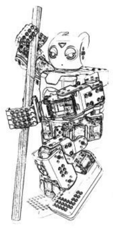
Using ARDUINO with ROBOTIS Systems

By Chi N. Thai



CNT Robotics LLC, Buford

2022

Chapter 2: Using OpenRB-150

ROBOTIS released the **OpenRB-150** in Fall 2022, and it is the first member of a new series of ROBOTIS Arduino Variant Boards called **OpenRB**. It comes in the Arduino MKR format, essentially combining a MKR ZERO and a DXL MKR SHIELD (see Fig. 2.1). The OPENRB-150 comes with Serial2 already set up with the usual UART 4-pin port (near the DXL-VIN jumper), but the user will have to do some soldering work to put a 1x4 Pin Header Socket (Pitch 2 mm) before it is usable. Then a BT-210/410 module can be used there as Serial2. For more information, please visit this web link https://emanual.robotis.com/docs/en/parts/controller/OPENRB-150/.

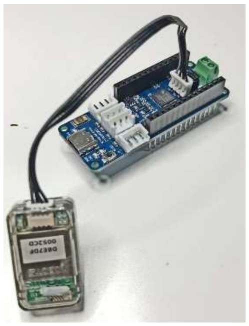


Fig. 2.1 OpenRB-150 Board with BT-210.

2.1 OpenRB-150 Features

The OpenRB-150 uses a USB C connector for programming and provides 4 DXL Ports (3-Pin TTL - JST type). The user has the option of powering this board via the USB C connector at 5V (usable with XL-330s), but the author recommends using the terminal block VIN to power only the Dynamixels and keeping the USB C Port just for programming and powering various peripherals connected to its 2 headers. Details for several power options for this board are available at this link (https://emanual.robotis.com/docs/en/parts/controller/OpenRB-150/#connecting-power).

Using a special ROBOTIS Arduino sketch, this board can function as a U2D2 module to help the user manage various Dynamixels with the Dynamixel Wizard 2 tool, but **up to a baud rate of 1 Mbps** only. More information is available for the U2D2 module and the Dynamixel Wizard tool below:

- https://emanual.robotis.com/docs/en/parts/controller/OpenRB-150/#dynamixel-wizard-20
- https://emanual.robotis.com/docs/en/parts/interface/u2d2/#introduction.
- https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_wizard2/#introduction.

The OpenRB-150's pinout is identical to the one used for the MKR ZERO (https://emanual.robotis.com/docs/en/parts/controller/openrb-150/#pinout). This board offers several types of peripheral ports accessible via its headers:

- One **I2C** Port at Pins D11 (SDA) and D12 (SCL).
- One **SPI** Port at Pins D8 (COPI/MOSI), D9 (SCK), D10 (CIPO/MISO).
- ROBOTIS already set up 3 UART Ports for the user:
    - **Serial1** is internally used for bi-directional communications with the Dynamixels at Half-Duplex.
    - **Serial2** (Full-Duplex) is connected to the 4-Pin Port (TX, RX, VDD, GND) located near the VIN(DXL)-5V Jumper Block. Serial2 is designed to be

connected straight to Bluetooth modules such as BT-210/410 or LN-101 (wired connection).

- **Serial3** (Full Duplex) is routed out to Pins D13 (RX) and D14 (TX).

As the OpenRB-150 uses a SAMD21 Cortex-M0+ MCU, it has 3 more Serial Ports that the user can access to, via GPIO programming (https://www.arduino.cc/en/Tutorial/SamdSercom). Section 2.4 will show the reader how to access **Serial4** and **Serial5** and how to program them to be compatible with the RC-100 Communication Protocol (https://emanual.robotis.com/docs/en/parts/communication/rc-100/#communication-packet).

2.2 Robot Demonstration Platforms

For Velocity Control (Wheel Mode) applications, the author used a simple CarBot frame made out of parts from the DREAM robotic system, with XL-320s or XL-330s used as drive actuators (see Fig. 2.2). Various peripherals could then be deployed on this CarBot frame wherever they were needed.

Readers probably already notice that the author "really" likes to use 3M Dual-Lock tape (a habit picked up from his days of helping out with FIRST Robotics Competitions). They are available everywhere - https://www.amazon.com/Dual-Reclosable-Fastener-SJ3560-Clear/dp/B07QNMBB28.

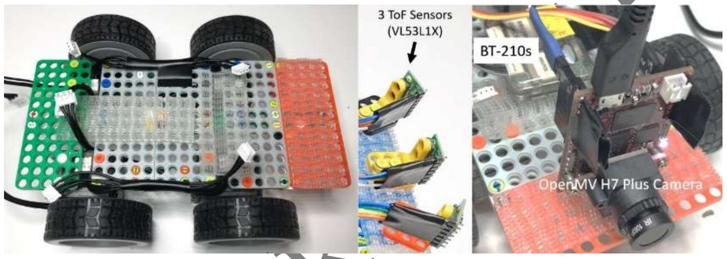


Fig. 2.2 CarBot Platform used for Velocity Control Applications.

For Position Control (Servo Mode) applications, the author used a Mobile Manipulator based on AX-12As (Phantom Arm) and AX-12Ws for the Rover component (see Fig. 2.3).

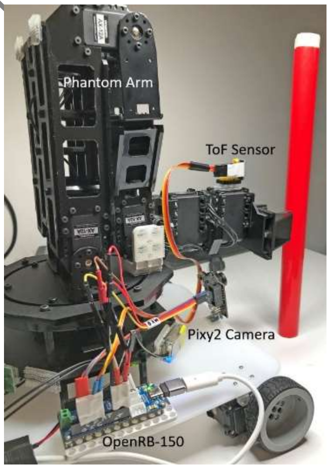


Fig. 2.3 Mobile Manipulator Platform used for Position Control Applications.

