Hardware-Software Stack for an RC car for testing autonomous driving algorithms

Preprint · October 2019

DOI: 10.13140/RG.2.2.14768.30728

3 authors:

Kakul Shrivastava
Aligarh Muslim University
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Maruthi S Inukonda
Indian Institute of Technology Hyderabad
3 PUBLICATIONS **5** CITATIONS
SEE PROFILE

Sparsh Mittal
Indian Institute of Technology Hyderabad
120 PUBLICATIONS **1,557** CITATIONS
SEE PROFILE

Hardware-Software Stack for an RC car for testing autonomous driving algorithms

Kakul Shrivastava†, Maruthi Inukonda§ and Sparsh Mittal§
† Department of Electronics Engineering, Zakir Husain College of Engineeering and Technology, Aligarh Muslim University, India
§Department of Computer Science and Engineering, IIT Hyderabad, India.
E-mail:kakul.shrivastava.1@gmail.com,{cs18resch01001,sparsh}@iith.ac.in.

Abstract

In this paper, we report our ongoing work on developing hardware and software support for a toy RC car. This toy car can be used as a platform for evaluating algorithms and accelerators for autonomous driving vehicles (ADVs). We describe different sensors and actuators used and interfacing of them with two processors, viz., Jetson Nano and Raspberry Pi. Where possible, we have used ROS nodes for interfacing. We discuss the advantages and limitations of different sensors and processors and issues related to their compatibility. We include both software (e.g., python code, linux commands) and hardware (e.g., pin configuration) information which will be useful for reproducing the experiments. This paper will be useful for robotics enthusiasts and researchers in the area of autonomous driving.

Index Terms

RC car, deep learning, autonomous driving vehicle, low-power, computer vision, robotics

1 Introduction

Autonomous driving vehicles[1] have attracted the interest of researchers in recent years. A crucial challenge in ADV research is that experimenting with proposed algorithms/accelerators requires real-life vehicles. This incurs huge amount of investment and risk, and the facility may not be available for most researchers. Further, a thorough design-space exploration is required for testing with all system-parameters.

At the early design stage, researchers can experiment their ideas on a small RC car, which is much cheaper, easily accessible and risk-free. As a step towards addressing this need, we have developed the hardware-software stack for a toy-car.

Contributions: We report the challenges faced in the assembling different software tools and hardware platforms. We discuss the reasons for choosing certain devices/components. Assembling the hardware/-software components from different vendors is a challenging task because they have hardware/software dependencies which are not immediately obvious from their specifications and datasheets. We have obtained information from heterogeneous sources and hence, availability of all this information in a single paper will be helpful for both novice and expert in the field of robotics. Figure 1 showing outer and inner view of the assembled RC car.

- *IIT Hyderabad Technical Report number 2019-CSE-CANDLE-01. Support for this work was provided by Science and Engineering Research Board (SERB), India, award number ECR/2017/000622. Kakul worked on this paper while working as an intern at IIT Hyderabad.*

1. We use the following acronyms frequently in this paper: autonomous driving vehicles (ADVs), camera serial interface (CSI), deep neural network (DNN), electronic speed controller (ESC), global positioning system (GPS), inertial measurement unit (IMU), inter-integrated circuit [1] (I2C), light detection and ranging (LIDAR), light emitting diode (LED), national marine electronics association (NMEA), pulse width modulation (PWM), radio controlled (RC), robot operating system (ROS), system on a chip (SoC).

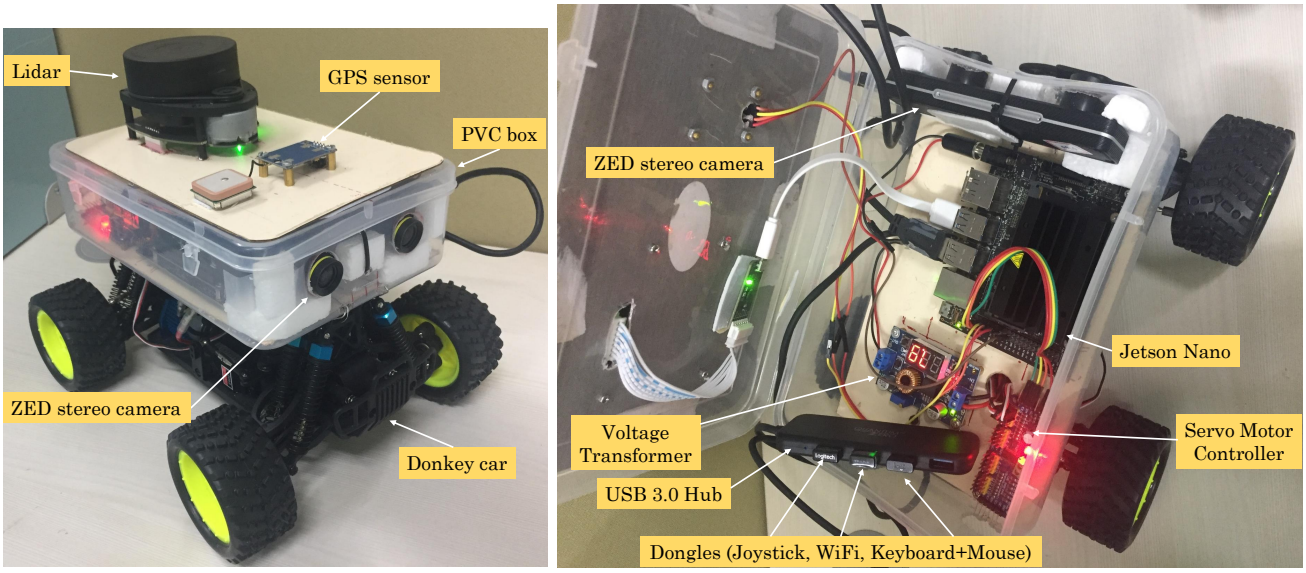

Fig. 1. (Left) Outer view of the car. (Right) Inner view of the car

Presently, we have configured two processing units, of which, one can be used by a user. These processing units are Jetson Nano (abbreviated as Nano) and Raspberry Pi 3B+ (abbreviated as RPi). Figure 2 shows the block-diagram with Nano as the processing unit and Figure 3 shows the block-diagram when processing is done in RPi. Evidently, there are minor differences between the overall flow and the reasons for this will be clear in subsequent sections.

