Haaga Cleaning Robot

Guide Manual

Ver. 2

Written by Jeongmin 2016/4/26

Table of contents

1. Overview

The autonomous Haaga cleaning robot has two front brushes and each brush rotates in opposite directions grabbing the debris directly in front of the sweeper while the brush roller on the bottom picks up all the fine particles.

This manual provides both the general and technical details you need to operate your Haaga cleaning robot and to begin developing robot hardware and software.

The default firmware is encoded. In order to change the robot algorithm, the firmware needs to be changed, compiled and reprogrammed as explained in this github.

2. Acknowledgement

The hardware design was originally based on the Haaga-697 sweeper.

Haaga cleaning robot now uses the Bluno mega 2560 board to control DC motors for autonomous driving, replacing hand-powered operation.

3. Haaga System

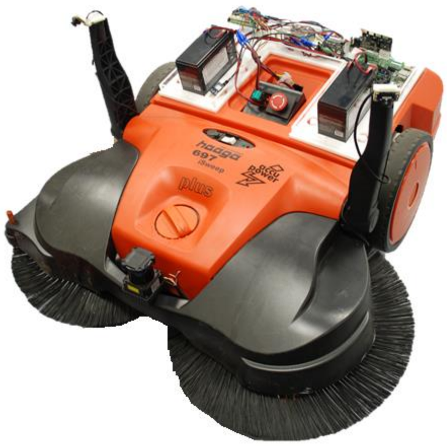


Figure 1. Hardware configuration

3.1 Basic components

- Haaga-697 (modified)
- Battery charger
- DC motor x2
- Encoder x2
- 12V Battery x3
- Fisheye camera x2
- Lidar
- Arduino board with Bluetooth module
- IMU sensor
- GPS sensor
- Single board computer
- Motor drive
- Emergency stop button
- Power switch

3.2 System architecture

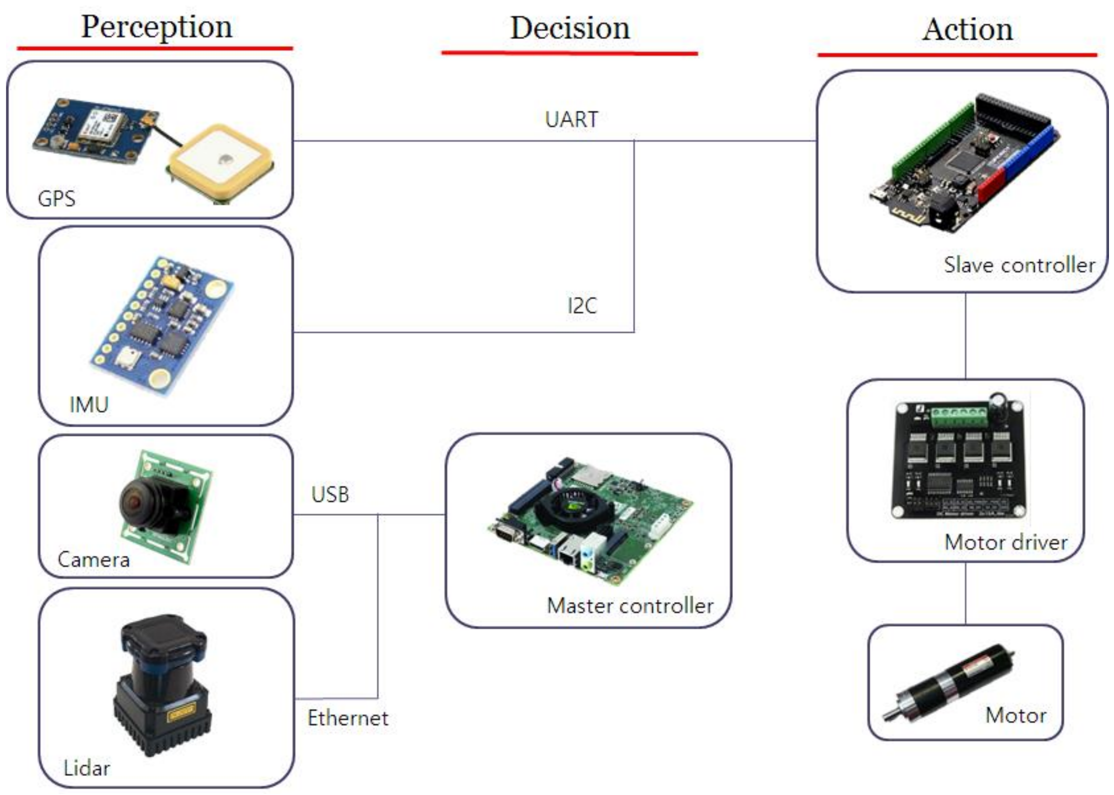
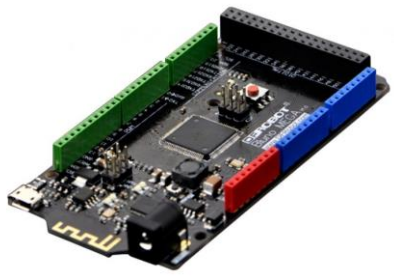


Figure 2. System architecture

There are two controllers for autonomous driving of cleaning robot. Master and slave controller are connected with serial port via micro USB. GPS and IMU sensor are connected to slave controller for localization, and Camera and Lidar are connected to master controller to recognize objects.

3.3 Hardware Specification

3.3.1 Slave Control Board

	**Bluno mega 2560 (Arduino+BLE)** · Microcontroller：ATmega2560 · Clock frequency: 16MHZ · Working voltage：+5V · External input voltage (recommended)：7 ~ 12V DC · External input voltage (range)：6 ~ 20V DC Digital I/O outputs: 54 (14 PWM outputs) Analog inputs: 16 I/O current: 40mA Flash capacity: 256KB (4k for bootloader) Static storage capacity of SRAM: 8KB Memory capacity of EEPROM: 4KB

Bluno Mega2560 inherits the numerous ports and the abundant resources of Mega series, and adds Bluetooth 4.0 wireless communication function. It has 54 digital I/O (input/output) ports (15 of which can be used as PWM output), 16 analog input and 4 UART (hardware serial ports), and uses 16 MHz imported crystal oscillators. The following figure shows Bluno mega controller with connected ports, which includes:

- Power: 12V Battery power
- Serial connection with Master controller
- 4 interrupted pin with Encoders
- 2 Digital input with GPS
- 2 Digital input with Gyro
- 4 Digital output with motor driver

where the numbering 1 means right and 2 means left. (motor1=right motor)

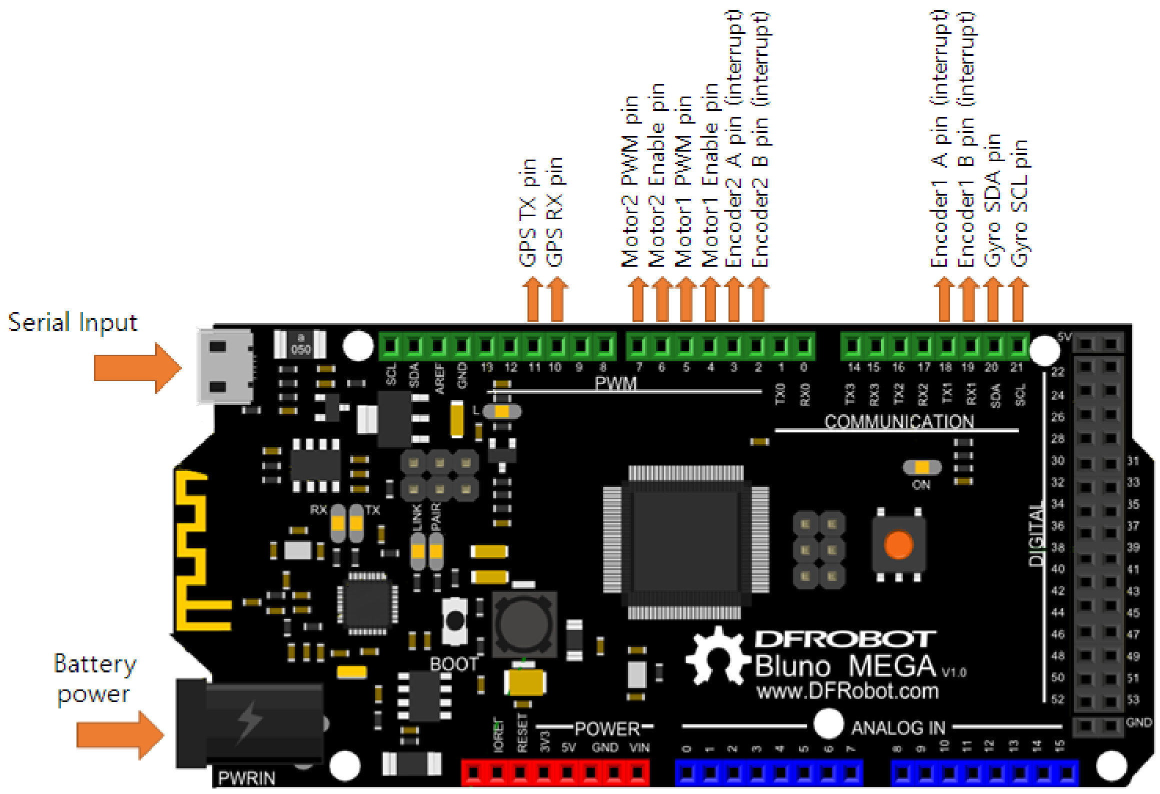


Figure 3. Bluno mega pin out

Bluno mega have the same serial port configuration, with only adding a serial port for Bluetooth 4.0 which uses Serial0 wireless passthrough. Because bluno mega has four serial ports, if you want to upload your code, use Serial2 or Serial3 ports (pin 14,15,16,17).

