Note: The key for communication is to write a simple protocol, and split a float into 4 bytes, so we can send 1 byte each time.

I2C by GPIO(General-purpose input/output)

Reason

Sometimes, we may want to save I2C pin to other device, or we may want to connect multiple arduino to raspberry pi. In this sections, we will use GPIO pins to connect our arduino by i2c.

Wiring

Rasberry Pi 3	Arduino UNO
GND	GND
Pin 19	SDA (pin above AREF)
Pin 13	SCL (pin above SDA)

And you can power Arduino in whatever way you want.

Code

The arduino code is the same as above (I2C section)

The following is the code on Pi, make sure you have pigpio installed and running.

```
import pigpio
import time

pi = pigpio.pi()
address = 0x04

SDA = 19
SCL = 13


def communication():

    while True:
        connection = pi.bb_i2c_open(SDA,SCL,9600)
        var = int(input("Enter 1  ^ ^  9: "))
        if not var:
            continue
        pi.bb_i2c_zip(SDA,[4,address,0x02,0x07,0x01,var,0x03,0x00])
        print("RPI: Hi Arduino, I sent you ", var)

        time.sleep(1)

        number = pi.bb_i2c_zip(SDA,[4,address,0x02,0x06,0x01,0x03,0x00])
        print("Arduino: Hey RPI, I received a digit ", number)
        print()

        pi.bb_i2c_close(SDA)


if __name__ == '__main__':
    try:
        communication()
    except:
        pi.bb_i2c_close(SDA)
```

Resources

- pigpio documentation

3.2.4 PI and TFMini Lidar Communication

Setup

To search for available serial ports, enter the following command in terminal:

```
dmesg | grep tty
```

If the output looks like:

```
pi@raspberrypi:~ $ dmesg | grep tty
[    0.000000] Kernel command line: 8250.nr_uarts=1 bcm2708_fb.fbwidth=1824 bcm2708_
↪fb.fbheight=984 bcm2708_fb.fbswap=1 dma.dmachans=0x7f35
bcm2709.boardrev=0xa02082 bcm2709.serial=0x11f38c9c bcm2709.uart_clock=48000000␣
↪smsc95xx.macaddr=B8:27:EB:F3:8C:9C vc_mem.mem_base=0x3dc00000
vc_mem.mem_size=0x3f000000  dwc_otg.lpm_enable=0 console=tty1 console=ttyS0,115200␣
↪root=/dev/mmcblk0p7 rootfstype=ext4 elevator=deadline
fsck.repair=yes rootwait splash plymouth.ignore-serial-consoles
[    0.001365] console [tty1] enabled
[    0.343313] console [ttyS0] disabled
[    0.343481] 3f215040.uart: ttyS0 at MMIO 0x3f215040 (irq = 59, base_baud =␣
↪31250000) is a 16550
[    1.078177] console [ttyS0] enabled
[    2.210431] 3f201000.uart: ttyAMA0 at MMIO 0x3f201000 (irq = 87, base_baud = 0) is␣
↪a PL011 rev2
[    3.527349] systemd[1]: Expecting device dev-ttyS0.device...
[    4.653975] systemd[1]: Starting system-serial\x2dgetty.slice.
[    4.669517] systemd[1]: Created slice system-serial\x2dgetty.slice.
```

The console needs to be disabled on the serial port `ttyAMA0`.

To do so, run the configuration command

```
sudo raspi-config
```

and navigate to option 5, Interfacing Options. Choose P6, Serial.

When prompted, answer No to “Would you like a login shell to be accessible over serial?” and Yes to “Would you like the seria port hardware to be enabled?”.

Enter the following command to reboot and search for available ports again:

```
sudo reboot
dmesg | grep tty
```

The output now should look like:

```
pi@raspberrypi:~ $ dmesg | grep tty
[    0.000000] Kernel command line: 8250.nr_uarts=1 bcm2708_fb.fbwidth=1824 bcm2708_
↪fb.fbheight=984 bcm2708_fb.fbswap=1
dma.dmachans=0x7f35 bcm2709.boardrev=0xa02082 bcm2709.serial=0x11f38c9c bcm2709.uart_
↪clock=48000000
smsc95xx.macaddr=B8:27:EB:F3:8C:9C vc_mem.mem_base=0x3dc00000 vc_mem.mem_
↪size=0x3f000000  dwc_otg.lpm_enable=0
console=tty1 root=/dev/mmcblk0p7 rootfstype=ext4 elevator=deadline fsck.repair=yes␣
↪rootwait splash plymouth.ignore-serial-consoles
[    0.001345] console [tty1] enabled
[    0.343464] 3f215040.uart: ttyS0 at MMIO 0x3f215040 (irq = 59, base_baud =␣
↪31250000) is a 16550
[    1.146776] 3f201000.uart: ttyAMA0 at MMIO 0x3f201000 (irq = 87, base_baud = 0) is␣
↪a PL011 rev2
```

Wiring

Rasberry Pi 3	TFmini
+5V	5V (RED)
GND	GND (BLACK)
TXD0 (pin8)	RX (WHITE)
RXD0 (pin10)	TX (GREEN)

Note: the white wire on TFmini Lidar is used to write command to it. If we just want to read from it, we can leave the white wire not connected.