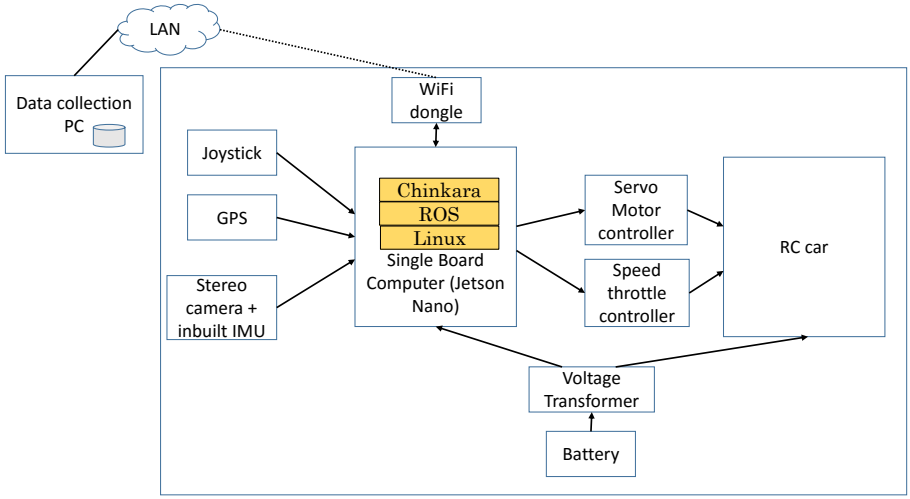

Fig. 2. Block-diagram when the processing is done on Jetson Nano

Table 1 shows a rough estimate of the bill of material in Indian Rupees (INR).

The rest of the paper is organized as follows. Section 2 discusses the harware engines (RPi and Nano) and ROS. Section 3 discusses about sensors and actuators used in our work. Section 4 discusses their interfacing with RPi and Nano. Section 5 presents the software support for the RC car. Section 6 discusses our efforts in building and testing with a driving-track. Section 7 concludes this paper and also mentions some avenues for future work.

2 RC Car, Hardware Engines and Operating System

2.1 Donkey car

As for the RC car, we use the "donkey car" [2]. The reason for choosing this was that the donkey car is fully assembled car with a compute engine (Raspberry PI) and open source software distribution. However, we

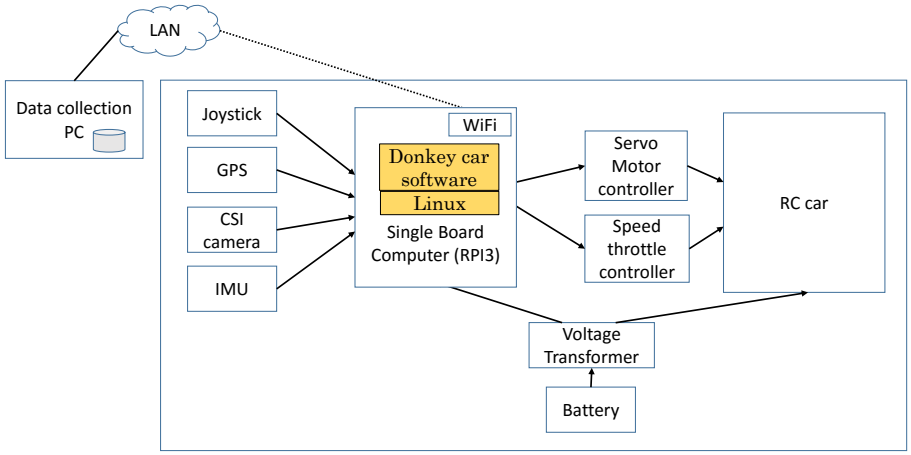

Fig. 3. Block-diagram when the processing is done on RPi

TABLE 1
Bill of material (1 USD ≈ 70 INR).

Sl. No	Item Model no	Item description	Price INR
1	Jetson Nano	Single board computer to connect sensors on the car	8,899
2	Raspberry PI Model B	Single board computer to connect sensors on the car	3,070
3	Donkey HSP 94186 Brushed RC Car	Toy car with servo motor, battery pack, charger, 3D-top-cage, laser-cut base plate	19455
4	Zed stereo camera	3D camera	31430
5	Slamtech RP Lidar	12 mtr range	8745
6	TP-Link TL-WN823N Wifi 300N usb dongle		599
7	XL4015 5A Step Down Adjustable Power Supply with LED Voltmeter		593
8	GPS module	GPS module	999
9	MPU6050 IMU	IMU device	210
10	Servo Driver PCA 9685	Servo motor controller	500
11	128GiB microSD card		2230
12	Logitech f710 gaming joystick	Wireless joystick	3150
13	Nano voltage adapter		692
14	Jumper cables		60
15	Transparent PVC box	For safely holding RPi or Nano	150
	Total		80,782

found that Raspberry PI hardware does not have GPU for running machine learning algorithms, and software distribution does not support compute engine with GPU (like Jetson Nano). DL based perception algorithms ran very slow on RPI3. Also, we could not connect stereo camera to RPI. For this reason, we had to integrate Jetson Nano and develop a software based on ROS.

2.2 Raspberry Pi

The Raspberry Pi is a low-price single-board computer [3] and is shown in the left part of Figure 4. It has a Broadcom BCM2837B0 SoC with an ARM 1.4GHz quad core Cortex-A53 CPU and Broadcom VideoCore IV GPU running at 250 MHz. It has bluetooth and on-board 802.11n Wi-Fi. RPi 3B+ can be booted over USB or network and hence, it does not require an SD card. The peak-performance of GPU of RPi is much lower than that of its CPU and hence, the main computing engine of RPi is its CPU. The pin layout of RPi is shown in Figure 5.

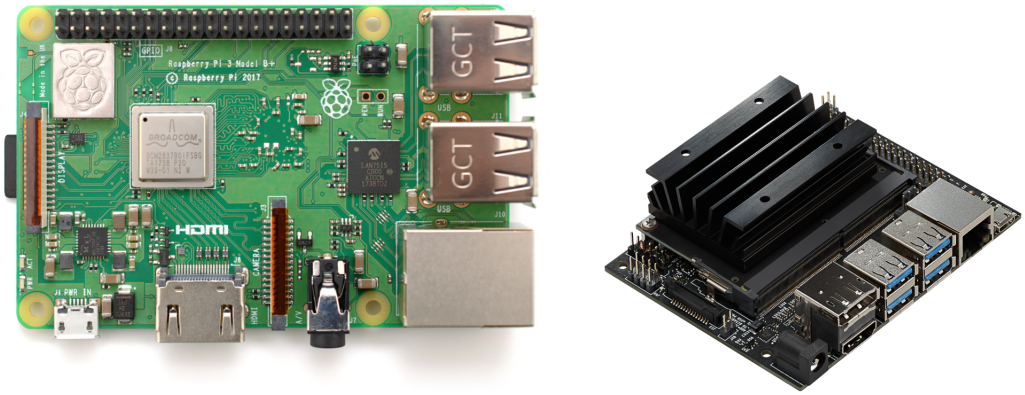

Fig. 4. Raspberry Pi 3B+ (left) and Jetson Nano (right)