This board is also used to PID control of the DC motors via motor driver and acquiring GPS and IMU sensor’s data. The default loop frequency is 20Hz. By the commands through the serial port, you can change the PID gains and motor speed. The valid commands are as follow table. All the commands’ type is string and the commands should be end with ‘E’ character. And every loop, Bluno send the robot status that containing motor speed, encoder counts, GPS coordinates and heading via serial port. The type of output is also string and each value are divided by comma.

Table 1. Commands Lists

Input Commands	Description	Type
M+E	Mode change to Bluetooth	String
M-E	Mode change to Automatic	String
S+E	Increase motors acceleration	String
S-E	Decrease motors acceleration	String
P+E	Increase ‘P’ gain of Motor1	String
P-E	Decrease ‘P’ gain of Motor1	String
O+E	Increase ‘I’ gain of Motor1	String
O-E	Decrease ‘I’ gain of Motor1	String
I+E	Increase ‘D’ gain of Motor1	String
I-E	Decrease ‘D’ gain of Motor1	String
L+E	Increase ‘P’ gain of Motor2	String
L-E	Decrease ‘P’ gain of Motor2	String
K+E	Increase ‘I’’ gain of Motor2	String
K-E	Decrease ‘I’ gain of Motor2	String
J+E	Increase ‘D’ gain of Motor2	String
J-E	Decrease ‘D’ gain of Motor2	String
R[value]L[value]E	Set motor’s RPM. For example, R100L100E command means that setting the RPM of right motor and left motor to 100. The maximum RPM value of the motor is 300 (-300 ~ 300).	String

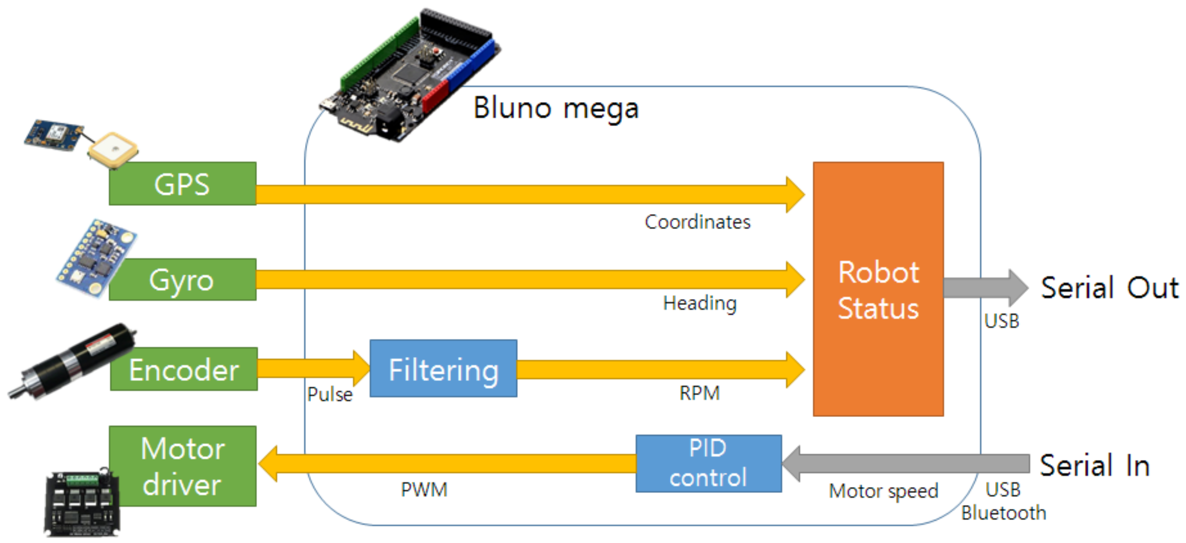


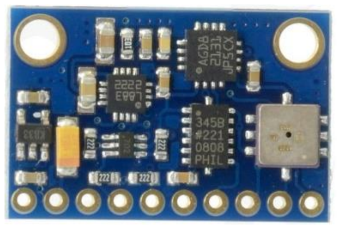
Figure 4. Flow Diagram of Slave Controller

3.3.2 GPS

	Ublox neo 6m · u-blox NEO-6M onboard, with high-gain active antenna · IPX interface, for connecting different active antennas · Chargeable backup battery, keeps the ephemeris data when power down, supports hot starts · Onboard EEPROM for storing config information

Receiver type	50 channels, GPS L1(1575.42Mhz) C/A code, SBAS:WAAS/EGNOS/MSAS
Horizontal position accuracy	2.5mCEP (SBAS:2.0mCEP)
Navigation update rate	5Hz maximum (1HZ default)
Capture time	Cool start: 27s (fastest) ; Hot start: 1s
Tracking & Navigation sensitivity	-161dBm
Communication protocol	NMEA(default)/UBX Binary
Serial baud rate	4800, 9600(default), 19200, 38400, 57600, 115200, 230400
Operating temperature	-40℃ ~ 85℃
Operating voltage	2.7V~5.0V(power supply input via VCC)
Operating current	45mA
TXD/RXD impedance	510Ohms

3.3.3 IMU

	**Gy-801** · Three selectable full scales (250/500/2000dps) · Factor in all g-ranges, up to 13-bit resolution at ±16 g · 12-Bit ADC Coupled with Low Noise ARM Sensors Achieves 5 milli-gauss Resolution in ±8 Gauss Fields · Pressure range: 300 to1100hPa (+9000m to -500m above sea level) · I2C interfaces · Power supply: 3-5v

This IMU 1DOF is based on the sensor **L3G4200D**, **ADXL345**, **HMC5883L** and **BMP180**.

The **L3G4200D** is a low-power three-axis angular rate sensor able to provide unprecedented stablility of zero rate level and sensitivity over temperature and time. It includes a sensing element and an IC interface capable of providing the measured angular rate to the external world through a digital interface (I2C/SPI).

The **ADXL345** is a small, thin, low power, three-axis accelerometer with high resolution (13-bit) measurement up to 16g. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4mg/LSB) enables resolution of inclination changes of as little as 0.25°. The Honeywell HMC5883L is designed for low-field magnetic sensing with a digital interface for applications such as low-cost compassing and magnetometry

The **HMC5883L** includes high-resolution HMC118X series magneto-resistive sensors plus an ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, and a 12-bit ADC that enables 1° to 2° compass heading accuracy.

The **BMP180** is the function compatible successor of the BMP085, a new generation of high precision digital pressure sensors for consumer applications. The ultra-low power, low voltage electronics of the BMP180 is optimized for use in mobile phones, PDAs, GPS navigation devices and outdoor equipment.

3.3.4 Motor

	Geared Motor IG-52GM 04TYPE (24V) Rated Voltage: DC 24V Rated Torque: Max. 100Kg-cm Size: Ø54.3 * L 95.3mm Shaft: Ø12mm, key-way 5 x 18mm Reduction Ratio: 1/12 Rated Speed: 285 RPM No Load Speed: 333 RPM

04 TYPE MOTOR (DC 24V)	
Rated torque	1,300 (gf-cm)
Rated speed	3,550 (RPM)
Rated current	2,850 (mA)
No load speed	4,000 (RPM)
No load current	700 (mA)
Rated output	48.6 (W)

3.2.5 Encoder

	CUI AMT103 · patented capacitive technology · 16 dip switch programmable resolutions · straight (radial) and right-angle (axial) versions · 9 different mounting hole options for AMT102 · 6 different mounting hole options for AMT103 · low power consumption · line driver output available (CUI-10XE-10) · 40°C to 100°C operating temperature · TTL voltage output · modular package

Output phase difference	**90° (A ch leads B ch in CW direction viewed from front)**
Output current	5mA max.
Output waveform	TTL voltage square wave
Output signals	A, B, Z1 phase (A, B, C line driver available with CUI-10XE-10)
Current consumption	6 mA typ., 10 mA max.
Supply voltage	3.6 ~ 5.5 V dc
Output resolution (ppr)	48, 96, 100, 125, 192, 200, 250, 256, 384, 400, 500, 512, 800, 1000, 1024, 2048
Frequency response	250 kHz max
Accuracy	±15 arc min (at 192, 384, 400, 500, 800, 1000, 1024, 2048 ppr) ±30 arcmin (at 96, 200, 250, 512 ppr) ±60 arcmin (at 48, 100, 125, 256 ppr)
Max. rotational speed	7500 rpm (at 2048, 1000, 800, 384 ppr) 15000 rpm (at 1024, 500, 400, 192 ppr) 30000 rpm (at 48, 96, 100, 125, 200, 250, 256, 512 ppr)

3.2.6 Motor Driver

	DC Motor driver 2x15A_lite · Input Voltage:4.8-35V · Maximum output current: 15A@13.8V per channel · Peak output current: 20A@13.8V per channel · PWM capability: up to 25 kHz · Interfaces: 4 digital IO(2 PWM output include) · Driving mode: Dual high-power H-bridge driver · Other specifications: · Galvanic isolation to protect the microcontroller · Dual current detection diagnostic functions · Short circuit, overheating, over-voltage protection · Size:73x68x14mm · For applications of more than 15A per channel · Fast switching might damage the board, best to smooth it by software · Avoid higher rating motors, and use lower PWM whenever possible

3.2.7 Master Control Board

	NVIDIA Jetson tk1 · File system based in Ubuntu Linux kernel. · Tegra K1 SOC (CPU+GPU+ISP in a single chip): **GPU**: NVIDIA Kepler "GK20a" GPU with 192 SM3.2 CUDA cores **CPU**: NVIDIA "4-Plus-1" 2.32GHz ARM quad-core Cortex-A15 CPU