Code

```
# tfmini.py
# supports Python 2
# prints distance from sensor

#coding: utf-8
import serial
import time
ser = serial.Serial("/dev/ttyS0", 115200)

def getTFminiData():
while True:
    count = ser.in_waiting
    #count = 0
    #print(count)
    if count > 8:
        recv = ser.read(9)
        ser.reset_input_buffer()
        if recv[0] == 'Y' and recv[1] == 'Y': # 0x59 is 'Y'
            low = int(recv[2].encode('hex'), 16)
            high = int(recv[3].encode('hex'), 16)
            distance = low + high * 256
            print('distance is: ')
            print(distance)
            time.sleep(1)

if __name__ == '__main__':
    try:
        if ser.is_open == False:
            ser.open()
            getTFminiData()
    except KeyboardInterrupt:   # Ctrl+C
        if ser != None:
            ser.close()
```

```
# tfmini_2.py
# supports Python 2 or Python 3
# prints distance and strength from sensor

#coding: utf-8
```

(continues on next page)

(continued from previous page)

```
import serial
import time

ser = serial.Serial("/dev/ttyS0", 115200)

def getTFminiData():
    while True:
        #time.sleep(0.1)
        count = ser.in_waiting
        if count > 8:
            recv = ser.read(9)
            ser.reset_input_buffer()
            # type(recv), 'str' in python2(recv[0] = 'Y'), 'bytes' in python3(recv[0] = 89)
            # type(recv[0]), 'str' in python2, 'int' in python3

            if recv[0] == 0x59 and recv[1] == 0x59:     #python3
                distance = recv[2] + recv[3] * 256
                strength = recv[4] + recv[5] * 256
                print('(', distance, ',', strength, ')')
                ser.reset_input_buffer()

            if recv[0] == 'Y' and recv[1] == 'Y':     #python2
                lowD = int(recv[2].encode('hex'), 16)
                highD = int(recv[3].encode('hex'), 16)
                lowS = int(recv[4].encode('hex'), 16)
                highS = int(recv[5].encode('hex'), 16)
                distance = lowD + highD * 256
                strength = lowS + highS * 256
                print(distance, strength)

            # you can also distinguish python2 and python3:
            #import sys
            #sys.version[0] == '2'    #True, python2
            #sys.version[0] == '3'    #True, python3


if __name__ == '__main__':
    try:
        if ser.is_open == False:
            ser.open()
        getTFminiData()
    except KeyboardInterrupt:   # Ctrl+C
        if ser != None:
            ser.close()
```

Use GPIO pin for reading

If we connect TX (green wire on TFmini Lidar) to the GPIO pin23, we can use it as a simulative port and read from it.

```
# -*- coding: utf-8 -*
import pigpio
import time

RX = 23
```

(continues on next page)

(continued from previous page)

```
pi = pigpio.pi()
pi.set_mode(RX, pigpio.INPUT)
pi.bb_serial_read_open(RX, 115200)

def getTFminiData():
  while True:
    #print("############")
    time.sleep(0.05)  #change the value if needed
    (count, recv) = pi.bb_serial_read(RX)
    if count > 8:
      for i in range(0, count-9):
        if recv[i] == 89 and recv[i+1] == 89: # 0x59 is 89
          checksum = 0
          for j in range(0, 8):
            checksum = checksum + recv[i+j]
          checksum = checksum % 256
          if checksum == recv[i+8]:
            distance = recv[i+2] + recv[i+3] * 256
            strength = recv[i+4] + recv[i+5] * 256
            if distance <= 1200 and strength < 2000:
              print(distance, strength)
            #else:
              # raise ValueError('distance error: %d' % distance)
            #i = i + 9

if __name__ == '__main__':
  try:
    getTFminiData()
  except:
    pi.bb_serial_read_close(RX)
    pi.stop()
```

In this way, we can save the TX port for other device, or connect multiple lidars to raspberry pi

Resources

- Read and write from serial port with Raspberry Pi
- TFmini-RaspberryPi

3.2.5 Pi Camera Usage

Connection

Install the Raspberry Pi Camera module by inserting the cable into the Raspberry Pi. The cable slots into the connector situated between the Ethernet and HDMI ports, with the silver connectors facing the HDMI port.