For Motion Programming applications, the author used the Articulated 4-Wheel Platform - Hybrid (A4WP-H) – see Fig. 2.4. The A4WP-H robot is actuated by XL-430s set in Timed Position Control Mode (https://emanual.robotis.com/docs/en/dxl/x/2xl430-w250/#drive-mode10). It is based on the Quadruped robot used in the ROBOTIS Engineer Kit 1 (Thai, 2021).

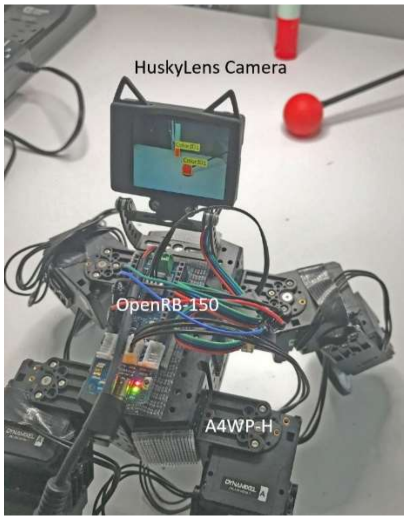


Fig. 2.4 Articulated 4-Wheel Platform - Hybrid used for Motion Programming Applications.

Thus, representative actuators from both Protocol 1 and Protocol 2 groups are used/demonstrated in this chapter so as to serve the most users possible.

2.3 Software Tools

As of Fall 2022, the author had problems with using the Arduino IDE V.2, thus he had stayed with Arduino IDE V.1.8.19 for this book.

Arduino has numerous support materials for installing Arduino hardware and software:

- For downloading and installing either Arduino IDE versions, please visit this link https://www.arduino.cc/en/software.
- Use this link for more support information https://support.arduino.cc/hc/en-us/categories/360002212660-Software-and-Downloads.
- To add an Arduino board to your Arduino IDE, please use this link https://support.arduino.cc/hc/en-us/articles/360016119519-Add-a-board-to-Arduino-IDE.
- To add a third party Arduino Variant board, please visit this link https://support.arduino.cc/hc/en-us/articles/360016466340-Add-or-remove-third-party-boards-in-Boards-Manager.

ROBOTIS has their own web information for installing Dynamixel Wizard 2 and Arduino IDE on Linux, Mac and Windows platforms at this link (https://emanual.robotis.com/docs/en/parts/controller/OpenRB-150/#development-environment).

To install the OpenRB-150 into the Arduino Board Manager, please use this link https://emanual.robotis.com/docs/en/parts/controller/OpenRB-150/#install-board-manager. The procedure listed at this link works well if the OpenRB-150 is your first ROBOTIS board that the reader installs. But if the reader happens to have other 3rd party boards already installed, then he/she needs to click on the button to the right of the **Additional Boards Manager URLs** text field. This will pop up a small window, the reader can then add the OpenRB-150 JSON link *https://raw.githubusercontent.com/ROBOTIS-GIT/OpenRB-150/master/package_openrb_index.json* into its own **separate line** in this pop-up list. Finally, click OK on everything and restart the Arduino IDE. The reader should be then able to see the OpenRB-150 as one of the options in the Arduino Board Manager.

Next, the reader needs to install the library **Dynamixel2Arduino** as per this web link https://emanual.robotis.com/docs/en/parts/controller/OpenRB-150/#library-api.

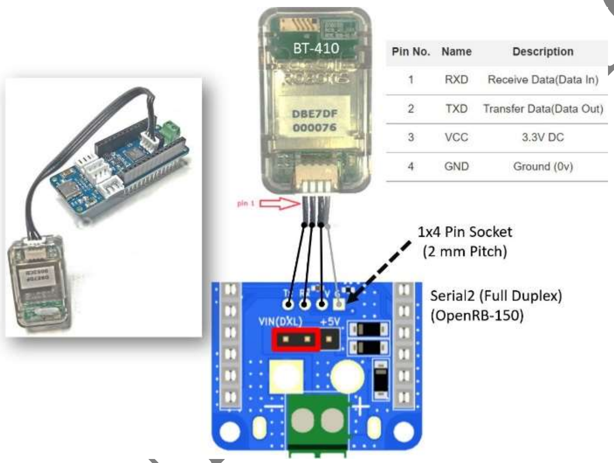
2.4 UART Communications

2.4.1 Wanting to use BT-210/410s with UART Ports

Let's say that the reader wants to use the BT-210/410 modules with the OpenRB-150 and via its various Serial UART Ports: **Serial2, Serial3, Serial4** and **Serial5**.

2.4.2 Hardware Hook Up for Serial2

You will need to solder a 1x4 Pin Header (male to male, 2 mm pitch) (https://www.amazon.com/Uxcell-a15062900ux0315-Single-Straight-Connector/dp/B012TBLZTC) to the UART holes located near the VIN(DXL) jumper (https://emanual.robotis.com/docs/en/parts/controller/OpenRB-150/#terminal-vin). There is not much room in this area, so it does require some soldering skills. But once this work is done, you can just plug a BT-210/410 straight into this 1x4 header, but make sure to match the "gray-wire" to the GND pin on this header (see Fig. 2.9). Basically, the RXD line from the BT module needs to connect to the TX pin on the OpenRB-150 and the TXD line from the BT module needs to connect to the RX pin on the OpenRB-150.