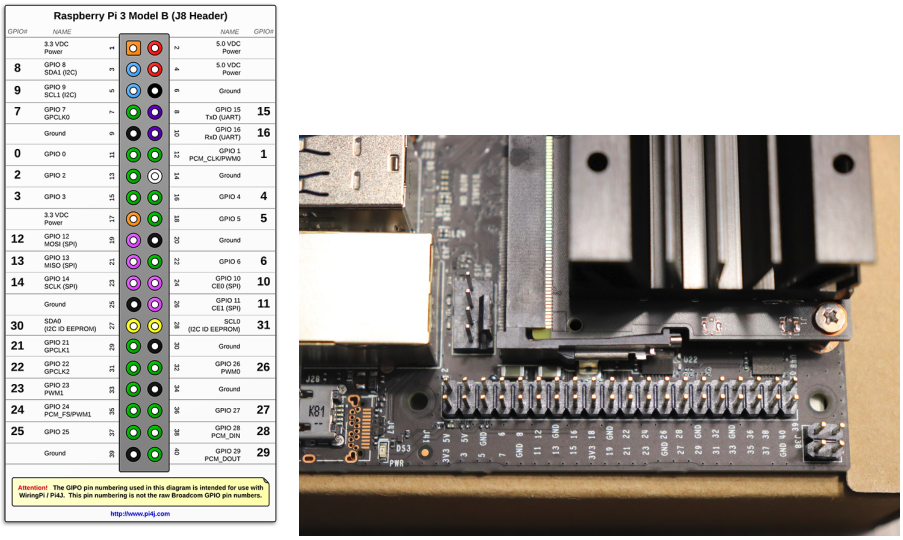

Fig. 5. Pin layout of RPi (left) and Jetson Nano (Right)

2.3 Jetson Nano

Jetson Nano is an embedded "system-on-module" (SoM) from the NVIDIA Jetson series of processors [4, 5]. Nano is a smaller version of Jetson TX2. It is shown in the right part of Figure 4. It has heterogeneous CPU-GPU architecture [6, 7], with a quad-core ARM A57 CPU, integrated 128-core Maxwell GPU, 4GB LPDDR4 memory [8] running at 1600MHz with a bandwidth of 25.6 GB/s. It includes a 16GB eMMC 5.1 flash storage. and also supports MIPI CSI-2 and PCIe Gen2 high-speed I/O. It has a peak performance of 472 GFLOPS (FP16) and consumes 5 to 10W power. Hence, it is ideally suited for running AI applications [9, 10].

Important points about power consumption of Nano: Jetson Nano Developer Kit requires a 5V power supply capable of supplying 2A current. The developer kit is configured to accept power via the Micro-USB connector. If the developer kit's total load is expected to exceed 2A, e.g., due to peripherals attached to the carrier board, one has to connect the J48 Power Select Header pins to disable power supply via Micro-USB and enable 5V-4A via the J25 power jack. Another option is to supply 5V-6A via the J41 expansion header (3A per pin). In case the load exceeds, Nano powers off itself automatically. The power through micro-USB is not enough when ZEDM camera is used.

2.4 Robot operating system

Robot operating system (ROS) is an open-source, meta-operating system which provides services such as "hardware abstraction, low-level device control, implementation of commonly-used functionality,

message-passing between processes, and package management" [11]. It is a distributed framework of processes (called nodes) that allow executables to be developed separately and loosely coupled at execution time. These processes are combined into "Packages" and "Stacks" for easy distribution.

3 Sensors and Actuators

In this section, we discuss various sensors and actuators used in the project.

3.1 CSI Camera

We have used Raspberry Pi camera module v2 as shown in Figure 6, to record the images to capture the dataset as well as to keep a visual track of the surrounding while the car is driving autonomously. The v2 camera module has a Sony IMX219 8-megapixel sensor. The camera board attaches via a 15-way ribbon cable.

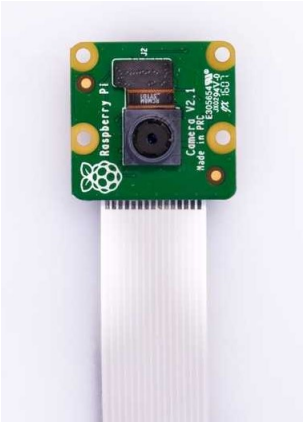

Fig. 6. Raspberry Pi v2 Camera Module

3.2 Stereo Camera

One of the major task in autonomous driving is object detection so that the car can drive safely without hitting any object. To perform this task and to take the corresponding proper action of braking or changing the path can only be taken if it is known that how far the obstacle is. This task can be done with the help of stereo camera which contains two camera sensors and therefore can simulate human binocular vision and possess the ability to capture three dimensional images. We have used Zed Mini by Stereolabs [12] which is shown in Figure 7. It uses triangulation to create a three-dimensional understanding of the scene, which helps in understanding the space and motion.

Fig. 7. Zed Mini Stereo Camera

3.3 Global Positioning System (GPS)

GPS is a Global Navigation Satellite System that provides location and time information sent by satellites in space which are received by GPS receivers on Earth. GPS receiver module gives output in standard NMEA string format [13]. It provides output serially on Tx pin with default 9600 baud rate. This NMEA string output from GPS receiver contains different parameters separated by commas like longitude, latitude, altitude, time etc. Each string starts with $ and ends with carriage return/line feed sequence. The NMEA string starting with $GPGGA can be used to read time, longitude, latitude and altitude along with directions. Its format is as follows:

```
$GPGGA, UTC Time, Latitude, N/S Indicator, Longitude, E/W Indicator, Quality
Indicator, Satellites Used, Horizontal Dilution of Precision, Mean Sea Level
Altitude, Unit of Altitude, Undulation, Unit of Undulation, Age of Correction
Data, Differential Base Station ID, Checksum
```

We used U-blox NEO-6M GPS Module, as shown in Figure 8.

Fig. 8. U-blox NEO-6M GPS Module

We found that GPS coverage is not strong at all places in our campus. A red LED blinks when there is a coverage.