Capture an image

```
sudo raspistill -o image.jpg
```

Record a video for 10 seconds

```
sudo raspivid -o video.h264 -t 10000
```

Resources

- How to install/use the pi camera
- python code and rapid capturing

3.2.6 PI and IMU communication

I2C Method by LSM9DS1 Library

Setup

In order to use the LSM9DS1 Library, we need to install WiringPi first. Enter the following command in Pi terminal:

```
sudo apt-get install libi2c-dev
git clone git://git.drogon.net/wiringPi
cd wiringPi
git pull origin
./build
```

Then we can install the LSM9DS1 Library:

```
git clone https://github.com/akimach/LSM9DS1_RaspberryPi_Library.git
cd LSM9DS1_RaspberryPi_Library
make
sudo make install
```

To test it, we can run the python sample code inside the library once we connect the IMU:

```
cd LSM9DS1_RaspberryPi_Library/example
sudo python LSM9DS1_Basic_I2C.py
```

Wiring

RPI	IMU
3.3v (Pin1)	Vcc
SDA (Pin3)	SDA
SCL (Pin5)	SCL
GND (Pin6)	Gnd

Resources

- LSM9DS1_RaspberryPi_Library

I2C Method

The example code for this section in the `PiCar/src/pi/imu`.

To compile, use the command:

```
gcc -o <programname> runi2c.c -lm
```

Wiring:

same as above did

The connection is by SMBUS.

For RPI, go to `/usr/include/linux`, replace `i2c_dev.h` with the header file in the repository

(Method 'enableIMU' needs further development to enable IMU configuration setting)

See Also:

- IMU datasheet

Resources

- I2C SPI Reference page

Contributors: Jerry Kong, Shadi Davari, Josh Jin

3.2.7 Installing ROS on Raspbian

This is how you build something.

3.3 Software

The software section will document all the heavy duty programming programming aspects of the project. There may be some overlap with the Electronics section.

3.3.1 Socket File Transfer

Basic File Transfer

Server side

```
1 import socket                    # Import socket module
2
3 port = 60000                     # Reserve a port for your service.
4 s = socket.socket()              # Create a socket object
5 host = socket.gethostbyaddr("your IP static IP if under same Wi-Fi")[0]      # Get␣
  ↪local machine name
6 s.bind((host, port))             # Bind to the port
7 s.listen(5)                      # Now wait for client connection.
```

(continues on next page)

(continued from previous page)

```

print ('Server listening....'.encode('ascii'))

while True:
    conn, addr = s.accept()     # Establish connection with client.
    print ('Got connection from', addr)
    data = conn.recv(1024)
    print('Server received', repr(data))

    filename='your file name'
    f = open(filename,'rb')
    l = f.read(1024)
    while (l):
       conn.send(l)
       print('Sent ',repr(l))
       l = f.read(1024)          #alter this to control data sending rate
    f.close()

    print('Done sending')
    conn.close()
```

Client side

```
import socket                   # Import socket module

s = socket.socket()             # Create a socket object
host = 'your ip address'     # Get local machine name
port = 60000                    # Reserve a port for your service.

s.connect((host, port))
s.send("Hello server!".encode('ascii'))

with open('received_file', 'wb') as f:
    print ('file opened')
    while True:
        print('receiving data...')
        data = s.recv(1024)          #must be identical to the data rate at server␣
↪side
        print('data=%s', (data))
        if not data:
            break
        # write data to a file
        f.write(data)

f.close()
print('Successfully get the file')
s.close()
print('connection closed')
```

Advanced Folder Transfer

Creator: Jerry Kong

To meet our need of a neat and organized data structure, this script is created. It has the capability to transfer the

entire folder to another remote desktop, no matter whether it is on a Windows System or Unix system. The script rests in `PiCar/src/Logging` To use the script, first set-up the IP addresses like in the basic version, change the root variable to the root folder name. Place the script at the same level as the root folder. Start the server script and then start the client script. The folder would then be transferred. A better protocol could be implemented, since the protocol being used now is not really efficient.

Wi-Fi Router Settings

Creator: Jerry Kong

This section is dedicated to users who are not familiar with Wi-Fi network setting, TCP protocol and wireless connection

To establish communication between two machine we need to know their IP address. Moreover, to provide a consistent network experience, a machine would have many ports to receive connections of different forms, with other devices. Thus we also need to agree on the port that two machines establish the connection on. However, depending on different internet environment and different ways of connection (Wi-Fi or ethernet), the IP address would also vary. With this section, you would get a sense of how this complicated system works and hopefully learn how to cope with "Connection fails" error when you are using the script.

IP Address

IP's full name is Internet Protocol. It's a scheme that specifies how computers find each other in the pool of Internet. The rules behind it is complicated, but the most important thing is that it serves as an identification for modern devices connected to Internet.