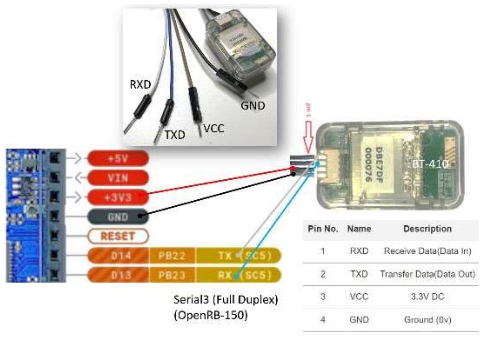


Pin No.	Name	Description
1	RXD	Receive Data(Data In)
2	TXD	Transfer Data(Data Out)
3	VCC	3.3V DC
4	GND	Ground (0v)

Fig. 2.9 Connecting a BT-210/410 to Serial2 Port on the OpenRB-150.

2.4.3 Hardware Hook Up for Serial3

If your wire splicing skills are better than your soldering skills, you can consider using Serial3 out of Pins D13 and D14 instead. You will need to "sacrifice" the short ribbon cable that came with the BT-210/410 module and cut out the end that has the **larger JST connector**. Next, you'll need to splice in 4 Dupont Jumper Cables (https://www.amazon.com/ZYAMY-120PCS-Connector-Multicolor-Breadboard/dp/B0742RS6YL) to these open ends of the ribbon cable (see Fig. 2.10).



Pin No.	Name	Description
1	RXD	Receive Data(Data In)
2	TXD	Transfer Data(Data Out)
3	VCC	3.3V DC
4	GND	Ground (0v)

Fig. 2.10 Connecting a BT-210/410 to Serial3 Port on the OpenRB-150.

2.4.4 Software Setup for Serial2 and Serial3

As shown in Fig. 2.6, **software wise**, the OpenRB-150 Core already took care of setting up Serial2 and Serial3 "behind-the-scenes" for the programmer. So at this point, all basic Arduino Functions related to the **Serial Device** are ready to be used for Serial2 and Serial3 as normal, for example: **begin(), println(),** … , just like for the **Serial Monitor** tool of the Arduino IDE (https://www.arduino.cc/reference/en/language/functions/communication/serial/).

2.4.5 RC-100 Data Flow via Serial2 and Serial3

In the bottom right of Fig. 2.11, the reader can see the **Virtual Remote Controller** panel which functions identically to the physical RC-100B Controller, whereas the operator can push on the Buttons U/D/L/R/1/2/3/4/5/6 singly or in combination, and a proper Remocon Packet will be sent to the OpenRB-150 via the BT-210 to Serial2/Serial3. The structure of this Remocon Packet is described at this link (https://emanual.robotis.com/docs/en/parts/communication/rc-100/#communication-packet).

2.4.6 RC-100 Data Flow via Serial4 and Serial5

As previously shown in Figs. 2.6 and 2.7, ROBOTIS had only setup **Serial2** and **Serial3** to be used as possible **RC100** Ports. Thus, to set up **Serial4** and **Serial5** as **RC100** Ports would require programmers to build **Serial4** and **Serial5** "from the ground up". Essentially, we will be using the Arduino **Serial** Library along with Local Functions mimicking the functionalities of an **RC100** Port (as if we were using the **RC100** Library).

Fig. 2.21 shows how BT-210/410s are hooked up to **Serial2, Serial4** and **Serial5** on the OpenRB-150. The reader can use the previous setup with two instances of TASK 2 to check out the runtime performances of the Sketch **RC100_DataFlow_Serials_4_5.ino**.

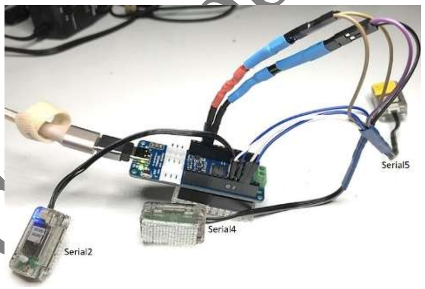


Fig. 2.21 OpenRB-150 with BT-210/410s connected to **Serial2, Serial4** and **Serial5**.

The user should verify that the run-time behaviors of Serial4 and Serial5 are the same as the ones listed for Serial2 and Serial3 at the end of Sub-Section 2.4.5: keyboard processing speed, "all-keys-release" special packet and multiple-keys usage.

2.4.7 Discussions

Thus, overall, the OpenRB-150 has **Serial2**, **Serial3** and **Serial4** for sure available for the programmer to use as UART Ports connecting to any other Arduino Boards, other ROBOTIS CM controllers, SBCs, and even PCs. In these cases, the reader can choose other baud rates than the default 57.6 Kbps rate used by ROBOTIS, especially when BT-210/410 are not involved.

2.5 Actuators in Wheel Mode

2.5.1 Control Tables and Dynamixel2Arduino

Let's now get into the application of the **Dynamixel2Arduino** library to the control of ROBOTIS Actuators. Fig. 2.2 showed the physical CarBot platform used in this Section 2.5, whereas the author could mount either XL-320s or XL-330s as Drive Actuators.

In general, ROBOTIS actuators have some settings in EEPROM and some in RAM. For the EEPROM settings, the TORQUE voltage to the actuators needs to be OFF before the programmer can change them, and after the changes are made, TORQUE needs to be set back ON to retain the changes. The RAM settings can be changed while the TORQUE is ON.

Furthermore, with V. 0.2.1 of the OpenRB-150 Core, the **Dynamixel2Arduino** Methods **ping() and scan()** were not working consistently for the author, for example they had reported **incorrectly** that some Dynamixels were not working properly (when they were working perfectly fine). Thus, in such situations, the author recommends using the Sketch **usb_2_dynamixel.ino** and **Dynamixel Wizard 2.0** to check that all Dynamixels used are properly setup, baud rate wise and operating mode wise, in the way wanted by the user before using the example codes provided with this book. Hopefully, by the time this book gets published in early 2023, all these issues are resolved for all users.