3.3.1 Pin Configuration:

- VCC: Power supply (3.3 V)
- GND: Ground
- TX: Transmit data serially which gives information about location and time and is connected to Rx of controller
- RX: Receive Data serially. It is required when we want to configure GPS module.

To read the GPS data one can use the `cat` command to directly view the received data on terminal window.

3.4 Inertial Measurement Unit

IMU is used to measure a body's linear and angular motion. We have used MPU6050 [14] which is shown in Figure 9. It is an integrated 6 axis motion tracking device. It has 3 axis gyroscope and 3 axis accelerometer. One can communicate with this module using I2C communication protocol.

3.4.1 Pin Configuration

- VCC: This is used for powering the MPU6050 module
- GND: This is the ground pin
- SDA: SDA pin is used to read data from MPU6050 module
- SCL: SCL pin is used for clock input
- XDA: This is used for configuring and reading from external sensors (not used in our experiments)
- XCL: This is used for configuring and reading from external sensors (not used in our experiments)
- ADO: I2C Slave Address LSB (not applicable in our experiments)
- INT: Interrupt pin for indication of data ready (not used in our experiments)

To interface with any microcontroller, we have to connect the first four pins of MPU6050 with the corresponding pins of microcontroller.

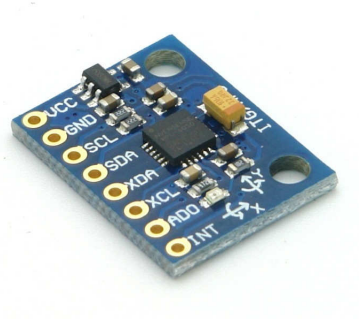

Fig. 9. MPU6050 Module

3.5 LIDAR

LIDAR measures the distance with the use of laser light as the target is illuminated and the reflected light is measured using the sensor. We have used RPLIDAR A1 by Slamtec [15] which is shown in Figure 10. The core of the sensor runs clockwise to perform a 360 degree omnidirectional laser range scanning for its surrounding environment and then generate an outline map for the environment. It has a sample rate of upto 8000 Sa/s and a distance range of upto 12 meters.

Fig. 10. RPLIDAR A1

3.6 Joystick Controller

To train our RC car run autonomously, we require the dataset to train the model. That dataset is capture by first driving the car manually and recording the data of all the sensors which can later be used to train the model. Joystick controller is required to run the car manually. We have used Logitech F710 wireless gamepad [16, 17], which is shown in Figure 11.

3.7 Steering and Throttle Control

There are two motors in the donkey car to control its speed and direction: a DC brushed motor to control the speed of all the 4 wheels and a servo motor to control the steering. The motors are controlled using PWM/servo driver PCA9685 [18] as shown in Figure 12. It uses `I2C` communication. The servo motor is directly controlled using the servo driver while the throttle output of servo driver is first given to an electronic speed controller (ESC) which then controls the speed of the throttle motor, as shown in Figure 13.

Fig. 11. Logitech F710 Wireless Gamepad

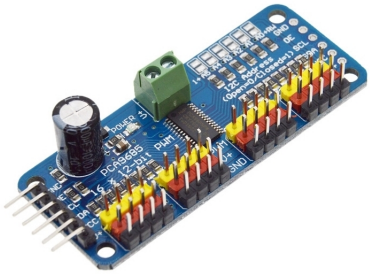

Fig. 12. PWM/Servo Driver PCA9685

Fig. 13. Electronic Speed Controller

3.7.1 Pin Configuration of PCA9685

Power Pins

- GND - This is the ground pin
- VCC - This is the logic power pin, connect this to the logic level you want to use for the PCA9685 output, should be 3 - 5V max.
- V+ - This is an optional power pin that will supply distributed power to the servos. It is not used at all by the chip. You can also inject power from the 2-pin terminal block at the top of the board. You should provide 5-6VDC if you are using servos.

Control Pins

- SCL - I2C clock pin, connect to your microcontrollers I2C clock line.
- SDA - I2C data pin, connect to your microcontrollers I2C data line.
- OE - Output enable. Can be used to quickly disable all outputs. When this pin is low all pins are enabled. When the pin is high the outputs are disabled. Pulled low by default so it's an optional pin.

Output Ports

There are 16 output ports. Each port has 3 pins: V+, GND and the PWM output. Each PWM runs completely independently but they must all have the same PWM frequency.

4 INTERFACING OF DIFFERENT COMPONENTS

In this section, we discuss interfacing of different components with RPi and Nano.

4.1 Interfacing of Camera

We have interfaced the camera with RPi and Nano.

4.1.1 Interfacing of Camera with Raspberry Pi

- While interfacing with Raspberry Pi, first enable the camera support from configuration settings:
  `$ sudo raspi-config`
- One can make use of four Raspberry Pi applications to interface the camera which are : `raspistill`, `raspivid`, `raspiyuv` and `raspividyuv`. On example is as follows:

  ```
  $ raspivid -h 1080 -w 1080 -t 10000
  ```

  One can set many other parameters too.

4.1.2 Interfacing using Robot Operating System (ROS)

Camera can be interfaced using ROS through already available ROS packages like `cv_camera` and `jetson_csi_cam`. One can follow the procedure as follows:

- **Installing and building ROS package**
  ```
  $ git clone https://github.com/OTL/cv_camera.git
  $ catkin_make
  $ source devel/setup.bash
  ```
- **Starting ROS Node**
  ```
  $ rosrun cv_camera cv_camera_node
  ```
- **View stream using Image Viewer**
  ```
  $ rosrun image_view image_view image:=/cv_camera/image_raw
  ```

Issues Faced while Interfacing:
In Raspberry Pi, when starting the ROS node, the device could not be opened. To resolve this issue, run following command:
`$ sudo modprobe bcm2835-v4l2`

4.1.3 Interfacing of Camera with Jetson Nano

- From GUI login, test the mipi-csi camera directly without ROS
  ```
  $ gst-launch-1.0 nvarguscamerasrc ! 'video/x-raw(memory:NVMM),width=3820,
  height=2464, framerate=21/1, format=NV12' ! nvvidconv flip-method=0
  ! 'video/x-raw,width=960, height=616' ! nvvidconv ! nvegltransform !
  nveglglessink -e
  ```

- **Issue 1 faced :**
  `gst-launch` fails with permission error

  Resolution:
  Logged-in user must belong to video group. Logout and login after adding self to video group.
- **Issue 2 faced :**
  `gst-launch` fails with following error:
  ```
  nvbuf_utils: Could not get EGL display connection
  Setting pipeline to PAUSED ...
  Using winsys: x11
  ERROR: Pipeline doesn't want to pause.
  ```

```
Setting pipeline to NULL ...
Freeing pipeline ...
```

Resolution:
Run this command from GUI login.