IPv4 vs IPv6

As a protocal, IP would have different versions, the latest version is version 6 and thus called IPv6. While IPv6 is stronger and has a larger pool of Internet, the older version IPv4 is not obsolete. The logic behind the two protocals are the same, hence we would now stick with IPv4, since it has a more concise format. (XXX.XX.XX.XXX)

Wi-Fi vs Ethernet

Wi-Fi is more convenient while wired connection (ethernet) offer steadiness and low latency. However, it is important to note that a computer connected to Wi-Fi does not have an IP, or at least, an acknowledged IP. The Wi-Fi or the router serves as a broadcaster and spread the connection from the ethernet to multiple machines, but they have the same IP address. The router can identify each machine by the IP address it assigns to the machine, but the machine can't use that address as the identification on the internet. Conclusively, machines under the same Wi-Fi build up a small intranet where these machines can identify each other by the address they are assigned, but once outside Wi-Fi network they are no longer acknowledged.

See Setup static IP address for RaspberryPi , so a machine would be assigned the same IP address when connected to the Wi-Fi.

TCP

TCP, Transmission Control Protocol, is a higher level protocol that enables data sending via the connection established by IP. Socket, a method based on TCP is typical method used for data transfer.

Port Forwarding

With the knowledge about address in mind we could start the connection once we have the right port. It is easy to do so if both machines are on Internet or under the same Wi-Fi, since they can identify with each other. Just pick up an empty port and they are good to go. However, we do want to establish connection between two machines even if one is on Wi-Fi and the other is on Internet. To do so, we use port forwarding. With port forwarding, a client can find the address of the router and use the port that is forwarded to connect with the machine.

For example, the address of the router is `172.10.10.111`, and a machine under the Wi-Fi is assigned static IP `192.168.1.188`. The routher and the machine agree on that the connection to the port `30000` of the routher would be forwarded to the port `6000` of the machine and vice versa. Thus a laptop could setup a connection with `172.10.10.111` on port `30000` to connect the port 6000 on machine with static IP `192.168.1.188`.

See How to setup port forwarding

3.3.2 Sensors (Lidar, IMU)

Setup

Make sure you have already connected TFmini Lidar and IMU as TFmini Lidar , IMU by LSM9DS1 did, and download the corresponding libraries.

Code

Under sub-directory `PiCar/src/pi/IMU_Lidar`

Steps

1. Download the repository and connect sensors correctly
2. Run the python script `Lidar_IMU_read_optimize.py`

3. After the program ends, you should see two csv files under the same directory. One records the time between two consecutive reads, and the other one contains data from sensors in the format: `timestamp, distance, accelaration in x,y,z, angular velocity in x,y,z`

3.3.3 Camera data by rapid capturing

Connection

Connect the camera correctly as mentioned in Getting started with picamera

Code

```
import time
import picamera
import datetime

frames = 20

def filenames():
```

(continues on next page)

(continued from previous page)

```
    frame = 0
    while frame < frames:
        current = datetime.datetime.now()
        yield '%s.jpg' % current
        frame += 1

with picamera.PiCamera(resolution=(480,480), framerate=100) as camera:
    camera.start_preview()
    # Give the camera some warm-up time
    time.sleep(2)
    start = time.time()
    camera.capture_sequence(filenames(), use_video_port=True)
    finish = time.time()
print('Captured %d frames at %.2ffps, in %f seconds' % (
    frames,
    frames / (finish - start), (finish - start)))
```

This will give you real time and fps.

Resources

- PiCamera Module
- Rapid capture and processing

3.3.4 Sensors & Camera concurrent reading using Timers

Connection

Connect the IMU, TFmini Lidar, and PiCamera as before.

Code

The code for this part is under directory `PiCar/src/pi/pythonTimer`

Resources

- Python multiprocessing–Process-based Parallelism
- Python threading timer object

3.3.5 Data Logging

Version Alpha (Camera data, IMU data, LiDar Data)

Creator : Jerry Kong

Ensure to correctly connect all electronics.

Code

The code could be found in‘‘ PiCar/src/pi/IMU_Lidar‘‘, you can find the method to enable IMU library here

IMPORTANT: If you have gone through the process before 06/18/2018, make sure you execute all steps again, few more functions and wrappers are added to the library

Run the script, a folder under the same directory would be generated, its name would be the starting timestamp of the script.

The file itself contains several straight forward methods that can be used to get data from IMU LiDar. The method it uses to take pictures is currently only viable within the script.

The IMU setting functions can't be used outside the script.

If called from command line or python shell, the script would place the image capturing and data logging proceses into two different cores on RaspberryPi

Use the command line option, you can bring up the usage page

```
python Lidar_IMU_data_optimize_delta.py -h
```

The script is based on delta timing (timers) method. A constant value of 0.0007 is subtracted from the period to maintain a consistent reading frequency.