2.5.2 Setting Up and Using Wheel Mode

This Sketch **4W_Car_XL-330_Check.ino** demonstrates how to set 4 XL-330s into Velocity Control (Wheel) Mode. It also illustrates the "relationship" between **Goal Velocity** (Addr. 104) and **Velocity Limit** (Addr. 44).

2.5.3 Discussions

In Fig. 2.26, Lines 75-78 are used to turn on the 4 servos in this order ID=1, 2, 3 then 4. In Fig. 2.27, the FOR Loop used iterates twice: on the first iteration, Servos 1 and 2 are enabled (i.e., back wheels together), while on the 2nd iteration, Servos 3 and 4 are enabled (i.e., front wheels together). So, the reader may wonder about which approach is "better"?

2.6 Carbot's Remote Control (RC100 vs. BLE)

We are now ready to formulate a complete Remote-Control (RC) solution for the CarBot (see Fig. 2.28). This Section 2.6 demonstrates two RC solutions: one using the ROBOTIS RC100 protocol with BT-210/410, and the other using a BLE peripheral such as the Adafruit Bluefruit LE SPI Friend (https://www.adafruit.com/product/2633).

2.6.1 Data Flow Characteristics for BLE

The installation tutorial for the Adafruit **Bluefruit LE SPI Friend** is available at this link (https://learn.adafruit.com/introducing-the-adafruit-bluefruit-spi-breakout). This BLE peripheral comes with a free Mobile App named "LE Connect" (https://learn.adafruit.com/bluefruit-le-connect) for iOS and Android OSes. This App provides a Control Pad that can serve as a Remote Controller for the CarBot (see Fig. 2.29) – but it does not have pads for Buttons 5 and 6, like for the Virtual RC100 Controller (see Fig. 2.11).

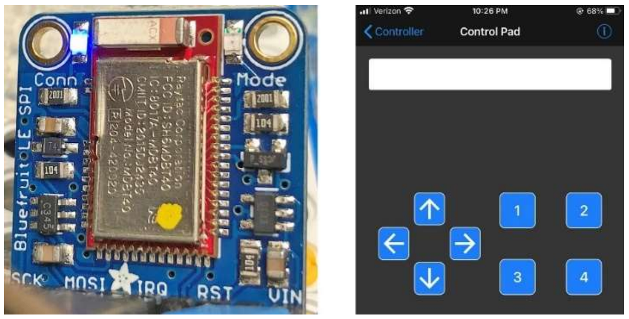


Fig. 2.29 Adafruit **Bluefruit LE SPI Friend** and Mobile App **LE Connect**.

BLE stands for "Bluetooth Low Energy", thus the way this technology sends and receives data is very different from the RC100 protocol when used with BT-210/410 – even though both use Bluetooth. From Sections 2.4.5 and 2.4.6, the user can appreciate that as long as he/she pushes on the UDLR123456 Buttons, the RC100 Virtual Controller sends out **continuously** corresponding Remocon Packets to the OpenRB-150. This process uses lots of energy, as compared to the BLE's way which sends out the Status of a given Touch Area **ONLY ONCE** – when a specific Touch Area is **first pressed** and when it is **released**.

These results showed the author that **multiple-buttons handling** would have to be **designed differently when using BLE connections vs. RC100 connections** (see Sub-Section 2.4.5).

2.6.2 RC Solution with RC100 Protocol

The solution Sketch is named **RC100_4W_Car_XL-330_S2.ino**. It uses Serial2 as an RC100 Port.

2.6.3 RC Solution with BLE Protocol

As shown previously in Sub-Section 2.6.1, Remote Control with the Adafruit BLE Module carries the issue of not "registering" the "button release" event properly most of the time. This means that the robot very likely would continue executing a Remote Control "command" even though the operator had released the corresponding "button". Thus, the author designed a RESET/PANIC button and assigned it to Button 2 which would, when pressed, stop the robot and reset all control variables to their "safe" values.

The solution Sketch **RC_BLE_4W_Car_XL-330.ino** is illustrated via Fig. 2.41 through Fig. 2.48.

2.6.4 Discussions

For responsive Remote-Control applications needing Multiple Button/Key Control, the RC100 protocol via a UART Port is obviously the one to use.

The author is not certain about the cause of the BLE "release-event/packet-dropping" issue for the Adafruit Bluefruit SPI Module that is used in this book: it could be from the **LE Connect** App or it could come from the BLE protocol itself? Perhaps readers who happen to use other BLE SPI Modules can help clarify this issue?

2.7 Tracker-Follower with ToF Sensors

For Sections 2.7 and 2.8, the author added Time-of-Flight (ToF) sensors such as the VL53L1X which is available from several vendors such as Pololu, Adafruit and SparkFun. The author chose the Pololu VL53L1X (https://www.pololu.com/product/3415) because of its flexible Arduino software library.

Fig. 2.49 shows the overall configuration for these ToF sensors onto an OpenRB-150 platform.

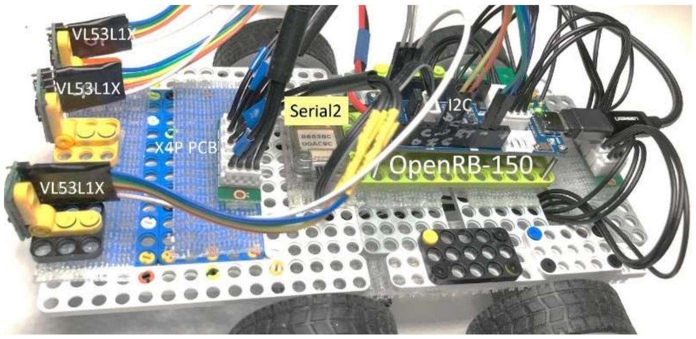


Fig. 2.49 OpenRB-150 Platform with ToF sensors VL53L1X.