- **Procedure:**

```
$ roslaunch jetson_csi_cam jetson_csi_cam.launch
or
$ roslaunch jetson_csi_cam jetson_csi_cam.launch width:=3840 height:=2160
fps:=15
```

4.1.4 Calculation of true framerate:

Use the following command:

```
$ rostopic hz /csi_cam/image_raw
```

4.2 Interfacing of stereo camera

Zed Mini stereo cam requires USB 3.0 ports, but Raspberry Pi 3B+ board does not have 3.0 ports. Therefore other options we had were Hikey 960, Odroid XU4 and NVIDIA kits. The `ZED SDK` which is necessary for calculating depth requires `CUDA` and `NVIDIA GPU` so Hikey 960 and Odroid XU4 cannot be used as well. So, we used the zed mini with NVIDIA Jetson Nano.

The ZED is a UVC camera so you can capture stereo video in MP4 or AVI format using any third-party software. However to use the ZED SDK, you will need to record your videos in Stereolabs SVO format. SVO files contains additional metadata such as timestamp and sensor data. To record SVO videos, you can use the ZED Explorer application in GUI or command-line mode or build your own recording app using the ZED API. ZED Explorer can only be installed through ZES SDK but it is given that we can install SDK without CUDA support too and can record video in SVO format. SVO videos can be recorded using various compression modes.SVO videos can be recorded using various compression modes. But recorded videos are not what we require.

The Stereolabs provides ZED ROS wrapper [19] which lets us use the ZED stereo cameras with ROS. It provides access to the following data:

- Left and right rectified/unrectified images
- Depth map
- Colored 3D point cloud
- Visual odometry: Position and orientation of the camera
- Pose tracking: Position and orientation of the camera fixed and fused with IMU data

Procedure:

- **Build the Package:**
  `zed_ros_wrapper` is a catkin package. To install open a bash terminal, clone the package from Github and build it.

```
$ cd  /catkin_ws/src/ #use your current catkin folder
$ git clone https://github.com/stereolabs/zed-ros-wrapper.git
$ cd ..
$ catkin_make -DCMAKE_BUILD_TYPE=Release
$ echo source $(pwd)/devel/setup.bash >>  /.bashrc
$ source  /.bashrc
```

- **Starting the ROS Node:**
  The ZED is then in ROS as a node that publishes its data to topics. Open a terminal and use roslaunch to start the ZED node:

```
$ roslaunch zed_wrapper zedm.launch
```

- **Displaying ZED Data**
  We subscribed the following topics to collect the required information :
  - /zed/zed_nodedepth/camera_info
  - /zed/zed_node/imu/data_raw
  - /zed/zed_node/left/image_rect_color
  - /zed/zed_node/right/image_rect_color
  - /zed/zed_node/depth/depth_registered

    4.2.0.1

We can specify the parameters to be used by the ZED node by modifying the values in the files `params/common.yaml`, and `params/zedm.yaml`

4.3 Interfacing of GPS

The GPS receiver module uses UART communication to communicate with controller or PC terminal. We interfaced GPS both with Raspberry Pi 3B+ as well as NVIDIA Jetson Nano. Before using UART on Raspberry Pi, one should configure and enable it. To read the GPS data one can use the `cat` command to directly view the received data on terminal window.

4.3.1 Interfacing of GPS with Raspberry Pi

To enable UART communication on Raspberry Pi and to interface GPS to receive the data, follow the following steps:

- Open the `/boot/config.text` file using following command:
  ```
  $ sudo nano /boot/config.text
  ```
- Edit the above to add the following lines at the bottom:
  ```
  dtoverlay=pi-disable-bt
  core_freq=250
  enable_uart=1
  force_turbo=1
  ```
- Make a copy of `/boot/cmdline.txt` file and then open it
  ```
  $ sudo cp /boot/cmdline.txt /boot/cmdline.txt.backup
  $ sudo nano /boot/cmdline.txt
  ```
- Edit this file to remove `console=serial0,115200` and modify `root=/dev/mmcblk0p2` and reboot the pi.
- Stop and diable the Pi's serial ttyS0 service and reboot the Pi.
  ```
  $ sudo systemctl stop serial-getty@ttyS0.service
  $ sudo systemctl disable serial-getty@ttyS0.service
  $ sudo reboot
  ```
- Enable the Pi's ttyAMA0 service:
  ```
  $ sudo systemctl enable serial-getty@ttyAMA0.service
  ```
- To verify, use the command `ls -l /dev`
- Use the following command the view the data received by GPS:
  ```
  $ sudo cat /dev/ttyAMA0 | grep GPGGA
  ```

4.3.2 Interfacing of GPS with Jetson Nano

To interface the GPS with Jetson Nano follow the following steps:

- Connect the RX and TX pins of GPS to TX and RX of UART2 of Jetson Nano i.e. pin number 8 and 10 respectively.
- Use the following command the view the data received by GPS:
  ```
  $ sudo cat /dev/ttyTHS1 | grep GPGGA
  ```

4.3.3 Interfacing Results

We received the following data as shown in Figure 14.

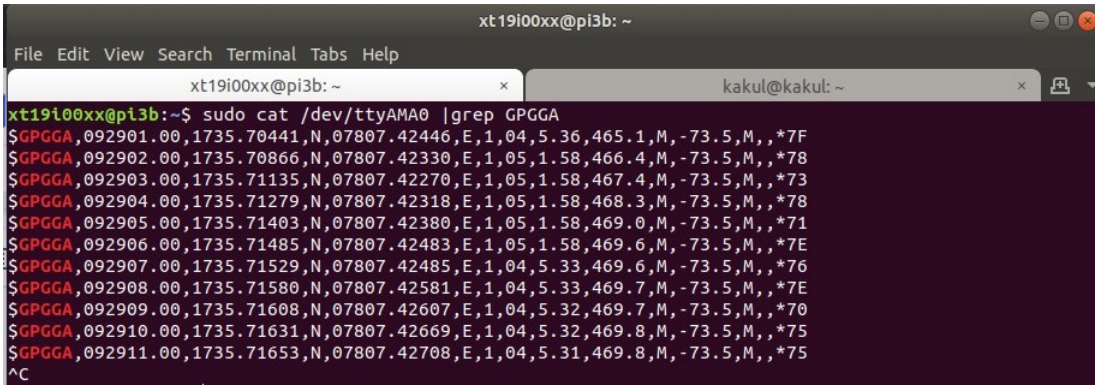

Fig. 14. Data Received from GPS

4.3.4 ROS Implementation

We used an inbuilt ROS package `nmea_navsat_driver` to interface the GPS using ROS [20–22]. Follow the steps as below:

- Install the package.
- Start the ROS node :

```
$ roscore
$ rosparam set
$ rosrun nme_navsat_driver nmea_serial_driver _port:=/dev/ttyTHS1
$ rostopic echo /vel
```

4.4 Interfacing of IMU and sensor calibration

To interface MPU6050, one should ensure that I2C protocol is turned on in the microcontroller used. We have interfaced the IMU using Python on Raspberry Pi 3B+ as well as NVIDIA Jetson Nano. One can enable I2C communication on Raspberrypi by changing the congifuration setting using following command:

```
$ sudo raspi-config
```

We used smbus library to interface the IMU which can be installed using following command:

```
$ sudo apt-get install python-smbus
```

While using IMU, orientation of the sensor should be fixed otherwise gravity effects along the axes due to change in orientation will give incorrect data. When the sensor is at rest the IMU may still give acceleration values having a constant bias which also includes an acceleration equal to gravity (g) in z axis. To remove this we can first take a certain set of data at the beginning, take their average and subtract this constant bias from further readings to get accurate data. We have written python code to implement this which is shown in Appendix A.

4.5 Interfacing of LIDAR

ROS has an inbuilt package `rplidar` which can be used to interface lidar [23, 24]. We have interfaced it with NVIDIA Jetson Nano as follows:

- Clone the package into the catkin workspace:
  `$ git clone https://github.com/Slamtec/rplidar_ros.git`
- Build the package:
  `$ catkin_make`
- **Run ROS package:**
  There are two ways to run this package:

  - **Run rplidar node and view in the rviz:**

    ```
    $ roslaunch rplidar_ros view_rplidar.launch
    ```
    You should see rplidar's scan result in the rviz.
  - **Run rplidar node and view using test application:**

    ```
    $ roslaunch rplidar_ros rplidar.launch
    ```

    Use the following command to view the raw data:
    ```
    $ rosrun rplidar_ros rplidarNodeClient
    ```
- **Calibration:**
  It also needs to be calibrated to know the angle corresponding to each direction. The calibration data for our car is as follows :
  - Left : (+127 to +51) degree
  - Front : (+38 to -35) degree
  - Right : (-48 to -123) degree
  - Rear : (-126 to -178) and (+180 to +158) degree

4.6 Interfacing of joystick

We have interfaced the joystick using ROS on NVIDIA Jetson Nano. It can be done using an inbuilt ROS package named `joy` as follows:

- **Installation:**
  ```
  $ sudo apt-get install ros-melodic-joy
  ```
- **Configuring Joystick:**
  The device name can be known using the following command:
  ```
  $ ls /dev/input/
  ```
  It is in the format `jsX`, where X is a number greater than or equal to 0. We need the set this device name in the param list of ROS node. When booted up, joystick will be configured as js0.
- You can test the joystick using following command:
  ```
  $ sudo jstest /dev/input/jsX
  ```
- **Starting ROS Node:**
  The joy package contains `joy_node`, a node that interfaces a generic Linux joystick to ROS. This node publishes a "Joy" message, which contains the current state of each one of the joystick's buttons and axes. You can start theROS node and subscribe to messages as follows:
  ```
  $ roscore
  $ rosparam set joy_node/dev "/dev/input/jsX"
  $ rosrun joy joy_node
  $ rostopic echo joy
  ```

4.6.1 Modes in Joystick

There is a sliding button at the back with D and X as two positions which is for selection among two modes `XInput mode` and `Direct Input Mode`. When it is slided to and from D to X, configured name changes i.e. its index increments from 0 by 1 every time so we would have to set the parameter everytime. Also, it will not work in Direct mode.

4.7 Interfacing of WiFi

The following commands should be used for configuring WiFi on the command line.

- Edit `/etc/network/interfaces` and write:
  ```
  auto wlan0 iface wlan0 inet dhcp wpa-ssid <ssid> wpa-psk <password>
  ```
- After that write and close file and use command:
  ```
  dhclient wlan0
  ```

Or if it hangs use:
`ifup wlan0`

5 SOFTWARE SUPPORT

5.1 Implementation of DonkeyCar Software Distribution

`Donkey-distribution` is a high level self driving library written in Python. Donkeycar has components to install on a host PC. This can be a laptop, or desktop machine. Donkeycar software components need to be installed on the robot platform of your choice. We implemented it on Raspberry Pi 3B+ and NVIDIA Jetson Nano [25]. After install, you will create the Donkeycar application from a template. This contains code which can be customized according to our requirements. Next we will train the Donkeycar to drive on it's own based on our driving style. This uses a supervised learning technique often referred to as behavioral cloning.

Following steps are required to be followed to implement it :

- Install Software on Host PC
  This includes installing miniconda and optional Tensorflow GPU and getting the latest donkeycar on to the host and creating its local working directory.
- Install Software On Donkeycar
  This includes installing dependencies, setting up virtual environment, installing opencv and donkeycar python code.
- Create donkeycar application from template using following command :
  `$ donkey createcar --path ~/mycar`
- Configure Options
  There is a configuration file named `myconfig.py` in newly created directory, `~/mycar`. In this file each line has a comment mark. The commented text shows the default value. If we want to make an edit to over-write the default, uncomment the line by removing the # and any spaces before the first character of the option. We need to change the default value of steering and throttle PWM and other defaults according to our requirement. Our modified `myconfig.py` is uploaded here [26].
- Configure PCA9685, Joystick and Camera
- Calibration of Car
  The point of calibrating the car is to make it drive consistently. The car is calibrated to get the appropriate steering PWM values for left and right and throttle PWM value for forward, backward and zero.

  Our Calibrated Values:

  Steering Calibration
  Full Left: 420
  Full Right: 345
  Middle/Straight: 385

  Throttle Calibration:
  Forward Throttle to just Start: 393
  Zero Throttle: 381
  Reverse Throttle:341

- Drive the car manually for 10-20 laps and collect the data. To drive the car in manual mode, one needs to run manage.py file, as follows `python3 manage.py drive`
- Train the model using the collected data.

5.1.1 Challenges Faced

We implemented the Donkeycar both on Raspberry Pi and Jetson Nano to collect the data but the collected data cannot be used to train the model as the quality of images from Pi camera were very bad and the training could not converge.