Precision defines the minimum time that the script goes to check the diffrence between the last time and current time and consequently defines within what time difference that measures of LiDar and IMU occur simultaneously.

A great part of the codes are from Josh Jin's sensor/camera reading code

Version Beta (Magnetic reading added to IMU)

Creator : Jerry Kong

The code could be found in PiCar/src/pi/IMU_Lidar, the socket_server_client.py file is a integrated and important part of this data logging script, to learn more about socket folder sending, take a look at ‘ socket based file sending <http://picar.readthedocs.io/en/latest/chapters/usage/software.html#advanced-folder-transfer>‘_

Endless mode is implemented. User could stop the experiment with KeyboardInterrupt, the logging file and camera file would still be saved

Using -i command line input, we could run the script in endless mode (i.e. the duration would be set to 1000 seconds, we could stop the program by using KeyboardInterrupt(Ctrl + C))

Logging file sending module is integarted into the logging script. After the multiprocessing finished (loggind and filming), the script would start a raw socket server and a client on another computer could use the client side script to receive the logging file.

The script could either be called from the terminal or from other script by calling the funtion getSensorAndCamera.

‘-s’ command line argument and save parameter for getSensorAndCamera is implemented so that users can decide whether they want the logging file to be saved locally.

For installation and usage see the previous section

Version Beta 2.0 (Code re-organization, Process, self contained, PMU reading)

Creator : Jerry Kong

The code could be found in PiCar/src/pi/IMU_Lidar, device_int is the main file

The code is factor out and classified in an interface-oriented manner(i.e. the objects are put into class by its interface)

The class structure is:

```
device
|-----camera
|-----sensor
        |-----pmucounter
        |-----IMU
        |-----LiDar
```

Instead of its different class structure, the parameter for the main function is also different

```
getSensorAndCamera(host='192.168.1.121',port=6000,save=False,duration=5,endless=False,
↪trAccRate=6,trGyroRate=6,
                    trMagRate=7,accScale=2,gyroScale=245,magScale=4,cameraFreq=5,
↪imuRate=50,lidarRate=50,precision=0.001,tm=[])
```

Currently, to stop the sending process, a remote desktop must reach to the Server

To pass a new device outside the file to the function, see the sample code below

```
import device_int
from multiprocessing import Process

class currentSensor(device_int.sensor):

  def __init__(self, name="CS"):
    self.name = name
    self.type = "currentSensor"
    self.__conn = currentSensorCommunicationPort

  def detect(self):
    return self.__conn.is_available()

  def getFieldSize(self):
    """
    return a int
    """
    return 1

  def getHeader(self):
    """
    return a list
    """
    return ["current"]

  def getValue(self):
    """
    return the sensor reading
    """
    return [self.__conn.getCurrent()]

 cs = currentSensor()

 currentTimer = device_int.Timer(cs, currentSensor_read_period)

 p = Process(target = device_int.getSensorAndCamera, args = (host,port,save,duration,
↪endless,trAccRate,trGyroRate,
```

(continues on next page)

(continued from previous page)

```
                    trMagRate,accScale,gyroScale,magScale,cameraFreq,imuRate,
↪lidarRate,precision,[currentTimer]))

p.start()
#do some operation
p.terminate()
```

3.3.6 Data Analysis

Creator: Feiyang Jin

Data and photo synchronization

Once we get all data/photo from one experiment and save somewhere, we would like to synchronize them.

As camera speed is much slower than sensors speed, the sychronization is not perfect.

Algorithm: first match each photo to a row of data based on timestamp(best fit), then for unmatched data, find its previous closed photo and take it.