2.7.1 Data Flow Characteristics of Multiple VL53L1Xs

The 3 VL53L1Xs use the same I2C Port on the OpenRB-150, and the two main configuration issues were:

1. How to assign a different I2C address to each of the 3 VL53L1Xs used.
2. Evaluation of the two Sensor-Read approaches that are possible with the VL53L1X, one with BLOCKING and the other with NON-BLOCKING properties. Also, to determine how each approach affects UART communications via Serial2 and Serial3.

2.7.2 Tracker-Follower using 3 VL53L1Xs

In this project **4W_Car_XL-330_Tracker_Follower.ino**, the Left and Right VL53L1Xs are programmed to make the CarBot **track** a given object by spinning Left or Right as appropriate, while the Center VL53L1X is programmed **to follow** the object if it gets **too close** to the Center ToF sensor.

2.7.3 Discussions

These ToF sensors have a smaller "view-angle" than the NIR-based distance sensors that the author was used to, and they performed well. There are other models that can detect multiple objects (https://www.pololu.com/product/3416) or provide a 8x8 detection grid, with a much longer measurement time ~1 second per data set (https://www.pololu.com/product/3417).

2.8 Smart Avoider

The goal of this project is to blend Remote-Control and Autonomous Behaviors into the same CarBot, such that the operator can operate the robot via RC100 protocol as before (Sub-Section 2.6.2). But if the robot detects that it is being driven "too close" to an obstacle, it will ignore the operator's commands and will execute autonomously a combination of moves to get it to "safety", before yielding back the Remote-Control feature to the operator (in other words, an "anti-crash" feature).

2.8.1 Using Serial2 with 3 VL53L1Xs

In the Sketch **3_VL53L1X_Serial2_B_NB.ino**, the reader is provided with both Blocking and Non-Blocking Read procedures for the reader to try out.

The procedures for setting up and using the ToF sensors and Serial2 with the RC100 protocol had been described in details previously, thus Fig. 2.58 goes straight to Function **loop()**:

- The reader can choose whether to use a Non-Blocking (Line 75) or Blocking (Line 76) procedure for reading in data from the 3 VL53L1Xs.

2.8.2 Smart Avoider using 3 VL53L1Xs & Serial2

The Overall algorithm used for this project can be described with the following pseudo-code:

This algorithm also shows the author's intent to "resolve" the "obstacle" issue within **one cycle** of Function **loop()**. There are other ways to solve this issue, but the author believes that the presented version would be easier to understand for most readers.

2.8.3 Discussions

We are fortunate that there are several manufacturers for the same type of peripherals or sensors that are needed for our robotics project, but we need to check out the manufacturers specific APIs to see if they allow Non-Blocking procedures with their products. As Blocking procedures would likely prevent access to the other communications lines available on the OpenRB-150. The author had found that the same issue applies for the ROBOTIS OpenCM-904 (Chapter 4) and the MKR ZERO (Chapter 3), as well as the Portenta H7 (Chapter 5); meaning that this involves the inner workings of the Arduino Core which are beyond the reach of most end users.

2.9 Color Tracker

The goal of this project is also to blend Remote-Control and Autonomous- Behavior into the same CarBot, such that the operator can operate the robot via RC100 protocol as before (Sub-Section 2.6.2). But the Operator can also set the CarBot to track a Color Target via a Video Camera at the same time.

In this Section 2.9, we will be using 3 brands of Video Cameras: Pixy2, HuskyLens and OpenMV H7+:

2.9.1 Using Pixy2

Fig. 2.70 shows how the Pixy2 camera is powered by the OpenRB-150 via its 5V Output Pin and controlled via its SPI Port. Pixy2 can also be controlled via I2C or UART connections.

2.9.2 Using HuskyLens

As compared to the Pixy2 Camera, the HuskyLens Camera can only be controlled via I2C or UART connections. The author chose to use I2C for this camera (see Fig. 2.79).

2.9.3 Using OpenMV H7+

Fig. 2.89 shows how the OpenMV H7+ camera is powered by the OpenRB-150 via its 5V Output Pin and controlled via its SPI Port. H7+ can also be controlled via I2C or UART connections also (https://openmv.io/products/openmv-cam-h7-plus). The user needs to use a micro-SD card to store various Python scripts custom made to work with his/her Arduino's projects. This card will map to the user's PC as an external USB drive, when the camera is connected to a PC via a standard USB Port.

2.9.4 Discussions

For Color Tracking purposes, the HuskyLens camera provided the best Data Throughput, next in line is the Pixy2 camera, and then the OpenMV H7+ camera. All three cameras have many more features that are unfortunately outside the scope of this book. If the reader is a beginner in Machine Vision, the author recommends the use of the Husky Lens or Pixy2 cameras to start out. If some Machine Learning features are needed, the OpenMV H7+ camera is the way to go, but its Data Throughput may be too slow for a robot that requires "snappy" behaviors.

2.10 Pan-Tilt Color Tracker

This project builds on the previous Color Tracker project by adding a Pan-Tilt platform using XL-330s set in Position Control mode, so that a HuskyLens camera can be aimed independently from the orientation of the wheels platform (see Fig. 2.100).

2.10.1 Position Control Basics

In this project, the Pan-Tilt platform uses the same type of XL-330 actuators (ID=11 & 12) which were used for the wheeled platform, except that they are now set in **Position Control Mode** (or Joint Mode as it used to be called for XL-320 and AX-12/18 servos). Essentially, these two actuators can be programmed to hold a given Rotational Position A indefinitely or to move to a new Rotational Position B and then hold Position B (as long as the servos are powered of course).