5.2 Chinkara Software Distribution

Due to the failure of Donkeycar, we decided to use our stereo camera zed mini for capturing images and collection of data. But zed camera is not compatible with donkeycar distribution. So, we developed our own software distribution named `Chinkara`, and its code is available here [27]. For this, we developed the ROS nodes for all the sensors and hardware components we have used like zed mini stereo camera, GPS, Lidar, joystick controller and for actuation using servo driver and ESC. We created a launch file to launch all the required ROS nodes simultaneously. All the required parameters are setup in the launch file so a user can edit the launch file to make the required changes in parameters. When launched, all the ROS nodes will start publishing sensor messages to the corresponding topics which can be collected using the following command :

```
   $ rosbag record /joy /scan /zed/zed_node/depth/camera_info
/zed/zed_node/imu/data_raw /zed/zed_node/left/image_rect_color
/zed/zed_node/right/image_rect_color /zed/zed_node/depth/depth_registered
```

5.2.1 Challenges Faced

1. Usually due to the voltage drop in the battery after sometime, the maximum throttle that we set is not enough to drive the car. Hence, if the battery becomes low, increase the max throttle pulse to increase the PWM to the motor

```
 <arg name="max_throttle_pulse" default="520" />
```

To change the steering angles change the corresponding left and right steering pulse

```
<arg name="max_steering_pulse" default="390" />
<arg name="min_steering_pulse" default="345" />
```

Our zed mini camera processes a lot of data and is publishing heavy data on ROS topics. This is creating latency while controlling the car manually using joystick. Messages are dropped at the publishing nodes and system becomes unresponsive. sar system profiler showed 20-30% of iowait by the rosbag record.

We initially issued the command `rosbag record -a` for recording the messages and later issued `rosbag record -a -x ``(.*)/compressed(.*)''` command which allows compressing the messages [28].

Further, we recorded only specific topics which are relevant

```
$ rosbag record /joy /scan /zed/zed_node/depth/camera_info \
/zed/zed_node/imu/data_raw /zed/zed_node/left/image_rect_color \
/zed/zed_node/right/image_rect_color /zed/zed_node/depth/depth_registered
```

Furthermore, we removed stereo camera node from launch file. On applying these optimizations, the system is now responsive.

6 Track Building

Building a track was necessary to collect the dataset for the training of the car. We made multiple attempts for building the track, as follows:

1. Track on the floor:
Track was constructed in C-block first floor open balcony (south), as shown in Figure 15 and the dataset was collected using donkey car software distribution. But while training, the model did not converge. This may be due to the lines in floor tiles' joints.
2. Track on polythene sheet:
Track was constructed on polythene sheet in C-block third floor corridor but the illumination was not sufficient as visible in left-side figure of Figure 16. Track was moved to C-block first floor sit-out area (north) though reflection from daylight on the track and creases on the sheet posed some challenges as visible in the right-side figure of Figure 16. The angle of camera to the track also seemed to be playing an important role. We need a dedicated area for making a track where we can conduct experiments over several days.

Fig. 15. Track made on floor

Fig. 16. Track made on Polythene Sheet on third-floor (left) and first-floor (right) of C-block building

7 Concluding Remarks

In our future work, we plan to implement the following capabilities and perform the following experiments.

- We plan to use Intel Movidius neural compute stick for accelerating AI algorithms. This can be plugged to RPi and it provides much higher performance than the CPU of RPi. Similarly, we also plan to experiment with Raspberry Pi version 4, Coral Edge TPU and FPGA-based accelerators [29] such a PYNQ. Our long term goal is to compare the performance of GPU, FPGA and ASIC platforms.
- Currently, the playback of recorded video is not working due to differences in the voltage.
- The rating of our battery is 8V, 1100 mAh which means that it can provide 1.1A current for 1 hour. However, the rating of Nano is 5V, 4A. Thus, due to the power consumption of Nano, battery's life will be reduced to 15 minutes.
- We also plan to test with various techniques for managing power consumption of Jetson Nano and other accelerators [30, 31].

Developing a toy RC car testbed for ADV requires deep knowledge of both hardware and software. In this paper, we reported the development of a hardware-software stack for enabling ADV. Our experiences and insights will be highly helpful for future researchers and practitioners and will shorten their development time.

References

[1] I. Info, "I2C Info I2C Bus, Interface and Protocol," https://i2c.info/, 2019.

[2] "DonkeyCar," http://docs.donkeycar.com/.
[3] E. Upton and G. Halfacree, *Raspberry Pi user guide*. John Wiley & Sons, 2014.
[4] "Jetson Nano," https://developer.nvidia.com/embedded/jetson-nano-developer-kit, 2019.
[5] S. Mittal, "A Survey on Optimized Implementation of Deep Learning Models on the NVIDIA Jetson Platform," *Journal of Systems Architecture*, 2019.
[6] S. Mittal and J. Vetter, "A Survey of CPU-GPU Heterogeneous Computing Techniques," *ACM Computing Surveys*, vol. 47, no. 4, pp. 69:1–69:35, 2015.
[7] S. Mittal, "A Survey Of Techniques for Architecting and Managing Asymmetric Multicore Processors," *ACM Computing Surveys*, vol. 48, no. 3, pp. 45:1–45:38, 2016.
[8] S. Mittal and M. S. Inukonda, "A Survey of Techniques for Improving Error-Resilience of DRAM," *Journal of Systems Architecture*, 2018.
[9] S. Mittal and S. Vaishay, "A Survey of Techniques for Optimizing Deep Learning on GPUs," *Journal of Systems Architecture*, 2019.
[10] S. Mittal, "A Survey of Techniques for Architecting and Managing GPU Register File," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2016.
[11] "Robot operating system (ROS)," http://wiki.ros.org/ROS/Introduction, 2019.
[12] Stereolabs, "Zed camera," https://www.stereolabs.com/zed/, 2018.
[13] http://www.catb.org/gpsd/NMEA.html.
[14] T. Corporation, "MPU-6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTracking Devices," https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/, 2019.
[15] Slamtec, "RPLIDAR A1 360 degrees laser scanner," http://www.slamtec.com/en/lidar/a1, 2019.
[16] Logitech, "F710 Wireless GAmepad," https://www.logitechg.com/en-in/products/gamepads/f710-wireless-gamepad.html, 2019.
[17] Logitech, "F710 Wireless GAmepad," https://www.logitech.com/assets/34885/3/f710620-002923003403gswamr.pdf.
[18] N. Semiconductors, "PCA9685: 16-channel, 12-bit PWM Fm+ I2C-bus LED controller," https://www.nxp.com/products/power-management/lighting-driver-and-controller-ics/ic-led-controllers/16-channel-12-bit-pwm-fm-plus-ic-bus-led-controller:PCA9685, 2019.
[19] Stereolabs, "Getting Started with ROS," https://www.stereolabs.com/docs/ros/.
[20] http://wiki.ros.org/ublox.
[21] https://github.com/KumarRobotics/ublox.
[22] http://wiki.ros.org/nmea_navsat_driver.
[23] "RPLidar ROS wiki," http://wiki.ros.org/rplidar.
[24] "RPLidar ROS package," https://github.com/Slamtec/rplidar_ros.
[25] http://docs.donkeycar.com/guide/robot_sbc/setup_jetson_nano/.
[26] https://bit.ly/2IuiA0g.
[27] "Chinkara : Autonomous driving research platform stack used at CANDLE Research Lab, Department of CSE, IIT Hyderabad, India," https://github.com/iithcandle/chinkara, 2019.
[28] https://answers.ros.org/question/251403/image_transport-rosbag-issue/.
[29] S. Mittal, "A Survey of FPGA-based Accelerators for Convolutional Neural Networks," *Neural computing and applications*, 2018.
[30] S. Mittal, "A Survey of Techniques For Improving Energy Efficiency in Embedded Computing Systems," *International Journal of Computer Aided Engineering and Technology (IJCAET)*, vol. 6, no. 4, pp. 440–459, 2014.
[31] S. Mittal and J. Vetter, "A Survey of Methods for Analyzing and Improving GPU Energy Efficiency," *ACM Computing Surveys*, 2015.