Hardware Features:
Dimensions: 5" x 5" (127mm x 127mm) board **DRAM**: 2GB DDR3L 933MHz EMC x16 using 64-bit data width **Storage**: 16GB fast eMMC 4.51 (routed to SDMMC4) **mini-PCIe**: a half-height single-lane PEX slot **SD/MMC card**: a full-size slot (routed to SDMMC3) **USB 3.0**: a full-size Type-A female socket **USB 2.0**: a micro-AB female socket **HDMI**: a full-size port **RS232**: a full-size DB9 serial port (routed to UART4) **Audio**: an ALC5639 Realtek HD Audio codec with Mic in and Line out jacks **Ethernet**: a RTL8111GS Realtek 10/100/1000Base-T Gigabit LAN port using PEX **SATA**: a full-size port that supports 2.5" and 3.5" disks, but is not hot-pluggable **JTAG**: a 2x10-pin 0.1" port for professional debugging **Power**: a 12V DC barrel power jack and a 4-pin PC IDE power connector **Fan**: a fan-heatsink running on 12V
The following signals are available through the 125-pin 2mm-pitch expansion port:
Camera ports: 2 fast CSI-2 MIPI camera ports (one 4-lane and one 1-lane) **LCD port**: LVDS and eDP Display Panel **Touchscreen ports**: Touch SPI 1 x 4-lane + 1 x 1-lane CSI-2 **UART** **HSIC** **I2C**: 3 ports

GPIO: 7 x GPIO pins (1.8V). Camera CSI pins can also be used for extra GPIO
Hardware-accelerated APIs supported:
CUDA 6.0 (SM3.2, roughly the same as desktop SM3.5) **OpenGL 4.4** **OpenGL ES 3.1** **OpenMAX IL multimedia codec** including H.264, VC-1 and VP8 through Gstreamer **NPP** (CUDA optimized NVIDIA Performance Primitives) **OpenCV4Tegra** (NEON + GLSL + quad-core CPU optimizations) **VisionWorks**

In this version of Haaga, **Jetson tk1** board is used to master controller for the object recognition and path planning by vision based algorithm. Jetson with GPU-accelerated parallel processing is the embedded visual computing platform. It features high-performance, low-energy computing for deep learning and computer vision making the Jetson platform ideal for compute-intensive embedded projects like drones, autonomous robotic system. Although vision algorithms are not encoded, you can freely write the codes using default systems by additional USB ports.

This board is connected with Bluno via serial communication. If you send the desired motor speed value as string command via serial communication, status of robot is returned as 1d array of string.

3.2.8 Camera

	Fisheye CMOS camera Connector: USB 2.0 FOV: 170~190 deg Resolution: 640x480 1280x960 1920x1080

FOV	170 Degee
Sensor	1/3" CMOS ov2710
Lens	200w, 2.0 mega pixels, 1920(H) X1080(V) pixels
Max. Resolution	1920x1080
Image Format	MJPEG
USB protocol	USB2.0 HS/ FS, usb1.1 FS
Driver support	USB Video class(UVC) 1.1
Auto exposure	Support
Auto white balance	Support
Frame rates	1920(H) x 1080(V) pixel, MJPEG 30 fps, yuy2 6 fps 1280(H) x 1024(V) pixel, MJPEG 30 fps, yuy2 6 fps 1280(H) x 720(V) pixel, MJPEG 60 fps, yuy2 9 fps 1024(H) x 768(V) pixel, MJPEG 30 fps, yuy2 9 fps 800(H) x 600(V) pixel, MJPEG 60 fps, yuy2 21 fps 640(H) x 480(V) pixel, MJPEG 120 fps, yuy2 30 fps 352(H) x 288(V) pixel, MJPEG 120 fps, yuy2 30 fps 320(H) x 240(V) pixel, MJPEG 120 fps, yuy2 30 fps
Voltage	DC 5V
Current	150mA
Operating temperature	Degree (- 20~70)
Dimension	32x32mm/ 38*38
OS Support	winxp/ vista/ win7/ Win8 Linux UVC

3.2.9 Lidar

	Hokuyo UTM-30LX-EW · Supply voltage 12V · Measurement distance 30m · Field of view 270° · Interface Ethernet · Multi-echo support

Light Source	Laser Semiconductor λ = 905nm Laser Class 1
Power Source	12VDC ± 10%
Supply Current	MAX: 1A, Normal : 0.7A
Power Consumption	Less than 8W
Detection Range and Detection Object	Guaranteed Range:0.1 to 30m Maximum Range:0.1 to 60m Minimum detectable width at 10m : 130mm
Accuracy	0.1 to 10m: ±30mm,10 to 30m : ±50mm Under 3000lx : White Kent Sheet : ±30mm*1(0.1 to 10m) Under 100000lx : White Kent Sheet : ±50mm*1(0.1 to 10m)
Measurement	1mm
Resolution and Repeated Accuracy	0.1 to 10m: σ<10mm, 10 to 30m: σ<30mm Under 3000lx: σ=10mm Under 100000lx: σ=30
Scan Angle	270°
Angular Resolution	0.25°(360°/1440)
Scan Speed	25ms (Motor speed: 2400rpm)
Interface	Ethernet 100BASE-TX(Auto-negotiation)
Output	Synchronous Output 1-Point
Ambient Condition (Temperature, Humidity)	-10 to +50 degrees C Less than 85%RH (Without Dew, Frost)
Storage Temperature	-25 to +75 degrees C

4. ArduHaaga Firmware

The latest firmware for Haaga cleaning robot is available from the github. This page provides additional links to a number of specific builds that are considered “significant” - for example, the last builds of firmware to fit on the **Bluno mega 2560** board.

4.1 Compile and Upload

To compile ‘ArduHaaga.ino’, you have to install following tools in the Linux kernel.

Tools need

- Arduino IDE
- TinyGPS
- HMC5883L
- ROS
  - ROSSerial

TinyGPS is library to acquire coordinates from GPS sensor and **HMC5883L** is library for robot’s heading from IMU.

The **ROS** (Robot Operating System) is a set of software libraries and tools that help you build haaga applications.

The **ROSSerial** is a protocol for wrapping standard ROS serialized messages and multiplexing multiple topics and services over a character device such as a serial port. Installation guide is in the Chapter 7.

4.2 Source

4.2.1 Functions

void setup()
The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.
void loop()
After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.
void rencoder1() & void rencoder2()
Get the encoder counter value from interrupt pins
void checkEvent()
In the every loop, check the commands which come from serial port.
void calculateRPM(String **command**)
From the command, check the desired value of motor speed.
void speedcalculation(unsigned long **dt**)
Calculate motor speed using encoder value **Parameters** · **dt** frequency of the firmware loop.
void buildVelProfile(float **desired_vel**, float **current_vel**, float ***speed_profile**)

Build the trapezoidal velocity profile of the motor.

Parameters

- **desired_vel** input value of the motor speed from string command.
- **current_vel** calculated value of the motor speed from encoder.
- **speed_profile** result value of the motor speed.

float updatePid(float **targetValue**, float **currentValue**, unsigned long **dt**, float **Kp**, float **Ki**, float **Kd**)

PID control function for the desired input data

Parameters

- **targetValue** desired value of the motor speed.
- **current_Value** calculated value of the motor speed from encoder.
- **dt** frequency of the firmware loop.
- **Kp** P gain
- **Ki** I gain
- **Kd** D gain

Return

PWM value of the motor

float lpFilter(float **value**, float **prev_value**, float **beta**)

Pass the signals of the encoder value using Low-pass algorithm

Parameters

- **value** raw data
- **prev_value** filtered previous data
- **beta** filtered coefficient

Return

Filtered data

void gpsdump(TinyGPS **&gps**)

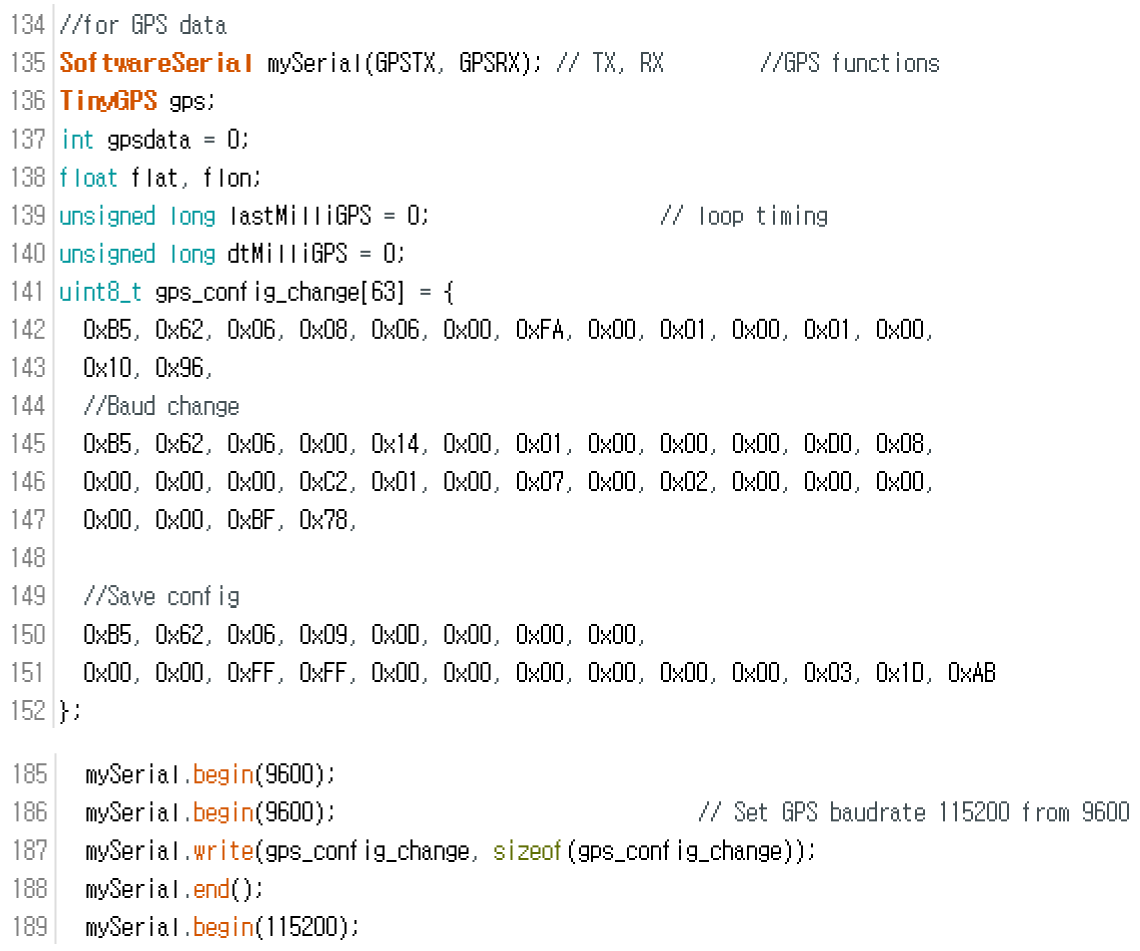
Create the TinyGPS object and get the GPS coordinates.

void callback(const Test::Request **&req**, Test::Response **&res**)
Check the request and response messeages from the ROS topic
void publishStatus()
Send the robot’s status (motor speed, heading, coordinates) to the ROS topic.