2.10.2 Tasks Prioritization with Duty Cycles

In Sub-Section 2.9.2, the HuskyLens camera was used in a fixed location on the CarBot platform (see Fig. 2.79), and it was shown that the "target-tracking" (TT) process needed to be less frequent that the "remote-controlling" (RC) process to balance out the use of Arduino Core Services between these two processes. The reason was that the TT process required more computing resources via Wire services than the RC process which needed only UART services via Serial2.

2.10.3 New & Revised Tasks in rc_car_PT()

U/D/L/R Arrows (on the PC) were already used in Section 2.9 to control the motions F/B/L/R of the CarBot platform. For the Remote Control of the Pan-Tilt Platform, the author chose to also use U/D/L/R Buttons, and to combine them with Button 6, thus essentially:

- To PAN the Camera LEFT or RIGHT, the Operator should hold Button 6 down and then tap on the LEFT or RIGHT Arrows accordingly.
- Likewise, to TILT the Camera UP or DOWN, the Operator should hold Button 6 down and then tap on the UP or DOWN Arrows accordingly.

2.10.4 New & Revised Tasks in target_track()

Camera-related Procedures and Functions were revised from the ones used in Section 2.9.2 because their usage is slightly different in this project.

To make coding more efficient, a stand-alone Function **get_target_data()** was created (see Fig. 2.108):

Section 2.10 showed that it is quite important to study the Data Flow Characteristics of Sensing and Communication Devices and their interactions, so that the needed Arduino Core Services can be balanced via appropriate programming techniques such as using Duty Cycles for competing robotic tasks.

2.11 Dual Controllers for Pan-Tilt Color Tracker

This project takes a further step along the Distributed Computing path by using a second OpenRB-150 (Pan-Tilt Controller - PTC) for the tasks of controlling the Pan-Tilt Platform and the HuskyLens Camera, while leaving the first OpenRB-150 (CarBot Controller - CBC) to handle the task of controlling the needed CarBot maneuvers. These two Controllers also keep a wired intra-communication line using Serial3 from both sides (see Figs. 2.114 and 2.115).

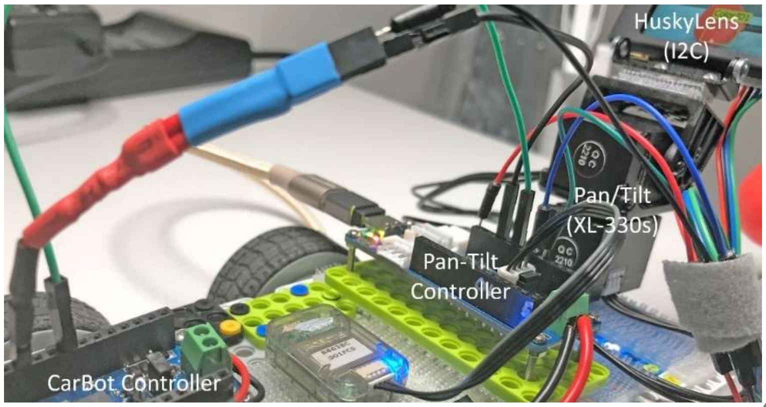


Fig. 2.114 Dual-Controllers version of Pan-Tilt Color Tracker Robot.

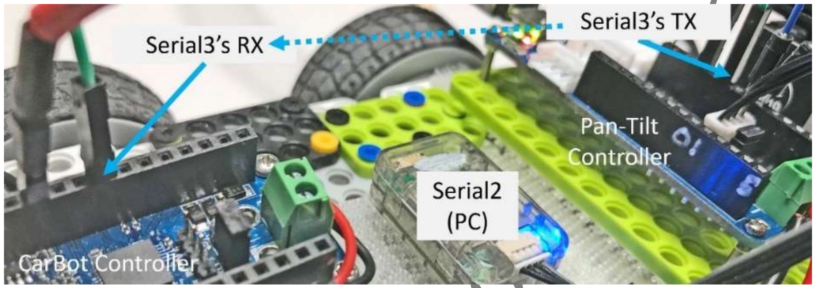


Fig. 2.115 Intra-Communication Line using Serial3 between two OpenRB-150s.

2.11.1 Tasks Prioritization with Duty Cycles

The Duty Cycles approach described earlier is also used in this project with the goal of balancing communication resources usage.

2.11.2 Revised Tasks & Data Flows

Fig. 2.118 shows that Function **send_data()** is revised so that the programmer can send each RC100 Packet via Serial3 – see Lines 208-212.

2.11.3 Discussions

For this Dual-Controllers Project, the most important concept for readers to note is how communication streams are maintained from the Operator's Keyboard Inputs to PTC and CBC, and eventually to the Sensors and Actuators connected to PTC and CBC:

- The Operator's Keyboard Inputs flow from the PC to PTC via a BT-210 only in **one direction** (PC >>> BT-210's TX >> PTC's Serial2 RX).
- PTC receives inputs from its LOCAL Sensor (HuskyLens Camera) and acts upon its LOCAL Actuators (Servos 11 and 12). If PTC needs to control the Wheel Servos (connected only to CBC), PTC sends appropriate RC100 packets via PTC's Serial3 to CBC's Serial3, also only **in one direction** (PTC's Serial3 TX to CBC's Serial3 RX).

The author had found that this **one-way communication** approach worked well for 1-stage communication (PC-to PTC) and smoothly enough for 2-stage communication (PC to PTC to CBC).

Later, we will see how this one-way communication approach can be applied to a similar situation with the Portenta H7 with its two M4 and M7 Cores in Section 5.11.