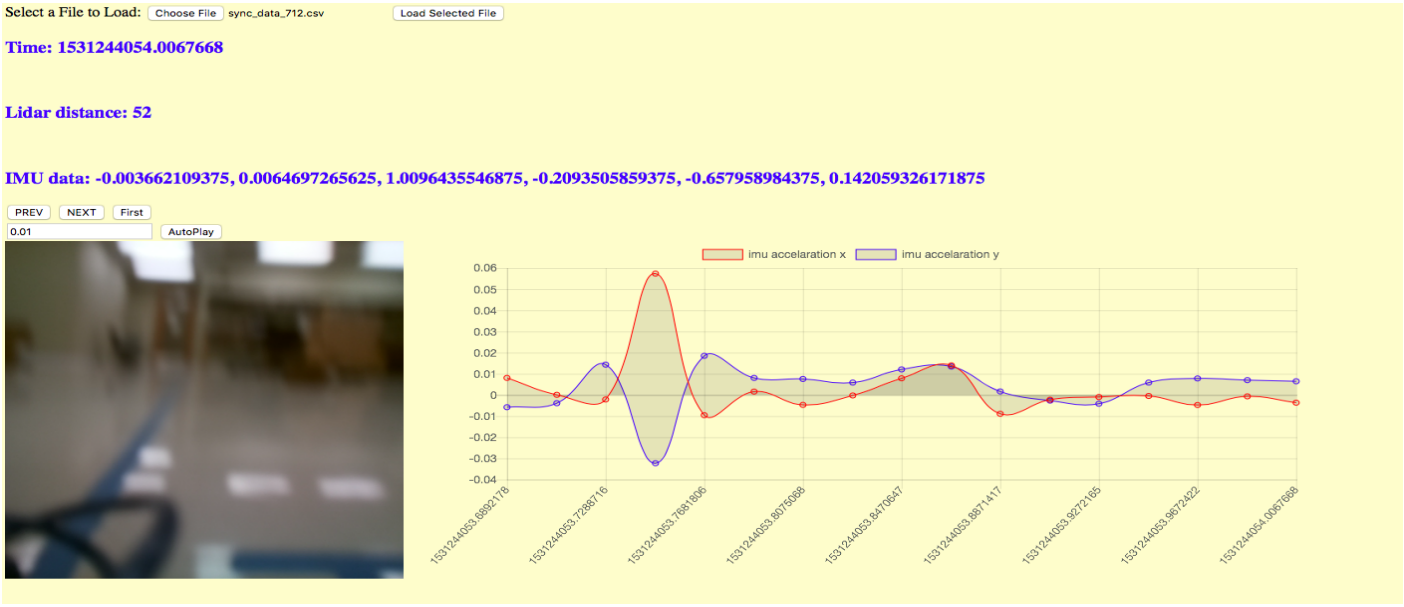
Note: This algorithm is a work in progress. If you have better strategy, please contact me.

The code is called `sync_time.py` under `PiCar/src/dataAnalysis`, and all raw data/photo are under `data_photo` under same directory.

Result: The python programe will output a csv file in the same directory, the format is [data1][data2]……[matched_photo]

Display Synchronized Data and Photo

The sychronized csv file provides us unlimited possibility. The following image shows the display site we built.



Source for this website is under `PiCar/src/dataAnalysis/Display`, the html requires you to upload the sychronized csv file, and then give you all the magic.

Note: You will need to install chart.js first; for papaParse.js, I include the package for you.

3.4 Datasheet

3.4.1 RaspberryPi

RaspberryPi 3 B+

This section documents some information about the PI 3 MODEL B+ our team is using for easier reference.

Check CPU information:

```
lscpu
```

For the model we are using:

Architecture	armv7l
Byte Order	Little Endian
CPU(s)	4
On-line CPU(s) list	0-3
Thread(s) per core	1
Core(s) per socket	4
Socket(s)	1
Model	4
Model name	ARMv7 Processor rev 4 (v7l)
CPU max MHz	1400.0000
CPU min MHz	600.0000
BogoMIPS	38.40
Flags	half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32

Important: Notice that the list says that the memory architecture is armv7l. However, on the raspberrypi 3 B+ official site, it says that the architecture is armv8. The diffrence is probably caused by the OS that the RaspberryPi is using(Raspbian). It is by default a 32bit system, and armv7 is 32bit while armv8 is 64bit, causing the architecture to adapt to 32bit

For more information about this model, click here

Based on armv7l, the memory architecture for this model is:

L1-instruction Cache	32-bytes cache line size 16 KB 2-way set-associative Cache
L1-data Cache	64-bytes cache line size 16 KB 4-way set-associative cache
L2 Cache	128 KB in size

(However, in RaspberryPi, L2 Cache is devoted to GPU, the benefit and necessity of enabling it to data cache needs further exploration)

Contributor: Jerry Kong

3.5 The Picar Module

class picar.**device**(*name='D'*)
Base class for all Devices

whoAmI()
A function that could be called to tell what device it is

class picar.**sensor**(*name='S'*)
Base class for all Sensors

detect()
A sensor should implement this method to tell if the sensor is currently available

getConn()
The connection could be returned by this function

getFieldSize()
A sensor should implement this method to return the field size requested by the sensor in logging file

getHeader()
A sensor should implement this method to return the header of it requested field in a list

getValue()
A sensor should implement this method to return values in a list

setConn(*conn*)
A sensor's connection should be able to be reset by this function

whoAmI()
A function that could be called to tell what device it is

class picar.**pmucounter**(*name='P'*)
PMU (Performance Monitoring Unit) reader

detect()
A sensor should implement this method to tell if the sensor is currently available

getConn()
The connection could be returned by this function

getFieldSize()
A sensor should implement this method to return the field size requested by the sensor in logging file

getHeader()
A sensor should implement this method to return the header of it requested field in a list

getValue()
A sensor should implement this method to return values in a list

setConn(*conn*)
A sensor's connection should be able to be reset by this function

whoAmI()
A function that could be called to tell what device it is

class picar.**IMU**(*name='I'*)
This class for IMU sensor

calibrate()
A wrapper for IMU calibration

detect()
This function tests if accel, gyro and mag are all available

getConn()
The connection could be returned by this function

getFieldSize()
Field size = 9

getHeader()
Header: AccelX, AccelY, AccelZ, GyroX, GyroY, GyroZ, MagX, MagY, MagZ

getValue()
Return a 9-elements list containing all IMU reading

setConn(*conn*)