4.2.2 The Code Explained

```
#include <SoftwareSerial.h>
#include <Wire.h>
#include <TinyGPS.h>
#include <HMC5883L.h>
#include <ros.h>
#include <std_msgs/String.h>
#include <rosserial_arduino/Test.h>
```

As a part of every ROS Arduino program, you need to include the ‘ros.h’ header file and header files for string messages that you will be using.



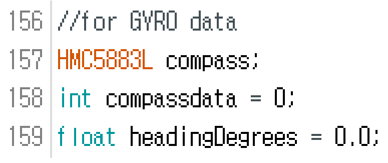
```
//for GPS data
SoftwareSerial mySerial(GPSTX, GPSRX); // TX, RX        //GPS functions
TinyGPS gps;
int gpsdata = 0;
float flat, flon;
unsigned long lastMilliGPS = 0;                     // loop timing
unsigned long dtMilliGPS = 0;
uint8_t gps_config_change[63] = {
  0xB5, 0x62, 0x06, 0x08, 0x06, 0x00, 0xFA, 0x00, 0x01, 0x00, 0x01, 0x00,
  0x10, 0x96,
  //Baud change
  0xB5, 0x62, 0x06, 0x00, 0x14, 0x00, 0x01, 0x00, 0x00, 0x00, 0xD0, 0x08,
  0x00, 0x00, 0x00, 0xC2, 0x01, 0x00, 0x07, 0x00, 0x02, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xBF, 0x78,

  //Save config
  0xB5, 0x62, 0x06, 0x09, 0x0D, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x1D, 0xAB
};
```

```
  mySerial.begin(9600);
  mySerial.begin(9600);                              // Set GPS baudrate 115200 from 9600
  mySerial.write(gps_config_change, sizeof(gps_config_change));
  mySerial.end();
  mySerial.begin(115200);
```

Figure 5. Initialize GPS serial communication

The above codes are initialization of the GPS sensor in the Bluno board. GPS's TX is connected to 11 pin of Bluno and RX is connected to 10 pin. For the increasing of GPS data transfer speed, GPS baudrate should be 115200 (default is 9600). If GPS baudrate is lower than 115200, main loop frequency will decrease.



```
//for GYRO data
HMC5883L compass;
int compassdata = 0;
float headingDegrees = 0.0;
```

Figure 6. Variables for Gyro sensor

There are many outputs from IMU sensor. In this system, we just get the heading of robot using HMC5883L chip. The type of 'compassdata' variable is Boolean and that variable means status of connection with sensor. The value of robot’s heading will be stored in ‘’headingDegrees’.

```
if (compass.begin())
{
  compassdata = 1;
  compass.setRange(HMC5883L_RANGE_1_3GA);   // Set measurement range
  compass.setMeasurementMode(HMC5883L_CONTINOUS);   // Set measurement mode
  compass.setDataRate(HMC5883L_DATARATE_30HZ);    // Set data rate
  compass.setSamples(HMC5883L_SAMPLES_8);    // Set number of samples averaged
  compass.setOffset(0, 0);    // Set calibration offset. See HMC5883L_calibration.ino
}
else
  compassdata = 0;
```

In the setup function, you can set the variables of measurement range, mode, frequency, samples and offset.

```
// Calculate heading
float heading = atan2(norm.YAxis, norm.XAxis);
float declinationAngle = (8.0 + (21.0 / 60.0)) / (180 / M_PI);
                        // Formula: (deg + (min / 60.0)) / (180 / M_PI);
heading += declinationAngle;
// Correct for heading < 0deg and heading > 360deg
if (heading < 0)
{
  heading += 2 * PI;
}
if (heading > 2 * PI)
{
  heading -= 2 * PI;
}
headingDegrees = heading * 180 / M_PI; // Convert to degrees
```

Calculate heading when the magnetometer is level, then correct for signs of axis. Atan2() function automatically check the correct formula taking care of the quadrant you are in. Once you have robot’s heading, you must add your ‘Declination angle’ which is the error of the magnetic field in your location. Find yours here: http://www.magnetic-declination.com/. If you find your declination, modify the formula of line 354.

```
ros::ServiceServer<Test::Request, Test::Response> server("motor_cmd", &callback);
char message[256] = "";
char tempBuffer[256];
```

```
  // Publish motor status
  publishStatus();
  nh.spinOnce();
```

```
void publishStatus()
{
  if ((millis() - lastMilliPrint) >= dtMillimonitor)
  {
    lastMilliPrint = millis();

    dtostrf(speed_act1_filtered , 4, 0, tempBuffer);
    strcat(message, tempBuffer); strcat(message, ",");
    dtostrf(speed_act2_filtered , 4, 0, tempBuffer);
    strcat(message, tempBuffer); strcat(message, ",");
    dtostrf(count1 , 10, 0, tempBuffer);
    strcat(message, tempBuffer); strcat(message, ",");
    dtostrf(count2 , 10, 0, tempBuffer);
    strcat(message, tempBuffer); strcat(message, ",");

    dtostrf(gpsdata , 1, 0, tempBuffer);
    strcat(message, tempBuffer); strcat(message, ",");
    dtostrf(flat , 5, 5, tempBuffer);
    strcat(message, tempBuffer); strcat(message, ",");
    dtostrf(flon , 5, 5, tempBuffer);
    strcat(message, tempBuffer); strcat(message, ",");
    dtostrf(headingDegrees , 5, 2, tempBuffer);
    strcat(message, tempBuffer); strcat(message, ",");

    status_msg.data = message;
    chatter.publish( &status_msg );
    message[0] = '₩0';
  }
}
```

In the publishStatus function, 'motor_cmd' node publishes status of the Haaga and calls spinOnce() where all of the ROS communication callbacks are handled.

5. Applications

This chapter provides basic information of the applications for Haaga robot.

5.1 Haaga Controller

This application is controller of the Haaga cleaning robot using BLE of the Bluno mega for the Android mobile.

5.1.1 Using the App in your Android

Steps for Android

1. Install the APK file into your Android phone.
2. Run the application.
3. Click the ‘Scan’ button for scanning and select the device.
4. After connected, Click the directional buttons. The default RPM of the motor is 50 and can be changed using the slide.
5. The Bluno will reply with the command. Thus you can see the same "received data"
6. You can send the commands to modify PID gain using text box.
7. Notice that the board RX and TX led will blink when sending and receiving the message.