2.12 Mobile Manipulator (MM)

The focus of Sections 2.12 and 2.13 will be on the Position Control features of ROBOTIS actuators. Section 2.12 uses the older actuators such as AX-12A and AX-12W (https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/ and https://emanual.robotis.com/docs/en/dxl/ax/ax-12w/) with the Mobile Manipulator (MM) Project, while Section 2.13 uses current technologies with the XL-430 series (https://emanual.robotis.com/docs/en/dxl/x/2xl430-w250/ and https://emanual.robotis.com/docs/en/dxl/x/xl430-w250/) for the Articulated 4-Wheels Platform (A4WP-H).

Fig. 2.126 displays the various components of the Mobile Manipulator robot:

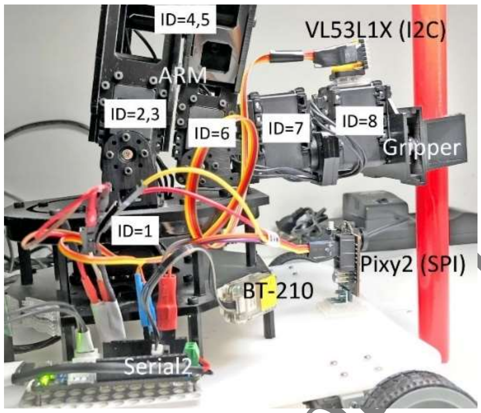


Fig. 2.126 Major Components of the Mobile Manipulator robot.

Details for the Power and Control configurations for MM are shown in Fig. 2.127:

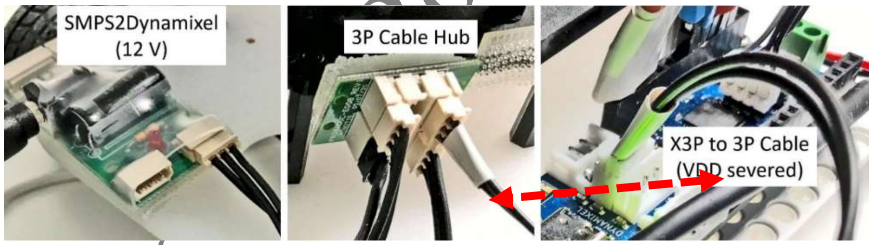


Fig. 2.127 Power and Control Configurations for the Mobile Manipulator robot.

2.12.1 MM: Remote Control

The first MM Project is a basic Remote-Control application allowing the manipulation of different joints of the Reactor Arm. The solution-sketch is named **RC_MobileManipulator.ino**.

The above procedure provides the best approach to synchronize wheels activation when using older actuators that do not have the Shadow-ID features.

2.12.2 MM: Autonomous Color Tracker

The second MM Project uses the Pixy2 Camera (using SPI Port) and the ToF Sensor VL53L1X (using I2C Port) to create a robot that can autonomously detect a colored dowel, approaches this dowel, then grabs it and moves it aside. This Sketch solution is named **Pixy2_MobileManipulator.ino**.

2.12.3 Discussions

This MM Project shows that programming a Manipulator Arm required lots of detailed discrete steps that would be best presented in a Flow Chart by a beginning user. Readers should take notes of the interactions between Goal Position, Present Position and Goal Velocity commands.

For the AX-12, there are other features such as “Torque” and “Joint Offset” that some readers may be interested in: see Chapter 6 in Thai (2017).

2.13 Articulated 4-Wheeled Platform (A4WP-H)

In this Section 2.13, we will explore selected advanced features of ROBOTIS Actuators set in Position Control Mode, such as Sync Write/Read, Time-based Position Control and Motion Programming using Motion Arrays and Offsets, and we apply them to a Hybrid Walking/Rolling Quadruped named A4WP-H (see Fig. 2.152).

The newer DXLs do have extra features such as PWM Control, Multi-turn Control, Current Control that the reader would have to explore on his/her own if needed, using this Section’s materials as foundations to spring from.

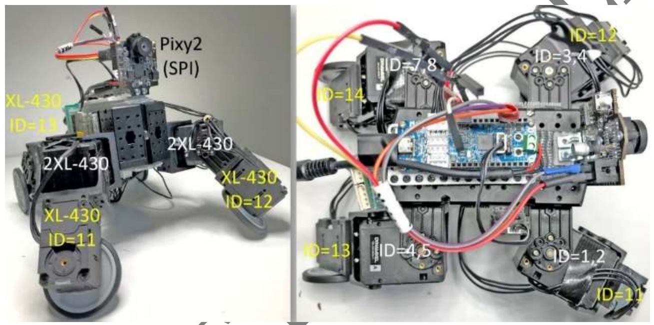


Fig. 2.152 Major Components of the A4WP-H robot.

The A4WP-H robot uses four 2XL-430 actuators for the ones attached to the Quadruped body and they are set into Time-based Position Control Mode. The 4 leg’s ends use XL-430s set into Velocity Control Mode and have thin wheels attached to them to act as “rolling claws”. The Pixy2 Camera (in SPI mode) is attached directly to the main robot body and a special technique using Goal Position Offsets will be used to tilt the body+camera platform up and down as needed. Serial 2 Port is used for Remote-Control from the PC via the Virtual RC-100 Controller and a BT-210. The Back Legs provide only Wheeled Locomotion while the Front Legs are programmed to Walk/Crawl/Roll.

The DXL Power Circuit operates at 12 V and is kept separate from the DXL Control Circuit (plus OpenRB-150 board) which runs at 5V via the DXL Hub on the OpenRB-150. However, they do share a common Ground Node – see post (https://forum.robotis.com/t/openrb-150-u2d2-phb-ttl/1493/2) for review if needed.