Parameters **conn** – an IMU object created by imu library

setIMUScale(*aScl=2*, *gScl=245*, *mScl=4*)
Scale for IMU_SETUP

Available rate for accel : 2, 4, 8, 16

Available rate for gyro : 245, 500, 2000

Available rate for mag : 4, 8, 12, 16

(Value set other than these value might cause IMU to crush)

setIMUodr(*aRate=6*, *gRate=6*, *mRate=7*)
Output rate setter for IMU

Available rate for accel : 1 = 10 Hz 4 = 238 Hz

2 = 50 Hz 5 = 476 Hz

3 = 119 Hz 6 = 952 Hz

Available rate for gyro : 1 = 14.9 4 = 238

2 = 59.5 5 = 476

3 = 119 6 = 952

Available rate for mag : 0 = 0.625 Hz 4 = 10 Hz

1 = 1.25 Hz 5 = 20 Hz

2 = 2.5 Hz 6 = 40 Hz

3 = 5 Hz 7 = 80 Hz

whoAmI()
A function that could be called to tell what device it is

class picar.**LiDar**(*name='L'*)

detect()
A sensor should implement this method to tell if the sensor is currently available

getConn()
The connection could be returned by this function

getFieldSize()
FieldSize = 1

getHeader()
Header : LiDar

getValue()
Return a 1 element list

setConn(*conn*)

Parameters **conn** – a serial port

whoAmI()
A function that could be called to tell what device it is

class picar.**Camera**(*name='C'*, *res=(480, 480)*, *fr=40*)

capture(*gen*, **args*)

Parameters

- **gen** – a filename generator that has timing functionality
- ***args** – contains all the arguments gen needs

setFrameRate(*fr*)

Parameters **fr** – int in Hz

setRes(*res*)

Parameters **res** – resolution (length,width) in a tuple

whoAmI()
A function that could be called to tell what device it is

class picar.**Timer**(*kit*, *gap*)
A Timer class that helps get delta timing for a sensor

read(*t*)

Parameters **t** – current time to be compared with the time kept by the object

class picar.**Killer**(*state*)
A elegant killer, to make sure the process and subprocesses functions as expected

picar.**pre_exec**()
Easy to use function to prevent the subprocess to receive KeyboardInterrupt

picar.**filenames**(*alive*, *duration*, *cameraFreq*, *beginTime*)
Filename generator: Used with Camera class to generate a series of filename for the picture to be stored at

Parameters

- **alive** – The global variable keeping the state of the program and processes
- **duration** – Experiment duration
- **cameraFreq** – Camera Frequency
- **beginTime** – root directory as a timestamp

picar.**getCamera**(*gen*, *alive*, *duration*, *cameraFreq*, *beginTime*)
The filming function executed in a different core :param gen: the filenames generator :param alive: the global variable to track the state in the main function :param duration: the elapse time for the test :param cameraFreq: the cameraFrequncy in Hz :param beginTime: (string) the directory of the root folder for the logging files

picar.**getSensor**(*alive*, *rowList*, *duration*, *precision*, *datafile*, *timers*)

The data logging function executed in a different core :param alive: the global variable to track the state in the main function :param rowList: a list stores timestamp and logging data :param duration: the elapse time for the test :param precision: the gap between two visit of the script to sensors :param datafile: file name of the datafile :param timers: a list of timers holding sensors

picar.**getSensorAndCamera**(*host='192.168.1.121'*, *port=6000*, *save=False*, *duration=5*, *endless=False*, *trAccRate=6*, *trGyroRate=6*, *trMagRate=7*, *accScale=2*, *gyroScale=245*, *magScale=4*, *cameraFreq=5*, *imuRate=50*, *lidarRate=50*, *precision=0.001*, *tm=[]*)

A easy to use logging version supporting camera data logging, IMU reading, Lidar reading

CHAPTER 4

Results

The PiCar platform has been used in many research project. This page gives a comprehensive list of different results that were achieved.