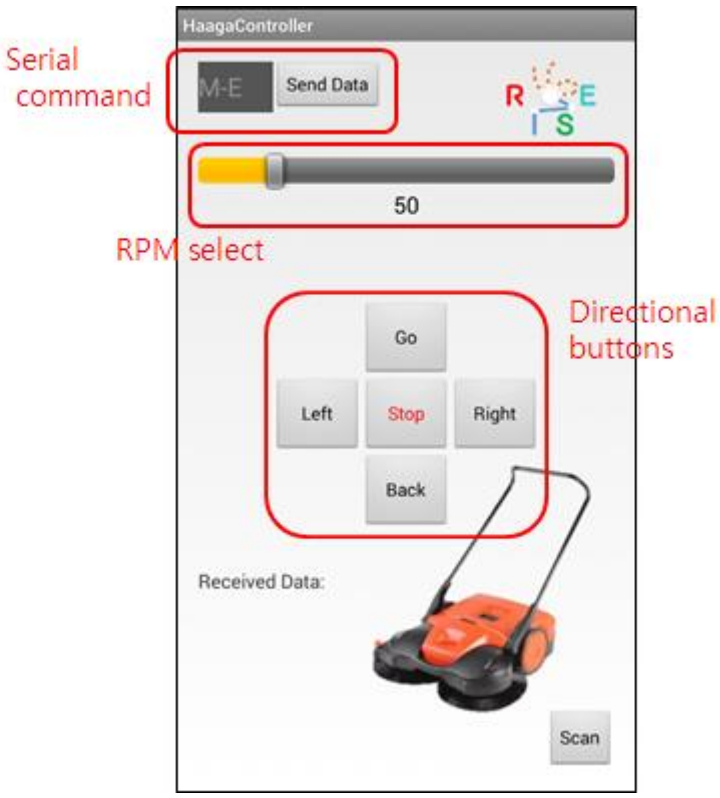


Figure 7. Haaga controller Application

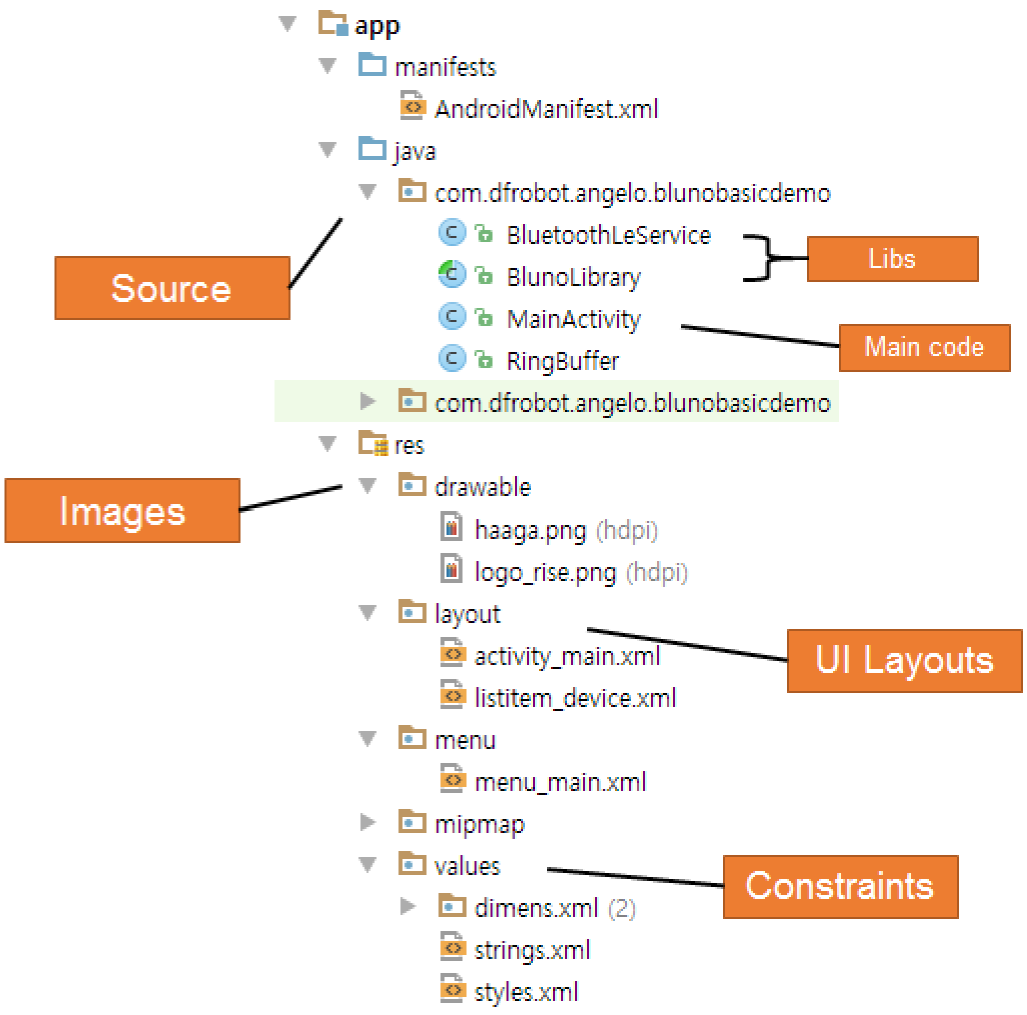
5.1.2 Project setup and Compile

If you want to recompile the application on Android device, you will need the Android Studio with the followings.

Requirements

- Bluno library
- Android 4.3+

Download the source code from the github and open the project 'HaagaController' in the IDE.



```
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ com.dfrobot.angelo.blunobasicdemo
        BluetoothLeService
        BlunoLibrary
        MainActivity
        RingBuffer
    ▶ com.dfrobot.angelo.blunobasicdemo
  ▼ res
    ▼ drawable
        haaga.png (hdpi)
        logo_rise.png (hdpi)
    ▼ layout
        activity_main.xml
        listitem_device.xml
    ▼ menu
        menu_main.xml
    ▶ mipmap
    ▼ values
      ▶ dimens.xml (2)
        strings.xml
        styles.xml
```

Figure 8. File Structure of application

Go to this site for more tutorials of the BlunoLibrary.

http://www.dfrobot.com/wiki/index.php/Bluno_SKU:DFR0267

5.2 Fisheye Camera Calibration

Two fisheye cameras are attached in front of the Haaga. If you want to calibrate the cameras, this application will help you. The source of calibration is in the github.

Requirements

- C++ compilation environments properly set
- OpenCV installed (3.0 or higher)

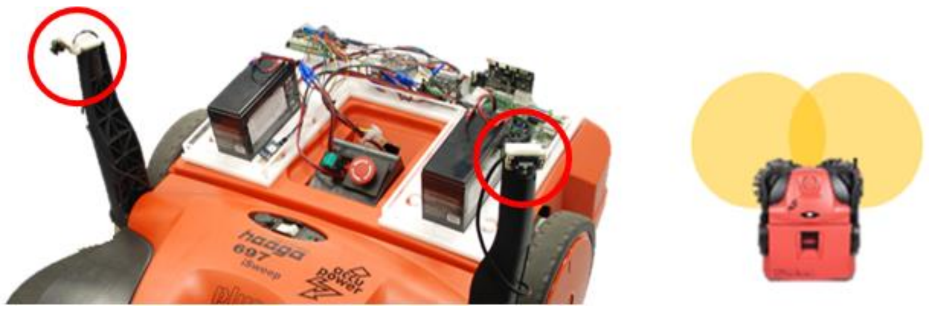


Figure 11. Camera setup and FOV

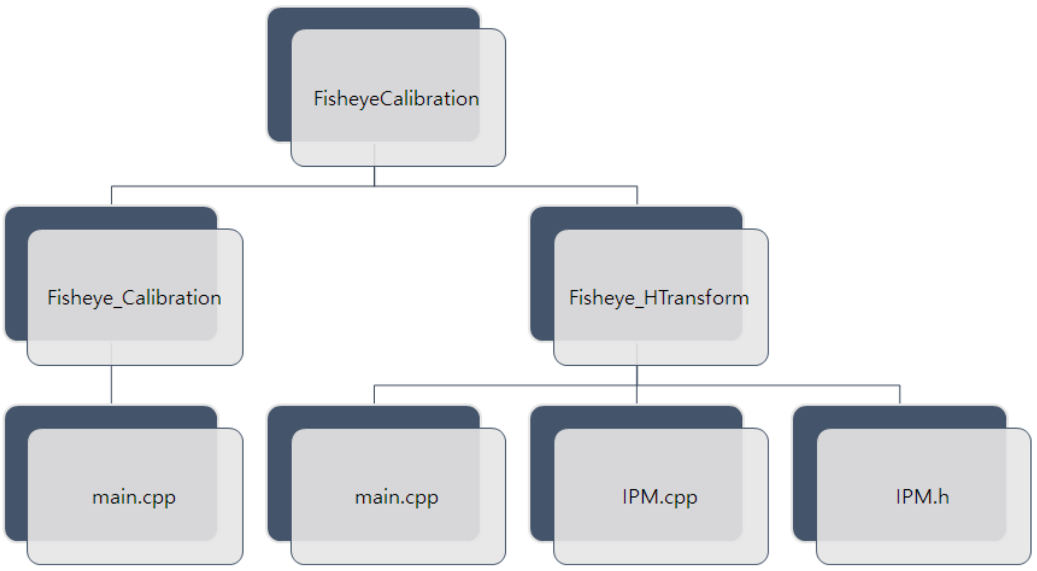


Figure 9. File Structure of FisheyeCalibration Project

5.2.1 Fisheye_Calibration

This application will

- Determine the distortion matrix
- Determine the camera matrix
- Take input from Image file list or Camera
- Read configuration from XML file
- Save the results into XML file

Usage:

Fisheye_Calibration [settings.xml] [imagelists.xml]

Output:

Out_camera_data.xml

This program has a single argument: the name of its configuration file. If none is given, then it will try to open the one named “default.xml”. We provide the configuration file (setting.xml) in XML format. In the configuration file you may choose to use camera as an input or an image list.

Let there be this input chessboard pattern which has a size 9x6. From the fisheye camera, images are saved into the Images folder and imagelists.xml file is created that describes which images to use:

Imagelists.xml

```
<?xml version="1.0"?>
<opencv_storage>
<images>
Images/images1.png
Images/images2.png
Images/images3.png
Images/images4.png
```

```
Images/images5.png
</images>
</opencv_storage>
```

Here's a chessboard pattern found during the runtime of the application:

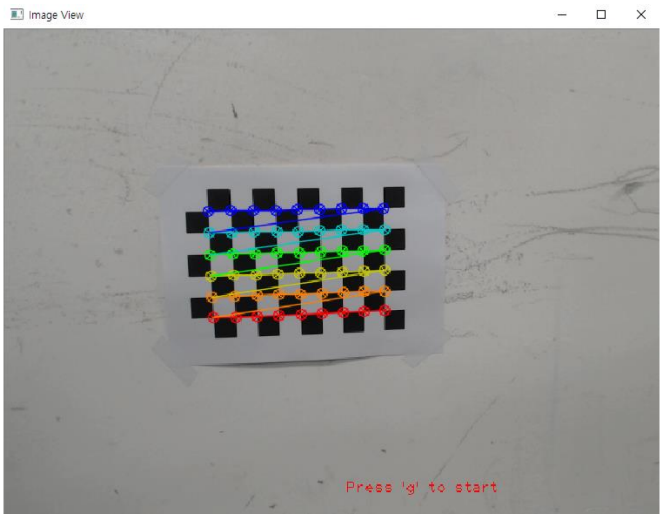


Figure 10. Detected pattern

After running of this program, you'll find the camera and distortion coefficients matrices from the out_camera_data.xml.