2.13.1 Sync Read & Sync Write

In the previous sections, the reader has surely noted that Actuator Programming had been done using the general technique of sending a single command/packet to a single actuator each time and then using loops to access all actuators of interest on a given robot design/platform. So far, the robotics applications showcased in this book are such that this long sequence of little packets had not yielded noticeable performance differences.

Starting with Section 2.13.1, we will use a “Big-Packet” technique whereas this Big-Packet is sent to all affected actuators at the same time (more or less) as we still have to use Serial Communications in the background with the Arduino Core.

We are now ready to integrate the previous concepts into the design of an Arduino Sketch for the Remote-Control of the A4WP-H robot which can roll on all 4 wheels, but also can tilt its body up/down to aim the Pixy2 Camera and walks/rolls with its front legs.

2.13.3 A4WP-H: Motion Arrays

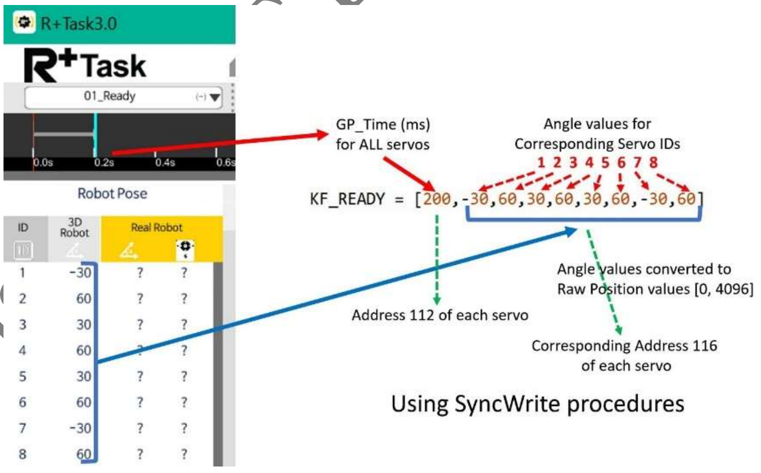
If the reader has no experience in doing Motion Programming with ROBOTIS hardware and software, the author recommends the viewing of this video (https://www.youtube.com/watch?v=qKqCr8MwG4A) for a primer of the MOTION tool that comes with the ROBOTIS TASK V.3 Application. The main goal is for the reader to understand the concept of KEY-FRAME and TIME-LINE where those KEY-FRAMES are set, so as to create an overall coordinated movement for the robot at runtime. Direct analogies can be made with Stop-Motion techniques used in making movies or the standard creation of cartoons. For Motion Programming details with the XL-320s (i.e., MINI robot), please refer to Thai (2020).

If the reader plans to use Motion Programming extensively, the reader should invest in an OpenCM-904 because only this Controller can run either ROBOTIS Firmware which will allow the use of TASK 3 or switch to its Arduino configuration. The author used this approach to get started on the creation of the Motion Arrays that are used in this Arduino project.

ROBOTIS also has made two informative videos to help users create and use custom Motions for any user-designed robot using OpenCM-904 and TASK 3, but the robot must be constructed using ROBOTIS actuators (https://www.youtube.com/watch?v=YqAucpqsP6o and https://www.youtube.com/watch?v=UWNSzDkCjpE).

Getting back to the A4WP-H robot, as it is derived from the QUADRUPED robot that comes with the ROBOTIS ENGINEER Kit 1 (Thai, 2021), the author started from the appropriate KEY-FRAMEs created for the QUADRUPED and modified them into a "Timing + Goal Position" data structure that is suitable for Time-based Position Control on the OpenRB-150 (as previously demonstrated in Sub-Section 2.13.2).

Fig. 2.173 is a visual description of this conversion process:


R+Task3.0
R+Task
01_Ready
0.0s
0.2s
0.4s
0.6s
Robot Pose
GP_Time (ms) for ALL servos
Angle values for Corresponding Servo IDs
1 2 3 4 5 6 7 8
Address 112 of each servo
Angle values converted to Raw Position values [0, 4096]
Corresponding Address 116 of each servo
Using SyncWrite procedures

```
KF_READY = [200,-30,60,30,60,30,60,-30,60]
```

ID	3D Robot	Real Robot	
1	-30	?	?
2	60	?	?
3	30	?	?
4	60	?	?
5	30	?	?
6	60	?	?
7	-30	?	?
8	60	?	?

Fig. 2.173 Conversion of a ROBOTIS Key-Frame into Arduino Motion Array.

2.13.4 A4WP-H: Motion Programming & Remote Control

Let's now have a closer look at the integration of RC-100 Button inputs from the Operator into the actual triggering of the proper Motion Units F/B/L/R. We'll use the case of the UP Button to illustrate the typical programming steps needed to produce a Forward Motion for the A4WP-H robot.

go_robot_go(robot_direction) to physically activate the appropriate robot Motions needed to keep track of the Target. Please note that **robot_moving and wheels_only** are reset to FALSE at the end of each tracking cycle (Lines 517 and 520).Again, the Duty Cycle approach is used to balance out the RC and Target Tracking tasks.

For the readers convenience, two versions of this “RC + Target Tracking” Project with the A4WP-H robot are provided: **Pixy2_RC_A4WP_H.ino** uses the old “Unpacked Data Protocol” and **Pixy2_RC_A4WP_H_P.ino** uses the new “Packed Data Protocol”.

2.13.6 Discussions

Advanced readers may have noticed a short coming in the way Motion Programming is implanted in this Section 2.13, more specifically in the way that Method **Speed_Motion_Set()** uses a FOR Loop and always waits for each Key-Frame to finish before it can do anything else (see Line 774 in Fig. 2.178).