APPENDIX A

```
import smbus                    #import SMBus module of I2C
from time import sleep          #import

#some MPU6050 Registers and their Address
```

```
PWR_MGMT_1   = 0x6B
SMPLRT_DIV   = 0x19
CONFIG       = 0x1A
GYRO_CONFIG  = 0x1B
INT_ENABLE   = 0x38
ACCEL_XOUT_H = 0x3B
ACCEL_YOUT_H = 0x3D
ACCEL_ZOUT_H = 0x3F
GYRO_XOUT_H  = 0x43
GYRO_YOUT_H  = 0x45
GYRO_ZOUT_H  = 0x47


def MPU_Init():
        #write to sample rate register
        bus.write_byte_data(Device_Address, SMPLRT_DIV, 7)

        #Write to power management register
        bus.write_byte_data(Device_Address, PWR_MGMT_1, 1)

        #Write to Configuration register
        bus.write_byte_data(Device_Address, CONFIG, 0)

        #Write to Gyro configuration register
        bus.write_byte_data(Device_Address, GYRO_CONFIG, 24)

        #Write to interrupt enable register
        bus.write_byte_data(Device_Address, INT_ENABLE, 1)

def read_raw_data(addr):
        #Accelero and Gyro value are 16-bit
        high = bus.read_byte_data(Device_Address, addr)
        low = bus.read_byte_data(Device_Address, addr+1)

        #concatenate higher and lower value
        value = ((high << 8) | low)

        #to get signed value from mpu6050
        if(value > 32768):
                value = value - 65536
        return value


bus = smbus.SMBus(1)     # or bus = smbus.SMBus(0) for older version boards
Device_Address = 0x68    # MPU6050 device address

MPU_Init()

print (" Reading Data of Gyroscope and Accelerometer")

n=0
Ax_cal=0
Ay_cal=0
```

```
Az_cal=0
Gx_cal=0
Gy_cal=0
Gz_cal=0
while n<100:

        #Read Accelerometer raw value
        acc_x = read_raw_data(ACCEL_XOUT_H)
        acc_y = read_raw_data(ACCEL_YOUT_H)
        acc_z = read_raw_data(ACCEL_ZOUT_H)

        #Read Gyroscope raw value
        gyro_x = read_raw_data(GYRO_XOUT_H)
        gyro_y = read_raw_data(GYRO_YOUT_H)
        gyro_z = read_raw_data(GYRO_ZOUT_H)

        #Full scale range +/- 250 degree/C as per sensitivity scale factor
        Ax = acc_x/16384.0
        Ay = acc_y/16384.0
        Az = acc_z/16384.0

        Gx = gyro_x/131.0
        Gy = gyro_y/131.0
        Gz = gyro_z/131.0

        print ("Gx=%.2f" %Gx, "Gy=%.2f" %Gy, "Gz=%.2f" %Gz,"Ax=%.2f g" %Ax, "Ay=%.2f g

        Ax_cal=Ax_cal+Ax
        Ay_cal=Ay_cal+Ay
        Az_cal=Az_cal+Az
        Gx_cal=Gx_cal+Gx
        Gy_cal=Gy_cal+Gy
        Gz_cal=Gz_cal+Gz
        n=n+1
        sleep(0.001)

Ax_cal=Ax_cal/100
Ay_cal=Ay_cal/100
Az_cal=Az_cal/100
Gx_cal=Gx_cal/100
Gy_cal=Gy_cal/100
Gz_cal=Gz_cal/100
print ("Calibrated Values")
print ("Gx_cal=%.2f" %Gx_cal, "Gy_cal=%.2f" %Gy_cal, "Gz_cal=%.2f" %Gz_cal,
"Ax_cal=%.2f g" %Ax_cal, "Ay_cal=%.2f g" %Ay_cal, "Az_cal=%.2f g" %Az_cal)

while True :
         #Read Accelerometer raw value
        acc_x = read_raw_data(ACCEL_XOUT_H)
        acc_y = read_raw_data(ACCEL_YOUT_H)
        acc_z = read_raw_data(ACCEL_ZOUT_H)

        #Read Gyroscope raw value
```

```
gyro_x = read_raw_data(GYRO_XOUT_H)
gyro_y = read_raw_data(GYRO_YOUT_H)
gyro_z = read_raw_data(GYRO_ZOUT_H)

#Full scale range +/- 250 degree/C as per sensitivity scale factor
Ax = (acc_x/16384.0) - Ax_cal
Ay = (acc_y/16384.0) - Ay_cal
Az = (acc_z/16384.0) - Az_cal

Gx = (gyro_x/131.0) - Gx_cal
Gy = (gyro_y/131.0) - Gy_cal
Gz = (gyro_z/131.0) - Gz_cal

print("Gx=%.2f" %Gx,"Gy=%.2f" %Gy,"Gz=%.2f"%Gz,
"Ax=%.2f g" %Ax,"Ay=%.2f g"%Ay,"Az=%.2f g"%Az)
sleep(0.1)
```