5.2.2 Fisheye_HTransform

This application will

- Determine the Homography matrix to transform
- Read camera matrix of Right and Left fisheye camera
- Save the result into XML file

Usage:

```
Fisheye_HTransform  [right_camera_data.xml]  [left_camera_data.xml]  [image_r.bmp]
```

[image_l.bmp] [point_lists.xml]

Output:

H_matrix.xml

The argument files ‘right_camera_data.xml’ and ‘left_camera_data.xml’ are the results acquired from Fisheye_Calibration.xml. The point_lists.xml file is containing the coordinates of image pixel of the fisheye camera and around view image. You can see the XML format in the following table and order of the points is clockwise as shown in the Fig 13. The ‘pts_obj’ is pixel coordinates of undistortion image and ‘pts_dst’ is pixel coorThe type of the points is float.

Point_lists.xml

```
<?xml version="1.0"?>
  <opencv_storage>
  <R_pts_obj>
    point1.x point1.y point2.x point2.y point3.x point3.y point4.x point4.y</R_pts_obj>
  <R_pts_dst>
    point1.x point1.y point2.x point2.y point3.x point3.y point4.x point4.y</R_pts_dst>
  <L_pts_obj>
    point1.x point1.y point2.x point2.y point3.x point3.y point4.x point4.y</L_pts_obj>
  <L_pts_dst>
    point1.x point1.y point2.x point2.y point3.x point3.y point4.x point4.y</L_pts_dst>
</opencv_storage>
```

Using ‘left_camera.xml’ and ‘right_camera.xml’ which has intrinsic and extrinsic parameters, this application can remove the distortion effect. As a result, you’ll find the homography matrix from the saved XML file.

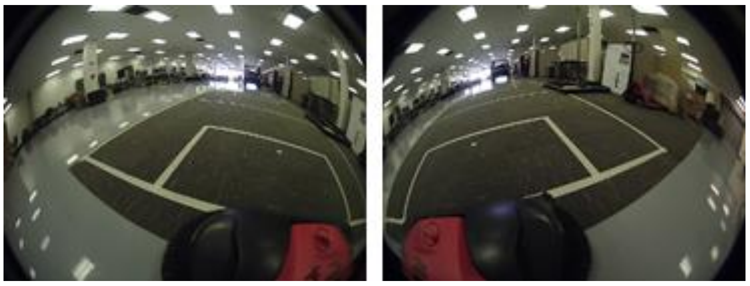


Figure 11. Distortion Image of Fisheye Camera

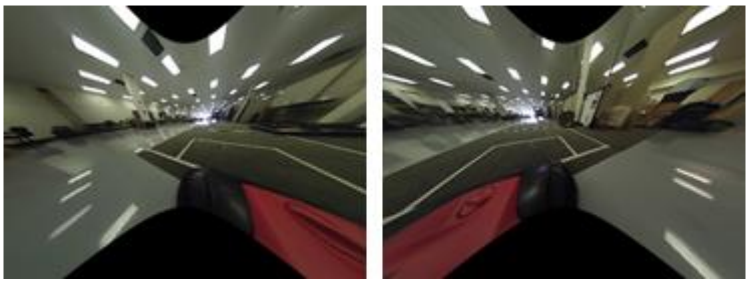


Figure 12. Undistortion Image of Fisheye Camera

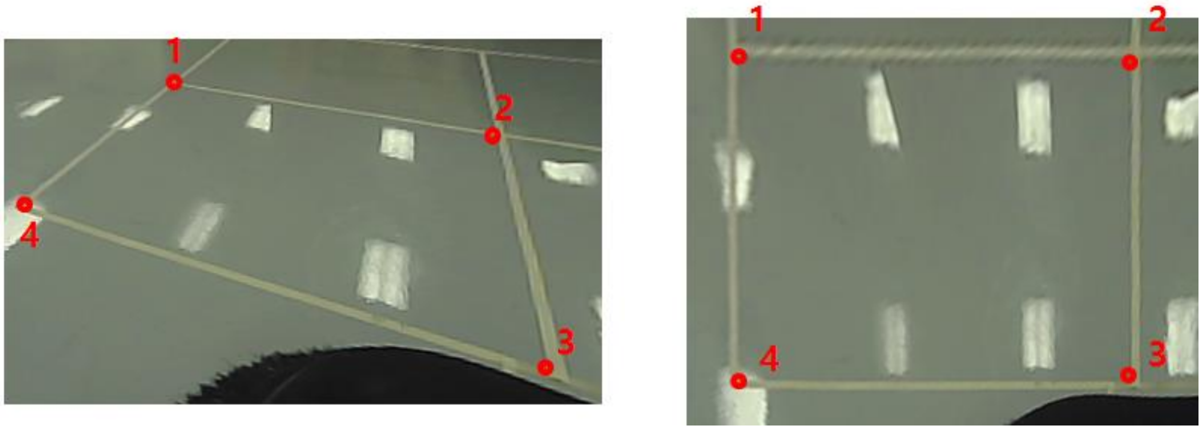


Figure 13. Left: Undistortion Image, Right: Transformed Image by Homography matrix

6. Tutorials

6.1 AroundViewer

From the previous chapter, we can get the camera matrix and homography transformation matrix to build around-view image. This example is the code that build the around-view image using two fisheye cameras which are attached to the Haaga robot.

Code Explanation

```
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/core/core.hpp>
```

1. To compile this example, you need to install OpenCV Library 3.0 or higher and include the headers.

```
const string camera_light_file ="calibration/right_camera.xml";
const string camera_left_file = "calibration/left_camera.xml";
const string topview_file = "calibration/H.xml";

FileStorage fs_R(camera_light_file, FileStorage::READ); // Read the settings
FileStorage fs_L(camera_light_file, FileStorage::READ);
FileStorage fs_H(topview_file, FileStorage::READ);

readData_R(fs_R);
readData_L(fs_L);
readData_H(fs_H);
```

2. Above three XML files are results of the ‘Fisheye_Calibration’ application. From the saved files, camera matrix and homography matrix are loaded.

```
cv::VideoCapture cap_R(argv[1]);
cv::VideoCapture cap_L(argv[2]);
if (!cap_R.isOpened() || !cap_L.isOpened())
        return -1;

cap_R.set(CV_CAP_PROP_FOURCC, CV_FOURCC('M', 'J', 'P', 'G'));
cap_R.set(CV_CAP_PROP_FRAME_WIDTH, imageWidth);
cap_R.set(CV_CAP_PROP_FRAME_HEIGHT, imageHeight);
cap_L.set(CV_CAP_PROP_FOURCC, CV_FOURCC('M', 'J', 'P', 'G'));
cap_L.set(CV_CAP_PROP_FRAME_WIDTH, imageWidth);
cap_L.set(CV_CAP_PROP_FRAME_HEIGHT, imageHeight);
```

3. cv::VideoCapture is the class for video capturing from video files, image sequences or cameras. The class provides C++ API for capturing video from cameras or for reading video files and image sequences. First argument is camera port of right camera and second argument is left.

VideoCapture.set is the function that set a property

- **CV_CAP_PROP_FRAME_WIDTH** Width of the frames in the video stream.
- **CV_CAP_PROP_FRAME_HEIGHT** Height of the frames in the video stream.
- **CV_CAP_PROP_FOURCC** 4-character code of codec.

If you want to increase quality of video steam, you can modify size of resolution and code of codec to ‘I’, ‘Y’, ‘U’, ‘V’.

```
Mat newCamMat_R, newCamMat_L;
float  dummy_query_data_R[10]  =  {  156.00713303635887,  0,  139.5047668805756,  0,
155.83043278608287, 305.505831367789, 0, 0, 1 };
float  dummy_query_data_L[10]  =  {  156.00713303635887,  0,  439.5047668805756,  0,
155.83043278608287, 305.505831367789, 0, 0, 1 };

newCamMat_R = cv::Mat(3, 3, CV_32F, dummy_query_data_R);
newCamMat_L = cv::Mat(3, 3, CV_32F, dummy_query_data_L);

fisheye::undistortImage(frame_R,   undistorted_frame_R,   cameraMatrix_R,   distCoeffs_R,
newCamMat_R, cv::Size(frame_R.cols*scale, frame_R.rows*scale));
fisheye::undistortImage(frame_L,   undistorted_frame_L,   cameraMatrix_L,   distCoeffs_L,
newCamMat_L, cv::Size(frame_L.cols * scale, frame_L.rows * scale));
```

4. The 'dummy_query_data' is camera matrix to build new camera matrix for undistortion and elements are as follows.

$$\text{Camera Matrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Where $\mathrm{f_x}$ and $\mathrm{f_y}$ are the focal lengths and $\mathrm{c_x}$ and $\mathrm{c_y}$ are the principal point that is usually at the image center. The function cv::fisheye::undistortImage is that transform an image to compensate for fisheye lens distortion.

void cv::fisheye::undistortImage(InputArray **distorted**, OutputArray **undistorted**, InputArray **K**, InputArray **D**, InputArray **Knew** = cv::noArray(), const Size &**new_size** = Size())

- **distorted:** image with fisheye lens distortion.
- **undistorted:** Output image with compensated fisheye lens distortion.
- **K:** Camera matrix
- **D:** Input vector of distortion coefficients (k1, k2, k3, k4).
- **Knew:** Camera matrix of the distorted image. By default, it is the identity matrix but you may additionally scale and shift the result by using a different matrix.
- **new_size:** The function transforms an image to compensate radial and tangential lens distortion.

```
cv::warpPerspective(undistorted_frame_R, topview_R, right_H, cv::Size(frame_R.cols, frame_R.rows));
cv::warpPerspective(undistorted_frame_L, topview_L, left_H, cv::Size(frame_L.cols, frame_L.rows));

roi_R = topview_R(cv::Rect(topview_R.cols / 2, 0, topview_R.cols / 2, topview_R.rows));
roi_L = topview_L(cv::Rect(0, 0, topview_R.cols / 2, topview_R.rows));
roi_R.copyTo(topview(cv::Rect(topview.cols / 2, 0, topview.cols / 2, topview.rows)));
roi_L.copyTo(topview(cv::Rect(0, 0, topview_R.cols / 2, topview_R.rows)));

cv::imshow("topview", topview);
```

5. cv::wawrpPerspective is the function that applies a perspective transformation to an image.

void warpPerspective(InputArray **src**, OutputArray **dst**, InputArray **M**, Size **dsize**, int **flags**=INTER_LINEAR, int **borderMode**=BORDER_CONSTANT, const Scalar& **borderValue**=Scalar())

- **src**: input image.
- **dst**: output image that has the size dsize and the same type as src .
- **M**: 3x3 transformation matrix.
- **dsize**: size of the output image.
- **flags**: combination of interpolation methods (INTER_LINEAR or INTER_NEAREST) and the optional flag WARP_INVERSE_MAP, that sets M as the inverse transformation
- **borderMode**: pixel extrapolation method (BORDER_CONSTANT or BORDER_REPLICATE).
- **borderValue**: value used in case of a constant border; by default, it equals 0.