4.1 IMU Autonomous Navigation

4.1.1 Proposal

There are three main objectives of our research:

1. Building a working PiCar to carry out experiments
2. Investigating IMU data
3. Design algorithms for self-driving navigation

First, working PiCar has to be built so that an autonomous driving system can be implemented. After the system has been implemented, IMU data then can be collected and explored through experiments. Finally, algorithms of locating the current location of the car based on IMU data can be developed.

4.1.2 Authors

- Sam Chai
- Moira Feng

4.1.3 Links

[Report] [Presentation] [Code]

4.2 Power Management

4.2.1 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.

4.2.2 Authors

- Yunshen Huang

4.2.3 Links

[Report]

4.3 Object Tracking in Low-Power Autonomous Systems

4.3.1 Proposal

Computer vision algorithms are typically reserved for platforms that can handle the computational workload needed to process the huge amount of data in images and videos. The recent surge in artificial intelligence, machine learning, and computer vision have guided the development of powerful processors that can quickly and more efficiently handle the computationally intensive algorithms. For this project, I aimed to go against the grain and implement computer vision and artificial intelligence on a Raspberry Pi, a low-power IoT device that is the on-board processor for a small autonomous vehicle project called the PiCar.

The first part of the project was the development and implementation of a real-time control algorithm using optical flow and machine learning to successfully navigate randomly generated obstacle fields. The processing was done entirely on a Raspberry Pi 3 and the video stream was provided from the standard Raspberry Pi camera module. The algorithm worked in the following manner:

1. Read images from video stream
2. Detect features using Shi-Tomasi corner detection
3. Calculate optical flow vectors
4. Calculate time to contact (TTC) for each tracked point (x,y)
5. Cluster three dimensional data (x,y,TTC) using DBSCAN
6. Sort clusters by lowest TTC
7. Calculate servomotor angle and motor PWM

8. Send signal to motor-controller via SPI

4.3.2 Authors

- William Luer

4.3.3 Links

[Report] [Code]

4.4 PiCar Mobile Movement Control

4.4.1 Proposal

This report is aimed at introducing new researchers to the findings and progress made on the PiCar Project as of July, 2018 regarding the movement control of the PiCar. This report will cover the currently implemented features of the PiCar, including semi-automated movement, a killswitch, real-time user control using WASD on a keyboard, a replay function, and data collection. Additionally, the last section of the report covers known issues with the PiCar which includes inaccurate servo controls and noisy Inertial Measurement Unit (IMU) data.

Note: While uploading the Arduino program, if there is an issue with the `Simpletimer` library, download the `Simpletimer.h` file from the Official Arduino Codebase, and place it in the same folder as the `WASD.ino` file.

4.4.2 Authors

- Hayden Sierra
- Daniel Kelly

4.4.3 Links

[Report] [Video 1] [Video 2]

4.5 LIDAR Obstacle Avoidance

4.5.1 Proposal

Object tracking and obstacle avoidance are two key features for a robot with mechanical movability and visual detection functionality. They are associated with many hot application fields such as path followers and self-driving cars. With the big picture in mind, we chose to implement those basic functions on the PiCar, a Raspberry Pi powered and wheeled robotic car, as a research project. By implementing the obstacle avoidance and the object tracking features onto the car, we aim to create powerful and practical algorithms that enable the car to follow a certain object wisely without crashing into any obstacles.

4.5.2 Authors

- Amelia Ma
- Chufan Chen

4.5.3 Links

[Report] [Website]

CHAPTER 5

Changelogs

v2.0 (latest)

- Changed chassis design to make PiCar more robust
- Added single encoder with DC motor for closed loop
- Incorporated LIDAR and IMU
- Added comprehensive ReadtheDocs documentation

v1.0 (Github)

- Designed and built entirely 3D printable chassis
- Used brushless DC motor, ESC and servo for steering and driving
- Designed and assembled PCB for power management, reading hall effect sensors and IMU data
- Used camera with optical flow computer vision algorithm for tracking

CHAPTER 6

Contributors

The PiCar project was successful due to the significant contributions of all its current and past members.

Current Team

- Advisor(s):
  - Dr. Xuan 'Silvia' Zhang
  - Dr. Christopher D. Gill
- Team Members:
  - An Zou
  - Adith Boloor
  - Feiyang Jin

Former Team

- Advisor(s):
  - Dr. Humberto Gonzalez
- Team Members:
  - Andrew O'Sullivan
  - Daniel Kelly
  - Hayden Sierra
  - Jacob Cytron
  - Jeffrey Gu
  - Jerry Kong
  - John Fordice
  - Karina N. Martinez-Reyes
  - Kristen Koyanagi

- Matt Kollada
- Meizhi Wang
- Moira Feng
- Patrick Naughton
- Reese Frerichs
- Sam Chai
- Shadi Davari
- William Luer
- Yak Fishman

Index