The undistortion images of the right and left image are need to be transformed. Then get the region of interest of each image and copy to the null image. The result of around-view image is shown as Fig. 15.

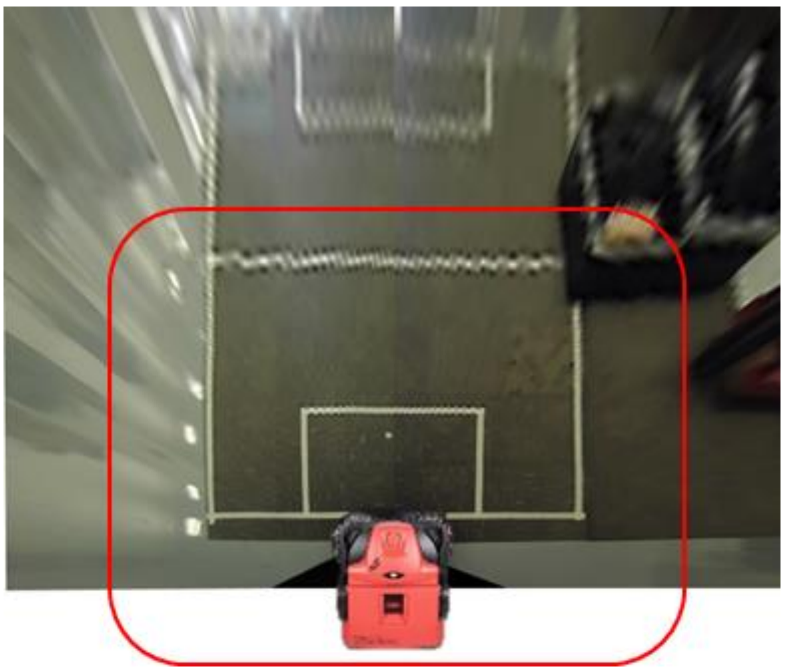


Figure 15. Around view Image

6.2 LineFollower

This example helps Haaga robot to set the appropriate direction by extract the curb in the road environment. The input is around-view image acquired from ‘AroundViewer’.

Code Explanation

```
cv::Mat img = imread(argv[1]);
cv::Mat img_gray, img_blur, img_canny;
```

1. Load the source (around-view) image and create a matrix of same size.

2. Convert the image to grayscale (using the function cvtColor).

```
void setwindowSettings() {
        cv::namedWindow("canny_var", 100);
        cv::createTrackbar("threshold1", "canny_var", &threshold1, 500, NULL);
        cv::createTrackbar("threshold2", "canny_var", &threshold2, 500, NULL);
        cv::createTrackbar("blur", "canny_var", &gaussblur, 20, NULL);
}
```

3. Create a trackbar window for the user to enter the threshold for Canny detector and Gaussian blur.

```
cv::cvtColor(img, img_gray, CV_BGR2GRAY);
cv::GaussianBlur(img_gray, img_blur, cv::Size(gaussblur*2-1, gaussblur*2-1), 3);
cv::Canny(img_blur, img_canny, threshold1, threshold2, 3);
```

4. Blur the gray image with a filter kernel (size must be odd) and extract edge using cv::Canny function.

void cvtColor(InputArray **src**, OutputArray **dst**, int **code**, int **dstCn**=0)

Converts an image from one color space to another

- **src** : input image: 8-bit unsigned, 16-bit unsigned (CV_16UC...), or single-precision floating-point.
- **dst** : output image of the same size and depth as src.
- **code**: color space conversion code
- **dstCn** : number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from src and code

void GaussianBlur(InputArray **src**, OutputArray **dst**, Size **ksize**, double **sigmaX**, double **sigmaY**=0, int **borderType**=BORDER_DEFAULT)

Blurs an image using a Gaussian filter

- **src** : input image; the image can have any number of channels, which are processed independently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.
- **dst** : output image of the same size and type as src.
- **ksize** : Gaussian kernel size. ksize.width and ksize.height can differ but they both must be positive and odd.
- **sigmaX** : Gaussian kernel standard deviation in X direction.
- **sigmaY** : Gaussian kernel standard deviation in Y direction
- **borderType** : pixel extrapolation method

void Canny(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3, bool L2gradient=**false**)

Finds edges in an image using the algorithm.

- **Image**: single-channel 8-bit input image.
- **edges**: output edge map; it has the same size and type as image.
- **threshold1**: first threshold for the hysteresis procedure.
- **threshold2**: second threshold for the hysteresis procedure.

- **apertureSize**: aperture size.
- **L2gradient**: a flag, indicating whether a more accurate.

```
vector<Vec4i> lines;
cv::HoughLinesP(img_canny, lines, 1, CV_PI / 180, 50, 20, 10);
```

5. Apply the transform with the arguments:

void HoughLinesP(InputArray **image**, OutputArray **lines**, double **rho**, double **theta**, int **threshold**, double **minLineLength**=0, double **maxLineGap**=0)

- **dst**: Output of the edge detector. It should be a grayscale image.
- **lines**: A vector that will store the parameters (x_1, y_1, x_2, y_2) of the detected lines
- **rho** : The resolution of the parameter r in pixels. We use 1 pixel.
- **theta**: The resolution of the parameter \theta in radians. We use 1 degree (CV_PI/180)
- **threshold**: The minimum number of intersections to "detect" a line
- **minLinLength**: The minimum number of points that can form a line. Lines with less than this number of points are disregarded.
- **maxLineGap**: The maximum gap between two points to be considered in the same line.

To detect the line, we use the 'HoughLinesP' function that more efficient implementation of the Hough Line Transform. It gives as output the detected lines.

```
std::vector<int> candidates;
for (size_t i = 0; i < lines.size(); i++)
{
   if ((lines[i][2] > 430 && lines[i][0] > 430) && (lines[i][1] > img.rows / 2 && lines[i][3] >
   img.rows / 2))
   candidates.push_back(i);
}
```

6. Then set the limit condition to remove outlier lines.

```
int minidx = 0;
if (candidates.size() > 1)
{
   for (size_t i = 1; i < candidates.size(); i++)
   {
      if (lines[candidates[minidx]][0]>lines[candidates[i]][0])
      minidx = i;
   }
}
else if (candidates.size() == 1)
   minidx = 0;
```

7. Using the candidate lines, sort the candidates in the order closer to the Haaga and save the index.

```
if (candidates.size()>0)
{
   Vec4i l = lines[candidates[minidx]];
   float dx, dy;
   float length, deg, theta;
   dy = l[2] - l[0];
   dx = l[1] - l[3];
   theta = atan2f(dy, dx);
   if (theta * 180 / CV_PI > 90)
      deg = -1 * (180 - theta * 180 / CV_PI);
   else
      deg = theta * 180 / CV_PI;
   line(result, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 0, 255), 2, CV_AA);
}
```

8. Calculate direction of the line using atan2() function and convert to deg from radian.

```
String headingText="Heading=";
stringstream ss;
ss << deg;
headingText += ss.str();
cv::putText(result, headingText, cv::Point(lines[candidates[minidx]][0],
                lines[candidates[minidx]][1]), FONT_HERSHEY_PLAIN, 1.1,
                cv::Scalar(0, 0, 255));
imshow("source", img);
imshow("detected lines", result);
```

9. Then display our result to image window.

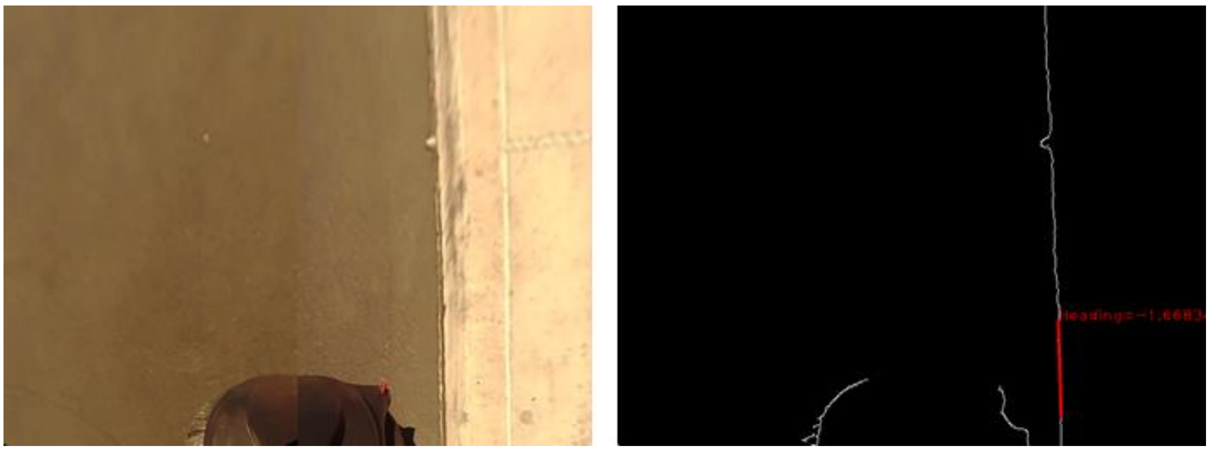


Figure 14. LineFollower results (Right) Input Around-view image

(Left) Detected curb edge and direction

7. ROS Programming

The Robot Operative System (ROS) is an open-source, meta-operating system for your robot maintained by the Open Source Robotics Foundation (OSRF). It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. A ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model.

The ROS Master allows all other ROS pieces of software (Nodes) to find and talk to each other. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.

7.1 ROSSerial

The 'rosserial' is a protocol for wrapping standard ROS serialized messages and multiplexing multiple topics and services over a character device such as a serial port or network socket.

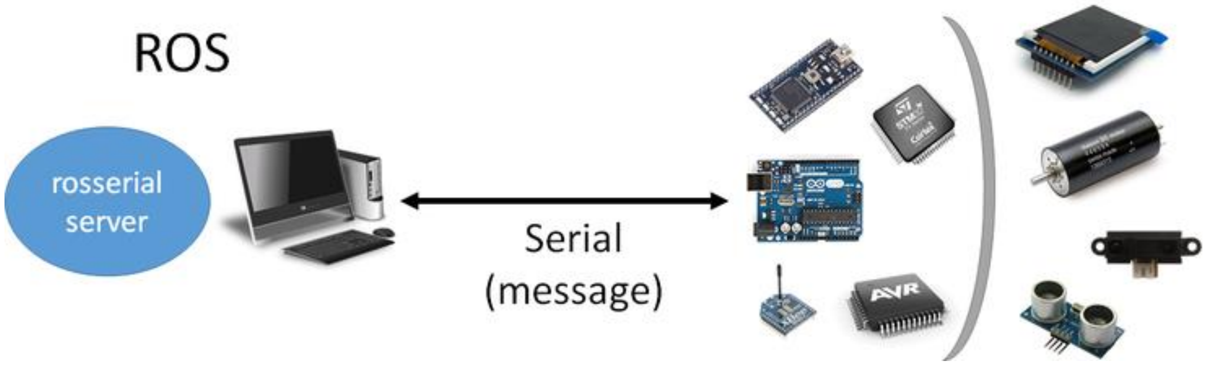


Figure 15. Structure of the rosserial

Installation

```
$ sudo apt-get install ros-indigo-rosserial ros-indigo-rosserial-server ros-indigo-rosserial-arduino
```

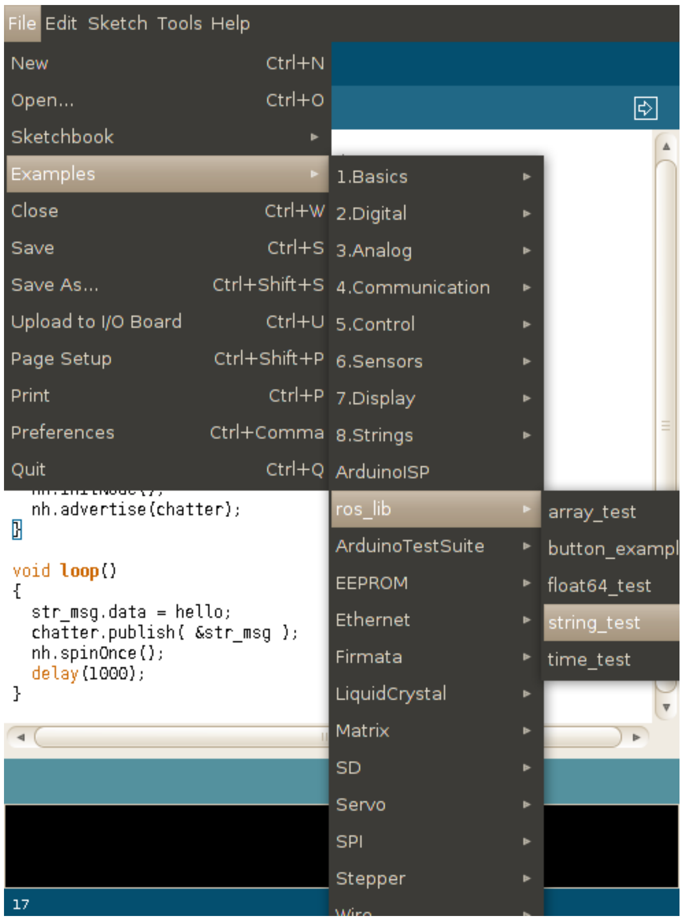
You can install rosserial for Arduino.

```
$cd /catkin_ws/src
$git clone https://github.com/ros-drivers/rosserial.git
$cd /catkin_ws/
$catkin_make
$catkin_make install
$source /catkin_ws/install/setup.bash
```

These commands clone rosserial from the github repository, generate the rosserial_msgs needed for communication, and make the ros_lib library.

```
$cd /Arduino/libraries
$rm -rf ros_lib
$rosrun rosserial_arduino make_libraries.py .
```

Install the ros_lib into the Arduino Environment.



```
  nh.advertise(chatter);
}

void loop()
{
  str_msg.data = hello;
  chatter.publish( &str_msg );
  nh.spinOnce();
  delay(1000);
}
```

Figure 16. ros_lib library in the Arduino IDE

7.2 Commnunication between Bluno and Jetson

Let’s try to communicate with Bluno (Haaga firmware installed) in the Jetson tk1 board.

```
cd /catkin_ws/src
catkin_create_pkg ros_arduino rospy
catkin_make
cd /catkin_ws/src/ros_arduino
mkdir src
gedit serial_node.py
```

Create ros_arduino pakage and build in the catkin workspace and write the below code to the serial_node.py file.

Code

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

import rospy
from rosserial_python import SerialClient, RosSerialServer
import multiprocessing

import sys

if __name__=="__main__":

    rospy.init_node("arduino_node")
    rospy.loginfo("ROS Serial Python Node")

    port_name = rospy.get_param('~port','/dev/ttyUSB0')
    baud = int(rospy.get_param('~baud','57600'))

    sys.argv = rospy.myargv(argv=sys.argv)
    if len(sys.argv) > 1 :
        port_name   = sys.argv[1]
        baud = int(sys.argv[2])

    rospy.loginfo("Connecting to %s at %d baud" % (port_name,baud) )
    client = SerialClient(port_name, baud)
    try:
        client.run()
    except KeyboardInterrupt:
        pass
```

```
port_name = rospy.get_param('~port','/dev/ttyUSB0')
baud = int(rospy.get_param('~baud','57600'))
```

Set the default port is /dev/ttyUSB0 and baudrate is 57600.

```
sys.argv = rospy.myargv(argv=sys.argv)
     if len(sys.argv) > 1 :
          port_name    = sys.argv[1]
          tcp_portnum = int(sys.argv[2])
    rospy.loginfo("Connecting to %s at %d baud" % (port_name,baud) )
     client = SerialClient(port_name, baud)
```

Load the arguments and modify the port_name and baud variables.

```
try:
     client.run()
except KeyboardInterrupt:
     pass
```

Try to connect via serial port until interrupt occurs.

Usage:

```
$roscore
```

Open the terminal and start ROS master

```
$cd /catkin_ws/src/ros_arduino/src
$python serial_node.py /dev/ttyACM0 57600
```

Open the new terminal and run the ROS node with your serial port argument.

```
$rostopic list
$rosservice list
```

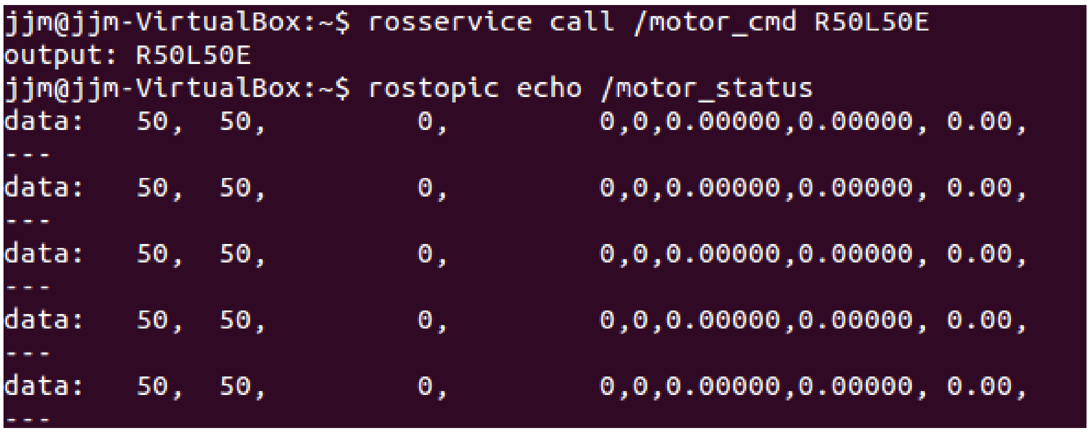
You can find the ros message lists (/motor_status topic and /motor_cmd service).

```
$rostopic echo /motor_status
```

From the /motor_status topic, you get the Haaga robot status.

```
$rosservice call /motor_cmd R50L50E
```

Through the /motor_cmd service, you can send the command to rotate the motor.



```
jjm@jjm-VirtualBox:~$ rosservice call /motor_cmd R50L50E
output: R50L50E
jjm@jjm-VirtualBox:~$ rostopic echo /motor_status
data:   50,  50,        0,        0,0,0.00000,0.00000, 0.00,
---
data:   50,  50,        0,        0,0,0.00000,0.00000, 0.00,
---
data:   50,  50,        0,        0,0,0.00000,0.00000, 0.00,
---
data:   50,  50,        0,        0,0,0.00000,0.00000, 0.00,
---
data:   50,  50,        0,        0,0,0.00000,0.00000, 0.00,
---
```

Figure 17. Communication with Bluno